SISTEMAS OPERATIVOS

Digitalización con propósito académico
Sistemas Operativos

SISTEMAS OPERATIVOS

Segunda edición

William Stallings

Traducción:

Juan Manuel Dodero Beardo
Enrique Torres Franco
Facultad y Escuela de Informática
Universidad Pontificia de Salamanca en Madrid

Miguel Katrib Mora
Facultad de Matemática y Computación
Universidad de la Habana

Revisión técnica:

Luis Joyanes Aguilar
Facultad y Escuela de Informática
Universidad Pontificia de Salamanca en Madrid

P R E N T I C E H A L L

Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima • Montevideo •San Juan • San José • Santiago • Sao Paulo • White Plañís

datos de colocación bibliográfica

STALLINGS W.,
Sistemas Operativos, 2ed.
P R E N T I C E H A L L. Madrid, 1997

ISBN: 84-89660-22-0
Materia:
Informática 681.3

Formato 200 x 250mm Páginas 732

WILLIAM STALLINGS
Sistemas Operativos, 2ed

ISBN: 84-89660-22-0
Depósito legal: M. 20.979-2000
5.ª reimpresión, 2000

Traducido de:
OPERATING SYSTEMS. Second edition.
P R E N T I C E H A L L, INC.- Simón & Schuster Intemational Group
Copyright © MCMXCV
ISBN: 0-13-180977-6

Editor de la edición en español: Andrés Otero
Diseño de cubierta: DIGRAF
Composición: ENYCE
Impreso por: FARESO S.A
IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Este libro ha sido impreso con papel y tintas ecológicos
Digitalización con propósito académico
Sistemas Operativos

Contenido

PRÓLOGO

Objetivos

Este libro trata sobre los conceptos, la estructura y los mecanismos de los sistemas operativos. Su finalidad es la de presentar de forma tan clara y completa como sea posible, la naturaleza y las características de los sistemas modernos.

Esta tarea constituye un desafío por varias razones.

En primer lugar, existe una enorme y variada gama de sistemas informáticos para los que se diseñan sistemas operativos. Entre estos sistemas se incluyen las estaciones de trabajo monousuario y los computadores personales, los sistemas compartidos de tamaño medio, los grandes computadores centrales, los supercomputadores y las máquinas especializadas como los sistemas de tiempo real. La variedad no está sólo en la capacidad y la velocidad de las máquinas, sino también en los requisitos de soporte de los sistemas y las aplicaciones. En segundo lugar, el veloz ritmo en los cambios que siempre ha caracterizado a los sistemas informáticos aumenta sin descanso. Una serie de campos clave en el diseño de los sistemas operativos son de origen reciente y la investigación en estos y en otros campos es continua.

A pesar de tal variedad y velocidad en los cambios, determinados conceptos fundamentales se pueden aplicar en todos los casos de una forma consistente. Para estar seguros, la aplicación de estos conceptos depende del estado actual de la tecnología y de los requisitos de las aplicaciones particulares. El cometido de este libro es proporcionar una discusión completa de los fundamentos del diseño de los sistemas operativos (SO) y hacer mención a las cuestiones de diseño y las tendencias actuales en el desarrollo de sistemas operativos.

El objetivo es proporcionar al lector una comprensión sólida de los mecanismos clave de los sistemas operativos modernos, las concesiones y las decisiones que acarrean el diseño de un SO y el contexto en el que éste opera (el hardware, otros programas del sistema, los programas de aplicación y los usuarios interactivos).

El concepto de proceso

El concepto de proceso es el elemento central del estudio de los sistemas operativos. Aunque todos los libros sobre la materia abordan este tema, ningún otro texto destacado dedica una sección importante a introducir y explicar los principios básicos de los procesos. En este libro, el Capítulo 3 está dedicado a esta labor. Los resultados son unas bases sólidas para el examen de las múltiples cuestiones que se abordan en los capítulos posteriores.

Desarrollos recientes en el diseño de sistemas operativos

Además de ofrecer cobertura a los fundamentos de los sistemas operativos, el libro examina los desarrollos recientes más importantes que se han alcanzado en el diseño de los sistemas operativos. Entre los temas tratados están los siguientes:

•*Hilos:* El concepto de proceso es algo más complicado y sutil que el que se suele presentar y, de hecho, engloba dos conceptos separados e independientes en potencia: uno relativo a la propiedad de los recursos y otro relativo a la ejecución. Esta distinción ha llevado al desarrollo, en algunos sistemas operativos, de unas estructuras conocidas como hilos.

• *Sistemas de tiempo real:* En los últimos años, el proceso en tiempo real ha llegado a verse como una importante disciplina emergente en informática e ingeniería. El sistema operativo y, en particular, el planificador, es quizás el componente más importante de un sistema de tiempo real.

• *Planificación de multiprocesadores:* Tradicional mente, se ha hecho una escasa distinción entre los métodos de planificación que se aplican en los monoprocesadores multiprogramados y los de los sistemas de multiprocesadores. Con el interés creciente en el empleo de hilos y en la programación paralela, la planificación de multiprocesadores se ha convertido en objeto de estudios y desarrollos intensos.

• *Sistemas distribuidos:* Con la disponibilidad cada vez mayor de computadores personales y minicomputadores baratas pero potentes, se ha producido una tendencia creciente hacia el proceso de datos distribuido (DDP, *Distributed Data Processing).* Con el DDP, los procesadores, los datos y otros elementos de un sistema distribuido de proceso de datos pueden estar dispersos para una organización. Muchas de las cuestiones de diseño de SO tratadas en este libro tratan sobre la complejidad añadida de los entornos distribuidos.

• *Migración de procesos;* La migración de procesos es la capacidad para trasladar un proceso activo de una máquina a otra; se ha convertido en un tema cada vez más candente en los sistemas operativos distribuidos. El interés por este concepto surgió de la investigación sobre formas de equilibrar la carga en varios sistemas conectados en red, aunque su aplicación se extiende ahora más allá de este campo. Hasta hace poco, algunos observadores creían que la migración de procesos era poco práctica. Se ha demostrado que tales aseveraciones eran demasiado pesimistas. Las nuevas implementaciones, incluyendo las de algunos productos comerciales, han alimentado un interés continuo y nuevos desarrollos en este campo.

• *Seguridad:* La seguridad ha sido durante mucho tiempo una preocupación en el diseño de los sistemas operativos. Sin embargo, los enfoques de diseño de la seguridad han evolucionado a medida que evolucionaron las amenazas. Entre los ejemplos de áreas de amenaza que presentan dificultades nuevas y complejas se incluyen los virus y los ataques a los sistemas operativos distribuidos. Un ejemplo de un nuevo enfoque para hacer frente a estas amenazas es el concepto de sistema de confianza.

Sistemas de ejemplo

Este texto está pensado para informar al lector sobre los principios de diseño y las cuestiones de implementación de los sistemas operativos contemporáneos. Por consiguiente, un tratamiento teórico o puramente conceptual no sería el adecuado. Para ilustrar los conceptos y asociarlos a las elecciones reales de diseño que deben hacerse, se han elegido tres sistemas operativos como ejemplo marco.

• *Windows NT:* Un sistema operativo monousuario y multitarea para computadores personales, estaciones de trabajo y servidores. Como nuevo sistema operativo que es, Windows NT incorpora de forma evidente los últimos avances en tecnología de sistemas operativos. Además, Windows NT es uno de los primeros sistemas operativos importantes que confían profundamente en los principios del diseño orientado a objetos.

• *UNIX:* Un sistema operativo multiusuario, pensado originalmente para minicomputadores, pero implementado en una amplia gama de máquinas, desde potentes microcomputadores hasta supercomputadores.

• *MVS:* El sistema operativo situado a la cabeza de la gama de computadores centrales de IBM y, quizás, el sistema operativo más complejo jamás desarrollado. Ofrece posibilidades tanto de tratamiento por lotes como de tiempo compartido.

Estos tres sistemas fueron elegidos por su importancia y representatividad. La mayoría de los sistemas operativos de computadores personales, estaciones de trabajo y servidores de las máquinas nuevas son sistemas monousuario y multitarea; Windows NT es un ejemplo del estado de la ciencia. UNIX ha llegado a ser el sistema operativo dominante en una amplia variedad de estaciones de trabajo y sistemas multiusuario. MVS es el sistema operativo de computadores centrales más usado. Así pues, la mayoría de los lectores tendrán que tratar con algunos de estos sistemas operativos en el momento en que utilicen este libro o dentro de unos pocos años.

Al igual que la técnica usada en mi libro *Organización y Arquitectura de Computadoras,* la discusión de los sistemas de ejemplo está distribuida a lo largo del texto, en vez de ensamblarla en un sólo capítulo o apéndice. Por ejemplo, durante el estudio de la memoria virtual, se describen los algoritmos de reemplazo de páginas de cada uno de los ejemplos y se discuten los motivos de las opciones individuales de diseño. Con este enfoque, los conceptos de diseño discutidos en un capítulo dado se ven inmediatamente reforzados con ejemplos del mundo real.

Público al que está dirigido

El libro está dirigido tanto a un público académico como profesional. Como libro de texto, está pensado para un curso de pregrado sobre sistemas operativos de un semestre, tanto para informática como ingeniería. Aborda todos los temas del curso de sistemas operativos recomendado en el plan de estudios de informática de 1991, del informe *ACM/IEEE-CS Joint Curriculum Task Forcé.*

El libro también sirve como volumen básico de referencia y es válido para el autoestudio**.**

Planificación del texto

La organización de los capítulos es la que sigue:

1. I*ntroducción a los sistemas informáticos:* Ofrece una visión general de la arquitectura y organización de los computadores, haciendo énfasis en los temas relacionados con el diseño de sistemas operativos.
2. *Introducción a los sistemas operativos:* Ofrece una visión general del resto del libro.
3. *Descripción y control de procesos:* Introduce el concepto de proceso y examina las estructuras de datos empleadas por el sistema operativo para controlar los procesos. También se discuten los conceptos afines de hilo y sesión.
4. *Concurrencia: exclusión mutua y sincronización:* Examina los aspectos clave de la concurrencia en sistemas sencillos, haciendo énfasis en la exclusión mutua y en los mecanismos de sincronización.
5. *Concurrencia: interbloqueo e inanición:* Describe la esencia del interbloqueo y de la inanición y los enfoques de diseño para su tratamiento.
6. *Gestión de memoria:* Ofrece un estudio completo de las técnicas de gestión de memoria.
7. *Memoria virtual:* Observa con detalle las estructuras del hardware que dan soporte a la memoria virtual y las características del software de un sistema operativo que disponga de memoria virtual.
8. *Planificación de monoprocesadores:* Ofrece una discusión comparativa de los diferentes enfoques de la planificación de procesos.
9. *Planificación de multiprocesadores y en tiempo real:* Examina estas dos áreas importantes de la planificación de procesos.
10. *Gestión de la EIS y planificación de discos:* Examina los problemas que acarrea el control de las funciones de E/S por parte del SO. Se dedica una atención especial a la E/S con el disco, que es un punto clave en el rendimiento del sistema.
11. *Gestión de archivos:* Proporciona una visión general de la gestión de archivos, con énfasis en aquellos aspectos que se suelen implementar como parte del sistema operativo o que están estrechamente relacionados con el sistema operativo.
12. *Redes y proceso distribuido:* Examina las principales tendencias en este campo, incluyendo el modelo OSI, los protocolos TCP/IP y la arquitectura cliente/servidor.

También se explican técnicas importantes de comunicación distribuida entre procesos, como son el paso de mensajes y las llamadas a procedimientos remotos.
13. *Gestión distribuida de procesos:* Describe algunos de los puntos clave de diseño en el desarrollo de sistemas operativos distribuidos, incluyendo la migración de procesos, la exclusión mutua y el interbloqueo.
14. *Seguridad:* Ofrece un estudio de las amenazas y de los mecanismos para conseguir seguridad en redes y en computadores.
A. *Análisis de colas:* Este apéndice es una guía práctica del empleo del análisis de colas para modelar el rendimiento.
B. *Diseño orientado a objetos:* Introduce los conceptos esenciales del diseño orientado a objetos.

Además, el libro incluye un amplio glosario, una lista de acrónimos usados frecuentemente y una bibliografía. Cada capítulo incluye problemas y propuestas de lecturas complementarias.

Qué hay de nuevo en la segunda edición

En los tres años que han pasado desde que se publicó la primera edición de *Sistemas Operativos* la materia se ha visto sujeta a mejoras e innovaciones continuas. En esta nueva edición, se intentan captar estos cambios y, a la vez, lograr una cobertura amplia y completa del tema. Además, muchos lectores de la primera edición han realizado comentarios constructivos que han provocado una reorganización sustancial de la materia.

En el cambio de los contenidos, uno de los más importantes es la introducción de las tecnologías orientadas a objetos. Muchos sistemas operativos ya utilizan técnicas de diseño orientadas a objetos y casi todos los nuevos sistemas operativos para computadores personales, estaciones de trabajo y servidores emplearán esta tecnología en el futuro. El vehículo para la explicación de los conceptos tratados en el libro es solamente la discusión de Windows NT. Aunque NT no es un sistema orientado a objetos "puro", implementa las características esenciales del diseño orientado a objetos en un sistema operativo. El libro también viene con un nuevo apéndice que proporciona una introducción al diseño orientado a objetos.

Se ha ampliado y actualizado el estudio de las redes y los sistemas operativos distribuidos. En el campo de los sistemas operativos distribuidos, el libro ofrece una mayor cobertura de la migración de procesos y una discusión ampliada del control distribuido del interbloqueo. En el campo de las redes, el libro contiene ahora una sección nueva e importante sobre la arquitectura cliente/servidor. Esta arquitectura se ha convertido en la norma de soporte del proceso distribuido y de las aplicaciones distribuidas. Además, el tratamiento de las redes que se hace en el libro incluye ahora una visión general del TCP/IP, la arquitectura de protocolos más importante en las redes.

El tratamiento de la concurrencia se ha ampliado para incluir una sección sobre los monitores. Este enfoque para lograr concurrencia está cada vez más difundido. El libro incluye

una discusión sobre las diferencias entre los antiguos monitores de Hoare y los de Lampson/Redell y explica las ventajas de este último enfoque.

Uno de los cambios más extensos en el libro es la amplia cobertura que se ha dado a la seguridad. El tratamiento de los virus y de las amenazas afines por software también se ha ampliado, incluyendo una nueva sección importante sobre los intrusos, que abarca la protección de contraseñas, la prevención y la detección de intrusiones.

El libro también se ha reorganizado para incluir más capítulos y más reducidos. Esto lo hace más manejable y accesible para los estudiantes. Además, se han añadido más problemas propuestos para ayudar a reforzar y ampliar la materia de cada capítulo.

Lista de envíos en internet

Se ha preparado una lista de envíos en intemet, de forma que los profesores que empleen este libro puedan intercambiar información, propuestas y preguntas entre ellos y con el autor. Para suscribirse, hay que enviar un mensaje a majordomo@shore.net con el siguiente texto: *subscribe ws-OS.* Para remitir un mensaje, hay que enviarlo a ws-OS@shore.net.

Agradecimientos

Este libro está basado en la primera edición, de forma que se ha beneficiado de las revisiones del manuscrito de dicha edición, incluyendo las realizadas por Ranee Cleaveland, Eric Cooper, Samuel Gulden, Gary Harkin, Evan Ivie, Phillip Krueger, Evelyn Obaid, E. K. Partí, Umakishore Ramachandran, Margaret Reek, Armin Roeseler, Sol Shatz, Charles Shub, James Silver, Mark Smotherman y Anand Tripathi.

Una serie de personas también revisaron todo el manuscrito de la segunda edición o parte del mismo, incluyendo a May C. Abboud, David Boddy, Phillip Krueger, Gwynne Larson, Mitchell L. Neilsen, Titus D. M. Purdin y Cindy Snow. También agradezco a K. C. Tai sus aportaciones en la temporización de sucesos distribuidos, a Steven Hartiey por algunos problemas propuestos, a Donaid Gillies por sus aportaciones en los monitores, a Richard Wright por su revisión del material de Windows NT y a Ralph Hilzer por el ejemplo de la barbería.

W.S.

Prólogo a la edición en español

El sistema operativo, como es bien conocido, es el programa del sistema que controla todos los recursos del computador y ofrece el soporte básico sobre el cual pueden escribirse los programas de aplicación. Por esta causa un libro que trate sobre los conceptos, estructura y mecanismos de los modernos sistemas operativos es un acontecimiento científico que ha de ser examinado cuidadosamente. Por otra parte, nos encontramos en el último cuatrimestre de 1996 en el que los computadores ya no trabajan de igual forma que en décadas pasadas y si el libro de sistemas operativos tiene presente esta situación, nos encontramos con una obra de utilidad para los profesionales informáticos.

En el pasado la mayoría de los computadores trabajaban de modo aislado y en consecuencia la mayoría de los sistemas operativos se diseñaban para ser ejecutados en un único procesador. Esta situación ha cambiado *casi* radicalmente. Ahora, los computadores están conectos en red, razón por la cual los sistemas operativos distribuidos son cada vez más importantes. Por otra parte, las tecnologías de objetos se imponen cada día más y con mayor intensidad en el mundo del software. Es por ello que un libro actualizado sobre sistemas operativos debería contemplar el estudio de los sistemas operativos distribuidos, y al menos tener presente las tecnologías de objetos.

La importancia del sistema operativo ha hecho que esta asignatura sea considerada fundamental en los estudios de licenciatura e ingeniería informática, así como en ingenierías electrónicas de telecomunicaciones y de sistemas computacionales. Este es el caso de Estados Unidos, donde las recomendaciones de los *curricula* de estudios de computación (informática) de la ACM y de la IEEE-CS, consideran esta materia en el informe *computing curricula 1991* emitido por ambas instituciones. En España, el espíritu se ha recogido también en los *curricula* y así lo ha plasmado el Consejo de Universidades en los nuevos planes de estudio de ingeniería informática técnica y superior, asignándole la categoría de materia troncal, lo que conlleva la obligatoriedad de su estudio por todas las universidades oficiales y privadas. Naturalmente no tendría sentido la recomendación si no viniera acompañada de los descriptores o temas mínimos que han de estudiarse en la citada asignatura. Esta misma situación se da también en Latinoamérica y así lo hemos comprobado con ocasión de diferentes viajes a México, Cuba, Venezuela, etc., donde hemos impartido conferencias, cursos o seminarios en universidades e institutos tecnológicos de estas naciones.

La obra de Stallings abarca ampliamente los descriptores de la materia troncal *sistemas operativos* en los planes de estudio españoles (Reales Decretos 1459, 1460, 1461/1990 de 16 de octubre de 1990) y las recomendaciones de la ACM/IEEE-CS, pero con una ventaja adicional, describe los citados descriptores actualizados al año 1995, fecha de publicación del libro.

Una de las fortalezas más sobresalientes del libro es el tratamiento y el enfoque práctico que hace de los sistemas operativos reales, implantados en el mercado informático. Así al final de cada capítulo se suele incluir ejemplos sobre los tres sistemas operativos estándar más importantes de la actualidad: Windows NT, UNIX y MVS.

La obra de Stallings incorpora una serie de temas avanzados relativos a los sistemas operativos distribuidos; especialmente destaca su acierto en el estudio de materias de gestión distribuida de procesos, tema difícil y complejo que pocas obras consideran. También da especial importancia a la administración o gestión del procesador.

Una novedad sobresaliente es la inclusión de un apéndice relativo al importante futuro de las tecnologías de objetos en el diseño de los modernos sistemas operativos orientados a objetos, analizando su concepto y los beneficios que su aplicación producirán en los sistemas operativos.

En resumen, **Sistemas Operativos** será de gran utilidad para uno o dos cursos universitarios, al incluir todo los temas usuales de una licenciatura o ingeniería informática, en sistemas de computación o de telecomunicaciones, así como para licenciaturas de Matemáticas y Físicas en sus especialidades de Ciencias de la Computación. El rigor científico del autor se muestra en cada página del libro, y el hecho de que el propio autor ha publicado una obra complementaria sobre *hardware* de los computadores *Computer Organization and Architecture, fourth edition* -que se acaba de publicar en castellano-, le augura un futuro brillante y estamos seguros será una obra de referencia y de alta estima entre profesores y alumnos de los citados estudios como libro básico y como libro de consulta para estudios afines del mundo científico de la programación.

AGRADECIMIENTOS

Todo el equipo de profesores responsables de la traducción de esta obra deseamos destacar y agradecer la lectura y revisión efectuada de las últimas pruebas de imprenta (galeradas) por el también profesor y compañero de sistemas operativos **Virgilio Yagüe Galaup**, del Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software, de la Facultad de Informática de la Universidad Pontificia de Salamanca en el *campus* de Madrid. Su lectura ha permitido detectar erratas y muchas de sus sugerencias han sido incorporadas a la adaptación final de esta obra.

Luis Joyones Aguilar
Director del Departamento de Lenguajes y
Sistemas Informáticos e Ingeniería del Software
Facultad y Escuela Universitaria de Informática Universidad Pontificia de Salamanca en Madrid

CAPÍTULO 1

Introducción a los sistemas informáticos

Un sistema operativo (SO) explota los recursos de hardware de uno o más procesadores para ofrecer un conjunto de servicios a los usuarios del sistema. El sistema operativo también gestiona la memoria secundaria y los dispositivos de entrada/salida (E/S) en nombre de los usuarios. Así pues, es conveniente disponer de una cierta comprensión del hardware del sistema informático subyacente antes de comenzar el estudio de los sistemas operativos.

Este capítulo ofrece una visión de conjunto del hardware de los sistemas informáticos. Este resumen es extremadamente breve en la mayoría de los campos, pues se asume que el lector está familiarizado con el tema. Sin embargo, algunas áreas se abordan con un cierto detalle, por su importancia en los asuntos que se tratarán más adelante en el libro.

1.1 ELEMENTOS BÁSICOS

En un alto nivel, un sistema informático consta de procesador, memoria y componentes de E/S, con uno o más módulos de cada tipo. Estas componentes están interconectados de alguna forma para llevar a cabo la función principal del computador, que es ejecutar programas. Así pues, se tienen cuatro elementos principales:

- *Procesador:* Controla la operación del computador y lleva a cabo las funciones de procesamiento de datos. Cuando hay un solo procesador, se suele denominar *unidad central de procesamiento (CPU, Central Processing Unit).*
- *Memoria Principal:* Almacena los datos y los programas. Esta memoria es normalmente volátil; también se le conoce como *memoria real o memoria primaria.*
- *Interconexión de sistemas:* Ciertos mecanismos y estructuras que permiten la comunicación entre procesadores, memoria principal y los módulos de E/S.

La figura 1.1 ilustra estos componentes de alto nivel. El procesador es normalmente quien lleva el control. Una de sus funciones es intercambiar los datos con la memoria. Para este propósito, hace uso de dos registros internos (al procesador): un registro de direcciones de memoria (MAR, *Memory Address Register),* el cual especifica la dirección en memoria de la próxima lectura o escritura y un registro intermedio *(buffer)* de memoria (MBR, *Memory Buffer Register),* que contiene los datos que van a ser escritos a memoria o que fueron leídos de la misma. De manera similar, un registro de direcciones de E/S (IOAR, *InputlOutput Address Register)* especifica un dispositivo particular de E/S. Un registro intermedio de E/S (IOBR, *InputtOutput Buffer Register)* se utiliza para intercambiar datos entre un módulo de E/S y el procesador.

Un módulo de memoria consta de un conjunto de ubicaciones definidas por direcciones enumeradas secuencialmente. Cada ubicación contiene un número binario que puede ser interpretado como una instrucción o como un dato. Un módulo de E/S transfiere datos desde los dispositivos externos hacia la memoria y el procesador y viceversa. Este contiene buffers internos para almacenar temporalmente los datos hasta que puedan ser enviados.

1.2 REGISTROS DEL PROCESADOR

Dentro del procesador, hay un conjunto de registros que ofrecen un nivel de memoria que es más rápido y pequeño que la memoria principal. Los registros del procesador sirven para dos funciones:

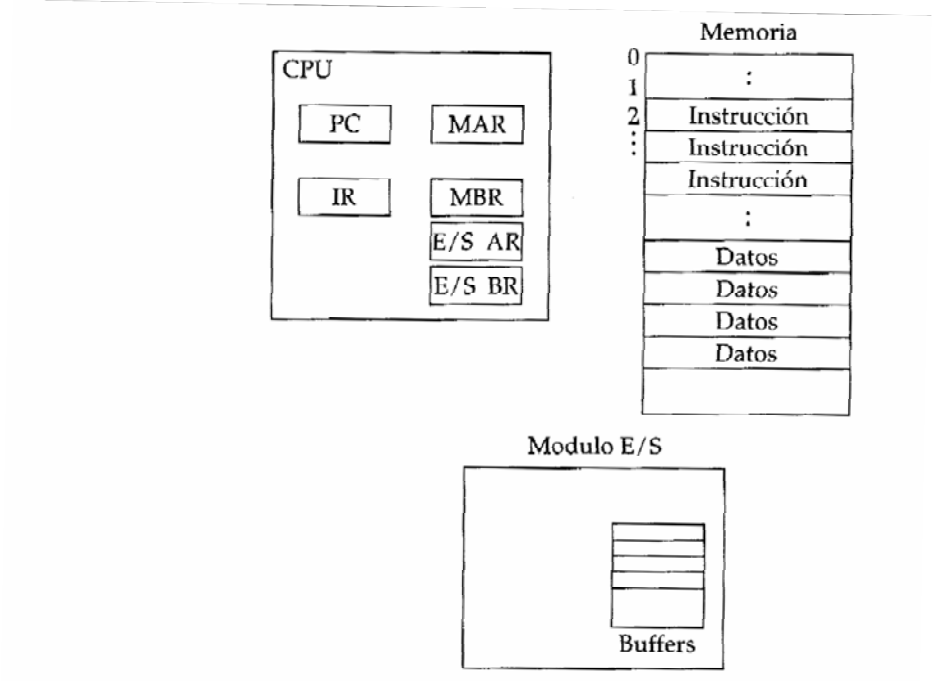


FIGURA 1.1 Componentes de un computador: Visión de alto nivel

• *Registros visibles de usuario:* Un programador de lenguaje de máquina o ensamblador puede minimizar las referencias a memoria principal mediante un uso óptimo de estos registros. Con lenguajes de alto nivel, un compilador que optimice código intentará hacer una selección inteligente de qué variables asignar a registros y cuáles a ubicaciones de la memoria principal. Algunos lenguajes de alto nivel, como C, permiten que el programador indique al compilador qué variables se deben almacenar en registros.
• *Registros de control y de estado:* Son utilizados por el procesador para el control de las operaciones o por rutinas privilegiadas del sistema operativo para controlar la ejecución de los programas.

No hay una separación clara de los registros en estas dos categorías. Por ejemplo, en algunas máquinas el contador de programa es visible para los usuarios, pero en otras muchas no lo es. Sin embargo, para el propósito de la discusión que viene a continuación, es conveniente emplear estas categorías.

Registros visibles de usuario

Un registro visible de usuario es aquél que puede ser referenciado por medio del lenguaje de máquina que ejecuta el procesador y es, por lo general, accesible para todos los programas, incluyendo tanto los programas de aplicación como los del sistema. Las clases de registro que, normalmente, están disponibles, son los registros de datos, los registros de dirección y los registros de códigos de condición.

Los **registros de datos** pueden ser asignados por el programador a diversas funciones. En algunos casos, son de propósito general y pueden ser empleados por cualquier instrucción de máquina que lleve a cabo operaciones sobre los datos. Sin embargo, suelen ponerse ciertas restricciones a menudo. Por ejemplo, pueden haber registros dedicados a operaciones en coma flotante.

Los **registros de dirección** contienen direcciones en la memoria principal de datos e instrucciones o una parte de la dirección que se utiliza en el cálculo de la dirección completa. Estos registros pueden ser de propósito general o pueden estar dedicados a un modo específico de direccionamiento. Entre los ejemplos se incluyen:

• *Registro índice:* El direccionamiento indexado es un modo común de direccionamiento que implica sumar un índice a un valor base para obtener la dirección efectiva.
• *Puntero[i] de segmento:* Con direccionamiento segmentado, la memoria se divide en segmentos, que son bloques de palabras de tamaño variable. Una referencia a memoria consta de una referencia a un segmento particular y un desplazamiento dentro del segmento; este modo de direccionamiento es importante en la discusión sobre la gestión de memoria de los capítulos 6 y 7. En este modo, se utiliza un registro que alberga una dirección base (ubicación inicial) de un segmento. Puede haber varios registros de este tipo: por ejemplo, uno para el sistema operativo (es decir, cuando se ejecuta código del sistema operativo en el procesador) y otro para la aplicación que está en ejecución.
• *Puntero de pila:* Si hay un direccionamiento de pila visible para los usuarios, la pila estará, por lo general, en la memoria principal, existiendo un registro dedicado a señalar

[i] El término inglés *pointer* suele traducirse en Latinoamérica por el término apuntador (N. del. T.)

la cima de la pila. Esto permite el uso de instrucciones que no contienen ningún campo de dirección, tales como *push* (poner) y *pop* (sacar). (Consúltese el Apéndice IB para una discusión sobre el tratamiento de pilas).

Una última categoría de registros que son, al menos, parcialmente visibles para los usuarios, son aquellos que contienen **códigos de condición** (también denominados *indicadores o flags).* Los códigos de condición son bits activados por el hardware del procesador como resultado de determinadas operaciones. Por ejemplo, una operación aritmética puede producir un resultado positivo, negativo, cero o desbordamiento. Además de almacenar el resultado de esta operación en un registro o en memoria, también se activará un código de condición. Este código puede consultarse posteriormente como parte de una operación de salto condicional.

Los bits de código de condición se agrupan en uno o más registros. Estos forman generalmente parte de un registro de control. En general, las instrucciones de máquina permiten leer estos bits mediante referencias implícitas, pero no pueden ser alterados por el programador.

En algunas máquinas, una llamada a un procedimiento o subrutina provocará que los registros visibles de usuario se salven automáticamente, para luego restaurarlos al retomar. Este proceso de salvar y restaurar lo lleva a cabo el procesador como parte de la ejecución de las instrucciones de llamada y retomo. Esto permite que cada procedimiento pueda usar los registros de forma independiente. En otras máquinas, es responsabilidad del programador salvar los contenidos de los registros de usuario visibles que sean relevantes antes de hacer la llamada a un procedimiento, incluyendo instrucciones en el programa con tal propósito. Así pues, las instrucciones de salvar y restaurar pueden ser llevadas a cabo por el hardware o por el software, dependiendo de la máquina.

Registros de control y de estado

Varios registros se emplean para controlar las operaciones del procesador. En la mayoría de las máquinas, la mayor parte de estos registros no son visibles para los usuarios. Algunos de ellos pueden estar accesibles a las instrucciones de máquina ejecutadas en un modo de control o modo del sistema.

Por supuesto, máquinas diferentes tendrán organizaciones diferentes de registros y podrán usar terminologías distintas. Aquí se da una lista bastante completa de tipos de registros, incluyendo una breve descripción de las mismas. Además de los registros **MAR, MBR, IOAR y IOBR** mencionados anteriormente, los siguientes registros son esenciales en la ejecución de instrucciones;

• *Contador de programa (PC, Program Counter):* Contiene la dirección de la instrucción a ser leída.

• *Registro de instrucción (IR, Instruction Regíster):* Contiene la última instrucción leída.

Todos los diseños de procesadores incluyen además un registro o conjunto de registros, conocidos a menudo como palabra de estado del programa (PSW, *Program Status Word),* que contiene información de estado. Normalmente, la PSW contiene códigos de condición junto a otra información de estado. Entre los campos e indicadores más comunes se incluyen los siguientes:

• *Signo:* Contiene el bit del signo de la última operación aritmética efectuada.

• *Cero:* Se activa cuando el resultado de una operación aritmética es cero.

• *Acarreo:* Se activa cuando, como resultado de una suma o una resta, se produce un acarreo más allá del bit más significativo. Se utiliza en operaciones aritméticas de más de una palabra.

- *Igualdad:* Se activa si una comparación lógica da como resultado la igualdad.
- *Desbordamiento:* Empleado para seflalar un desbordamiento aritmético.
- *Habilitar/inhahilitar interrupciónes:* Empleado para habilitar o inhabilitar interrupciones. Cuando Las interrupciones están inhabilitadas, el procesador las ignora. Esto es muy deseable cuando el sistema operativo está ocupado en el tratamiento deotra interrupción.
- *Supervisor:* Indica si el procesador está ejecutando en modo supervisor o en modo usuarlo. Ciertas instrucciones privilegiadas sólo se pueden ejecutar en modo supervisor y sólo se puede tener acceso a ciertas áreas de memoria en modo supervisor.

En el diseño de un procesador específico, se pueden encontrar una serie de registros relacionados con el estado y el control. Además de la PSW, puede haber un puntero a un bloque de memoria que contenga información de estado adicional. En máquinas que utilizan varios tipos de interrupción, se puede ofrecer una serie de registros con punteros a cada rutina de tratamiento de interrupción. Si se utiliza una pila para implementar ciertas funciones (por ejemplo, las llamadas a procedimientos), entonces se necesita un puntero a la pila (ver Apéndice 1B). El hardware para la gestión de memoria que se discute en los capítulos 6 y 7 necesitará registros dedicados. Por último, los registros también pueden utilizarse para el control de las operaciones de E/S.

Una serie de factores inciden en el diseño de la organización de los registros de control y estado. Un punto clave es el soporte del sistema operativo. Cierto tipo de información de control es de utilidad específica para el sistema operativo. Si el diseñador del procesador dispone de una visión funcional del sistema operativo, la organización de los registros puede adaptarse convenientemente.

Otra decisión clave del diseño es la asignación de información de control a los registros y la memoria. Es habitual dedicar los primeros centenares o miles de palabras (las más bajas) de memoria para el control. EL diseñador debe decidir la cantidad de información de control que debe residir en los rápidos y costosos registros, junto a la cantidad que debe permanecer en memoria principal, que es más lenta y barata.

1 .3

EJECUCIÓN DE INSTRUCCIONES

La tarea básica que realiza un computador es la ejecución de los programas. El programa a ejecutar consta de un conjunto de instrucciones almacenadas en memoria. El procesador lleva a cabo el trabajo, ejecutando las instrucciones especificadas en el programa.

Para alcanzar una mayor comprensión de esta función y de la manera en que los componentes principales del computador interactúan para ejecutar un programa, hace falta analizar con cierto detalle los elementos de la ejecución de un programa. EL punto de vista más sencillo es considerar que el procesamiento de instrucciones consta de dos pasos. El procesador (1) trae las instrucciones desde la memoria, una cada vez y (2) ejecuta cada instrucción. La ejecución de un programa consiste en la repetición de este proceso de lectura y ejecución de la instrucción. La ejecución de la instrucción puede involucrar varias operaciones y depende de la naturaleza de la instrucción.

El procesamiento requerido para una instrucción simple se llama *ciclo de instrucción.* El ciclo de instrucción se representa en la figura 1.2, empleándose esta descripción simplifi-

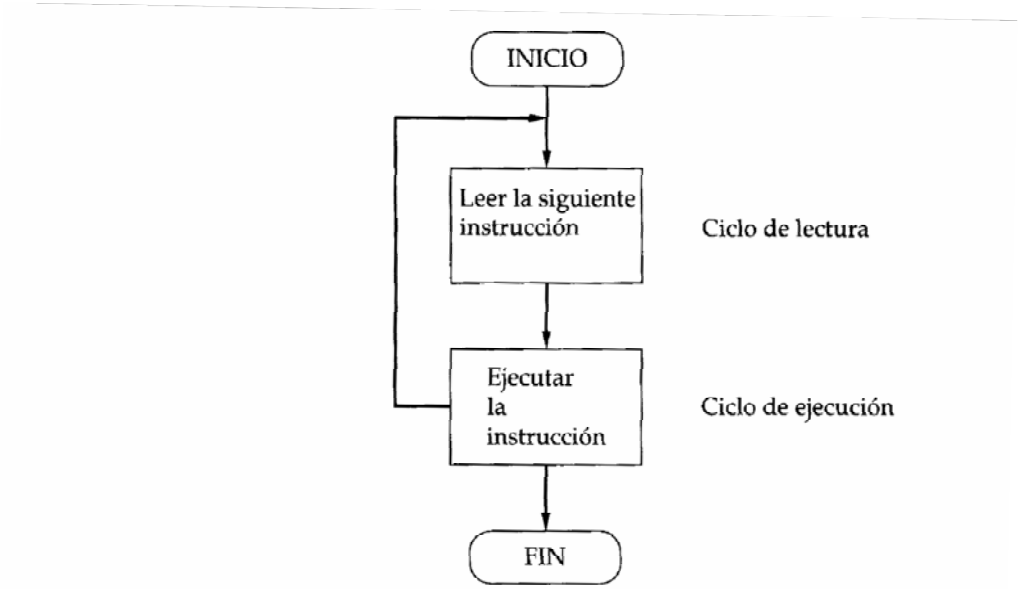


FIGURA 1.2 Ciclo básico de instrucción

cada de dos pasos que se acaba de explicar. Los dos pasos se llaman *ciclo de lectura (fetch) y ciclo de ejecución.* La ejecución del programa se detiene sólo si se apaga la máquina, ocurre algún tipo de error irrecuperable o se encuentra una instrucción en el programa que detiene el computador.

Lectura y ejecución de instrucciones

Al comienzo de cada ciclo de instrucción, el procesador lee una instrucción de la memoria. En un procesador típico habrá un registro llamado *contador de programa* (PC), que se usa para llevar la cuenta de cuál es la próxima instrucción a leer. A menos que se diga otra cosa, el procesador siempre incrementará el PC después de leer cada instrucción, de forma que después se lea la instrucción siguiente en la secuencia (es decir, la instrucción ubicada en la dirección inmediatamente superior de la memoria). Así pues, considérese por ejemplo un computador en la que cada instrucción ocupa una palabra de memoria de 16 bits. Supóngase que el contador de programa apunta a la ubicación 300. La próxima instrucción que va a leer el procesador es la que está en la ubicación 300. En los siguientes ciclos de instrucción, leerá las instrucciones de las ubicaciones 301, 302, 303 y así sucesivamente. Como se verá en breve, esta secuencia puede alterarse.

La instrucción leída se carga en un registro del procesador conocido como *registro de instrucción* (IR). La instrucción está en forma de código binario que especifica cuál es la acción que el procesador llevará a cabo. El procesador interpreta la instrucción y realiza la acción requerida. En general, estas acciones pueden clasificarse en las siguientes cuatro categorías:

- *Procesador-memoria:* Se transfieren datos del procesador a la memoria o viceversa.
- *Procesador-EIS:* Se transfieren datos desde o hacia un dispositivo periférico, realizándose la transferencia entre el procesador y un módulo de E/S.

• *Tratamiento de datos:* El procesador realiza alguna operación aritmética o lógica sobre los datos,

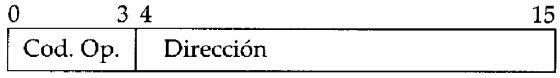
• *Control:* La instrucción pide que se altere la secuencia de ejecución. Por ejemplo, el procesador puede leer una instrucción de la ubicación 149, la cual especifica que la próxima instrucción sea la de la ubicación 182. El procesador recordará este hecho ajus-tando el valor del contador de programa a 182. De este modo, en el próximo ciclo de lectura, se traerá la instrucción de la ubicación 182 y no de la 150,

Por supuesto que la ejecución de una instrucción puede incluir una combinación de estas acciones.

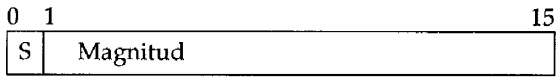
Como ejemplo sencillo, se considera una máquina hipotética que incluya las características enumeradas en la figura 1.3. El procesador contiene un único registro de datos, llamado *acumulador* (AC). Tanto las instrucciones como los datos son de 16 bits de longitud. Así pues, es conveniente organizar la memoria utilizando ubicaciones o palabras de 16 bits. El formato de instrucción dedica cuatro bits para el código de la operación (cod-op). Por tanto, pueden haber hasta $2^4 = 16$ códigos de operación diferentes y hasta $2^{12} = 4096$ (4K) palabras de memoria que se pueden direccionar directamente.

La figura 1.4 ilustra la ejecución parcial de un programa, mostrando las zonas pertinentes de la memoria y los registros del procesador. El fragmento de programa que se muestra suma el contenido de la palabra de memoria de la dirección 940 al contenido de la palabra de memoria de la dirección 941 y almacena el resultado en esta última dirección. Se requieren tres instrucciones, que se pueden describir con tres ciclos de lectura y tres de ejecución:

1. El PC contiene 300, la dirección de la primera instrucción. Se carga el contenido de la ubicación 300 en el IR. Nótese que este proceso podría involucrar el uso de un MAR y un MBR. Por simplicidad, se van a ignorar estos registros intermedios.



(a) Formato de instrucción



(b) Formato de un entero

Contador de programa (PC) = Dirección de la instrucción
Registro de instrucción (IR) = Instrucción que está ejecutándose
Acumulador (AC) = Almacenamiento temporal

(c) Registros internos de la CPU

0001 = Cargar de la memoria al AC
0010 = Almacenar el AC en memoria
0101 = Sumar al AC el contenido de la memoria

(d) Lista parcial de los códigos de operación

FIGURA 1.3 Características de una máquina hipotética

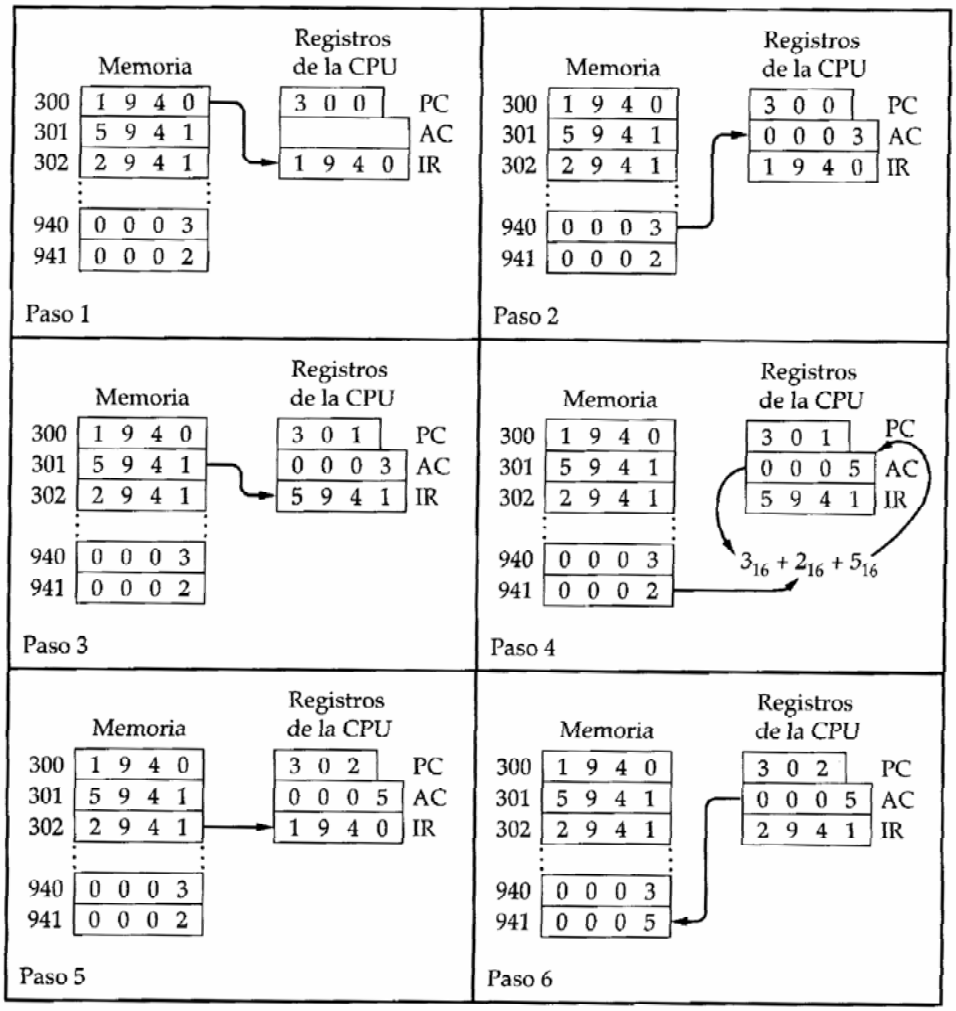


FIGURA 1.4 Ejemplo de ejecución de un programa

2. Los primeros 4 bits del IR indican que se cargará el AC. Los 12 bits restantes especifican la dirección, que es 940.

3. Se incrementa el PC y se lee la instrucción siguiente,

4. El contenido anterior del AC y el contenido de la ubicación 941 se suman y el resultado se almacena en el AC.

5. Se incrementa el PC y se lee la instrucción siguiente.

6. El contenido *del AC se almacena* en *la ubicación 941*.

En este ejemplo se necesitan tres ciclos de instrucción, donde cada uno consta de un ciclo de lectura y otro de ejecución, para sumar el contenido de la ubicación 940 al contenido de la ubicación 941. Con un conjunto de instrucciones más complejo harían falta menos ciclos. La mayoría de los procesadores actuales aportan instrucciones que incluyen más de una dirección. De esta manera, en el

ciclo de ejecución de una instrucción particular pueden participar más de una referencia a memoria. Además, en vez de referencias a memoria, una instrucción puede especificar una operación de E/S.

Funciones de E/S

Hasta ahora se ha discutido que el funcionamiento del computador es controlado por el procesador y se ha analizado en primer lugar la interacción entre el procesador y la memoria. En la discusión sólo se ha aludido al papel de la componente de E/S.

Los módulos de E/S (un controlador de disco, por ejemplo) pueden intercambiar datos directamente con el procesador. Al igual que el procesador puede iniciar una lectura o escritura en la memoria, indicando la dirección de una ubicación específica, el procesador también puede leer datos de un módulo de E/S o escribir datos en el módulo. En este último caso, el procesador identifica a un dispositivo específico que es controlado por un módulo de E/S determinado. De este modo se podría tener una secuencia de instrucciones similar a la de la figura 1.4, pero con instrucciones de E/S en lugar de referencias a memoria.

En algunos casos, es conveniente permitir que los intercambios de E/S se produzcan directamente con la memoria. En tal caso, el procesador dará autoridad a un módulo de E/S para leer o escribir en memoria, de modo que la transferencia de E/S ocurre sin obstruir al procesador. Durante la transferencia, el módulo de E/S emite órdenes de lectura o escritura en la memoria, librando de responsabilidades al procesador en el intercambio. Esta operación se conoce como *acceso directo a memoria* (DMA, *Direct Memory Access)* y será examinada más adelante en este mismo capítulo. Por ahora, todo lo que se necesita saber es que la estructura de interconexión del computador puede que tenga que permitir una interacción directa entre la memoria y la E/S.

1.4 INTERRUPCIONES

Casi todos los computadores tienen un mecanismo mediante el cual otros módulos (E/S, memoria) pueden interrumpir la ejecución normal del procesador. La tabla 1.1 enumera las clases más comunes de interrupciones.

Las interrupciones aparecen, principalmente, como una vía para mejorar la eficiencia del procesamiento. Por ejemplo, la mayoría de los dispositivos externos son mucho más lentos

TABLA 1.1 Clases de Interrupciones

De programa	Generadas por alguna condición que se produce como resultado de la ejecución de una instrucción, como el desbordamiento aritmético, la división por cero, el intento de ejecutar una instrucción ilegal de la máquina o una referencia a una zona de memoria fuera del espacio permitido al usuario.
De reloj	Generadas por un reloj interno del procesador. Esto permite al sistema operativo llevar a cabo ciertas funciones con determinada regularidad.
De E/S	Generadas por un controlador de E/S, para indicar que una operación ha terminado normalmente o para indicar diversas condiciones de error.
Por fallo del hardware	Generadas por fallos tales como un corte de energía o un error de paridad de la memoria.

que el procesador. Supóngase que el procesador está transfiriendo datos hacia una impresora, utilizando un esquema para el ciclo de instrucción como el de la figura 1.2. Después de cada operación ESCRIBIR, el procesador hará una pausa y permanecerá desocupado hasta que la impresora se ponga al corriente. La duración de esta pausa puede ser del orden de varios cientos o incluso miles de ciclos de instrucción en los que la memoria no está implicada. Está claro que esto es un derroche en la utilización del procesador.

La figura 1 -5 a ilustra este estado de las cosas para la aplicación que se indica en el párrafo anterior. El programa de usuario lleva a cabo una serie de llamadas a ESCRIBIR, intercaladas con el procesamiento. Los segmentos de código 1, 2 y 3 se refieren a secuencias de instrucciones que no implican E/S. Las llamadas a ESCRIBIR son, en realidad, llamadas a un programa de E/S, que es una utilidad del sistema que llevará a cabo la operación concreta de E/S. El programa de E/S consta de tres secciones:

• Una secuencia de instrucciones, etiquetada con un 4 en la figura, de preparación para la operación concreta de E/S, Esto puede incluir la copia de los datos de salida hacia un buffer especial y preparar los parámetros de la orden del dispositivo.

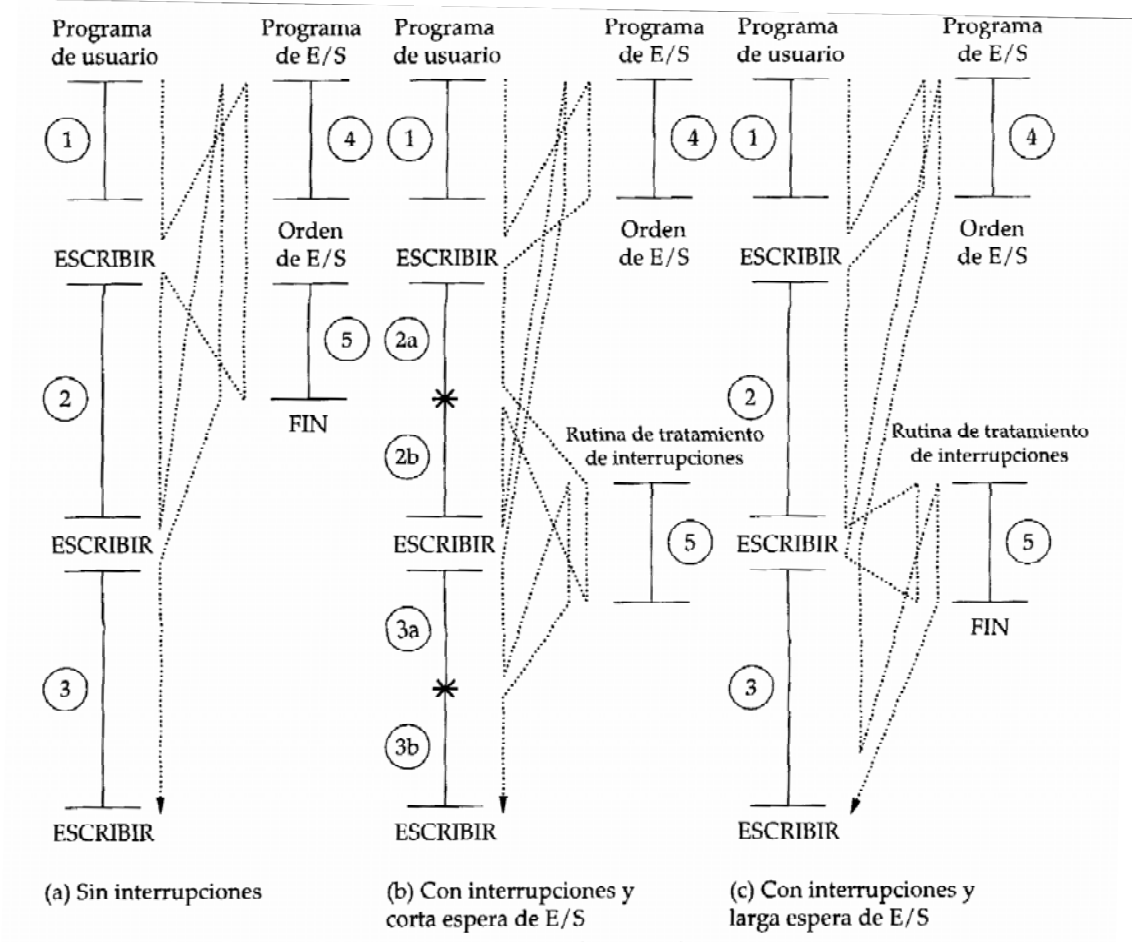


FIGURA 1.5 Flujo de control del Programa sin y con interrupciones

- La orden concreta de E/S. Sin el uso de interrupciones, una vez que se emita esta orden, el programa debe esperar a que el dispositivo de E/S lleve a cabo la función pedida. El programa puede esperar simplemente ejecutando de forma repetida una operación que compruebe si ya se realizó la E/S.
- Una secuencia de instrucciones, etiquetada con Un *5* en La figura, para completar la operación. Esto puede incluir la activación de un código de condición que indique el éxito o el fracaso de la operación.

Debido a que la operación de E/S puede tardar un tiempo relativamente grande en terminar, el programa de E/S puede quedar colgado esperando a que se complete la operación; así pues, el programa de usuario se detendrá por un tiempo considerable en el momento de la llamada a ESCRIBIR.

Las interrupciones y el ciclo de instrucción

Con las interrupciones, el procesador se puede dedicar a la ejecución de otras instrucciones mientras una operación de E/S está en proceso. Considérese el flujo de control de la figura 1 .5b. Al igual que antes, el programa de usuario alcanza un punto en el que hace una llamada al sistema en forma de una llamada ESCRIBIR. El programa de E/S que se invoca consta solo del código de preparación y de la orden concreta de E/S. Después de que se ejecuten estas pocas instrucciones, se devuelve el control al programa de usuario. Mientras tanto, el dispositivo externo estará ocupado recibiendo datos desde la memoria del computador e imprimiéndolos. Esta operación de E/S se lleva a cabo concurrentemente con la ejecución de las instrucciones del programa de usuario.

Cuando el dispositivo de E/S esté disponible, es decir, cuando esté preparado para aceptar más datos desde el procesador, el módulo de E/S de dicho dispositivo enviará una señal de *solicitud de interrupción* al procesador. El procesador responde suspendiendo la operación del programa en curso y saltando a un programa que da servicio al dispositivo de E/S en particular, conocido como *rutina de tratamiento de la interrupción (interrupt handler),* reanudando la ejecución original después de haber atendido al dispositivo. En La figura 1 .*5b,* el instante en que se produce tal interrupción viene indicado con un asterisco **(*).**

Desde el punto de vista del programa de usuario, una interrupción es solamente eso: una interrupción de la secuencia normal de ejecución. Cuando el tratamiento de la interrupción se termina, la ejecución continúa (figura 1.6). Así pues, el programa de usuario no tiene que disponer de ningún código especial para dar cabida a las interrupciones; el procesador y el sistema operativo son los responsables de suspender el programa de usuario y reanudarlo después en el mismo punto.

Para dar cabida a las interrupciones, se añade un *ciclo de interrupción* al ciclo de instrucción, como se muestra en la figura 1.7. En el ciclo de interrupción, el procesador comprueba si ha ocurrido alguna interrupción, lo que se indicará con la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador sigue con el ciclo de lectura y trae la próxima instrucción del programa en curso. Si hay una interrupción pendiente, el procesador suspende la ejecución del programa en curso y ejecuta una rutina de *tratamiento de la interrupción.*

La rutina de tratamiento de la interrupción forma parte generalmente del sistema operativo. Normalmente este programa determina la naturaleza de la interrupción y realiza cuantas acciones sean necesarias. De hecho, en el ejemplo que se ha estado siguiendo, la rutina de tratamiento determina el módulo de E/S que generó la interrupción y puede saltar a un programa que escribirá más datos a dicho módulo. Cuando termina la rutina de tratamiento de la interrupción, el procesador puede reanudar la ejecución del programa de usuario en el punto en que sucedió la interrupción.

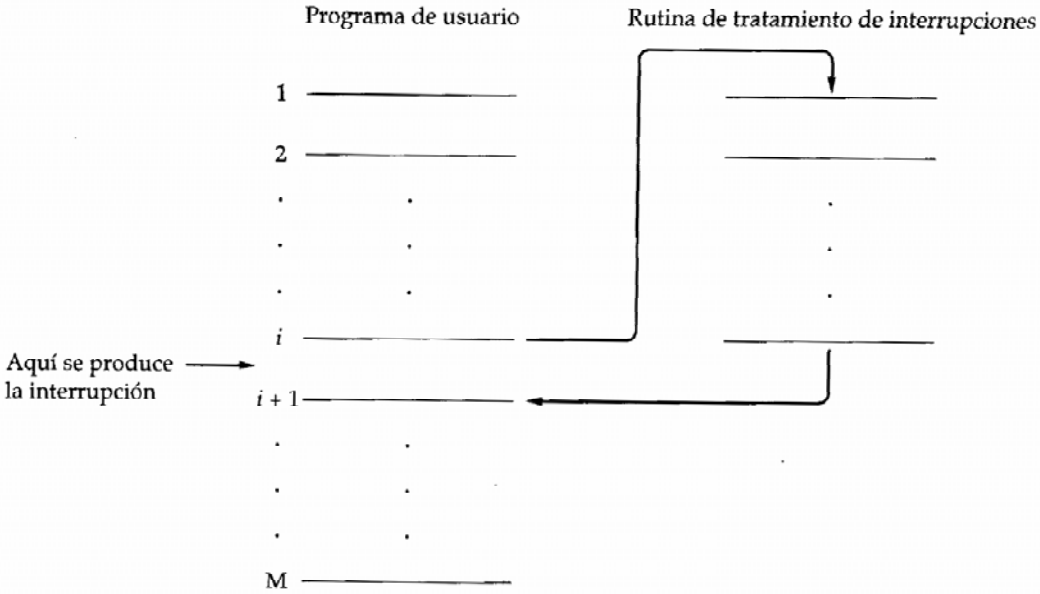


FIGURA 1.6 Transferencia de control por interrupciones

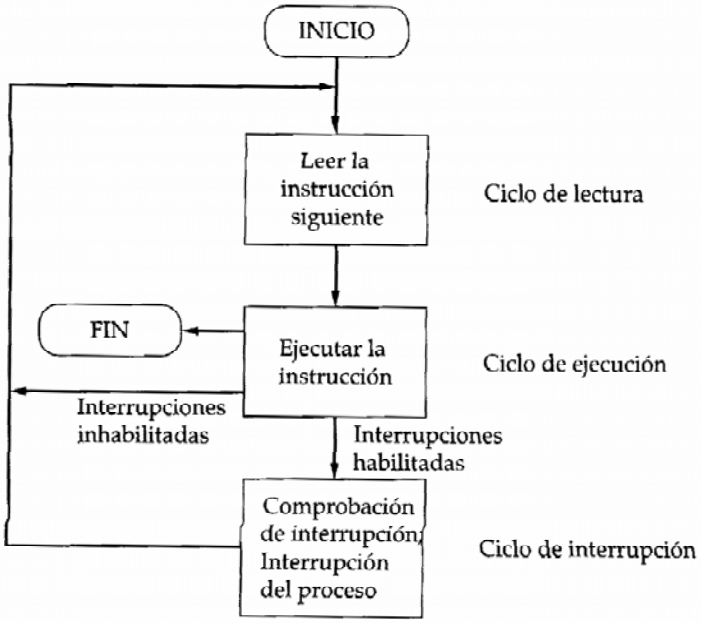


FIGURA 1.7 Ciclo de instrucción con interrupciones

Está claro que hay cierta sobrecarga en este proceso. Se deben ejecutar instrucciones extra (en la rutina de tratamiento de la interrupción) para determinar la naturaleza de la interrupción y decidir la acción apropiada. Sin embargo, por la cantidad de tiempo que se desperdicia esperando en una operación de E/S, puede aprovecharse el procesador de una manera mucho más eficaz con el uso de interrupciones.

Para apreciar el aumento en la eficiencia, considérese la figura 1.8, que es un diagrama de tiempos basado en el flujo de control de las figuras 1.5a y 1.5b.

En las figuras 1.5b y 1.8 se supone que el tiempo exigido para la operación de E/S es relativamente pequeño: menos que el tiempo para completar la ejecución de las instrucciones situadas entre dos operaciones ESCRIBIR del programa de usuario. El caso más normal, especialmente para un dispositivo lento, como es el caso de una impresora, es aquél en que la operación de E/S tardará mucho más tiempo que la ejecución de una secuencia de instruc-

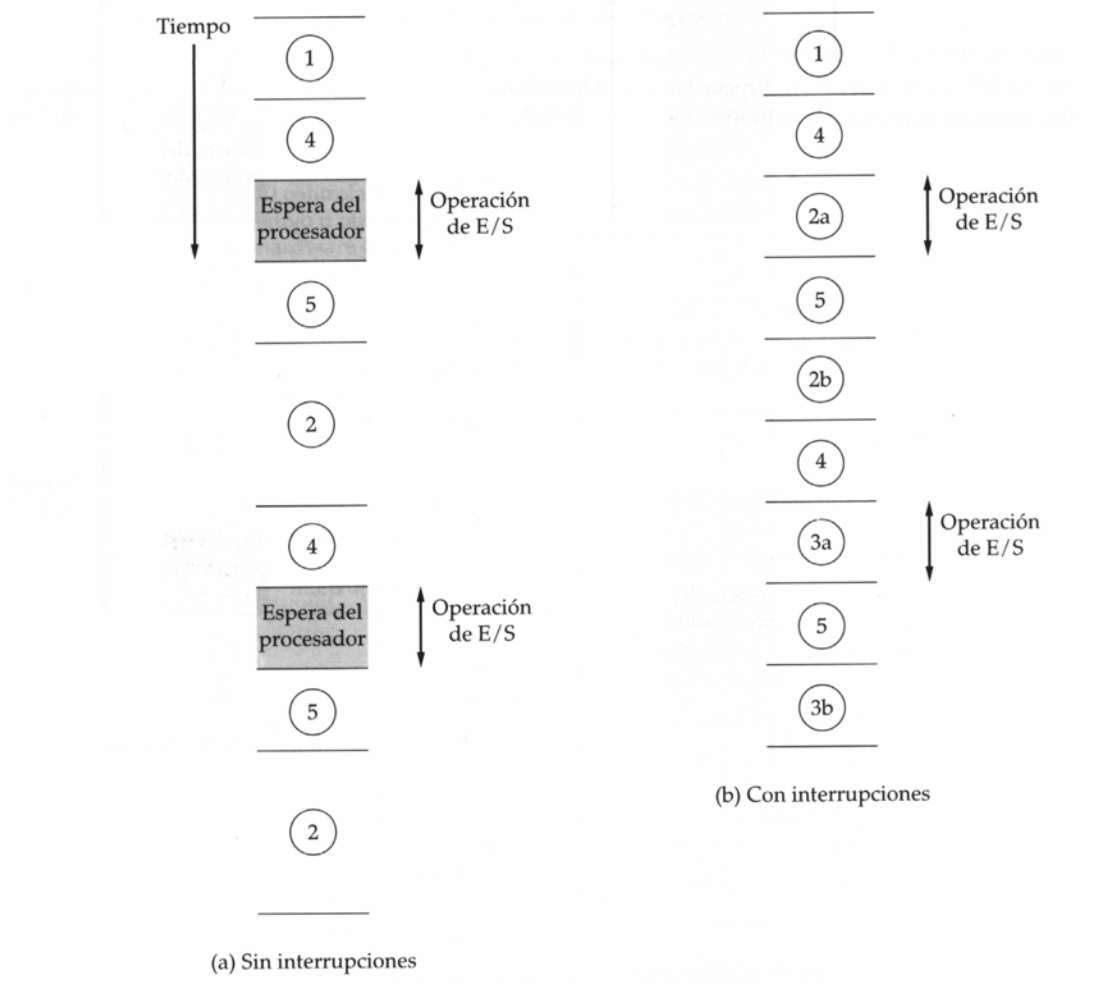


FIGURA 1.8 Diagrama de tiempos de un programa; espera corta de E/S

clones del usuario. La figura 1.5c muestra esta situación. En este caso, el programa de usuario alcanza la segunda llamada a ESCRIBIR antes de que se complete la operación de E/S desencadenada por la primera llamada. El resultado es que el programa de usuario se suspende en este punto. Cuando termine la operación anterior de E/S, se podrá procesar esta nueva llamada a ESCRIBIR y así comenzar una nueva operación de E/S. La figura 1.9 muestra el diagrama de tiempos para esta situación, con y sin uso de interrupciones. Se

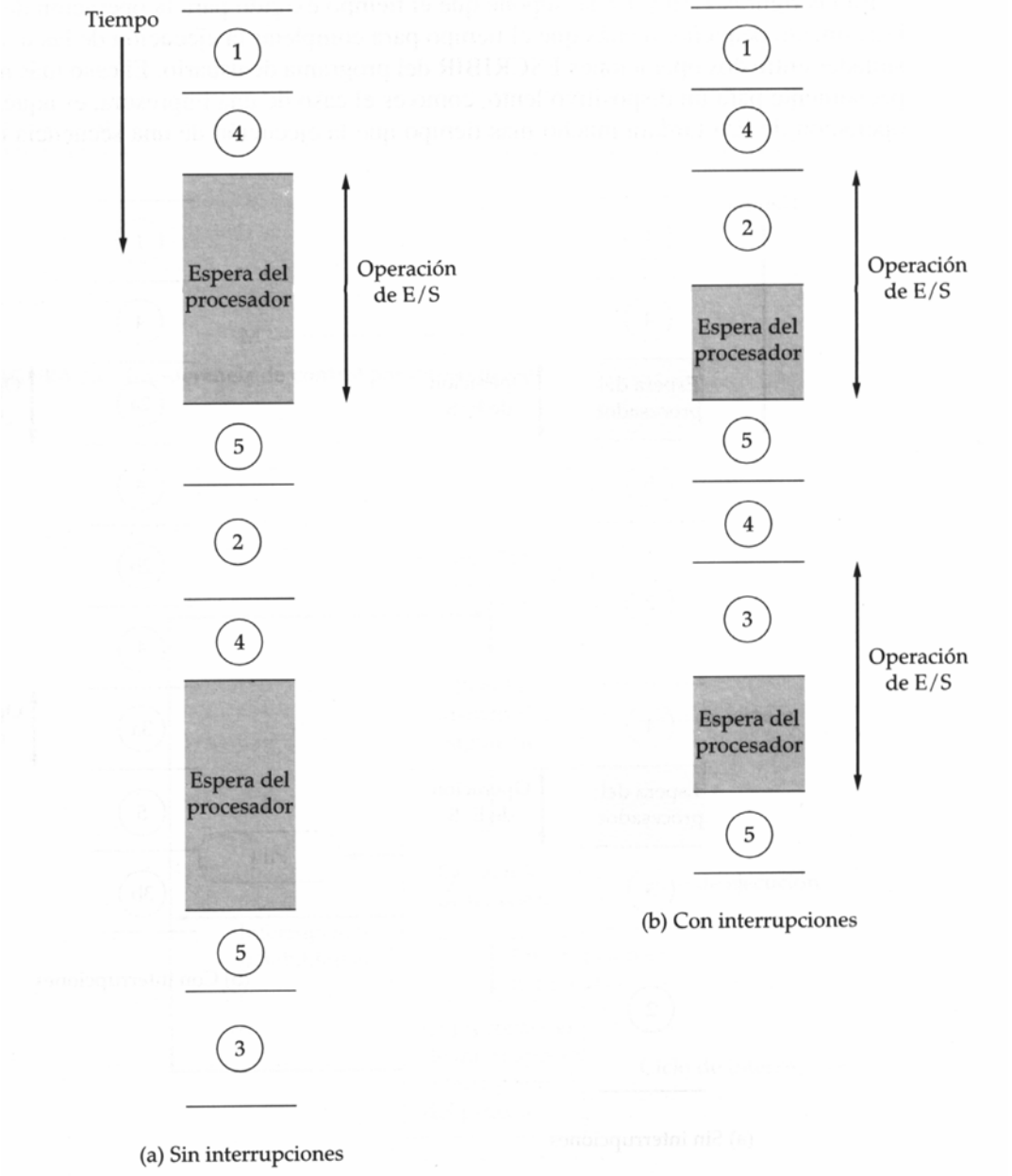


Figura 1.9 Diagramas de tiempo de un programa; espera larga de E/S

puede ver que así se produce un aumento de la eficiencia, pues parte del tiempo en que la operación de E/S está en marcha se solapa con la ejecución de las instrucciones del usuario.

Tratamiento de interrupciones

El acontecimiento de una interrupción desencadena una serie de sucesos, tanto en el hardware del procesador como en el software. La figura 1.10 muestra una secuencia típica. Cuando un dispositivo de E/S completa una operación de E/S, se produce en el hardware la siguiente secuencia de sucesos:

1. El dispositivo emite una señal de interrupción al procesador.
2. El procesador finaliza la ejecución de la instrucción en curso antes de responder a la interrupción, tal como se indica en la figura 1.7.
3. El procesador pregunta por la interrupción, comprueba que hay una y envía una señal de reconocimiento al dispositivo que generó la interrupción. Este reconocimiento le permite al dispositivo suprimir la señal de interrupción.
4. El procesador necesita ahora prepararse para transferir el control a la rutina de interrupción. Para empezar, hace falta salvar la información necesaria para reanudar la ejecución del programa en curso en el punto de la interrupción. La mínima información

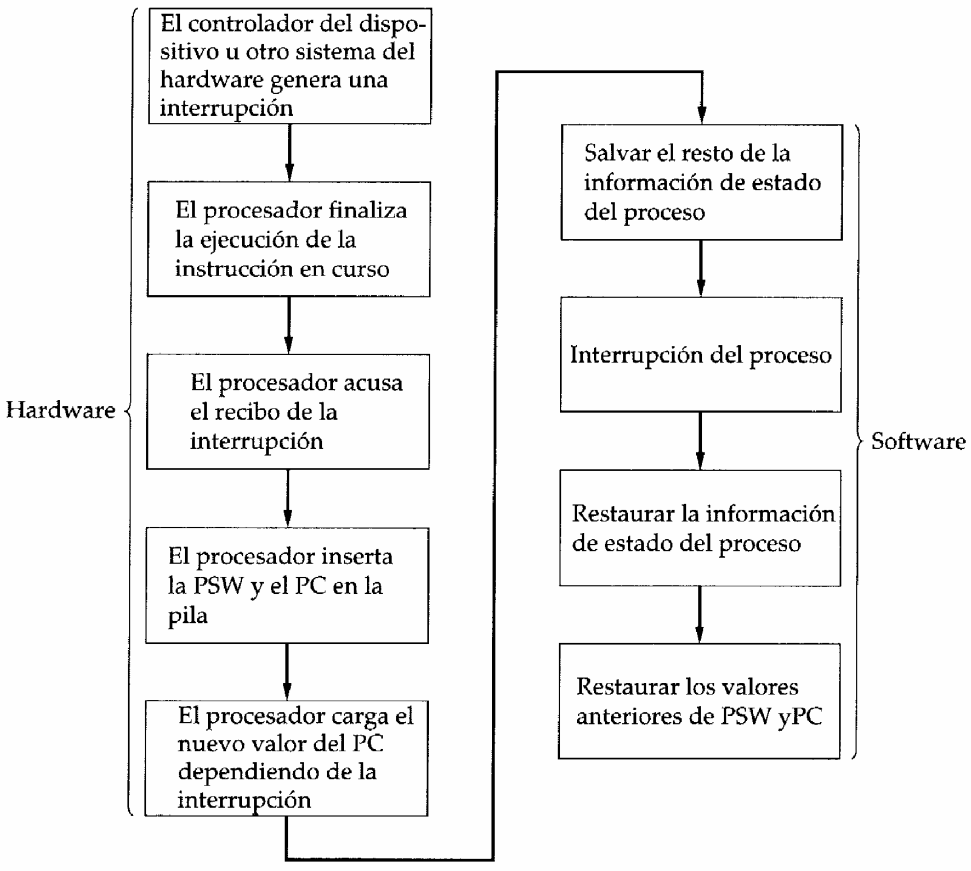


Figura 1.10 Tratamiento de una interrupción simple

requerida es la palabra de estado del programa (PSW) y la ubicación de la próxima instrucción a ejecutar, que se almacena en el contador de programa. Estos pueden meterse en la pila de control del sistema (ver el Apéndice IB).

5. El procesador carga ahora el contador de programa con la ubicación de entrada del programa de tratamiento de la interrupción. Dependiendo de la arquitectura del computador y del diseño del sistema operativo, puede haber un solo programa por cada tipo de interrupción, o uno por cada dispositivo y por cada tipo de interrupción. Si hay más de una rutina de tratamiento de interrupción, el procesador debe determinar a cuál invocar. Esta información pudiera estar incluida en la señal original de la interrupción, o el procesador debe preguntarle al dispositivo que creó la interrupción para obtener respuesta sobre la información que necesita.

Una vez que se ha cargado el contador de programa, el procesador procede con el próximo ciclo de instrucción, que comienza trayendo la próxima instrucción. Debido a que esta instrucción se determina por el contenido del contador de programa, el resultado es que el control se le transfiere al programa que trata la interrupción. La ejecución de este programa se traduce en las operaciones siguientes:

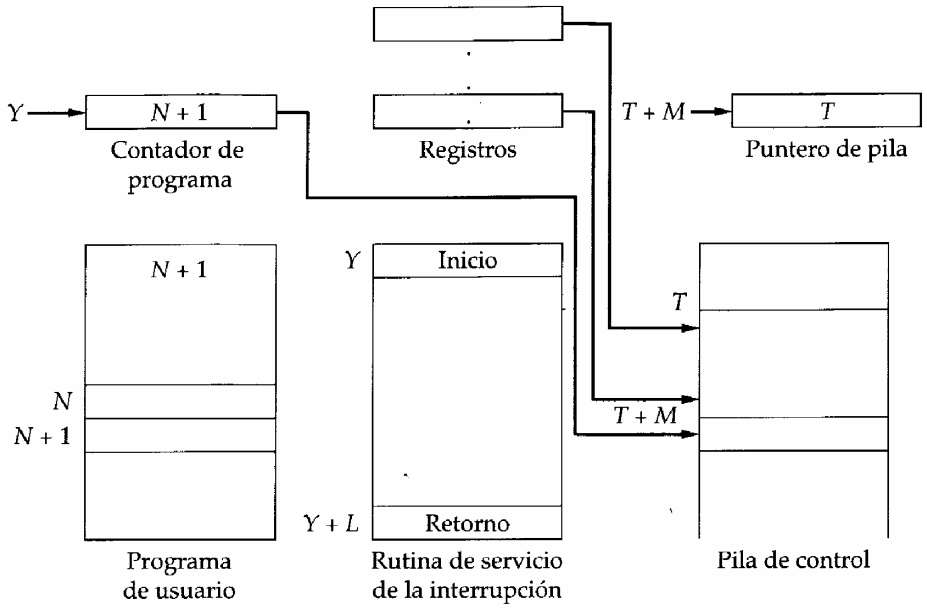
6. En este punto, el contador de programa y la PSW relativa al programa interrumpido han sido salvadas en la pila del sistema. Sin embargo, hay más información que es considerada parte del "estado" de ejecución del programa. En particular se necesita salvar el contenido de los registros del procesador ya que estos registros pudieran ser utilizados por la rutina de tratamiento de la interrupción. Así pues es necesario salvar todos estos valores más cualquier otra información sobre el estado. Normalmente la rutina de tratamiento de la interrupción comienza salvando en la pila el contenido de todos los registros. En el capítulo 3 se discute sobre otra información del estado que también puede que tenga que salvarse. La figura l.lla muestra un ejemplo simple. En este caso, se muestra un programa de usuario que es interrumpido después de la instrucción de la ubicación N. El contenido de todos los registros más la dirección de la próxima instrucción (N+l) son guardados en la pila. El puntero a la pila se actualiza para que apunte a la nueva cima y el contador de programa se actualiza para que apunte al comienzo de la rutina de servicio de la interrupción.

7. La rutina de tratamiento de la interrupción puede ahora proceder a procesar la interrupción. Esto incluirá un examen del estado de la información relativa a la operación de E/S o a cualquier otro evento que haya causado la interrupción. Esto puede también involucrar el envío adicional de órdenes o reconocimientos al dispositivo de E/S.

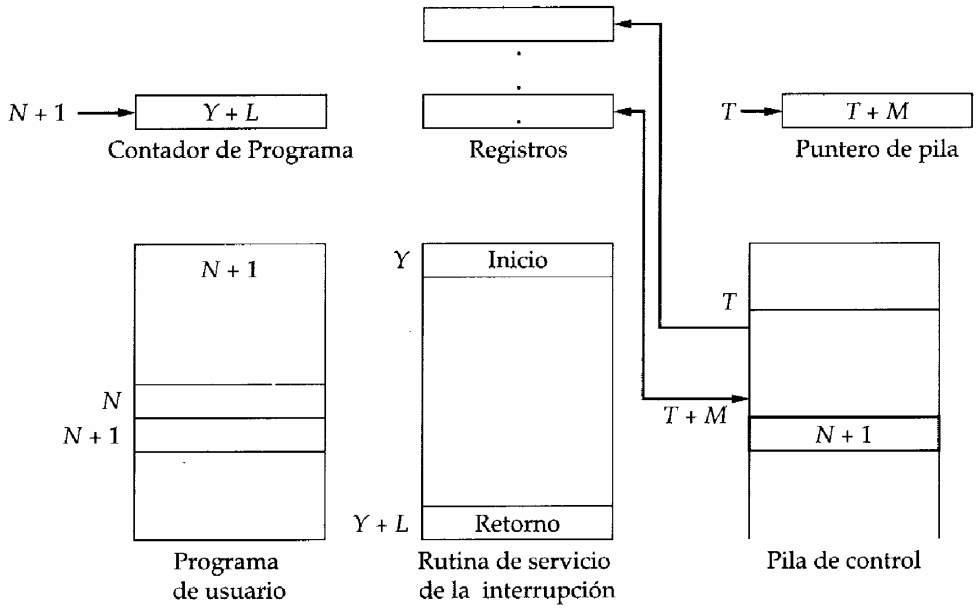
8. Cuando se completa el tratamiento de la interrupción, se recuperan de la pila los valores de los registros que se salvaron y se restauran los registros (ver por ejemplo la figura 1.1 Ib).

9. El acto final es restaurar los valores de la PSW y del contador de programa a partir de la pila. Como resultado, la próxima instrucción a ser ejecutada será del programa interrumpido previamente.

Es importante salvar toda la información sobre el estado del programa interrumpido para su reanudación posterior, porque la rutina de tratamiento de la interrupción no es una rutina llamada desde el programa. Por el contrario, la interrupción puede producirse en cualquier momento y por tanto en cualquier punto de la ejecución de un programa de usuario. Su ocurrencia es impredecible. Más aún, como se verá más adelante en esta sección, los dos programas pueden no tener nada en común y pueden pertenecer a dos usuarios diferentes.



(a) La interrupción se produce después de la instrucción de la posición N



(b) Retorno de la interrupción

FIGURA 1.11 cambios en la memoria y en los registros durante una interrupción

Interrupciones múltiples

En la discusión anterior se ha analizado el acontecimiento de una sola interrupción. Sin embargo, supóngase que pueden producirse múltiples interrupciones. Por ejemplo, un programa puede estar recibiendo datos de una línea de comunicaciones e imprimiendo resultados. La impresora generará una interrupción cada vez que se completa una operación de impresión. El controlador de la línea de comunicaciones generará una interrupción cada vez que llega una unidad de datos. Esta unidad puede ser un solo carácter o un bloque, dependiendo de la naturaleza de las normas de comunicación. En cualquier caso, es posible que se produzca una interrupción del controlador de comunicaciones cuando se está procesando una interrupción de la impresora.

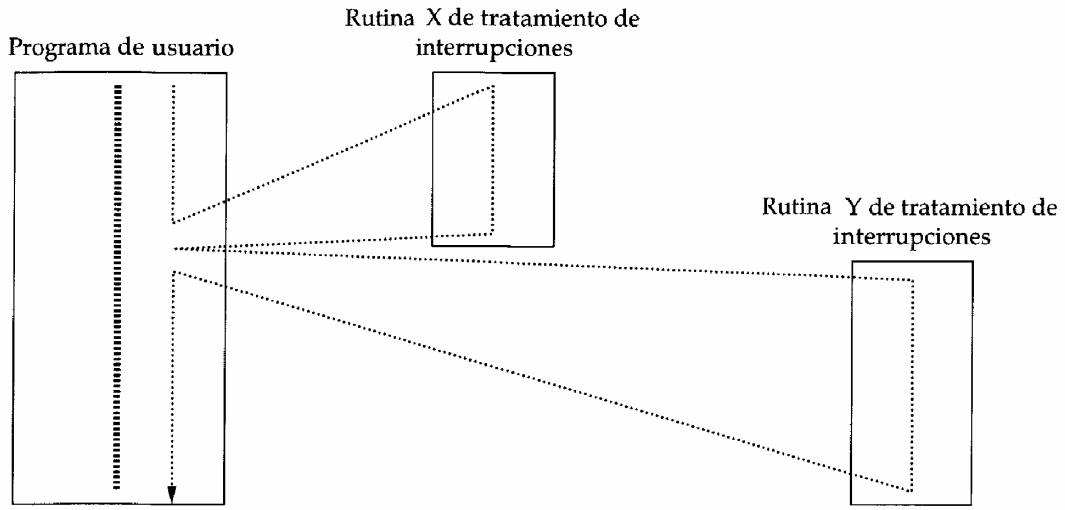
Hay dos enfoques para tratar las interrupciones múltiples. El primero es inhabilitar las interrupciones mientras se esté procesando una. Una *interrupción inhabilitada* quiere decir que el procesador ignorará la señal de interrupción. Si durante este tiempo se produce una interrupción, ésta generalmente quedará pendiente y será comprobada por el procesador después que éste habilite las interrupciones. De este modo, cuando un programa de usuario está ejecutándose y se produce una interrupción, se inhabilitan inmediatamente las interrupciones. Después de terminar la rutina que trata la interrupción, se habilitan las interrupciones antes de reanudar el programa de usuario y el procesador comprueba si se ha producido alguna interrupción adicional. Este enfoque es simple y elegante porque las interrupciones se tratan en un orden estrictamente secuencial (figura 1.12a).

La limitación de este enfoque es que no tiene en cuenta las prioridades relativas o necesidades críticas en tiempo. Por ejemplo, cuando llega una interrupción desde la línea de comunicaciones, se necesita atender ésta rápidamente para hacer sitio a nuevas entradas. Si el primer lote de entrada no ha sido aún procesado cuando llega el segundo, pueden perderse datos.

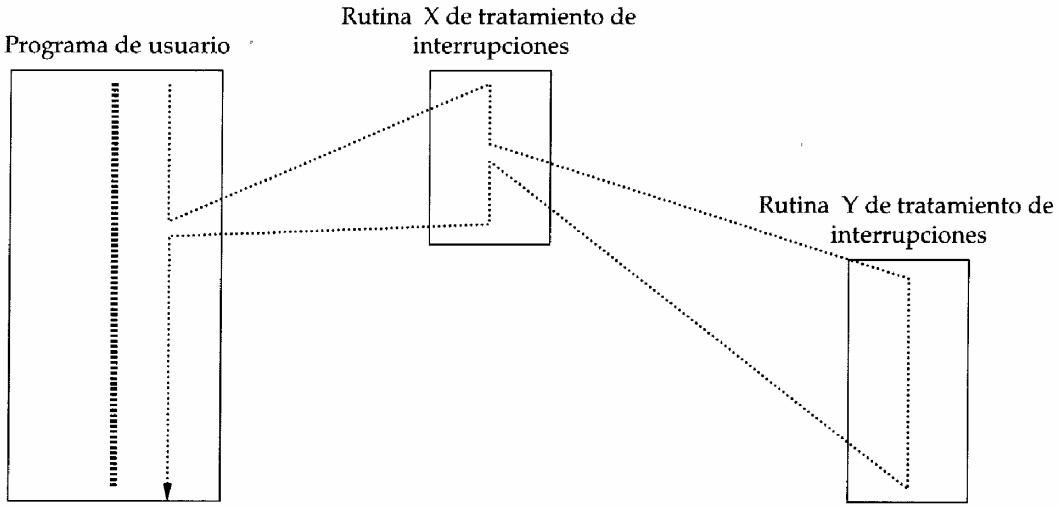
Un segundo enfoque es definir prioridades para las interrupciones y permitir que una interrupción de una prioridad más alta pueda interrumpir a la rutina de tratamiento de una interrupción de prioridad más baja (figura 1.12b).

Como ejemplo de este segundo enfoque, considérese un sistema con tres dispositivos de E/S: una impresora, un disco y una línea de comunicaciones, con prioridad creciente de 2,4 y 5, respectivamente. La figura 1.13 ilustra una posible secuencia. Un programa de usuario comienza en $t = 0$. En $t= 10$, se produce una interrupción de la impresora; la información del usuario se coloca en la pila del sistema y la ejecución continúa en la *rutina de servicio de la interrupción* (ISR, *Interrupt Service Routine)* de la impresora. Mientras esta rutina sigue ejecutándose, en $t = 15$, se produce una interrupción de comunicaciones. Como la línea de comunicaciones tiene una prioridad más alta que la impresora, la interrupción es atendida. La ISR es interrumpida, su estado se pone en la pila y la ejecución continúa en la ISR de comunicaciones. Mientras esta rutina está ejecutándose, se produce una interrupción del disco ($t = 20$). Debido a que esta interrupción es de más baja prioridad, queda retenida y la ISR de comunicaciones se ejecuta hasta el final.

Cuando se completa la ISR de comunicaciones *(*$t = 25$), se restaura el estado previo del procesador, lo que significa continuar ejecutando la ISR de la impresora. Sin embargo, antes de ejecutar una sola instrucción de esta rutina, el procesador atiende a la mayor prioridad de la interrupción del disco y transfiere el control a la ISR del disco. Sólo cuando se completa esta rutina ($t = 35$) se reanuda la ISR de la impresora. Cuando se termina esta última *(*$t = 40$), el control vuelve finalmente al programa de usuario.



(a) Tratamiento secuencial de interrupciones



(b) Tratamiento anidado de interrupciones

FIGURA 1.12 Transferencia de control con múltiples interrupciones

Multiprogramación

Aún con el uso de interrupciones, puede que un procesador no esté aprovechado de una manera muy eficiente. Por ejemplo, considérese de nuevo la figura 1.9b. Si el tiempo necesario para completar una operación de E/S es mucho mayor que el código del usuario entre llamadas de E/S (una situación habitual), entonces el procesador va a estar desocupado durante gran parte del tiempo. Una solución a este problema es permitir que varios programas de usuario estén activos a un mismo tiempo.

Supóngase, por ejemplo, que el procesador tiene que ejecutar dos programas. Uno de ellos es simplemente un programa que lee datos de la memoria y los saca por un dispositivo

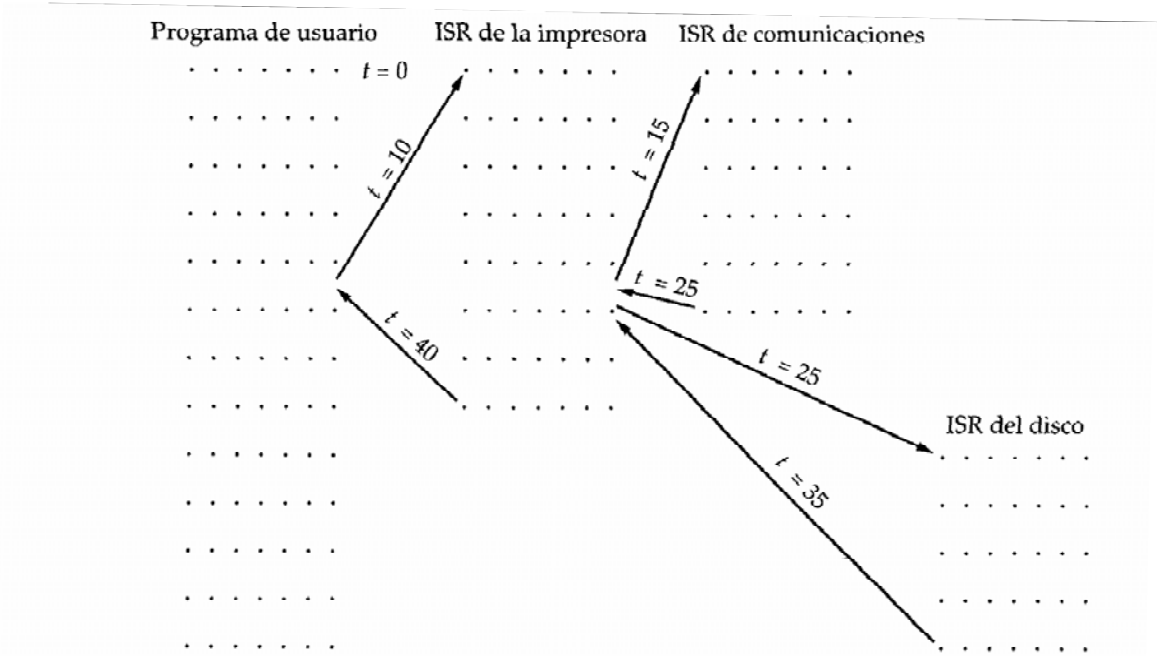


FIGURA 1.13 Ejemplo de secuencia de tiempo para interrupciones múltiples [TANE90]

externo; el otro es una aplicación que supone una gran cantidad de cálculos. El procesador puede dar comienzo al programa de salida, enviar una orden ESCRIBIR al dispositivo exterrno y luego comenzar la ejecución de la otra aplicación.

Cuando el procesador tiene que tratar con una serie de programas, la secuencia en que estos se ejecutan dependerá de su prioridad relativa y de si están esperando una E/S. Cuando un programa es interrumpido y se transfiere el control a la rutina de tratamiento de la interrupción, una vez que ésta haya terminado, puede que no se devuelva el control inmediatamente al programa de usuario que estaba ejecutándose en el momento de la interrupción. En su lugar, el control puede transferirse a algún otro programa pendiente que tenga mayor prioridad. Finalmente, cuando tenga la prioridad más alta, se reanudará el programa de usuario que fue interrumpido. Esta situación de varios programas que se ejecutan por turnos se conoce como *multiprogramación* y será discutida más adelante en el capítulo 2.

1.5 JERARQUÍA DE MEMORIA

Las limitaciones de diseño de la memoria de un computador se pueden resumir en tres preguntas: ¿qué cantidad?, ¿qué velocidad? y ¿qué coste?

La cuestión de qué cantidad está relativamente abierta. Según sea la capacidad, probablemente se construirán aplicaciones que la utilicen. La cuestión de la velocidad es, en cierto sentido, fácil de responder. Para lograr un mayor rendimiento, la memoria debe ser capaz de ir al ritmo del procesador. Es decir, mientras el procesador está ejecutando instrucciones, se-

ría conveniente no tener que hacer pausas esperando a instrucciones u operandos. La última pregunta también hay que tenerla en cuenta. Para un sistema práctico, el coste de la memoria debe ser razonable en relación a los otros componentes.

Como se puede suponer, estas tres características compiten entre sí: coste, capacidad y tiempo de acceso. Desde siempre se ha utilizado una gran variedad de tecnologías para implementar los sistemas de memoria. A lo largo de este abanico de tecnologías, se cumplen las siguientes relaciones:

- A menor tiempo de acceso, mayor coste por bit
- A mayor capacidad, menor coste por bit
- A mayor capacidad, mayor tiempo de acceso

El dilema que se plantea el diseñador es evidente. El diseñador desearía usar tecnologías de memoria que le ofrezcan una gran capacidad, porque se necesita y para que el coste por bit sea bajo. Sin embargo, para cumplir con los requisitos de rendimiento, puede necesitar usar memoria cara, de capacidad relativamente menor y con tiempos de acceso rápidos.

La salida a este dilema no es depender de un único componente de memoria o una tecnología, sino emplear *una jerarquía de memoria.* En la figura 1.14a se ilustra una jerarquía tradicional. A medida que se desciende por la jerarquía se tienen las siguientes condiciones:

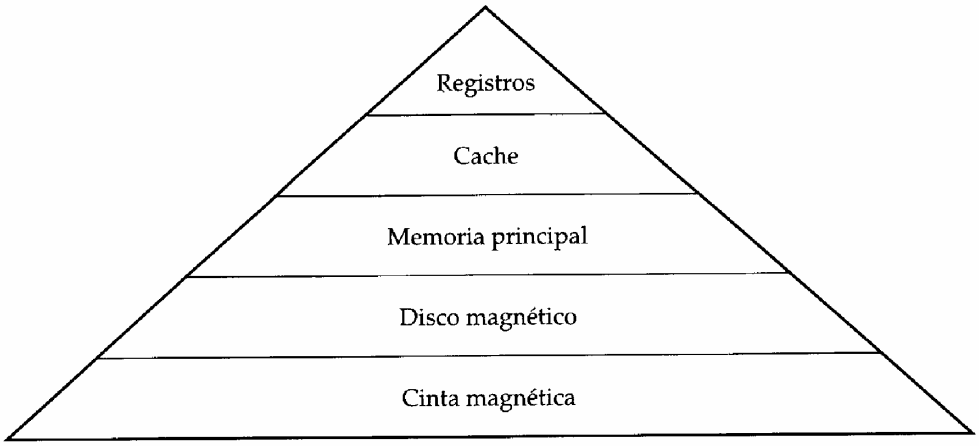
1. Disminución del coste por bit
2. Aumento de la capacidad
3. Aumento del tiempo de acceso
4. Disminución de la frecuencia de acceso a la memoria por parte del procesador

Así pues, las memorias más pequeñas, caras y rápidas son reemplazadas por memorias de más capacidad, más baratas y lentas. La clave del éxito de esta organización es el último punto: disminuir la frecuencia de acceso. Este concepto se examinará en mayor detalle cuando se discuta la *caché,* dentro de este capítulo y la memoria virtual, más adelante en el texto. Ahora se ofrecerá una breve explicación.

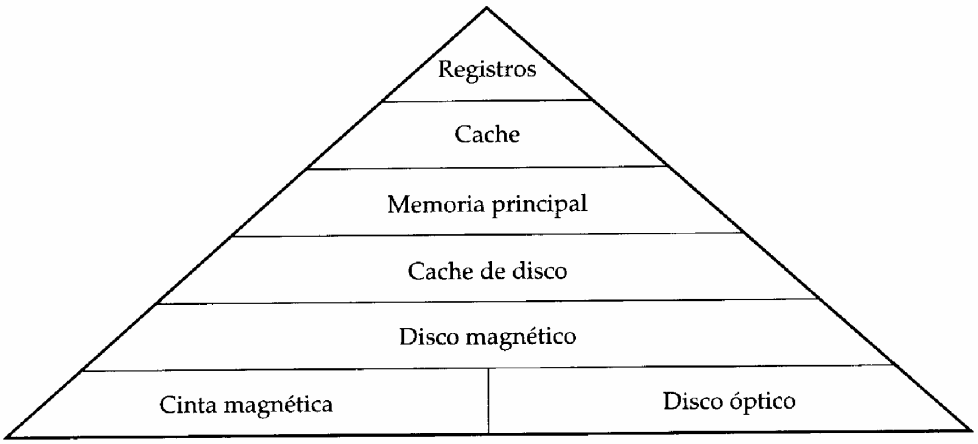
Supóngase que el procesador tiene acceso a dos niveles de memoria. El nivel 1 contiene 1000 palabras y tiene un tiempo de acceso de 0,1μs*;* el nivel 2 contiene 100.000 palabras y tiene un tiempo de acceso de l μs. Supóngase que, si la palabra a acceder está en el nivel 1, entonces el procesador accede a ella directamente. Si está en el nivel 2, entonces la palabra se transfiere primero al nivel 1 y después accede el procesador. Para simplificar, se ignorará el tiempo exigido por el procesador para saber si la palabra está en el nivel 1 o en el nivel 2. La figura 1.15 muestra la forma general de la curva que expresa esta situación. Como se puede observar, para altos porcentajes de accesos al nivel 1, el tiempo medio de acceso total es mucho más parecido al del nivel 1 que al del nivel 2.

Así pues, la estrategia funciona en principio, pero solamente si se cumplen las condiciones de la 1 a la 4. La figura 1.16 muestra las características típicas de otros sistemas de memoria. Esta figura demuestra que, empleando varias tecnologías, existe una gama de sistemas de memoria que satisfacen las condiciones de la 1 a la 3. Por fortuna, la condición 4 también se cumple en general.

La base del cumplimiento de la condición 4 es un principio conocido como *cercanía de referencias* [DENN68]. Durante el curso de ejecución de un programa, las referencias a memoria por parte del procesador, tanto para instrucciones como para datos, tienden a estar agrupadas. Los programas contienen normalmente un cierto número de bucles y de subruti-



(a) Jerarquía tradicional de memoria



(b) Jerarquía moderna de memoria

FIGURA 1.14 Jerarquía de memoria

nas iterativas. Una vez que se entra en un bucle o en una subrutina, se producirán referencias repetidas a un pequeño conjunto de instrucciones. De forma similar, las operaciones sobre tablas y vectores suponen el acceso a un conjunto de palabras de datos que están agrupadas. Durante un largo período, las agrupaciones en uso cambian, pero, en un período corto, el procesador trabaja principalmente con grupos fijos de referencias a memoria.

Por consiguiente, es posible organizar los datos en la jerarquía de modo que el porcentaje de accesos a los niveles inmediatamente inferiores sea considerablemente menor que el del nivel superior. Considérese el ejemplo de dos niveles ya presentado. Sea el nivel 2 de memoria el que contiene todos los datos e instrucciones del programa. Las agrupaciones en curso se pueden situar temporalmente en el nivel 1. De cuando en cuando, una de las agrupaciones del nivel 1 tendrá que ser descargada de nuevo al nivel 2, para hacer sitio a alguna nueva agrupación que entre en el nivel 1. En promedio, sin embargo, la mayoría de las referencias serán a instrucciones y datos contenidos en el nivel 1.

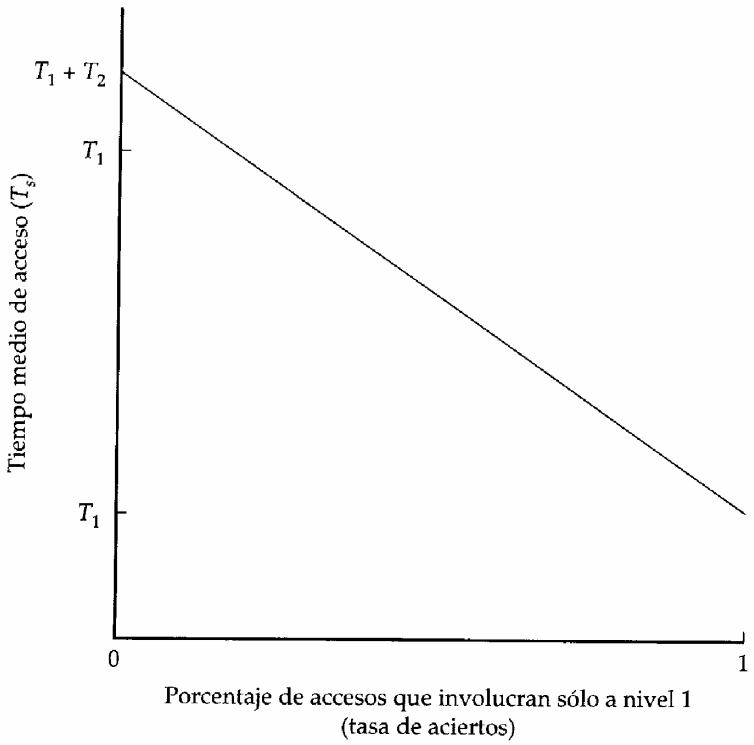


FIGURA 1.15 Rendimiento de una memoria simple de dos niveles

Este principio puede aplicarse con más de dos niveles de memoria. Considérese la jerarquía que se muestra en la figura 1. 14a. El tipo más rápido y caro de memoria consta de registros internos del procesador. Normalmente, un procesador tendrá unas pocas decenas de registros, aunque algunas máquinas poseen cientos de estos registros. Bajando dos niveles se tiene la memoria principal, también conocida como memoria real, que es la memoria interna principal del computador. Cada ubicación de la memoria principal tiene una única dirección y la mayoría de las instrucciones de máquina se refieren a una o más direcciones de la memoria principal. La memoria principal se suele ampliar con una pequeña memoria caché de alta velocidad. La caché no es generalmente visible para el programador o incluso el procesador. Es un dispositivo para encauzar el movimiento de los datos entre la memoria principal y los registros del procesador y así mejorar el rendimiento.

Las tres formas de memoria descritas son, generalmente, volátiles y emplean tecnologías de semiconductores. El uso de los tres niveles aprovecha la variedad de tipos de las memorias de semiconductores, que difieren en velocidad y coste. Los datos se almacenan de manera permanente en dispositivos externos de almacenamiento masivo, de los cuales los más comunes son los discos y las cintas magnéticas. La memoria externa, no volátil, también se denomina memoria secundaria o auxiliar. Ésta es usada para almacenar programas y archivos de datos y suelen ser visibles para el programador sólo en forma de archivos y registros y no mediante bytes o palabras individuales. Los discos también se usan para ofrecer una ampliación de la memoria principal, conocida como *almacenamiento virtual* o *memoria virtual,* que será discutida en los capítulos 6 y 7.

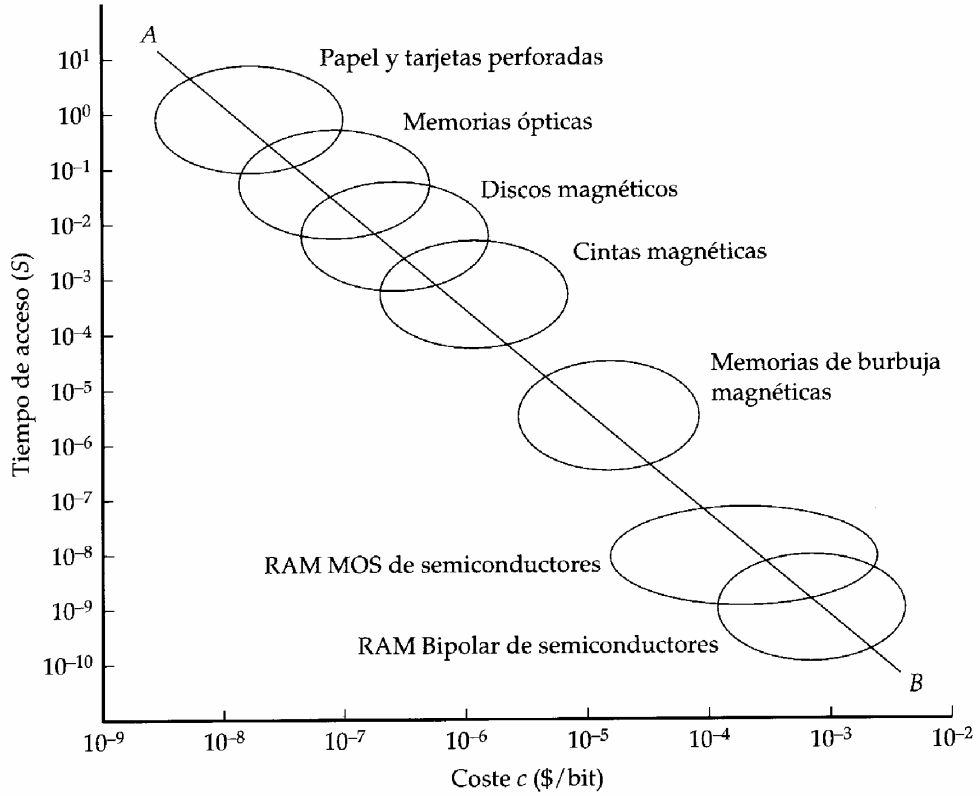


FIGURA 1.16 Abanico de costes/velocidades de las memorias [HAYE88]

Se pueden incluir otras formas de memoria en la jerarquía. Por ejemplo, los grandes computadores centrales *(mainframes)* de IBM incluyen una forma de memoria interna conocida como *memoria expandida,* que utiliza una tecnología de semiconductores más lenta y menos cara que la de la memoria principal. Estrictamente hablando, esta tecnología no encaja en la jerarquía sino que constituye una rama paralela: Los datos se pueden mover entre la memoria principal y la memoria expandida, pero no entre la memoria expandida y la memoria externa. Otras formas de memoria secundaria son los discos ópticos y la memoria de burbuja. Por último, se pueden añadir, por software, niveles adicionales a la jerarquía. Una parte de la memoria principal puede ser utilizada como un buffer para guardar temporalmente los datos transferidos con el disco. Dicha técnica, más conocida como *caché de disco* (y examinada en detalle en el capítulo 10), mejora el rendimiento de dos formas:

• Las escrituras a disco se agrupan. En lugar de muchas transferencias pequeñas de datos, se tienen unas pocas transferencias grandes de datos. Esto mejora el rendimiento del disco y reduce la utilización del procesador.

• Algunos datos destinados a la salida pueden ser referenciados por un programa antes del próximo volcado a disco. En tal caso, los datos son recuperados rápidamente desde la caché de software en lugar de hacerse lentamente desde el disco.

La figura 1.14b muestra una jerarquía moderna de memoria que incluye una caché de disco y un disco óptico como otros tipos de memoria secundaria.

El Apéndice 1A analiza las implicaciones que tienen en el rendimiento las estructuras de memoria multinivel.

1.6 MEMORIA CACHE[2]

Aunque la memoria caché es invisible para el sistema operativo, interactúa con otras partes del hardware de gestión de memoria. Es más, muchos de los principios utilizados en la me- moria virtual son también aplicables a la memoria caché.

Motivación

En todos los ciclos de instrucción, el procesador accede a la memoria, al menos una vez, para leer la instrucción y, a menudo, algunas veces más para leer los operandos y/o almacenar los resultados. La frecuencia con que el procesador puede ejecutar las instrucciones está claramente limitada por el tiempo de ciclo de memoria. Esta limitación ha sido de hecho un problema significativo, debido al persistente desacuerdo entre las velocidades del procesador y de la memoria principal. La figura 1.17, que puede considerarse representativa, ilustra esta situación.

La figura muestra que la velocidad de la memoria no se corresponde con la velocidad del procesador. Se debe adoptar un compromiso entre la velocidad, el coste y el tamaño de la memoria. En el mejor de los casos, la memoria principal se construiría con la misma tecno-

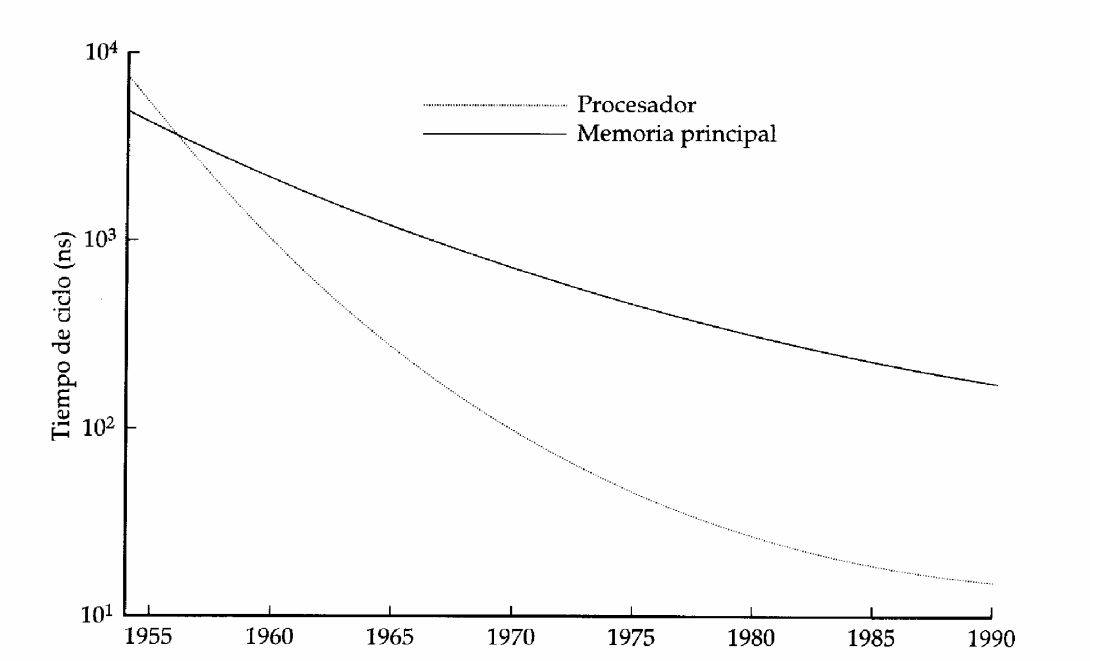


FIGURA 1.17 Rendimiento del procesador de un computador central de IBM con la memoria principal

2 El término inglés *cache* suele traducirse en la jerga técnica en español por el término *caché* (N. del T.)

logía que los registros del procesador, obteniéndose unos tiempos de ciclo de memoria comparables con los de los ciclos del procesador. Esta estrategia siempre ha resultado demasiado costosa. La solución es aprovechar el principio de cercanía, disponiendo una memoria pequeña y rápida entre el procesador y la memoria principal, es decir, la caché.

Principios de la cache

La memoria caché intenta obtener una velocidad cercana a la de las memorias más rápidas y, al mismo tiempo, aportar una memoria grande al precio de las memorias de semiconductores, que son menos costosas. Este concepto se ilustra en la figura 1.18. Hay una memoria principal más lenta y relativamente grande, junto a una memoria caché más pequeña y rápida. La caché contiene una copia de parte de la memoria principal. Cuando el procesador intenta leer una palabra de la memoria, se comprueba si la palabra está en la memoria caché. Si es así, la palabra se envía al procesador. Si no, se rellena la caché con un bloque de memoria principal, formado por un número fijo de palabras y, después, la palabra es enviada al procesador. Debido al fenómeno de la cercanía de referencias, cuando se carga en la caché un bloque de datos para satisfacer una sola referencia a memoria, es probable que ya se hayan hecho antes otras referencias a palabras del mismo bloque.

La figura 1.19 representa la estructura de un sistema de caché y memoria principal. La memoria principal consta de hasta 2^n palabras direccionables, teniendo cada palabra una única dirección de n bits. Se considera que esta memoria consta de un número de bloques de longitud fija de K palabras cada uno. Es decir, hay $M = 2^n/K$ bloques. La caché consta de C secciones de K palabras cada una y el número de secciones es considerablemente menor que el número de bloques de memoria principal *(C«M)*. En todo momento, algún subconjunto de los bloques de memoria reside en las secciones de la caché. Si se va a leer una palabra de un bloque de memoria que no está en caché, entonces el bloque se transfiere para alguna de las secciones de la caché. Debido a que hay más bloques que secciones, una sección individual no puede ser dedicada única y permanentemente a un bloque en particular. De este modo, cada sección incluye un indicador que identifica cuál es el bloque particular que está siendo actualmente almacenado en ella. Este indicador es usualmente un cierto número de los bits más significativos de la dirección.

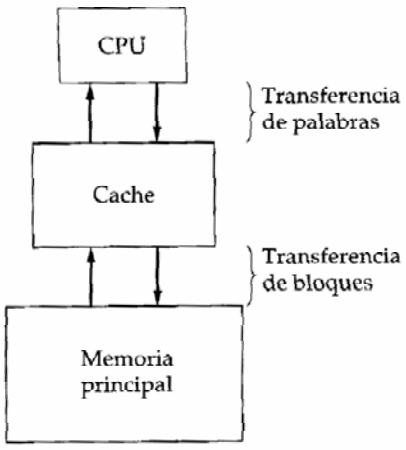


FIGURA 1.18 Cache y memoria principal

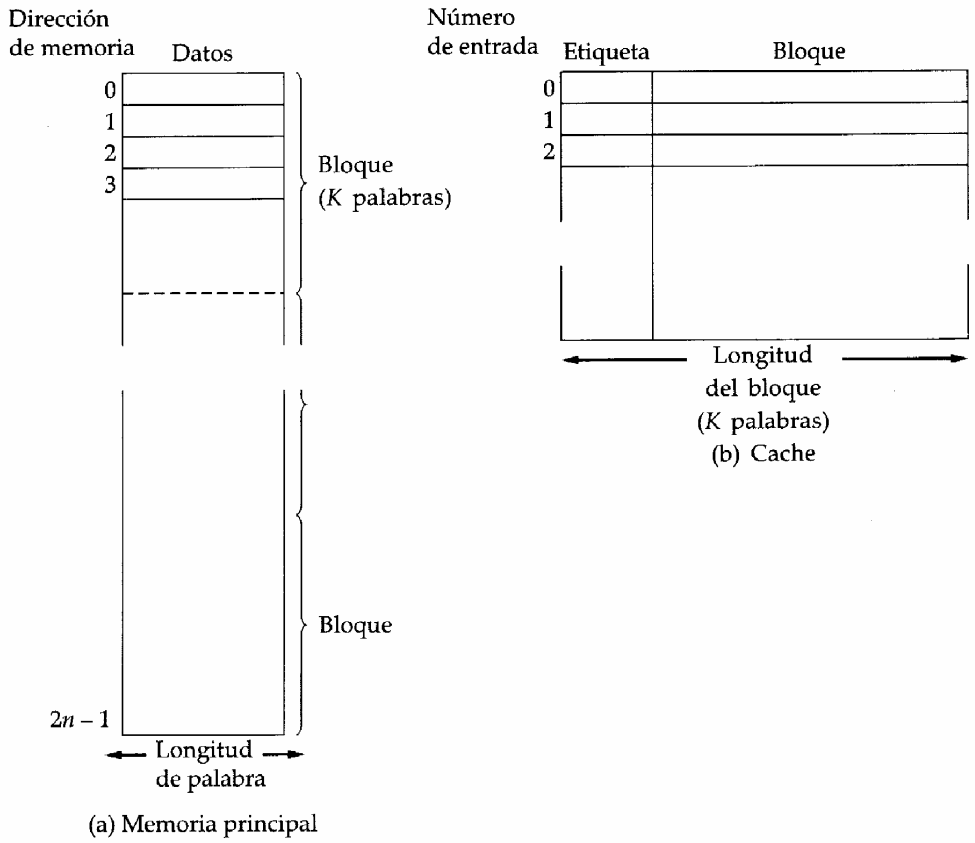


FIGURA 1.19 Estructura de cache/memoria principal

La figura 1.20 ilustra la operación de LEER. El procesador genera la dirección *DL,* de la palabra a leer. Si la palabra está en la cache es entregada al procesador. En caso contrario, el bloque que contiene a la palabra se carga en cache y la palabra se envía al procesador.

Diseño de la cache

Una discusión detallada sobre la estructura de la cache se escapa del alcance de este libro. Aquí se resumirán de forma breve los elementos clave. Se comprobará que hay que abordar cuestiones similares cuando se haga frente al diseño de la memoria virtual y de la cache del disco. Estos aspectos se encuadran en las siguientes categorías:

- Tamaño de cache.
- Tamaño del bloque.
- Función de correspondencia (*mapping*)
- Algoritmo de reemplazo.
- Política de escritura.

Ya se ha hablado sobre el **tamaño de cache**. De aquí se desprende que caches razonablemente pequeñas pueden tener un impacto significativo sobre el rendimiento. Otra cues-

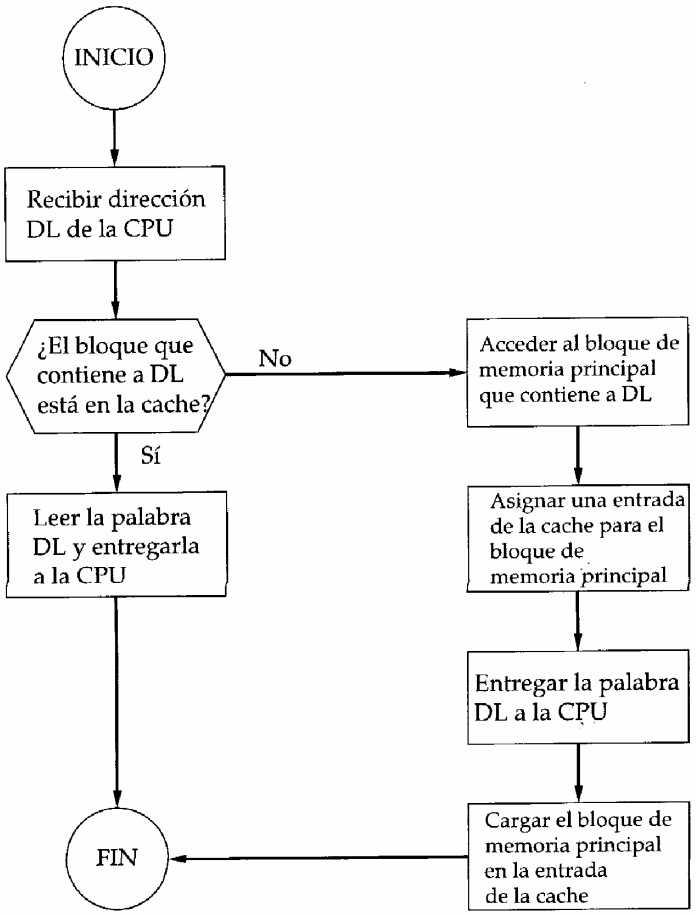


FIGURA 1.20 Operación de lectura de la cache

tión de tamaño es el **tamaño del bloque,** que es la unidad de intercambio de datos entre la cache y la memoria principal. A medida que el tamaño del bloque aumenta, desde los blo- ques más pequeños a los más grandes, la tasa de aciertos (proporción de veces que una re- ferencia se encuentre en la cache) aumentará al comienzo, debido al principio de cercanía: la alta probabilidad de que se haga referencia en un futuro próximo a datos cercanos a la palabra referenciada. A medida en que el tamaño del bloque crece, pasan a la cache más da- tos útiles. Sin embargo, la tasa de aciertos comenzará a disminuir, dado que el bloque se hace aún mayor y la probabilidad de uso del dato leído más recientemente se hace menor que la probabilidad de reutilizar el dato que hay que sacar de la cache para hacer sitio al bloque nuevo.

Cuando se trae a cache un nuevo bloque de datos, la **función de correspondencia** *(map-ping)* determina la posición de la cache que ocupará el bloque. Dos limitaciones influyen en el diseño de la función de traducción. En primer lugar, cuando un bloque se trae a la cache, puede que otro tenga que ser reemplazado. Convendría hacer esto de forma que se redujera la probabilidad de reemplazar un bloque que se vaya a necesitar en un futuro próximo.

Mientras más flexible sea la función de traducción, mayor será la envergadura del diseño del algoritmo de reemplazo que maximice la tasa de aciertos. En segundo lugar, cuanto más flexible sea la función de traducción, más compleja será la circuitería necesaria para determinar si un bloque dado está en la cache.

El **algoritmo de reemplazo** escoge, bajo las restricciones de la función de traducción, el bloque que hay que reemplazar: Sena conveniente reemplazar aquel bloque que tenga me- nos probabilidad de necesitarse en un futuro cercano. Aunque es imposible identificar un bloque tal, una estrategia bastante efectiva es reemplazar el bloque que lleva más tiempo en la cache sin que se hayan hecho referencias a él. Esta política se denomina algoritmo del *usado hace más tiempo* (LRU, *Least Recently Used)*. Se necesitan mecanismos de hardware para identificar el bloque usado hace más tiempo.

Si se modifica el contenido de un bloque de la cache, entonces hace falta escribirlo de nuevo a la memoria principal, antes de reemplazarlo. La **política de escritura** dicta cuándo tiene lugar la operación de escribir en memoria. Por un lado, la escritura puede producirse cada vez que el bloque se actualice. Por otro lado, la escritura se produce sólo cuando se reemplace el bloque. Esta última política reduce las operaciones de escritura en memoria pero deja la memoria principal en un estado obsoleto. Esto puede dificultar la operación de los multiprocesadores y el acceso directo a memoria por parte de los módulos de E/S.

1.7 TÉCNICAS DE COMUNICACIÓN DE E/S

Para las operaciones de E/S son posibles las tres técnicas siguientes:

- E/S programada
- E/S dirigida por interrupciones
- Acceso Directo a Memoria (DMA: Direct Memory Access)

E/S programada

Cuando el procesador está ejecutando un programa y encuentra una instrucción de E/S, ejecuta dicha instrucción, enviando una orden al módulo apropiado de E/S. Con E/S programada, el módulo de E/S llevará a cabo la acción requerida y luego activará los bits apropiados en el registro de estado de E/S. El módulo de E/S no lleva a cabo ninguna otra acción para avisar al procesador. En particular, no interrumpe al procesador. Así pues, es responsabilidad del procesador comprobar periódicamente el estado del módulo de E/S hasta saber que se ha completado la operación.

Con esta técnica, el procesador es el responsable de extraer los datos de la memoria principal cuando va a hacer una salida o poner los datos en la memoria principal cuando se hace una entrada. El software de E/S se escribe de manera tal que el procesador ejecute unas instrucciones que le otorguen el control directo sobre la operación de E/S, incluyendo la comprobación del estado de los dispositivos, el envío de órdenes de lectura o escritura y la transferencia de los datos. Por lo tanto, en el conjunto de instrucciones se incluyen instrucciones de E/S de las siguientes categorías:

- *Control:* Empleadas para activar un dispositivo externo y decirle qué debe hacer. Por ejemplo, una unidad de cinta magnética puede ser instruida para rebobinar o avanzar un registro.
- *Comprobación:* Empleadas para comprobar varias condiciones de estado asociadas con un módulo de E/S y sus periféricos.
- *Lectura, escritura:* Empleadas para transferir los datos entre los registros del procesador y los dispositivos externos.

La figura 1.21a muestra un ejemplo de utilización de la E/S programada para leer en un bloque de datos desde un dispositivo externo (por ejemplo, un registro de una cinta) hacia la memoria. Los datos se leen por palabra (por ejemplo, 16 bits) cada vez. Para cada palabra que se lea, el procesador debe permanecer en un bucle de comprobación del estado hasta que determine que la palabra está disponible en el registro de datos del módulo de E/S. El diagrama que se muestra en la figura 1.21 destaca la desventaja principal de esta técnica: Es un proceso que consume tiempo y que mantiene al procesador ocupado de forma innecesaria.

E/S dirigida por interrupciones

El problema de la E/S programada es que el procesador tiene que esperar un largo rato a que el módulo de E/S en cuestión esté listo para recibir o transmitir más datos. El procesador, mientras está esperando, debe interrogar repetidamente el estado del módulo de E/S. Como resultado, el nivel de rendimiento del sistema en conjunto se degrada fuertemente.

Una alternativa es que el procesador envíe una orden de E/S al módulo y se dedique a hacer alguna otra tarea útil. El módulo de E/S interrumpirá entonces al procesador para requerir sus servicios cuando esté listo para intercambiar los datos. El procesador ejecuta entonces la transferencia de los datos y reanuda el procesamiento anterior.

Seguidamente veremos cómo funciona esto, en primer lugar desde el punto de vista del módulo de E/S. Para la entrada, el módulo de E/S recibe una orden LEER desde el procesador. El módulo de E/S procede entonces con la lectura de los datos desde el periférico asociado. Una vez que los datos están en el registro de datos del módulo, éste envía una señal de interrupción al procesador a través de una línea de control. El módulo espera entonces a que los datos sean solicitados por el procesador. Cuando se haga esta solicitud, el módulo pondrá los datos en el bus de datos y estará listo para otra operación de E/S.

Desde el punto de vista del procesador, la acción para la entrada es como sigue. El procesador envía una orden LEER. Salva entonces el contexto (el contador de programa y los registros del procesador, por ejemplo) del programa en curso, se sale del mismo y se dedica a hacer otra cosa (por ejemplo, el procesador puede estar trabajando con diferentes programas al mismo tiempo). Al finalizar cada ciclo de instrucción, el procesador comprueba si hubo alguna interrupción (figura 1.7). Cuando se produce una interrupción desde el módulo de E/S, el procesador salva el contexto del programa que está ejecutando en ese momento y comienza la ejecución de la rutina de tratamiento de la interrupción. En este caso, el procesador lee la palabra de datos del módulo de E/S y la almacena en la memoria. Luego restaura el contexto del programa que emitió la orden de E/S (o de algún otro programa) y reanuda la ejecución.

La figura 1.2 Ib muestra el uso de E/S por interrupciones para leer en un bloque de datos, Compárese con la figura 1.2ª, La E/S por interrupciones es más eficiente que la E/S programada porque elimina las esperas innecesarias. Sin embargo, la E/S por interrupciones sigue

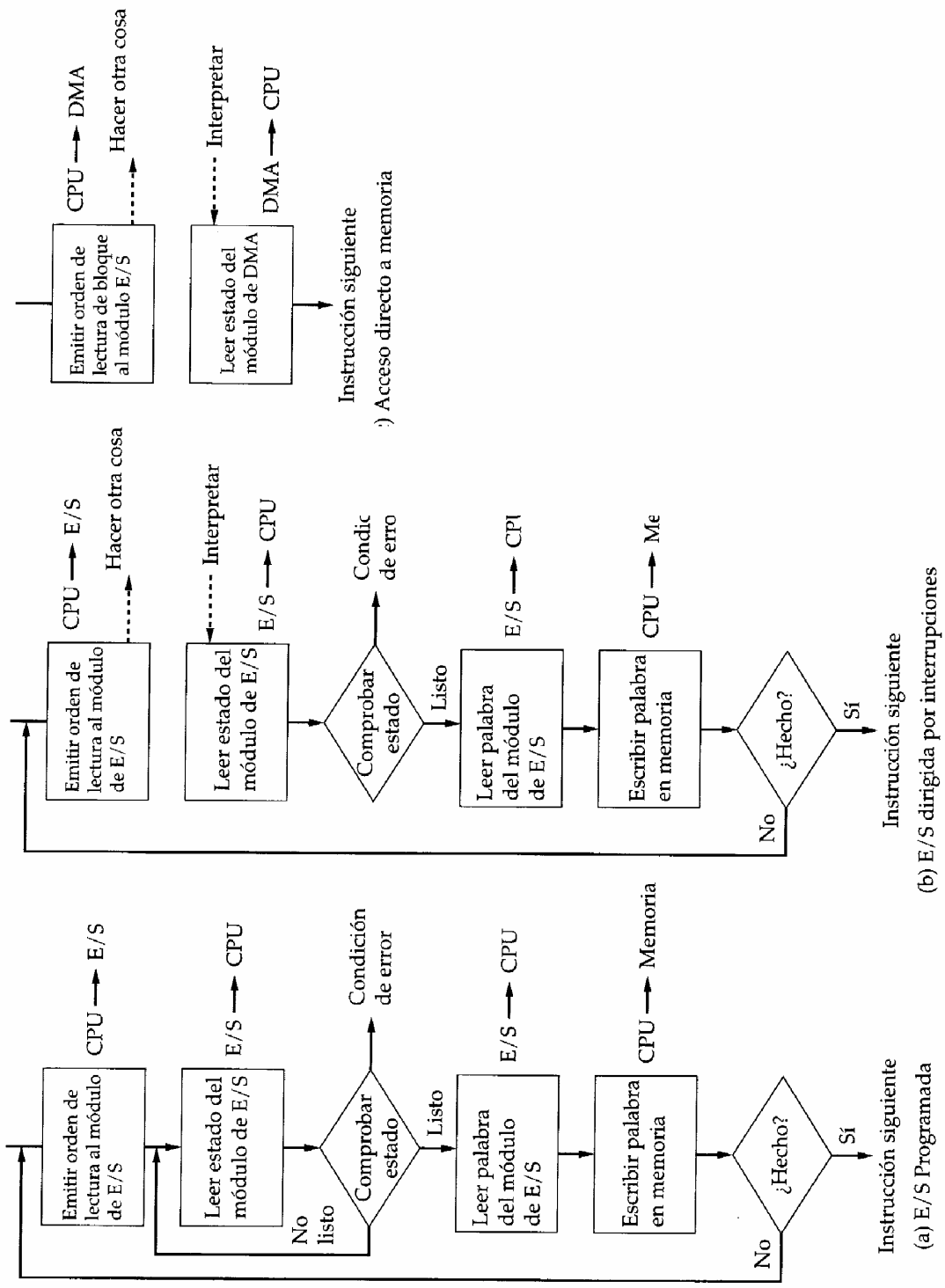


FIGURA 1.21 Tres técnicas para la entrada de un bloque de datos

consumiendo una gran cantidad de tiempo del procesador, debido a que cada palabra de datos que va de la memoria al módulo de E/S o del módulo de E/S a la memoria debe pasar a través del procesador.

Casi siempre habrá varios módulos de E/S en un sistema informático, así que hacen falta mecanismos que capaciten al procesador para determinar qué dispositivo causó la interrupción y decidir, en caso de varias líneas de interrupción, cuál debe tratar primero. En algunos sistemas, hay varias líneas de interrupción, de forma que cada módulo de E/S envía una señal por una línea diferente. Cada línea tiene una prioridad diferente. Otra solución sería habilitar una única línea de interrupción, pero utilizando líneas adicionales para indicar la dirección del dispositivo. De nuevo, a diferentes dispositivos se les asignarán prioridades diferentes.

Acceso directo a memoria

La E/S dirigida por interrupciones, aunque es más eficiente que la simple E/S programada, todavía requiere de la intervención activa del procesador para transferir los datos entre la memoria y un módulo de E/S y, además, cualquier transferencia de datos debe recorrer un camino que pasa por el procesador. Así pues, ambas formas de E/S adolecen de dos desventajas inherentes:

1. La velocidad de transferencia de E/S está limitada por la velocidad con la que el procesador puede comprobar y dar servicio a un dispositivo.
2. El procesador participa en la gestión de la transferencia de E/S; debe ejecutarse una serie de instrucciones en cada transferencia de E/S.

Cuando se tienen que mover grandes volúmenes de datos, se necesita una técnica más eficiente: el acceso directo a memoria (DMA, *Direct Memory Access)*. La función de DMA se puede llevar a cabo por medio de un módulo separado sobre el bus del sistema o puede estar incorporada dentro de un módulo de E/S. En cualquier caso, la técnica funciona como sigue. Cuando el procesador desea leer o escribir un bloque de datos, emite una orden hacia el módulo de DMA, enviándole la información siguiente:

- Si lo que se solicita es una lectura o una escritura
- La dirección del dispositivo de E/S involucrado
- La dirección inicial de memoria desde la que se va a leer o a la que se va a escribir
- El número de palabras a leer o escribir

El procesador continúa entonces con otro trabajo. Habrá delegado la operación de E/S en el módulo de DMA y dicho módulo es el que tendrá que encargarse de ésta. El módulo de DMA transfiere el bloque entero, una palabra cada vez, directamente hacia o desde la memoria, sin pasar por el procesador. Cuando se completa la transferencia, el módulo de DMA envía una señal de interrupción al procesador. De esta manera, el procesador se ve involucrado sólo al inicio y al final de la transferencia (figura 1.21c).

El módulo de DMA debe tomar el control del bus para transferir los datos con la me- moria. Debido a la competencia por la utilización del bus, puede ocurrir que el procesador necesite el bus pero deba esperar. Nótese que esto no es una interrupción; el procesador no salva el contexto y se dedica a hacer otra cosa. En su lugar, el procesador hace una pausa durante un ciclo del bus. El efecto general es hacer que el procesador ejecute con más len-

titud durante una transferencia de DMA. No obstante, para una transferencia de E/S de varias palabras, el DMA es bastante más eficiente que la E/S programada o la dirigida por interrupciones.

Cuando el módulo de E/S en cuestión es un sofisticado canal de E/S, el concepto de DMA debe tenerse en cuenta con más razón. Un canal de E/S es un procesador propiamente dicho, con un conjunto especializado de instrucciones, diseñadas para la E/S. En un sistema informático con tales dispositivos el procesador no ejecuta instrucciones de E/S. Dichas instrucciones se almacenan en la memoria principal para ser ejecutadas por el propio canal de E/S. Así pues, el procesador inicia una transferencia de E/S instruyendo al canal de E/S para ejecutar un programa en memoria. El programa especifica el o los dispositivos, la zona o zonas de memoria para almacenamiento, la prioridad y la acción que llevar a cabo bajo ciertas condiciones de error. El canal de E/S sigue estas instrucciones y controla la transferencia de datos, informando de nuevo al procesador al terminar.

1.8 LECTURAS RECOMENDADAS

[STAL93a] cubre en detalle todos los temas de este capítulo. Además, hay muchos otros textos sobre organización y arquitectura de computadores. Entre los más dignos de consideración, están los siguientes. [HENN90] es un completo estudio que hace énfasis en los aspectos cuantitativos del diseño. [HAYE88] es un libro muy bien organizado y escrito con una discusión particularmente buena sobre los aspectos del control de los sistemas informáticos, incluyendo una discusión detallada de la microprogramación. [MAN088] hace hincapié en el hardware de los computadores, con una observación detallada del diseño digital y de su uso para ímplementar las características principales del procesador. [TANE90] contempla un sistema informático en niveles lógicos, desde la lógica digital, hasta el nivel del sistema operativo, pasando por la microprogramación a nivel de máquina; así se ofrece un tratamiento unificado a través de la microprogramación, nivel de máquina, y otros temas. Muchos libros de arquitectura de computadores y de sistemas operativos ofrecen un tratamiento de los principios básicos de las interrupciones; una relación clara y minuciosa es [BECK90].

BECK90 BECK, L. *System Software* Addison-Wesley, Reading, MA, 1990.

HAYE88 HAYES, J. *Computer Architecture and Organization,* 2ª ed. Nueva York; McGraw Hill, 1988

HENN90 HENNESSY, J. y PATTERSON, *D.Computer Architecture: A Quantitatíve Approach.* Morgan Kaufmann, San Mateo, CA, 1990.

MAN088 MANO, M. *Computer Engineering Hardware Design.* Prentice Hall, Englewood Cliffs, NJ, 1988.

STALL93a STALLINGS, W. *Computer Organization and Architecture, Third Edition.* Macmillan, Nueva York, 1993.

TANE90 TANENBAUM, A. *Structured Computer Organization.* Prentice Hall, Englewood Cliffs, NJ, 1990.

PROBLEMAS

1.1 La máquina hipotética de la figura 1.3 tiene también dos instrucciones de E/S:
0011 = Carga AC desde B/S
0111 = Almacena el AC en La E/S
En estos casos, las direcciones de 12 bits identifican a un dispositivo externo en particular Mostrar la ejecución del programa (utilizando el formato de la figura 1.4) para el programa siguiente:
a) Cargar AC desde e1 dispositivo 5.
b) Sumar el contenido de la ubicación de memoria 940.
e) Almacenar el AC en el dispositivo 6.

1.2 Considérese un sistema informático que contiene un módulo de E/S que controla un teletipo sencillo de teclado/impresora. La CPU tiene los siguientes registros y datos están conectados directamente al bus del sistema:
RENT:Registro de Entrada, 8 bits
RSAL:Registro de Salida, 8 bits
IE: Indicador de Entrada, 1 bit
IS: Indicador de Salida, 1 bit
HI: Habilitar Interrupción, 1 bit
La entrada tecleada desde el teletipo y la salida hacia la impresora son controladas por el módulo de E/S. El teletipo está capacitado para codificar un símbolo alfanumérico en una palabra de 8 bits y para decodificar una palabra de 8 bits en un símbolo alfanumérico. El Indicador de Entrada se activa cuando se introduce, desde el teletipo, una palabra de 8 bits en el registro de entrada. El Indicador de Salida se activa cuando se imprime una palabra.
a) Describir cómo la CPU, utilizando los cuatro primeros registros enumerados en el problema, puede llevar a cabo la E/S con el teletipo.
b) Describir cómo puede llevarse a cabo la función de una forma más eficiente, empleando el registro HI.

1.3 Un sistema de memoria principal consta de un número de módulos de memoria conectados al bus del sistema. Cuando se hace una solicitud de escritura, el bus estará ocupado, durante 100 nanosegundos (ns), por datos, direcciones y señales de control. Durante los mismos 100 ns y para los 500 ns siguientes, el módulo de memoria direccionado ejecuta un ciclo aceptando y almacenando los datos. La operación de los módulos de memoria puede solaparse, pero solo una solicitud puede estar en el bus a un mismo tiempo.
a) Supóngase que hay ocho módulos conectados al bus. ¿Cuál es la máxima velocidad posible *(en* palabras por segundo) con que se pueden almacenar los datos?
Trazar un *gráfico* con la máxima velocidad de escritura en función del tiempo de ciclo de un módulo, suponiendo ocho módulos de memoria y un tiempo de ocupación del bus de 100 ns.

1.4 Para ahorrar puertas lógicas, los buses con frecuencia se multiplexan en el tiempo. Es decir, ciertas líneas de bus son utilizadas para dos funciones diferentes en dos momentos diferentes. Por ejemplo, las mismas líneas pueden utilizarse como líneas de dirección y como líneas de datos. Considérese una máquina con palabras de 48 bits y un disco con una velocidad de transferencia de 10^7 bps (bits por segundo) y un tiempo de acceso a memoria de 600 ns. Supóngase que cada transmisión por el bus requiere *750* ns para los bits de datos y varias operaciones de control de acuse de recibo. ¿Cuántos bits de datos se deben enviar en cada periodo de *750* ns para adelantarse al disco? y ¿ayudaría o estorbaría la multiplexación en el tiempo? ¿Qué fracción del ancho de banda de la memoria principal es consumida pon una operación de E/S al disco?

1.5 Generalizar las ecuaciones 1.1 y 1.2 para jerarquías de memoria de n niveles.

1.6 Considérese un sistema de memoria con los siguientes parámetros:
Tc = 100 ns Cc = 0,01 centavos/bit
Tm = 1.200 ns Cm = 0,001 centavos/bit
$H = 0,95$
a) ¿Cuál es el coste de 1 megabyte (MB) de memoria principal?
b) ¿Cuál es el coste de 1 MB de memoria principal utilizando tecnología de memoria cache?
c) Diseñar un sistema de memoria principal/cache con 1 MB de memoria

principal cuyo tiempo efectivo de acceso no sea mayor del 10% del tiempo de acceso de la cache. ¿Cuál es el coste?

1.7 Cuando varios módulos pueden generar interrupciones, es necesario que *exista una via para que* el procesador determine qué módulo produjo la interrupción. Esta técnica se conoce como *arbitraje vectorizado del bus,* En este esquema, un módulo debe primero obtener el control del bus, utilizando algún método de arbitraje, antes de poder elevar la solicitud de línea de interrupción. De este modo, sólo un módulo puede conseguir dicha línea cada vez. Cuando el procesador detecta la interrupción, responde elevando la línea de respuesta a interrupción. EL módulo que originó la interrupción sitúa una palabra en las 11-neas de datos. Esta palabra se denomina *vector y* es la dirección del módulo de E/S o bien alguna otra identificación única. En ambos casos, el procesador usa el vector como un puntero a la rutina apropiada de tratamiento de la interrupción.
¿Por qué el módulo que origina la interrupción sitúa el vector en las líneas de datos en lugar de en las líneas de dirección?

1.8 En casi todos los sistemas que incluyen módulos de DMA, el acceso por DMA a la memoria principal tiene una prioridad más alta que la del acceso del procesador a la memoria principal. ¿Por qué?

1.9 Un módulo de DMA transfiere caracteres a la memoria principal desde un dispositivo externo que trasmite a 9600 bps. El procesador puede leer las instrucciones a razón de 1 millón de instrucciones por segundo. ¿En cuánto se hará más lento el procesador debido a la actividad del DMA?

1.10 Un computador consta de *una* Cpu y un dispositivo D de E/S conectado a la memoria principal *M,* a través de un bus compartido con un ancho del bus de datos de una palabra. La CPU puede ejecutar un máximo de 106 instrucciones por segundo. Una instrucción requiere, en promedio, cinco ciclos de máquina, tres de los cuales utilizan el bus de memoria. Una operación de LEER o ESCRIBIR utiliza un ciclo de máquina. Supóngase que la CPU está ejecutando continuamente programas en modo subordinado *(background)* que requieren el 95% de la tasa de ejecución de instrucciones, sin ninguna instrucción de E/S. Ahora, supóngase que se transfieren bloques muy grandes de datos entre *M y D.*
   a) Si se utiliza E/S programada y cada transferencia de E/S de una palabra requiere que la CPU ejecute dos instrucciones, estimar la velocidad R_{Max} máxima de transferencia posible de E/S de datos a través de D.
   b) Estimar R_{Max} si se utiliza transferencia por DMA.

1.11 Supóngase que el procesador utiliza una pila para administrar las llamadas a procedimientos y los retornos. ¿Se puede eliminar el contador de programa utilizando la cima de la pila como contador de programa?

APÉNDICE 1A

RENDIMIENTO DE LAS MEMORIAS A DOS NIVELES

En este capitulo se hace referencia a una cache que actúa como un buffer entre la memoria principal y el procesador para crear una memoria interna de dos niveles. Esta arquitectura a dos niveles proporciona un mejor rendimiento en comparación con la memoria de un nivel, aprovechando la propiedad conocida como *cercanía,* que se analizará a continuación.El mecanismo de cache de la memoria principal es parte de la arquitectura del computador; está implementado por hardware y normalmente es invisible para el sistema operativo. Pon estas razones, tal mecanismo no es un objetivo de este libro. Sin embargo, hay otros dos casos de enfoques de memoria a dos niveles que también aprovechan la cercanía y que son, al menos en parte, implementados por el sistema operativo: la memoria virtual y la cache de

disco (tabla 1.2). Estos dos elementos se exploran en los capítulos 7 y 10, respectivamente. En este apéndice, se verán algunas de las características de rendimiento de las memorias a dos niveles que son comunes a los tres enfoques.

1A.1 Cercanía

La base para las ventajas del rendimiento de una memoria a dos niveles es un principio conocido como *cercanía de referencias* [DENN68]. Este principio afirma que las referencias a memoria tienden a estar agrupadas. Después de un largo periodo, las agrupaciones en uso cambian, pero en un periodo corto, el procesador estará trabajando principalmente con grupos fijos de referencias a memoria.

Desde un punto de vista intuitivo, el principio de cercanía tiene sentido. Considérese la línea de razonamiento siguiente:

1. Excepto en instrucciones de desvío y de llamada, las cuales constituyen sólo una pequeña fracción de todas las instrucciones, la ejecución de un programa es secuencial. Así pues, la próxima instrucción a leer estará inmediatamente a continuación de la última instrucción leída.
2. Es raro tener una larga secuencia ininterrumpida de llamadas a procedimientos seguida por la correspondiente secuencia de retornos. Es más, un programa permanece confinado en una ventana más bien estrecha en profundidad de llamadas a procedimientos. Por tanto, durante un corto periodo, las referencias a instrucciones tienden a estar localizadas en unos pocos procedimientos.
3. La mayoría de las estructuras iterativas constan de un pequeño número de instrucciones que se repiten muchas veces. Durante la iteración, el cálculo está confinado a una pequeña sección contigua del programa.
4. En muchos programas, gran parte de los cálculos involucran el procesamiento de estructuras de datos, tales como vectores o secuencias de registros. En muchos casos, las referencias sucesivas a estas estructuras serán a elementos de datos que se encuentran cerca unos de otros.

Esta línea de razonamiento ha sido confirmada en muchos estudios. Por ejemplo, considérese la 1ª afirmación. Se han realizado diversos estudios para analizar el comportamiento de los programas escritos en lenguajes de alto nivel. La tabla 1.3 incluye resultados clave que miden la aparición de varios tipos de instrucciones durante la ejecución, a partir de los siguientes estudios. El primer estudio del comportamiento de un lenguaje de programación, llevado a cabo por Knuth [KNUT71], examina una colección de programas en FORTRAN, utilizados

TABLA 1.2 Características de las Memorias a Dos Niveles

	Cache de Memoria Principal	Memoria Virtual (Paginación)	Cache de Disco
Tasas típicas de tiempo de acceso	5/1	1000/1	1000/1
Sistema de Gestión de memoria	Implementado por hardware especial	Combinación de hardware y software	Software del sistema
Tamaño típico de bloque	De 4 a 128 bytes	De 64 a 4096 bytes	De 64 a 4096 bytes
Acceso del procesador al segundo nivel	Acceso directo	Acceso indirecto	Acceso indirecto

TABLA1.3 Frecuencia Dinámica Relativa de las Operaciones de los Lenguajes de Alto Nivel

Estudio	[HUCK83]	[KNUT71]	[PATT82]		[TANE87]
Lenguaje	Pascal	FORTRAN	Pascal	C	SAL
Carga de trabajo	Científico	Estudiante	Sistema	Sistema	Sistema
Asignación	74	67	45	38	42
Bucle	4	3	5	3	4
Llamada	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
Otras	—	7	6	1	6

como ejercicios de estudiantes. Tanenbaum [TANE78] publicó unas medidas recogidas a partir de cerca de 300 procedimientos utilizados en los programas del sistema operativo y escritos en un lenguaje de programación estructurada (SAL). Patterson y Sequin [PATT82] analizaron un conjunto de medidas tomadas de compiladores, programas tipográficos, diseño asistido (CAD, *Computer Aided Design*), ordenación y comparación de archivos. Se estudiaron los lenguajes de programación C y Pascal. Huck [HUCK83] analizó cuatro programas dirigidos a representar una combinación de cálculos científicos de propósito general, incluyendo la transformada rápida de Fourier y la integración de sistemas de ecuaciones diferenciales. Se está de acuerdo, en los resultados de esta combinación de lenguajes y aplicaciones, en que las instrucciones de desvío o de llamada representan sólo una fracción de las instrucciones ejecutadas durante el tiempo de vida de un programa. Así pues, estos estudios confirman la 1ª afirmación.

Con respecto a la 2ª afirmación, los estudios presentados en Patterson [PATT85] la confirman. Esto se ilustra en la figura 1.22 que muestra el comportamiento de las llamadas/retornos. Cada llamada está representada por una línea que se dirige hacia abajo y a la derecha, mientras que cada retorno está representado por una línea que se dirige hacia arriba y a la derecha. En la figura, se define una *ventana* con una profundidad igual a 5. Sólo una secuencia de llamadas y retornos con un movimiento neto de 6 en una de las dos direcciones hace que la ventana se mueva. Como se puede ver, la ejecución del programa puede permanecer en una ventana estacionaria durante períodos bastante largos. Un estudio de programas en C y en Pascal por parte del mismo grupo demostró que una ventana de profundidad 8 necesitaría desplazarse poco más del 1% de las llamadas o los retornos [TAMI83].

Otros estudios han demostrado la validez de las afirmaciones 3 y 4 (por ejemplo, [DENN80b] y [CHU76]).

1A.2 *Funcionamiento de la memoria a dos niveles*

La propiedad de cercanía puede ser aprovechada para la construcción de una memoria a dos niveles. La memoria de nivel superior (M1) es más pequeña, más rápida y más cara (por bit) que la memoria de nivel inferior (M2). M1 se utiliza como un almacenamiento temporal para parte del contenido de M2. Cuando se hace una referencia a memoria, se intenta a acceder a un elemento de M1. Si tiene éxito, entonces el acceso será rápido. Si no, entonces se copia un bloque de posiciones de memoria de M2 a M1 y el acceso finalmente tiene lugar a través de M1. Por causa de la cercanía, una vez que un bloque se pasa a M1, deberá haber un cierto número de accesos a las posiciones de dicho bloque, resultando un servicio global más rápido,

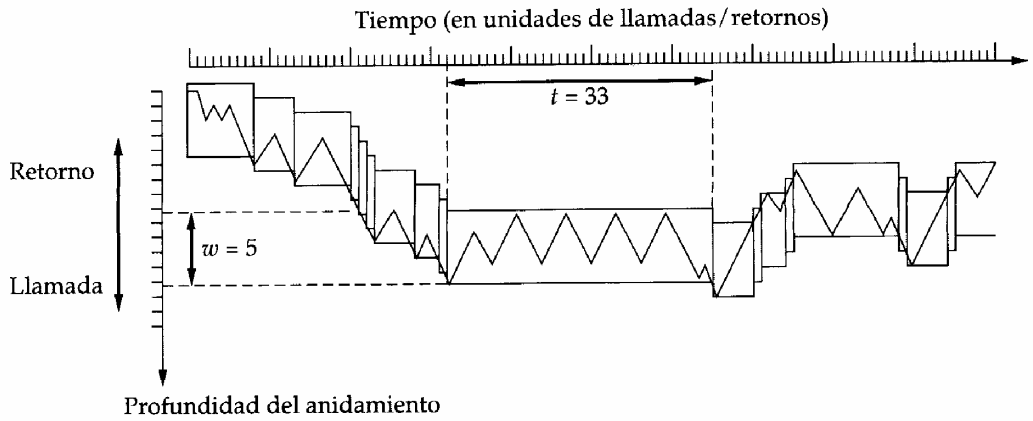


FIGURA 1.22 Comportamiento de llamadas/retornos en los programas

Para expresar el tiempo medio de acceso a un elemento, se deben considerar no sólo las velocidades de los dos niveles de la memoria, sino también la probabilidad de que una referencia dada se encuentre en MI. Esta probabilidad es conocida como la *tasa de aciertos.* Así se tiene:

$$T_s = H \times T_1 + (1 - H) \times (T_1 + T_2)$$
$$= T_1 + (1 - H)\, T_2 \qquad (1.1)$$

donde

T_S= tiempo medio de acceso (del sistema)

T_1= tiempo de acceso de MI (cache, cache de disco)

T_2= tiempo de acceso de M2 (memoria principal, disco)

H = tasa de aciertos (proporción de veces que la referencia se encuentra en MI)

La figura 1.15 muestra el tiempo medio de acceso en función de la tasa de aciertos. Como se puede ver, para un alto porcentaje de aciertos, el tiempo medio de acceso total está mucho más cerca del de MI que del de M2.

1A.3 Rendimiento

Se considerarán algunos de los parámetros relevantes para una valoración del mecanismo de memoria a dos niveles. Primero se considera el coste. Así se tiene:

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \qquad (1.2)$$

donde

C_S= coste medio por bit de los dos niveles combinados de memoria

C_1= coste medio por bit de MI

C_2= coste medio por bit de M2

S_1= tamaño de MI

S_2= tamaño de M2

Sería conveniente que $C_S \approx C_2$ Puesto que $C_1 \gg C_2$, esto exige que $S_1 \ll S_2$. La figura 1.23 muestra esta relación.

A continuación, se considera el tiempo de acceso. Para que una memoria a dos niveles aporte una mejora significativa en el rendimiento, hace falta que T_S, sea aproximadamente igual a T_1 ($T_S \approx T_1$). Dado que T_1 es mucho menor que T_2 ($T_1 \ll T_2$), se necesita una tasa de aciertos cercana a 1. Así pues, se desea que M1 sea suficientemente pequeña como para mantener el coste bajo y suficientemente grande como para mejorar la tasa de aciertos y, por tanto, el rendimiento. ¿Hay algún tamaño de M1 que satisfaga ambos requisitos en una medida razonable? Se puede contestar a esta pregunta con otra serie de preguntas:

- ¿Qué valor de la tasa de aciertos se necesita para satisfacer el rendimiento exigido?
- ¿Qué valor de M1 asegurará la tasa de aciertos que se necesita?
- ¿Satisface este tamaño las exigencias de coste?

Para responderlas, considérese la cantidad T_1/T_S, conocida como *eficiencia de accesos*. Esta es una medida de cuan cercano al tiempo de acceso a M1 (T_1) es el tiempo medio de acceso (T_S). De la Ecuación (1.1), se tiene:

$$\frac{T_1}{T_S} = \frac{1}{H + (1 - H)\frac{T_2}{T_1}} \tag{1.3}$$

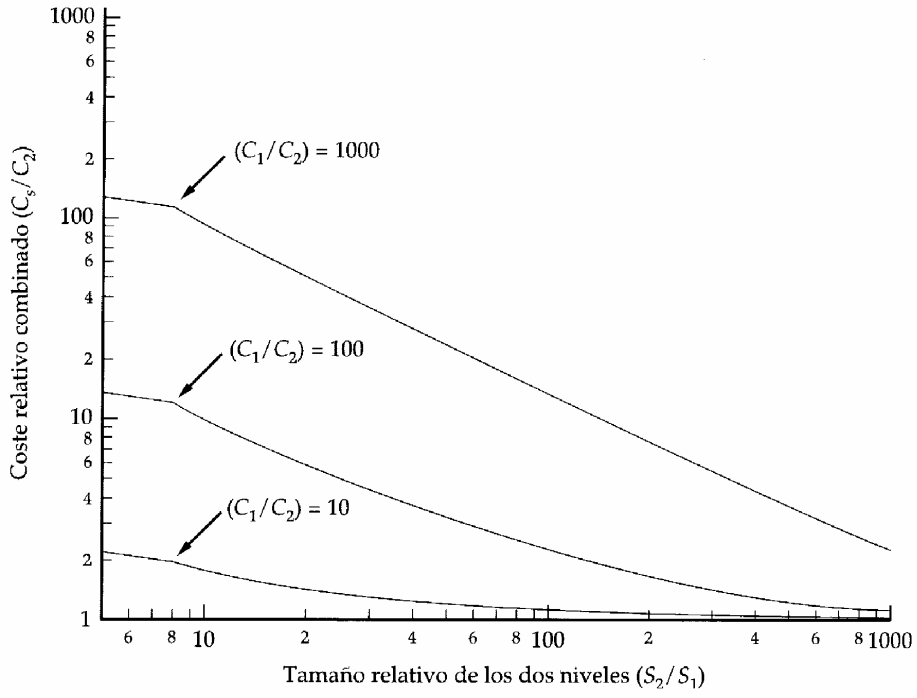


FIGURA 1.23 Relación entre el coste medio de la memoria y el tamaño relativo de la memoria a dos niveles

En la figura 1.24 se ha dibujado T_1/T_S en función de la tasa de aciertos *H,* con la cantidad T_2 / T_1 como parámetro. Normalmente, el tiempo de acceso a la cache es de cinco a diez veces más rápido que el tiempo de acceso a la memoria principal (es decir, T_2/T_1 es de 5 a 10) y el tiempo de acceso a la memoria principal es alrededor de 1000 veces más rápido que el tiempo de acceso al disco *(*T_2/T_1 = 1000). Por tanto, una tasa de aciertos en el rango de 0,8 a 0,9 parece ser necesaria para satisfacer los requisitos de rendimiento.

Ahora se puede plantear la pregunta sobre el tamaño relativo de la memoria con más exactitud. ¿Una tasa de aciertos de 0,8 o mejor es razonable para S_1 « *S_2?* Esto dependerá de una serie de factores, entre los que se incluyen la naturaleza del software ejecutado y los detalles del diseño de la memoria a dos niveles. El determinante principal es, desde luego, el grado de cercanía. La figura 1.25 sugiere el efecto que la cercanía tiene sobre la tasa de aciertos. Sin duda alguna, si M1 tiene el mismo tamaño que M2, entonces la tasa de aciertos será de 1,0; es decir, todos los elementos de M2 también estarán almacenados en M1. Supóngase ahora que no hay cercanía, es decir, que las referencias son completamente aleatorias. En este caso, la tasa de aciertos debe ser una función estrictamente lineal del tamaño relativo de la memoria. Por ejemplo, si M1 tiene la mitad de tamaño que M2, entonces, en cualquier momento, la mitad de los elementos de M2 estarán también en M1 y la tasa de aciertos será 0,5. En la práctica, sin embargo, hay un cierto grado de cercanía en las referencias. Los efectos de una cercanía estrecha o moderada se indican en la figura.

De este modo, si hay una fuerte cercanía, es posible alcanzar altos valores de la tasa de aciertos aún con tamaños relativamente pequeños de la memoria del nivel superior. Por ejemplo, numerosos estudios han demostrado que tamaños más bien pequeños de la cache

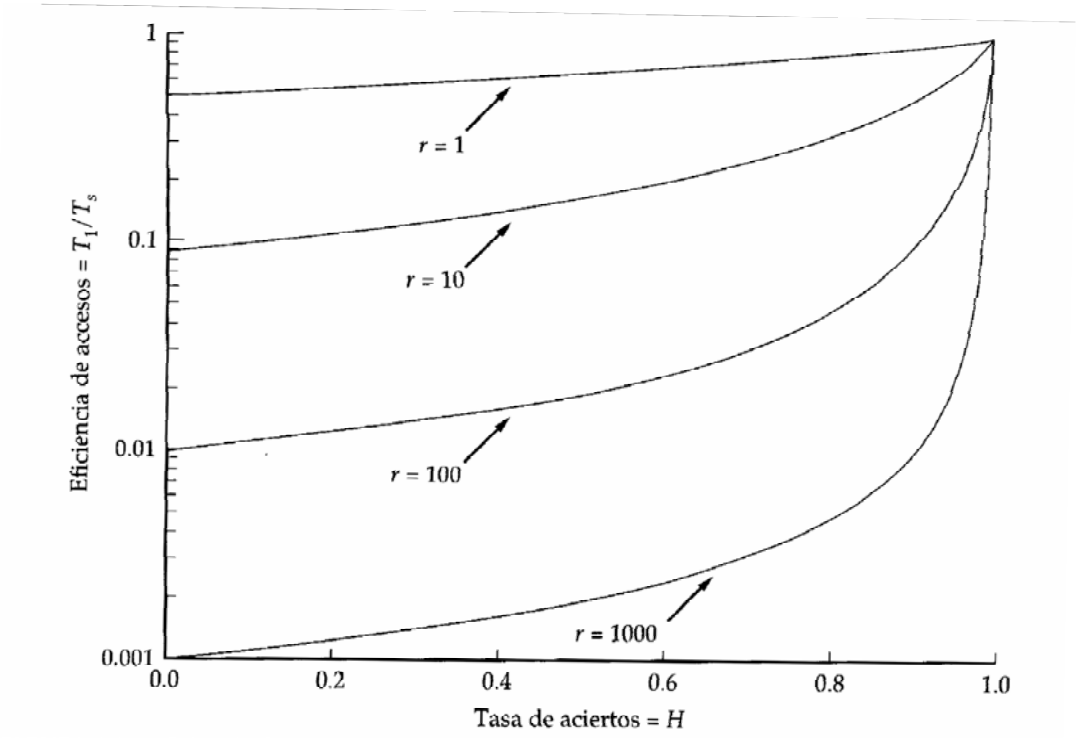


FIGURA 1.24 Eficiencia de Accesos en función de la tasa de aciertos (R = T2 / T1)

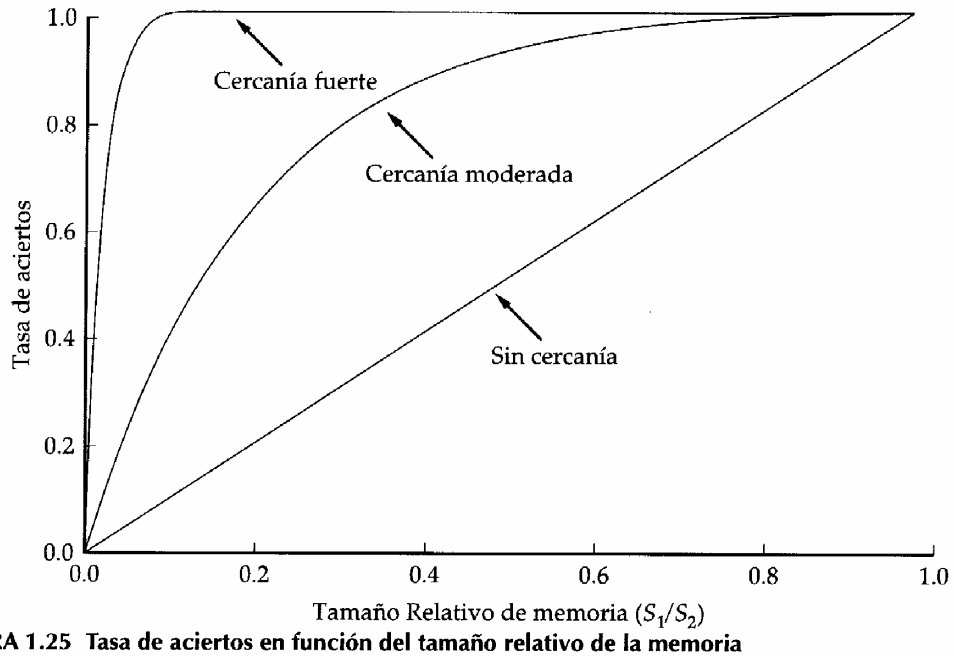


FIGURA 1.25 Tasa de aciertos en función del tamaño relativo de la memoria

dan una tasa de aciertos superior a 0,75, *independientemente del tamaño de la memoria principal* (por ejemplo, [AGAR89a], [AGAR89b], [PRZY88], [SMIT82] y [STRE83]). Una cache en un rango entre 1K y 128K palabras es, por lo general, adecuada, mientras que la memoria principal está ahora normalmente en el rango de varios megabyte». Al considerar la memoria virtual y la cache del disco, se pueden citar otros estudios que confirman el mismo fenómeno, es decir, que una MI relativamente pequeña da valores altos de la tasa de aciertos por causa de la cercanía.

Esto conduce a la última de las preguntas antes citadas: ¿El tamaño relativo de las dos memorias satisface las exigencias de coste? La respuesta es, sin duda, afirmativa. Si hace falta sólo una cantidad relativamente pequeña de memoria del nivel superior para lograr un buen rendimiento, entonces el coste medio por bit de los dos niveles de memoria se aproximará al de la memoria más barata del nivel inferior.

APÉNDICE 1B

CONTROL DE PROCEDIMIENTOS

Una técnica común para controlar las llamadas a procedimientos y los retornos es usar una pila. Este apéndice resume las propiedades básicas de las pilas y analiza su utilización en el control de los procedimientos.

1B.1 Implementación de las pilas

Una *pila* es un conjunto ordenado de elementos, de los cuales sólo uno puede ser accedido en un momento dado. El punto de acceso se llama *cima* de la pila. El número de elementos

de la pila o longitud de la pila, es variable. Se pueden añadir o eliminar elementos sólo por la cima. Por esta razón, una pila también se conoce como una lista *último en entrar, primero en salir* (FIFO, *Last-In, First-Out).*

La figura 1.26 muestra las dos operaciones básicas que se pueden llevar a cabo sobre las pilas. Se comienza en algún instante en el que la lista contiene un cierto número de elementos. Una operación METER *(PUSH)* añade un nuevo elemento a la cima de la pila. Una operación SACAR *(POP)* quita el elemento de la cima de la pila. En ambos casos, la cima de la pila se desplaza en consecuencia.

La implementación de la pila exige un cierto número de posiciones empleadas para almacenar los elementos. En la figura 1.27 se ilustra un enfoque típico de implementación. Se reserva en la memoria principal (o en la virtual) un bloque de posiciones contiguas para la pila. La mayor parte del tiempo, el bloque estará parcialmente lleno con los elementos de la pila y el resto permanecerá disponible para su crecimiento. Se necesitan tres direcciones para las operaciones, las cuales se almacenan a menudo en los registros del procesador:

• *Un puntero de pila:* Contiene la dirección de la cima de la pila. Si se añade un elemento a la pila (METER) o se elimina un elemento de la pila (SACAR), este puntero se incrementa o se decrementa para poder tener la dirección de la nueva cima de la pila.

• *Base de la pila:* Contiene la dirección del fondo de la pila en el bloque reservado para la misma. Ésta es la primera posición que se utiliza cuando se añade un elemento a una lista vacía. Si se intenta SACAR cuando la pila está vacía, se informará sobre el error.

• *Límite de la pila:* Contiene la dirección del otro extremo del bloque reservado para la pila. Si se intenta METER cuando la pila está llena, se informará sobre el error.

1B.2 Llamadas a procedimientos y retornos

Una técnica habitual para gestionar las llamadas a procedimientos y los retornos se basa en el uso de una pila. Cuando el procesador ejecuta una llamada, pone la dirección de retomo en la pila. Cuando ejecuta un retomo, utiliza la dirección que está en la cima de la pila. La figura 1.28 ilustra el uso de la pila para administrar los procedimientos anidados de la figura 1-29.

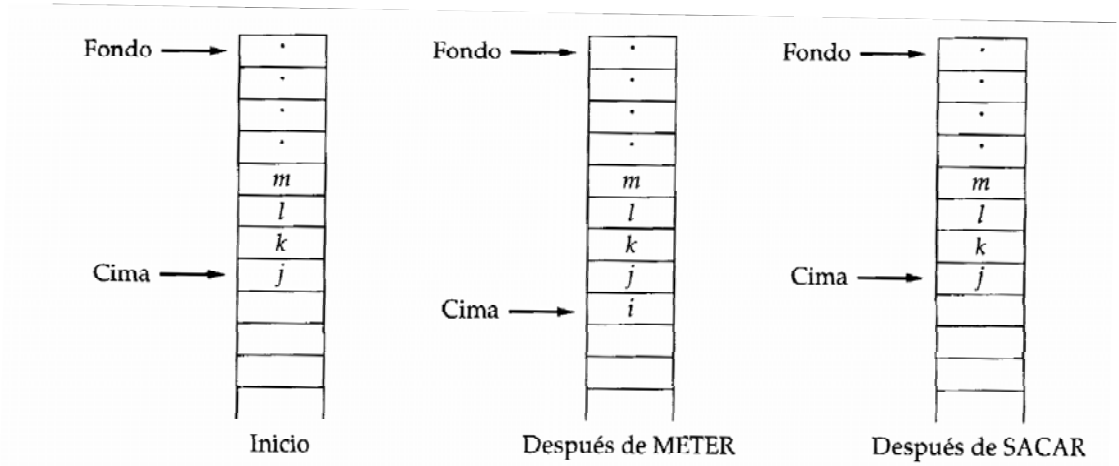


FIGURA 1.26 Operaciones básicas sobre una pila

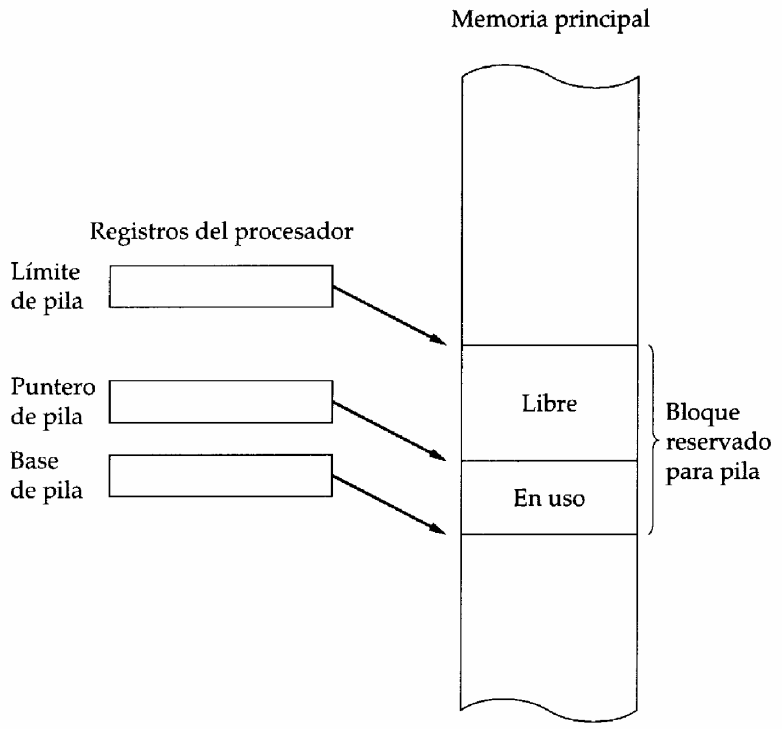


FIGURA 1.27 Organización típica de una pila

Además de dar la dirección de retomo, a menudo es necesario pasar unos parámetros en las llamadas a procedimientos. Estos podrían pasarse en registros. Otra posibilidad es almacenar los parámetros en la memoria justo antes de la instrucción LLAMAR *(CALL)*. En tal caso, el retomo debe ser a la posición que sigue a los parámetros. Ambos enfoques tienen sus inconvenientes. Si se utilizan los registros, el programa llamado deberá escribirse de modo que se asegure que los registros se usan correctamente. Almacenar los parámetros en memoria dificulta el intercambio de un número variable de parámetros.

Un método más flexible de paso de parámetros es la pila. Cuando el procesador ejecuta una llamada, no sólo apila la dirección de retomo, sino también los parámetros que tiene que pasarle al

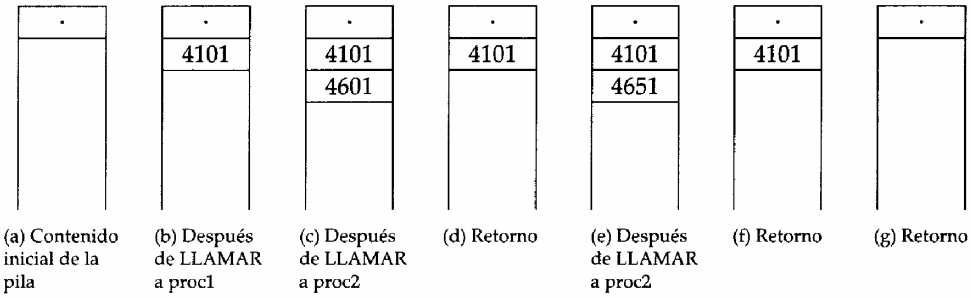


FIGURA 1.28 Empleo de una pila para implementar los procedimientos anidados de la figura 1.29

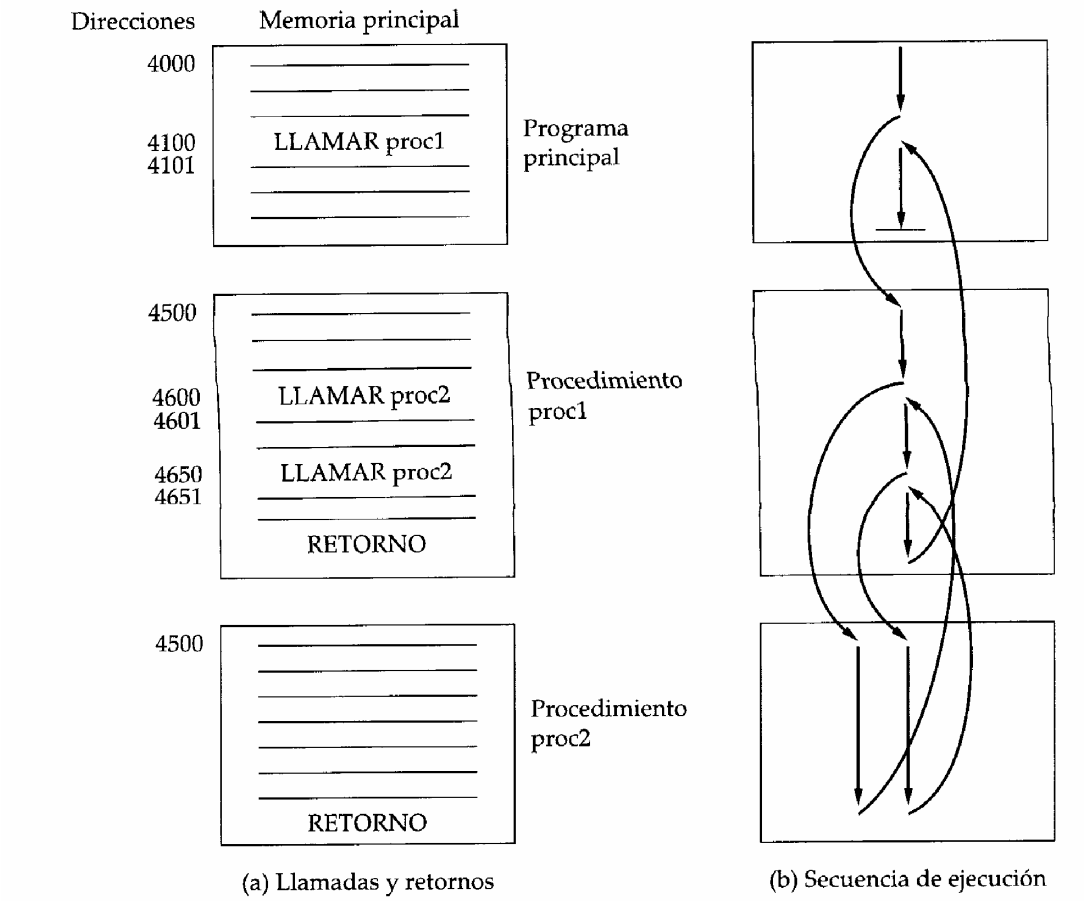


FIGURA 1.29 Procedimientos Anidados

procedimiento llamado. El procedimiento llamado puede acceder a los parámetros en la pila. Al volver, los parámetros de retomo también pueden ponerse en la pila, *bajo* la dirección de retomo. El conjunto completo de parámetros, incluyendo la dirección de retomo, que se almacena como producto de la invocación de un procedimiento, es conocido como marco de pila *(snack frame).* En la figura 1.30 se ofrece un ejemplo. Este se refiere a un procedimiento P en el que se declaran las variables locales *x*1 y x2 y un procedimiento Q, que puede ser llamado por P y en el que se declaran las variables locales y1 e y2. En la figura, el punto de retorno para cada procedimiento es el primer elemento almacenado en el marco de pila correspondiente. A continuación, se almacena un puntero al comienzo del marco anterior. Esto es necesario si el número o la longitud de los parámetros a almacenar son variables.

1B.3 Procedimientos reentrantes

Un concepto útil, particularmente en un sistema que da soporte a varios usuarios al mismo tiempo, es el de procedimiento reentrante. Un *procedimiento reentrante* es aquel en el que sólo una copia del código del programa puede estar compartida entre varios usuarios durante el mismo periodo.

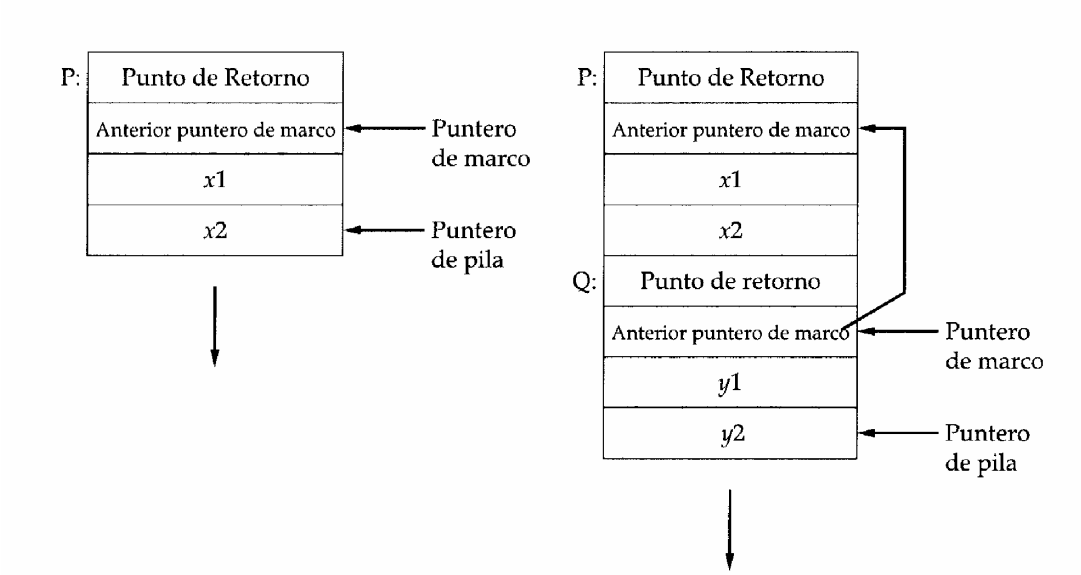


FIGURA 1.30 Crecimiento del marco de pila mediante los procedimientos de muestra P y Q [DEWA90]

La reentrada tiene dos aspectos clave: El código del programa no puede modificarse a sí mismo y los datos locales para cada usuario se deben almacenar por separado. Un procedimiento reentrante puede ser interrumpido y llamado por un programa y seguir ejecutando correctamente al retornar al procedimiento. En un sistema compartido, la reentrada permite un uso más eficiente de la memoria principal: Aunque se almacena una sola copia del código del programa en la memoria principal, más de una aplicación pueden llamar al procedimiento.

Así pues, un procedimiento reentrante debe tener una parte permanente (las instrucciones que constituyen el procedimiento) y una parte temporal (un puntero hacia atrás al procedimiento que llamó, así como memoria para las variables locales utilizadas por el programa). Cada caso particular de ejecución de un procedimiento, que se denomina *activación,* ejecutará el código que está en la parte permanente, pero tendrá su propia copia de las variables locales y de los parámetros. La parte temporal asociada con una activación en particular se conoce como *registro de activación.*

La forma más conveniente de dar soporte a los procedimientos reentrantes es por medio de una pila. Cuando se llama a un procedimiento reentrante, se puede almacenar en la pila el registro de activación de dicho procedimiento. De esta manera, el registro de activación se convierte en parte del marco de la pila que se crea al LLAMAR al procedimiento.

CAPÍTULO 2

Introducción a los Sistemas Operativos

El estudio de los sistemas operativos va a comenzar con una breve historia. Esta historia es interesante en sí misma, a la vez que ofrece una panorámica de los principios en que se basan los sistemas operativos.

El capítulo comienza con una ojeada a las funciones y objetivos de Los sistemas operativos, lo que servirá para definir los requisitos que debe satisfacer el diseño de un sistema operativo. Luego se vera cómo han evolucionado los sistemas operativos, desde los primitivos sistemas de proceso por lotes hasta los sofisticados sistemas multimodo y multiusuario. En el resto del capítulo se analiza la historia y las características generales de tres sistemas operativos que servirán de ejemplo a lo largo del libro. Es una feliz coincidencia que estos no sólo sean quizá los tres mejores ejemplos que podrían usarse en este libro, sino que además recogen los logros más importantes en la historia de los sistemas operativos.

2.1 FUNCIONES Y OBJETIVOS DE LOS SISTEMAS OPERATIVOS

Un sistema operativo es un programa que controla la ejecución de los programas de aplicación y que actúa como interfaz entre el usuario de un computador y el hardware de la misma. Puede considerarse que un sistema operativo tiene tres objetivos o lleva a cabo tres funciones:

- *Comodidad:* Un sistema operativo hace que un computador sea más cómoda de utilizar.
- *Eficiencia:* Un sistema operativo permite que los recursos de un sistema informático se aprovechen de una manera más eficiente.
- *Capacidad de evolución:* Un sistema operativo debe construirse de modo que permita el desarrollo efectivo, la verificación y la introducción de nuevas funciones en el sistema y, a la vez, no interferir en los servicios que brinda.

A continuación se van a tratar estos tres aspectos de los sistemas operativos.

El Sistema Operativo como Interfaz Usuario/Computadora

El hardware y el software que se utilizan para proveer de aplicaciones a los usuarios pueden contemplarse de forma estratificada o jerárquica, como se muestra en la figura 2.1. Al usuario de estas aplicaciones se le llama *usuario final* y, generalmente, no tiene que ocuparse de la arquitectura del computador. Por tanto, el usuario final ve al sistema informático en términos de aplicaciones. Las aplicaciones pueden construirse con un lenguaje de programación y son desarrolladas por programadores de aplicaciones. Si se tuviera que desarrollar un programa de aplicación como un conjunto de instrucciones máquina que sean del todo responsables del control del hardware, se tendría una tarea abrumadora y compleja. Para facilitar esta tarea, se ofrecen una serie de programas de sistemas. Algunos de estos programas se denominan *utilidades* e implementan funciones muy utilizadas que ayudan a la creación de los programas, la gestión de los archivos y el control de los dispositivos de E/S. Los programadores hacen uso de estos servicios en el desarrollo de una aplicación y ésta, mientras se está ejecutando, invoca a estas utilidades para llevar a cabo ciertas acciones. El programa de sistemas más importante es el sistema operativo. El sistema operativo oculta al programador los detalles del hardware y le proporciona una interfaz cómoda para utilizar el sistema. Actúa como mediador, facilitándole al programador y a los programas de aplicación el acceso y uso de todas esas características y servicios.

De forma resumida, un sistema operativo ofrece servicios en las áreas siguientes:

• *Creación de programas:* El sistema operativo ofrece una variedad de características y servicios, tales como los editores y los depuradores *(debuggers),* para ayudar al programador en la creación de programas. Normalmente, estos servicios están en forma de programas de utilidad que no forman realmente parte del sistema operativo, pero que son accesibles a través del mismo.

• *Ejecución de programas:* Para ejecutar un programa se necesita un cierto número de tareas. Las instrucciones y los datos se deben cargar en la memoria principal, los archivos y los dispositivos de E/S se deben inicializar y se deben preparar otros recursos. El sistema operativo administra todas estas tareas para el usuario.

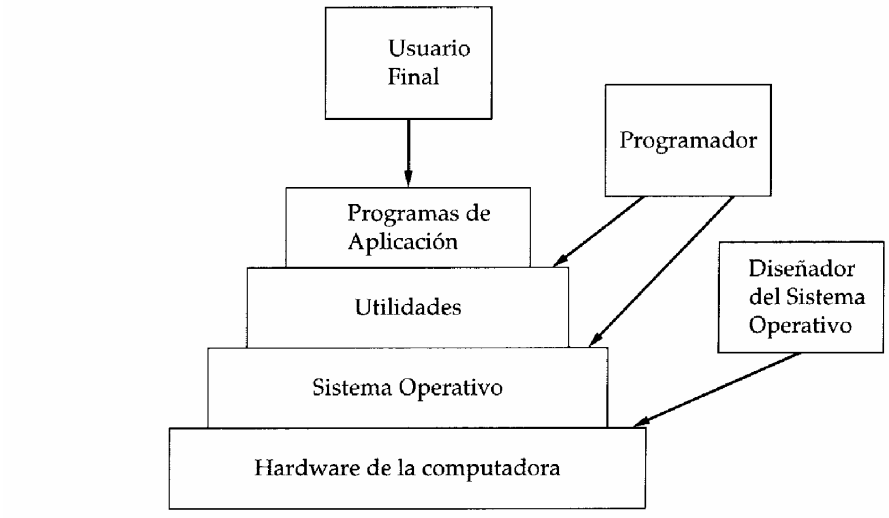


FIGURA 2.1. Niveles y vistas de un sistema informático

• *Acceso a los dispositivos de E/S:* Cada dispositivo de E/S requiere un conjunto propio y peculiar de instrucciones o de señales de control para su funcionamiento. El sistema operativo tiene en cuenta estos detalles de modo que el programador pueda pensar en forma de lecturas y escrituras simples.

• *Acceso controlado a los archivos:* En el caso de los archivos, el control debe incluir una comprensión, no sólo de la naturaleza del dispositivo de E/S (controlador de disco, controlador de cinta) sino del formato de los archivos y del medio de almacenamiento. Una vez más, es el sistema operativo el que se encarga de los detalles. Es más, en el caso de sistemas con varios usuarios trabajando simultáneamente, es el sistema operativo el que brinda los mecanismos de control para controlar el acceso a los archivos.

• *Acceso al sistema:* En el caso de un sistema compartido o público, el sistema operativo controla el acceso al sistema como un todo y a los recursos específicos del sistema. Las funciones de acceso pueden brindar protección, a los recursos y a los datos, ante usuarios no autorizados y debe resolver los conflictos en la propiedad de los recursos.

• *Detección y respuesta a errores:* Cuando un sistema informático está en funcionamiento pueden producirse varios errores. Entre estos se incluyen los errores internos y externos del hardware, tales como los errores de memoria, fallos o mal funcionamiento de dispositivos y distintos tipos de errores de software, como el desbordamiento aritmético, el intento de acceder a una posición prohibida de memoria y la incapacidad del sistema operativo para satisfacer la solicitud de una aplicación. En cada caso, el sistema operativo debe dar una respuesta que elimine la condición de error con el menor impacto posible sobre las aplicaciones que están en ejecución. La respuesta puede ser desde terminar el programa que produjo el error, hasta reintentar la operación o, simplemente, informar del error a la aplicación.

• *Contabilidad:* Un buen sistema operativo debe recoger estadísticas de utilización de los diversos recursos y supervisar los parámetros de rendimiento tales como el tiempo de respuesta. Para cualquier sistema, esta información es útil para anticiparse a la necesidad de mejoras futuras y para ajusfar el sistema y así mejorar su rendimiento. En un sistema multiusuario, la información puede ser utilizada con propósito de cargar en cuenta.

El sistema operativo como administrador de recursos

Un computador es un conjunto de recursos para el traslado, almacenamiento y proceso de datos y para el control de estas funciones. El sistema operativo es el responsable de la gestión de estos recursos.

¿Se puede afirmar que es el sistema operativo el que controla el traslado, almacenamiento y proceso de los datos? Desde un punto de vista, la respuesta es afirmativa: Administrando los recursos del computador, el sistema operativo tiene el control sobre las funciones básicas de la misma. Pero este control se ejerce de una manera curiosa. Normalmente, se piensa en un mecanismo de control como algo externo a lo controlado o, al menos, como algo distinto y una parte separada de lo controlado. (Por ejemplo, un sistema de calefacción de una estancia es controlado por un termostato, que es algo completamente diferente de los aparatos de generación de calor y de distribución del calor). Este no es el caso de un sistema operativo, que no es habitual como mecanismo de control en dos aspectos:

• El sistema operativo funciona de la misma manera que el software normal de un computador, es decir, es un programa ejecutado por el procesador.

• El sistema operativo abandona con frecuencia el control y debe depender del procesador para recuperarlo.

El sistema operativo es, de hecho, nada más que un programa del computador. Como otros programas de computador, da instrucciones al procesador. La diferencia clave está en el propósito del programa. El sistema operativo dirige al procesador en el empleo de otros recursos del sistema y en el control del tiempo de ejecución de otros programas. Pero para que el procesador pueda hacer estas cosas, debe cesar la ejecución del programa del sistema operativo y ejecutar otros programas. Así pues, el sistema operativo cede el control al procesador para hacer algún trabajo "útil" y luego lo retoma durante el tiempo suficiente para preparar el procesador para llevar a cabo la siguiente parte del trabajo. Los mecanismos involucrados se irán esclareciendo a medida que se avance en el capítulo.

La figura 2.2 propone los recursos principales que son administrados por el sistema operativo. Una parte del sistema operativo está en la memoria principal. En esta parte está el núcleo *(kernel),* que incluye las funciones utilizadas con más frecuencia en el sistema operativo y, en un momento dado, puede incluir otras partes del sistema operativo que estén en uso. El resto de la memoria principal contiene datos y otros programas de usuario. Como se

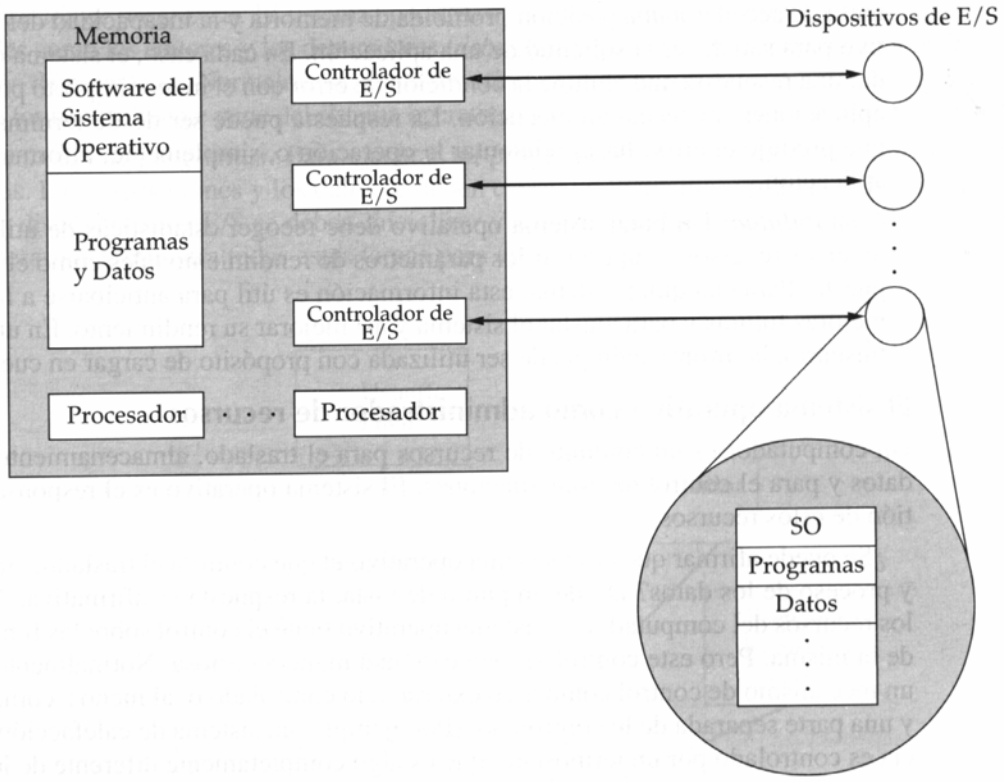


FIGURA 2.2 El sistema operativo como administrador de recursos

[1]Una cantidad creciente de sistemas operativos toma partido cada vez más por *firmware* en vez de por el software. Esto no altera los argumentos de modo sensible.

verá, la asignación de este recurso (la memoria principal) es controlada conjuntamente por el sistema operativo y por el hardware de gestión de memoria en el procesador. El sistema operativo decide cuándo puede utilizarse un dispositivo de E/S por parte de un programa en ejecución y controla el acceso y la utilización de los archivos. El procesador es, en si mismo, un recurso y es el sistema operativo el que debe determinar cuánto tiempo del procesador debe dedicarse a la ejecución de un programa de usuario en particular. En el caso de sistemas multiprocesador, la decisión debe distribuirse entre todos los procesadores.

Facilidad de evolución de un sistema operativo

Un sistema operativo importante evolucionará en el tiempo por una serie de razones:

- *Actualizaciones del hardware* y *nuevos tipos de hardware:* Por ejemplo, las primeras versiones de UNIX y OS/2 no empleaban mecanismos de paginación, porque funcionaban en máquinas sin hardware de paginación[2]. Las versiones más recientes se han modificado para aprovechar las capacidades de paginación. Además, el empleo de terminales gráficos y terminales de pantalla completa, en lugar de los terminales de líneas, pueden influir en el diseño de los sistemas operativos. Por ejemplo, un terminal de éstos puede permitirle al usuario ver diferentes aplicaciones al mismo tiempo, a través de "ventanas" en la pantalla. Esto necesita un soporte más sofisticado en el sistema operativo.

- *Nuevos servicios:* Como respuesta a Las demandas del usuario o a las necesidades de los administradores del sistema, el sistema operativo ampliará su oferta de servicios. Por ejemplo, si se determina que es difícil de mantener un buen rendimiento para los usuarios con las herramientas existentes, se deben añadir nuevas medidas y herramientas de control al sistema operativo. Otro ejemplo es el de las nuevas aplicaciones que exigen el uso de ventanas en la pantalla. Esta característica requiere actualizaciones mayores en el sistema operativo.

- *Correcciones:* Desafortunadamente, el sistema operativo tiene fallos que se descubrirán con el curso del tiempo y que es necesario corregir. Por supuesto, estas correcciones pueden introducir nuevos fallos a su vez y así sucesivamente.

La necesidad de hacer cambios en un sistema operativo de forma regular introduce ciertos requisitos en el diseño. Una afirmación obvia es que el sistema debe tener una construcción modular, con interfaces bien definidas entre los módulos y debe estar bien documentado. Para programas grandes, como normalmente son los sistemas operativos actuales, no es adecuado lo que podría denominarse modularización elemental [DENN80a]. Es decir, debe hacerse mucho más que dividir simplemente un programa en subrutinas. Se volverá a este tema más adelante en el capítulo

2.2

EVOLUCION DE LOS SISTEMAS OPERATIVOS

Para intentar comprender los requisitos básicos de un sistema operativo y el significado de las características principales de un sistema operativo contemporáneo, resulta útil considerar cómo han evolucionado los sistemas operativos a lo largo de los años.

[2]La paginación se introducirá de forma breve más adelante en este mismo capítulo y se discutirá en detalle en los capítulos 6 y 7.

Proceso en serie

En los primeros computadores, de finales de los 40 hasta mediados de los 50, el programador interactuaba directamente con el hardware; no había sistema operativo. La operación con estas máquinas se efectuaba desde una consola consistente en unos indicadores luminosos, unos conmutadores, algún tipo de dispositivo de entrada y una impresora. Los programas en código máquina se cargaban a través del dispositivo de entrada (un lector de tarjetas, por ejemplo). Si se detiene el programa por un error, la condición de error se indicaba mediante los indicadores luminosos. El programador podía examinar los registros y la memoria principal para determinar la causa del error. Si el programa continuaba hasta su culminación normal, la salida aparecería en la impresora.

Estos primeros sistemas presentaban dos problemas principales:

- *Planificación:* La mayoría de las instalaciones empleaban un formulario de reserva de tiempo de máquina. Normalmente, un usuario podía reservar bloques de tiempo en múltiplos de media hora o algo por el estilo. Un usuario podía reservar una hora y terminar a los 45 minutos; esto daba como resultado un desperdicio del tiempo del computador. Por el contrario, el usuario podía tener dificultades, no terminar en el tiempo asignado y verse forzado a parar sin haber solucionado el problema.
- *Tiempo de preparación:* Un programa sencillo, llamado trabajo, cargaba un compilador y un programa en lenguaje de alto nivel (programa fuente) en la memoria, salvaba el programa compilado (programa objeto) y luego montaba y cargaba el programa objeto junto con las funciones comunes. Cada uno de estos pasos podía implicar montar y desmontar cintas o preparar paquetes de tarjetas. Si se producía un error, el infortunado usuario tenía que volver al inicio de este proceso de preparación. De este modo, se perdía un tiempo considerable en preparar un programa para su ejecución.

Este modo de operación podría denominarse *proceso en serie* porque refleja el hecho de que los usuarios tenían que acceder al computador en serie. Con el paso del tiempo se desarrollaron varias herramientas de software de sistemas para intentar hacer más eficiente este proceso en serie. Entre éstas se incluían bibliotecas de funciones comunes, montadores, cargadores, depuradores y rutinas de manejo de E/S que estaban disponibles como un software común para todos los usuarios.

Sistemas sencillos de proceso por lotes

Las primeras máquinas eran muy caras y, por tanto, era importante maximizar la utilización de las mismas. El tiempo desperdiciado por la planificación y la preparación era inaceptable.

Para mejorar el uso, se desarrolló el concepto de sistema operativo por lotes *(batch)*. El primer sistema operativo por lotes fue desarrollado a mediados de los 50 por la General Motors para usar en un IBM 701 [WEIZ81]. Este concepto fue refinado posteriormente e implementado en un IBM 704 por una serie de clientes de IBM. A principios de los 60, un conjunto de constructores ya habían desarrollado sistemas operativos por lotes para sus computadores. IBSYS, el sistema operativo de IBM para las computadores 7090/7094, es particularmente notable por su amplia influencia en otros sistemas.

La idea central que está detrás del esquema sencillo de proceso por lotes es el uso de un elemento de software conocido como monitor. Con el uso de esta clase de sistema opera-

tivo, los usuarios ya no tenían acceso directo a la máquina. En su lugar, el usuario debía entregar los trabajos en tarjetas o en cinta al operador del computador, quien agrupaba secuencialmente los trabajos por lotes y ubicaba los lotes enteros en un dispositivo de entrada para su empleo por parte del monitor. Cada programa se construía de modo tal que volviera al monitor al terminar su procesamiento y, en ese momento, el monitor comenzaba a cargar automáticamente el siguiente programa.

Para entender cómo funciona este esquema, se va a ver desde dos puntos de vista: el del monitor y el del procesador. Desde el punto de vista del monitor, él es quien controla la secuencia de sucesos. Para que esto sea posible, gran parte del monitor debe estar siempre en memoria principal y disponible para su ejecución (figura 2.3). Esta parte del monitor se conoce como monitor residente. El resto del monitor consta de utilidades y funciones comunes que se cargan como subrutinas en los programas de los usuarios al comienzo de cualquier trabajo que las necesite. El monitor lee los trabajos uno a uno del dispositivo de entrada (normalmente, un lector de tarjetas o una unidad de cinta magnética). A medida que lo lee, el trabajo actual se ubica en la zona del programa de usuario y el control pasa al trabajo. Cuando el trabajo termina, se devuelve el control al monitor, quien lee inmediatamente un nuevo trabajo. Los resultados de cada trabajo se imprimen y entregan al usuario.

Considérese ahora esta secuencia desde el punto de vista del procesador. En un cierto momento, el procesador estará ejecutando instrucciones de la zona de memoria principal que contiene al monitor. Estas instrucciones hacen que el trabajo siguiente sea leído en otra zona de la memoria principal. Una vez que el trabajo se ha leído, el procesador encuentra en el monitor una instrucción de desvío que ordena al procesador continuar la ejecución en el inicio del programa de usuario. El procesador ejecuta entonces las instrucciones del programa de usuario hasta que encuentre una condición de finalización o de error. Cualquiera de estos

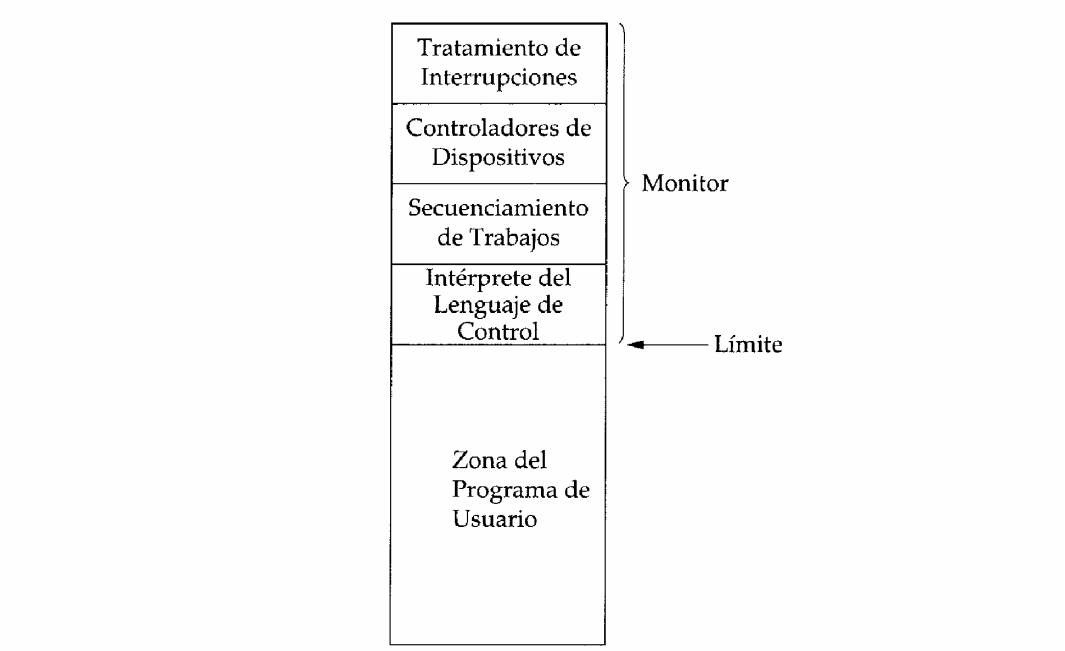


FIGURA 2.3 Disposición de la memoria con un monitor residente [SILB94]

dos sucesos provocan que el procesador vaya a por la instrucción siguiente del programa monitor. De este modo, la frase "el control se le pasa al trabajo" quiere decir simplemente que el procesador pasa a leer y ejecutar instrucciones del programa de usuario, mientras que la frase "el control vuelve al monitor" quiere decir que el procesador pasa ahora a leer y ejecutar las instrucciones del programa monitor.

Debe quedar claro que es el monitor el que gestiona el problema de la planificación. Se pone en cola un lote de trabajos y éstos son ejecutados tan rápido como es posible, sin que haya tiempo alguno de desocupación.

¿Qué ocurre con la preparación de los trabajos? El monitor también se encarga de esto. Con cada trabajo, se incluyen instrucciones de una forma primitiva de lenguaje de control de trabajos (JCL, *Job Control Lenguaje),* que es un tipo especial de lenguaje de programación empleado para dar instrucciones al monitor. La figura 2.4 muestra un ejemplo sencillo con entrada de trabajos desde tarjetas. En este ejemplo, el usuario envía un programa escrito en FORTRAN junto a unos datos que se utilizarán en el programa. Además de las tarjetas de FORTRAN y de datos, el paquete incluye instrucciones de control de trabajos, que se denotan mediante un signo dólar ($) al comienzo.

Para ejecutar el trabajo, el monitor lee la tarjeta $FTN y carga el compilador adecuado desde el dispositivo de almacenamiento masivo (generalmente una cinta). El compilador traduce el programa de usuario en código objeto, que se almacena en memoria o en el dispositivo de almacenamiento. Si se carga en memoria, la operación es conocida como "compilar, cargar y arrancar" *(compile, load, and go).* Si se almacena en cinta, entonces se requiere la tarjeta $LOAD. Esta tarjeta es leída por el monitor, quien retoma el control después de la operación de compilación. El monitor llama al cargador, que carga el programa objeto en memoria en el lugar del compilador y le transfiere el control. De esta manera, un segmento grande de memoria se puede compartir entre diferentes subsistemas, aunque en cada momento sólo uno de ellos tiene que estar presente y ejecutándose.

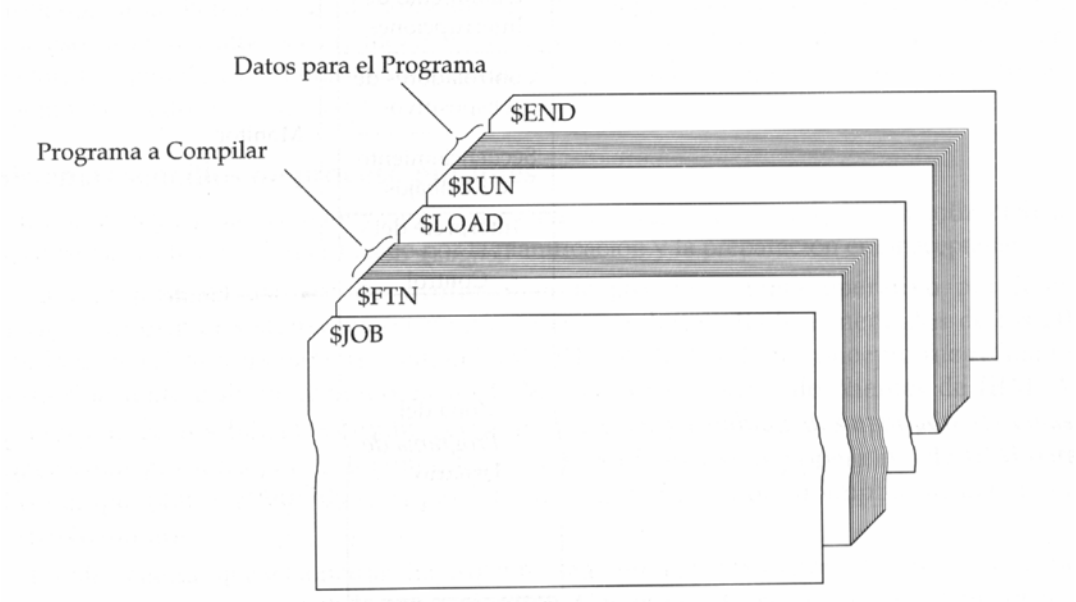


FIGURA 2.4 Paquete de tarjetas para un sistema sencillo por lotes

Durante la ejecución del programa de usuario, cada instrucción de entrada origina la lectura de una tarjeta de datos. La instrucción de entrada en el programa del usuario hace que se invoque una rutina de entrada, que forma parte del sistema operativo. La rutina de entrada se asegura de que el programa de usuario no ha leído accidentalmente una tarjeta JCL. Si esto sucede, se produce un error y el control se transfiere al monitor. Al terminar un trabajo, con o sin éxito, el monitor recorre las tarjetas de entrada hasta encontrar la próxima tarjeta JCL. De este modo, el sistema se protege contra un programa que tenga tarjetas de datos de más o de menos.

Se comprobará que el monitor o el sistema de proceso por lotes es simplemente un programa de computador. Se basa en la capacidad del procesador para traer y ejecutar instrucciones desde varias zonas de la memoria principal y así apoderarse y ceder el control de forma alterna. Para esto serian convenientes algunas otras características del hardware, entre las que se encuentran las siguientes:

• *Protección de memoria:* Mientras el programa de usuario esté ejecutándose, no debe modificar la zona de memoria en la que está el monitor. Si se hace un intento tal, el hardware del procesador deberá detectar el error y transferir el control al monitor. El monitor abortará entonces el trabajo, imprimirá el mensaje de error y cargará el siguiente trabajo.

• *Temporizador:* Se utiliza un temporizador para impedir que un sólo trabajo monopolice el sistema. El temporizador se lanza al comenzar cada trabajo. Si expira el tiempo, se producirá una interrupción y el control volverá al monitor.

• *Instrucciones Privilegiadas:* Ciertas instrucciones son designadas como privilegiadas y pueden ser ejecutadas solo por el monitor. Si el procesador encuentra una instrucción tal, cuando está ejecutando el programa del usuario, se producirá una interrupción de error. Entre las instrucciones privilegiadas se encuentran las instrucciones de E/S, de forma que el monitor retenga el control de todos los dispositivos de E/S. Esto impide, por ejemplo, que un programa de usuario lea accidentalmente instrucciones de control que son del trabajo siguiente. Si un programa de usuario desea realizar una *E/S,* debe solicitarse al monitor que haga la operación por él. Si el procesador encuentra una instrucción privilegiada cuando está ejecutando un programa de usuario, el hardware del procesador la considera como un error y transfiere el control al monitor.

• *Interrupciones:* Los primeros modelos de computadores no tenían esta capacidad. Esta característica aporta al sistema operativo más flexibilidad para ceder y retomar el control de los programas usuarios.

Naturalmente, se puede construir un sistema operativo sin estas características, pero los fabricantes de computadores comprobaron rápidamente que los resultados eran caóticos y, por tanto, incluso los sistemas operativos por lotes más primitivos ya disponían de estas características en el hardware. Por otro lado, hay que decir que el sistema operativo más utilizado del mundo, el PC-DOS/MS-DOS, no dispone de protección de memoria ni de instrucciones privilegiadas de E/S. Sin embargo, como este sistema está destinado a computadores personales de un solo usuario, los problemas que se pueden originar son menos graves.

En un sistema operativo por lotes, el tiempo de máquina se reparte entre la ejecución de programas de usuario y la ejecución del monitor. Así se tienen dos pérdidas: se entrega al monitor cierta cantidad de memoria principal y éste consume cierto tiempo de la máquina. Ambas pérdidas son una forma de sobrecarga. Aún con esta sobrecarga, los sistemas operativos sencillos por lotes mejoran el uso del computador.

Sistemas por lotes con multiprogramación

Aún con el secuenciamiento automático de los trabajos ofrecido por un sistema operativo sencillo por lotes, el procesador está desocupado a menudo. El problema es que los dispositivos de E/S son lentos comparados con el procesador. La figura 2.5 detalla un cálculo representativo. Los números corresponden a un programa que procesa un archivo de registros y ejecuta, en promedio, 100 instrucciones de máquina por cada registro. En este ejemplo, el computador gasta más del 96% del tiempo esperando a que los dispositivos de E/S terminen de transferir sus datos. La figura 2.6a ilustra esta situación. El procesador gasta parte del tiempo ejecutando hasta que encuentra una instrucción de E/S. Entonces debe esperar a que concluya la instrucción de E/S antes de continuar.

Esta ineficiencia no es necesaria. Se sabe que hay memoria suficiente para almacenar el sistema operativo (el monitor residente) y un programa de usuario. Supóngase que hay espacio suficiente para el sistema operativo y dos programas usuarios. Ahora, cuando un trabajo necesite esperar una E/S, el procesador puede cambiar al otro trabajo, que probablemente no estará esperando a la E/S (figura 2.6b). Además, se podría ampliar la memoria para almacenar tres, cuatro o más programas y conmutar entre todos ellos (figura 2.6c). Este proceso es conocido como multiprogramador o multitarea. Éste es el punto central de los sistemas operativos modernos.

Para ilustrar el beneficio de la multiprogramación, considérese un ejemplo basado en uno de Tumer [TURN86]. Sea un computador con 256K palabras de memoria disponible (no utilizadas por el sistema operativo), un disco, una terminal y una impresora. Tres programas, TRABAJO 1, TRABAJ02 y TRABAJOS, son enviados para su ejecución al mismo tiempo, con los atributos que se enumeran en la tabla 2.1. Se suponen unos requisitos mínimos de procesador para el TRABAJ02 y el TRABAJOS y un uso continuado del disco y de la impresora por parte del TRABAJOS. En un sistema sencillo por lotes, estos trabajos serían ejecutados en secuencia. Así pues, el TRABAJO 1 termina en 5 minutos. El TRABAJ02 debe esperar a que transcurran esos 5 minutos y terminar 15 minutos después. El TRABAJOS comienza después de los 20 minutos para terminar SO minutos después del momento en que fue lanzado. La utilización media de los recursos, la productividad y los tiempos de respuesta se ilustran en la columna de monoprogramación de la tabla 2.2. El uso de dispositivos queda ilustrado en la figura 2.7. Es evidente que hay una infrautilización neta de todos los recursos cuando se promedian los tiempos de uso en el período exigido de SO minutos.

Supóngase ahora que los trabajos se ejecutan concurrentemente en un sistema operativo con monoprogramación. Como hay poca contención de recursos entre los trabajos, cada uno de los tres puede ejecutarse en un tiempo cercano a 1 mínimo mientras coexiste con los otros en el computador (suponiendo que a TRABAJ02 y TRABAJO3 se les adjudica tiempo su-

Leer un registro	0,0015 segundos
Ejecutar 100 instrucciones	0,0001 segundos
Escribir un registro	0,0015 segundos
TOTAL	0,0031 segundos

Porcentaje de Utilización de la CPU = 0,0001 / 0,0031 = 0,032 = 3,2%

FIGURA 2.5 Ejemplo de utilización del sistema

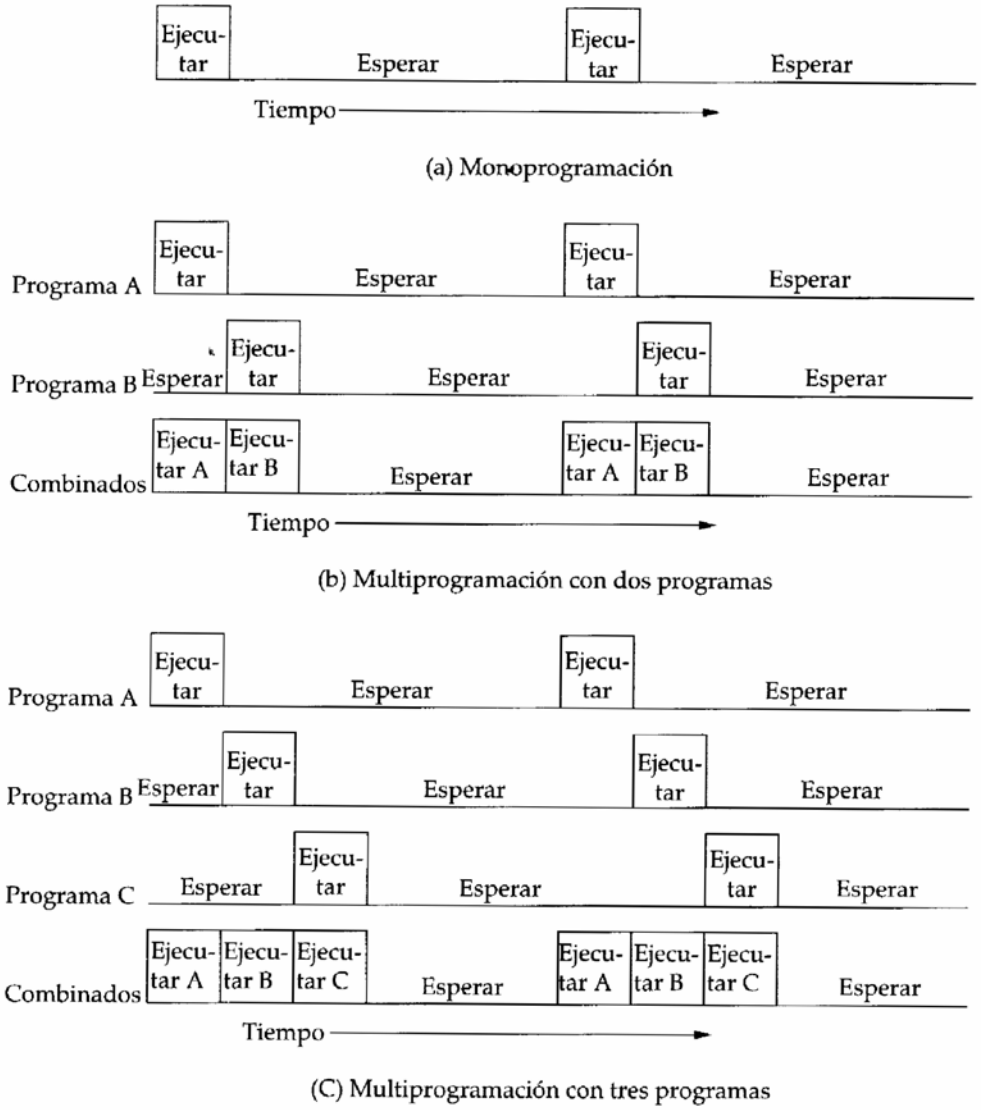


FIGURA 2.6 Multiprogramación

TABLA 2.1 Atributos de Ejemplo de la Ejecución de un Programa

	TRABAJO1	TRABAJO2	TRABAJO3
Tipo de trabajo	Cálculo intensivo	E/S intensiva	E/S intensiva
Duración	5 min	15 min	10 min
Memoria exigida	50 K	100 K	80 K
¿Necesita disco?	No	No	Sí
¿Necesita terminal?	No	Sí	No
¿Necesita impresora?	No	No	Sí

Tabla 2.2 Efectos de la Multiprogramación sobre la utilización de recursos

	Monoprogramación	Multiprogramación
Uso del procesador	17%	33%
Uso de la memoria	30%	67%
Uso del disco	33%	67%
Uso de la impresora	33%	67%
Tiempo transcurrido	30 min	15 min
Tasa de productividad	6 trabajos/hora	12 trabajos/hora
Tiempo medio de respuesta	18 min	10 min

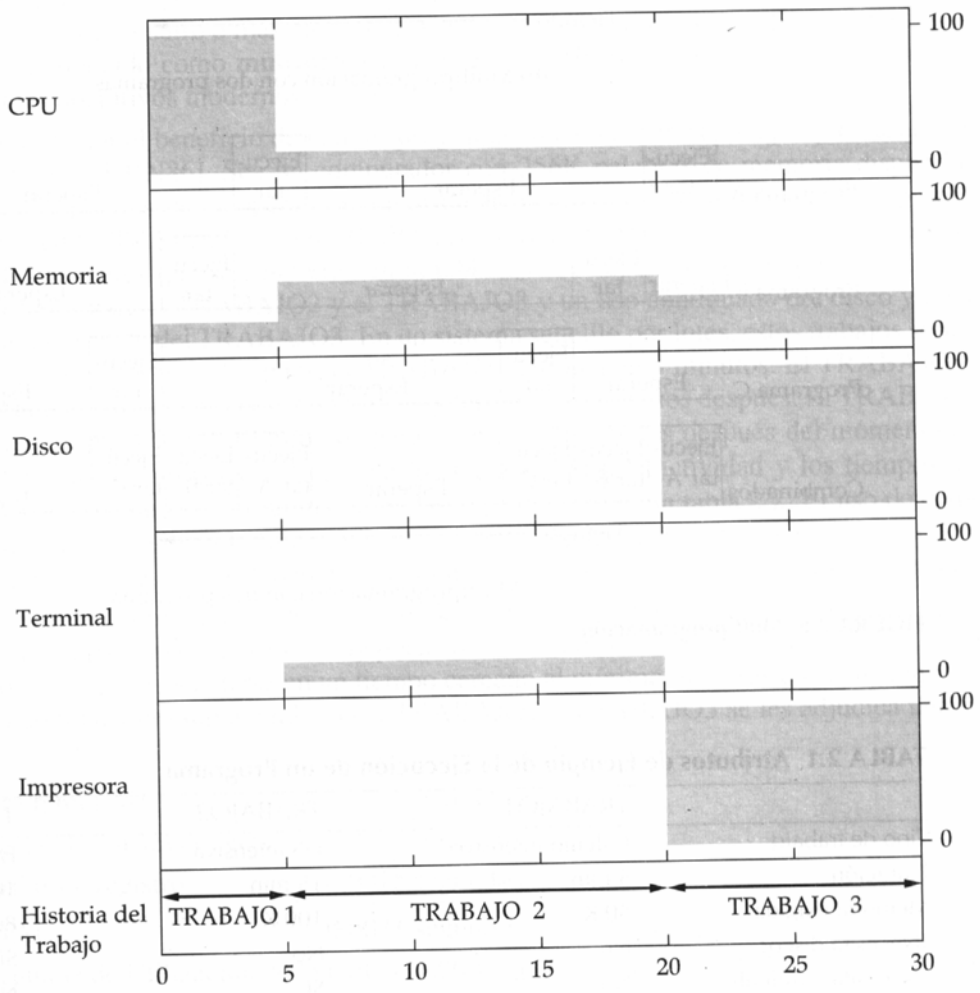


FIGURA 2.7 Histograma de utilización con monoprogramación

ficiente de procesador para mantener activas sus operaciones de E/S). El TRABAJO1 requerirá 5 minutos para terminar, pero al finalizar este tiempo, el TRABAJO2 estará terminado en una tercera parte y el TRABAJO3 estará a la mitad. Los tres trabajos habrán terminado dentro de 15 minutos. La mejora es evidente cuando se examina la columna de multiprogramación de la tabla 2.2, obtenida del histograma que se muestra en la figura 2.8.

Al igual que un sistema sencillo por lotes, un sistema por lotes con multiprogramación tiene que depender de ciertas características del hardware del computador. La característica adicional más notable y útil para la multiprogramación es que el hardware respalde las interrupciones de E/S y el DMA. Con E/S dirigida por interrupciones y con DMA, el procesador puede enviar una orden de E/S para un trabajo y continuar con la ejecución de otro, mientras la E/S es efectuada por el controlador del dispositivo. Cuando termina la operación de E/S, el proce-

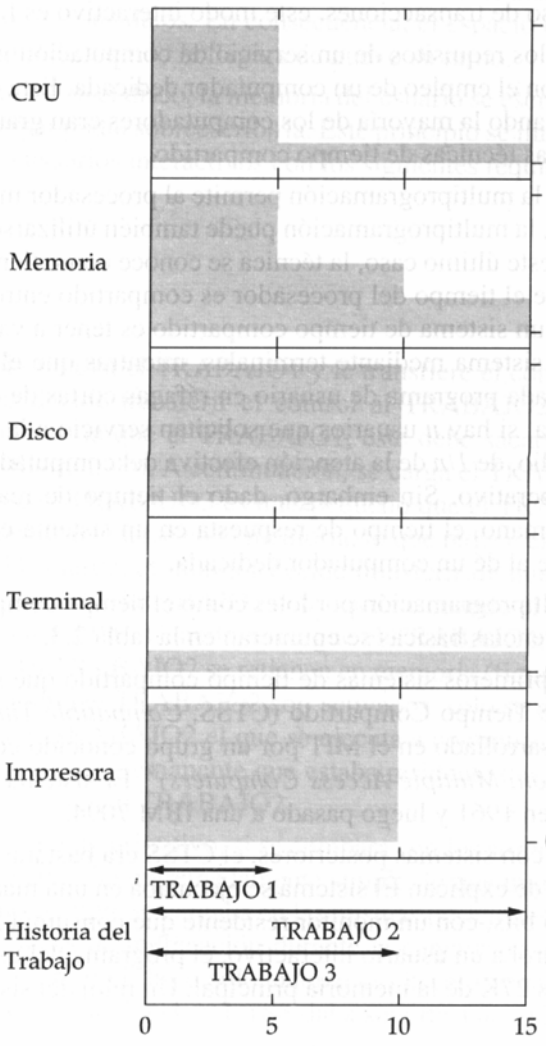


FIGURA 2.8. Histograma de utilización con multiprogramación [TURN86]

sador es interrumpido y el control pasa a un programa de tratamiento de interrupciones del sistema operativo. El sistema operativo le pasa entonces el control a otro trabajo.

Los sistemas operativos con multiprogramación son bastante más sofisticados en comparación con los sistemas de monoprogramación o de un solo programa. Para tener varios trabajos listos para ejecutar, éstos deben mantenerse en la memoria principal, lo que requiere cierto tipo de gestión de memoria. Además, si hay varios trabajos listos para ejecutarse, el procesador debe decidir cuál de ellos va a ejecutar, lo que requiere un algoritmo de planificación. Estos conceptos serán discutidos más adelante en este capítulo.

Sistemas de tiempo compartido

Con el uso de la multiprogramación, el tratamiento por lotes puede llegar a ser bastante eficiente. Sin embargo, para muchas tareas, es conveniente suministrar un modo en que el usuario interactúe directamente con el computador. De hecho, para algunos trabajos, tales como el proceso de transacciones, este modo interactivo es fundamental.

Hoy en día, los requisitos de un servicio de computación interactiva pueden y suelen llevarse a cabo con el empleo de un computador dedicada. Esta opción no estaba disponible en los años 60, cuando la mayoría de los computadores eran grandes y costosas. En su lugar, se desarrollaron las técnicas de tiempo compartido.

Al igual que la multiprogramación permite al procesador manejar varias tareas por lotes al mismo tiempo, la multiprogramación puede también utilizarse para manejar varias tareas interactivas. En este último caso, la técnica se conoce como tiempo compartido, porque refleja el hecho de que el tiempo del procesador es compartido entre los diversos usuarios. La técnica básica de un sistema de tiempo compartido es tener a varios usuarios utilizando simultáneamente el sistema mediante terminales, mientras que el sistema operativo intercala la ejecución de cada programa de usuario en ráfagas cortas de cómputo o cuantos *(quantum)*. De esta manera, si hay *n* usuarios que solicitan servicio a la vez, cada usuario sólo dispondrá, en promedio, de *Un* de la atención efectiva del computador, sin contar con la sobrecarga del sistema operativo. Sin embargo, dado el tiempo de reacción relativamente lento que tiene el ser humano, el tiempo de respuesta en un sistema correctamente diseñado debería ser comparable al de un computador dedicada.

Tanto la multiprogramación por lotes como el tiempo compartido utilizan multiprogramación. Las diferencias básicas se enumeran en la tabla 2.3.

Uno de los primeros sistemas de tiempo compartido que se desarrollaron fue el Sistema Compatible de Tiempo Compartido (CTSS, *Compatible Time-Sharing System)* [CORB62, CORB63], desarrollado en el MIT por un grupo conocido como Proyecto MAC *(Machine-Aided Cognition, Multiple-Access Computers)*[3]. El sistema fue desarrollado primero para una IBM 709 en 1961 y luego pasado a una IBM 7094.

Comparado con sistemas posteriores, el CTSS era bastante primitivo y su funcionamiento básico es fácil de explicar. El sistema se ejecutaba en una máquina con una memoria de 32K palabras de 36 bits, con un monitor residente que consumía 5K del total. Cuando había que asignar el control a un usuario interactivo, el programa del usuario y los datos eran cargados en las restantes 27K de la memoria principal. Un reloj del sistema generaba interrupciones a

[3] Conocimiento Asistido por Computadora, Computadoras de Acceso Múltiple (N. del T.)

TABLA 2.3 Multiprogramación por lotes frente a Tiempo Compartido

	Multiprogramación por lotes	Tiempo Compartido
Objetivo principal	Maximizar la utilización del procesador	Minimizar el tiempo de respuesta
Origen de las instrucciones al sistema operativo	Instrucciones de un lenguaje de control de trabajos incluidas junto con el trabajo	Ordenes dadas en el terminal

razón de aproximadamente una cada 0,2 segundos (sg). En cada interrupción de reloj, el sistema operativo se adueñaba del control y le podía asignar el procesador a otro usuario. De esta manera, a intervalos regulares, el usuario en curso era expulsado y se cargaba otro usuario en su lugar. Para conservar el estado del usuario anterior, para su reanudación posterior, los programas del usuario anterior y sus datos eran escritos en el disco antes de leer los programas del nuevo usuario y sus datos. En consecuencia, el espacio de memoria del usuario anterior debía ser restaurado cuando le llegara de nuevo su tumo.

Para minimizar el tráfico en el disco, la memoria del usuario se escribía a disco sólo cuando el nuevo programa a cargar podía sobrescribirla. Este principio se ilustra en la figura 2.9. Supóngase que hay cuatro usuarios interactivos con los siguientes requisitos de memoria:

- TRABAJO1: 15K
- TRABAJO2: 20K
- TRABAJO3: 5K
- TRABAJO4: I0K

Al principio, el monitor carga el TRABAJOl y le transfiere el control (figura 2.9a). Posteriormente, el monitor decide transferir el control al TRABAJ02. Puesto que el TRABAJ02 requiere más memoria que el TRABAJOl, éste debe sacarse primero, para luego cargar el TRABAJ02 (figura 2.9b). A continuación, se carga el TRABAJO3 para ser ejecutado. Sin embargo, como el TRABAJO3 es más pequeño que el TRABAJ02, entonces una parte del TRABAJ02 puede quedarse en la memoria, lo que reduce el tiempo de escritura en el disco (figura 2.9c). Más tarde, el monitor decide transferir de nuevo el control al TRABAJOl. Una parte adicional del TRABAJ02 debe sacarse cuando el TRABAJOl se cargue de nuevo a memoria (figura 2.9d). Cuando se cargue el TRABAJ04, parte del TRABAJOl y de la parte remanente del TRABAJO2 se retienen en memoria (figura 2.9e). En este punto, tanto si el TRABAJOl como el TRABAJO2 son activados, sólo se necesita una carga parcial. En este ejemplo es el TRABAJ02 el que se ejecuta a continuación. Esto exige que se saquen el TRABAJ04 y la parte remanente que estaba residente del TRABAJOl, para que se pueda leer la parte que falta del TRABAJ02.

El enfoque del CTSS era muy primitivo, si se compara con los sistemas actuales de tiempo compartido, pero funcionaba. Era extremadamente simple, lo que minimizaba el tamaño del monitor. Como un trabajo siempre se cargaba en las mismas posiciones de memoria, no había necesidad de utilizar técnicas de reubicación durante la carga (que se discutirán más adelante). La técnica de escribir en el disco sólo cuándo era necesario minimizaba la actividad con el disco. Ejecutado sobre una 7094, el CTSS daba soporte a un máximo de 32 usuarios.

El tiempo compartido y la multiprogramación plantean una multitud de problemas nuevos *para el sistema operativo. Si hay varios trabajos en memoria, entonces deben* protegerse de

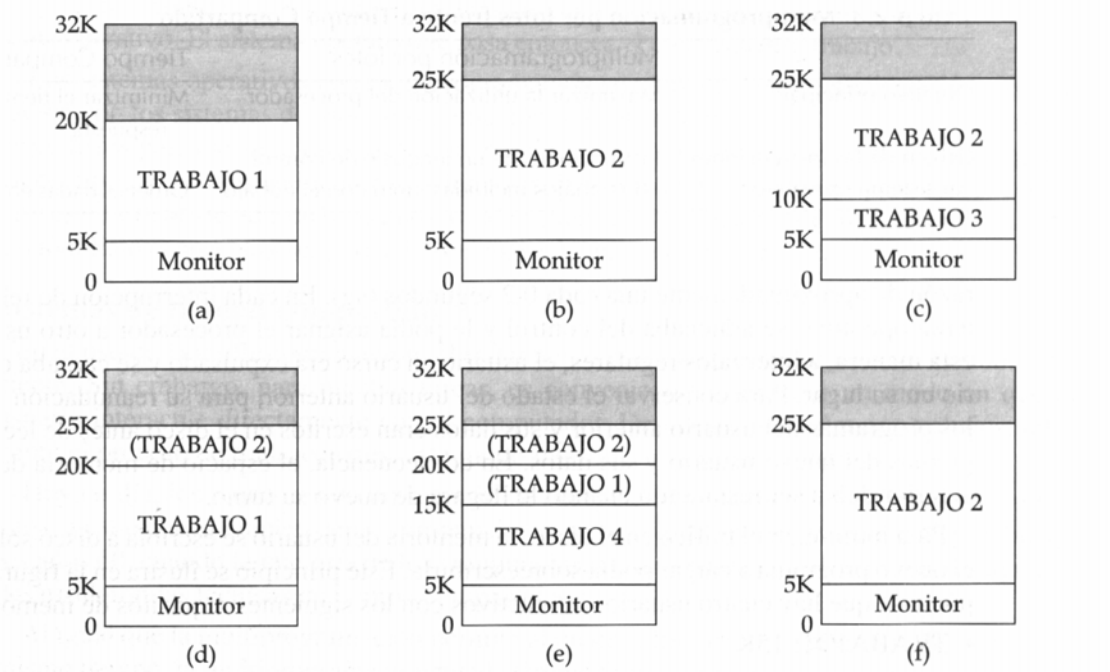


FIGURA 2.9 Funcionamiento del CTSS

injerencias unos de otros, como, por ejemplo, que uno modifique los datos de otro. Con varios usuarios interactivos, el sistema de archivos debe protegerse de forma que sólo los usuarios autorizados puedan tener acceso a un archivo en particular. La contención de recursos tales como la impresora y los dispositivos de almacenamiento masivo debe estar controlada. Este y otros problemas, con sus posibles soluciones, se encontrarán a lo largo del texto.

2.3 LOGROS PRINCIPALES

Los sistemas operativos están entre los elementos de software más complejos que se han desarrollado. Esto refleja el reto de tratar de conjugar las dificultades y, en algunos casos, objetivos opuestos de comodidad, eficiencia y capacidad de evolución. Denning y sus colegas [DENN80a] proponen que, hasta la fecha, se han obtenido cuatro logros intelectuales significativos en el desarrollo de los sistemas operativos:

- Los procesos
- La gestión de memoria
- La seguridad y la protección de la información
- La planificación y la gestión de recursos
- La estructura del sistema

Cada logro viene caracterizado por unos principios o abstracciones que se han desarrollado para solucionar las dificultades de los problemas prácticos. En conjunto, estos cinco

campos abarcan los puntos clave del diseño e implementación de los sistemas operativos modernos. La breve revisión que se hará de estos cinco campos en esta sección sirve como introducción a gran parte del texto restante.

Procesos

El concepto *de proceso es* fundamental en la estructura de los sistemas operativos. Este término fue acuñado por primera vez por los diseñadores de Multics en los años 60. Es un término algo más general que el de *trabajo.* Se han dado muchas definiciones para el término *proceso,* entre las que se incluyen las siguientes:

- Un programa en ejecución
- El "espíritu animado" de un programa
- La entidad que puede ser asignada al procesador y ejecutada por él.

El concepto de proceso debe quedar más claro a medida que se avance.

Tres líneas principales en el desarrollo de los sistemas informáticos crearon problemas de tiempos y de sincronización que contribuyeron al desarrollo del concepto de proceso: la operación por lotes con multiprogramación, el tiempo compartido y los sistemas de transacciones en tiempo real. Como se ha visto, la multiprogramación fue diseñada para mantener ocupados a la vez tanto procesador como los dispositivos de E/S, incluyendo los dispositivos de almacenamiento, de modo que se alcance la mayor eficiencia posible. La clave de este mecanismo es que, como respuesta a las señales que indiquen que ha terminado una transacción de E/S, el procesador cambia entre los diversos programas que residen en la memoria principal.

Una segunda línea de desarrollo fue la de los sistemas de tiempo compartido de propósito general. La justificación de tales sistemas es que los usuarios del computador son más productivos si pueden interactuar directamente con el computador desde algún tipo de terminal. En este caso, el objetivo clave del diseño es que el sistema sea sensible a las necesidades del usuario individual y que, además, por razones de coste, pueda dar soporte simultáneo a muchos usuarios. Estos objetivos son compatibles debido al tiempo de reacción relativamente lento que tienen los usuarios. Por ejemplo, si un usuario típico necesita, en promedio, 2 segundos de tiempo de procesamiento por minuto, entonces cerca de 30 usuarios deberían ser capaces de compartir el mismo sistema sin interferencias notables. Por supuesto, debe tenerse en cuenta la sobrecarga que impone el propio sistema operativo.

Otra línea importante de desarrollo la han constituido los sistemas de proceso de transacciones en tiempo real. En este caso, un cierto número de usuarios hacen consultas o actualizaciones sobre una base de datos. Un ejemplo clásico es un sistema de reservas de unas líneas aéreas. La diferencia clave entre un sistema de proceso de transacciones y un sistema de tiempo compartido es que el primero está limitado a una o pocas aplicaciones, mientras que los usuarios de un sistema de tiempo compartido pueden dedicarse al desarrollo de un programa, a la ejecución de trabajos y al uso de diferentes aplicaciones. En ambos casos, el tiempo de respuesta del sistema es primordial.

La herramienta principal disponible para los programadores de sistemas en el desarrollo de los primeros sistemas interactivos multiusuario y de multiprogramación fue la interrupción. La actividad de cualquier trabajo podía suspenderse por el acontecimiento de un suceso determinado, como la culminación de una E/S. El procesador debía entonces salvar al-

gún tipo de contexto (por ejemplo, el contador de programa y otros registros) y desviarse hacia una rutina de tratamiento de la interrupción, que determinaba la naturaleza de la interrupción, la procesaba y luego reanudaba el proceso del usuario en el trabajo interrumpido o en algún otro trabajo.

El diseño del software del sistema para coordinar estas diversas actividades resultó extraordinariamente difícil. Con muchos trabajos en progreso al mismo tiempo, donde cada uno involucraba numerosos pasos que dar de una manera secuencial, resultaba imposible de analizar todas las combinaciones posibles de secuencias de sucesos. En ausencia de un medio sistemático de coordinación y cooperación entre las actividades, los programadores recurrían a métodos *ad hoc* basados en su comprensión del entorno que el sistema operativo tenía que controlar. Estos esfuerzos estaban expuestos a errores sutiles de programación cuyos efectos podría ser que sólo se manifestasen si se producían ciertas secuencias de acciones relativamente raras. Estos errores eran muy difíciles de diagnosticar, porque era necesario poder distinguirlos de los errores del software de aplicación y de los del hardware. Aún cuando se detectase el error, resultaba muy difícil determinar las causas, porque las condiciones precisas bajo las que aparecían los errores resultaban muy difíciles de reproducir. En líneas generales, había cuatro causas principales de error [DENN80a]:

• *Sincronización incorrecta:* Es frecuente el caso en el que una rutina debe ser suspendida a la espera de un suceso en cualquier lugar del sistema. Por ejemplo, un programa inicia una lectura de E/S y debe esperar hasta que los datos estén disponibles en un buffer antes de continuar. En tales casos se requiere alguna señal proveniente de alguna otra rutina. Un diseño incorrecto del mecanismo de señalización puede dar como resultado la pérdida de señales o la recepción de señales duplicadas.

• *Fallos de exclusión mutua:* Es habitual el caso en que más de un usuario o programa intentan a la vez hacer uso de un recurso compartido. Por ejemplo, en un sistema de reservas de líneas aéreas, dos usuarios pueden intentar leer de la base de datos y, si hay un asiento disponible, actualizar la base de datos para hacer una reserva. Si no se controlan estos accesos, puede producirse un error. Debe existir algún tipo de mecanismo de exclusión mutua que permita que sólo una rutina a la vez pueda realizar una transacción sobre una determinada parte de los datos. Verificar la corrección de la implementación de dicha exclusión mutua bajo todas las secuencias posibles de sucesos es difícil.

• *Funcionamiento no determinista del programa:* Los resultados de un programa en particular deben depender normalmente sólo de la entrada del programa y no de las actividades de otros programas en un sistema compartido. Pero cuando los programas comparten memoria y sus ejecuciones son intercaladas por el procesador, entonces pueden interferir con otros, sobreescribiendo zonas comunes de memoria de forma incierta. Así pues, el orden en que se organiza la ejecución de varios programas puede influir en los resultados de un programa en particular.

• *Interbloqueos:* Es posible que dos o más programas estén suspendidos a la espera uno del otro. Por ejemplo, dos programas pueden requerir dos dispositivos de E/S para llevar a cabo cierta operación (por ejemplo, copiar de disco a cinta). Uno de los programas ha tomado el control de uno de los dispositivos y el otro programa tiene el control del otro dispositivo. Cada uno está esperando que el otro libere el recurso deseado. Dicho *interbloqueo* puede depender del ritmo imprevisto de la asignación y la liberación de recursos.

Lo que se necesita para enfrentarse a estos problemas es una forma sistemática de supervisar y controlar los distintos programas que pueden estar ejecutándose en el procesador. El concepto de proceso pone las bases. Se puede considerar que un proceso está formado por las tres componentes siguientes:

- Un programa ejecutable
- Los datos asociados necesarios para el programa (variables, espacio de trabajo, buffers, etc.)
- El contexto de ejecución del programa

Este último elemento es esencial. El contexto de ejecución incluye toda la información que el sistema operativo necesita para administrar el proceso y que el procesador necesita para ejecutar correctamente el proceso. Así pues, el contexto incluye los contenidos de varios registros del procesador, tales como el contador de programa y los registros de datos. También incluye información de utilidad para el sistema operativo, tal como la prioridad del proceso y si el proceso está esperando la terminación de un suceso particular de E/S.

La figura 2.10 indica una forma en la que pueden implementarse los procesos. Hay dos procesos, A y B, en secciones de la memoria principal. Esto es, a cada proceso se le debe asignar un bloque de memoria que contiene los programas, los datos y la información del contexto. Cada proceso es registrado en una lista de procesos construida y guardada por el sistema operativo. La lista de procesos contiene una entrada para cada proceso, la cual dispone de un puntero a la posición del bloque de memoria que contiene al proceso. Esta entrada también puede incluir parte o todo el contexto de ejecución del proceso. El resto del contexto de ejecución del proceso es almacenado en el mismo proceso. El registro de índice del proceso contiene el índice, dentro de la lista de procesos, del proceso que está actualmente controlando al procesador. El contador de programa apunta a la próxima instrucción del proceso que se ejecutará. Los registros de base y de límite definen la región de memoria ocupada por el proceso. El contador de programa y todas las referencias a datos se interpretan como relativas al registro de base y no deben exceder el valor del registro de límite. Esto impide las interferencias entre procesos.

En la figura 2.10, el registro de índice del proceso indica que el proceso B está ejecutándose. El proceso A estaba ejecutándose con anterioridad, pero ha sido interrumpido temporalmente. El contenido de todos los registros en el momento de la interrupción de A fue registrado en su contexto de ejecución. Más tarde, el procesador podrá llevar a cabo un cambio de contexto y reanudar la ejecución del proceso A. Cuando el contador de programa se cargue con un valor que apunte a la zona de programa de A, el proceso A reanudará automáticamente su ejecución.

De esta manera, el proceso es tratado como una estructura de datos. Un proceso puede estar ejecutándose o esperando su ejecución. El "estado" entero del proceso está contenido en su contexto. Esta estructura permite el desarrollo de técnicas potentes que aseguran la coordinación y la cooperación entre procesos. Se pueden diseñar e incorporar nuevas características al sistema operativo (por ejemplo, prioridades) mediante la ampliación del contexto para incluir cualquier nueva información que sea necesaria para dar soporte al nuevo atributo. A lo largo de este libro se verán varios ejemplos en los que se emplea esta estructura de proceso para resolver los problemas planteados por la multiprogramación y la compartición de recursos.

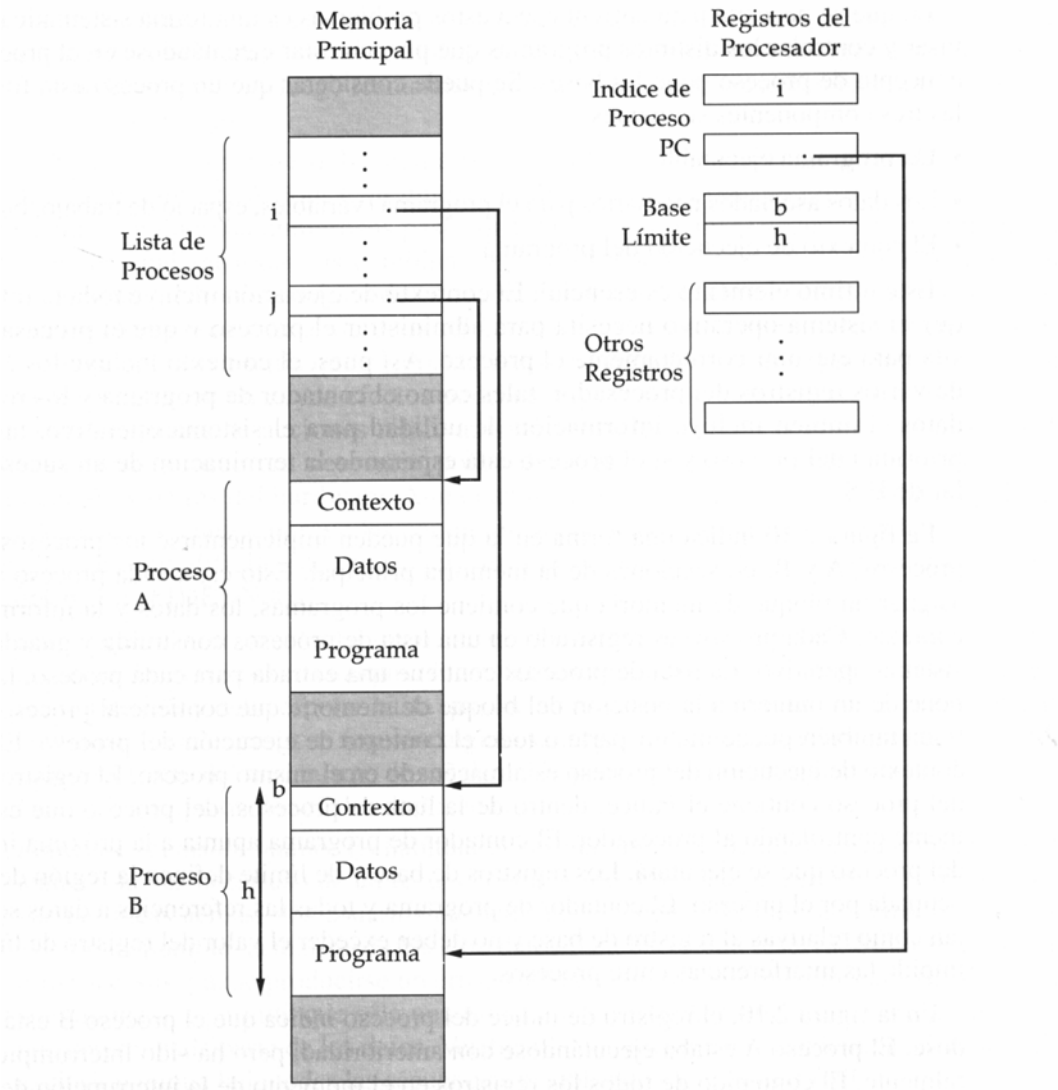


FIGURA 2.10 Implementación típica de los procesos

Los usuarios necesitan un entorno informático que dé soporte a la programación modular y la utilización flexible de los datos. Los administradores de sistemas necesitan un control eficiente y ordenado de la asignación del almacenamiento. Para satisfacer estos requisitos, el sistema operativo tiene cinco responsabilidades principales en la gestión del almacenamiento [DENN71], que son:

• *Aislamiento del proceso:* El sistema operativo debe procurar que cada proceso independiente no interfiera con los datos y la memoria de ningún otro.

• *Asignación y gestión automáticas:* A los programas se les debe asignar memoria dinámicamente en la jerarquía de memoria, según la vayan necesitando. Este proceso debe ser

transparente para el programador. De este modo, el programador se libera de todo lo concerniente a las limitaciones de memoria y el sistema operativo puede lograr eficiencia asignando memoria a los trabajos según la vayan necesitando.

• *Soporte para la programación modular:* Los programadores deben ser capaces de definir módulos de programa y de crear, destruir y alterar el tamaño de los módulos dinámicamente.

• *Protección y control de acceso:* Compartir la memoria en algún nivel de la jerarquía de memoria origina la posibilidad de que un programa pueda direccionar el espacio de memoria de otro programa. Algunas veces, esto es conveniente, sobre todo cuando se necesita compartición en una aplicación en particular. En otros casos, esto amenaza la integridad de los programas y del mismo sistema operativo. El sistema operativo debe permitir que las secciones de memoria estén accesibles de varias maneras para los diversos usuarios.

• *Almacenamiento a largo plaza:* Muchos usuarios y aplicaciones necesitan medios para almacenar información por largos periodos de tiempo.

Normalmente, los sistemas operativos satisfacen estos requisitos mediante la memoria virtual y los servicios del sistema de archivos. La *memoria virtual* es un servicio que permite a los programas direccionar la memoria desde un punto de vista lógico, sin depender del tamaño de la memoria principal física disponible. Cuando se está ejecutando, solo una parte del programa y de los datos pueden estar realmente en memoria principal. Las partes restantes del programa y de los datos se mantienen en bloques en el disco. Se verá en capítulos posteriores cómo esta separación de la memoria en vistas lógicas y físicas ofrece al sistema operativo una potente herramienta para lograr sus objetivos.

El sistema de archivos da cuenta del almacenamiento a largo plazo, almacenándose la información en unos objetos con nombre denominados *archivos.* El archivo es un concepto práctico para el programador y es una unidad útil de control de acceso y de protección en el sistema operativo.

La figura 2.11 ofrece un esquema general de un sistema de almacenamiento administrado por un sistema operativo. El hardware del procesador, junto con el sistema operativo, dotan al usuario de un "procesador virtual" que tiene acceso a la memoria virtual. Este almacén puede ser un espacio lineal de direcciones de memoria o una colección de segmentos, que son bloques de direcciones contiguas con longitud variable. En cualquier caso, las instrucciones del lenguaje de programación pueden hacer referencia a los programas y las posiciones de los datos en la memoria virtual. El aislamiento de los procesos se puede lograr dándole a cada proceso una única memoria virtual que no se solape con otra. La compartición entre los procesos se puede lograr solapando secciones de dos espacios de memoria virtual. Los archivos se mantienen en un almacén permanente. Los archivos o una parte de los mismos pueden copiarse en la memoria virtual para su manipulación por parte de los programas.

El punto de vista que el diseñador tiene del almacenamiento también se ilustra en la figura 2.11. El almacenamiento consta de una memoria principal directamente direccionable (mediante instrucciones de la máquina) y una memoria auxiliar de velocidad inferior a la que se accede indirectamente, cargando los bloques en la memoria principal. Se coloca un hardware de traducción de direcciones *(mapper)* entre el procesador y la memoria. Los programas hacen referencia a las posiciones utilizando direcciones virtuales, que son traducidas a direcciones reales de la memoria principal. Si se hace una referencia a una dirección virtual que no está en la memoria real, entonces una parte del contenido de la memoria real se expulsa hacia la me-

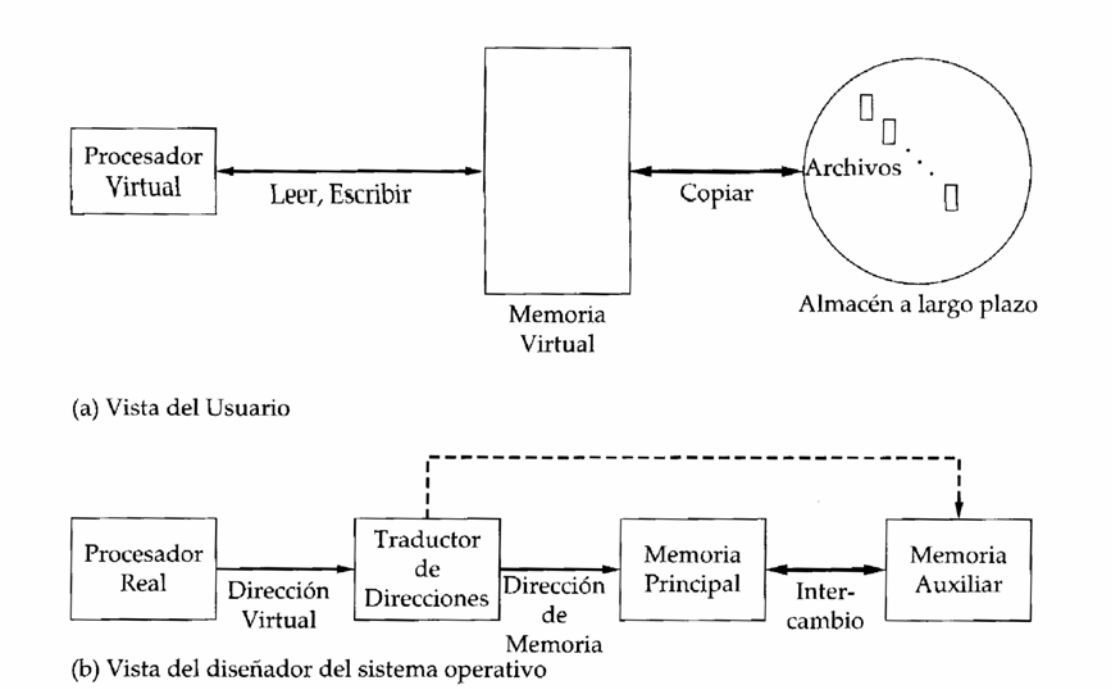


FIGURA 2.11 Dos puntos de vista de un sistema de almacenamiento [DENN80a]

moría auxiliar, intercambiándose con el bloque de memoria deseado. Durante esta actividad, debe suspenderse el proceso que generó la referencia a la dirección. Es tarea del diseñador construir un mecanismo de traducción de direcciones que genere poca sobrecarga y una política de asignación del almacenamiento que minimice el tráfico entre los niveles de memoria.

Seguridad y protección de la información

El crecimiento de la utilización de los sistemas de tiempo compartido y, más recientemente, las redes de computadores, ha traído consigo un aumento de las preocupaciones por la protección de la información.

Una publicación de la Oficina Nacional de Estándares[4] identifica algunas de las amenazas a las que es necesario atender en el campo de la seguridad [BRAN78]:

1. Intentos organizados y deliberados de obtener información económica y mercantil de las organizaciones competitivas del sector privado.
2. Intentos organizados y deliberados de obtener información económica de las oficinas del gobierno.
3. Adquisición inadvertida de información económica o mercantil.
4. Adquisición inadvertida de información sobre las personas.
5. Fraude intencional a través del acceso ilegal a bancos de datos en computadores, con énfasis, en orden decreciente de importancia, en la adquisición de datos financieros, económicos, de aplicación de leyes y personales.

[4] *National Bureau of Standards* (N. del T.)

6. Intromisión del gobierno en los derechos individuales.
7. Atropello de los derechos individuales por la comunidad.

Estos son ejemplos de amenazas específicas que una organización o un individuo (o una organización en nombre de sus empleados) puede sentir la necesidad de contrarrestar. La naturaleza de las amenazas que conciernen a una organización pueden variar enormemente de un conjunto de circunstancias a otro. Sin embargo, pueden construirse algunas herramientas de propósito general dentro de los computadores y de los sistemas operativos para dar soporte a varios mecanismos de protección y seguridad. En general, interesan los problemas de control de acceso a los sistemas informáticos y a la información almacenada en ellos. Se han identificado cuatro clases de políticas generales de protección, en orden creciente de dificultad [DENN80a]:

• *No compartición:* En este caso, los procesos están completamente aislados uno del otro y cada proceso tiene control exclusivo sobre los recursos que le fueron asignados estática o dinámicamente. Con esta política, los procesos suelen compartir los programas o archivos de datos haciendo copias y pasándolas a su propia memoria virtual.

• *Compartiendo los originales de los programas o archivos de datos:* Con el uso de código reentrante (ver Apéndice IB), una única copia física de un programa puede aparecer en varios espacios de memoria virtual como archivos de sólo lectura. Se requieren mecanismos especiales de bloqueo para compartir archivos de datos en los que se puede escribir e impedir que usuarios simultáneos interfieran unos con otros.

• *Subsistemas confinados o sin memoria:* En este caso, los procesos se agrupan en subsistemas para cumplir una política de protección en particular. Por ejemplo, un proceso "cliente" llama a un proceso "servidor" para llevar a cabo cierta tarea con los datos. El servidor se protegerá de que el cliente descubra el algoritmo con el cual lleva a cabo su trabajo y el cliente se protegerá de que el servidor retenga alguna información sobre el trabajo que está llevando a cabo.

• *Diseminación controlada de la información:* En algunos sistemas se definen clases de seguridad para cumplir una determinada política de diseminación de la información. A los usuarios y a las aplicaciones se les dan credenciales de seguridad de un cierto nivel, mientras que a los datos y a otros recursos (por ejemplo, los dispositivos de E/S) se les dota de clasificaciones de seguridad. La política de seguridad hace cumplir las restricciones relativas a qué usuarios tienen acceso a qué clasificaciones. Este modelo es útil no sólo en contextos militares, sino también en aplicaciones comerciales.

Gran parte del trabajo que se ha realizado en la seguridad y protección de los sistemas operativos puede agruparse, en grandes líneas, en las tres categorías siguientes.

• *Control de acceso:* Tiene que ver con la regulación del acceso del usuario al sistema completo, a los subsistemas y a los datos, así como a regular el acceso de los procesos a los recursos y objetos del sistema.

• *Control del flujo de información:* Regula el flujo de datos dentro del sistema y su distribución a los usuarios.

• *Certificación:* Es relativa a la demostración de que el acceso y los mecanismos de control del flujo se llevan a cabo de acuerdo a las especificaciones y a que estas cumplen las políticas de protección y seguridad deseadas.

Planificación y gestión de recursos

Una tarea clave del sistema operativo es administrar los recursos que tiene disponibles (espacio de memoria, dispositivos de E/S, procesadores) y planificar su utilización por parte de los diferentes procesos en activo. Cualquier política de asignación de recursos y de planificación debe tener en cuenta los tres factores siguientes:

• *Equidad:* Normalmente, sería conveniente que a todos los procesos que compiten por el uso de un determinado recurso les sea otorgado un acceso al recurso que sea aproximadamente igualitario y equitativo. Esto es especialmente así para los trabajos de una misma clase, es decir, trabajos con demandas similares, que son cargados con la misma tasa.

• *Sensibilidades diferenciales:* Por otro lado, el sistema operativo puede tener que discriminar entre las diferentes clases de trabajos con diferentes requisitos de servicio. El sistema operativo debe intentar tomar decisiones de asignación y planificación que satisfagan la totalidad de los requisitos. El sistema operativo debe contemplar estas decisiones dinámicamente. Por ejemplo, si un proceso está esperando por el uso de un dispositivo de E/S, el sistema operativo puede querer planificar la ejecución de dicho proceso tan pronto como sea posible y así tener disponible al dispositivo para las demandas de otros procesos.

• *Eficiencia:* Dentro de las restricciones de equidad y eficiencia, el sistema operativo debe intentar maximizar la productividad, minimizar el tiempo de respuesta y, en el caso de tiempo compartido, alojar a tantos usuarios como sea posible.

La tarea de planificación y gestión de recursos es básicamente un problema de investigación operativa, así que se pueden aplicar los resultados matemáticos de esta disciplina. Además, la medición de la actividad del sistema es importante para poder controlar el rendimiento y poder hacer ajustes.

La figura 2.12 propone los elementos principales del sistema operativo que están involucrados en la planificación de procesos y en la asignación de recursos para un entorno de multiprogramación. El sistema operativo mantiene una serie de colas, cada una de las cuales no es más que una lista de procesos esperando a algún recurso. La cola a corto plazo está formada por procesos que están en memoria principal (o que, por *lo* menos, una parte mínima básica está en memoria principal) y están listos para ejecutar. Alguno de estos procesos podría ser el siguiente en usar el procesador. Depende del planificador a corto plazo o distribuidor *(dispatcher)* el escoger a uno. Una estrategia habitual es asignar por turnos una cierta cantidad de tiempo a cada proceso de la cola; esta técnica es conocida como turno rotatorio o *round-robin.* También se pueden utilizar niveles de prioridad.

La cola a largo plazo es una lista de nuevos trabajos que esperan para usar el sistema. El sistema operativo añade los trabajos al sistema transfiriendo un proceso desde la cola a largo plazo hacia la cola a corto plazo. En ese momento, al proceso que se trae se le debe asignar una parte de la memoria principal. De este modo, el sistema operativo debe estar seguro de que no sobrecarga la memoria o el tiempo de procesamiento por admitir demasiados procesos en el sistema. Hay una cola de E/S para cada dispositivo de E/S. Más de un proceso pueden solicitar el uso de un mismo dispositivo de E/S. Todos los procesos esperando por el uso de un determinado dispositivo están alineados en la cola de dicho dispositivo. Una vez más, es el sistema operativo el que debe determinar a qué proceso se le debe asignar un dispositivo de E/S cuando esté disponible.

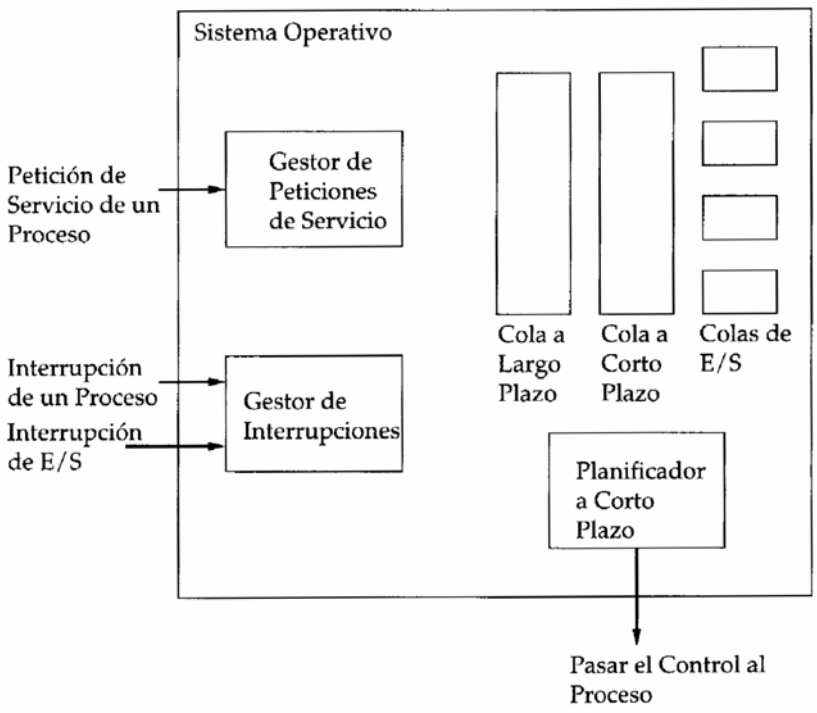


FIGURA 2.12 Elementos clave de un sistema operativo para la multiprogramación

Si se produce una interrupción, el sistema operativo toma el control del procesador mediante la rutina de tratamiento de interrupciones. Un proceso puede invocar especialmente algún servicio del sistema operativo, como un manejador de dispositivos de E/S, por medio de peticiones de servicio. En este caso, el manejador de peticiones de servicio es el punto de entrada al sistema operativo. En cualquier caso, una vez que la interrupción o la petición de servicio es atendida, se invoca al planificador a corto plazo para que seleccione un proceso para su ejecución.

Esta descripción es solamente funcional; los detalles y el diseño modular de esta parte del sistema operativo difieren en los diversos sistemas. En cualquier caso, deben llevarse a cabo estas funciones generales. Gran parte del esfuerzo de investigación y desarrollo en sistemas operativos se ha dedicado a los algoritmos de selección y a las estructuras de datos para que estas funciones brinden equidad, sensibilidades diferenciales y eficiencia.

Estructura del sistema

En la medida en que se añaden más características a los sistemas operativos y en que el hardware se hace más complejo y versátil, el tamaño y la complejidad de los sistemas operativos ha ido creciendo (figura 2.13). El sistema CTSS *(Compatible Time-Sharing System),* puesto en funcionamiento en el MIT en 1963, constaba, como máximo, de aproximadamente 32.000 palabras de 36 bits. El OS/360, presentado posteriormente por IBM, tenía más de un millón de instrucciones de máquina. Hacia 1973, el sistema Multics, desarrollado por el MIT y los Laboratorios Bell, había crecido a más de 20 millones de instrucciones. Es cierto que, más recientemente, se han desarrollado sistemas operativos más simples, para

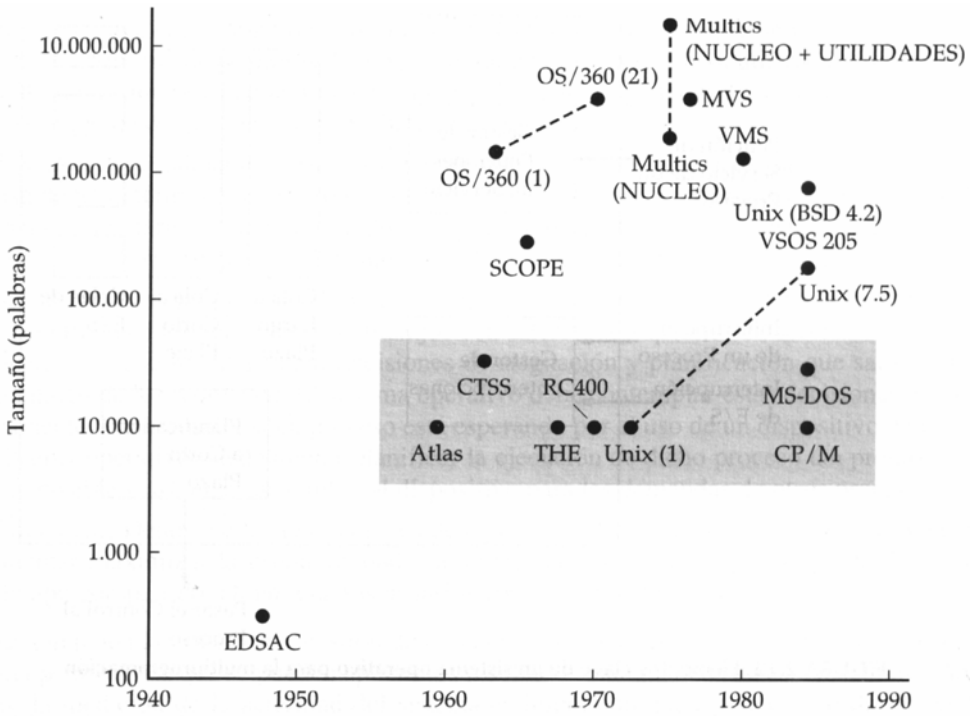


FIGURA 2.13 El tamaño de algunos sistemas operativos (en palabras)

sistemas más pequeños, pero estos han ido creciendo inevitablemente en la medida en que el hardware y los requisitos de los usuarios han crecido. Así pues, UNIX es hoy mucho más complejo que el sistema casi de juguete puesto en marcha por unos pocos programadores con talento en los primeros años de los 70 y el simple PC-DOS ha cedido el paso a la rica y compleja potencia de OS/2.

El tamaño de un sistema operativo completo y la dificultad de las tareas que lleva a cabo plantean tres problemas desafortunados pero demasiado habituales. Primero, los sistemas operativos, cuando se entregan, ya están cronológicamente retrasados. Esto conduce a nuevos sistemas operativos y a actualizaciones de los anteriores. Segundo, los sistemas tienen fallos latentes que se manifiestan en el terreno y que deben ser detectados y corregidos. Y, por último, su rendimiento no es a menudo el que se esperaba.

Para gestionar la complejidad de los sistemas operativos y solucionar estos problemas, se ha prestado mucha atención durante los últimos años a la estructura del software de los sistemas operativos. Ciertos puntos parecen obvios. El software debe ser modular. Esto ayuda a organizar el proceso de desarrollo de software y reduce las tareas de diagnóstico y detección de errores. Los módulos tienen que tener interfaces bien definidas entre sí y estas interfaces deben ser tan simples como sea posible. Esto facilita la labor de programación y también hace más fácil la evolución del sistema. Con interfaces claras y mínimas entre los módulos, se puede cambiar un módulo y que el impacto sobre los otros sea mínimo.

Para grandes sistemas operativos, que van desde cientos de miles a millones de líneas de código, la programación modular por si sola no es suficiente. En su lugar, ha ido creciendo el uso de conceptos como los de niveles jerárquicos y abstracción de la información. La estructura jerárquica de un sistema operativo moderno separa sus funciones de acuerdo a su complejidad, su escala característica de tiempo y su nivel de abstracción. Se puede contemplar al sistema como una serie de niveles. Cada nivel lleva a cabo un determinado subconjunto de funciones requeridas por el sistema operativo. Este se basa en el nivel inferior para llevar a cabo funciones más primitivas y ocultar los detalles de dichas funciones. A su vez, cada nivel ofrece servicios al nivel superior. En el mejor de los casos, los niveles deben estar definidos de forma que los cambios en un nivel no requieran cambios en otros niveles. De este modo, se descompone un problema en un número de subproblemas más manejables.

En general, las capas más bajas trabajan con escalas de tiempo más cortas. Algunas partes del sistema operativo deben interactuar directamente con el hardware del computador, donde los sucesos pueden tener una escala de tiempo tan breve como unas pocas billonésimas de segundo. En el otro extremo del abanico, las partes del sistema operativo que se comunican con el usuario, que envía las órdenes con un ritmo más tranquilo, pueden trabajar en una escala de tiempo de unos pocos segundos. El uso de un conjunto de niveles se adapta bien a este entorno.

La forma en que se aplican estos principios varía enormemente entre los distintos sistemas operativos actuales. Sin embargo, es útil en este punto, con el fin de obtener una visión general de los sistemas operativos, presentar un modelo de sistema operativo jerárquico. Es útil tomar un sistema propuesto por Brown y sus colegas [BROW84] y por Denning y Brown [DENN84], aunque no corresponde a ningún sistema operativo en particular. El modelo está definido en la tabla 2.4 y consta de los siguientes niveles:

- Nivel 1: Consta de circuitos electrónicos, donde los objetos que se tratan son registros, celdas de memoria y puertas lógicas. Las operaciones definidas sobre estos objetos son acciones tales como borrar un registro o leer una posición de memoria.
- Nivel 2: Es el conjunto de instrucciones del procesador. Las operaciones a este nivel son aquellas permitidas por el conjunto de instrucciones del lenguaje de la máquina, tales como SUMAR, RESTAR, CARGAR y DEPOSITAR.
- Nivel 3: Añade el concepto de procedimiento o subrutina, así como las operaciones de llamada y retomo.
- Nivel 4: Introduce las interrupciones, las cuales hacen que el procesador salve el contexto actual e invoque a una rutina de tratamiento de la interrupción.

Estos primeros cuatro niveles no forman parte del sistema operativo, sino que constituyen el hardware del procesador. Sin embargo, algunos de los elementos de los sistemas operativos comienzan a aparecer en estos niveles, tales como las rutinas de tratamiento de interrupción. Es en el nivel 5 en el que comienza a alcanzarse el sistema operativo propiamente dicho y en el que comienzan a aparecer los conceptos asociados con la multiprogramación.

- Nivel 5: En este nivel se introduce la noción de proceso como un programa en ejecución. Entre los requisitos fundamentales de un sistema operativo que ofrezca soporte para múltiples procesos se incluye la capacidad de suspender y reanudar los procesos. Esto exige salvaguardar los registros del hardware, de modo que la ejecución pueda cambiar de un proceso a otro. Además, si los procesos necesitan cooperar, hace falta algún método de sincronización. Una de las técni

TABLA 2.4 Jerarquía de Diseño de un Sistema Operativo

Nivel	Nombre	Objetos	Ejemplos de Operaciones
13	Shell	Entorno de programación del usuario	Sentencias de un lenguaje de shell
12	Procesos de usuario	Procesos de usuario	Salir, eliminar, suspender, reanudar
11	Directorios	Directorios	Crear, destruir, conectar, desconectar, buscar, listar
10	Dispositivos	Dispositivos externos tales como impresoras, pantallas y teclados	Crear, destruir, abrir, cerrar, leer, escribir
9	Sistema de archivos	Archivos	Crear, destruir, abrir, cerrar, leer, escribir
8	Comunicaciones	Tubos (*pipes*)	Crear, destruir, abrir, cerrar, leer, escribir
7	Memoria Virtual	Segmentos, páginas	Leer, escribir, traer (*fetch*)
6	Almacenamiento secundario local	Bloques de datos, canales de dispositivos	Leer, escribir, asignar, liberar
5	Procesos Primitivos	Procesos primitivos, semáforos, colas de procesos listos	Suspender, reanudar, esperar, señalizar
4	Interrupciones	Programas de tratamiento de interrupciones	Invocar, enmascarar, desenmascarar, reintentar
3	Procedimientos	Procedimientos, pila de llamadas, visualización	Marcar la pila, llamar, retornar
2	Conjunto de instrucciones	Evaluación de la pila, intérprete de microprogra-, mas, vectores de datos y escalares	Cargar, almacenar, sumar, restar, bifurcar
1	Circuitos electrónicos	Registros, puertas, *buses*, etc	Borrar, transferir, activar, complementar

cas más simples, pero un concepto importante en el diseño de sistemas operativos, es el semáforo. El semáforo es una técnica sencilla de señalización que se examinará en el capítulo 4,

• *Nivel 6:* Tiene que ver con los dispositivos de almacenamiento secundario del computador. En este nivel se sitúan las funciones de ubicación de las cabezas de lectura y escritura, y se producen las transferencias reales de bloques. El nivel 6 se apoya en el nivel 5 para planificar las operaciones y notificar al proceso que hizo la solicitud que la operación ha culminado. Los niveles más altos se ocupan de las direcciones de los datos pedidos del disco y son los que presentan la solicitud de los bloques apropiados al manejador del dispositivo del nivel 5.

• *Nivel 7:* Crea un espacio de direcciones lógicas para los procesos. Este nivel organiza el espacio de direcciones virtuales en bloques que se pueden mover entre la memoria principal y la memoria secundaria. Tres son los esquemas de uso más habitual: los que utilizan páginas de longitud fija, los que usan segmentos de longitud variable y los que utilizan los dos. Cuando el bloque necesario no está en memoria, la lógica de este nivel le solicita una transferencia al nivel 6.

Hasta este punto, el sistema operativo se ocupa de los recursos de un solo procesador. Empezando por el nivel 8, el sistema operativo trata con objetos externos, tales como dispositivos periféricos y, posiblemente, redes y computadores conectados. Los objetos de estos ni-

veles superiores son lógicos, objetos con nombre que pueden ser compartidos por varios procesos en un mismo computador o en varios computadores.

• *Nivel 8:* Se dedica a la comunicación de información y mensajes entre los procesos. Mientras que el nivel 5 proporciona el mecanismo de señalización primitivo que permite la sincronización entre procesos, este nivel trata con una forma más completa de compartir información. Una de las herramientas más potentes en este nivel es el tubo *(pipe),* que es un canal lógico para el flujo de datos entre los procesos. Un *tubo* se define con su salida en un proceso y su entrada en otro proceso. También se pueden usar para enlazar dispositivos externos o archivos con los procesos. El concepto se discute en el capítulo 4.

• *Nivel 9:* Da soporte al almacenamiento a largo plazo de los archivos con nombre. En este nivel, los datos del almacenamiento secundario se contemplan en términos de entidades abstractas de longitud variable, en contraste con el enfoque orientado al hardware del nivel 6, en términos de pistas, sectores y bloques de tamaño fijo.

• *Nivel 10:* Es el que proporciona acceso a los dispositivos externos mediante interfaces estandarizadas.

• *Nivel 11:* Es responsable de mantener la asociación entre los identificadores externos e internos de los recursos y objetos del sistema. El identificador externo es un nombre que puede ser empleado por una aplicación o un usuario. El identificador interno es una dirección que es utilizada en los niveles inferiores del sistema operativo para ubicar y controlar un objeto. Estas asociaciones se mantienen en un directorio. Las entradas incluyen no sólo las asociaciones externo/interno, sino otras características, como los derechos de acceso.

• *Nivel 12'.* Proporciona servicios completos de soporte a los procesos. Esto va mucho más allá que lo que se ofrece en el nivel 5. En el nivel 5, sólo se mantienen los contenidos de los registros del procesador asociados con un proceso, junto a la lógica para expedir a los procesos. En el nivel 12, se da soporte a toda la información necesaria para la gestión ordenada de los procesos. Esto incluye el espacio de direcciones virtuales del proceso, una lista de objetos y procesos con los que puede interactuar y las limitaciones de dicha interacción, los parámetros pasados al proceso en su creación y cualesquiera otras características del proceso que pudieran ser utilizadas por el sistema operativo para su control.

• *Nivel 13:* Ofrece al usuario una interfaz con el sistema operativo. Se denomina caparazón o *shell* porque separa al usuario de los detalles y le presenta el sistema operativo como un simple conjunto de servicios. El *shell* acepta las órdenes del usuario o las sentencias de control de trabajos, las interpreta, crea y controla los procesos según sea necesario.

Este modelo hipotético de un sistema operativo proporciona una descripción útil de la estructura, a la vez que sirve como guía de implementación. Se puede volver a esta estructura en el transcurso de este libro para observar el contexto de cualquier aspecto particular de diseño que esté en discusión.

2.4 SISTEMAS DE EJEMPLO

Este texto está concebido para dar a conocer los principios de diseño y los aspectos de implementación de los sistemas operativos contemporáneos. De acuerdo con esto, un trata-

miento puramente teórico o conceptual no sería adecuado. Para ilustrar los conceptos y asociarlos con las decisiones reales de diseño que se deben tomar, se han seleccionado tres sistemas operativos actuales:

• *Windows NT:* Un sistema operativo monousuario y multitarea diseñado para que pueda ejecutar sobre una amplia variedad de PC y estaciones de trabajo. Este es, en realidad, uno de los pocos sistemas operativos que han sido diseñados básicamente desde cero. Como tal, está en posición de incorporar de una forma clara los últimos desarrollos en la tecnología de los sistemas operativos.

• *UNIX:* un sistema operativo multíusuario dirigido originalmente a minicomputadores pero implementado en un amplio rango de máquinas, desde potentes minicomputadores hasta supercomputadores.

• *MVS (Múltiple Virtual Storage):* Es el sistema operativo situado en la cima de la línea de grandes sistemas de IBM y uno de los sistemas operativos más complejos que se han desarrollado. Brinda tanto capacidades para el tratamiento por lotes como de tiempo compartido.

Se han escogido estos tres sistemas debido a su relevancia y a su representatitividad. La mayoría de los sistemas operativos de los computadores personales son sistemas monousuario y multitarea, y Windows NT da un buen ejemplo de liderazgo. UNIX ha llegado a ser el sistema operativo dominante en una gran variedad de estaciones de trabajo y sistemas multiusuario. MVS es el sistema operativo de computadores centrales más ampliamente utilizado. Por tanto, la mayoría de los lectores se enfrentarán con alguno de estos sistemas operativos durante el tiempo que empleen este libro o dentro de pocos años.

En esta sección se dará una breve descripción e historia de cada uno de estos sistemas operativos.

WINDOWS NT

Historia

El antepasado más distante de Windows NT es un sistema operativo desarrollado por Microsoft para los primeros computadores personales de IBM y conocido como MS-DOS o PC-DOS. La versión inicial, el DOS 1.0, fue lanzada en Agosto de 1981. Esta constaba de 4000 líneas de código fuente en lenguaje de ensamblador y ejecutaba en 8K de memoria utilizando el microprocesador Intel 8086.

Cuando IBM desarrolló su computador personal basada en disco duro, el PC XT, Microsoft desarrolló el DOS 2.0, lanzado en 1983. Este tenía soporte para un disco duro y ofrecía directorios jerárquicos. Hasta ese momento, un disco podía contener sólo un directorio de archivos y dar soporte a un máximo de 64 archivos. Aunque esto era adecuado en la era de los discos flexibles, era demasiado limitado para un disco duro y la restricción de un único directorio era demasiado molesta. La nueva versión permitía que los directorios pudieran tener subdirectorios, así como archivos. También contenía un conjunto más completo de órdenes incluidas en el sistema operativo, que brindaban funciones que en la versión 1 tenían que llevarse a cabo con programas externos. Entre las capacidades que se le añadieron estaban algunas características del tipo de las de UNIX, tales como el redireccionamiento de E/S, que es la capacidad de cambiar la identidad de la entrada o la salida de una aplicación y la impresión de fondo *(background).* La parte residente en memoria creció hasta 24KB.

Cuando IBM anunció el PC AT en 1984, Microsoft introdujo el DOS 3.0. El AT incorporaba el procesador Intel 80286, que estaba provisto con un direccionamiento ampliado y con recursos de protección de memoria. Estos no fueron utilizados por el DOS. Para mantenerse compatible con las versiones anteriores, el sistema operativo utilizaba el 80286 simplemente como un "8086 rápido". El sistema operativo no ofrecía ningún soporte para los nuevos teclados y discos duros. Aún así, los requisitos de memoria aumentaron a 36KB. Hubo varias actualizaciones notables en la versión 3.0. El DOS 3.1, lanzado en 1984, tenía ya soporte para redes de PC. El tamaño de la parte residente no cambió; esto se consiguió aumentando el volumen del sistema operativo que podía intercambiarse. El DOS 3.3, lanzado en 1987, ya daba soporte para la nueva línea de máquinas IBM, los PS/2. Una vez más, esta versión no sacaba ventajas de las capacidades del procesador del PS/2, provisto con el 80286 y el 80386, de 32 bits. La parte residente del sistema había crecido hasta un mínimo de 46KB, necesitando más cantidad si se seleccionaban ciertas opciones adicionales.

En este tiempo, el DOS estaba siendo utilizado en un entorno que iba más allá de sus capacidades. La introducción del 80486 y del Pentium de Intel introdujeron potencia y características que no pueden ser explotadas por el ingenuo DOS. Mientras tanto, a comienzos de los 80, Microsoft había comenzado a desarrollar una interfaz gráfica de usuario (GUI, *Graphical User Interface)* que podía colocarse entre el usuario y el DOS. La intención de Microsoft era competir con los Macintosh, cuyo sistema operativo era insuperable en su facilidad de uso. Hacia 1990, Microsoft tenía una versión de GUI, conocida como Windows 3.0, que se parecía a la interfaz de usuario del Macintosh. Sin embargo, éste continuaba maniatado por la necesidad de ejecutar encima del DOS.

Después de una tentativa frustrada por parte de Microsoft de desarrollar conjuntamente con IBM un sistema operativo de nueva generación[5], que aprovecharía la potencia de los nuevos microprocesadores y que incorporaría las características de facilidad de uso de Windows, Microsoft continuó por su cuenta y desarrolló Windows NT. Windows NT puede aparecer ante los usuarios como si fuera Windows 3.1, pero está basado en un concepto radicalmente diferente. Windows NT aprovecha la potencia de los microprocesadores actuales y ofrece una multitarea completa en un entorno monousuario.

Multitarea monousuario

Windows NT es, quizá, el ejemplo más importante de lo que se ha convertido en la nueva ola de los sistemas operativos de computadores personales (otros ejemplos son OS/2 y el Sistema 7 de Macintosh). Windows NT se guió por la necesidad de aprovechar la tremenda potencia de los microprocesadores de 32 bits de hoy en día, los cuales rivalizan con las grandes computadores y las minicomputadores de hace unos pocos años, tanto en velocidad como en sofisticación del hardware y en capacidad de la memoria.

Una de las características más significativas de estos nuevos sistemas operativos es que, aunque siguen estando orientados a dar soporte a un sólo usuario interactivo, son sistemas operativos multitarea. Dos desarrollos principales han disparado la necesidad de la multitarea en los computadores personales. Primero, con el aumento de la velocidad y de la capacidad de memoria de los microprocesadores, junto con el apoyo de la memoria virtual, las

[5]IBM continuó desarrollando el sistema OS/2 por su cuenta. Al igual que Windows NT, OS/2 es un sistema operativo monousuario, multitarea y multihilo.

aplicaciones se han hecho más complejas e interrelacionadas. Un usuario puede querer utilizar un procesador de textos, un programa de dibujo y una hoja de cálculo simultáneamente en una misma aplicación para generar un documento. Sin la multitarea, si el usuario desea crear un dibujo e incluir éste dentro de un documento creado con un procesador de textos, debería seguir los pasos siguientes:

1. Abrir el programa de dibujo
2. Crear el dibujo y salvar éste temporalmente en un archivo o en un portapapeles temporal.
3. Cerrar el programa de dibujo.
4. Abrir el programa de proceso de textos.
5. Insertar el dibujo en el lugar adecuado.

Si se desean hacer cambios, el usuario debe cerrar el procesador de textos, abrir el programa de dibujo, editar la imagen gráfica, salvarla, cerrar el programa de dibujo, abrir de nuevo el procesador de textos e insertar la imagen actualizada. Esto se convierte en algo tedioso de inmediato. En la medida en que los servicios y las capacidades disponibles para los usuarios se hacen más potentes y variadas, el entorno monotarea se hace más molesto y poco amistoso. En un entorno de multitarea, el usuario abre cada aplicación según lo necesita y *la deja abierta.* La información se puede mover con facilidad entre las distintas aplicaciones. Cada aplicación tiene una o más ventanas abiertas y una interfaz gráfica con un dispositivo que sirve de apuntador, como puede ser un ratón, que permite al usuario navegar con facilidad en este entorno.

Una segunda motivación para la multitarea es el crecimiento del proceso cliente/servidor. Con el proceso cliente/servidor, un computador personal o una estación de trabajo (el cliente) y un sistema anfitrión o *host* (el servidor) se unen para llevar a cabo una aplicación particular. Las dos están conectadas entre sí y a cada una se le asigna la parte del trabajo que esté acorde con sus capacidades. El cliente/servidor se puede lograr en un red local de computadores personales y servidores, o por mediación de un enlace entre un sistema de usuario y un *host* grande, como puede ser un computador central. En una aplicación pueden entrar en juego uno o más computadores personales y uno más dispositivos servidores. Para brindar el nivel de respuesta requerido, el sistema operativo tiene que dar soporte a un hardware sofisticado de comunicaciones en tiempo real y a los protocolos asociados de comunicación y arquitecturas de transferencia de datos, mientras que, a la vez, da soporte a la interacción con el usuario.

Descripción

Muchos elementos influyeron en el diseño de Windows NT. Este ofrece el mismo tipo de GUI que los productos anteriores de Windows, incluyendo el uso de ventanas, menús y la interacción "señalar y seleccionar" *(point and click)*. Su estructura interna está inspirada en el sistema operativo Mach, el cual está basado, a su vez, en UNIX.

La figura 2.14 ilustra la estructura global de Windows NT. Esta estructura tan modular le da a Windows NT una flexibilidad impresionante. NT puede ejecutar sobre varias plataformas de hardware y dar soporte a las aplicaciones escritas para otros sistemas operativos distintos.

Como casi todos los sistemas operativos, NT diferencia el software de aplicación del software del sistema operativo. Este último ejecuta en lo que se denomina *modo privilegiado o modo núcleo.* El software en modo núcleo tiene acceso a los datos del sistema y al hardware.

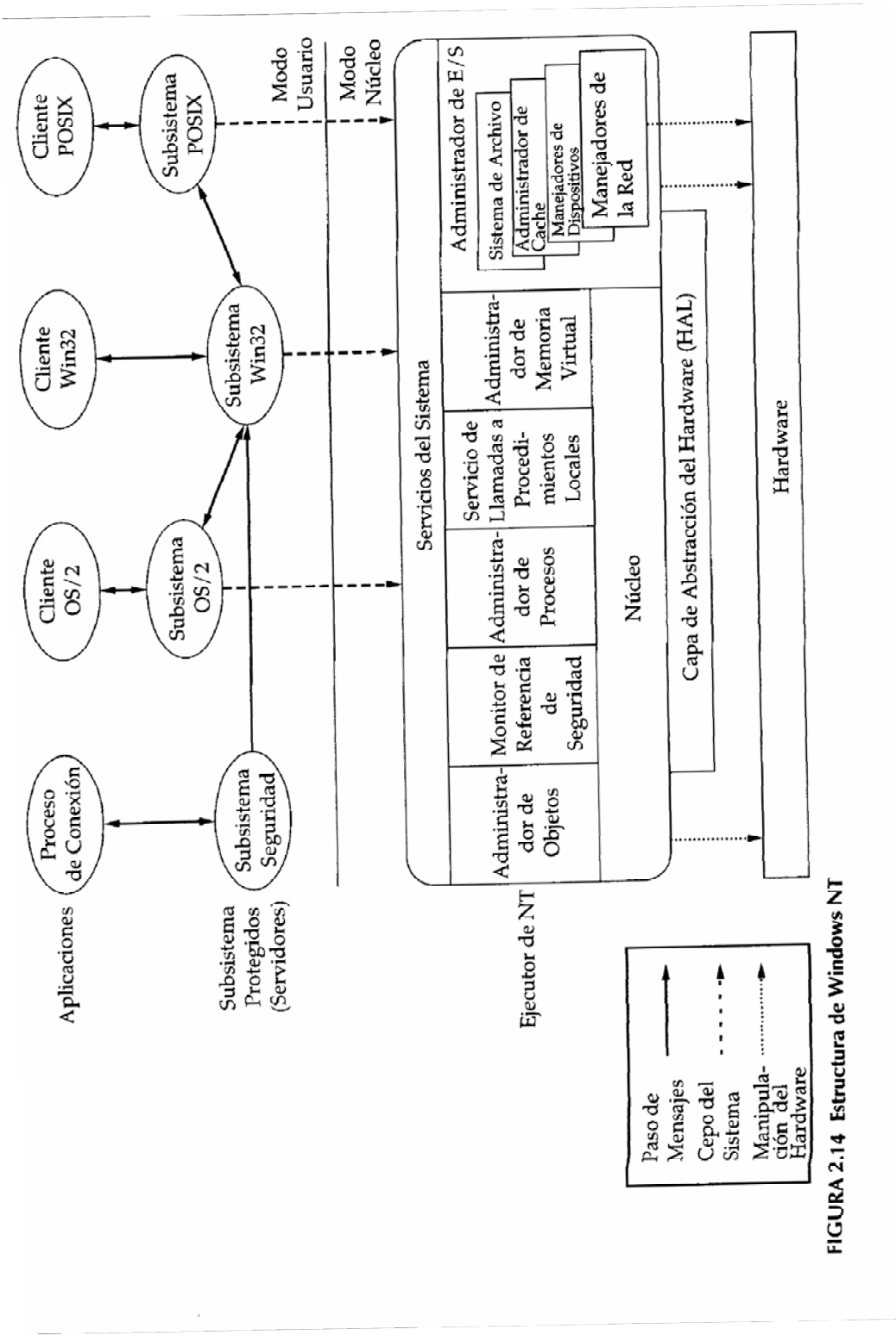


FIGURA 2.14 Estructura de Windows NT

El software restante, que ejecuta en modo usuario, tiene acceso limitado a los datos del sistema. El software en este modo núcleo se denomina ejecutor de NT.

Tanto si está ejecutando sobre un monoprocesador como sobre un multíprocesador, sobre un sistema CISC o uno RISC, la mayor parte de NT tiene la misma visión del hardware subyacente. Para alcanzar esta independencia, el sistema operativo consta de cuatro niveles:

• *Capa de Abstracción de hardware* (HAL, *Hardware Abstractíon Layer):* Establece una correspondencia entre las órdenes y respuestas genéricas del hardware y aquellas que son propias de una plataforma específica, como un Intel 486 o un Pentium, un Power PC de Motorola o un procesador Alpha de Digital Equipment Corporation (DEC). La HAL hace que el bus del sistema de cada máquina, el controlador de DMA, el control ador de interrupciones, los relojes de sistema y el módulo de memoria parezcan los mismos para el núcleo. También ofrece el soporte necesario para el multíproceso simétrico, que se explicará más adelante.

• *Núcleo:* Consta de las componentes más usadas y fundamentales del sistema operativo. El núcleo administra la planificación y el cambio de contexto, la gestión de excepciones e interrupciones y la sincronización de multiprocesadores.

• *Subsistemas:* Incluyen varios módulos con funciones específicas que hacen uso de los servicios básicos proporcionados por el núcleo.

• *Servicios del sistema:* Ofrece una interfaz al software en modo usuario.

Un subsistema concreto, el administrador de E/S, se salta la HAL para interactuar directamente con el hardware. Esto es necesario para lograr la eficiencia y la productividad requeridas por las operaciones de E/S.

La potencia de Windows NT proviene de su habilidad para dar soporte a aplicaciones escritas para otros sistemas operativos. La forma en que Windows NT da este soporte con un ejecutor único y compacto es a través de los subsistemas protegidos. Los subsistemas protegidos son aquellas partes de NT que interactúan con el usuario final. Los subsistemas protegidos ofrecen al usuario una interfaz gráfica o de líneas de órdenes que definen la apariencia del sistema operativo para un usuario. Además, cada subsistema protegido proporciona la interfaz para los programas de aplicación (API, *Application Programming Interface)* del entorno particular de operación. Esto significa que las aplicaciones creadas para un entorno particular de operación pueden ejecutarse sin modificaciones sobre NT, porque la interfaz del sistema operativo que ven es la misma que para la que fueron escritas. De modo que, por ejemplo, las aplicaciones basadas en OS/2 pueden ejecutarse en NT sin modificaciones. Más aún, puesto que el sistema NT está en sí mismo diseñado para que sea independiente de la plataforma, a través del uso de la HAL, tanto los subsistemas protegidos como las aplicaciones que les dan soporte deben ser relativamente fáciles de transportar desde una plataforma de hardware a otra. En muchos casos, todo lo que se necesita es una recompilación.

La forma en que el ejecutor, los subsistemas protegidos y las aplicaciones se estructuran en NT es por medio del modelo cliente/servidor. La arquitectura cliente/servidor es un modelo de computación para sistemas distribuidos que se discutirá en el capítulo 12. Esta misma arquitectura puede adoptarse para su uso interno en un mismo sistema, como es el caso de NT.

Cada servidor se implementa con uno o más procesos. Cada proceso espera de un cliente la solicitud de un servicio, como por ejemplo, servicios de memoria, de creación de procesos o de planificación del procesador. Un cliente, que puede ser un programa de aplicación u otro módulo del sistema operativo, solicita un servicio enviando un mensaje. El mensaje

es encaminado a través del ejecutor hasta el servidor apropiado. El servidor lleva a cabo las operaciones solicitadas y devuelve los resultados o la información de estado por medio de otro mensaje, que es encaminado a través del ejecutor de regreso al cliente.

[CLJST93] señala las ventajas siguientes en una arquitectura cliente/servidor:

• Simplifica el sistema operativo de base, el ejecutor de NT. Puesto que el ejecutor no ofrece una API, es posible construir varias API sin ningún conflicto o duplicación en el ejecutor. Se pueden añadir fácilmente nuevas API.

• Mejora la fiabilidad. Cada servidor ejecuta un proceso independiente, con su propia partición de memoria, protegido de otros servidores. Más aún, los servidores no pueden acceder directamente al hardware o modificar la memoria en la que se almacena el ejecutor. Un servidor puede fallar sin que se hunda o corrompa el resto del sistema operativo.

• Proporciona una base natural para el proceso distribuido. Normalmente, el proceso distribuido hace uso del modelo cliente/servidor, con llamadas a procedimientos remotos u órdenes remotas que se implementan mediante módulos clientes y servidores distribuidos, y el intercambio de mensajes entre clientes y servidores. En NT, un servidor local puede pasar un mensaje a otro remoto para su procesamiento en nombre de las aplicaciones locales clientes. Los clientes no necesitan saber si una solicitud es atendida por un servidor local o remoto. El hecho de que una solicitud sea atendida de forma local o remota puede cambiar dinámicamente, en función de las condiciones actuales de carga y en los cambios dinámicos de configuración.

Hilos[6]

Un aspecto importante de Windows NT es su soporte de hilos dentro de los procesos. Los hilos incorporan algunas de las funciones asociadas tradicionalmente con los procesos. Se puede hacer la siguiente distinción:

• *Hilo:* Una unidad de trabajo que se puede expedir para ejecución. Es lo que se ejecuta secuencialmente y es interrumpible para que el procesador puede pasar a otro hilo. Desde el punto de vista de la planificación y la expedición, este concepto es equivalente al de proceso en la mayoría de los demás sistemas operativos.

• *Proceso:* Una colección de uno o más hilos y recursos del sistema asociados (tales como memoria, archivos abiertos y dispositivos). Esto se acerca mucho al concepto de programa en ejecución. Dividiendo una aplicación sencilla en varios hilos, el programador tiene un gran control sobre la modularidad de la aplicación y la temporización de los sucesos relativos a la aplicación. Desde el punto de vista del trabajo o la aplicación, este concepto es equivalente al de proceso en la mayoría de los sistemas operativos.

Los hilos y los procesos se examinarán en detalle en el capítulo 3.

Multiproceso simétrico

Hasta hace poco, casi todos los computadores personales y las estaciones de trabajo tenían un único microprocesador de propósito general. A medida que las demandas de rendimiento aumentan y el coste de los microprocesadores continúa bajando, esta imagen cambia. Los fabricantes están introduciendo sistemas con varios microprocesadores. Para alcanzar una

[6] *Threads,* en el original (N. del T.)

máxima eficiencia y fiabilidad, es conveniente un modo de operación conocido como *multiproceso simétrico* (SMP, *Symmetric MultiProcessing)*. Básicamente, con SMP, cualquier hilo o proceso puede ser asignado a cualquier procesador. Esto incluye a los procesos e hilos del sistema operativo.

Uno de los requisitos clave en el diseño de los sistemas operativos contemporáneos es la capacidad de explotar el SMP. [CUT93] enumera las siguientes características de Windows NT que dan soporte al uso de SMP:

• Las rutinas del sistema operativo se pueden ejecutar en cualquier procesador disponible y rutinas diferentes pueden ejecutarse simultáneamente en diferentes procesadores.

• NT permite el uso de varios hilos de ejecución dentro de un solo proceso. En un mismo proceso pueden ejecutarse varios hilos en diferentes procesadores simultáneamente.

• Los procesos servidores pueden utilizar varios hilos para procesar solicitudes de más de un cliente simultáneamente.

• NT brinda los mecanismos oportunos para compartir datos y recursos entre los procesos, así como capacidades flexibles de comunicación entre procesos.

Objetos de Windows NT

Windows NT se sustenta fuertemente en los conceptos del diseño orientado a objetos. Este enfoque facilita la compartición de recursos y datos entre los procesos y la protección de los recursos ante usuarios no autorizados. Entre los conceptos clave empleados por NT están los siguientes:

• *Encapsulamiento:* Un objeto consta de uno o más elementos de datos, llamados *atributos* y uno o más procedimientos que pueden actuar sobre los datos, llamados *servicios.* La única manera de acceder a los datos de un objeto es invocando a uno de los servicios del objeto. Así pues, los datos en el objeto pueden ser protegidos fácilmente ante usos no autorizados e incorrectos (por ejemplo, tratar de ejecutar un elemento de datos no ejecutable).

• *Clases e instancias:* Una clase de objeto es una plantilla que enumera los atributos y los servicios de un objeto y define ciertas características del objeto. El sistema operativo puede crear instancias específicas de una clase de objeto cada vez que lo necesite. Por ejemplo, hay una clase de objeto para procesos simples y un objeto proceso para cada proceso en activo. Este enfoque simplifica la creación y gestión de objetos.

El lector no familiarizado con los conceptos de orientación a objetos debería revisar el Apéndice B al final del libro.

No todas las entidades de Windows NT son objetos. Los objetos se usan en los casos en que los datos se abren para su acceso en modo usuario o cuando el acceso a los datos es compartido o restringido. Entre las entidades representadas por objetos se encuentran los archivos, los procesos, los hilos, los semáforos, los temporizadores y las ventanas. NT crea y administra todos los tipos de objetos de una manera uniforme, a través de un volumen de código del núcleo, conocido como el administrador de objetos. El administrador de objetos es responsable de la creación y destrucción de los objetos a petición de las aplicaciones, así como de conceder el acceso a los servicios y datos de un objeto.

Un objeto (por ejemplo, un proceso) puede hacer referencia a otro (por ejemplo, un archivo) abriendo un descriptor *(handle)* del mismo. Básicamente, un descriptor es un puntero

al objeto referido. En NT, los objetos pueden tener nombre o no. Cuando un proceso crea un objeto sin nombre, el administrador de objetos devuelve un descriptor del objeto, que es la única forma de hacer referencia al objeto. Los objetos con nombre disponen de un nombre que pueden usar otros procesos para obtener un descriptor del objeto. Por ejemplo, si el proceso A desea sincronizarse con el proceso B, podría crear un objeto suceso con nombre y pasarle a B el nombre del suceso. El proceso B podría abrir y usar este objeto suceso. Sin embargo, si A sólo desea utilizar el suceso para sincronizar dos hilos dentro de sí mismo, entonces podría crear un objeto suceso sin nombre, porque no hay necesidad de poder usar el suceso para otros procesos.

Una distinción importante entre los objetos con nombre y sin nombre es que los primeros siempre tienen información de seguridad asociada a ellos, en forma de una señal de acceso *(access token)*. Puede emplearse la información de seguridad para restringir el acceso al objeto. Por ejemplo, un proceso puede crear un semáforo con nombre con la intención de que sólo los procesos que conozcan su nombre sean capaces de abrir y usar el semáforo. La señal de acceso asociada con el objeto semáforo enumerará aquellos procesos (en realidad, los identificadores de los usuarios que son propietarios de los procesos) que pueden acceder al semáforo.

Windows NT no es un sistema operativo completamente orientado a objetos. No está implementado en un lenguaje orientado a objetos. Las estructuras de datos que residen por completo dentro de un mismo componente del ejecutor no se representan como objetos. Además, NT no ofrece soporte para algunas capacidades habituales de los sistemas orientados a objetos, tales como la herencia y el polimorfismo. No obstante, NT ilustra la potencia de la tecnología orientada a objetos y representa la tendencia creciente a usar esta tecnología en el diseño de sistemas operativos.

SISTEMA UNIX, VERSIÓN V

Historia

La historia de UNIX es una historia muchas veces contada que no se va a repetir aquí con mucho detalle. En su lugar, se ofrecerá un breve resumen.

UNIX fue desarrollado inicialmente en los Laboratorios Bell y llegó a ser operativo en una PDP-7 en 1970. Algunas personas involucradas de los Laboratorios Bell también habían participado en el trabajo sobre tiempo compartido que se llevaba a cabo en el proyecto MAC del MIT[7]. Aquel proyecto llevó primero al desarrollo del CTSS y luego al de Multics. Aunque es habitual decir que UNIX es una versión a pequeña escala de Multics, los desarrolladores de UNIX dicen estar más influenciados en realidad por CTSS [RITC78b]. Sin embargo, UNIX incorporó muchas ideas de Multics.

Merece la pena comentar algo sobre Multics. Multics estuvo no sólo años sino décadas adelantado a su tiempo. Incluso a mediados de los 80, casi 20 años después de que llegara a ser operativo, Multics disponía de características de seguridad superiores y una mayor sofisticación en la interfáz de usuario y en otros campos que los sistemas operativos contemporáneos de los grandes computadores que puedan compararse. Aunque los desarrolladores de

[7] Siglas de *Massachusetts instituto ofTechnology* (N. del T.)

UNIX abandonaron el proyecto de Multics porque, según su opinión, éste fue un fracaso, Multics fue cedido después a Honeyweil y llegó a disfrutar de un modesto éxito comercial. Lo que tuvo Honeyweil no lo tuvieron los otros dos sistemas operativos de grandes computadores, uno de los cuales fue comercializado de forma muy agresiva. Multics pudo haber tenido un gran éxito. Sin embargo, Multics permaneció como un producto de Honeyweil con una pequeña, pero fiel, base de clientes, hasta que Honeyweil abandonó el negocio de la informática a finales de los 80.

Mientras tanto, el trabajo sobre UNIX en los Laboratorios Bell y, después, en otros lugares, produjo una serie de versiones de UNIX. El primer hito notable fue llevar el sistema UNIX de la PDP-7 a una PDP-11. Esta fue la primera señal de que UNIX sería un sistema operativo para todos los computadores. El siguiente hito importante fue la reescritura de UNIX en el lenguaje de programación C. Esta fue una estrategia inaudita en aquel tiempo. Era de consenso general que algo tan complejo como un sistema operativo, que tenía que tratar con sucesos de tiempo crítico, tenía que ser escrito exclusivamente en lenguaje ensamblador. La implementación en C demostró las ventajas de usar un lenguaje de alto nivel para la mayor parte del código del sistema, si no todo. Hoy en día, casi todas las implementaciones de UNIX están escritas en C.

Estas primeras versiones de UNIX fueron muy populares en los Laboratorios Bell. En 1974, el sistema UNIX fue descrito por primera vez en una revista técnica [RITC74]. Esto despertó gran interés en el sistema. Se otorgaron licencias de UNIX a instituciones comerciales y universidades. La primera versión ampliamente disponible fuera de los Laboratorios Bell fue la Versión 6, en 1976. La siguiente, la versión 7, lanzada en 1978, es el antepasado de la mayoría de los sistemas UNIX modernos. El más importante de los sistemas no desarrollados por AT&T[8] fue el realizado en la Universidad de California en Berkeley. Se le llamó UNIX BSD y ejecutaba primero en una PDP y, más tarde, en máquinas VAX. AT&T continuó desarrollando y refinando el sistema. Hacia 1982, los Laboratorios Bell habían combinado varias variantes del UNIX de AT&T en un único sistema, que fue comercializado como Sistema ÜNÍX, versión *III.* Posteriormente se le añadió un cierto número de elementos hasta llegar al Sistema UNIX, versión V.

Este libro utiliza el Sistema UNIX, versión V como ejemplo de UNIX. En el momento de escribir este libro, parece que ésta llegará a ser la versión comercial dominante de UNIX. Además, incorpora la mayoría de las características importantes desarrolladas en cualquier sistema UNIX y lo hace de forma integrada y factible comercialmente. El Sistema V funciona actualmente en máquinas que van desde microprocesadores de 32 bits hasta supercomputadores y es, sin duda, uno de los sistemas operativos más importantes que se han desarrollado.

Descripción

La figura 2.15 ofrece una descripción general de la arquitectura de UNIX. El hardware básico está rodeado por el software del sistema operativo. El sistema operativo se llama a menudo *núcleo del sistema* o, simplemente, núcleo *(kernel),* para realzar su aislamiento de las aplicaciones y de los usuarios. Esta parte de UNIX será de interés para su utilización como

[7]Siglas de *American Telegraph ana Telephone* (N. del T.)

ejemplo en este libro. Sin embargo, UNIX viene equipado con una serie de servicios de usuario e interfaces que se consideran parte del sistema. Estos pueden agruparse en un *shell,* algún otro software de interfaz y las componentes del compilador de C (compilador, ensamblador, cargador). La capa exterior está formada por las aplicaciones de los usuarios y una interfaz de usuario con el compilador C.

En la figura 2.16 se ofrece una mirada de cerca al núcleo. Los programas de usuario pueden invocar a los servicios del sistema operativo directamente o a través de programas de biblioteca. La interfaz de llamadas al sistema es la frontera con el usuario y le permite al software de alto nivel el acceso a las funciones específicas del núcleo. En el otro extremo, el sistema operativo contiene rutinas primitivas que interactúan directamente con el hardware. Entre estas dos interfaces, el sistema está dividido en dos partes fundamentales, una ocupada del control de los procesos y la otra relacionada con la gestión de archivos y la E/S. El subsistema de control de procesos es el responsable de la gestión de memoria, la planificación y expedición de los procesos y la sincronización y comunicación entre procesos. El sistema de archivos intercambia los datos entre la memoria y los dispositivos externos, tanto en flujos de caracteres como en bloques. Para lograr esto, se utilizan varios manejadores de dispositivo. Para las transferencias de bloques se utiliza un método de cache de disco: Se coloca un buffer del sistema en la memoria principal entre el espacio de direcciones del usuario y el dispositivo externo.

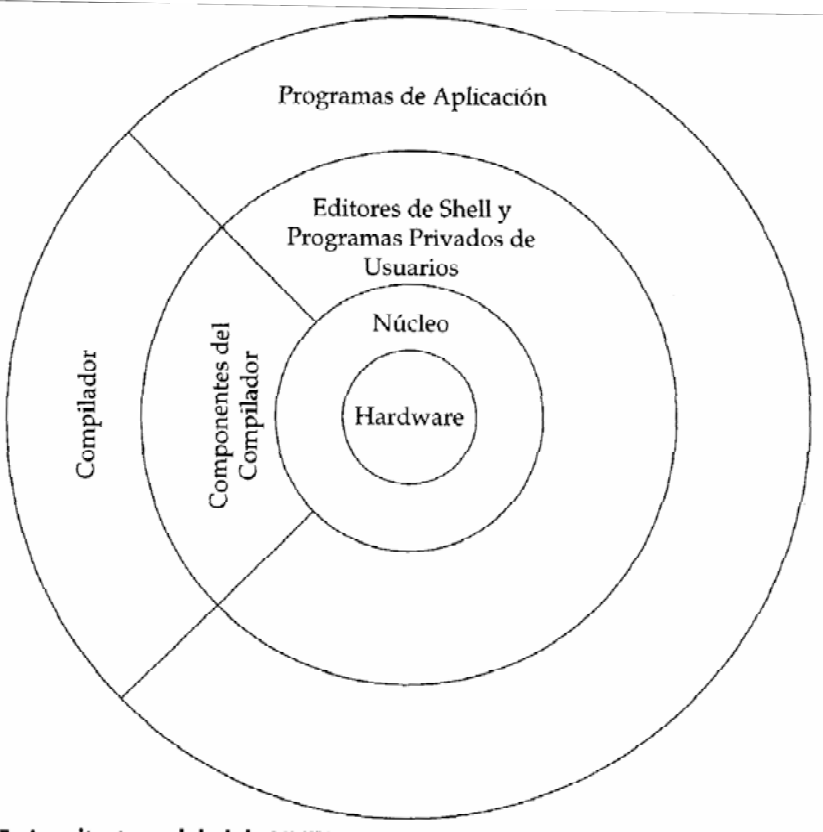


FIGURA 2.15 Arquitectura global de UNIX

MVS

Historia

Hacia 1964, IBM estaba firmemente establecida en el mercado de computadores con sus máquinas de la serie 7000. En aquel año, IBM anunció el Sistema/360, una nueva familia de productos de computadores. Aunque el anuncio en sí no fue una sorpresa, venía con algunas noticias poco agradables para los clientes de IBM: la línea de productos 360 era incompatible con las viejas máquinas de IBM. Por tanto, la transición al 360 sería difícil para la base de usuarios. Esto fue un paso audaz de IBM pero que se creía necesario para romper con algunas de las limitaciones de la arquitectura 7000 y fabricar un sistema capaz de evolucionar con la nueva tecnología de circuitos integrados. La estrategia salió bien tanto financiera como técnicamente. El 360 fue el éxito de la década y asentó IBM como el irresistible ven-

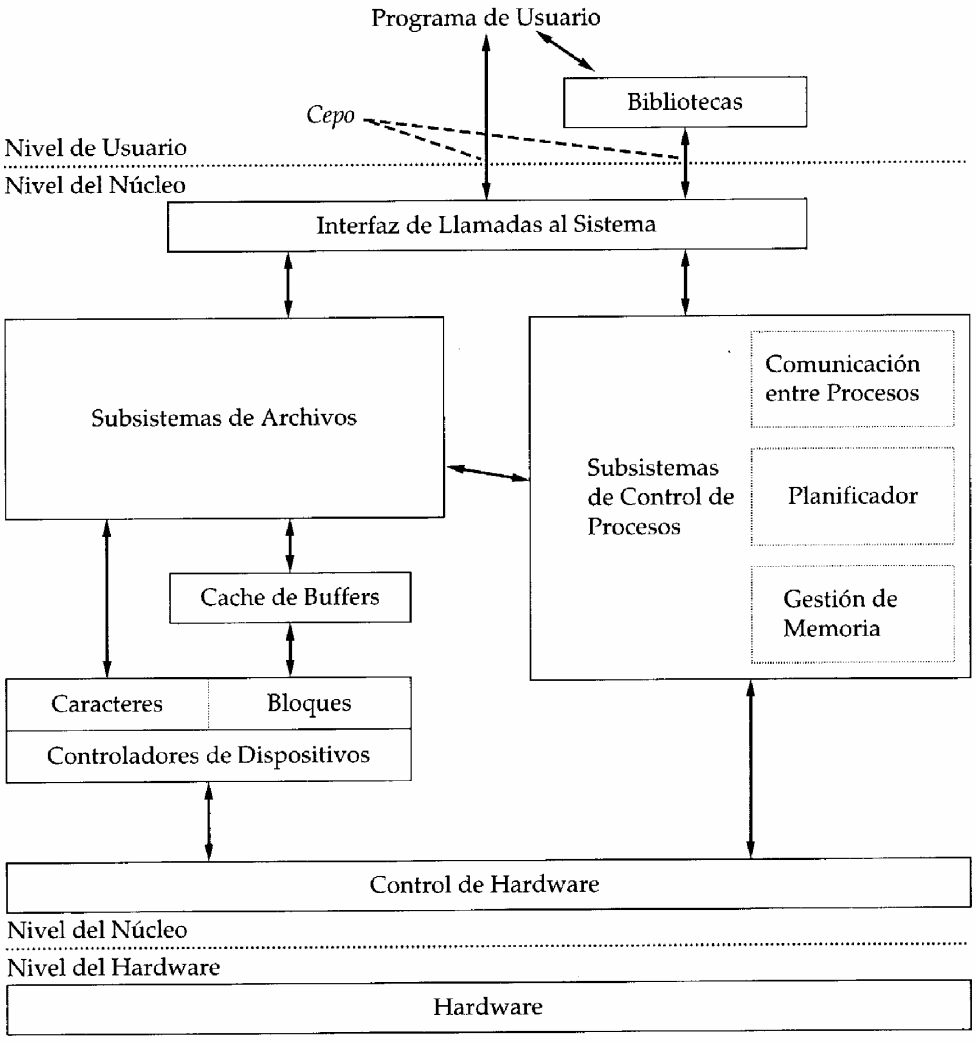


FIGURA 2.16 Diagrama de bloques del núcleo de un sistema UNIX [BACH86]

dedor de computadores con una cuota de mercado superior al 70%. Con algunas modificaciones y ampliaciones, La arquitectura del 360 se ha mantenido hasta el día de hoy, más de un cuarto de siglo después, como la arquitectura de Los grandes computadores IBM.

Aunque el software del sistema operativo de los computadores centrales de IBM ha cambiado más allá de lo que se reconoce como la oferta inicial del 360, sus orígenes se remontan a aquellos primeros tiempos. El Sistema/360 fue la primera familia de computadores planificada por la industria. La familia cubría un amplio rango de rendimiento y coste. Los diferentes modelos eran compatibles en el sentido de que un programa escrito para un modelo era capaz de ejecutarse en otro modelo de la misma familia con diferencias sólo en el tiempo necesario para ejecutar. Por esto, el primer sistema operativo que fue anunciado para las nuevas máquinas, el 05/360, intentaba ser un sistema operativo ünico que pudiera operar con todas las máquinas de la familia. Sin embargo, el amplio abanico de la familia 360 y la diversidad de las necesidades del usuario obligaron a IBM a introducir varios sistemas operativos. Se centrará la atención en la línea de desarrollo que originó el MVS.

El OS/360 original fue un sistema por lotes con multiprogramación y permaneció así por un cierto tiempo. Su versión más completa, el MVT (Multiprogramación con un número Variable de Tareas), fue lanzado en 1969 y fue la más flexible de las variantes del OS/360. La asignación de memoria para un trabajo era variable y no tenla que decidirse hasta la ejecución. Llegó a ser el sistema operativo más popular de las grandes IBM/360 y en Las primeras IBM/370. MVT omitió algunas de las características presentes en las ofertas de sus competidores más avanzados, como la capacidad para dar soporte a multiprocesadores, el almacenamiento virtual y depuración a nivel de fuente, pero brindó un conjunto de servicios y utilidades de apoyo más completo que cualquiera de los sistemas operativos contemporáneos. En campos tales como la planificación de trabajos, el soporte de periféricos, la cantidad de sistemas diferentes y la conversión desde los sistemas anteriores, el 0S/360 fue inigualable.[WEIZ81]

MVT permitía que sólo ejecutaran 15 trabajos concurrentemente. OS/SVS *(Single Virtual Storage,* Almacenamiento Virtual Simple) fue introducido en el año 1972 como un sistema operativo provisional para sacar partido de la arquitectura IBM/370. El añadido más notable fue dar soporte a la memoria virtual. En el SVS se establecía un espacio de direcciones virtual de 16MB. Sin embargo, este espacio de direcciones tenía que ser compartido entre el sistema operativo y todos los trabajos activos. Muy pronto, incluso esta cantidad de memoria se haría inadecuada.

Como respuesta al crecimiento de las necesidades de memoria de los programas de aplicación, IBM introdujo el MVS. Al igual que en el SVS y como dictaba la arquitectura 370, las direcciones virtuales estaban limitadas a 24 bits (de ahí los 16 MB). Sin embargo, con el MVS, el límite es de 16 MB *por trabajo.* Es decir, cada trabajo tiene su propia memoria virtual dedicada de 16 MB. El sistema operativo traduce los bloques de memoria virtual a la memoria real y es capaz de seguir la pista de las memorias virtuales separadas para cada trabajo. En realidad, cada trabajo normalmente dispone de algo menos de la mitad de la memoria virtual asignada; el resto está disponible para el sistema operativo.

El espacio de direcciones de 24 bits, con memoria virtual separada para cada trabajo, en breve llegó a ser poco apropiado para algunos usuarios. IBM amplió su procesador básico para que manejase direcciones de 31 bits, una característica conocida como *direcciones ampliadas* (XA, *extended addressing).* Para sacar partido de este nuevo hardware, en 1983 se introdujo una nueva versión de MVS, conocida como MVS/XA. Con el MVS/XA, el espacio de direcciones por ta-

rea creció a un máximo de 2GB (gigabytes). Esto, se crea o no, aún se considera inapropiado para algunos entornos y aplicaciones. Por consiguiente, en lo que puede representar la última mayor ampliación de la arquitectura 370 (de la 360, en realidad), IBM desarrolló la Arquitectura de Sistemas Empresariales (ESA, *Enterprise System Architecture)* y un sistema operativo mejorado, el MVS/ESA. El mismo espacio de direcciones de 2GB por trabajo que estaba disponible en el MVS/XA está también disponible para programas y datos. Lo nuevo es que hay hasta 15 espacios de direcciones adicionales de 2GB de datos disponibles sólo para un trabajo específico. Por tanto, el espacio máximo direccionable de memoria virtual por trabajo es de 32GB.

Descripción

MVS es, probablemente, el sistema operativo más grande y complejo desarrollado jamás. Requiere un mínimo de 2MB de almacenamiento residente. Una configuración más realista requiere 6MB para el sistema operativo. Los cuatro factores más importantes que han determinado el diseño de MVS son los siguientes:

- Soporte para trabajos interactivos y por lotes
- Almacenamiento virtual de hasta 32GB por trabajo o usuario
- Multiproceso fuertemente acoplado; esta arquitectura, que se examinará en el capítulo 9, consiste en una serie de procesadores que comparten la misma memoria principal.
- Asignación sofisticada de recursos y servicios de supervisión para lograr un uso eficiente de la gran memoria del sistema, múltiples procesadores y estructura compleja de canales de E/S.

La necesidad de tratar con varios procesadores es un requisito que no se plantearon OS/2 ni el Sistema UNIX, versión V. Cuando hay varios procesadores, donde cada uno de los cuales puede ejecutar cualquiera de los procesos y cuando se da soporte a la comunicación entre procesos que ejecutan en diferentes procesadores, la complejidad del sistema operativo puede ser significativamente mayor que en una máquina con un monoprocesador.

La figura 2.17 da una visión simplificada de los bloques principales constituyentes. Un *shell* extemo contiene los servicios y las interfaces visibles para los usuarios y los operadores del sistema responsables de su mantenimiento y ajuste. Además de una colección de programas necesarios para la creación y compilación de programas, hay un subsistema de gestión de trabajos con las siguientes funciones:

- Interpreta las órdenes del operador (desde la consola del operador) y encamina los mensajes adecuados.
- Lee los datos de entrada del trabajo y escribe los datos de salida del trabajo en los dispositivos periféricos.
- Asigna los dispositivos de E/S a un trabajo y notifica al operador de cualquier unidad física de datos (rollo de cinta, paquete de discos) que debe montarse antes de ejecutar el trabajo.
- Convierte cada trabajo en tareas que pueden ser procesadas por el administrador de tareas.

Por último, el *shell* externo incluye un elaborado subsistema de gestión para la recuperación de errores, que asegura que los fallos del trabajo quedan aislados de modo que no dificulten el resto del funcionamiento y que puedan diagnosticarse. Sí es posible, se permitirá a los trabajos afectados por el error que continúen su ejecución.

El núcleo del sistema operativo consta de un conjunto de módulos principales o subsistemas que interactúan uno con otro, el hardware y el *shell* externo. Estos incluyen lo siguiente:

• *Distribuidor:* El distribuidor puede verse como el administrador de los procesadores. Su función es recorrer la cola de tareas listas y planificar la ejecución de una.

• *Tratamiento de interrupciones:* La arquitectura del Sistema/370 da soporte a una amplia variedad de interrupciones. Cualquier interrupción hace que se suspenda el proceso en curso y se pase el control a la rutina de tratamiento apropiada.

• *Gestión de tareas:* Una tarea es, básicamente, un proceso, en el sentido en que se ha usado este término. La gestión de tareas es la responsable de la creación y eliminación de las tareas, el control de sus prioridades, la gestión de las colas de tareas de los recursos reutilizables en serie (por ejemplo, los dispositivos de E/S) y la sincronización de los sucesos,

• *Gestión de pro gramas:* Este módulo es responsable de enlazar los pasos necesarios involucrados en la ejecución de un programa. Este módulo puede ser controlado por órdenes del JCL o en respuesta a las solicitudes de los usuarios para compilar y ejecutar un programa.

• *Gestión del almacenamiento:* Este módulo es el responsable de administrar la memoria real y virtual.

• *Gestión de recursos del sistema:* Este modulo es el responsable de la asignación de los recursos a los espacios de direcciones (procesos).

• *Métodos de acceso:* Un programa de aplicación suele emplear un método de acceso para llevar a cabo una operación de E/S. Un método de acceso es una interfaz entre el programa de aplicación y el supervisor de E/S. Los métodos de acceso liberan al programa de aplicación de la carga de escribir programas para los canales, construyendo los bloques de control requeridos por el supervisor de B/S y manejando las condiciones de terminación.

• *Supervisor de E/S*: El supervisor de E/S lleva a cabo la inicialización y terminación de las operaciones de E/S a nivel del hardware. Genera la instrucción de inicio de E/S, que provoca que el procesador de un canal de E/S ejecute un programa de E/S en la memoria principal y también trata la interrupción que se origina al completarse la operación de E/S.

Merece la pena comentar unas palabras sobre el gestor de recursos del sistema (SRM, *System Resource Manager).* El SRM dota al MVS de un grado de sofisticación único entre los sistemas operativos. Ningún otro sistema operativo de computadores centrales e, incluso, ningún otro sistema operativo, puede igualar las funciones que lleva a cabo el SRM.

El concepto de recurso incluye al procesador, la memoria real y los canales de E/S. Para llevar a cabo la tarea de asignación de recursos, el SRM acumula estadísticas relativas al uso del procesador, los canales y varias estructuras de datos clave. Su propósito es el de ofrecer un rendimiento óptimo, basándose en el análisis y supervisión del rendimiento. Durante la instalación se establecen varios objetivos de rendimiento y estos sirven de gula al SRM, que modifica dinámicamente las características de la instalación y del rendimiento de los trabajos en función de la utilización del sistema. Sucesivamente, el SRM ofrece los informes que capacitan al operador formado para refinar la configuración y los valores de los parámetros y así mejorar el servicio al usuario.

Un ejemplo puede mostrar el sabor de las actividades del SRM. La memoria real se divide en bloques de igual tamaño, denominados *marcos* (encuadres), de los que puede haber varios millares. Cada marco puede albergar un bloque de memoria virtual, que se conoce como *página.* El SRM recibe el control aproximadamente 20 veces por segundo e inspecciona cada uno de los marcos de página. Si la página no ha sido referenciada o cambiada, se incrementa en 1 un contador. Después de un cierto periodo, el SRM promedia estos números para determinar el número medio de segundos que una página permanece sin tocar.

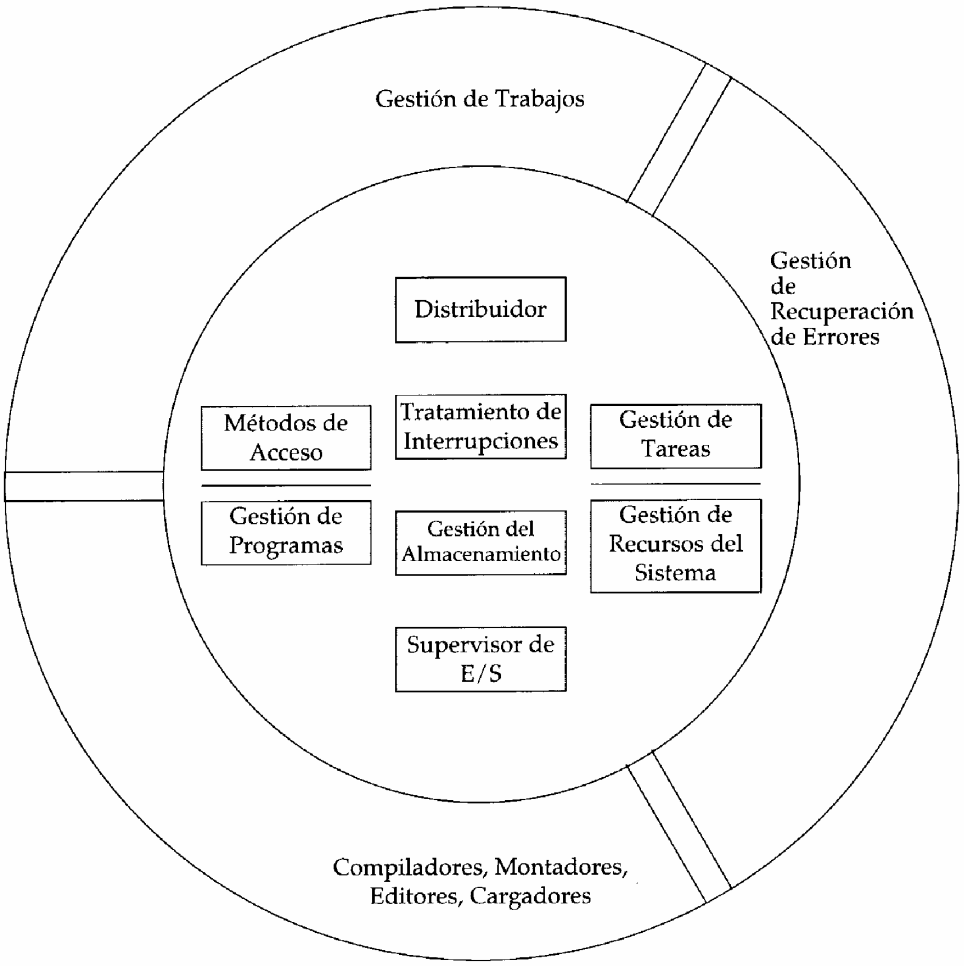


FIGURA 2.17 Visión simplificada de los bloques constituyentes del sistema operativo MVS [PRAS81]

El operador del sistema puede revisar esta cantidad para determinar el nivel de "tensión" del sistema. Reduciendo el número de trabajos activos permitidos en el sistema, este promedio puede mantenerse alto. Una guía útil es que la media debe mantenerse por encima de los 2 minutos para evitar problemas serios de rendimiento [JOHN89], Esto puede parecer demasiado, pero no lo es.

2.5 VISIÓN GENERAL DEL RESTO DEL LIBRO

Los dos primeros capítulos han aportado una panorámica de la arquitectura de los computadores y los sistemas operativos y una introducción al resto del libro. Esta sección ofrece una breve sinopsis de los capítulos restantes.

Descripción y Control de Procesos
El concepto de proceso es central para el estudio de los sistemas operativos y se examina en detalle en el Capitulo 3. Se definen los descriptores de procesos y los descriptores de recursos lógicos. Un examen de los elementos típicos de estos descriptores sienta las bases para una discusión de las funciones relativas a los procesos llevadas a cabo por el sistema operativo. Se describen las primitivas de control de procesos. También se presenta una descripción del estado de los procesos (listo, ejecutando, bloqueado, etc.). El nuevo e importante concepto de hilo también se examina con cierto detalle en este capítulo.

Concurrencia
Los dos temas centrales de los sistemas operativos modernos son la multiprogramación y el proceso distribuido. La concurrencia es fundamental para ambos temas y fundamental para el diseño de la tecnología de los sistemas operativos. Cuando se ejecutan varios procesos concurrentemente, bien sea en el caso real de un sistema multiprocesador o en el caso virtual de un sistema monoprocesador multiprogramado, aparecen cuestiones de resolución de conflictos y de cooperación. El capítulo 4 examina los mecanismos de resolución de conflictos en el contexto de las secciones críticas que deben controlarse en la ejecución de los procesos. Los semáforos y los mensajes son dos técnicas clave empleadas en el control de las secciones críticas, aplicando una disciplina de exclusión mutua. Por ello, el capítulo 5 analiza dos problemas que atormentan todos los esfuerzos de realizar proceso concurrente: el interbloqueo y la inanición.

Gestión de memoria
Los capítulos 6 y 7 tratan de la gestión de la memoria principal. El capítulo 6 brinda una descripción de los objetivos de la gestión de memoria en términos del problema de la superposición y de la necesidad de protección y compartición. El capítulo continúa con una discusión de la carga de programas y del concepto de reubicación. Esto conduce a una discusión sobre la paginación y la segmentación. Se discuten los mecanismos de direccionamiento necesarios para dar soporte a la paginación y la segmentación. Después, el capítulo 7 trata sobre el uso de la paginación o de la paginación y la segmentación, para disponer de memoria virtual. En el capítulo se incluye una discusión sobre la interacción entre el hardware y el software, la memoria principal, la memoria secundaria, la cache, la segmentación y la paginación. El objetivo es mostrar como todos estos objetos y mecanismos pueden integrarse en un esquema global de gestión de memoria.

Planificación
El capítulo 8 comienza con un examen de los tres tipos de planificación del procesador: a largo plazo, a medio plazo y a corto plazo. Las cuestiones de la planificación a largo y medio plazo también se examinan en los capítulos 3 y 5, por lo que el grueso del capítulo se centra en los aspectos de la planificación a corto plazo. Se examinan y comparan los diferentes algoritmos que se han probado. El capítulo 9 trata de los aspectos de planificación relativos específicamente a las configuraciones con multiprocesadores. Por último, el capítulo atiende a las consideraciones de diseño de la planificación en tiempo real.

Gestión de la E/S y planificación de discos
El capítulo 10 comienza con un resumen de los aspectos de E/S de la arquitectura del computador y luego pasa a los requisitos que la E/S impone sobre el sistema operativo. Se analizan y comparan diversas estrategias de amortiguamiento (*buffering)*. Después se exploran algunos aspectos relativos a la E/S con el disco, incluyendo la planificación de discos y el uso de caches de disco.

Gestión de archivos
El capítulo 11 discute la organización física y lógica de los datos. Examina los servicios relativos a la gestión de archivos que un sistema operativo típico proporciona a los usuarios. Después se observan los mecanismos y las estructuras de datos específicas que forman parte de un sistema de gestión de archivos.

Redes y proceso distribuido
Los computadores operan cada vez más no de forma aislada, sino formando parte de una red de computadores y terminales. El capítulo 12 comienza con una revisión del concepto de arquitectura de comunicaciones, poniendo un énfasis especial en el Modelo de Interconexión de Sistemas Abiertos (OSI, *Open Systems Interconection)* y en el TCP/IP. El capítulo estudia el concepto cada vez más importante de proceso cliente/servidor y los requisitos que esta arquitectura impone a los sistemas operativos. El resto el capítulo analiza dos técnicas de comunicación entre procesos que son claves para el proceso distribuido: el paso distribuido de mensajes y las llamadas a procedimientos remotos. Después, el capítulo 13 examina los elementos clave de los sistemas operativos distribuidos, incluyendo la migración de procesos, la determinación de estados globales distribuidos y los mecanismos de exclusión mutua y de detección y prevención del interbloqueo.

Seguridad
El capítulo 14 comienza examinando los tipos de amenaza a los que se enfrentan los computadores y las comunicaciones. El grueso del capítulo trata de las herramientas específicas que pueden usarse para aumentar la seguridad. Lo primero en examinarse son los enfoques tradicionales de la seguridad de los computadores, que se basan en la protección de los distintos recursos del computador y, entre ellos, la memoria y los datos. Se sigue con una discusión de un reciente tipo de amenaza, cada vez más preocupante: el planteado por los virus y mecanismos similares. A continuación, se examina un enfoque relativamente nuevo de la seguridad, los sistemas de confianza. La parte principal del capítulo termina con una mirada a la seguridad en redes. Por último, en un apéndice de este capítulo se presenta el cifrado, que es una herramienta básica empleada en muchas aplicaciones de seguridad.

Análisis de colas
Una herramienta importante que debe estar al alcance de cualquier interesado en la informática es el análisis de colas. Muchas problemas del diseño de los sistemas operativos y del proceso distribuido, así como en muchos otros campos de la informática, pueden representarse por un modelo de colas. El análisis de colas posibilita al analista el rápido desarrollo de una caracterización aproximada del comportamiento de un sistema bajo un régimen de carga. El Apéndice A ofrece una introducción práctica sobre cómo realizar un análisis de colas.

Diseño orientado a objetos

Los conceptos de orientación a objetos están teniendo una importancia creciente en el diseño de los sistemas operativos. Uno de los sistemas de ejemplo de este libro, Windows NT, hace un amplio uso de las técnicas de orientación a objetos. El Apéndice B ofrece una panorámica de los principios esenciales del diseño orientado a objetos.

Orden de los temas

Es normal que el lector se pregunte sobre el orden particular que siguen los temas de este libro. Por ejemplo, el tema de planificación (capítulos 8 y 9) está estrechamente relacionado con los de concurrencia (capítulos 5 y 6) y con el tema general sobre procesos (capítulo 3) y podría abordarse de forma razonable inmediatamente después de dichos temas.

La dificultad estriba en que los distintos temas están muy interrelacionados. Por ejemplo, en la discusión de la memoria virtual, es útil poder hacer referencia a los aspectos de la planificación relativos a los fallos de página. Por supuesto, también es útil poder hacer referencia a los aspectos de gestión de memoria cuando se discuten las decisiones de planificación. Este tipo de ejemplo puede repetirse indefinidamente: Una discusión de la planificación necesita cierta comprensión de la gestión de la E/S y viceversa.

La figura 2.18 propone algunas de las interrelaciones más importantes entre los temas. Las líneas gruesas señalan las relaciones más fuertes, desde el punto de vista de las decisiones de diseño e implementación. Basándose en este diagrama, parece tener sentido empezar con un examen básico de los procesos, que se realizará, de hecho, en el capítulo 3. Después de esto, el orden es, en cierto modo, arbitrario. Muchos estudios de los sistemas operativos cargan todas las tintas al principio sobre los procesos, para luego ocuparse de otros temas. Esto es válido, desde luego. Sin embargo, la significación central de la gestión de memoria, que yo considero de igual importancia que la gestión de procesos, ha llevado hacia la conclusión de presentar este tema antes de abordar una visión en profundidad de la planificación.

La solución ideal para el estudiante es, después de completar sucesivamente los capítulos del 1 al 3, leer y asimilar los siguientes capítulos en paralelo: el 4 seguido (opcionalmente) por el 5; el 6 seguido (opcionalmente) por 7; el 8 seguido (opcionalmente) por el 9; y el 10. Por último, se pueden estudiar los restantes capítulos en cualquier orden: el 11; el 12 seguido por el 13; y el 14. Sin embargo, aunque el cerebro humano puede dedicarse al proceso paralelo; el estudiante puede encontrar imposible (y costoso) trabajar de manera fructífera con cuatro copias de un mismo libro abiertas simultáneamente por cuatro capítulos diferentes. Dada la necesidad de una ordenación lineal, creo que el orden empleado en este libro es el más efectivo.

2.6 LECTURAS RECOMENDADAS

Al igual que en el campo de la arquitectura de los computadores, hay muchos libros sobre los sistemas operativos. [SILB94J], [TANE92], [MILE92] y [DEIT90] cubren los principios básicos usando

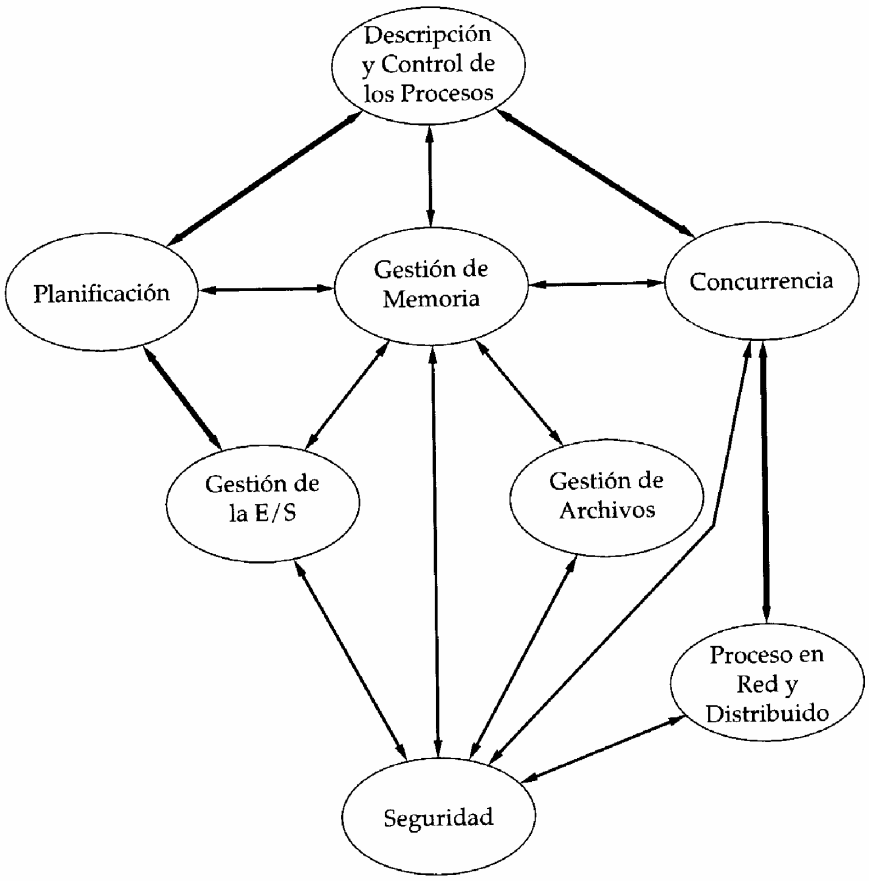


FIGURA 2.18 Temas sobre un Sistema Operativo

un número importante de sistemas operativos como casos de estudio. [SING94a], [NUTT92] y [MAEK97] tratan los temas de los sistemas operativos a un nivel más avanzado.

Ahora se consideran los libros dedicados a los sistemas operativos de ejemplo tomados en esta obra. En el caso de UNIX, a veces parece que hay tantos libros como computadores funcionando con dicho sistema. En el último recuento, sólo Prentice Hall tenía más de 100 libros de UNIX en lista. Sin embargo, casi todos estos libros tratan sobre el uso de UNIX más que sobre sus mecanismos internos y arquitectura. Para el Sistema UNIX, versión V, [BACH96] ofrece un tratamiento definitivo, con amplios detalles técnicos. Una referencia menos detallada y más asequible es [SHAW87]. El estudiante que intente hacer una investigación minuciosa del Sistema V haría bien en leer primero esta última referencia y después atacar la de Bach. Otro libro que cubre las interioridades de UNIX es [ANDL90]; éste proporciona un tratamiento conciso, pero sistemático y exhaustivo, de las características de UNIX y de sus mecanismos. Para el UNIX 4.3BSD de Berkeley, tan popular en la enseñanza, [LEFF881 está muy recomendado. En el número de julio-agosto de 1978 del *Bell System Technical Journal* y en el número de octubre de 1984 del *Bell Laboratorios Technical Journal* de AT&T se publicaron colecciones importantes de artículos sobre UNIX. Éstos se volvieron a editar como [ATT87a y b].

Hasta ahora, no se ha escrito mucho sobre las interioridades de Windows NT. [CUST93] proporciona un tratamiento excelente.

También hay pocos libros sobre las interioridades de MVS. Una excepción destacable es [70HN89]. Este libro está dirigido principalmente a los administradores de sistemas y a los operado- res de los centros de cálculo, explicando MVS desde esta perspectiva. Sin embargo, al hacerlo, aporta una cantidad razonable de detalles técnicos del MVS. El logro principal de esta obra es que realiza un estudio claro y exhaustivo del MVS en un solo libro. Los otros libros tratan el MVS desde el punto de vista del rendimiento: [PAAN86] y [SAMS90].

El lector también puede desear investigar sobre el sistema operativo VMS, que ejecuta en las máquinas VAX de DEC. Este es uno de los sistemas operativos mejor diseñados, documentados y más estudiados. El tratamiento definitivo es el de [KENA91]. Es un modelo de cómo documentar un sistema operativo.

ANDL90 ANDLEIGH, P. *UNIX System Architecture.* Prentice Hall, Englewood Cliffs, NJ, 1990.

ATT87a AT&T *UNIX Sytem V Readings and Examples, Volume I.* Prentice Hall, Englewood Cliffs,NJ,1987.

ATT87b AT&T *UNIX System Reading and Examples, Volume II.* Prentice Hall, Englewood Cliffs,NJ,1987.

BACH86 BACH, M. *The Design ofthe UNIX Operating System.* Prentice Hall, Englewood Cliffs,NJ,1986.

CUST93 CUSTER H. *Inside Windows NT.* Microsoft Press, Redmont, WA, 1993.

DEIT90 DEITEL, H. *An Introduction to Operating System.* Reading, MA: Addison-Wesley, 1990.

JOHN89 JOHNSON, R. *MVS Concepts and Facilities.* McGraw-Hill, Nueva York, 1989.

KENA91 KENAH, L., GOLDENBERG, R. y BATE, S. *VAX/MVS Infernal and Data Structures. D*igital Press, Bedford, MA, 1991.

LEFF88 LEFLER, S., MCKUSICK, M., KARELS, M., QUANTERMAIN, J-, y STETTNER, A. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley, Reading, MA, 1988.

MAEK87 MAEKAWA, M., OLDEHOEFT, A., y OLDEHOEFT, R. *Operating Systems: Advanced Concepts.* Benjamín Cummings, Menlo Park, CA, 1987.

MILE92 MILENKOVIC, M. *Operating Systems: Concepts and Design.* McGraw-Hill, Nueva York, 1992.

NUTT92 NUTT, G. *Centralized and Distributed Operating System.* Prentice Hall, Englewood Cliffs, NJ, 1992.

PAAN86 PAANS, R. A *Cióse Look at MVS Systems: Mechanisms, Performance, and Security.* Elsevier, Nueva York, 1986.

SAMS90 SAMSON, S *MVS Performance Management.* McGraw-Hill, Nueva York, 1990.

SHAW87 SHAW, M. y SHAW, S. *UNIX Internáis: A System Operations Handbook.* Tab Books, Blue Ridge Summit, PA, 1987.

SILB94 SILBERSCHATZ, A. y GALVIN, P. *Operating System Concepts.* Addison-Wesley, Reading, MA, 1994.

SING94a SINGHAL, M. y SHIVARATRI, N. *Advanced Concepts in Operating Systems.* McGraw-Hill, Nueva York, 1994.

TANE92 TANENBAUM, A. *Modern Operating Systems.* Prentice Hall, Englewood Cliffs, NJ, 1992.

2.7

PROBLEMAS

2.1 Supóngase que se tiene un computador multiprogramada en la que cada trabajo tiene características idénticas. En un período de cómputo T, la mitad del tiempo de un trabajo se pasa en E/S y la otra mitad en actividad del procesador. Cada trabajo se ejecuta durante un total de N periodos. Se definen las cantidades siguientes:
- Tiempo de retorno = tiempo real de terminación de un trabajo
- Productividad = número medio de tareas terminadas por periodo de tiempo T
- Utilización del procesador = porcentaje de tiempo en el que el procesador está activo (no esperando)

Calcular estas cantidades para uno, dos y cuatro trabajos simultáneamente, suponiendo que el período T se distribuye de cada una de las formas siguientes:
a) E/S durante la primera mitad, procesador durante la segunda mitad
b) E/S durante el primer y cuarto cuartos, procesador durante el segundo y el tercer cuartos.

2.2 Si se define el tiempo de espera como el tiempo que un trabajo pasa esperando en la cola a corto plazo (figura 2.12), obtener una ecuación que relacione el tiempo de retomo, el tiempo que el procesador está ocupado y el tiempo de espera.

2.3 Un programa con carga de E/S es aquél que, si ejecuta en solitario, gastarla más tiempo esperando en E/S que usando el procesador. Un programa con carga de procesador es el contrario. Supóngase que un algoritmo de planificación a corto plazo favorece a aquellos programas que han empleado poco tiempo del procesador en el pasado reciente. Explicar por qué este algoritmo favorece a los programas con carga de E/S pero sin denegarle permanentemente tiempo del procesador a los programas con carga de procesador.

2.4 Un computador tiene una cache, una memoria principal y un disco utilizado como memoria virtual. Si una palabra está en la cache se requieren A ns para acceder a ella. Si está en la memoria principal pero no en el cache, se requieren B ns para cargarla en la cache y luego la referencia empieza de nuevo. Si la palabra no está en memoria principal, se requieren C ns para traerla del disco, seguidos de B ns para traerla a la cache. Si la tasa de aciertos de la cache es (n-1)/n y la tasa de aciertos de la memoria principal es *(rn-l)/m,* ¿cuál es el tiempo medio de acceso?

2.5 Comparar las políticas de planificación que se podrían usar cuando se intenta optimizar un sistema de tiempo compartido con las mismas políticas que se utilizarían para optimizar un sistema multiprogramado por lotes

CAPITULO 3

Descripción y control de procesos.

El diseño de un sistema operativo debe reflejar con seguridad los requisitos que se pretende que éste cumpla. Todos los sistemas operativos de multiprogramación, desde los sistemas monousuario, como Windows NT, hasta los sistemas de grandes computadores, como MVS, que puede dar soporte a miles de usuarios, están construidos en torno al concepto de proceso. Por tanto, los requisitos principales que debe satisfacer un sistema operativo están expresados haciendo referencia a los procesos:

- El sistema operativo debe intercalar la ejecución de un conjunto de procesos para maximizar la utilización del procesador ofreciendo a la vez un tiempo de respuesta razonable.
- El sistema operativo debe asignar los recursos a los procesos en conformidad con una política especifica (por ejemplo, ciertas funciones o aplicaciones son de prioridad más alta), evitando, al mismo tiempo, el interbloqueo.[1]
- El sistema operativo podría tener que dar soporte a la comunicación entre procesos y la creación de procesos por parte del usuario, labores que pueden ser de ayuda en la estructuración de las aplicaciones.

Puesto que el proceso es fundamental en todos los requisitos clave de los sistemas operativos, se comenzará el estudio detallado de los sistemas operativos con un examen a la forma en que se representan y controlan los procesos en los sistemas operativos. El capitulo se abre con una discusión sobre los estados del proceso, que caracterizan el comportamiento de los mismos. Después, se verán las estructuras de datos que hacen falta para que los sistemas operativos representen el estado de cada proceso, así como otras características de los procesos que son necesarias para que el sistema operativo alcance sus objetivos. A continuación, se descubrirá que el concepto de proceso es más complejo y sutil que el presentado al principio y que, de hecho, incorpora dos conceptos separados independientes en potencia: el relativo a la propiedad de los recursos y el relativo a la ejecución. Esta distinción ha llevado al desarrollo, en algunos sistemas operativos, de una estructura conocida como *hilo (thread)*. Por último, se verán las formas en que los sistemas operativos tomados como ejemplo manejan los conceptos expuestos en este capitulo.

[1] El interbloqueo se examina en el capítulo 5. Básicamente, se produce un interbloqueo si hay dos procesos que necesitan los mismos dos recursos para continuar y cada uno de ellos se ha apropiado de uno de los recursos. Cada proceso espera indefinidamente al recurso que le falta.

3.1 ESTADOS DE UN PROCESO

La misión principal del procesador es ejecutar las instrucciones de la máquina que residen en la memoria principal. Estas instrucciones se dan en forma de programas que contienen secuencias de instrucciones. Como se discutió en el capitulo anterior, por razones de eficiencia y de facilidad de programación, un procesador puede intercalar la ejecución de un conjunto de programas en el tiempo.

De este modo, desde el punto de vista del procesador, éste ejecutará instrucciones de entre un repertorio en una secuencia dictada por los valores cambiante de un registro conocido como el *contador de programa* **(PC,** *Program Counter)* o puntero a las instrucciones. A lo largo del tiempo, este contador puede apuntar a código de programas diferentes que son parte de diferentes aplicaciones. Desde el punto de vista de un programa individual, su ejecución involucra una secuencia de instrucciones del programa. La ejecución de un programa individual se conoce como *proceso* o *tarea.*

El comportamiento de un proceso individual puede caracterizarse por el listado de La secuencia de instrucciones que se ejecutan para dicho proceso. Dicho listado se llama *traza* del proceso. Véase, por ejemplo, el tratamiento riguroso que da Hoare a las trazas [HOAR85]. El comportamiento del procesador puede caracterizarse mostrando la forma en que se intercalan las trazas de varios procesos.

Considérese un ejemplo muy simple. La figura 3.1 muestra La disposición en la memoria de tres procesos. Para simplificar la discusión, se supondrá que no se emplea memoria virtual; de esta manera, los tres procesos están representados por programas que están cargados por completo en la memoria principal. Además, hay un pequeño programa distribuidor que

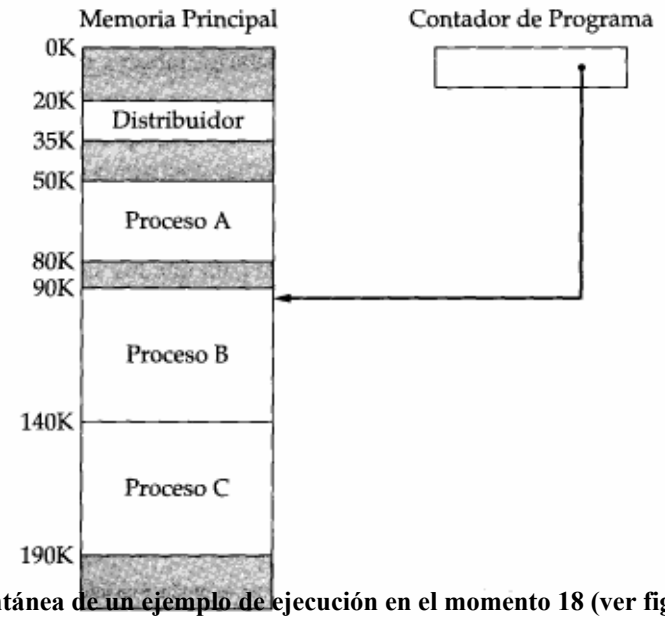


FIGURA 3.1 instantánea de un ejemplo de ejecución en el momento 18 (ver figura 3.3)

asigna el procesador de un proceso a otro. La figura 3.2 muestra las trazas de los tres procesos individuales durante la primera parte de la ejecución. Se muestran las 12 primeras instrucciones ejecutas en los procesos A y C. El proceso B ejecuta 4 instrucciones y se supone que la cuarta instrucción invoca una operación de E/S por la que el proceso debe esperar.

Van a considerarse ahora estas trazas desde el punto de vista del procesador. La figura 3.3 muestra las trazas intercaladas resultantes de los primeros *52* ciclos de instrucción. Se supone que el sistema operativo permite a un proceso continuar su ejecución solo por un máximo de seis ciclos de instrucción, después de los cuales es interrumpido; esto impide que un solo proceso monopolice el tiempo del procesador. Como se ilustra en la figura, se ejecutan las primeras seis instrucciones del proceso A, seguidas del fin del plazo *(time-out)* asignado y la ejecución de cierto código del distribuidor, que devuelve el control al proceso B[2]. Después de ejecutar cuatro instrucciones de este proceso, éste solicita una acción de E/S por la que debe esperar. Por tanto, el procesador detiene la ejecución del proceso B y avanza, por vía del distribuidor, hasta el proceso C. Después de vencer el tiempo, el procesador pasa de nuevo al proceso A. Cuando este proceso consume su tiempo, el proceso B todavía está esperando que termine la operación de E/S, así que el distribuidor avanza de nuevo hasta el proceso C.

Un modelo de procesos con dos estados

La responsabilidad principal del sistema operativo es el control de la ejecución de los procesos; esto incluye la determinación de las pautas de intercalado que se van a seguir y la asignación de recursos a los procesos. Para poder diseñar el sistema operativo de una forma efectiva, se necesita tener un modelo claro del comportamiento de un proceso. El primer paso para diseñar un programa que controle los procesos es describir el comportamiento que se querría que presentaran los procesos.

(a) Traza del Proceso A	(b) Traza del Proceso B	(c) Traza del Proceso C
α + 0	ß + 0	γ + 0
α + 1	ß + 1	γ + 1
α + 2	ß + 2	γ + 2
α + 3	ß + 3	γ + 3
α + 4		γ + 4
α + *5*		γ + 5
α + 6		γ + 6
α + 7		γ + 7
α + 8		γ + 8
α + 9		γ + 9
α + 10		γ + 10
α + 11		γ + 11

α = Dirección inicial del programa del proceso A
ß = Dirección inicial del programa del proceso B
γ = Dirección inicial del programa del proceso C

FIGURA 3.2 Trazas de los procesos de la figura 3.1

[2] El pequeño número de instrucciones ejecutadas por cada proceso y por el distribuidor son ilusoriamente bajas; se hace así para simplificar el ejemplo y aclarar la discusión.

1	$\alpha + 0$	28	$\gamma + 5$
2	$\alpha + 1$	---------- fin de plazo	
3	$\alpha + 2$	29	$\delta + 0$
4	$\alpha + 3$	30	$\delta + 1$
5	$\alpha + 4$	31	$\delta + 2$
6	$\alpha + 5$	32	$\delta + 3$
---------- fin de plazo		33	$\delta + 4$
7	$\delta + 0$	34	$\delta + 5$
8	$\delta + 1$	35	$\alpha + 6$
9	$\delta + 2$	36	$\alpha + 7$
10	$\delta + 3$	37	$\alpha + 8$
11	$\delta + 4$	38	$\alpha + 9$
12	$\delta + 5$	39	$\alpha + 10$
13	$\beta + 0$	40	$\alpha + 11$
14	$\beta + 1$	---------- fin de plazo	
15	$\beta + 2$	41	$\delta + 0$
16	$\beta + 3$	42	$\delta + 1$
---------- Solicitud de E/S		43	$\delta + 2$
17	$\delta + 0$	44	$\delta + 3$
18	$\delta + 1$	45	$\delta + 4$
19	$\delta + 2$	46	$\delta + 5$
20	$\delta + 3$	47	$\gamma + 6$
21	$\delta + 4$	48	$\gamma + 7$
22	$\delta + 5$	49	$\gamma + 8$
23	$\gamma + 0$	50	$\gamma + 9$
24	$\gamma + 1$	51	$\gamma + 10$
25	$\gamma + 2$	52	$\gamma + 11$
26	$\gamma + 3$	---------- fin de plazo	
27	$\gamma + 4$		

δ = Dirección de comienzo del programa distribuidor

FIGURA 3.3 ***Traza combinada de los procesos de la figura 3.1***

El modelo más sencillo que puede construirse tiene en cuenta que, en un momento dado, un proceso puede estar ejecutándose en el procesador o no. Así pues, un proceso puede estar en uno de dos estados: Ejecución o No Ejecución. Esto queda ilustrado en la figura 3.4a. Cuando el sistema operativo crea un nuevo proceso, éste entra en el sistema en estado de No Ejecución. De este modo, el proceso existe, es conocido por el sistema operativo y está esperando la oportunidad de ejecutarse. De cuando en cuando, el proceso que está ejecutando será interrumpido y el programa distribuidor del sistema operativo seleccionará un nuevo proceso para que se ejecute. El proceso anterior pasa del estado de Ejecución al estado de No Ejecución y uno de los demás procesos pasará al estado de Ejecución.

Incluso en este modelo tan simple ya se comienzan a apreciar algunos de los elementos de diseño del sistema operativo. Cada proceso debe representarse de forma que el sistema operativo pueda seguirle la pista. Esto es, debe haber información relativa a cada proceso, incluyendo su estado actual y su posición en memoria. Aquellos procesos que no están ejecutándose tienen que guardarse en algún tipo de cola, para que esperen su tumo de ejecución.

TABLA 3.1 Razones para la Creación de Procesos

Nuevo trabajo por lotes	El sistema operativo esta provisto de un flujo de control de trabajos por lotes, generalmente para cinta o disco. Cuando el sistema operativo se prepara para tomar un nuevo trabajo, leerá a próxima secuencia de Ordenes de control de trabajos.
Conexión interactiva	Un usuario entra en el sistema desde un terminal.
Creado por el SO para dar un servicio	El sistema operativo puede crear un proceso para llevar a cabo una función de parte de un programa usuario, sin que el usuario tenga que esperar (por ejemplo, imprimir).
Generado por un proceso existente	**Con afán de modularidad o para aprovechar el parale-**ismo, un programa de usuario puede ordenar la creación de una serie de procesos.

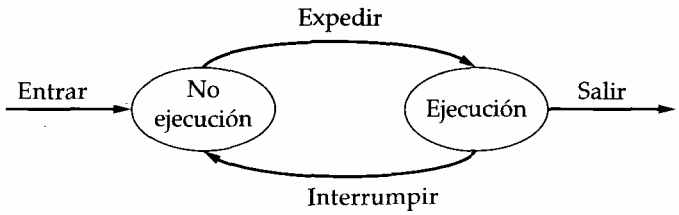
Tradicionalmente, todos los procesos eran creados por el sistema operativo de una forma transparente para el usuario o el programa de aplicación y es así como todavía se mantiene en la mayoría de los sistemas operativos actuales. Sin embargo, puede ser útil permitir que un proceso pueda originar la creación de otro proceso. Por ejemplo, el proceso de una aplicación puede crear otro proceso para recibir los datos que la aplicación esté generando e ir organizando estos datos de una forma conveniente para el análisis posterior. El nuevo proceso ejecuta en paralelo con la aplicación y es activado de cuando en cuando, cada vez que haya nuevos datos disponibles. Este plan puede ser muy útil para estructurar la aplicación. Con otro ejemplo, un proceso servidor (por ejemplo, un servidor de impresión o un servidor de archivos) puede crear un nuevo proceso por cada solicitud que reciba. Cuando un proceso es creado por el sistema operativo tras la solicitud explicita de otro proceso, la acción se conoce como *generación de procesos (process spawning).*

Cuando un proceso genera otro, el proceso generador se conoce como *proceso padre* y el proceso generado es el *proceso hijo.* Normalmente, estos procesos "emparentados" necesitarán comunicarse y cooperar entre si. Lograr esta cooperación es una tarea difícil para el programador; este tema es discutido en el capitulo 4.

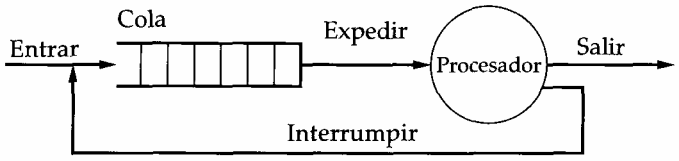
Terminación de procesos

La tabla 3.2, basada en el trabajo de Pinkert y Wear [PINK98], resume las causas típicas de terminación de un proceso.

En cualquier sistema informático, debe haber alguna forma de que un proceso pueda indicar que ha terminado. Un trabaja por lotes debe incluir una instrucción de detención (*Halt*) o una llamada explicita a un servicio del sistema operativo para la terminación. En el primer caso la instrucción *Halt* generará una interrupción para avisar al sistema operativo que el sistema ha concluido. En una aplicación interactiva es la acción del usuario la que indica cuando termina el proceso. Por ejemplo en un sistema de tiempo compartido, el proceso de un usuario particular terminará cuando éste se desconecte del sistema o apague el terminal. En un computador personal o una estación de trabajo, el usuario puede abandonar una aplicación (por ejemplo, el procesador de textos o una hoja de cálculo). Todas estas acciones provocan al final una petición de servicio de sistema operativo para terminar con el proceso demandante.



(a) Diagrama de transición de estados



(b) Diagrama de colas

FIGURA 3.4 Modelo de Proceso de dos estados

La figura 3.4b propone una estructura. Hay una cola sencilla de procesos. Cada entrada de la cola es un puntero a un proceso en particular. Por otra parte, la cola consiste en una lista enlazada de bloques de datos en la que cada bloque representa a un proceso; en la próxima sección se tratará esta última implementación.El comportamiento del distribuidor se puede describir en términos de un diagrama de colas. Cuando un proceso se interrumpe, se le pasa a la cola de procesos en espera. Por otra parte, si un proceso termina o se abandona, se le descarta del sistema (sale del sistema). En cualquier caso, el, distribuidor selecciona entonces un proceso de la cola para ejecutarlo.

Creación y terminación de procesos

Antes de intentar refinar el simple modelo de dos estados, será de utilidad discutir sobre la creación y la terminación de los procesos; en definitiva, independientemente del modelo que se emplee para el comportamiento de los procesos, la vida de un proceso está limitada por su creación y su terminación

Creación de procesos

Cuando se añade un proceso a los que ya está administrando el sistema operativo, hay que construir las estructuras de datos que se utilizan para administrar el proceso (como se describe en la sección 3.2) y asignar el espacio de direcciones que va a utilizar el proceso. Estas acciones constituyen la creación de un nuevo proceso.

Cuatro sucesos comunes llevan a la creación de un proceso, como se indica en la tabla 3.1. Primero, en un entorno de trabajo por lotes, un proceso se crea como respuesta a la remisión de un trabajo. En un entorno interactivo, se crea un proceso cuando un nuevo usuario intenta conectarse. En ambos casos, el sistema operativo es el responsable de la creación del nuevo proceso. Un tercer caso en el que el sistema operativo crea un proceso es de parte de una aplicación. Por ejemplo, si un usuario solicita la impresión de un archivo, el sistema operativo creará un proceso que gestionará dicha impresión. Esto le permitirá al proceso demandante continuar independientemente del tiempo que tarde en terminar la tarea de impresión.

TABLA 3.2 Razones para la Terminación de un Proceso [PINK89]

Terminación normal	El proceso ejecuta una llamada a un servicio del SO que indica que ha terminado de ejecutar.
Tiempo limite excedido	El proceso ha ejecutado por más del tiempo limite total especificado. Hay varias posibilidades para la clase de tiempo que se mide. Entre éstas se incluyen el tiempo total transcurrido ("tiempo de reloj"), el tiempo que ha estado ejecutando y, en el caso de un proceso interactivo, el tiempo transcurrido desde que el usuario real izó su última entrada de datos.
No hay memoria disponible	El proceso necesita más memoria de la que el sistema le puede proporcionar
Violación de límites	El proceso trata de acceder a una posición de memoria a la que no le está permitido acceder.
Error de protección	El proceso intenta utilizar un recurso o un archivo que no le está permitido utilizar, o trata de utilizarlo de forma incorrecta, como escribir en un archivo que es solo de lectura.
Error aritmético	El proceso intenta hacer un cálculo prohibido, como una división por cero, o trata de almacenar un número mayor del que el hardware acepta.
Tiempo máximo de espera rebasado	El proceso ha esperado más allá del tiempo máximo especificado para que se produzca cierto suceso.
Fallo de EJS	Se produce un error en la entrada o la salida, tal como la inca - pacidad de encontrar un archivo, un fallo de lectura o escritura después de un número máximo de intentos (cuando, por ejemplo, hay un región defectuosa en una cinta), o una operación ilegal (como intentar leer de una impresora).
Instrucción inválida	El proceso intenta ejecutar una instrucción inexistente (a me - nudo como resultado de un salto a una zona de datos para intentar ejecutar los datos).
Instrucción privilegiada	El proceso intenta usar una instrucción reservada para el sistema operativo.
Mal uso de los datos	Un elemento de dato es de un tipo equivocado o no está inicializado.
Intervención del operador o del SO	Por alguna razón el operador o el sistema operativo termina con el proceso (por ejemplo, si existe un interbloqueo).
Terminación del padre	Cuando un proceso padre finaliza, el sistema operativo puede diseñarse para terminar automáticamente con todos SUS descendientes.
Solicitud del padre	Un proceso padre tiene normalmente la autoridad de terminar con cualquiera de sus descendientes.

Además, una serie de errores y condiciones de fallo pueden llevarnos a la terminación de un proceso. La tabla 3.2 enumera algunas de las condiciones más habituales.[3]

Por último, en algunos sistemas operativos, un proceso puede ser eliminado por el proceso que lo creó o al terminar el proceso padre.

[3] Un sistema operativo misericordioso puede, en algunos casos, permitir que un usuario se recupere de un fallo sin tener que terminar con el proceso. Por ejemplo, si un usuario solicita el acceso a un archivo y se le niega, el sistema operativo puede informarle simplemente al usuario que el acceso le ha sido denegado y permitir al proceso que continúe.

Un modelo de cinco estados

Si todos los procesos estuvieran siempre listos para ejecutar, entonces la disciplina de cola propuesta en la figura 3.4b seria eficaz. La cola es una lista "primero en entrar, primero en salir" (FIFO, *First-in, First-Out)* y el procesador opera según un turno rotatorio *(round-robin)* con todos los procesos disponibles (a cada proceso de la cola se le otorga una cierta cantidad de tiempo para ejecutar y luego vuelve a la cola, a menos que se bloquee). Sin embargo, *aún* en el simple ejemplo que se ha descrito, esta implementación no es adecuada:

Algunos procesos en el estado de No Ejecución están listos *para* ejecutar, mientras que otros están bloqueados, esperando a que termine *una* operación de E/S. Así pues, utilizando una cola sencilla, el distribuidor podría no seleccionar exactamente el proceso que está en el extremo más antiguo de la cola. Más bien, el distribuidor tendría que recorrer la lista buscando el proceso que no este no bloqueado" y que lleve mas tiempo en la cola.

Una forma más natural de afrontar esta situación es dividir el estado de No Ejecución en dos estados: *Listo y Bloqueado.* Esto se muestra en la figura *3.5.* Por añadidura, se han incorporado dos estados más que se mostrarán de utilidad. Los cinco estados de este nuevo diagrama son los siguientes:

- *Ejecución:* El proceso que está actualmente en ejecución. En este capitulo se suponen computadores con un único procesador, de forma que solo un proceso, a lo sumo, puede estar en este estado en un instante dado.
- *Listo:* Proceso que está preparado para ejecutar, en cuanto se le dé la oportunidad.
- *Bloqueados:* Proceso que no puede ejecutar hasta que se produzca cierto suceso, como la terminación de una operación de E/S.
- *Nuevo:* Proceso que se acaba de crear, pero que aún no ha sido admitido por el sistema operativo en el grupo de procesos ejecutables.
- *Terminado:* Un proceso que ha sido excluido por el sistema operativo del grupo de procesos ejecutables, bien porque se detuvo o porque fue abandonado por alguna razón.

Los estados Nuevo y Terminado son construcciones útiles para la gestión de procesos. El estado Nuevo corresponde a los procesos que acaban de ser definidos. Por ejemplo, si un nuevo usuario intenta conectarse a un sistema de tiempo compartido o si un nuevo trabajo por lotes es remitido para su ejecución, el sistema operativo puede definir un nuevo proceso en dos pasos. Primero, el sistema operativo lleva a cabo *algunas* tareas necesarias de gestión

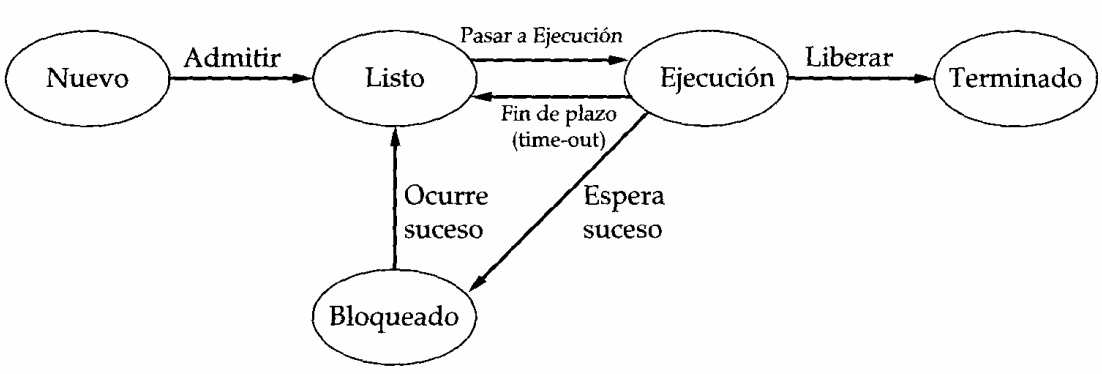


FIGURA 3.5 Modelo de procesos de cinco estados

interna. Se le asocia un identificador al proceso y se construyen y asignan algunas tablas necesarias para gestionar el proceso. En este punto, el proceso estará en el estado Nuevo. Esto significa que el sistema operativo ha llevado a cabo las acciones necesarias para crear el proceso pero no se ha comprometido aún a su ejecución. Por ejemplo, un sistema operativo puede limitar la cantidad de procesos que pueden estar en el sistema por razones de rendimiento o de limitación de memoria.

Asimismo, un proceso sale de un sistema en dos pasos. Primero, el proceso termina cuando liega al punto normal de terminación, cuando se abandona debido a un error irrecuperable o cuando otro proceso con la debida autoridad hace que el proceso abandone. La terminación pasa el proceso al estado Terminado. En este punto, el proceso ya no se elige más para la ejecución. Sin embargo, las tablas y otra información asociada con el trabajo son conservadas temporalmente por el sistema operativo. Esto le da tiempo a otros programas auxiliares o de soporte para extraer la información necesaria. Por ejemplo, puede ser que un programa de contabilidad necesite registrar el tiempo de procesador y otros recursos usados por el proceso con fines de facturación. Un programa de utilidades puede tener que extraer información sobre La historia del proceso con fines relativos a un análisis de uso o de rendimiento. Una vez que estos programas han extraído la información necesaria, el sistema operativo ya no necesita mantener más datos relativos al proceso y éstos se borran del sistema. La tabla 3.3 ofrece más detalles. Se verán cada una de las posibilidades por turnos:

- *Nulo → Nuevo:* Se crea un nuevo proceso para ejecutar un programa. Este suceso se produce por algunas de las razones enumeradas en la tabla 3.1.
- *Nuevo → Listo:* El sistema operativo pasará un proceso del estado Nuevo al estado Listo cuando esté preparado para aceptar un proceso más. La mayoría de los sistemas ponen un límite en función del número de procesos existente o en la cantidad de memoria virtual dedicada a los procesos existentes. El motivo de este límite es asegurar que no haya tantos procesos activos como para degradar el rendimiento.
- *Listo → - Ejecución:* Cuando es hora de seleccionar un nuevo proceso para ejecutar, el sistema operativo elige a uno de los procesos del estado Listo. La cuestión de qué proceso se escoge es estudiada en el capitulo 8.
- *Ejecución → Terminado:* El proceso que se está ejecutando es finalizado por el sistema operativo si indica que terminó 0 Si se abandona.
- *Ejecución → Listo:* La razón más común de esta transición es que el proceso que está en ejecución ha alcanzado el tiempo máximo permitido de ejecución interrumpida; casi todos los sistemas operativos con multiprogramación imponen este tipo de norma de tiempo. Hay otras causas alternativas para esta transición que no están implementadas en todos los sistemas operativos. Por ejemplo, si el sistema operativo asigna diferentes niveles de prioridad a los diferentes procesos. Supóngase que un proceso A está ejecutándose con un cierto nivel de prioridad y que el proceso B, con un nivel de prioridad mayor, está bloqueado. Si el sistema operativo se entera de que el suceso que B estaba esperando se ha producido, pasando así B a! estado Listo, entonces puede interrumpir al proceso A y expedir a B. Por último, otro caso es que un proceso ceda voluntariamente el control del procesador.
- *Ejecución → Bloqueado:* Un proceso se pone en el estado Bloqueado si solicita algo por lo que debe esperar. Las solicitudes al sistema operativo suelen ser en forma de llamadas a ser

TABLA 3.3 Transiciones de Estado de un Proceso

DESDE	HACIA				
	Nuevo	Ejecución	Listo	Bloqueado	Terminado
	El SO responde a una solicitud de control de trabajos; se conecta un usuario de tiempo compartido; Un proceso crea un hijo.	X	X	X	X
Nuevo		X	El SO está preparado para aceptar un proceso más.	X	X
Ejecución	X	X	Se acabó el plazo de tiempo; petición de servicio al SO; el SO lo expulsa a causa de otro de mayor prioridad; el proceso cede el control.	Petición de servicio al SO; petición de recurso; petición de suceso.	El proceso termina; el proceso cancela.
Listo	X	Seleccionado por el distribuidor como el próximo proceso a ejecutar.	X	Proceso finalizado	Proceso finalizado por el padre.
Bloqueado	X	X	Se produce el suceso.	X	Proceso finalizado por el padre.

vicios del sistema, es decir, llamadas desde el programa que está ejecutándose a un procedimiento que forma parte del código del sistema operativo. Por ejemplo, un proceso puede solicitar un servicio que el sistema operativo no está preparado para llevar a cabo de inmediato. Puede pedir un recurso, tal y como un archivo o una sección compartida de memoria virtual, que no esté inmediatamente disponible. O bien el proceso puede iniciar una acción, como una operación de E/S, que debe terminarse antes de que el proceso pueda continuar. Al comunicarse los procesos unos con otros, uno se puede quedar bloqueado cuando espera a que otro proceso le proporcione una cierta entrada o cuando espera un mensaje del otro proceso.

- *Bloqueado → Listo:* Un proceso que está en el estado Bloqueado pasará al estado Listo cuando se produzca el suceso que estaba esperando.
- *Listo → Terminado:* Por razones de claridad, esta transición no se muestra en el diagrama de estados de la figura *3.5*. En algunos sistemas, un padre puede terminar con un proceso hijo en cualquier momento. Además, si el padre termina, todos los procesos hijos asociados con él pueden ser finalizados.
- *Bloqueado → Terminado:* Se aplica el mismo comentario que en el caso anterior.

Volviendo al ejemplo planteado, la tabla 3.4 muestra el movimiento de cada proceso entre los cinco estados. La figura 3.6a sugiere la forma en la que podría implementarse una disciplina de colas. Ahora hay dos colas: Una cola de Listos y una cola de Bloqueados. A medida que se admiten procesos en el sistema, se sitúan en la cola de Listo. Cuando llega la hora de que el sistema operativo escoja otro proceso para ejecutar, selecciona uno de la cola de Listos. En ausencia de un esquema de prioridades, ésta puede ser una simple cola FIFO. Cuando un proceso que se está ejecutando es apartado de la ejecución, o bien se le finaliza o bien se pone en la cola de Listos o de Bloqueados, dependiendo de las circunstancias.

Por último, cuando se produce un suceso, todos los procesos de la cola de Bloqueados que están esperando a dicho suceso se pasan a la cola de Listos.

Esta última medida significa que, cuando se produce un suceso, et sistema operativo debe recorrer toda la cola de Bloqueados, buscando aquellos procesos que esperan al suceso. En un sistema operativo grande, puede haber cientos o incluso miles de procesos en dicha cola. Por tanto, serla más eficiente tener una serie de colas, una para cada suceso. En tal caso, cuando se produzca un suceso, la lista entera de procesos de la cola correspondiente al suceso puede pasarse al estado Listo (figura 3 .6b).

TABLA 3.4 Estados de los procesos para la traza de la figura 3.3

Tiempo	Proceso A	Proceso B	Proceso C
1—6	Ejecución	**Listo**	**Listo**
7—12	**Listo**	**Listo**	**Listo**
13—18	**Listo**	Ejecución	**Listo**
19—24	**Listo**	**Bloqueado**	**Listo**
25—28	**Listo**	**Bloqueado**	Ejecución
29—34	**Listo**	**Bloqueado**	**Listo**
35—40	Ejecución	**Bloqueado**	**Listo**
41—46	**Listo**	**Bloqueado**	**Listo**
47—52	**Listo**	**Bloqueado**	Ejecución

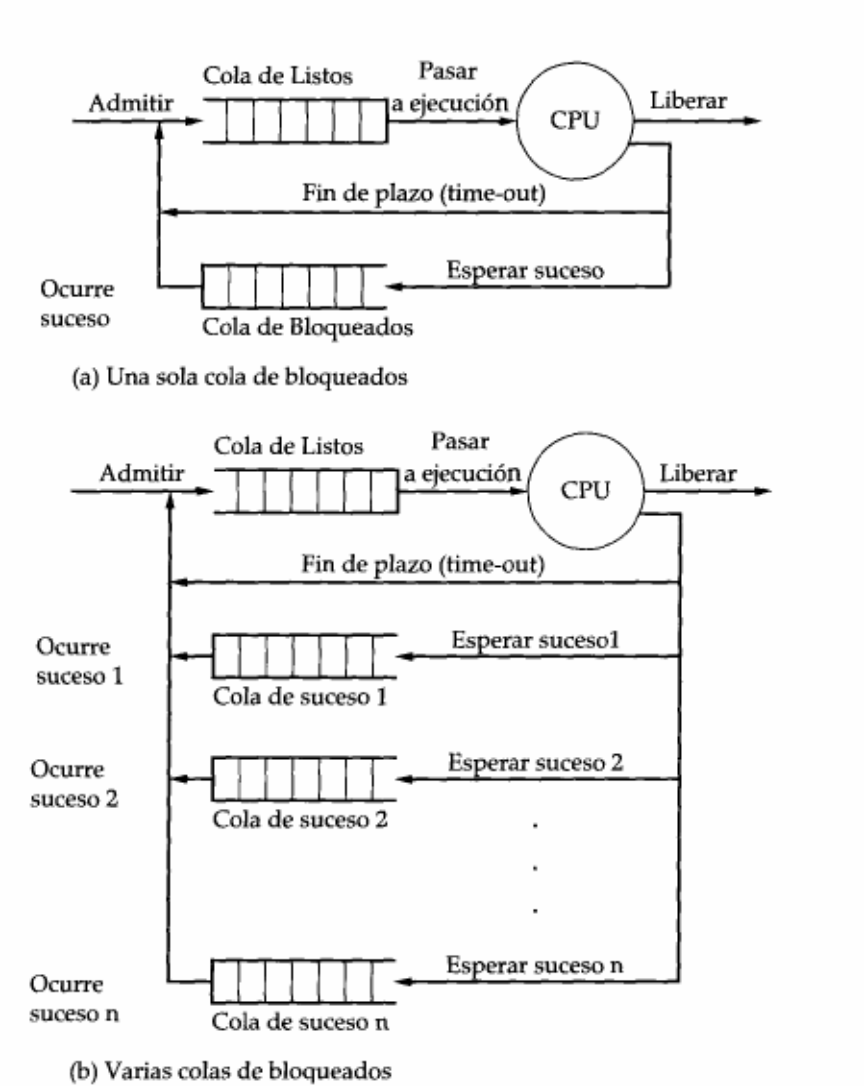


FIGURA 3.6 Modelo de Colas de la figura 3.5

Un retoque final: Si la expedición de procesos está dictada por un esquema de prioridades, entonces es conveniente tener un cierto número de colas de Listos, una para cada nivel de prioridad. El sistema operativo podrá determinar fácilmente cuál es el proceso de prioridad más alta que lleva más tiempo esperando.

Procesos suspendidos

Necesidad del intercambio

Los tres estados principales que se han descrito (Listo, Ejecución, Bloqueado) ofrecen una forma sistemática de modelar el comportamiento de los procesos y de guiar la implementación del sistema operativo. Se han construido muchos sistemas operativos empleando solamente estos tres estados.

Sin embargo, hay una buena justificación para añadir más estados al modelo. Para ver los beneficios de estos nuevos estados, considérese un sistema que no utiliza memoria virtual. Cada proceso que va a ejecutarse debe ser cargado por completo en la memoria principal. De esta manera, en la figura 3.6b, todos los procesos de todas las colas deben estar residentes en la memoria principal.

Recuérdese que la razón de todo este complicado mecanismo es que las actividades de E/S son mucho más lentas que las de cálculo y, por tanto, el procesador en un sistema de monoprogramación está parado la mayor parte del tiempo. Pero la disposición de la figura3.6b no resuelve por completo el problema. Es verdad que, en este caso, la memoria contiene varios procesos y que el procesador puede dedicarse a otro proceso cuando uno esté esperando. Pero el procesador es tan rápido comparado con la E/S que suele ser habitual que todos los procesos de memoria estén esperando por E/S. Así pues, incluso con multiprogramación, el procesador podría estar desocupado la mayor parte del tiempo.

¿Qué hay que hacer? La memoria principal podría ampliarse para alojar más procesos, pero habría dos defectos en este enfoque. Primero, se tiene un coste asociado con la memoria principal, que, aunque es pequeño cuando se trata de bits, comienza a acumularse cuando se trata de megabytes y gigabytes de almacenamiento. Segundo, el apetito de los programas por la memoria ha crecido tan rápidamente como el coste de la memoria ha disminuido. Por tanto, una memoria mayor significa procesos mayores, pero no más procesos.

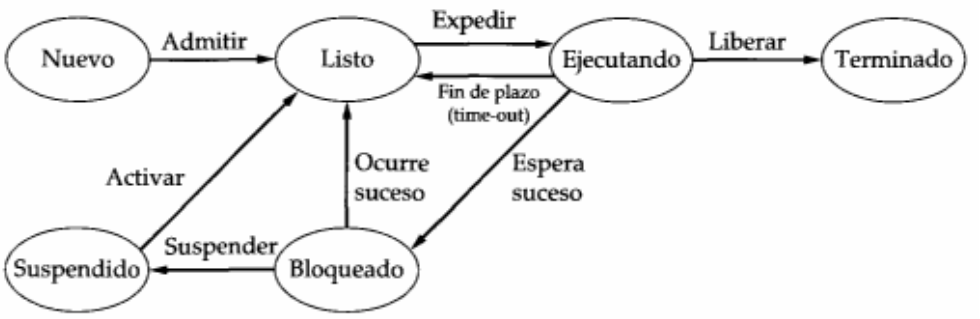
Otra solución es el intercambio, lo que significa mover una parte del proceso o todo el proceso de la memoria principal a disco. Cuando ninguno de los procesos en memoria principal está en estado Listo, el sistema operativo expulsa a disco a uno de los procesos que esté Bloqueado y lo pasa a una cola de Suspendidos. Esta es una cola de procesos existentes que han sido sacados temporalmente de la memoria principal o suspendidos. El sistema operativo trae entonces otro proceso de la cola de Suspendidos o atiende la demanda de crear un nuevo proceso. La ejecución continúa entonces con el nuevo proceso que ha llegado.

El intercambio, no obstante, es una operación de E/S y, por tanto, hay una posibilidad de que el problema empeore en vez de mejorar. Pero como la E/S con el disco es, en general, la E/S más rápida de un sistema (en comparación, por ejemplo, con una cinta o una impresora), el intercambio suele mejorar el rendimiento.

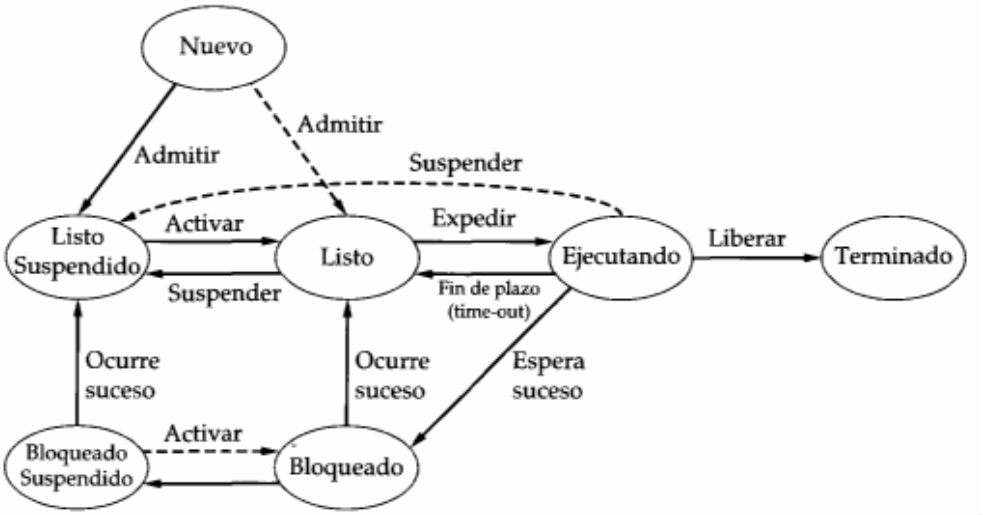
Al emplear el intercambio, tal y como se acaba de describir, se debe añadir un nuevo estado al modelo del comportamiento de los procesos (figura 3.7a), el estado Suspendido. Cuando todos los procesos de la memoria principal están en el estado Bloqueado, el sistema operativo puede suspender un proceso poniéndolo en estado Suspendido y transferirlo a disco. El espacio que se libera de la memoria principal puede utilizarse entonces para traer otro proceso.

Cuando el sistema operativo haya realizado una operación de intercambio de un proceso a disco, tendrá dos opciones para seleccionar el proceso que se va a traer a memoria: Puede admitir un proceso recién creado o puede traer un proceso suspendido previamente. Puede parecer que la preferencia debe ser traer un proceso suspendido previamente para darle servicio, en lugar de hacer crecer la carga total de procesos en el sistema.

Pero esta línea de razonamiento presenta una dificultad. Todos los procesos que fueron suspendidos estaban en el estado Bloqueado en el momento de la suspensión. Realmente no haría ningún bien traer de nuevo a memoria principal un proceso Bloqueado porque no está



(a) Con un estado Suspendido



(b) Con dos estados Suspendido

FIGURA 3.7 Diagrama de transición de estados de un proceso con estados de suspensión

todavía listo para ejecutarse. Debe reconocerse, sin embargo, que cada proceso en estado Suspendido fue bloqueado originalmente por un suceso concreto. Cuando se produzca tal suceso, el proceso se desbloqueara y, posiblemente, estará disponible para su ejecución.

Por tanto, no se necesita volver a pensar sobre este aspecto del diseño. Aquí se tienen dos conceptos independientes: si un proceso está esperando un suceso (bloqueado o no), y si un proceso ha sido expulsado de la memoria principal (suspendido o no). Para ordenar estas combinaciones, hacen falta los cuatro estados siguientes:

- *Listo:* El proceso está en memoria principal y listo para la ejecución.
- *Bloqueado:* El proceso está en memoria principal esperando un suceso.
- *Bloqueado y suspendido:* El proceso está en memoria secundaria esperando un suceso.
- *Listo y suspendido:* El proceso está en memoria secundaria pero está disponible para su ejecución tan pronto como se cargue en la memoria principal.

Antes de observar el diagrama de transición de estados que engloba a los dos nuevos estados de suspensión, debe hacerse mención a otro punto. En la discusión se ha supuesto hasta ahora que no se utiliza memoria virtual y que un proceso estará bien en memoria principal o bien fuera de ella por completo. Con un esquema de memoria virtual, es posible ejecutar un proceso que esté solo parcialmente en memoria principal. Si se hace una referencia a una dirección del proceso que no está en memoria principal, entonces la parte apropiada del proceso debe traerse a memoria. El uso de la memoria virtual parece eliminar la necesidad del intercambio explicito, ya que cualquier dirección deseada de cualquier proceso puede ser trasladada dentro o fuera de la memoria principal por el hardware de gestión de memoria del procesador. Sin embargo, como se vera en el capitulo 7, el rendimiento del sistema de memoria virtual puede desplomarse si hay un número suficientemente grande de procesos activos, todos los cuales están en parte en la memoria principal. Por tanto, incluso en un sistema de memoria virtual, el sistema operativo siempre tendrá que expulsar de cuando en cuando algunos procesos, de forma completa y explicita, en aras del rendimiento.

Va a observarse ahora, con la figura 3.7b y la tabla *3.5,* el modelo de transición de estados que se ha desarrollado. (Las líneas discontinuas en la figura indican transiciones posibles pero no necesarias). Las nuevas e importantes transiciones son las siguientes:

- *Bloqueado → Bloqueado y suspendido:* Si no hay procesos Listos, entonces al menos un proceso Bloqueado se expulsa para dar cabida a otro proceso que no esté bloqueado. Esta transición puede hacerse aun cuando hay procesos listos disponibles, cuando el sistema operativo determina que el proceso que está actualmente en Ejecución o un proceso Listo que serla conveniente expedir requiere más memoria principal para mantener un rendimiento adecuado.
- *Bloqueado y suspendido → Listo y suspendido:* Un proceso en estado Bloqueado y suspendido se pasa al estado Listo y suspendido cuando ocurre el suceso que estaba esperando. Nótese que esto requiere que esté accesible para el sistema operativo la infomación relativa a los procesos Suspendidos.
- *Listo y suspendido → Listo:* Cuando no hay procesos Listos en la memoria principal, el sistema operativo tendrá que traer uno para continuar la ejecución. Además, puede darse el caso de que un proceso en estado Listo y suspendido tenga una prioridad mayor que la de un proceso en estado Listo. En tal caso, el diseñador del sistema operativo puede decidir que es más importante tomar el proceso de mayor prioridad que minimizar el intercambio.
- *Listo → Listo y suspendido:* Generalmente, el sistema operativo prefiere suspender a un proceso Bloqueado en vez de a uno Listo, ya que el proceso Listo podría ejecutarse de inmediato, mientras que el proceso Bloqueado estará ocupando espacio en la memoria principal sin poder ejecutarse. Sin embargo, puede ser necesario suspender un proceso Listo si ésta es la única forma de liberar un bloque lo suficientemente grande de memoria principal. Por último el sistema operativo puede escoger suspender un proceso Listo de más baja prioridad en lugar de uno Bloqueado que sea de prioridad más alta si él cree que el proceso Bloqueado pronto estará listo.

Otras transiciones son también dignas de consideración:

- ***Nuevo → Listo, Suspendido y Nuevo → Listo:*** Cuando se crea un nuevo proceso, se le puede añadir a la cola de listos o a la de listos y suspendidos. En ambos casos, el sis

TABLA 3.5 Transiciones de Estados de los Procesos con Estados de Suspensión

DESDE	HACIA						
	Nuevo	Ejecución	Listo	Bloqueado	Listo y Suspendido	Bloqueado y Suspendido	Terminado
El SO responde a una solicitud de control de trabajos; se conecta un usuario de tiempo compartido; un proceso crea un hijo.	X	X	X	X	X	X	X
Nuevo		X	El SO está preparado para aceptar un proceso más.	X	El SO esta preparado para aceptar un proceso más.	X	X
Ejecución	X	X	Termino su tiempo; petición de servicio al SO; el SO lo expulsa a causa de otro proceso de mayor prioridad; el proceso cede el control.	Petición de petición de petición de suceso.	El usuario servicio al SO; suspensión del recurso; solicita a suspensión del proceso.	X solicita la proceso; el SO	El proceso termina; el proceso cancela.

Listo	X	Seleccionado por el distribuidor como el próximo proceso a ejecutar.	X	X	El SO expulsa algún proceso para liberar memoria.	X	Proceso finalizado por el padre.	
Bloqueado	X	X	Ocurre un suceso	X	X	El SO expulsa algún proceso hacia fuera para liberar memoria; el SO u otro proceso solicitan la suspensión del proceso.	Proceso finalizado por el padre.	
Listo y Suspendido	X	X	El SO intercambia un proceso en memoria	X	X	X	Proceso finalizado por el padre.	
Bloqueado y Suspendido	X	X	X	El SO intercambia un proceso en memoria.	Se produce un suceso.	X	Proceso finalizado por el padre.	

tema operativo necesita construir unas tablas para poder administrar el proceso y asignarle un espacio de direcciones. Podría ser preferible que el sistema operativo llevara a cabo estas labores en un primer momento, de modo que se mantuviera una reserva grande de procesos que no están bloqueados. Con esta estrategia serla frecuente el caso de que hubiese poco espacio en memoria principal para un nuevo proceso; de ahí el uso de la nueva transición Nuevo → Listo y suspendido. Por otro lado, puede argumentarse que una filosofía de creación de los procesos "justo a tiempo", retrasando la creación todo lo que se pueda, reducirla la sobrecarga del sistema operativo y le permitirla llevar a cabo las tareas de creación de procesos en el momento en el que el sistema esté atascado de todas maneras con procesos Bloqueados.

- *Bloqueado y suspendido → Bloqueado:* La inclusión de esta transición puede parecer resultado de un mal diseño. Después de todo, si un proceso no está listo para ejecutarse y aún no está en memoria principal, ¿cuál es el interés por traerlo a memoria? Pero la siguiente situación es posible: Un proceso termina, liberando memoria principal. Hay un proceso en la cola de Bloqueados y suspendidos que tiene una prioridad mayor que La de cualquier proceso de la cola de Listos y suspendidos, así que el sistema operativo tiene razones para suponer que pronto ocurrirá el suceso por el que el proceso está bloqueado. En estas circunstancias, podría parecer razonable traer un proceso Bloqueado a memoria antes que un proceso Listo.
- *Ejecución → Listo y suspendido:* Generalmente, un proceso en Ejecución pasa al estado Listo cuando expira su fracción de tiempo asignado. Sin embargo, si se está expulsando al proceso porque hay un proceso de prioridad mayor en la lista de Bloqueados y suspendidos que se acaba de desbloquear, entonces el sistema operativo podría pasar el proceso en Ejecución directamente a la cola de Listos y suspendidos, liberando espacio en la memoria principal.
- *Varios → Terminado:* Normalmente, los procesos terminan mientras están ejecutándose, bien porque se completaron o bien por causa de alguna condición drástica de error. Sin embargo, en algunos sistemas operativos, un proceso puede ser finalizado por el proceso que lo creó o bien finalizar cuando termina el proceso padre. Si se permite esto, un proceso situado en cualquier estado podrá pasar al estado Terminado.

Otros usos de la suspensión

Hasta ahora, se ha identificado el concepto de proceso Suspendido con el hecho de que el proceso no está en memoria principal. Un proceso que no esté en memoria no estará disponible de inmediato para su ejecución, esté o no esperando un suceso.

Se puede generalizar este concepto de proceso Suspendido. Se define proceso Suspendido como aquel que tiene las características siguientes:

1. Un proceso que está suspendido no está disponible de inmediato para ejecución.
2. El proceso puede estar esperando o no un suceso. Si lo está, la condición de Bloqueado es independiente de La condición de Suspendido y el acontecimiento del suceso bloqueante no lo habilita para la ejecución.
3. El proceso fue situado en el estado suspendido por un agente (por sI mismo, por el proceso padre o por el sistema operativo) con el fin de impedir su ejecución.
4. El proceso no puede apartarse de este estado hasta que el agente lo ordene explícitamente.

La tabla 3.6 enumera algunas razones para la suspensión de un proceso. Una razón que ya se ha discutido es la necesidad de expulsar un proceso a disco para dar cabida a un proceso Listo o, simplemente, para aliviar la presión sobre el sistema de memoria virtual de forma que los procesos restantes tengan disponible más memoria principal. El sistema operativo puede tener otros motivos para suspender un proceso. Por ejemplo, puede emplearse un proceso de auditoria o de seguimiento para supervisar la actividad del sistema; el proceso puede emplearse para registrar el nivel de utilización de diversos recursos (procesador, memoria, canales) y la velocidad de avance de los procesos de usuario en el sistema. El sistema operativo, bajo el control del operador, puede conectar o desconectar este proceso de cuando en cuando. Si el sistema operativo detecta o sospecha un problema, puede suspender un proceso. Un ejemplo de esto es el interbloqueo, que se discutirá en el capitulo *5.* Otro ejemplo:

Se detecta un problema en una línea de comunicaciones y el operador tiene que hacer que el sistema operativo suspenda al proceso que esté usando la línea mientras que se ejecutan algunas pruebas.

Otra serie de razones tienen que ver con las acciones de los usuarios interactivos. Por ejemplo, si un usuario sospecha un defecto en el programa, puede depurarlo suspendiendo la ejecución del programa, examinando y modificando el programa o los datos y reanudando la ejecución. También puede haber un proceso de fondo que recoja estadísticas de contabilidad o seguimiento y que el usuario puede querer activar y desactivar.

Las consideraciones de tiempos pueden conducimos también a un intercambio. Por ejemplo, si un proceso se va a activar periódicamente, pero está libre la mayor parte del tiempo, entonces deberla ser expulsado entre cada uso. Un ejemplo es el de un programa que supervise La utilización o la actividad de los usuarios.

Por último, un proceso padre puede querer suspender a un proceso descendiente. Por ejemplo, el proceso A puede generar un proceso B para llevar a cabo la lectura de un archivo. Posteriormente, el proceso B encuentra un error en el procedimiento de lectura del archivo e informa al proceso A. El proceso A suspende al proceso B para investigar la causa del error.

En todos estos casos, la activación de un proceso Suspendido es solicitada por el agente que solicitó al principio la suspensión.

TABLA 3.6 Razones para la Suspensión de procesos

Intercambio	El sistema operativo necesita liberar suficiente memoria principal para cargar un proceso que está listo para ejecutarse.
Otra razón del SO	El sistema operativo puede suspender un proceso de fondo, de utilidad o cualquier proceso que se sospecha sea el causante de un problema.
Solicitud de un usuario con	Un usuario puede querer suspender a ejecución de un programa fines de depuración o en conexión con el uso de un recurso.
Por tiempo	Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso de contabilidad o de supervisión del sistema) y puede ser suspendido mientras espera el siguiente intervalo de tiempo.
Solicitud del proceso padre	Un proceso padre puede querer suspender a ejecución de un descendiente para examinar o modificar el proceso suspendido o para coordinar la actividad de varios descendientes.

3.2
DESCRIPCIÓN DE PROCESOS

El sistema operativo es el controlador de los sucesos que se producen en un sistema informático. Es el sistema operativo el que planifica y expide a los procesos para su ejecución en el procesador, el que asigna los recursos a los procesos y el que responde a las solicitudes de servicios básicos realizadas por los programas de usuario. Esencialmente, se puede imaginar al sistema operativo como una entidad que administra el uso que hacen los procesos de los recursos del sistema.

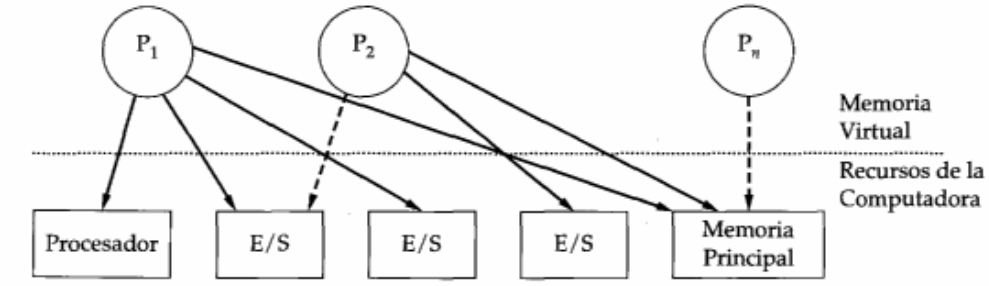
Este concepto queda ilustrado en la figura 3.8. En un entorno de multiprogramación, habrá un cierto número de procesos *(P_1,..., P_n)* que se han creado y que existen en la memoria virtual. Durante el curso de su ejecución, cada proceso necesita tener acceso a ciertos recursos del sistema, entre los que se incluyen el procesador, los dispositivos de E/S y la memoria principal. En la figura, el proceso *P1* está ejecutándose; a! menos una parte del proceso está en memoria principal y tiene el control de dos dispositivos de E/S. El proceso P_2 también está en memoria principal, pero está bloqueado esperando a un dispositivo de E/S que está asignado a *P1*. El proceso *Pn* ha sido descargado a! disco y, por tanto, está suspendido.

Los detalles de la gestión de estos recursos por el sistema operativo para los procesos será explorado en capítulos posteriores. Aquí se está abordando una cuestión más básica: ¿Qué necesita el sistema operativo para ser capaz de controlar los procesos y administrar los recursos para ellos?

Estructuras de control del sistema operativo

Si el sistema operativo va a administrar los procesos y los recursos, entonces tiene que disponer de información sobre el estado actual de cada proceso y de cada recurso. El método universal para obtener esta información es sencillo: El sistema operativo construye y mantiene tablas de información sobre cada entidad que esté administrando. En la figura 3.9 se ofrece una idea general del alcance de este procedimiento, que muestra cuatro tipos de tablas diferentes mantenidas por el sistema operativo: de memoria, de E/S, de archivos y de procesos. Aunque los detalles pueden diferir de un sistema operativo a otro, en lo fundamental todos los sistemas operativos organizan la información en estas cuatro categorías.

Las Tablas de memoria se utilizan para seguir la pista de la memoria principal (real) y secundaria (virtual). Parte de la memoria principal está reservada para el uso del sistema



FIGIJRA 3.8 Procesos y recursos

operativo; el resto está disponible para el uso de los procesos. Los procesos se mantienen en memoria secundaria mediante alguna *forma* de memoria virtual o por un simple mecanismo de intercambio. Las tablas de memoria deben incluir la información siguiente:

- La asignación de memoria principal a los procesos
- La asignación de memoria secundaria a los procesos
- Cualesquiera atributos de protección de segmentos de memoria principal o virtual, tales como qué procesos pueden acceder a ciertas regiones compartidas de memoria
- Cualquier información necesaria para gestionar la memoria virtual

En los capítulos 6 y 7 se estudiarán en detalle las estructuras de datos para la gestión de memoria.

Las **Tablas de E/S** son utilizadas por el sistema operativo para administrar los dispositivos y los canales de E/S del sistema informático. En un momento dado, un dispositivo de E/S puede estar disponible o estar asignado a un proceso en particular. Si hay una operación de E/S en marcha, el sistema operativo necesita conocer el estado de la operación de E/S y la posición de memoria principal que se está utilizando como origen o destino de la transferencia de E/S. La gestión de la E/S se examinará en el capitulo 10.
El sistema operativo también puede mantener **tablas de archivos**, las cuales ofrecen información sobre la existencia de los archivos, su posición en la memoria secundaria, su estado actual y otros atributos. Gran parte de esta información, si no toda, puede ser mantenida

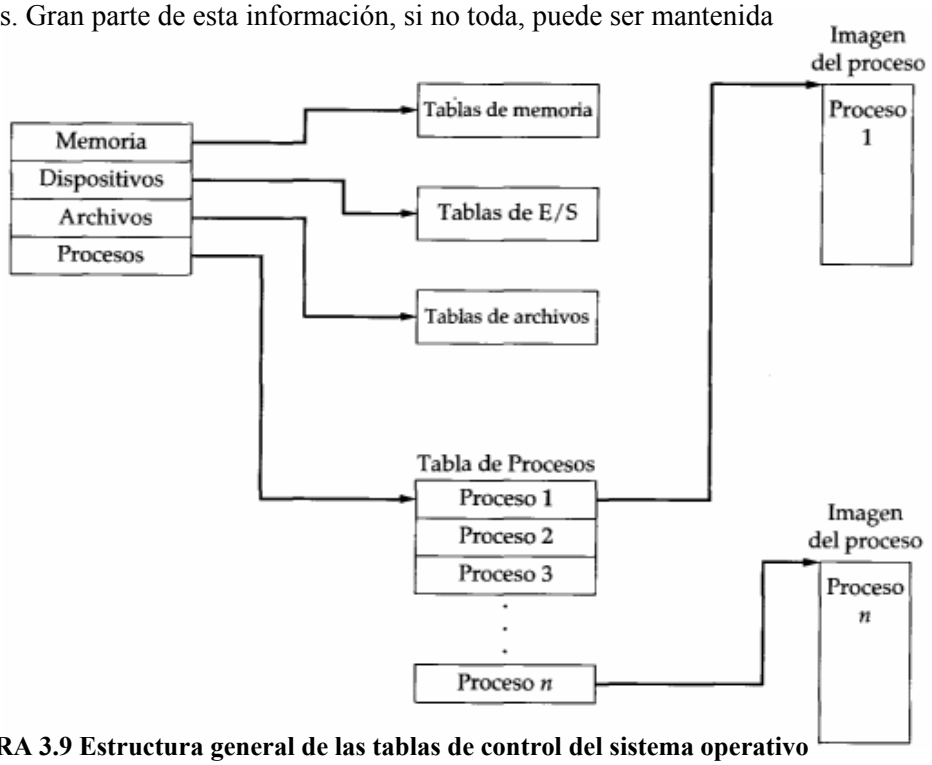


FIGURA 3.9 Estructura general de las tablas de control del sistema operativo

y utilizada por un sistema de gestión de archivos, en cuyo caso el sistema operativo tendrá poco o ningún conocimiento de los archivos. En otros sistemas operativos, gran parte de los detalles de la gestión de archivos son gestionados por el propio sistema operativo. Este tema se explora en el capitulo 11.

Finalmente, el sistema operativo debe mantener tablas de procesos para administrarlos. El resto de esta sección se dedica a un examen de las tablas de procesos requeridas. Antes de continuar con esta discusión, se deben señalar dos puntos. En primer lugar, aunque la figura 3.9 muestra cuatro conjuntos de tablas distintos, debe quedar claro que estas tablas deben estar enlazadas o disponer de referencias cruzadas de alguna manera. La memoria, la E/S y los archivos son administrados en nombre de los procesos, por lo que debe haber alguna referencia directa o indirecta a estos recursos en las tablas de procesos. Los archivos que son referidos en las tablas de archivos son accesibles a través de un dispositivo de *E/S* y, algunas veces, estarán en memoria principal o en memoria virtual. Así pues, hacen falta referencias cruzadas. Las tablas en sí mismas deben ser accesibles por medio del sistema operativo y, por tanto, están sujetas a la gestión de memoria.

El segundo punto tiene que ver con la afirmación anterior de que el sistema operativo crea y mantiene estas tablas. *Surge* entonces la pregunta sobre la procedencia. ¿Cómo crea el sistema operativo las tablas por primera vez? Desde luego, el sistema operativo debe tener algún conocimiento sobre el entorno básico, tal y como cuánta memoria principal hay, cuáles son los dispositivos de E/S y sus identificadores, etc. Este es un asunto de configuración, es decir, cuando se inicializa el sistema operativo, este debe tener acceso a algunos datos de configuración que definan el entorno básico y estos datos deben crearse fuera del sistema operativo, con la asistencia humana.

Estructuras de control de procesos

Considérese lo que debe conocer un sistema operativo si tiene que administrar y controlar a los procesos. En primer lugar, debe saber dónde está ubicado el proceso y, en segundo lugar, debe conocer los atributos del proceso que son necesarios para su administración.

Ubicación de los procesos

Antes de tratar las cuestiones sobre dónde se ubican los procesos o sobre cuAles son sus atributos, se tiene que abordar una cuestión más fundamental aún: ¿Cuál es la manifestación física de un proceso? Como mínimo, un proceso debe incluir un programa o un conjunto de programas que sean ejecutados. Asociados a estos programas hay un conjunto de ubicaciones de datos para las variables locales y globales, y Las constantes definidas. Así pues, un proceso constará, al menos, de la memoria suficiente para albergar los programas y los datos del proceso. Además de esto, en la ejecución de un programa entra en juego normalmente una pila (ver Apéndice IB), que se utiliza para llevar la cuenta de las llamadas a procedimientos y de los parámetros que se pasan entre los procedimientos. Por último, asociado a cada proceso hay una serie de atributos utilizados por el sistema operativo para el control del proceso. Estos atributos se conocen como el bloque de control del proceso.[4] Esta colección de programa, datos, pila y atributos puede llamarse imagen del proceso (tabla 3.7).

[4] Otros nombres habituales utilizados para esta estructura de datos son bloque de control de tarea, descriptor del proceso y descriptor de tarea.

La ubicación de la imagen de un proceso depende del esquema de gestión de memoria utilizado. En el caso más sencillo posible, la imagen del proceso se guarda como un bloque contiguo de memoria. Este bloque se mantiene en memoria secundaria, normalmente en el disco. Para que el sistema operativo pueda administrar el proceso, al menos una pequeña parte de su imagen, que contiene la información a usar por el sistema operativo, debe mantenerse en memoria principal. Para ejecutar el proceso, la imagen completa debe cargarse en la memoria principal. Por tanto, el sistema operativo necesita conocer la ubicación de cada proceso en el disco y también la ubicación de los procesos que estén en memoria principal. En el capitulo 2 se vio una variación ligeramente más compleja de este esquema (el sistema operativo CTSS). Cuando un proceso se descarga al disco en el CTSS, parte de su imagen permanece en memoria principal. De esta forma, el sistema operativo puede seguir la pista de las partes de la imagen de cada proceso que se quedan en memoria principal.

La mayoría de los sistemas operativos modernos utilizan algún tipo de esquema de gestión de memoria en el que la imagen de un proceso consiste en un conjunto de bloques que no tienen por qué estar almacenados consecutivamente. Dependiendo del tipo de esquema utilizado, estos bloques pueden ser de longitud variable (llamados segmentos) o de longitud fija (llamadas paginas) o una combinación de ambos. En cualquier caso, tales esquemas permiten al sistema operativo tener que traer solo una parte de un proceso en particular. De este modo, en un momento dado, una parte de La imagen de un proceso puede estar en la memoria principal y el resto en la memoria secundaria.[5] Por tanto, las tablas de procesos deben mostrar la ubicación de cada segmento y/o página de cada imagen de proceso.

La figura 3.9 representa la estructura de la información de ubicación de la manera siguiente. Hay una tabla principal de procesos con una entrada para cada proceso. Cada entrada contiene, al menos, un puntero a la imagen de un proceso. Si la imagen del proceso contiene varios bloques, entonces esta información estará guardada directamente en la tabla principal o con referencias cruzadas a entradas de las tablas de memoria. Desde luego, esta

TABLA 3.7 Elementos Típicos de una Imagen de Proceso

Datos de Usuario	La parte modificable del espacio de usuario. Puede guardar datos del programa, una zona para una pila del usuario y programas que pueden modificarse.
Programa de Usuario	El programa a ejecutar.
Pila del Sistema	Cada proceso tiene una o más pilas (el último que entra es el primero en salir) asociadas a él. Una pila se utiliza para almacenar los parámetros y as direcciones de retorno.
Bloque de Control de Proceso	Información necesaria para que el sistema operativo controle al proceso (ver tabla 3.8)

[5] Esta breve discusión omite algunos detalles. En particular, toda imagen de un proceso activo está siempre en memoria Secundaria. Cuando una parte de la imagen se carga en la memoria principal, esta se copia en vez de moverse. De este modo, la memoria secundaria conserva una copia de todos los segmentos y/o páginas. Sin embargo, si la parte de la imagen que está en memoria principal se modifica, la copia respectiva de la memoria secundaria quedará desactualizada hasta que la parte en memoria principal se vuelva a copiar al disco.

representación es genérica; cada sistema operativo tiene su propia forma de organizar la in formación de ubicación.

Atributos del proceso

En un sistema de multiprogramación sofisticado, se requiere una gran cantidad de información de cada proceso para su administración. Como ya se dijo, puede considerarse que esta información reside en un bloque de control del proceso. Diferentes sistemas organizarán esta información de modo diferente. Se verán varios ejemplos de esto en la sección *3.5.* Por ahora, se intentará simplemente explorar el tipo de información que podría usarse en un sistema operativo, sin detallar la forma en que se organiza esta información.

La tabla 3.8 enumera las categorías típicas de la información que requiere el sistema operativo para cada proceso. Uno se puede sorprender al principio por la cantidad de información necesaria. Al avanzar en el libro y adquirir una mejor apreciación de las responsabilidades del sistema operativo, esta lista parecerá más razonable.

Se puede agrupar la información de los bloques de control del proceso en las tres categorías generales siguientes:

- Identificación del proceso
- Información del estado del procesador
- Información de control del proceso

Con respecto a la identificación del proceso, en casi todos los sistemas operativos se le asigna a cada proceso un identificador numérico único. El identificador puede ser tan simple como un Índice en la tabla principal del proceso (véase la figura 3.9). Si no hay identificadores numéricos, entonces debe haber una correspondencia que permita al sistema operativo ubicar las tablas apropiadas a partir del identificador de proceso. Este identificador es útil en varios sentidos. Muchas de las otras tablas controladas por el sistema operativo pueden usar identificadores de procesos como referencias cruzadas a las tablas de procesos. Por ejemplo, las tablas de memoria pueden organizarse de manera que ofrezcan un mapa de la memoria principal con indicaciones sobre qué proceso está asignado a cada región de memoria. Referencias similares aparecerán en las tablas de archivos y de E/S. Cuando los procesos se comunican unos con otros, se utiliza el identificador de proceso para informar al sistema operativo del destino de cada comunicación en particular. Cuando se permite que los procesos creen otros procesos, se utilizan identificadores para señalar al padre y a los descendientes de cada proceso.

Además de estos identificadores de proceso, un proceso también puede tener asignado un identificador de usuario que indica quién es el usuario responsable del trabajo.

El siguiente conjunto importante de información es la información de estado del procesador. Básicamente, está formada por el contenido de los registros del procesador. Por supuesto, mientras un proceso está ejecutándose, la información está en los registros. Cuando se interrumpe el proceso, toda la información de los registros debe salvarse de forma que pueda restaurarse cuando el proceso reanude su ejecución. La naturaleza y el número de los registros involucrados dependen del diseño del procesador. Normalmente, en el conjunto de registros se incluyen los registros visibles para el usuario, los registros de control y de estado

TABLA 3.8 Elementos Básicos de un Bloque de Control de Proceso

Identificación de Proceso

Identificadores

Los identificadores numéricos que se pueden guardar en el bloque de control de proceso incluyen:

- Identificador de este proceso
- Identificador del proceso que creó a este proceso (el proceso padre)
- Identificador del usuario

Información de Estado del Procesador

Registros Visibles para el Usuario

Un registro visible para el usuario es aquél al que puede hacerse referencia por medio del lenguaje máquina que ejecuta el procesador. Normalmente, existen de 8 a 32 de estos registros, aunque algunas implementaciones RISC tienen más de 100.

Registros de Control y de Estado

Hay varios registros del procesador que se emplean para controlar su funcionamiento. Entre estos se incluyen:

- *Contador de programa:* Contiene la dirección de la próxima instrucción a ser tratada.
- *Códigos de condición:* Muestran el resultado de la operación aritmética o lógica más reciente (signo, cero, acarreo, igualdad, desbordamiento).
- *Información de estado:* incluye los indicadores de habilitación o inhabilitación de interrupciones y el modo de ejecución.

Punteros de pila

Cada proceso tiene una o más pilas LIFO del sistema asociadas. Las pilas se utilizan para almacenar los parámetros y las direcciones de retorno de los procedimientos y de las llamadas al sistema. El puntero de pila siempre apunta a la cima de la pila.

Información de Control del Proceso

Información de Planificación y de Estado

Esta es la información que se necesita por el sistema operativo para llevar a cabo sus funciones de planificación. Los elementos típicos de esta información son los siguientes:

- *Estado del proceso:* Define la disposición del proceso para ser planificado para ejecutar (en ejecución, listo, esperando, detenido).
- *Prioridad:* Se puede usar uno o más campos para describir la prioridad de planificación de los procesos. En algunos sistemas se necesitan varios valores (por omisión, actual, la más alta permitida).
- *Información de planificación:* Esta dependerá del algoritmo de planificación utilizado. Como ejemplos se tienen la cantidad de tiempo que el proceso ha estado esperando y la cantidad de tiempo que el proceso ejecutó la última vez.
- *Suceso:* La identidad del suceso que el proceso está esperando antes de poder reanudarse.

Estructuración de Datos

Un proceso puede estar enlazado con otros procesos en una cola, un anillo o alguna otra estructura. Por ejemplo todos los procesos que están en estado de espera de un nivel determinado de prioridad pueden estar enlazados en una cola. Un proceso puede mostrar una relación padre-hijo (creador - creado) con otro proceso. El bloque de control de proceso puede contener punteros a otros procesos para dar soporte a estas estructuras.

Comunicación entre Procesos

Puede haber varios indicadores, señales y mensajes asociados con la comunicación entre dos procesos independientes. Una parte de esta información o toda ella se puede guardar en el bloque de control de proceso.

(continua)

TABLA 3.8 ***(Continuación)***

Información de Control del Proceso
Privilegios de los procesos A los procesos se es otorgan privilegios en términos de la memoria a la que pueden acceder y el tipo de instrucciones que pueden ejecutar. Además, también se pueden aplicar privilegios al uso de los servicios y utilidades del sistema.
Gestión de Memoria Esta sección puede incluir punteros a las tablas de páginas y/o segmentos que describen la memoria virtual asignada al proceso.
Propiedad de los Recursos y Utilización Se pueden indicar los recursos controlados por el proceso, tales como los archivos abiertos. También se puede incluir un histórico de la utilización del procesador o de otros recursos; esta información puede ser necesaria para el planificador.

y los punteros de pila. Los *registros visibles para el usuario* son aquellos accesibles para los programas de usuario y que se usan para almacenar datos temporalmente. La mayoría de los procesadores incluyen de 8 a 32 registros. Algunas arquitecturas recientes con juegos reducidos de instrucciones (RISC, *Reduced-Instruction Set Computer)* disponen de más de 100 registros.

Se emplean varios registros de *control* y de *estado* para controlar la operación del procesador: La mayoría de éstos, en la mayoría de las máquinas, no son visibles para los usuarios. Algunos de ellos pueden ser visibles para las instrucciones de máquina ejecutadas en modo de control o del sistema operativo. Un registro de control que se encuentra en todos los procesadores es el contador de programa o registro de instrucción, que contiene la dirección de la próxima instrucción que debe leerse. Además, todos los diseños de procesadores incluyen un registro o conjunto de registros, a menudo conocido como palabra de estado del programa (PSW, *Program Status Word),* que contiene la información de estado. Normalmente, la PSW contiene los códigos de condición junto a otra información de estado.

Un buen ejemplo de palabra de estado del programa es la de las máquinas VAX, que se muestra en la figura 3.10 y en la tabla 3.9. Este formato es el que utiliza el sistema operativo VMS y el UNTX BSD de Berkeley, que se ejecutan en los VAX.

Por último, uno o más *registros de pila* proporcionan los punteros a las pilas empleadas por el sistema operativo para controlar la ejecución de los programas y llevar la cuenta de las interrupciones.

A la tercera categoría general de información del bloque de control de proceso se le puede llamar, a falta de un nombre mejor, **información de control del proceso.** Esta es la información adicional necesaria para que el sistema operativo controle y coordine los diferentes procesos activos. La última parte de la tabla 3.8 indica el ámbito de esta información. A medida que se examinen los detalles de la funcionalidad de los sistemas operativos en los capítulos sucesivos, quedara más clara la necesidad de algunos de los elementos de esta lista.

La figura 3.11 muestra la estructura de la imagen de un proceso en memoria virtual. Cada imagen de proceso consta de un bloque de control de proceso, una pila de usuario, el espacio de direcciones privadas del proceso y cualquier otro espacio de direcciones que corn-

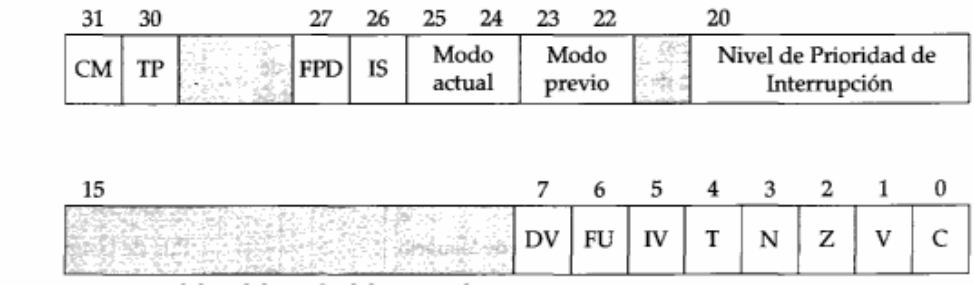


FIGURA 3.10 Palabra del estado del procesador VAX

TABLA 3.9 Campos de la Palabra de Estado del Procesador VAX

Códigos de condición (N, Z, V, C)

Estos bits reflejan el resultado de a instrucción más reciente que les afectó: N = negativo, Z = cero *(zero),* V = desbordamiento *(overflow),* C = acarreo *(carry).* Las instrucciones de salto condicional deben comprobar estos bits.

Indicadores de Habilitación de Cepos (DV, FU, IV, T)

Estos bits se utilizan para habilitar e inhabilitar los tipos de interrupción llamados cepos: DV = desbordamiento decimal *(decimal overflow),* FU = desbordamiento de coma flotante *(floating underfiow),* IV = desbordamiento de enteros *(integer overflow),* T = traza.

Nivel de Prioridad de lnterrupciones (IPL, Interrupt Priority Level)

El nivel actual de prioridad del procesador. Solo serán reconocidas las interrupciones de mayor prioridad.

Modo Previo

Valor del modo de acceso anterior al actual

Modo Actual

Modo actual de acceso del procesador, que determina las instrucciones que el procesador puede ejecutar y las posiciones de la memoria virtual a las que puede acceder la instrucción actual. Los cuatro modos son: núcleo, ejecutor, supervisor y usuario. Véase el problema 3.5.

Pila de lnterrupciones (IS, Interrupt Stack)

Indica si el procesador está procesando una interrupción. Si es así, el procesador hace uso de una pila especial llamada pila de interrupciones.

Primera Parte Hecha (FPD, First Part Done)

Usado en las instrucciones que pueden ser interrumpidas durante su ejecución. Si FDP = 1, cuando el procesador retorne de una interrupción, se reanudará la operación en donde se dejó, en vez de reiniciar la instrucción.

Traza Pendiente (TP Trace Pending)

El servicio de traza o seguimiento permite al depurador tomar el control después de a ejecución de cada instrucción, usando una interrupción de seguimiento. Cuando las interrupciones de seguimiento están habilitadas, el bit TP asegura que solo se puede producir una determinada interrupción para cada instrucción.

Modo de Compatibilidad (CM, Compatibility Mode)

Cuando el procesador VAX-11 está en modo de compatibilidad, ejecuta instrucciones del PDP-1 1 en vez de instrucciones del VAX. Otros miembros de la familia VAX ofrecen esta función por emulación de software.

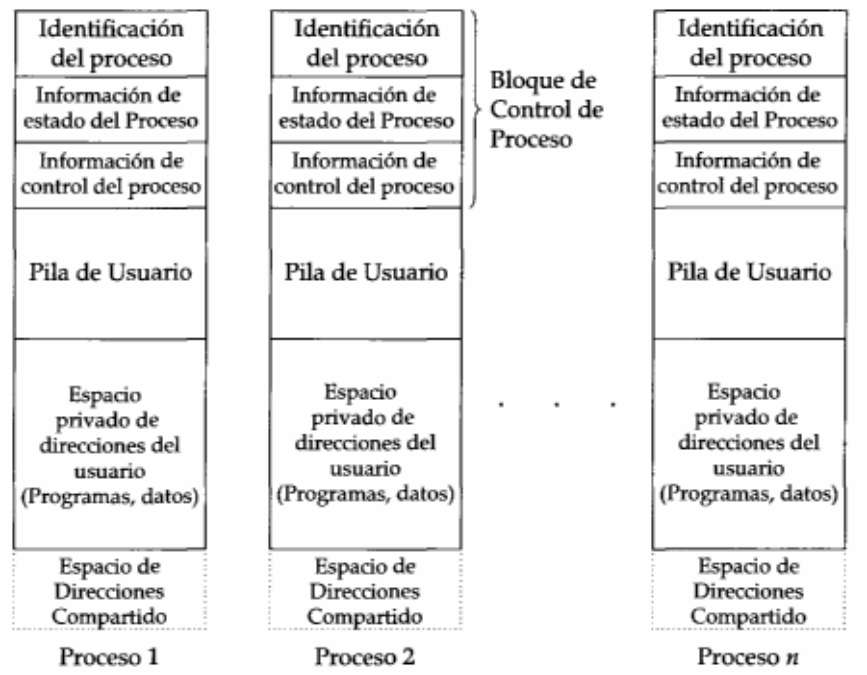


FIGURA 3.11 Procesos de usuario en memoria virtual

parta con otros procesos. En la figura, cada imagen de proceso aparece como un rango contiguo de direcciones. En una implementación real, éste puede que no sea el caso, sino que dependa del esquema de gestión de memoria y de la forma en la que el sistema operativo organiza las estructuras de control.

Como se indica en la tabla 3.8, el bloque de control de proceso puede contener informa-ción de estructuración, incluyendo punteros que permiten enlazar los bloques de control de procesos. Por tanto, las colas que se describieron en la sección precedente podrían implementarse como listas enlazadas de los bloques de control de procesos. Por ejemplo, la estructura de cola de la figura 3.6a podría implementarse como se propone en la figura 3.12.

El papel del bloque de control del proceso

El bloque de control de proceso es la estructura de datos central y más importante de un sistema operativo. Cada bloque de control de proceso contiene toda la información de un proceso necesaria para el sistema operativo. Los bloques son leídos y/o modificados por casi todos los módulos de un sistema operativo, incluyendo aquellos que tienen que ver con la planificación, la asignación de recursos, el tratamiento de interrupciones y el análisis y supervisión del rendimiento. Puede decirse que el conjunto de los bloques de control de procesos definen el estado del sistema operativo.

Esto saca a relucir una cuestión importante de diseño. Una serie de rutinas del sistema operativo necesitarán acceder a la información de los bloques de control de procesos. La provisión de acceso directo a estas tablas no es difícil. Cada proceso está dotado de un único ID que puede utilizarse como índice en una tabla de punteros a los bloques de control de procesos. La dificultad no está en el acceso, sino más bien en la protección. Existen dos problemas:

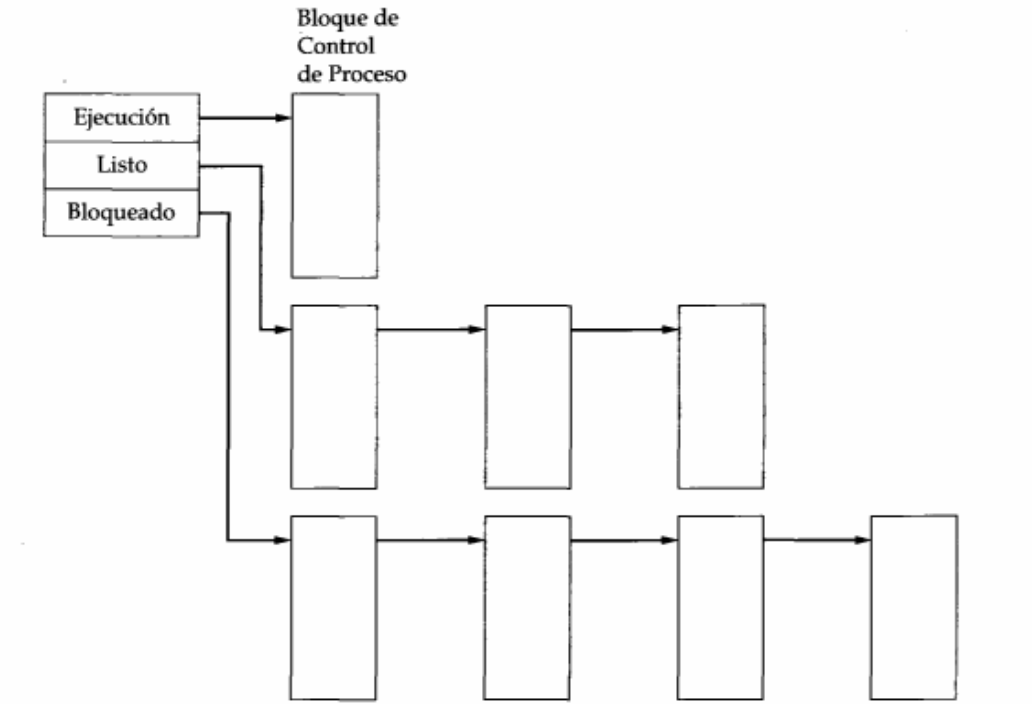


FIGURA 3.12 Estructuras de colas de procesos

- Un error en una sola rutina, como la de tratamiento de interrupciones, puede dañar los bloques de control de procesos, lo que destruirla la capacidad del sistema para administrar los procesos afectados.

- Un cambio de diseño en la estructura o en la semántica del bloque de control de procesos podría afectar a varios módulos del sistema operativo.

Estos problemas se pueden abordar exigiendo a todas las rutinas del sistema operativo que pasen a través de una rutina de manejo, cuya única tarea serla la de proteger los bloques de control de proceso y que se constituiría en el único árbitro para leer y escribir en estos bloques. La concesión en el empleo de una rutina tal está en el rendimiento y en el grado con el que pueda confiarse en que el resto del software del sistema sea correcto.

3.3

CONTROL DE PROCESOS

Modos de ejecución

Antes de continuar la discusión sobre la forma en que el sistema operativo gestiona los procesos, hace falta distinguir entre el modo de ejecución del procesador que normalmente se asocia con el sistema operativo y el modo que normalmente se asocia con los programas

de usuario. La mayoría de los procesadores dan soporte para dos modos de ejecución por lo menos. Ciertas instrucciones pueden ejecutarse solo en modo privilegiado. Entre éstas están la lectura o modificación de registros de control (como la palabra de estado del programa), instrucciones primitivas de E/S e instrucciones relativas a la gestión de memoria. Además, se puede acceder a ciertas regiones de memoria solo en el modo más privilegiado.

El modo menos privilegiado a menudo se conoce como *modo de usuario,* ya que los programas de usuario ejecutan normalmente en ese modo. Al modo más privilegiado normalmente se le conoce como *modo del sistema, modo de control* o*, modo del núcleo.* Este último término se refiere al núcleo del sistema operativo, que es la parte del sistema operativo que lleva a cabo las funciones importantes del sistema. La tabla 3.10 enumera las funciones que normalmente se hallan en el núcleo de un sistema operativo.

La razón por La que se usan dos modos debe quedar clara. Es necesario proteger al sistema operativo y a las tablas importantes del mismo, tales como los bloques de control de procesos, de las injerencias de los programas de usuario. En el modo del núcleo, el software tiene control completo del procesador y de todas sus instrucciones, registros y memoria. Este nivel de control no es necesario y, por seguridad, tampoco conveniente para los programas de usuario.

Surgen dos preguntas: ¿Cómo conoce el procesador en qué modo va a ejecutar? ¿Cómo se cambia de modo? Para la primera pregunta, normalmente hay un bit en la PSW que indica el modo de ejecución. El bit es cambiado como respuesta a ciertos sucesos. Por ejemplo, cuando un usuario hace una llamada a un servicio del sistema operativo, el modo se cambia al de núcleo. Esto se suele llevar a cabo ejecutando una instrucción que cambia el modo. Un ejemplo de cómo se hace esto es la instrucción de Cambio de Modo (CHM, *Change Mode)* del VAX. Cuando el usuario hace una llamada a un servicio del

TABLA 3.10 Funciones Básicas del Núcleo de un Sistema Operativo

Gestión de Procesos

- Creación y terminación de los procesos
- Planificación y expedición de los procesos
- Cambio de procesos
- Sincronización de procesos y soporte para la comunicación entre procesos
- Gestión de los bloques de control de procesos

Gestión de memoria

- Asignación de espacios de direcciones a los procesos
- Intercambio
- Gestión de páginas y segmentos

Gestión de E/S

- Gestión de buffers
- Asignación de canales de E/S y dispositivos a los procesos

Funciones de Soporte

- Tratamiento de interrupciones
- Contabilidad
- Supervisión

sistema o cuando una interrupción transfiere el control a una rutina del sistema, la rutina ejecuta CHM para entrar en un modo más privilegiado y la ejecuta de nuevo para pasar a un modo menos privilegiado, antes de devolver el control al proceso del usuario. Si un programa de usuario intenta ejecutar un CHM, se originará simplemente una llamada al sistema operativo, que devolverá un error a menos que esté permitido el cambio de modo.

Creación de procesos

Anteriormente, en la sección 3.1, se discutieron los sucesos que conducían a la creación de un nuevo proceso. Una vez tratadas las estructuras de datos asociadas a los procesos, se está en condiciones de describir brevemente los pasos que entran en juego en la creación real de los procesos.

Una vez que el sistema operativo decide, por alguna razón (tabla 3.1), crear un nuevo proceso, éste puede proceder como sigue:

1. Asignar un único identificador al nuevo proceso. En ese momento se añade una nueva entrada a la tabla principal de procesos, que contiene una entrada por proceso.
2. Asignar espacio para el proceso. Esto incluye todos los elementos de la imagen del proceso. Así pues, el sistema operativo debe saber cuánto espacio se necesitará para el espacio privado de direcciones del usuario (programas y datos) y para la pila del usuario. Estos valores se pueden asignar por omisión en función del tipo de proceso o bien se puede asignar a partir de la solicitud del usuario cuando se crea el trabajo. Si un proceso es generado por otro, el proceso padre puede pasarle al sistema operativo los valores necesarios como parte de la solicitud de creación del proceso. Si algún espacio de direcciones existente se va a compartir con este nuevo proceso, entonces se deben establecer los enlaces adecuados. Por último, se debe asignar espacio para el bloque de control del proceso.
3. Debe inicializarse el bloque de control del proceso. La parte de identificación del proceso contiene el ID de este proceso junto a otros ID apropiados, tales como el del proceso padre. La parte de información del estado del procesador normalmente se inicializa con la mayor parte de las entradas a cero, excepto para el contador de programa (que se prepara con el punto de entrada del programa) y los punteros a las pilas del sistema (que establecen los limites de la pila del proceso). La parte de información de control del procesador se inicializa a partir de los valores estándares por omisión y los atributos que se han solicitado para el proceso. Por ejemplo, el estado del proceso suele inicializarse a Listo o a Listo y suspendido. La prioridad puede asignarse por omisión al valor más bajo, a menos que se haya hecho una solicitud explicita de un valor mayor. Inicialmente, puede que el proceso no posea ningún recurso (dispositivos de E/S, archivos), a menos que se haya hecho una solicitud explicita de los mismos o a menos que se hayan heredado del proceso padre.
4. Se deben establecer los enlaces apropiados. Por ejemplo, si el sistema operativo mantiene cada cola de planificación como una lista enlazada, entonces el proceso nuevo se debe poner en la cola de Listos o de Listos y suspendidos.

5. Puede haber otras estructuras de datos que crear o ampliar. Por ejemplo, el sistema operativo puede mantener un archivo de contabilidad para cada proceso que sea utilizado más tarde con propósitos de facturación y/o evaluación del rendimiento.

Cambio de proceso

A primera vista, la función de cambio de proceso parece sencilla. En cierto momento, un proceso que está ejecutándose se interrumpe, el sistema operativo pone a otro proceso en el estado de Ejecución y pasa el control a dicho proceso. Sin embargo, surgen diversas cuestiones de diseño. En primer lugar, ¿qué sucesos provocan un cambio de proceso? Otra cuestión es que se debe hacer una distinción entre cambio de contexto y cambio de proceso. Por último, ¿qué debe hacer el sistema operativo con las diferentes estructuras de datos bajo su control para llevar a cabo un cambio de proceso?

Cuándo cambiar de proceso

Un cambio de proceso puede producirse en cualquier momento en que el sistema operativo haya tornado el control a partir del proceso que está actualmente ejecutándose. La tabla 3.11 propone los sucesos posibles que pueden dar el control al sistema operativo.

En primer lugar, se van a tener en cuenta las interrupciones del sistema. Se pueden distinguir, como hacen muchos sistemas, dos clases de interrupciones del sistema, una conocida simplemente como interrupción y otra conocida como cepo. La primera es originada por algún tipo de suceso que es externo e independiente del proceso que está ejecutándose, como la culminación de una operación de E/S. La segunda tiene que ver con una condición de error o de excepción generada dentro del proceso que está ejecutándose, como un intento ilegal de acceso a un archivo. En una **interrupción ordinaria, el control** se transfiere primero a un gestor de interrupciones, quien lleva a cabo algunas tareas básicas y, después, se salta a una rutina del sistema operativo que se ocupa del tipo de interrupción que se ha producido. Algunos ejemplos son los siguientes:

- *Interrupción de reloj:* El sistema operativo determina si el proceso que está en ejecución ha estado ejecutando durante la fracción máxima de tiempo permitida. Si esto ocurre, el proceso debe pasar al estado Listo y se debe expedir otro proceso.
- *Interrupción de E/S:* El sistema operativo determina exactamente que se ha producido una acción de E/S. Si la acción constituye un suceso que están esperando uno o más procesos, entonces el sistema operativo traslada todos los procesos bloqueados correspondientes al estado Listo (y los procesos Bloqueados y suspendidos pasan al estado de Listos y suspendidos). El sistema operativo debe entonces decidir si se reanuda La ejecución del proceso que está actualmente en estado de Ejecución o se expulsa a dicho proceso en favor de un proceso Listo de mayor prioridad.

TABLA 3.11 Mecanismos para la interrupción de la Ejecución de un Proceso [KRAK88]

Mecanismo	Causa	Uso
Interrupción	Externa a la ejecución de la instrucción en curso	Reacción a un suceso asincrónico externo
Cepo	Asociada con a ejecución de la instrucción en curso	Tratamiento de un error o de una condición excepcional
Llamada del supervisor	Solicitud explicita	Llamada a una función del sistema operativo

- *Fallo de memoria:* El procesador encuentra una referencia a una dirección de memoria virtual de una palabra que no está en memoria principal. El sistema operativo debe traer el bloque (página o segmento) que contiene la referencia, de la memoria secundaria a la memoria principal. Después de hacer la solicitud de E/S para traer el bloque de memoria, el sistema operativo puede llevar a cabo un cambio de contexto para reanudar la ejecución de otro proceso; el proceso que cometió el fallo de memoria se pasa a estado Bloqueado. Después de que el bloque en cuestión se cargue en memoria, dicho proceso se pondrá en estado Listo.

En los cepos, el sistema operativo determina si el error es fatal. Si lo es, el proceso que se estaba ejecutando pasa al estado de Terminado y se produce un cambio de proceso. Si no es fatal, la acción del sistema operativo dependerá de La naturaleza del error y del diseño del sistema operativo. Se puede intentar algún procedimiento de recuperación o, simplemente, notificarlo al usuario. Se puede hacer un cambio de proceso o, simplemente, reanudar el mismo proceso que se estaba ejecutando.

Finalmente, el sistema operativo puede activarse mediante una llamada de supervisor desde e programa que se está ejecutando. Por ejemplo, está ejecutándose un proceso de usuario y se alega a una instrucción que solicita una operación de E/S, tal como abrir un archivo. Esta llamada provoca la transferencia a una rutina que forma parte del código del sistema operativo. Por lo general, el uso de una llamada al sistema hace que el proceso de usuario pase al estado Bloqueado.

Cambio de contexto

En el capitulo 1 se discutió sobre la inclusión de un ciclo de interrupción como parte del ciclo de instrucción. Recuérdese que, en el ciclo de interrupción, el procesador comprueba si se ha producido alguna interrupción, lo que se indicaría por la presencia de una señal de interrupción. Si no hay pendiente ninguna interrupción, el procesador continúa con el ciclo de lectura de la instrucción siguiente del programa en curso del proceso actual. Si hay alguna interrupción pendiente, el procesador hace lo siguiente:

1. Salva el contexto del programa que está ejecutándose.
2. Asigna al contador de programa el valor de la dirección de comienzo de un programa de *tratamiento de la interrupción.*

El procesador continúa entonces con el ciclo de lectura de instrucción y trae la primera instrucción del programa de tratamiento de interrupciones, que atenderá a la interrupción.

Una pregunta que puede plantearse es: ¿Que es lo que constituye el contexto que se salva? La respuesta es que se debe incluir cualquier información que pueda alterarse por la ejecución de la rutina de tratamiento de la interrupción y que pueda sen necesaria para reanudar el programa que fue interrumpido. Así pues, debe salvarse la parte del bloque de control del proceso que se denomina información de estado del procesador. Esto incluye el contador de programa, otros registros del procesador y la información de la pila.

¿Hace falta hacer algo más? Ello dependerá de lo que ocurra a continuación. La rutina de tratamiento de la interrupción es normalmente un programa corto que lleva a cabo unas pocas tareas básicas relacionadas con una interrupción. Por ejemplo, se marca el indicador que señala la presencia de una interrupción, puede enviar un acuse de recibo a la entidad que produjo la interrupción (como un módulo de E/S) y puede hacer algunas labores básicas relacio

nadas con los efectos del suceso que causó la interrupción. Por ejemplo, si la interrupción está relacionada con un suceso de E/S, el gestor de interrupciones comprobará condiciones de error. Si se ha producido un error, la rutina de tratamiento puede enviar una señal al proceso que solicitó originalmente la operación de E/S. Si La interrupción es de reloj, el gestor deberá darle el control al distribuidor, quien querrá pasarle el control a otro proceso, puesto que la fracción de tiempo asignada al proceso que estaba ejecutándose habrá expirado.

¿Qué sucede con el resto de información del bloque de control del proceso? Si la interrupción va a venir seguida de un cambio a otro proceso, entonces hace falta hacer cierto trabajo. No obstante, la palabra clave en la oración anterior es la condiciona *si.* En la mayoría de los sistemas operativos, el acontecimiento de una interrupción no provoca necesariamente un cambio de proceso. Es posible que después de que el gestor de interrupciones haya ejecutado, el proceso que estaba ejecutándose reanude su ejecución. En tal caso, todo lo que hay que hacer es salvar la información de estado del procesador cuando se produzca la interrupción y restaurar dicha información cuando el control vuelva al programa que estaba en marcha. Las funciones de salvar y restaurar suelen llevarse a cabo en el hardware.

Cambio de estado de los procesos

Está claro entonces que el cambio de contexto es un concepto distinto del cambio de proceso.[6] Puede producirse un cambio de contexto sin cambiar el estado del proceso que está actualmente en estado de Ejecución. En tal caso, salvar el contexto y restaurarlo posteriormente involucra un pequeño coste extra. Sin embargo, si el proceso que estaba ejecutándose tiene que pasar a otro estado (Listo, Bloqueado, etc.), el sistema operativo tiene que llevar a cabo cambios substanciales en su entorno. Los pasos involucrados en un cambio completo de proceso son los siguientes:

1. Salvar el contexto del procesador, incluyendo el contador de programa y otros registros.
2. Actualizar el bloque de control del proceso que estaba en estado de Ejecución. Esto implica cambiar el estado del proceso a alguno de los otros estados (Listo, Bloqueado, Listo y suspendido, Terminado). También se tienen que actualizar otros campos significativos, incluyendo la razón por la que se abandona el estado de Ejecución y la información de contabilidad.
3. Mover el bloque de control del proceso a la cola apropiada (Listos, Bloqueados por el Suceso *i,* Listos y suspendidos).
4. Seleccionar otro proceso para ejecución; este tema se explora en los capítulos 6 y 7.
5. Actualizar el bloque de control del proceso seleccionado. Esto incluye cambiar el estado del proceso a Ejecución.
6. Actualizar las estructuras de datos de gestión de memoria. Esto puede hacer falta dependiendo de cómo se gestione la traducción de direcciones; este tema se analiza en los capítulos 6 y 7.
7. Restaurar el contexto del procesador a aquel que existía en el momento en el que el proceso seleccionado dejó por última vez el estado de Ejecución, cargando los valores previos del contador de programa y de otros registros.

[6] Desafortunadamente, algunos libros de texto sobre el tema usan el término *cambio de contexto* queriendo decir *cambio de proceso* y no tienen un término particular para la sencilla acción que aquí se ha definido como cambio de contexto.

Así pues, el cambio de proceso, que supone un cambio de estado, requiere un esfuerzo considerablemente mayor que un cambio de contexto.

Ejecución del sistema operativo

En el capitulo 2 se señalaron algunos hechos curiosos sobre los sistemas operativos:

- El sistema operativo funciona de la misma forma que un software corriente, es decir, es un programa ejecutado por el procesador.
- El sistema operativo abandona frecuentemente el control y debe depender de que el procesador le permita recuperarlo.

Si el sistema operativo es solamente una colección de programas y si es ejecutado por el procesador, como cualquier otro programa, ¿es el sistema operativo un proceso? Si lo fuese, ¿cómo se controla?

Estas interesantes preguntas han merecido variadas respuestas de los diseñadores de sistemas operativos. La figura 3.13 ilustra un rango de enfoques que pueden encontrarse en varios de los sistemas operativos contemporáneos.

Núcleo fuera de todo proceso

Un enfoque bastante tradicional y habitual en muchos de los sistemas operativos más antiguos es ejecutar el núcleo del sistema operativo fuera de cualquier proceso (figura 3.1 3a). Con este enfoque, cuando el proceso en ejecución es interrumpido o hace una llamada de supervisor, se salva el contexto del procesador para este proceso y se pasa el control al núcleo. El sistema operativo tiene su propia región de memoria y su propia pila del sistema para controlar las llamadas y retornos de procedimientos. El sistema operativo puede llevar a cabo cualquier función deseada y luego restaurar el contexto del proceso interrumpido para reanudarlo. Otra solución seria que el sistema operativo pudiese completar la función de salvar el entorno del proceso y continuar con la planificación y expedición de otro proceso. Que sea esto lo que pase depende de la causa de la interrupción y de las circunstancias del momento.

En cualquier caso, el punto clave es que se considera que el concepto de proceso se aplica solo a los programas del usuario. El código del sistema operativo se ejecuta como una entidad separada que opera en modo privilegiado.

Ejecución dentro de los procesos de usuario

Una alternativa que es común en los sistemas operativos de máquinas pequeñas (minicomputadores y microcomputadores) es ejecutar casi todo el software del sistema operativo en el contexto de un proceso de usuario. El enfoque es que el sistema operativo es principalmente una colección de rutinas que el usuario llama para llevar a cabo varias funciones y que son ejecutadas dentro del entorno del proceso de usuario, como se ilustra en la figura 3. 13b. En un punto dado cualquiera, el sistema operativo estará gestionando *N* imágenes de procesos. Cada imagen incluye no solo las regiones que se ilustran en la figura 3.11, sino también zonas de programa, de datos y de pila para los programas del núcleo.

La figura 3.14 propone una estructura típica para la imagen de un proceso de usuario según esta estrategia. Una pila del núcleo separada se utiliza para gestionar las llamadas y los retornos mientras que el proceso esté en el modo del núcleo. El código y los datos del sistema operativo están en el espacio de direcciones compartidas y son compartidos por todos los procesos de usuario.

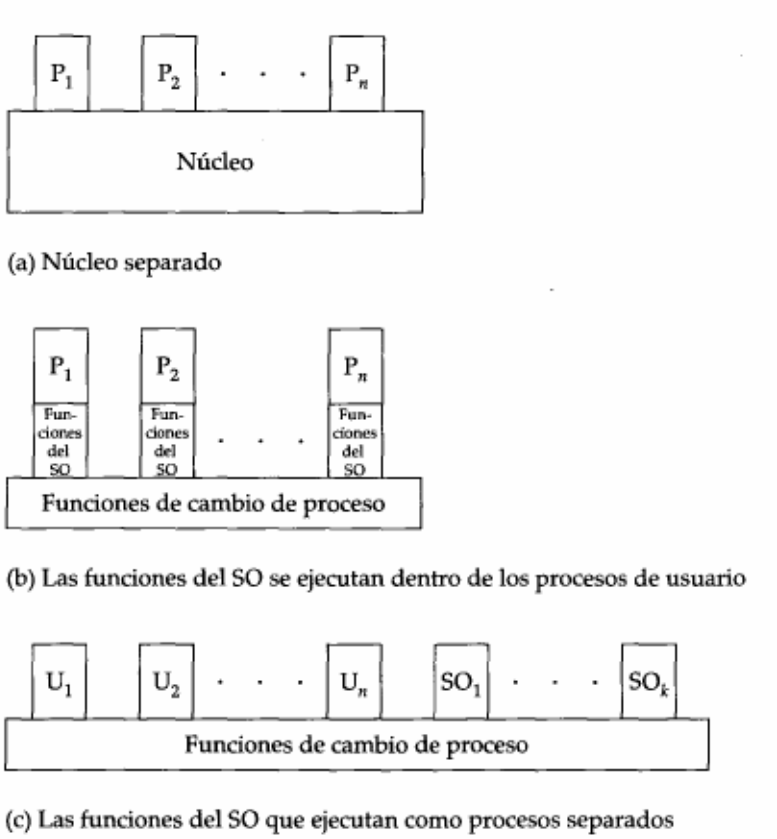


FIGURA 3.13 Relación entre el sistema operativo y los procesos de usuario

Cuando se produce una interrupción, un cepo o una llamada del supervisor, el procesador se pone en modo del núcleo y el control pasa al sistema operativo. Con tal fin, se salva el contexto del procesador y tiene lugar un cambio de contexto hacia una rutina del sistema operativo. Sin embargo, la ejecución continúa dentro del proceso de usuario en curso. De esta manera, no se ha llevado a cabo un cambio de proceso, sino un cambio de contexto dentro del mismo proceso.

Si el sistema operativo, al completar su trabajo, determina que el proceso en curso debe continuar ejecutando, entonces se lleva a cabo un cambio de contexto para reanudar el programa interrumpido del proceso en curso. Esta es una de las ventajas clave de este enfoque: Un programa de usuario se interrumpe para emplear alguna rutina del sistema operativo, luego se reanuda y todo se produce sin la penalización de dos cambios de proceso. No obstante, si se determina que va a producirse un cambio de proceso en vez de retomar a! programa que estaba ejecutándose previamente, entonces se pasa el control a una rutina que hace el cambio entre procesos. Esta rutina puede o no ejecutar en el proceso en curso, dependiendo del diseño del sistema. Sin embargo, en algún punto, el proceso en curso debe ponerse en estado de No Ejecución y otro proceso debe designarse como proceso en Ejecución. Durante esta fase, es lógicamente más conveniente considerar que la ejecución tiene lugar fuera de todos los procesos.

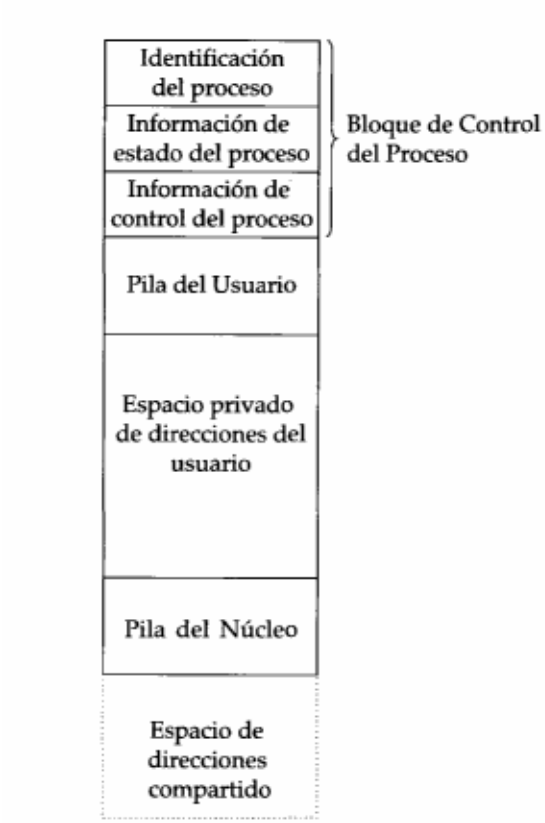


FIGURA 3.14 Imagen del proceso: el sistema operativo ejecuta dentro del proceso de usuario

En cierto sentido, este enfoque del sistema operativo es bastante singular Expuesto de forma simple, cada cierto tiempo, se salvará la información de estado de un proceso, se escogerá otro proceso para ejecutar de entre todos los que estén listos y se cederá el control a dicho proceso. La razón por la que ésta no es una situación arbitraria ni mucho menos caótica es que, durante el tiempo critico, el código que se ejecuta en el proceso de usuario es código compartido por el sistema operativo y no código del usuario. Debido al concepto de modo de usuario y modo del núcleo, el usuario no puede entrometerse ni estorbar a las rutinas del sistema operativo, aún cuando éstas estén ejecutando en el entorno del proceso de usuario. Esto sirve para recordar una vez más que existe una distinción entre los conceptos de proceso y programa y que la relación entre los dos no es de uno a uno. Dentro de un proceso pueden ejecutarse tanto un programa de usuario como uno del sistema operativo y los programas del sistema operativo que ejecutan en los diferentes procesos de usuario son idénticos.

Sistema operativo basado en procesos

Una última alternativa, ilustrada en la figura 3. 13c, es la de implementar el sistema operativo como una colección de procesos del sistema. Al igual que en las otras opciones, el software que forma parte del núcleo ejecutará en modo de núcleo. En este caso, sin embargo, las funciones más importantes del núcleo se organizan en procesos separados. Una vez más, puede haber una pequeña cantidad de código de cambio de procesos que se debe ejecutar fuera de todo proceso.

Este enfoque tiene varias ventajas. Impone unas normas de diseño de programas que promueven el uso de un sistema operativo modular con unas interfaces mínimas y claras entre los módulos. Además, algunas funciones no criticas del sistema operativo se pueden implementas como procesos separados. Por ejemplo, se mencionó anteriormente un programa supervisor que registrara el nivel de utilización de Los recursos (procesador, memoria, canales) y la velocidad de progreso de los procesos de usuario en el sistema. Como este programa no provee un servicio particular a ningún proceso activo, puede ser invocado solamente por el sistema operativo. Como un proceso, la función podrá ejecutar con un nivel de prioridad asignado y ser intercalada con otros procesos bajo el control del distribuidor. Por último, implementar el sistema operativo como un conjunto de procesos es útil en un entorno de multiprocesador o de varios computadores, en el cual algunos de los servicios del sistema operativo pueden enviarse a procesadores dedicados, mejorando así el rendimiento.

Micronúcleos

Un concepto que ha recibido mucha atención últimamente es el de micronúcleo *(microkernel)*. Un micronúcleo es un pequeño núcleo de sistema operativo que proporciona las bases para ampliaciones modulares. Sin embargo, el término es algo confuso, pues hay una serie de cuestiones sobre los micronúcleos cuyas respuestas son diferentes según los distintos equipos de diseño de sistemas operativos. Estas cuestiones son sobre lo pequeño que debe ser el núcleo para ser calificado de micronúcleo, sobre cómo diseñar los controladores *(drivers)* de dispositivos para alcanzar el mejor rendimiento a la vez que sus funciones se independizan del hardware, si las operaciones que no sean del núcleo deben ejecutar en el espacio del núcleo o en el del usuario, y si se debe mantener el código de los subsistemas existentes (por ejemplo, una versión de UNIX) o empezar desde cero.

El enfoque de los micronúcleos fue popularizado por el sistema operativo Mach y sus implementaciones en la línea de computadores de Next. En teoría, el enfoque del núcleo se supone que brinda una gran flexibilidad y modularidad. En la práctica, este beneficio fue hasta cierto punto denegado por el sistema operativo servidor monolítico BSD 4.3 que Next construyó en torno a Mach. Otra aplicación muy conocida del enfoque de micronúcleos es Windows NT, que proclama no solo la modularidad, sino la portabilidad, como los beneficios clave. El núcleo está rodeado por una serie de subsistemas compactos de forma que facilita la tarea de implementar NT en una gran variedad de plataformas. Actualmente, otros productos presumen de tener implementaciones de micronúcleo y este enfoque general de diseño parece que va asentarse en casi todos los computadores personales, estaciones de trabajo y sistemas operativos servidores que se desarrollen en el futuro próximo [VARH94].

La filosofía en que se basa el micronúcleos es que solo las funciones absolutamente esenciales del sistema operativo deben permanecer en el núcleo. Las aplicaciones y los servicios menos esenciales se construyen sobre el micronúcleo. Aunque la línea divisoria de lo que esta adentro y lo que está afuera del micronúcleo varia de un diseño a otro, la característica común es que muchos servicios que tradicionalmente han formado parte del sistema operativo ahora son subsistemas externos que interactúan con el núcleo y con otros **subsistemas; aquí se incluyen** los sistemas de archivo, los servicios de ventana y los servicios de seguridad.

Una vez más aunque los diseños específicos de cada micronúcleo pueden diferir en general la arquitectura de micronúcleo tiende a reemplazar la estratificación tradicional, en ca-

pas verticales, de un sistema operativo, por una horizontal. Las componentes del sistema operativo externas al micronúcleo interactúan una con otra sobre una base común, normalmente a través de mensajes distribuidos a través del micronúcleo. De este modo, el micronúcleo funciona como un distribuidor de mensajes: Valida los mensajes, los pasa entre las componentes y otorga el acceso al hardware. Esta estructura es ideal para entornos de proceso distribuido, ya que el micronúcleo puede pasar mensajes tanto en local como en remoto, sin necesidad de cambios en otras componentes del sistema operativo.

3.4 PROCESOS E HILOS

En la discusión llevada a cabo, se ha presentado el concepto de proceso incluyendo las dos características siguientes:

- *Unidad de propiedad de los recursos:* A cada proceso se le asigna un espacio de direcciones virtuales para albergar a la imagen del proceso y, de cuando en cuando, a! proceso se le puede asignar memoria virtual y otros recursos, tales como canales de E/S, dispositivos de E/S y archivos.

- *Unidad de expedición:* Un proceso es un camino de ejecución (traza) a través de uno o más programas. Esta ejecución puede ser intercalada con la de otros procesos. De este modo, un proceso tiene un estado de ejecución (Ejecución, Listo, etc.) y una prioridad de expedición. La unidad planificada y expedida por el sistema operativo es el proceso.

En la mayoría de los sistemas operativos, estas dos características son, de hecho, la esencia de un proceso. Sin embargo, algunos argumentos pueden convencer de que estas dos características son independientes y que deben ser tratadas de manera independiente por el sistema operativo. Esto se hace así en una serie de sistemas operativos, en particular en algunos sistemas operativos de desarrollo reciente. Para distinguir estas dos características, la unidad de expedición se conoce corno **hilo** *(thread)* o proceso ligero *(lightweight process),* mientras que a la unidad de propiedad de los recursos se le suele llamar **proceso** o **tarea.**[7]

Varios hilos en un solo proceso

La utilización que más salta a la vista del concepto de hilo es la de una disposición en la que pueden existir varios hilos dentro de un mismo proceso. Algo aproximado a este enfoque es lo que se hace en MVS. De una forma más explicita, este enfoque es el asumido por Windows NT, OS/2, la versión que tiene Sun del UNIX y un importante sistema operativo conocido como Mach LAC92, RASH89, TEVA89]. Mach es una evolución ampliada de UNIX que se utiliza en las estaciones Next y que forma la base de la versión de UNIX de la Fundación para el Software Abierto (OSF, *Open Software Foundation).* Este apartado describe el enfoque tornado en Mach; las técnicas utilizadas en Windows NT y MVS se discuten en la sección *3.5.*

[7]Desafortunadamente, incluso este grado de consistencia no se puede mantener. En el MVS, los conceptos de espacio de direcciones y tarea, respectivamente, se corresponden a grandes rasgos con los conceptos de proceso e hilo que se describen en esta sección.

Mach está diseñado específicamente para trabajar en un entorno multiprocesador, aunque también se adapta bien a los sistemas monoprocesadores. En Mach, una tarea se define como la unidad de protección o unidad de asignación de recursos. A las tareas se les asocian los siguientes elementos:

- Un espacio de direcciones virtuales, que contiene la imagen de la tarea
- Acceso protegido a los procesadores, otros procesos (para la comunicación entre procesos), archivos y recursos de E/S (dispositivos y canales)

En una tarea pueden haber uno o más hilos, cada uno con lo siguiente:

- El estado de ejecución del hilo (Ejecución, Listo, etc.)
- El contexto del procesador, que se salva cuando no está ejecutando; una forma de contemplar al hilo es con un contador de programa independiente operando dentro de una tarea
- Una pila de ejecución
- Almacenamiento estático para las variables locales
- Acceso a la memoria y a los recursos de la tarea, que se comparten con todos los otros hilos de la tarea

Los beneficios clave de los hilos se derivan de las implicaciones del rendimiento: Se tarda mucho menos tiempo en crear un nuevo hilo en un proceso existente que en crear una nueva tarea, menos tiempo para terminar un hilo y menos tiempo para cambiar entre dos hilos de un mismo proceso. Por tanto, si hay una aplicación o una función que pueda implementarse como un conjunto de unidades de ejecución relacionadas, es más eficiente hacerlo con una colección de hilos que con una colección de tareas separadas.[8] Algunos estudios llevados a cabo por los desarrolladores de Mach demuestran que la aceleración en la creación de procesos, comparada con la de Las implementaciones de UNIX que no utilizan hilos, está en un factor de 10 [ITEVA87].

Un ejemplo de aplicación que podría hacer uso de hilos es un servidor, como puede ser un servidor de archivos de una red de área local. Cada vez que llegue una solicitud de un nuevo archivo, se puede generar un nuevo hilo para el programa de gestión de archivos. Puesto que el servidor debe manejar muchas solicitudes, se crearán y destruirán muchos hilos en un corto periodo de tiempo. Si el servidor es un multiprocesador, se pueden ejecutar varios hilos de una misma tarea simultáneamente y en diferentes procesadores. Los hilos son también útiles en los monoprocesadores para simplificar la estructura de los programas que lleven a cabo diversas funciones. Otros ejemplos de uso efectivo de los hilos es en las aplicaciones de proceso de comunicaciones [COOP90] y en los supervisores de transacciones.

Otra forma en la que los hilos aportan eficiencia es en la comunicación entre diferentes programas en ejecución. En la mayoría de los sistemas operativos, Ia comunicación entre procesos independientes requiere la intervención del núcleo para ofrecer protección y para proporcionar

[8] Es interesante comentar que han aparecido conceptos similares al mismo tiempo en campos tales como los protocolos de comunicación, la arquitectura de computadores y los sistemas operativos. En la arquitectura de computadores, el enfoque de las computadores con juego reducido de instrucciones (RISC, *Reduced Instruction Set Computer)* ha mejorado la velocidad del procesador, perfeccionando la arquitectura del procesador. En los protocolos de comunicaciOn, la necesidad de altas velocidades de transferencia de datos entre las redes de computadores ha provocado el desarrollo de protocolos ligeros de transporte. Estos dos conceptos se discuten con amplitud en [TAL93a] y [TAL94a], respectivamente.

los mecanismos necesarios para la comunicación. Sin embargo, puesto que los hilos de una misma tarea comparten memoria y archivos, pueden comunicarse entre sI sin invocar al núcleo.

[ETW88] da cuatro ejemplos de uso de los hilos en un sistema de multitarea:

- *Trabajo interactivo* y *de fondo:* Esto se produce en el sentido de la interacción directa con el usuario, no en el de sesiones interactivas y de fondo. Por ejemplo, en un programa de hoja de cálculo, un hilo puede estar visualizando los menús y leyendo la entrada del usuario mientras que otro hilo ejecuta las órdenes y actualiza la hoja de cálculo. Esta medida suele aumentar la velocidad que se percibe de la aplicación, permitiendo que el programa pida la orden siguiente antes de terminar la anterior.
- *Proceso asíncrono:* Los elementos asíncronos del programa se pueden implementar como hilos. Por ejemplo, para protegerse de un corte de alimentación, se puede diseñar un procesador de textos que escriba su buffer de la RAM al disco una vez por minuto. Se puede crear un hilo cuya única tarea sea hacer estas copias de respaldo periódicas y que se planifique directamente con el sistema operativo; no hay necesidad de ningún código superfluo en el programa principal que haga la comprobación de tiempo o que coordine la entrada y la salida.
- *Aceleración de la ejecución:* Un proceso con hilos múltiples puede computar un lote de datos mientras lee el lote siguiente de un dispositivo. En un sistema con multiproceso, varios hilos de un mismo proceso podrán ejecutar realmente a la vez.
- *Organización de los programas:* Los programas que suponen una variedad de actividades o varios orígenes y destinos de entrada y salida pueden hacerse más fáciles de diseñar e implementar mediante hilos.

La planificación y la expedición se llevan a cabo con los hilos; por tanto, la mayor parte de la información de estado relacionada con la ejecución se mantiene en estructuras de datos al nivel de los hilos. Sin embargo, hay varias acciones que afectan a todos lo hilos de una tarea y que el sistema operativo debe gestionar al nivel de las tareas. La suspensión implica la descarga del espacio de direcciones fuera de la memoria principal. Puesto que todos los hilos de una tarea comparten el mismo espacio de direcciones, todos deben entrar en el estado Suspendido al mismo tiempo. De manera similar, la terminación de una tarea supone terminar con todos los hilos dentro de dicha tarea.

Ejemplo — Aldus PageMaker

Como ejemplo del uso de los hilos, considérese la aplicación *Aldus PageMaker,* que se ejecuta para OS/2. PageMaker es una herramienta de composición, diseño y producción de publicaciones. La estructura de hilos, que se muestra en la figura 3.15 RON9O], se escogió para optimizar el grado de respuesta de la aplicación. Siempre hay tres hilos activos: un hilo de gestión de sucesos, un hilo para el dibujo de la pantalla y un hilo de servicio.

En general, OS/2 responde peor en la gestión de las ventanas si hay algún mensaje de entrada que necesita mucho procesamiento. Los principios generales de OS/2 establecen que ningún mensaje puede necesitar más de 1/10 de segundo. Por ejemplo, llamar a una subrutina para imprimir una página mientras se está procesando una orden de impresión podría impedir que el sistema atendiera a algún mensaje posterior para alguna aplicación, lo que ralentiza el rendimiento. Para satisfacer este criterio, las operaciones largas del usuario con PageMaker, como la impresión, la importación de datos y el flujo de textos, se llevan a cabo mediante un hilo de servicio. La inicialización del programa también se lleva a cabo con un

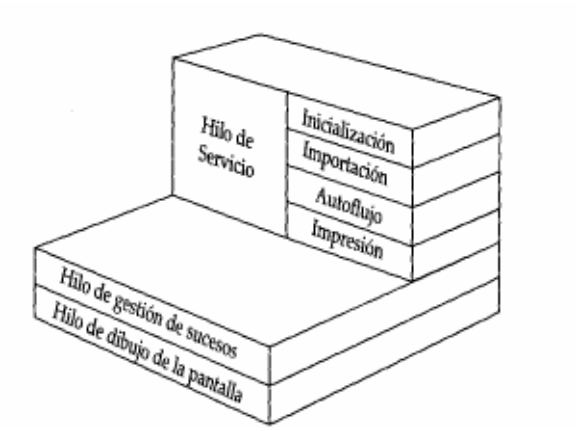


FIGURA 3.15 Estructura de hilos de Aldus PageMaker

hilo de servicio, que aprovecha el tiempo libre mientras que el usuario invoca un dialogo para la creación de un nuevo documento o para abrir un documento existente. *Un* hilo aparte espera nuevos mensajes de sucesos.

La sincronización entre el hilo de servicio y el hilo de gestión de sucesos es complicada porque el usuario puede seguir tecleando o moviendo el ratón, lo que activa el hilo de ge- stión de sucesos, mientras que el hilo de servicios se mantiene ocupado. Si este conflicto se produce, PageMaker filtra estos mensajes y acepta solo algunos básicos, como cambiar el tamaño de las ventanas.

El hilo de servicio envía un mensaje para indicar que ha terminado su tarea. Hasta que esto ocurra, la actividad del usuario en PageMaker está limitada. El programa indica esta restricción inhabilitando los elementos de los menús y visualizando un cursor de "ocupado". El usuario es libre de cambiarse a otra aplicación y, cuando el cursor "ocupado" se traslada a otra ventana, se cambiará por el cursor adecuado para la aplicación.

Se utiliza un hilo independiente para dibujar la pantalla por dos razones:

1. PageMaker no limita el número de objetos que pueden aparecer en una página; por tanto, procesar una petición de dibujo puede exceder fácilmente de la pauta de 1/10 sg.
2. Utilizar un hilo separado permite que el usuario abandone el dibujo. En este caso, cuando el usuario cambia la escala de una página, ésta puede dibujarse de inmediato. El programa es menos sensible si termina visualizando la página en la escala anterior y después la vuelve a dibujar por completo con la nueva escala.

El desplazamiento dinámico *(dynamic scrolling)* —volver a dibujar la pantalla a medida que el usuario mueve el indicador de desplazamiento— también es posible. El hilo de gestión de sucesos supervisa la barra de desplazamiento y vuelve a dibujar las regias de los márgenes (que se vuelven a dibujar rápidamente, ofreciéndole al usuario una sensación de respuesta inmediata). Mientras tanto, el hilo de dibujo de la pantalla trata constantemente de volver a dibujar la pantalla y así ponerse al corriente.

Implementar el dibujo dinámico sin el uso de varios hilos impone una gran sobrecarga a la aplicación, al obligarla a sondear los mensajes en varios puntos. Los múltiples hilos permiten que se puedan separar las actividades concurrentes en el código de una manera más natural.

Otras estructuras

Como se ha dicho, los conceptos de unidad de asignación de recursos y de unidad de expedición se han englobado tradicionalmente dentro del concepto único de proceso, es decir, como una relación de uno a uno entre hilos y procesos. Recientemente, ha habido un gran interés en disponer de varios hilos dentro de un mismo proceso, es decir, una relación de muchos a uno. Sin embargo, como se muestra en la tabla 3.12, también se han investigado las otras dos combinaciones, las llamadas relaciones de muchos a muchos y las relaciones de uno a muchos.

Relación de muchos a muchos

La idea de tener una relación muchos a muchos entre los kilos y los procesos se ha explorado en el sistema operativo experimental TRIX [SIEB83, WARD8O]. En TRIX, se tienen los conceptos de dominio y de hilo. Un dominio es una entidad estática que consta de un espacio de direcciones y de unos "puertos" a través de los cuales se envían y reciben los mensajes. Un hilo es un camino sencillo de ejecución, con una pila de ejecución, el estado del procesador y la información de planificación.

Al igual que Mach, se pueden ejecutar varios hilos en un solo dominio, lo que aporta el aumento de eficiencia discutido anteriormente. Sin embargo, también es posible que una actividad de un solo usuario o aplicación se pueda llevar a cabo en varios dominios. En tal caso, existirá un hilo que se puede mover de un dominio a otro.

El uso de un mismo hilo en varios dominios parece motivado, más que nada, por el deseo de brindar al programador una herramienta de estructuración. Por ejemplo, considérese un programa que utiliza un subprograma de E/S. En un entorno de multiprogramación que permita la generación de procesos de usuario, el programa principal puede generar un nuevo proceso que se encargue de la E/S y luego continúe ejecutándose. Sin embargo, si el progreso futuro del programa principal depende del resultado de la operación de E/S, el programa principal tendrá que esperar a que termine el programa de E/S. Hay varias formas de implementar esta aplicación:

1. El programa entero se puede implementar como un único proceso. Esta es una solución razonable y sencilla. Hay algunos inconvenientes relativos a la gestión de memoria. El proceso global puede exigir un volumen considerable de memoria principal para ejecutar eficientemente, mientras que el subprograma de E/S requiere un espacio

TABLA 3.12 Relación Entre Hilos y Procesos

Hilos:Procesos	Descripción	Sistemas de Ejemplo
1:1	Cada hilo de ejecución es un 6nico proceso con sus propios recursos y espacio de direcciones.	UNIX System V
M:1	Un proceso define un espacio de direcciones y recursos dinámicos propios. Pueden crearse varios hilos que ejecuten en dicho proceso.	OS/2, MVS, MACH
1:M	Un hilo puede emigrar del entorno de un proceso a otro. Esto permite que un hilo se pueda mover fácilmente entre sistemas distintos.	Ra
M:M	Combina los atributos de los casos M:1 y 1 :M	TRIX

de direcciones relativamente pequeño para el buffer de E/S y para manejar la pequeña cantidad de código del programa. Puesto que el programa de *E/S* se ejecuta en el espacio de direcciones del programa mayor, o bien el proceso entero debe permanecer en memoria principal durante la operación de E/S o bien la operación de E/S está sujeta al intercambio. Este efecto en la gestión de memoria también se produce si el programa principal y el subprograma de E/S se implementaran como hilos del mismo espacio de direcciones.

2. El programa principal y el subprograma de E/S pueden implementarse como dos procesos separados. Esto incurre en la sobrecarga de crear el proceso subordinado. Si las actividades de *E/S* son frecuentes, se puede dejar vivo al proceso subordinado, lo que consume recursos de gestión o bien crear y destruir frecuentemente el subprograma, lo que es ineficiente.
3. Tratar al programa principal y al subprograma de E/S como una actividad única que se debe implementar como un único hilo. Sin embargo, puede crearse un espacio de direcciones (dominio) para el programa principal y otro para el subprograma de E/S. Así pues, el hilo puede moverse entre los dos espacios de direcciones a medida que avanza la ejecución. El sistema operativo puede administrar los dos espacios de direcciones independientemente, sin incurrir en sobrecarga alguna de creación de procesos. Más aún, el espacio de direcciones usado por el subprograma de E/S puede estar compartido también por otros subprogramas sencillos de E/S.

La experiencia de los desarrolladores de TRIX observa el mérito de la tercera opción y demuestra que ésta puede ser la solución más eficaz para algunas aplicaciones.

Relación de uno a muchos

En el campo de los sistemas operativos distribuidos (diseñados para controlar sistemas informáticos distribuidos), ha habido un gran interés en el concepto de hilo, principalmente como una entidad que se puede mover entre los espacios de direcciones.[9] Un ejemplo notable de esta investigación es el sistema operativo Clouds [DASG88] y espacialmente en su núcleo, conocido como Ra [ERN89]. Otro ejemplo es el sistema Emerald UL88].

Un hilo en Clouds es una unidad de actividad desde la perspectiva del usuario. Un proceso es un espacio de direcciones virtuales con un bloque de control de proceso asociado. Al crearse, un hilo comienza a ejecutar en un proceso invocando un punto de entrada a un programa de dicho proceso. Los hilos se pueden mover de un espacio de direcciones a otro, atravesando de hecho las fronteras de una máquina (es decir, se mueven de un computador a otra). Al trasladarse, un hilo debe llevar consigo cierta información, tal como el terminal de control, unos parámetros globales y las guías de planificación (por ejemplo, la prioridad).

El enfoque de Clouds ofrece una forma efectiva de aislar al usuario y al programador de los detalles del entorno distribuido. Una actividad del usuario puede estar representada por un solo hilo y el movimiento de dicho hilo entre las máquinas puede estar dictado por el sistema operativo, debido a razones propias del sistema, tales como la necesidad de acceder a un recurso remoto o equilibrar la carga.

---

[9] El traslado de los procesos o hilos entre espacios de direcciones de máquinas diferentes se ha convertido en un tema candente en los últimos años (véase, por ejemplo, [RTS89a,b]). Este tema se explorará en el capitulo 13.

3.5 EJEMPLOS DE DESCRIPCION Y CONTROL DE PROCESOS

Sistema UNIX, versión V

UNIX utiliza un servicio de procesos simple pero potente, que es muy visible para el usuario. Todos los procesos del sistema, excepto dos procesos básicos, son creados por órdenes de programas del usuario.

Estados de un proceso

Un total de nueve estados de proceso son los reconocidos por el sistema operativo UNIX; éstos están reflejados en La tabla 3.13 y un diagrama de transición de estados se muestra en La figura 3.16. Esta figura es bastante similar ala figura 3.7, con los dos estados de Dormido de UNIX correspondientes a los dos estados de Bloqueado. Las diferencias pueden resumirse rápidamente a continuación:

- UNIX emplea dos estados de Ejecución, que indican si el proceso está ejecutando en modo de usuario o en modo núcleo.
- Se hace una distinción entre los estados: Listo para Ejecutar y en Memoria, frente al estado de Expulsado. Estos dos estados son, básicamente, el mismo, como se indica por la línea de puntos que los une. Se hace la distinción para enfatizar la forma en que se pasa al estado Expulsado. Cuando un proceso esté ejecutándose en modo núcleo (como resultado de una llamada del supervisor, una interrupción de reloj, o una interrupción de E/S), llegara un momento en el que el núcleo termine su trabajo y esté listo para devolver el control al programa de usuario. En este punto, el núcleo puede decidir expulsar el proceso en curso en favor de alguno que esté listo y tenga mayor prioridad. En tal caso, el proceso en curso se pasa al

TABLA 3.13 Estados de un Proceso en UNIX

Ejecución en modo de usuario	Ejecutando en modo usuario.
Ejecución en modo del núcleo	Ejecutando en modo del núcleo.
Listo para ejecutar y en memoria	Listo para ejecutar tan pronto como el núcleo lo planifique.
Dormido y en memoria	No dispuesto para ejecutar hasta que se produzca un suceso; el proceso está en memoria principal.
Listo para ejecutar y descargado	El proceso está listo para ejecutar, pero se debe cargar el proceso en memoria principal antes de que el núcleo pueda planificarlo para su ejecución.
Dormido y descargado	El proceso está esperando un suceso y ha sido expulsado al almacenamiento secundario.
Expulsado	El proceso retorna del modo núcleo al modo usuario pero el n6cleo lo expulsa y realiza un cambio de contexto para para planificar otro proceso.
Creado	El proceso está recién creado y aún no está aún listo para ejecutar.
Zombie	El proceso ya no existe, pero deja un registro para que lo recoja el proceso padre.

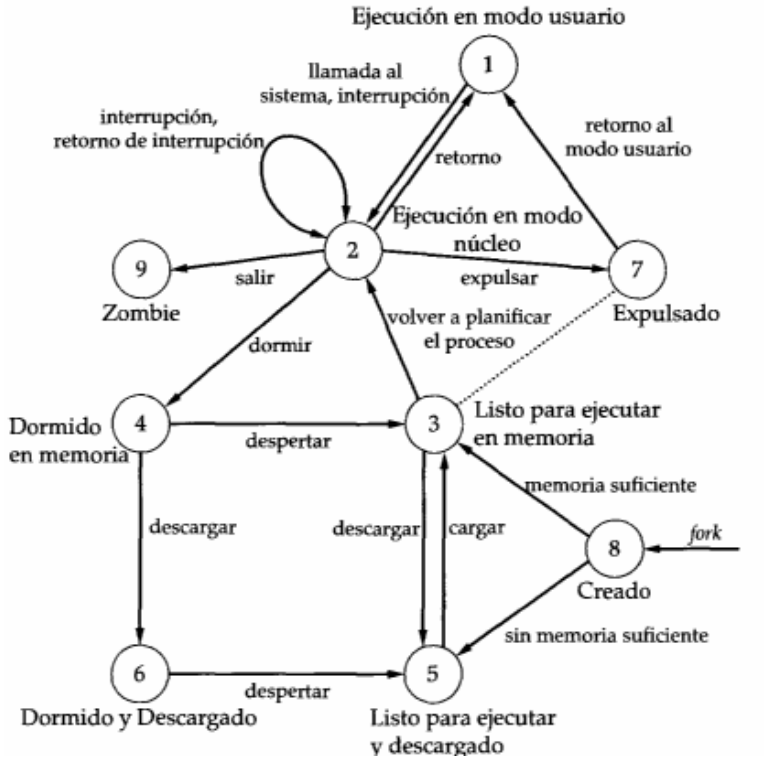


FIGURA 3.16 Diagrama de transición de estados de los procesos en UNIX [ACH86]

estado Expulsado. Sin embargo, a efectos de la expedición, aquellos procesos que están en estado Expulsado y aquellos que están en Listos para Ejecutar y en Memoria forman una sola cola.

La expulsión puede producirse solo cuando va a moverse un proceso del modo núcleo al modo usuario. Mientras un proceso esté ejecutándose en modo núcleo no puede ser expulsado. Esto hace que UNIX sea inadecuado para el procesamiento en tiempo real. El capitulo 9 discute los requisitos del procesamiento en tiempo real.

Hay dos procesos que son únicos en UNIX. El proceso 0 que se crea cuando el sistema arranca; en realidad, es una estructura de datos predefinida que se carga en el momento del arranque y que se denomina proceso de intercambio. Además, el proceso 0 genera el proceso 1, conocido como proceso *init;* todos los demás procesos del sistema tienen al proceso 1 como antepasado. Cuando se conecta un nuevo usuario interactivo en el sistema, es el proceso 1 el que crea un nuevo proceso para dicho usuario. A continuación, el programa de usuario puede crear procesos hijos en forma de árbol, de forma que una aplicación en particular puede constar de un conjunto de procesos relacionados.

Descripción de procesos

Un proceso en UNIX es un conjunto más bien complejo de estructuras de datos que proporcionan al sistema operativo toda la información necesaria para administrarlos y expedirlos.

La tabla 3.14 resume los elementos de la imagen de un proceso, los cuales se organizan en tres partes: contexto del usuario, contexto de los registros y contexto del sistema.

El contexto del usuario contiene los elementos básicos de un programa de usuario y puede ser generado directamente a partir un archivo de código compilado. El programa del usuario se separa en zonas de texto y datos; la zona de texto es solo de lectura y está destinada a albergar las instrucciones del programa. Mientras que el proceso está ejecutando, el procesador utiliza la zona de pila del usuario para las llamadas a procedimientos, los retornos y el paso de parámetros. La región de memoria compartida es una zona de datos que es compartida con otros procesos. Hay una sola copia física de una región de memoria compartida, pero gracias a la memoria virtual, a cada proceso le parece que esta región compartida está en su propio espacio de direcciones.

Cuando un proceso no está ejecutándose, la información del estado del procesador se almacena en la zona de contexto de los registros.

Por último, el contexto del sistema contiene la información restante que el sistema operativo necesita para administrar el proceso. Está formada por una parte estática, que tiene un tamaño fijo y permanece asociada a un proceso durante toda su existencia y una parte dinámica que varia de tamaño durante la vida del proceso. Un elemento de la parte estática es la entrada de la tabla de procesos. Esta es, en realidad, una parte de la tabla de procesos que

TABLA 3.14 Imagen de un proceso en UNIX

Contexto del Usuario	
Texto del Proceso	Instrucciones de máquina ejecutables del programa.
Datos del proceso	Datos del proceso accesibles por el programa.
Pila del usuario	Contiene los argumentos, las variables locales, y los punteros de las funciones que ejecutan en modo usuario.
Memoria compartida	Memoria compartida con otros procesos, utilizada para la comunicación interproceso.
Contexto de los Registros	
Contador de programa	Dirección de la próxima instrucción a ejecutar; puede estar en el espacio de memoria del núcleo o del usuario, de este proceso.
Registro de estado	Contiene el estado del hardware en el momento de la expulsión; del procesador el formato y el contenido dependen del hardware.
Puntero de pila	Señala a la cima de la pila del núcleo o de la pila del usuario, dependiendo del modo de operación en el momento de a expulsión.
Registros de propósito	Dependientes del hardware. general
Contexto del Sistema	
Entrada de la tabla	Define el estado de un proceso; esta información está siempre de procesos accesible para el sistema operativo.
Área U (de usuario)	La información de control del proceso a la que se necesita acceder solo en el contexto del proceso.
Tabla de regiones	Define una traducción de direcciones virtuales a físicas; también contiene de preproceso un campo de permiso que indica el tipo de acceso permitido para el proceso: solo lectura, lectura/escritura o lectura/ejecución.
Pila del núcleo	Contiene los marcos de pila de los procedimientos del núcleo cuando el proceso ejecuta en modo del núcleo.

mantiene el sistema operativo, que dispone de una entrada por proceso. La entrada de la tabla de procesos contiene la información de control accesible para el núcleo en todo momento; de haí que en un sistema de memoria virtual todas las entradas de la tabla de procesos se mantengan en memoria principal. La tabla 3.15 enumera los contenidos de una entrada de La tabla de procesos. La zona de usuario o zona U, contiene información de control adicional del proceso que es necesaria para el núcleo solo cuando está ejecutando en el contexto del proceso. La tabla 3.16 muestra el contenido de esta tabla.

La distinción entre la entrada de la tabla de procesos y la zona U refleja el hecho de que el núcleo de UNIX siempre ejecuta en el contexto de algún proceso. Durante una gran parte del tiempo, el núcleo tendrá que estar tratando con todo lo relativo al proceso. Sin embargo, durante algún tiempo, como cuando el núcleo está ejecutando un algoritmo de planificación para preparar la expedición de otro proceso, el núcleo necesitará acceder a la información de otros procesos.

La tercera parte estática del contexto del sistema es la tabla de regiones de los procesos, que es utilizada por el sistema gestor de memoria. Por último, La pila núcleo es La parte dinámica del contexto del sistema. Esta pila se utiliza cuando el proceso está ejecutando en el modo núcleo. Contiene la información que se debe salvar y restaurar cada vez que se producen interrupciones y llamadas a procedimientos.

Control de procesos

Como ya se mencionó, el Sistema UNIX, versión v, sigue el modelo de la figura 3.13b, en el que la mayor parte del sistema operativo ejecuta en el entorno de un proceso de usuario.

TABLA 3.15 Entradas de la labia de Procesos en UNIX

Estado del Proceso	Estado actual del proceso.
Punteros	Al área U y a la zona de memoria del proceso (texto, datos, pila).
Tamaño del proceso	Permite que el sistema operativo sepa cuánto espacio *asignar* al proceso.
Identificadores	El ID real de usuario identifica al usuario que es responsable del proceso que de usuario se está ejecutando. El ID efectivo de usuario sirve para que un proceso disponga temporalmente de los privilegios asociados a un programa particular; mientras se ejecuta el programa como parte del proceso, éste opera con el ID efectivo de usuario.
Identificadores de proceso	ID del proceso; ID del proceso padre.
Descriptor de suceso	Válido cuando el proceso está en un estado dormido cuando se produce el suceso, el proceso pasa al estado de listo para ejecutar.
Prioridad	Empleada en la planificación del proceso.
Señal	Enumera las señales enviadas a un proceso y que aún no se han tratado.
Temporizadores	Incluye el tiempo de ejecución del proceso, el uso de los recursos del núcleo y un temporizador de usuario empleado para enviar señales de alarma al proceso.
Enlace-P	Puntero al siguiente enlace en la cola de listos (valido si el proceso está listo para ejecutarse).
Estado de la memoria decargado	Indica si la imagen del proceso está en memoria principal o se ha al disco. Si está en memoria, este campo también indica si el proceso se puede descargar o si está temporalmente bloqueado en memoria principal.

TABLA 3.16 Área U de UNIX

Puntero a la Tabla de Procesos	Indica la entrada que corresponde al área U.
Identificadores de usuario	Los IDs de usuario reales y efectivos.
Temporizadores	Registro del tiempo que el proceso (y sus descendientes) dedicaron ejecutando en modo usuario y en modo núcleo.
Vector de tratamiento de señales	Para cada tipo de señal definida en el sistema, indica cOmo reaccionará el proceso al recibirla (terminar, ignorar, ejecutar una función especificada por el usuario).
Terminal de control	Indica el terminal de conexión para el proceso (si procede).
Campo de error	Registra los errores producidos durante una llamada al sistema.
Valor de retorno	Contiene el resultado de las llamadas al sistema.
Parámetros de E/S	Describen la cantidad de datos a transferir, la dirección origen (o destino) del vector de datos del espacio de usuario y los desplazamientos en los archivos para la EJS.
Parámetros de archivos	El directorio actual y el directorio raíz actual describen el entorno del sistema de archivos para el proceso.
Tabla de descriptores	Registra los archivos que el proceso tiene abiertos. de los archivos del usuario
Campos de limites	Restringe el tamaño del proceso y el tamaño de un archivo en que éste puede escribir.
Campos de modos de protección	Mascara con los modos de protección de los archivos creados por el proceso.

Por tanto, hacen falta dos modos, el de usuario y el del núcleo. Algunas partes del sistema operativo, en especial el proceso de intercambio *(swapper),* operan como procesos separados y se les conoce como *procesos del núcleo.*

La creación de procesos en UNTX se hace por medio de la llamada *fork* al núcleo del sistema. Cuando un proceso emite una petición de *fork,* el sistema operativo realiza las siguientes operaciones ACH86J:

1. Asigna una entrada en la tabla de procesos para el nuevo proceso.
2. Asigna un ID único de proceso al proceso hijo.
3. Hace una copia de La imagen del proceso padre, a excepción de la memoria compartida.
4. Incrementa los contadores de los archivos que son propiedad del padre, para reflejar el hecho que hay un nuevo proceso ahora que también es propietario de esos archivos.
5. Pone al proceso hijo en el estado Listo para Ejecutar.
6. Devuelve al proceso padre el número de ID del hijo y devuelve un valor cero al proceso hijo.

Todo este trabajo se acomete en el modo núcleo en el proceso padre. Cuando el núcleo haya terminado con estas funciones, podrá realizar alguna de las siguientes como parte de la rutina distribuidora:

1. Permanecer en el proceso padre. El control vuelve al modo de usuario en el punto de la llamada fork del padre.

2. Transferir eL control al proceso hijo. El proceso hijo comienza ejecutando en el mismo punto del código que el padre, es decir, en el punto de retomo de la llamada fork.

3. Transferir el control a otro proceso. Tanto el padre como el hijo se dejan en el estado Listo para Ejecutar.

Puede que sea difícil hacer ver este método de creación de procesos, porque tanto el padre como el hijo están ejecutando el mismo trozo de código. La diferencia es la siguiente:

Cuando se retorna del *fork,* se pregunta por el parámetro de retomo. Si el valor es cero, entonces se está en el proceso hijo y se puede ejecutar una bifurcación hacia el programa de usuario apropiado para continuar la ejecución. Si el valor es diferente de cero, entonces se está en el proceso padre y puede continuar la línea principal de ejecución.

WINDOWS NT

El diseño de los procesos de Windows NT está dirigido por la necesidad de dar soporte a varios entornos de sistemas operativos. Los procesos aportados por los distintos sistemas operativos son diferentes en varios aspectos, incluyendo los siguientes:

- Cómo se les denomina a los procesos
- Si hay hilos disponibles dentro de los procesos
- Cómo se representan los procesos
- Cómo se protegen los recursos de los procesos
- Qué mecanismos se emplean para la comunicación y la sincronización entre procesos
- Cómo están relacionados los procesos entre sí

Por consiguiente, la estructura nativa de los procesos y de los servicios que brinda el núcleo de NT es relativamente simple y de propósito general, permitiendo a cada subsistema emular la estructura y la funcionalidad particular de los procesos de un sistema operativo. Las características más importantes de los procesos de NT son las siguientes:

- Los procesos de NT se implementan como objetos.
- Un proceso ejecutable puede tener uno o más hilos.
- Los objetos proceso y los objetos hilo tienen capacidades predefinidas de sincronización.
- El núcleo de NT no conserva ninguna relación entre los procesos que crea, incluyendo las relaciones padre-hijo.

La figura 3.17 ilustra la forma en que un proceso se refiere a los recursos que controla o utiliza. La señal de acceso *(access token),* descrita en el capitulo 2, controla si el proceso puede cambiar sus propios atributos. En tal caso, el proceso no tendrá un descriptor abierto de su señal de acceso: Si el proceso intenta abrir un descriptor, el sistema de seguridad determinará si se le permite y, por tanto, si el proceso puede cambiar sus propios atributos.

También tienen que ver con el proceso una serie de bloques que definen el espacio de direcciones virtuales asignado. El proceso no puede modificar directamente estas estructuras, sino que debe depender del administrador de memoria virtual, quien le proporciona al proceso un servicio de asignación de memoria.

Finalmente, el proceso incorpora una tabla de objetos, con los descriptores de otros objetos que conoce. Existe un descriptor para cada hilo del proceso. En la figura se muestra un solo hilo. Además, el proceso tiene acceso a un objeto archivo y a un objeto sección que define una sección de memoria compartida.

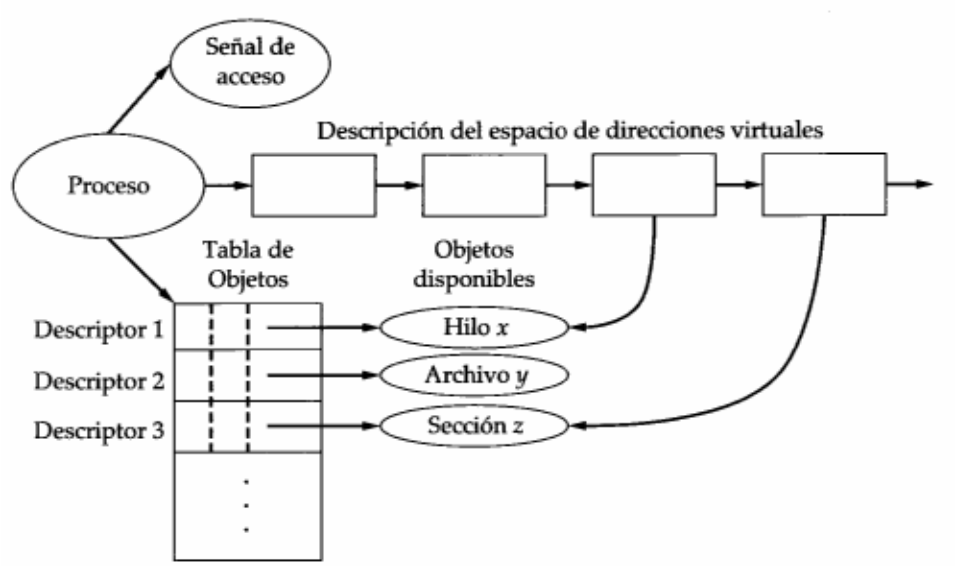


FIGURA 3.17 Un proceso de NT y sus recursos

Objetos proceso y objetos hilo

La estructura orientada a objetos de NT facilita el desarrollo de un servicio de procesos de propósito general. NT hace uso de dos tipos de objetos relacionados con los procesos: los procesos y los hilos. Un proceso es una entidad correspondiente a un trabajo de usuario o una aplicación, que dispone de sus propios recursos, tales como memoria y archivos abiertos. Un hilo es una unidad de trabajo que se puede expedir para su ejecución secuencial y que es interrumpible, de forma que el procesador pueda pasar de un hilo a otro.

Cada proceso de NT está representado por un objeto, cuya estructura general se muestra en la figura 3.1 8a. Cada proceso queda definido por una serie de atributos y encapsula una serie de acciones o servicios que puede desempeñar. Un proceso realizará un servicio cada vez que reciba un mensaje apropiado; la única manera de invocar a dichos servicios es por medio de mensajes al objeto proceso que ofrece el servicio.

Cada vez que NT crea un proceso nuevo, utiliza la clase de objeto o tipo de objeto definido para los procesos de NT como plantilla para generar nuevos casos o instancias de objetos. En el momento de la creación, se asignan unos valores a los atributos. La tabla 3.17 da una definición breve de cada uno de los atributos de un objeto proceso.

Un proceso de NT debe tener, al menos, un hilo para ejecutar. Dicho hilo puede crear entonces otros hilos: En un sistema multiprocesador, pueden ejecutarse en paralelo varios hi-los de un mismo proceso.

La figura 3.1 8b representa la estructura de objeto para un objeto hilo y la tabla 3.18 define los atributos de un objeto hilo: nótese que alguno de los atributos de un hilo se parecen a los de un proceso. En tales casos, el valor del atributo del hilo se obtiene a partir del valor del atributo del proceso. Por ejemplo, la afinidad que tiene el hilo con los procesadores es el conjunto de procesadores de un sistema multiprocesador que pueden ejecutar al hilo; este conjunto es igual o es un subconjunto de la afinidad que tiene el proceso con los procesadores.

FIGURA 3.18 Objeto proceso y objeto hilo en Windows NT

TAB LA 3.17 Atributos de un Objeto Proceso en Windows NT

ID de proceso	Un valor único que identifica al proceso ante el sistema operativo.
Señal de acceso	Un objeto del ejecutor que contiene información de seguridad sobre el usuario conectado al sistema y representado por el proceso.
Prioridad de base	Prioridad de partida para los hilos del proceso.
Afinidad por omisión	Conjunto de procesadores por omisión en los cuales se puede ejecutarse los hilos del proceso.
Limites de cuota	La cantidad máxima de memoria paginada y no paginada del sistema, el espacio para los archivos de paginación y el tiempo de procesador que un proceso de usuario puede emplear.
Tiempo de ejecución	La cantidad total de tiempo que todos los hilos del proceso han consumido en ejecución
Contadores de E/S	Variables que registran el número y el tipo de las operaciones de E/S que han llevado a cabo los hilos del proceso.
Contadores de operaciones de MV	Variables que registran el número y el tipo de las operaciones de memoria virtual que los hilos del proceso han realizado.
Puertos de excepción/depuración	Canales de comunicación entre procesos a los que el administrador de procesos envía mensajes cuando uno de los hilos del proceso provoca una excepción.
Estado de terminación	La razón de la terminación de un proceso.

Nótese que uno de los atributos del objeto hilo es el contexto. Esta información permite que los hilos puedan ser suspendidos y reanudados. Además, se puede alterar el comportamiento de un hilo modificando su contexto cuando esté suspendido.

Hilos múltiples

NT da soporte para la concurrencia entre procesos, pues los hilos de procesos diferentes pueden ejecutarse concurrentemente. Es más, pueden asignarse varios hilos del mismo proceso a distintos procesadores y así ejecutarse concurrentemente. Un proceso con hilos múltiples puede lograr la concurrencia sin la sobrecarga del empleo de varios procesos. Los hilos de un mismo proceso pueden intercambiar información a través de la memoria compartida y tener acceso a los recursos compartidos del proceso.

Un proceso orientado a objetos con hilos múltiples es una forma eficiente de construir aplicaciones servidoras. La figura 3.19 describe el concepto general. Un único proceso servidor puede dar servicio a varios clientes. Cada demanda de un cliente provoca la creación de un nuevo hilo en el servidor.

Soporte para subsistemas

Los servicios generales de procesos e hilos deben dar soporte a las estructuras de hilos y procesos particulares de varios clientes del SO. Es responsabilidad de cada subsistema el aprovechar las características de los procesos e hilos de NT para emular los servicios de procesos e hilos del sistema operativo correspondiente.

Observando cómo es la creación de un proceso, se puede esbozar una idea de cómo se proporciona soporte para hilos y procesos. La creación de un proceso comienza con la petición de un proceso nuevo desde una aplicación del SO. La petición de crear un proceso se emite desde una aplicación hacia el subsistema protegido correspondiente. El subsistema, a su vez, emite una petición de un nuevo proceso al ejecutor de NT. NT crea un objeto proceso

TABLA 3.18 Atributos de un Objeto Hilo en Windows NT

ID del cliente	Valor único que identifica a un hilo cuando llama al servidor.
Contexto hilo	El conjunto de valores de registros y otros datos volátiles que definen el estado de ejecución de un hilo.
Prioridad dinámica	La prioridad de ejecución del hilo en un momento dado.
Prioridad de base	El limite inferior de la prioridad dinámica del hilo.
Afinidad del hilo	El conjunto de procesadores en los que puede ejecutar el hilo, que es con el procesador un subconjunto o coincide con la afinidad con el procesador del proceso que contiene el hilo.
Tiempo de ejecución	El tiempo acumulado que el hilo ha ejecutado en modo usuario y en modo del hilo núcleo.
Estado de alerta	Un indicador que dice si el hilo debe ejecutar una llamada a un procedimiento asíncrono.
Contador	El número de veces que la ejecución del hilo se ha suspendido sin reanudarse de suspensión.
Señal de imitación	Una señal de acceso temporal que permite a un hilo realizar operaciones en nombre de otro proceso (utilizado por los subsistemas).
Puerto de terminación	Un canal de comunicación entre procesos al que el administrador de procesos envía un mensaje cuando el hilo termina (utilizado por los subsistemas).
Estado de terminación	La razón de terminación del hilo.

y le devuelve un descriptor de dicho objeto al subsistema. Ahora, cuando NT crea un proceso, no crea automáticamente un hilo. En los casos de Win32 y OS/2, siempre se crea un hilo junto con el nuevo proceso. Por tanto, para estos sistemas operativos, el subsistema llamará de nuevo al administrador de procesos de NT para crear un hilo para el nuevo proceso, recibiendo de NT el descriptor del hilo. La información correspondiente sobre el hilo y el

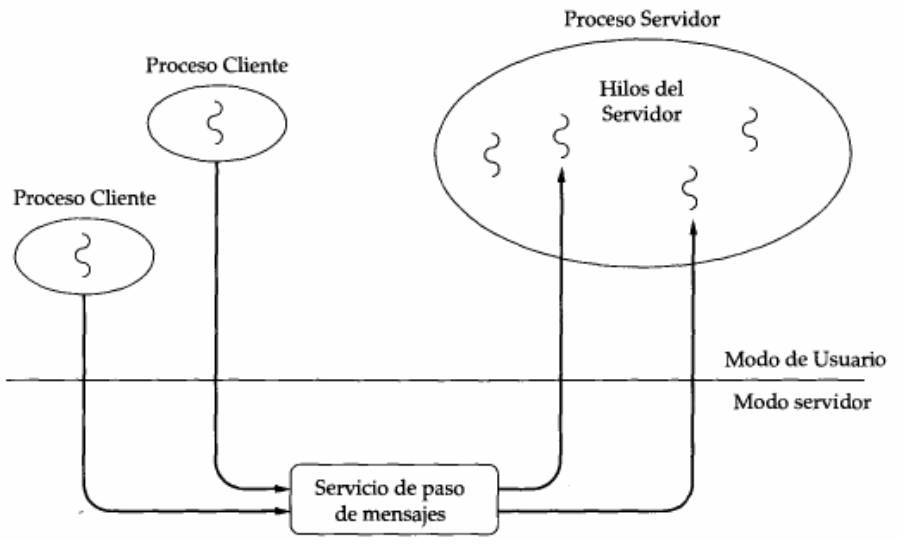


FIGURA 3.19 Servidor de hilos múltiples en Windows NT

proceso es devuelta a la aplicación. Windows de 16 bits y POSIX no tienen soporte de hilos. Por tanto, para estos sistemas operativos el subsistema obtiene de NT un hilo para el nuevo proceso, de forma que se pueda activar el proceso, pero solo se devuelve a la aplicación la información del proceso. El hecho de que el proceso de la aplicación se implemente mediante un hilo no es visible para la aplicación.

Cuando se crea un proceso nuevo en Win32 o en OS/2, éste hereda muchos de sus atributos del proceso creador. Sin embargo, en el entorno de NT, esta creación del proceso se hace indirectamente. Un proceso cliente de aplicación envía su petición de crear un proceso al subsistema del SO; después, un proceso del subsistema enviará a su vez una solicitud de creación de proceso al ejecutor de NT. Puesto que el efecto deseado es que el nuevo proceso herede las características del proceso cliente y no las del proceso servidor, NT permite que el subsistema especifique el padre del nuevo proceso. El nuevo proceso hereda entonces la señal de acceso del padre, los limites de cuota, la prioridad de base y la afinidad por omisión con los procesadores.

MVS

Tareas de MVS

El sistema operativo MYS emplea un entorno de procesos muy estructurado. En MYS, la memoria virtual se divide en grandes regiones llamadas *espacios de direcciones.* A grandes rasgos, a cada aplicación le corresponde un solo espacio de direcciones. Dentro del espacio de direcciones, se pueden generar varias tareas. La tarea de MVS constituye la unidad de expedición.

Normalmente, existen a la vez más de un centenar de espacios de direcciones, cada uno de los cuales puede tener muchas tareas. De este modo, el MVS puede manejar y, a menudo, maneja miles de tareas concurrentemente.

La figura 3.20, basada en otra de Leban y Arnold [LEBA84a], ofrece un ejemplo de la forma en la que se generan varias tareas en un mismo espacio de direcciones. La aplicación da soporte para transacciones de consulta desde una serie de terminales. El programa está dividido en cuatro módulos, un programa principal y tres programas para las tres clases de consulta: de clientes, de pedidos y de productos. Cuando se carga la aplicación en el sistema, el espacio de direcciones comienza con una parte de la memoria reservada para MVS y con el programa principal cargado (figura 3.20a). Después, un usuario introduce una consulta de clientes desde un terminal. Como resultado, el programa principal realiza una solicitud de creación de una nueva tarea para ejecutar el módulo cliente y éste se carga en el espacio de direcciones (figura 3.20b). Mientras esta transacción está en curso, un segundo terminal realiza un pedido y se carga el modulo correspondiente como una tarea separada (figura 3.20c). A continuación, mientras ambas transacciones siguen en curso, un tercer terminal realiza otro pedido. El código del modulo solicitado ya está en el espacio de direcciones y puede ser utilizado por más de una tarea. Sin embargo, cada tarea necesita su propia región privada de memoria para las variables locales y el tratamiento de la pita, como se ilustra en la figura 3.20d.

Explícitamente, el MVS hace uso de solo tres estados para una tarea: Listo, Activo (en ejecución) y Esperando. Sin embargo, se puede descargar a memoria secundaria un espacio de direcciones entero. Por tanto, según se ha venido usando el término, todas las tareas de dicho espacio de direcciones van a quedar suspendidas.

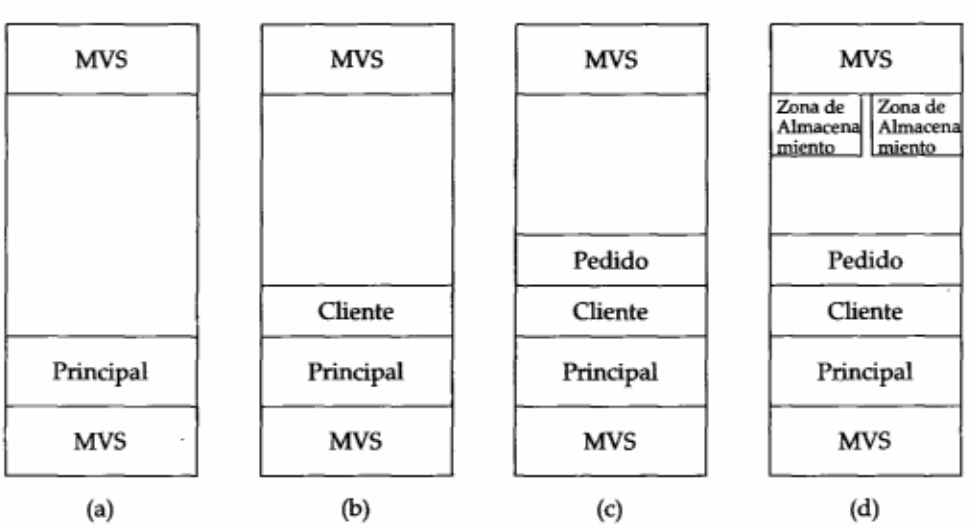


FIGURA 3.20 Esquema del espacio de dirección para una aplicación MVS

Cada tarea está representada por un bloque de control de tarea (TCB, *Task Control Block)*. Además, existe otro tipo de entidad que se puede expedir, la petición de servicio. La *petición de servicio* es una "tarea del sistema", de la que existen dos clases. Algunas tareas del sistema ofrecen servicios a una aplicación específica y se ejecutan en el espacio de direcciones de dicha aplicación; otras están involucradas en las operaciones realizadas entre espacios de direcciones y no ejecutan en ningún espacio de direcciones de usuario, sino en el espacio de direcciones reservado para el sistema. Estas tareas pueden considerarse en conjunto como el núcleo del sistema operativo MVS. Cada tarea del sistema tiene su propio bloque de petición de servicio (SRB, *Service Request Block)* y, cuando llega el momento de expedir una nueva tarea, el distribuidor escoge de entre todos los SRBs y TCBs que estén listos.

De esta forma, en MVS se pueden hallar, a su manera, dos conceptos aparentemente tan modernos como son el de hilos dentro de un proceso y el de contemplar al sistema operativo como un conjunto de procesos (figura 3. 13c).

Estructuras de control de tareas

La tabla 3.19 enumera las estructuras principales de control que utiliza el MYS para administrar las tareas de usuario y las tareas del sistema. En conjunto, estas estructuras agrupan gran parte de lo que se ha venido en llamar bloque de control de proceso. Los bloques globales de petición de servicio se usan para administrar las tareas del sistema que no ejecutan en un espacio de direcciones de usuario. Las estructuras restantes tienen que ver con la gestión del espacio de direcciones. El bloque de control de espacio de direcciones (ASCB, *Address Space Control Block) y* el bloque ampliado de espacio de direcciones (ASXB, *Address Space eXtension Block)* contienen información sobre los espacios de direcciones. La distinción es que MVS necesita la información del ASCB cuando está ejecutando en el espacio de direcciones reservado del sistema, mientras que el ASXB contiene información adicional necesaria solo cuando MVS está tratando con un espacio de direcciones específico de un usuario. Por último, los SRBs y TCBs locales contienen información sobre las entidades que se pueden expedir en un espacio de direcciones.

La figura 3.21 muestra como se organizan estas estructuras y propone la disciplina de expedición aplicada en MVS. Los SRBs globales se mantienen en una cola de prioridades descendentes, en la zona de colas del sistema, que es una región de la memoria principal que no puede descargarse al disco. De este modo, los SRBs globales siempre están disponibles en la memoria principal. Cuando un procesador está disponible, MVS mira primero en esta cola en busca de algún SRB que esté Listo. Si no encuentra ninguno, entonces busca un espacio de direcciones que tenga al menos un TCB o un SRB Listo. Con tal propósito se mantiene una cola de ASCB en la zona de colas del sistema. A cada espacio de direcciones se le asigna un nivel de prioridad global y, por tanto, los ASCBs se pueden ordenar por prioridades descendentes.

TABLA 3.19 Estructuras de Control de Procesos en MVS

Bloque Global de Petición de Servicio (SRB)	Representa a una tarea del sistema que es independiente de cualquier espacio de direcciones del usuario. Incluye: • Dirección del ASCB al que atenderá la rutina • Punto de entrada a la rutina • Dirección de la lista de parámetros que pasar a la rutina • Nivel de prioridad
Bloque de Control de Espacio direcciones (ASCB)	Contiene información del sistema sobre un espacio de direcciones, es decir, la información que necesita MVS del espacio de direcciones cuando no está ejecutando en ningún espacio de direcciones particular. La información incluye: • Punteros a las colas de creación de ASCBs • Si el espacio de dirección ha sido descargado a disco. • Prioridad de expedición • Memoria real y virtual asignada al espacio de direcciones • Número de TCBs listos en este espacio de direcciones • Puntero al ASXB
Bloque Ampliado de Espacio de direcciones (ASXB)	Contiene información sobre un espacio de direcciones que no es de interés global. Incluye: • N6mero de TCBs en el espacio de direcciones • Puntero a la cola de expedición de TCB para este espacio de direcciones • Puntero a la cola de SRB locales para el espacio de direcciones • Puntero a la zona de salvaguarda del gestor de interrupciones
Bloque Local de direcciones (SRB)	Representa a una tarea del sistema que se ejecuta en un espacio de direcciones del usuario. Incluye: • Dirección del ASCB al que atenderá la rutina • Punto de entrada a la rutina • Dirección de la lista de parámetros que pasará a la rutina • Nivel de prioridad
Bloque de Control de Tareas (TCB)	Representa a un programa de usuario en ejecución. Contiene a formación necesaria para administrar una tarea en un espacio de direcciones. Incluye: • Información del estado del procesador • Punteros a los programas que forman parte de la tarea.

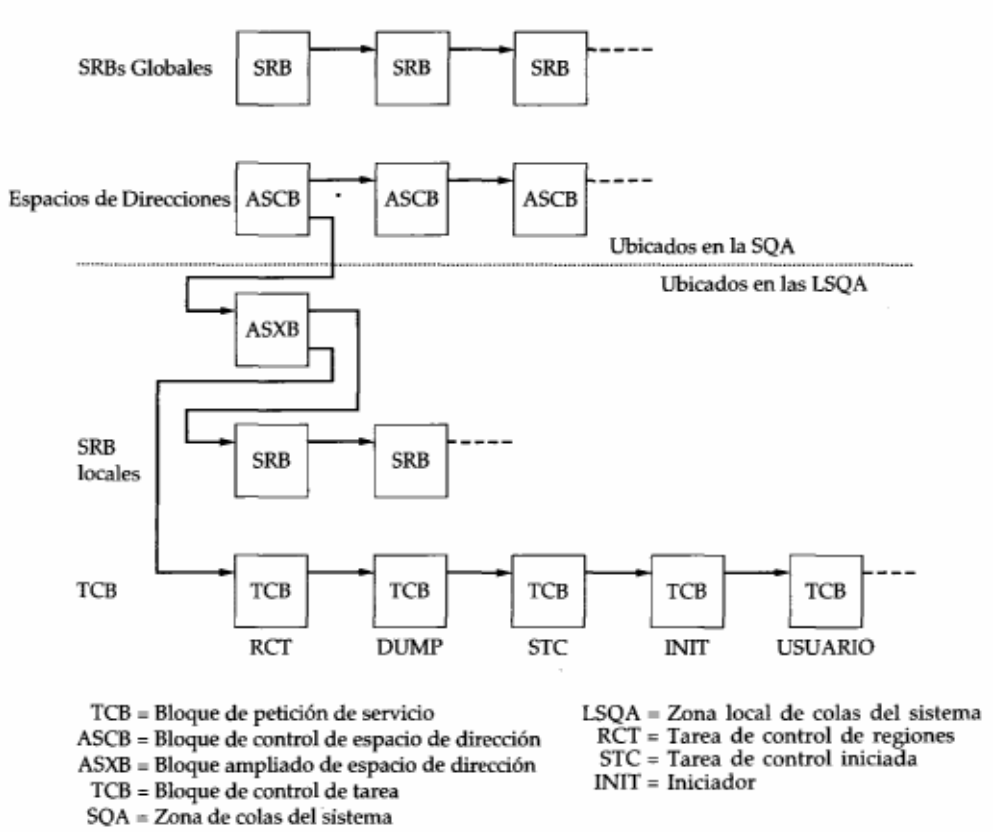


FIGURA 3.21 Colas de planificación de MVS

Una vez que se selecciona un espacio de direcciones, MVS puede trabajar con las estructuras de dicho espacio, conocidas como la zona local de colas del sistema. MVS expide a! SRB que tenga prioridad mayor y, si no hay ninguno, elige al TCB de prioridad mayor. En este ejemplo se muestra la estructura de colas de los TCB para un tote de trabajos, que consta de las Siguientes tareas:

- *Tarea de Control de Regiones (RCT, Region Control Task):* Responsable de la administración del espacio de direcciones en nombre de todas las tareas que ejecutan en él.
- *Volcado (dump):* Responsable de volcar a disco el espacio de direcciones si éste termina de una forma anormal.
- *Tarea de Control Iniciada (STC, Started Control Task):* El TCB del programa que da inicio al espacio de direcciones.
- *Iniciador:* Responsable de cargar el flujo de trabajos por lotes.
- *Tarea de usuario:* La aplicación puede estar formada por una o más tareas de usuario.

La ventaja de dividir la información de control en global y local es que, si es necesario, se puede descargar al disco tanta información de un espacio de direcciones como sea posible, reservando así la memoria principal.

3.6 RESUMEN

La piedra angular de los sistemas operativos modernos es el proceso. La función principal del sistema operativo es crear, administrar y terminar los procesos. Mientras que haya procesos activos, el sistema operativo debe velar por que se le asigne a cada uno un tiempo de ejecución en el procesador, por coordinar sus actividades, gestionar los conflictos en las solicitudes y asignar recursos del sistema a los procesos.

Para llevar a cabo las funciones de gestión de procesos, el sistema operativo debe disponer de una descripción de cada proceso. Cada proceso está representado por una imagen de proceso, que incluye el espacio de direcciones en el que ejecuta el proceso y un bloque de control del proceso. Este último contiene toda la información necesaria para que el sistema operativo administre el proceso, incluyendo su estado actual, los recursos que le han sido asignados, la prioridad y otros datos relevantes.

Durante su existencia, un proceso transita por varios estados. Los más importantes son: Listo, en Ejecución y Bloqueado. Un proceso Listo es aquel que no está ejecutándose en un momento dado, pero que está preparado para ejecutar tan pronto como el sistema operativo lo decida. Un proceso en Ejecución es aquel que está ejecutándose en el procesador. En un sistema multiprocesador puede haber más de un proceso en este estado. Un proceso bloqueado es el que está esperando a que termine algún suceso (como una operación de E/S).

Un proceso en Ejecución puede verse cortado por una interrupción, que es un suceso que se produce fuera del proceso y que es reconocido por el procesador o bien puede ser interrumpido por una llamada del supervisor al sistema operativo. En ambos casos, el procesador lleva a cabo un cambio de contexto y pasa el control a una rutina del sistema operativo:

Después de que ésta haya terminado su trabajo, el sistema operativo podrá reanudar al proceso interrumpido o cambiar a otro proceso.

Algunos sistemas operativos hacen una distinción entre los conceptos de proceso e hilo. El primero se refiere ala propiedad de los recursos y el segundo se refiere ala ejecución de programas. Este enfoque puede llevar a una mejora de la eficiencia y hacer más cómoda la programación.

3.7 LECTURAS RECOMENDADAS

Todos los libros de texto enumerados en la sección 2.6 cubren los temas de este capitulo: Algunas referencias buenas sobre UNIX, Windows NT y MVS son [BACH86], [POWE93] y [KATZ84], respectivamente. [NEHM75] es un estudio interesante sobre los estados de un proceso y sobre las primitivas de un sistema operativo necesarias para la expedición de los procesos.

BACH86 BACH, M. *The Design of the UNIX Operating System.* Prentice Hall, Englewood Cliffs, NJ, 1986.

KATZ84 KATZAN, H. y THARAYIL, D. *Invitation to MVS: Logic and Debugging.* Petrocelli, Nueva York, 1984.

NEHM75 NEHMER, J. “Dispatcher Primitives for the Construction of Operating System Kernels”. *Acta Inform4tica,* vol. 5, *1975*

POWE93 POWELL, J. *Multitask Windows NT.* Waite Group Press, Corte Madera, CA, 1993.

3.8 PROBLEMAS

3.1 La tabla 3.20 muestra los estados de los procesos en el sistema operativo VAX/VMS.

¿Puede darse una justificación para la existencia de tantos estados de espera diferentes?

Decir por qué los estados siguientes no tienen una versión residente y otra descargado: espera por fallo de página, espera por colisión de pagina, espera por un suceso común, espera por pagina libre y espera por recurso.

Dibujar el diagrama de transición de estados e indicar la acción o el acontecimiento que provoca cada transición.

3.2 Pinker y Wear definieron los estados siguientes para un proceso: ejecutando (Ejecución), activo (Listo), bloqueado y suspendido [PINK89]. Un proceso está bloqueado si esta esperando el permiso para usar un recurso y está suspendido si está esperando que termine una operación sobre un recurso que ya ha conseguido. En muchos sistemas operativos, estos dos estados se unen en el estado de Bloqueado, mientras que el estado de Suspendido se define igual que en este capitulo. Compárense los méritos relativos de estas dos definiciones.

3.3 Para el modelo de procesos de siete estados de la figura 3.7b, dibujar un diagrama de colas similar al de la figura 3.6b.

3.4 Considérese el diagrama de transición de estados de la figura 3.7b. Supóngase que es hora de que el sistema operativo expida un proceso y que existen procesos tanto en el estado Listo como en estado de Listo y Suspendido y que almenos un proceso en el estado de Listo y suspendido tiene una prioridad de planificación más alta que cualquiera de los procesos en el estado Listo. Dos políticas extremas son (1) expedir siempre un proceso que esté en estado Listo para así minimizar

TABLA 3.20 Estados **de los Procesos en VAX/VMS**

Estado del Proceso	Condición del Proceso
Ejecutando actualmente	El proceso está ejecutándose.
Computable (residente)	Listo y residente en memoria principal.
Computable (descargado)	Listo, pero descargado en disco.
Espera por fallo de página	El proceso ha hecho referencia a una página que no estaba en memoria principal y tiene que esperar a que se lea la página.
Espera por colisión de página	El proceso ha hecho referencia a una página compartida que espera por fallo de página en otro proceso o ha hecho referencia a una página privada que se está leyendo o escribiendo actualmente.
Espera por un suceso común	Esperando una señal de un suceso compartido (las señales de sucesos son mecanismos de un solo bit para la señalización entre procesos).
Espera por página libre	Esperando a que se añada una página libre a la colección de páginas de memoria principal reservadas para el proceso (el conjunto de trabajo del proceso).
Espera en hibernación (residente)	El proceso se pone a sí mismo en estado de espera.
Espera en hibernación (descargado)	El proceso en hibernación es descargado de la memoria principal.
Espera por suceso local (residente)	El proceso esta en memoria principal y esperando una señal de un suceso local (normalmente la terminación de una E/S).
Espera por un suceso local (descargado)	El proceso en espera del suceso local es descargado de la memoria principal.
Espera suspendida (residente)	El proceso fue puesto en estado de espera por otro proceso.
Espera suspendida (descargado)	El proceso suspendido se descarga de la memoria principal.
Espera por recurso	El proceso está esperando algún recurso variado del sistema.

el intercambio con el disco y (2) dar siempre preferencia al proceso con prioridad más alta, aún cuando esto pueda significar un intercambio innecesario con el disco.

3.5 El sistema operativo VAX/MVS tiene cuatro modos de acceso al procesador para facilitar la protección de los recursos del sistema y su compartición entre los procesos. El modo de acceso determina lo siguiente:

Los privilegios de ejecución de las instrucciones: Las instrucciones que el procesador puede ejecutar.

Los privilegios de acceso a memoria: Las posiciones de la memoria virtual a las que puede acceder la instrucción en curso.

Los cuatro modos son los siguientes:

Núcleo: Ejecuta el núcleo del sistema operativo VMS, lo que incluye a la gestión de memoria, el tratamiento de interrupciones y las operaciones de E/S (figura 2.14).

Ejecutivo: Ejecuta muchas de las llamadas a servicios del sistema operativo, incluyendo las rutinas de gestión de archivos y registros (en disco o cinta).

Supervisor: Ejecuta otros servicios del sistema operativo, tales como las respuestas a las órdenes del usuario.

Usuario: Ejecuta programas de usuario y utilidades tales como compiladores, editores, montadores y depuradores.

Un proceso que esté ejecutando en un modo menos privilegiado tiene que llamar a menudo a procedimientos que ejecutan en un modo más privilegiado. Por ejemplo, un programa de usuario solicita un servicio del sistema operativo. Esta llamada se lleva a cabo mediante una instrucción CHM, que provoca una interrupción y transfiere el control a una rutina con un nuevo modo de acceso. Se ejecuta un retomo con la instrucción REÍ (retomo de excepción o de interrupción).

Algunos sistemas operativos tienen dos modos, de núcleo y de usuario. ¿Cuáles son las ventajas y las desventajas de tener cuatro modos en lugar de dos?

¿Se puede encontrar un motivo para tener aún más de cuatro modos?

3.6 El esquema del VMS discutido en el problema anterior suele conocerse como *estructura de protección en anillo,* como se ilustra en la figura 3.22. En realidad, el sencillo esquema de núcleo/usuario, como el descrito en la sección 3.3, es una estructura de dos anillos. Lisberschatz y Galvin señalaron un problema de este enfoque [SILB94]:

La desventaja principal de la estructura (jerárquica) de anillos es que no permite que se aplique el principio del "tener que conocer". En particular si un objeto tiene que estar accesible en un dominio D_i pero no en un dominio D_j, entonces se debe cumplir que y < ;'. Pero esto significa que todos los segmentos accesibles en D_i también estén accesibles en D_j.

Explicar con claridad el problema mencionado en la cita anterior.

Sugerir una forma para que un sistema operativo con estructura de anillos pueda solucionar este problema.

3.7 La figura 3.6b sugiere que un proceso puede estar en una sola cola de sucesos a la vez.

¿Existe la posibilidad de que se desee permitir que un proceso espere a más de un suceso a la vez? Dar un ejemplo.

En tal caso, ¿cómo se modificaría la estructura de las colas de la figura para dar cabida a esta nueva situación?

3.8 En algunas de los primeros computadores, una interrupción hacía que los valores de los registros se almacenaran en posiciones fijas asociadas con la señal de interrupción. ¿Bajo qué circunstancias es práctica esta técnica? Explicar por qué no es conveniente en general.

3.9 Se ha señalado que dos de las ventajas de usar hilos múltiples dentro de un proceso son: (1) acarrea menos trabajo crear un nuevo hilo en un proceso existente que crear un nuevo proceso y (2) se simplifica la comunicación entre los hilos de un mismo proceso. ¿También se da el caso de que el cambio de contexto entre dos hilos del mismo proceso acarrea menos trabajo que el cambio de contexto entre dos hilos de procesos diferentes?

3.10 En la sección 3.5 se comentó que UNIX no se ajusta a las aplicaciones de tiempo real, ya que

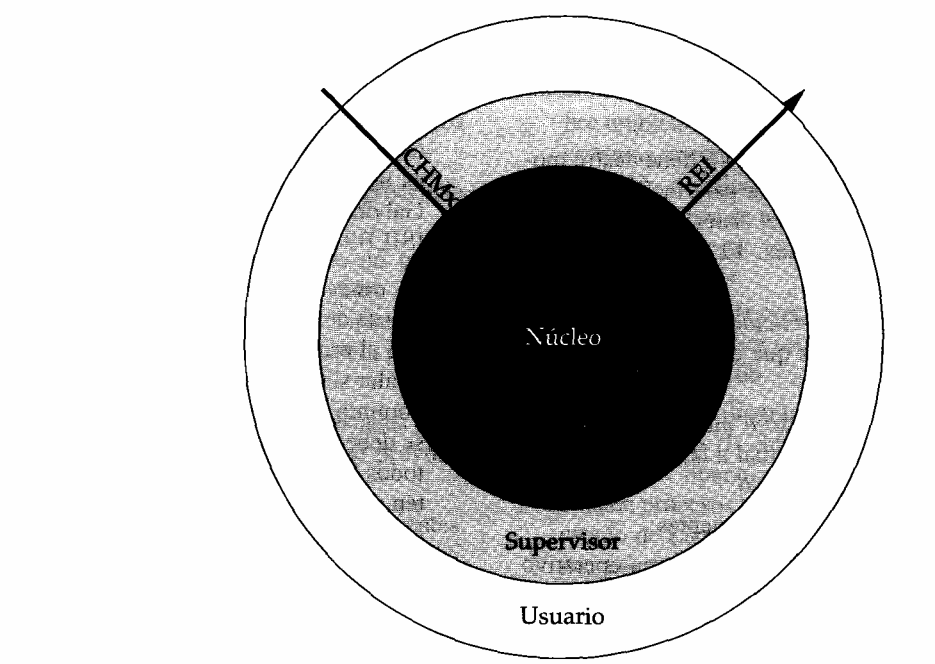


FIGURA 3.22 Modos de acceso del VAX/VMS

un proceso que esté ejecutando en el modo del núcleo no puede ser expulsado. Explíquese con más detalle.

3.11 El número de conceptos que tienen que ver con los procesos de OS/2 puede reducirse de tres a dos, eliminado las sesiones y asociando la interfaz del usuario (teclado, ratón y pantalla) con los procesos. De este modo, habrá un proceso de fondo en cada instante. Para una estructuración mayor, los procesos se pueden dividir en hilos.

¿Qué beneficios se pierden con este enfoque?

Si se decide seguir adelante con esta modificación, ¿dónde hay que asignar los recursos (memoria, archivos, etc.), a nivel de proceso o a nivel de hilo?

3.12 Un proceso completa su trabajo y termina. ¿Qué operaciones básicas son necesarias para limpiar y continuar con otro proceso?

CAPITULO 4

Concurrencia: exclusión mutua y sincronización

Los conceptos claves en los que se basan los sistemas operativos modernos son el de proceso y el de hilo. Los temas fundamentales de diseño de sistemas operativos están relacionados con la gestión de procesos:

- *Multiprogramación:* Es la gestión de varios procesos dentro de un sistema monoprocesador. La mayoría de los computadores personales, estaciones de trabajo, sistemas monoprocesador y sistemas operativos actuales de estas máquinas, tales como Windows 3.0, OS/2 y el Sistema 7 de Macintosh dan soporte a la multiprogramación. Los sistemas operativos como UNIX incorporan multiprogramación de sistemas monoprocesador compartidos.
- *Multiproceso:* Es la gestión de varios procesos dentro de un sistema multiprocesador. Hasta no hace mucho, los sistemas multiprocesador se utilizaban únicamente en grandes sistemas, computadores centrales y minicomputadores. Los sistemas operativos como MVS y VMS dan soporte de multiproceso. Más recientemente, el multiproceso ha empezado a ganarse la aprobación de los servidores y las estaciones de trabajo. Windows NT es un ejemplo de un sistema operativo diseñado para estos entornos.
- *Proceso distribuido:* Es la gestión de varios procesos que ejecutan en sistemas de computadores múltiples y remotas.

La concurrencia es el punto clave de los tres campos anteriores y fundamentales para el diseño de sistemas operativos. La concurrencia comprende un gran número de cuestiones de diseño, incluyendo la comunicación entre procesos, compartición y competencia por los recursos, sincronización de la ejecución de varios procesos y asignación del tiempo de procesador a los procesos. Se vera que estas cuestiones no solo surgen en entornos de multiprocesadores y proceso distribuido, sino incluso en sistemas multiprogramados con un solo procesador.

La concurrencia puede presentarse en tres contextos diferentes:

- *Varias aplicaciones:* La multiprogramación se creó para permitir que el tiempo de procesador de la máquina fuese compartido dinámicamente entre varios trabajos o aplicaciones activas.

- *Aplicaciones estructuradas:* Como ampliación de los principios del diseño modular y la programación estructurada, algunas aplicaciones pueden implementarse eficazmente como un conjunto de procesos concurrentes.
- *Estructura del sistema operativo:* Las mismas ventajas de estructuración son aplicables a los programadores de sistemas y se ha comprobado que algunos sistemas operativos están implementados como un conjunto de procesos.

Debido a la importancia de este tema, hay cuatro capítulos de este libro dedicados a conceptos relacionados con la concurrencia. Este capitulo y el siguiente tratan sobre la concurrencia en los sistemas multiprogramados y de multiproceso. Los capítulos 12 y 13 examinan los elementos de concurrencia relativos al proceso distribuido. Aunque el resto del libro trata de otra serie de temas importantes en el diseño de los sistemas operativos, la concurrencia jugará un papel fundamental en el tratamiento de estos temas.

El capitulo comienza con una introducción al concepto de concurrencia y las implicaciones de la ejecución de varios procesos concurrentes[1]. Se hallará que la exigencia básica de soporte de la concurrencia es la posibilidad de hacer cumplir la exclusión mutua, es decir, la capacidad de prohibir a los demás procesos realizar una acción cuando un proceso haya obtenido el permiso. En la segunda sección de este capitulo, se estudiarán algunos métodos para conseguir la exclusión mutua. Todos ellos son soluciones de software y tienen que emplear una técnica conocida como *espera activa (busy-waiting)*. Debido a la complejidad de estas soluciones y a los inconvenientes de la espera activa, se buscarán soluciones que no necesiten esta espera, con soporte del sistema operativo o aplicadas por los compiladores. Se examinarán tres métodos: semáforos, monitores y paso de mensajes.

Para ilustrar estos conceptos y comparar los métodos presentados en este Capitulo, se usan dos problemas clásicos de concurrencia. En primer lugar, se presentará el problema del productor/consumidor como ejemplo típico. El capitulo termina con el problema de los lectores/escritores.

4.1 PRINCIPIOS GENERALES DE CONCURRENCIA

En un sistema multiprogramado con un único procesador, los procesos se intercalan en el tiempo para dar la apariencia de ejecución simultánea (figura 4.1 a). Aunque no se consigue un proceso paralelo real y aunque se produce una cierta sobrecarga en los intercambios de procesos de un sitio a otro, la ejecución intercalada produce beneficios importantes en la *efi*ciencia del procesamiento y en la estructuración de los programas.

En un sistema con varios procesadores, no solo es posible intercalar los procesos, sino también superponerlos (figura 4.1b).

A primera vista, podría parecer que la intercalación y la superposición representan formas de ejecución muy diferentes y que introducen problemas distintos. De hecho, ambas técni

[1] Porsimplicidad, generalmente se hará referencia ala ejecución concurrente de procesos. De hecho, como se vio en el capitulo anterior, en algunos sistemas la unidad básica de concurrencia es el hilo en vez del proceso.

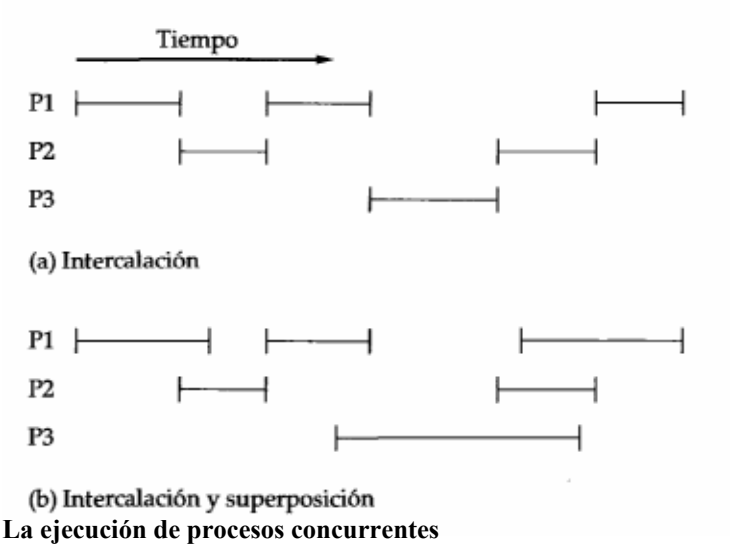


FIGURA 4.1 La ejecución de procesos concurrentes

cas pueden contemplarse como ejemplos de proceso concurrente y ambas plantean los mismos problemas. En el caso de un sistema monoprocesador, los problemas creados por la multiprogramación parten del hecho de que la velocidad relativa de ejecución de los procesos no puede predecirse. Depende de la actividad de otros procesos, de la forma en que el sistema operativo trata las interrupciones y de las políticas de planificación. Así surgen las siguientes dificultades:

1. La compartición de recursos globales está llena de riesgos. Por ejemplo, si dos procesos hacen uso al mismo tiempo de la misma variable global y ambos llevan a cabo tanto lecturas como escrituras sobre la variable, el orden en que se ejecuten las lecturas y escrituras es critico. En el siguiente apartado se ofrece un ejemplo de este problema.
2. Para el sistema operativo resulta difícil asignar los recursos de forma optima. Por ejemplo, el proceso A puede solicitar el uso de un canal de E/S en particular y suspenderse antes de hacer uso del canal. Si el sistema operativo bloquea el canal e impide su uso por parte de otros procesos, se tiene una cierta ineficiencia.
3. Resulta difícil localizar un error de programación porque los resultados no son normalmente reproducibles; véase [LEBL87] y [CARR89] para una discusión sobre este punto.

Todas las dificultades anteriores se presentan también en los sistemas multiprocesador, ya que también en ellos es impredecible la velocidad relativa de ejecución de los procesos. Un sistema multiprocesador debe solucionar, además, los problemas originados por el hecho de que varios procesos puedan estar ejecutando a la vez. Sin embargo, los problemas son, fundamentalmente, los mismos que en el caso anterior. Esto se esclarecerá conforme avance el estudio.

Un ejemplo sencillo

Considérese el siguiente procedimiento:

```
procedure echo;
   var sal, ent: caracter;
begin
      entrada (ent, teclado); sal ent;
      salida (sal, terminal)
end.
```

Este procedimiento muestra los elementos esenciales de un programa que implemente la función de eco *(echo)* de un carácter en el terminal; la entrada se obtiene del teclado, con una pulsación cada vez. Cada carácter de la entrada se almacena en la variable *ent.* Después se transfiere a la variable *sal* y se muestra en la pantalla. Cualquier programa puede llamar a este procedimiento repetidas veces para recibir la entrada del usuario y mostrarla en la pantalla.

Considérese ahora un sistema multiprogramado con un único procesador que da soporte a un solo usuario. El usuario puede pasar de una aplicación a otra y todas las aplicaciones usan el mismo teclado para la entrada y la misma pantalla para la salida. Puesto que cada aplicación tiene que usar el procedimiento *echo,* tiene sentido que éste sea un procedimiento compartido que se cargue en una parte de memoria global para todas las aplicaciones. De este modo, solo se usa una copia del procedimiento y se ahorra espacio.

La compartición de memoria principal entre los procesos es útil para una interacción cercana y eficiente entre los procesos. Sin embargo, esta compartición puede dar lugar a problemas. Considérese la siguiente secuencia:

1. El proceso P1 llama al procedimiento *echo* y es interrumpido inmediatamente después de terminar la función entrada. En este punto, el último carácter leído, x, estará almacenado en la variable *ent.*
2. Se activa el proceso P2 y llama al procedimiento *echo,* que ejecuta hasta el final y muestra un carácter y en la pantalla.
3. Se reanuda el proceso P1. Ahora, el valor de x ha sido sobreescrito en la variable *ent* y, por tanto, se ha perdido. En su lugar, *ent'* contiene el valor y, que se transfiere a la variable *sal* y se visualiza.

Así pues, se pierde el primer carácter y el segundo se visualiza dos veces. La parte esencial de este problema es la compartición de la variable global *ent.* Varios procesos tienen acceso a esta variable. Si un proceso actualiza la variable y es interrumpido, otro proceso puede alterar la variable antes de que el primer proceso pueda usar ese valor. Supóngase, sin embargo, que se impone que, aunque *echo* sea un procedimiento global, solo pueda estar ejecutándolo un proceso cada vez. Entonces, la secuencia anterior quedarla como sigue:

1. El proceso P1 llama al procedimiento *echo* y es interrumpido inmediatamente después de terminar la función entrada. En este punto, el último carácter leído, x, está almacenado en la variable *ent.*
2. Se activa el proceso P2 y llama al procedimiento *echo.* Sin embargo, puesto que P1 permanece aún en el procedimiento, aunque suspendido, P2 se bloquea al entrar al procedimiento. Por tanto, se suspende a P2 mientras espera poder acceder al procedimiento *echo.*

3. Más adelante, el proceso P1 se reanuda y completa la ejecución de *echo.* Se visualiza el carácter correcto, *x*.
4. Cuando P1 abandona *echo,* se retira el bloqueo de P2. Cuando se reanude más tarde P2, la llamada al procedimiento *echo* se hará con éxito.

La lección que hay que aprender de este ejemplo es que es necesario proteger las variables globales compartidas (y otros recursos globales compartidos) y que la única forma de hacerlo es controlar el código que accede a la variable. Si se impone la norma de que solo un proceso puede acceder a *echo* en cada instante y que, una vez en *echo,* el procedimiento debe ejecutar hasta el final antes de estar disponible para otro proceso, no se producirá el tipo de error expuesto antes. El tema principal de este capitulo es cómo imponer esta norma.

Este problema se ha enunciado bajo el supuesto de que se dispone de un sistema operativo multiprogramado con un único procesador. El ejemplo demuestra que los problemas de concurrencia se producen incluso cuando hay un único procesador. En un sistema multiprocesador surge el mismo problema de protección de los recursos compartidos y es válida la misma solución. En primer lugar, supóngase que no hay ningún mecanismo para controlar el acceso a la variable global compartida:

1. Los procesos P1 y P2 están ejecutando, cada uno en un procesador diferente. Ambos invocan al procedimiento *echo.*
2. Se producen los siguientes sucesos; los sucesos de la misma línea tienen lugar en paralelo:

PROCESO P1	PROCESO P2
•	•
entrada (ent, teclado)	•
	entrada (ent, teclado)
sal := ent	sal := ent
salida (sal, terminal)	
•	salida (sal, terminal)
	•

El resultado es que el carácter de entrada de P1 se pierde antes de visualizarse y el carácter de entrada de P2 se visualiza tanto por P1 como por P2. De nuevo, se va a aplicar la norma de que solo un proceso puede estar en *echo* en cada instante. Entonces, se tiene la siguiente secuencia:

1. Los procesos P1 y P2 están ejecutando, cada uno en un procesador diferente. P1 invoca al procedimiento *echo.*
2. Mientras P1 está en el procedimiento *echo,* P2 invoca a *echo.* Puesto que P1 todavía está ejecutando el procedimiento (tanto si está suspendido como si está ejecutando), P2 se bloquea al entrar al procedimiento. Por tanto, P2 se suspende en espera de poder acceder al procedimiento *echo.*
3. Más tarde, el proceso P1 termina la ejecución de *echo,* lo abandona y continúa la ejecución. Inmediatamente después de que P1 salga de *echo,* P2 se reanuda y comienza a ejecutar *echo.*

En el caso de un sistema monoprocesador, La razón por la que se presenta el problema es que una interrupción puede detener la ejecución de instrucciones en cualquier punto de un proceso. En el caso de un sistema multiprocesador, se tiene la misma condición y, además, el problema puede ser causado por dos procesos que estén ejecutando simultáneamente y que intenten ambos acceder a la misma variable global. Sin embargo, la solución para ambos tipos de problemas es la misma: controlar el acceso al recurso compartido.

Labores del sistema operativo

¿Qué elementos de gestión y diseño surgen por causa de la concurrencia? Se pueden enumerar los siguientes:

1. El sistema operativo debe ser capaz de seguir la pista de los distintos procesos activos. Esto lo hace por medio de bloques de control de procesos, como se describió en el capitulo 3.
2. El sistema operativo debe asignar y quitar los distintos recursos a cada proceso activo. Entre estos recursos se incluyen:
   - *Tiempo de procesador:* Es función de la planificación tratada en los capítulos 8 y 9.
   - *Memoria:* La mayoría de los sistemas operativos emplean esquemas de memoria vir-tual. Este tema se abordará en los capítulos 6 y 7.
   - *Archivos:* Tratados en el capitulo 11.
   - *Dispositivos de E/S:* Tratados en el capitulo 10.
3. El sistema operativo debe proteger los datos y los recursos físicos de cada proceso contra injerencias no intencionadas de otros procesos. Esto supone emplear técnicas relativas a la memoria, archivos y dispositivos de E/S, que se estudian en los capítulos correspondientes. En el capitulo 14 se puede encontrar un estudio más general de la protección.
4. Los resultados de un proceso deben ser independientes de la velocidad relativa a la que se realiza la ejecución con respecto a otros procesos concurrentes. Este es el objeto de este capitulo.

Para comprender cómo se puede abordar la independencia de la velocidad, hace falta estudiar las formas en las que los procesos pueden interactuar.

Interacción entre procesos

Se ha señalado que la concurrencia de los procesos puede ser resultado de la multiprogramación de aplicaciones independientes, de aplicaciones con varios procesos y del uso de la estructura de procesos en el sistema operativo. Teniendo en cuenta estas posibilidades, se van a considerar las maneras en que interactúan los procesos.

Es posible clasificar estas interacciones en función del nivel de conocimiento que cada proceso tiene de la existencia de los demás. La tabla 4.1 enumera tres niveles de conocimiento y las consecuencias de cada uno de ellos:

- *Los procesos no tienen conocimiento de los demás:* Estos son procesos independientes que no están pensados para operar juntos. El mejor ejemplo de esta situación es la multi-programación de varios procesos independientes. Estos pueden ser tanto trabajos por lotes como sesiones interactivas o una combinación de ambos. Aunque los procesos no trabajen juntos, el sistema operativo tiene que encargarse de la competencia por los recursos. Por

TABLA 4.1 Interacción entre Procesos

Grado de Conocimiento	Relación	Influencia de un proceso en los otros	Posibles Problemas de Control
Los procesos no tienen conocimiento de los demás	Competencia	• Los resultados de un proceso son independientes de las acciones de los otros Los tiempos de los procesos pueden verse afectados	• Exclusión mutua • Interbloqueo (recursos renovables) • Inanición
Los procesos tienen conocimiento indirecto de los otros (por ejemplo, objetos compartidos)	Cooperación por compartición	• Los resultados de un proceso pueden depender de la información obtenida de los otros • Los tiempos de los procesos pueden verse afectados	• Excusión mutua • Interbloqueo (recursos renovables) • Inanición • Coherencia de datos
Los procesos tiene conocimiento directo de los otros (hay disponibles unas primitivas de comunicaciones)	Cooperación por comunicación	• Los resultados de un procesos pueden depender de la información obtenida de los otros • Los tiempos de los procesos pueden verse afectados	• Interbloqueo (recursos consumibles) • Inanición

ejemplo, dos aplicaciones independientes pueden querer acceder al mismo disco, archivo o impresora. El sistema operativo debe regular estos accesos.

- *Los procesos tienen un conocimiento indirecto de los otros:* Los procesos no conocen necesariamente a los otros por su nombre, pero comparten el acceso a algunos objetos, tales como un buffer de E/S. Estos procesos muestran cooperación para compartir el objeto común.
- *Los procesos tienen un conocimiento directo de los otros:* Los procesos son capaces de comunicarse con los demás por el nombre y están diseñados para trabajar conjuntamente en alguna actividad. Estos procesos también muestran cooperación.

Hay que señalar que las condiciones no estarán siempre tan claramente diferenciadas como se propone en la tabla 4.1. Más bien, los diversos procesos pueden mostrar tanto aspectos de competencia como de cooperación. No obstante, resulta productivo examinar de forma separada cada una de las tres condiciones y determinar sus consecuencias en el sistema operativo.

Competencia entre procesos por los recursos

Los procesos concurrentes entran en conflicto cuando compiten por el uso del mismo recurso. Básicamente, es posible describir esta situación como sigue. Dos o más procesos necesitan acceder a un recurso durante el curso de su ejecución. Cada proceso no es consciente de la existencia de los otros y no se ye afectado por su ejecución. De aquí se obtiene que cada proceso debe dejar tal y como esté el estado de cualquier recurso que utilice. Como ejemplos de recursos se tienen a los dispositivos de E/S, la memoria, el tiempo de procesador y el reloj.

No hay intercambio de información entre los procesos en competencia. Sin embargo, la ejecución de un proceso puede influir en el comportamiento de los procesos que compiten. En particular, si dos procesos desean acceder a un único recurso, el sistema operativo le asignará el recurso a uno de ellos y el otro tendrá que esperar. Por tanto, el proceso a! que se niega el acceso se retrasará. En el peor caso, el proceso bloqueado puede que no consiga nunca acceder al recurso y, por tanto, no terminará con éxito nunca.

En el caso de que haya procesos en competencia, se deben solucionar tres problemas de control. El primero es la necesidad de exclusión mutua. Supóngase que dos o más procesos quieren acceder a un único recurso no compartible, como una impresora. Durante el curso de la ejecución, cada proceso enviará órdenes al dispositivo de E/S, recibiendo información de estado, enviando y/o recibiendo datos. A estos recursos se les llamará *recursos críticos* y la parte del programa que los utiliza se conoce como *sección crítica* del programa. Es importante que solo un programa pueda acceder a su sección crítica en un momento dado. No se puede confiar simplemente en el sistema operativo para aceptar y hacer cumplir esta restricción, pues los requisitos específicos pueden no ser tan obvios. En el caso de la impresora, por ejemplo, se desea que un proceso dado tome el control mientras imprime un archivo completo. De otro modo, se intercalarán las lineas de los procesos en competencia.

Hacer que se cumpla la exclusión mutua crea dos problemas de control adicionales. Uno es el interbloqueo. Considérense dos procesos, P1 y P2 y dos recursos críticos, R1 y R2. Supóngase que cada proceso necesita acceder a ambos recursos para llevar a cabo una parte de su función. En tal caso, es posible que se presente la siguiente situación: el sistema operativo asigna R1 a P2 y R2 a P1. Cada proceso está esperando a uno de los dos recursos. Ninguno liberará el recurso que ya posee hasta que adquiera el otro y ejecute su sección crítica. Ambos procesos están ínter bloqueados.

Un último problema de control es la inanición. Supóngase que tres procesos, P1, P2 y P3, necesitan acceder periódicamente al recurso R. Considérese la situación en la que P1 está en posesión del recurso y tanto P2 como P3 están parados, esperando al recurso. Cuando P1 abandona su sección crítica, tanto P2 como P3 deben poder acceder a R. Supóngase que se le concede el acceso a P3 y que, antes de que termine su sección crítica, P1 solicita acceso de nuevo. Si se le concede el acceso a P1 después de que P3 termine y si P1 y P3 se conceden el acceso repetidamente el uno al otro, se puede negar definidamente a P2 el acceso al recurso, aunque no se produzca una situación de interbloqueo.

El control de la competencia involucra al sistema operativo inevitablemente, porque es el sistema operativo el que asigna los recursos. Además, los procesos deben ser capaces por sí mismos de expresar de algún modo los requisitos de exclusión mutua, como puede ser bloqueando los recursos antes de usarlos. Cualquier solución conlleva alguna ayuda del sistema operativo, como la provisión del servicio de bloqueo. La figura 4.2 expresa en términos abstractos el mecanismo de exclusión mutua. Hay que ejecutar *n* procesos concurrentemente. Cada proceso incluye una sección crítica que opera sobre algún recurso R y una parte restante que no involucra a R. Para hacer cumplir la exclusión mutua, se proporcionan dos funciones: *entrada_crítica* y *salida_crítica.* Cada función toma como argumento el nombre del recurso que es objeto de la competencia. Se hace esperar a cualquier proceso que intente entrar en su sección crítica mientras otro proceso esté en la suya por el mismo recurso.

Quedan por examinar los mecanismos específicos para implementar las funciones *entrada_crítica* y *salida _ crítica.* Por el momento, se aplazará esta cuestión para poder considerar los otros casos de interacción entre procesos.

Cooperación entre procesos por compartición

El caso de cooperación por compartición comprende a los procesos que interactúan con otros sin tener conocimiento explicito de ellos. Por ejemplo, varios procesos pueden tener acceso a variables compartidas, archivos o bases de datos compartidas. Los procesos pueden emplear y actualizar los datos compartidos sin hacer referencia a los otros procesos, pero son conscientes de que estos otros pueden tener acceso a los mismos datos. Así pues, los procesos deben cooperar pasa asegurar que los datos que se comparten se gestionan correctamente. Los mecanismos de control deben garantizar la integridad de los datos compartidos.

Puesto que los datos se guardan en recursos (dispositivos, memoria), también se presentan los problemas de control de exclusión mutua, interbloqueo e inanición. La única diferencia es que se puede acceder a los datos de dos formas distintas, para lectura y para escritura. Solo las operaciones de escritura deben ser mutuamente excluyentes.

Sin embargo, antes que estos problemas, se debe introducir un nuevo requisito: la coherencia de los datos. Un ejemplo sencillo, basado en uno tornado de Raynal, es una aplicación de contabilidad en la que se pueden actualizar varios elementos [RAYN86]. Supóngase que

```
program ExclusiónMutua;
const n =…; (* nürnero de procesos *)
procedure P(i: entero);
begin
   repeat
        entrada _ critica (R);
       <sección critica>;
       salida _ critica (R);
       <resto>
   forever
end;
begin (* programa principal *)
   parbegin
       P(1);
       P(2);
       …
       P(n);
   parend
end.
```

FIGURA 4.2 Exclusion Mutua

dos datos, a y ***b,*** deben cumplir la relación a = ***b.*** Es decir, cualquier programa que actualice un valor debe también actualizar el otro para que se siga cumpliendo la relación. Considérense ahora los siguientes procesos:

```
P1:   a := a + 1;
      b := b + 1;
P2:   b := 2*b;
      a := 2*a;
```

Si al principio el estado es consistente, cada proceso por separado dejará los datos corn-partidos en un estado consistente. Considérese ahora la siguiente ejecución concurrente, en la que los dos procesos respetan la exclusión mutua para cada dato individual (a y ***b):***

```
a  := a + 1;
b  := 2 * b;
b := b + l;
a := 2 * a;
```

Al final de esta secuencia de ejecución, ya no se mantiene la condición *a = b.* El problema puede evitarse declarando la secuencia completa de cada proceso como sección crítica, incluso aunque, estrictamente hablando, ningún recurso crítico se vea involucrado.

De este modo, se ye que el concepto de sección critica es importante en el caso de cooperación por compartición. Se pueden usar las mismas funciones abstractas *entrada _ crítica* y *salida _ crítica* que se estudiaron antes (figura 4.2). En este caso, el argumento de las funciones puede ser una variable, un archivo o cualquier otro objeto compartido. Además, si se usan secciones críticas para garantizar la integridad de datos, puede no haber ningún recurso o variable que se pueda identificar como argumento. En tal caso, se puede considerar al argumento como un identificador que se comparte entre los procesos concurrentes para identificar las secciones criticas que deben ser mutuamente excluyentes.

Cooperación entre procesos por comunicación

En los dos primeros casos expuestos, cada proceso posee su propio entorno aislado, que no incluye a los otros procesos. Las interacciones entre los procesos son indirectas. En ambos casos, existe compartición. En caso de competencia, los procesos están compartiendo recursos sin tener conocimiento de la existencia de otros procesos. En el segundo caso, están compartiendo valores y, aunque cada proceso no tiene conocimiento explicito de los otros, si es consciente de la necesidad de conservar la integridad de los datos. Cuando los procesos cooperan por comunicación, en cambio, los distintos procesos participan en una labor común que une a todos los procesos. La comunicación es una manera de sincronizar o coordinar las distintas actividades.

Normalmente, la comunicación puede caracterizarse por estar formada por mensajes de algún tipo. Las primitivas para enviar y recibir mensajes pueden venir dadas como parte del lenguaje de programación o por el núcleo del sistema operativo.

Puesto que en el paso de mensajes no se comparte nada entre los procesos, no es necesario el control de la exclusión mutua para este tipo de cooperación. Sin embargo, los proble

más de interbloqueo e inanición siguen presentes. Como ejemplo de interbloqueo, dos procesos pueden estar bloqueados, cada uno esperando una comunicación del otro.

Como ejemplo de inanición, considérense tres procesos, P1, P2 y P3, que muestran el comportainiento siguiente. P1 intenta comunicar repetidas veces bien con P2 o con P3 y tanto P2 como P3 intentar comunicar con P1. Puede surgir una secuencia en la que P1 y P2 intercambien información repetidamente, mientras P3 está bloqueado esperando una comunicación desde P1. No hay interbloqueo porque P1 permanece activo, pero P3 sufre de inanición.

Requisitos para la exclusión mutua

El uso adecuado de la concurrencia entre procesos exige la capacidad de definir secciones críticas y hacer cumplir la exclusión mutua. Esto es fundamental para cualquier esquema de proceso concurrente. Cualquier servicio o capacidad que dé soporte para la exclusión mutua debe cumplir los requisitos siguientes:

1. Debe cumplirse la exclusión mutua: Solo un proceso, de entre todos los que poseen secciones críticas por el mismo recurso u objeto compartido, debe tener permiso para entrar en ella en un instante dado.
2. Un proceso que se interrumpe en una sección no crítica debe hacerlo sin estorbar a los otros procesos.
3. Un proceso no debe poder solicitar acceso a una sección crítica para después ser demorado indefinidamente; no puede permitirse el interbloqueo o la inanición.
4. Cuando ningún proceso está en su sección crítica, cualquier proceso que solicite entrar en la suya debe poder hacerlo sin dilación.
5. No se pueden hacer suposiciones sobre la velocidad relativa de los procesos o su número.
6. Un proceso permanece en su sección crítica solo por un tiempo finito.

Hay varias formas de satisfacer los requisitos de exclusión mutua. Una manera es dejar la responsabilidad a los procesos que deseen ejecutar concurrentemente. Así pues, tanto si son programas del sistema como de aplicación, los procesos deben coordinarse unos con otros para cumplir la exclusión mutua, sin ayuda por parte del lenguaje de programación o del sistema operativo. Estos métodos se conocen como *soluciones por software.* Aunque las soluciones por software son propensas a errores y a una fuerte carga de proceso, resulta útil estudiar estos métodos para tener un mejor entendimiento de la complejidad del proceso concurrente. Este tema está cubierto en la sección 4.2. Un segundo método propone el uso de instrucciones de la máquina a tal efecto. Estas tienen la ventaja de reducir la sobrecarga pero, sin embargo, se vera en la sección 4.3 que no son interesantes. El tercer método consiste en dar algún tipo de soporte en el sistema operativo. Más adelante en el capitulo, se examinarán algunas de las técnicas más importantes.

4.2

EXCLUSION MUTUA: SOLUCIONES POR SOFTWARE

Pueden implementarse soluciones de software para los procesos concurrentes que ejecuten en máquinas monoprocesador o multiprocesador con una memoria principal compartida. Normalmente, estas soluciones suponen que existe una exclusión mutua elemental en el ac-

ceso a memoria ([LAMP91]; véase el problema 4.4). Es decir, los accesos simultáneos (lecturas y/o escrituras) a la misma posición de memoria se hacen en serie, por medio de algún tipo de árbitro de memoria, aunque el orden en el que se conceden los accesos no se conoce por adelantado. Aparte de esto, no se requiere ningún soporte del hardware, del sistema operativo o del lenguaje de programación.

Algoritmo de Dekker

Dijkstra [DIJ65] presentó un algoritmo de exclusión mutua para dos procesos que diseñó el matemático holandés Dekker. Según Dijkstra, la solución se desarrolla por etapas. Este método tiene la ventaja de ilustrar la mayoría de los errores habituales que se producen en la construcción de programas concurrentes. A medida que se construya el algoritmo, se emplearán algunas ilustraciones pintorescas tomadas de Ben-Ari para escenificar la acción [BEN82].

Primer intento

Como se mencionó anteriormente, cualquier intento de exclusión mutua debe depender de algunos mecanismos básicos de exclusión en el hardware. El más habitual es la restricción de que solo se puede realizar un acceso a una posición memoria en cada instante. Como metáfora de este arbitrio de la memoria, la figura 4.3 muestra el "protocolo del iglú". Tanto la entrada como el mismo iglú son tan pequeños que solo puede entrar una persona a la vez en el iglú. Dentro, hay una pizarra en la que se puede escribir un único valor.

El protocolo es el siguiente. Un proceso (P0 ó P1) que desee ejecutar su sección crítica entra primero en el iglú y examina la pizarra. Si su número está escrito en ella, el proceso puede abandonar el iglú y continuar con su sección critica. En otro caso, abandona el iglú y se ye obligado a esperar. De vez en cuando, el proceso vuelve a entrar en el iglú para mirar la pizarra. Esta operación la repite hasta que se le permite entrar en su sección crítica. Este procedimiento se denomina *espera activa* porque un proceso frustrado no puede hacer nada productivo hasta que obtiene permiso para entrar en su sección crítica. En su lugar, debe persistir y comprobar periódicamente el iglú; así pues, consume tiempo del procesador (está activo) mientras espera su oportunidad.

Después de que un proceso haya obtenido acceso a su sección crítica y una vez que termine con ella, debe volver al iglú y escribir el número del otro proceso en la pizarra.

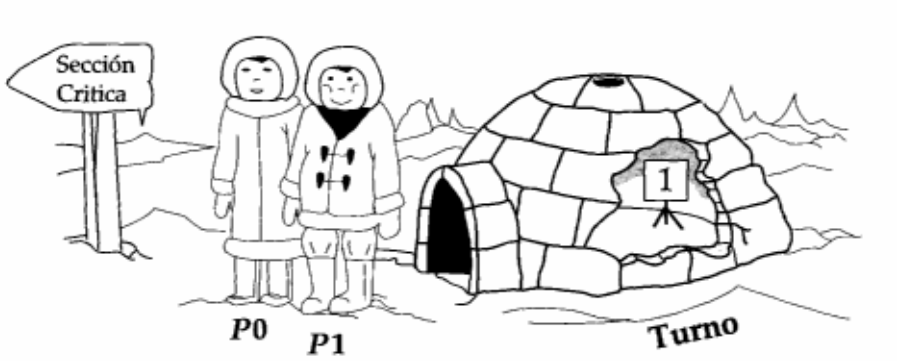


FIG URA 4.3 Un iglú para la exclusión mutua

En términos formales, hay una variable global compartida:

var tumo:0 .. 1;

El programa para los dos procesos:

PROCESO 0	**PROCESO 1**
•	•
•	•
while turno ≠ 0 do { nada };	**whil**e turno ≠ 1 do { nada };
<sección crítica>	<sección crítica>
turno := 1;	turno:= 0;
•	•

Esta solución garantiza el cumplimiento de la exclusión mutua. Hay dos inconvenientes en esta solución. Primero, los procesos deben alternarse de forma estricta en el uso de sus secciones críticas; así pues, el ritmo de ejecución viene dictado por el más lento. Si P0 usa su sección crítica solo una vez por hora, pero P1 quiere usarla con una tasa de 1000 veces por hora, P1 está obligado a adoptar el ritmo de P0. Un problema mucho más seno es que si un proceso falla (por ejemplo, se lo come un oso polar en su camino hacia el iglú), el otro proceso se bloquea permanentemente. Esto es cierto tanto si un proceso falla en su sección critica como fuera de ella.

La estructura anterior es la de una corrutina. Las corrutinas se diseñaron para poder pasar el control de la ejecución de un proceso a otro. Aunque es una técnica de estructuración útil para un solo proceso, no resulta apropiada para dar soporte al proceso concurrente.

Segundo intento

El problema de la primera tentativa es que se almacenaba el nombre del proceso que podía entrar en su sección crítica cuando, de hecho, lo que hace falta es tener información del estado de ambos procesos. En realidad, cada proceso debería tener su propia llave de la sección crítica para que, si un oso polar elimina a uno de ellos, el otro pueda seguir accediendo a su sección crítica. Esta filosofía queda ilustrada en la figura 4.4. Cada proceso tiene ahora su propio iglú y puede mirar ha pizarra del otro, pero no modificarla. Cuando un proceso desea entrar en su sección critica, comprueba periódicamente la pizarra del otro hasta que encuentra escrito en ella "falso", lo que indica que el otro proceso no está en su sección crítica. Entonces, se dirige rápidamente hacia su propio iglú, entra y escribe "cierto" en la pizarra. El proceso puede ahora continuar con su sección crítica. Cuando deja su sección crítica, cambia su pizarra para que ponga *"falso"*.

La variable global compartida es ahora:

var señal: **array** [0..1] of booleano;

que está inicializada con falso. El programa para los dos procesos es:

PROCESO 0	**PROCESO 1**
•	•
•	•
while señal [1] do { nada }*;*	**while** señal [0] do { nada };

FIGURA 4.4 Una solución con dos iglúes a la exclusión mutua

señal [0] := cierto;	señal [1] := cierto;
<sección crítica>	<sección crítica>
señal [0] := falso;	señal [1] := falso;
•	•

Ahora, si uno de los procesos falla fuera de la sección crítica, incluyendo el código para dar valor a las señales, el otro proceso no se queda bloqueado. De hecho, el otro proceso puede entrar en su sección crítica tantas veces como quiera, porque la señal del otro proceso está siempre puesta a falso. Sin embargo, si un proceso falla en su sección crítica, el otro proceso está bloqueado permanentemente.

En realidad, esta solución es, si acaso, peor que el primer intento, pues no siempre se garantiza la exclusión mutua. Considérese la siguiente secuencia:
P0 ejecuta la sentencia **while** y encuentra señal [1] a falso.
P1 ejecuta la sentencia **while** y encuentra señal [0] a falso.
P0 pone señal [0] a cierto y entra en su sección crítica.
P1 pone señal [1] a cierto y entra en su sección crítica.
Puesto que ambos procesos están en sus secciones críticas, el programa es incorrecto. El problema está en que la solución propuesta no es independiente de la velocidad de ejecución relativa de los procesos.

Tercer intento

El segundo intento falla porque un proceso puede cambiar su estado después de que el otro proceso lo ha comprobado pero antes de que pueda entrar en su sección crítica. Quizá se pueda arreglar este problema con un simple intercambio de dos líneas:

PROCESO 0	PROCESO **1**
•	•
•	S
señal [0] cierto;	señal [1] := cierto;

```
while señal do { nada };        while señal do { nada };
<sección critica>               <sección critica>
señal [0] := falso;             señal [1] := falso;
•                               •
```

Como antes, si un proceso falla dentro de la sección crítica, incluyendo el código para dar valor a las señales que controlan el acceso a la sección crítica, el otro proceso se bloquea y si un proceso falla fuera de su sección critica, el otro proceso no se bloquea.

A continuación, se comprobará que la exclusión mutua está garantizada desde el punto de vista del proceso P0. Una vez que P0 ha puesto señal a "cierto", P1 no puede entrar a su sección critica hasta que P0 haya entrado y abandonado la suya. Puede ser que P1 esté todavía en su sección crítica cuando P0 activa su señal. En ese caso, P0 se bloqueará en la sentencia while hasta que P1 deje su sección critica. El mismo razonamiento es aplicable desde el punto de vista de P1.

Esta solución garantiza la exclusión mutua, pero origina un problema más. Si ambos procesos ponen sus señales a "cierto" antes de que ambos hayan ejecutado la sentencia while, cada uno pensará que el otro ha entrado en su sección critica. El resultado es un interbloqueo.

Cuarto intento

En el tercer intento, un proceso fijaba su estado sin conocer el estado del otro. El interbloqueo se produce porque cada proceso puede insistir en su derecho para entrar en la sección crítica; no hay opción para volver atrás desde esta situación. Se puede intentar arreglar esto haciendo que los procesos sean más educados: Deben activar su señal para indicar que desean entrar en la sección crítica, pero deben estar listos para desactivar la señal y ceder la preferencia al otro proceso:

```
PROCESO 0                       PROCESO 1
•                               •
•                               •
señal [0] := cierto;            señal [1] := cierto;
while señal ~1] do              while señal ~0] do
   begin                           begin
      señal [0] := falso;             señal ~1] := falso;
      <espera cierto tiempo>          <espera cierto tiempo>
      señal ~0] := cierto;            señal ~1] := cierto;
   end;                            end;
<sección critica>               <sección critica>
señal [0] := falso;             señal [1] := falso;
•                               •
```

Esta solución se aproxima a la correcta pero todavía es defectuosa. La exclusión mutua aún está garantizada con un razonamiento similar al seguido en el estudio del tercer intento. Sin embargo, considérese la siguiente secuencia de sucesos:

P0 pone señal [0] a cierto
P1 pone señal [1] a cierto
P0 comprueba señal [1]
P1 comprueba señal [0]
P0 pone señal [0] a falso
P1 pone señal [1] a falso
P0 pone señal [0] a cierto
P1 pone señal [1] a cierto

Esta secuencia podría prolongarse indefinidamente y ningún proceso podría entrar en su sección critica. Estrictamente hablando, esto no es un interbloqueo, porque cualquier cambio en la velocidad relativa de los dos procesos romperla este ciclo y permitirla a uno entrar en la sección crítica. Aunque no es probable que esta secuencia se mantenga por mucho tiempo, es una situación posible. Así pues, se rechaza el cuarto intento.

Una solución correcta

Hay que poder observar el estado de ambos procesos, que viene dado por la variable señal. Pero, como demuestra el cuarto intento, esto no es suficiente. De algún modo, se hace necesario imponer algún orden en la actividad de los dos procesos para evitar el problema de "cortesía mutua" que se acaba de observar. La variable *turno* del primer intento puede usarse en esta labor; en este caso, la variable indica qué proceso tiene prioridad para exigir la entrada a la sección critica.

Es posible describir esta solución en términos de iglúes fijándose en la figura 4.5. Ahora hay un iglú "árbitro" con una pizarra llamada "tumo". Cuando P0 quiere entrar en su sección critica, pone su señal a "cierto". A continuación, va y mira la señal de P1. Si ésta está puesta a falso, P0 puede entrar inmediatamente en su sección crítica. En otro caso, P0 va a consultar al árbitro. Si encuentra el *turno* = 0, sabe que es momento de insistir y comprueba periódicamente el iglú de P1. Este otro se percatará en algún momento de que es momento de ceder y escribirá "falso" en su pizarra, permitiendo continuar a P0. Después de que P0 haya ejecutado su sección crítica, pone su señal a *"falso"* para liberar la sección crítica y pone *turno* a 1 para traspasar el derecho de insistir a P1.

La figura 4.6 ofrece una especificación del algoritmo de Dekker. La demostración se deja como ejercicio (ver problema 4.4).

Algoritmo de Peterson

El algoritmo de Dekker resuelve el problema de la exclusión mutua pero con un programa complejo, difícil de seguir y cuya corrección es difícil de demostrar. Peterson [PETE81] desarrollado una solución simple y elegante. Como antes, la variable global señal indica La posición de cada proceso con respecto a la exclusión mutua y la variable global *turno* resuelve los conflictos de simultaneidad. El algoritmo se expone en la figura 4.7.

Se puede demostrar fácilmente que se cumple La exclusión mutua. Considérese el proceso P0. Una vez que ha puesto señal [0] a cierto, P1 no puede entrar en su sección critica. Si P1 está aún en su sección critica, señal [1] = cierto y P0 está bloqueado para entrar en su sección crítica. Por otro lado, se impide el bloqueo mutuo. Supóngase que P0 está bloqueado en su

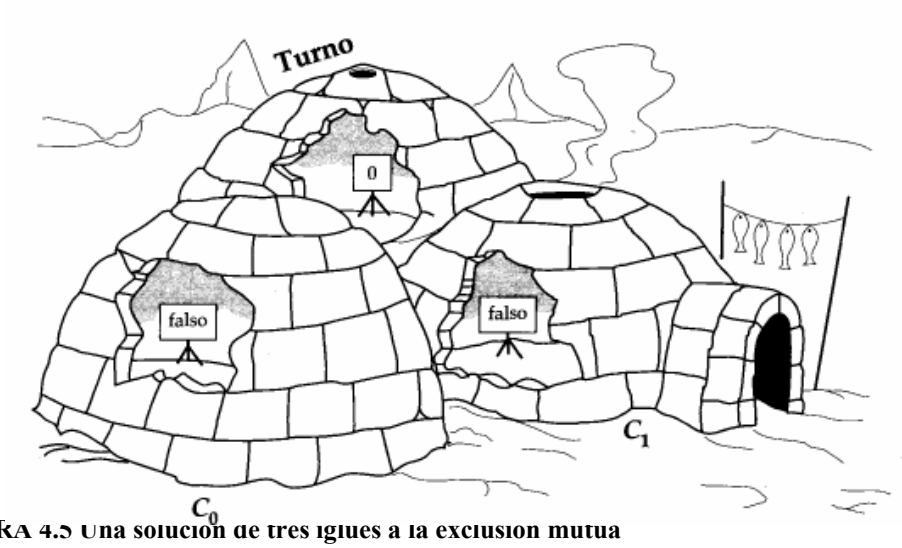


FIGURA 4.5 Una solución de tres iglues a la exclusión mutua

bucle **while.** Esto significa que señal es cierto y *turno* = 1. P0 puede entrar en su sección crítica cuando señal [1] se ponga a falso o cuando *turno* se ponga a 0. Considérense ahora los siguientes casos exhaustivos:

1. P1 no está interesado en entrar en su sección crítica. Este caso es imposible porque implica que señal [l]= falso.
2. P1 está esperando entrar en su sección crítica. Este caso es también imposible porque si *turno* = 1, P1 podrá entrar en su sección critica.
3. P1 entra en su sección crítica varias veces y monopoliza el acceso a ella. Esto no puede pasar porque P1 está obligado a dar a P0 una oportunidad poniendo *turno* a 0 antes de cada intento de entrar en su sección crítica.

Así pues, se tiene una solución simple al problema de la exclusión mutua para dos procesos. Es más, el algoritmo de Peterson se puede generalizar fácilmente al caso de *n* procesos OFR9O].

4.3

EXCLUSION MUTUA: SOLUCIONES POR HARDWARE

Inhabilitación de interrupciones

En una máquina monoprocesador, la ejecución de procesos concurrentes no puede superponerse; los procesos solo pueden intercalarse. Es más, un proceso continuará ejecutando hasta que solicite un servicio del sistema operativo o hasta que sea interrumpido. Por tanto, para garantizar la exclusión mutua, es suficiente con impedir que un proceso sea interrumpido. Esta capacidad puede ofrecerse en forma de primitivas definidas por el núcleo del sistema para habilitar o inhabilitar las interrupciones. Un proceso puede hacer cumplir la exclusión mutua del siguiente modo (compárese con la figura 4.2):

```
var    señal: array [0 .. 1] of booleano;
       turno: 0 .. 1;
procedure P0;
begin
   repeat
      señal [0] := cierto;
      while señal [1] do if turno = 1 then
                         begin
                            señal [0] := falso;
                               while turno = 1 do { nada };
                            señal [0] = cierto;
                         end;
      < sección crítica >;
      turno := 1;
      señal [0] := falso;
      < resto >
   forever
end;
procedure P1;
begin
   repeat
      señal [1] := cierto;
      while señal [0] do if turno = 0 then
                         begin
                            señal [1] := falso;
                               while turno = 0 do { nada };
                            señal [1] = cierto;
                         end;
      < sección crítica >;
      turno := 0;
      señal [1] := falso;
      < resto >
   forever
end;
begin
   señal [0] := falso;
   señal [1] := falso;
   turno := 1;
   parbegin
      P0; P1
   parend
end.
```

FIG URA 4.6 Algoritmo de Dekker

```
var     señal: array [0 .. 1] of booleano;
        turno: 0 .. 1;
procedure P0;
begin
    repeat
        señal [0] := cierto;
        turno := 1;
        while señal [1] and turno = 1 do { nada };
        < sección crítica >;
        señal [0] := falso;
        < resto >
    forever
end;
procedure P1;
begin
    repeat
        señal [1] := cierto;
        turno := 0;
        while señal [0] and turno = 0 do { nada };
        < sección crítica >;
        señal [1] := falso;
        < resto >
    forever
end;
begin
    señal [0] := falso;
    señal [1] := falso;
    turno := 1;
    parbegin
        P0; P1
    parend
end.
```

FIGURA 4.7 Algoritmo de Peterson para dos procesos

repeat
<inhabilitar interrupciones>;
<sección critica>;
<habilitar interrupciones>;
<resto>
forever.

Puesto que la sección critica no puede ser interrumpida, la exclusión mutua está garantizada. Sin embargo, el precio de esta solución es alto. La eficiencia de la ejecución puede

verse notablemente degradada debido a que se limita la capacidad del procesador para intercalar programas. Un segundo problema es que esta técnica no funciona en arquitecturas de multiprocesador. Cuando el sistema tenga más de un procesador, es posible (y habitual) que haya más de un proceso ejecutando al mismo tiempo. En este caso, inhabilitar las interrupciones no garantiza la exclusión mutua.

Instrucciones especiales de la máquina

En configuraciones de multiprocesador, varios procesadores comparten el acceso a una memoria principal común. En este caso, no hay una relación maestro/esclavo, sino que los procesadores funcionan independientemente en una relación de igualdad. No hay un mecanismo de interrupciones entre los procesadores en el que se pueda basar la exclusión mutua.

A nivel del hardware, como se ha mencionado, los accesos a posiciones de memoria excluyen cualquier otro acceso a la misma posición. Con esta base, los diseñadores han propuesto varias instrucciones de máquina que realizan dos acciones atómicamente, tales como leer y escribir o leer y examinar, sobre una misma posición de memoria en un único ciclo de lectura de instrucción. Puesto que estas acciones se realizan en un único ciclo de instrucción, no están sujetas a injerencias por parte de otras instrucciones.

En esta sección se consideran dos de las instrucciones implementadas más habitualmente. Se describen otras en [RAYN86] y [STON90].

Instrucción Comparar y Fijar
La instrucción Comparar y Fijar (TS, *Test and Set)* puede definirse de la siguiente forma:

```
function TS (var i: entero): booleano;
begin
 if  i = 0 then
    begin
          i := I;
          TS := cierto;
    end
else TS := falso
end.
```

La instrucción examina el valor de su argumento, *i.* Si el valor es 0, lo cambia por 1 y devuelve cierto. En otro caso, el valor no se modifica y se devuelve falso. La función Comparar y Fijar se ejecuta inmediatamente en su totalidad, es decir, no está sujeta a interrupciones.

La figura 4.8a muestra un protocolo de exclusión mutua basado en el uso de esta instrucción. Se da un valor 0 inicial a una variable compartida *cerrojo.* El único proceso que puede entrar en su sección crítica es el que encuentre *cerrojo* igual a 0. Todos los demás procesos que intenten entrar pasan a un modo de espera activa. Cuando un proceso abandona su sección critica, vuelve a poner *cerrojo* a 0; en este momento solo uno de los procesos que esperan obtendrá acceso a su sección critica. La elección del proceso depende del que ejecute la instrucción Comparar y Fijar a continuación.

Instrucción Intercambiar
La instrucción Intercambiar se puede definir como sigue:

```
procedure intercambiar (var r: registro; var m: memoria); var temp;
begin
    temp := m;
    m := r;
    r := temp
end.
```

Esta instrucción intercambia el contenido de un registro con el de una posición de memoria. Durante la ejecución de la instrucción, se bloquea el acceso a la posición de memoria de cualquier otra instrucción que haga referencia a la misma posición.

La figura 4.8b muestra un protocolo de exclusión mutua que se basa en el empleo de esta instrucción. Se da un valor 0 inicial a una variable compartida *cerrojo*. El único proceso que puede entrar en su sección critica es el que encuentre *cerrojo* igual a 0. Poniendo *cerrojo* a 1 se excluye a todos los demás procesos de entrar en la sección crítica. Cuando un proceso abandona su sección crítica, vuelve a poner *cerrojo* a 0, permitiendo que otros procesos obtengan acceso a su sección crítica.

```
program exclusión_mutua;
const n=...; (*número de procesos*);
var cerrojo: entero;
procedure P (i: entero);
begin
    repeat
        repeat { nada }until TS (cerrojo);
            < sección crítica >;
            cerrojo := 0;
            < resto >
        forever
end;
begin (* programa principal *)
    cerrojo := 0;
    parbegin
        P(1);
        P(2);
        ...
        P(n);
    parend
end.
```

(a) Instrucción Comparar y Fijar

```
program exclusión_mutua;
const n=...; (*número de procesos*);
var cerrojo: entero;
procedure P (i: entero);
var clave_i: entero;
begin
    repeat
        clave_i := 1;
        repeat intercambiar (clave_i, cerrojo) until clave_i = 0;
        < sección crítica >;
        intercambiar (clave_i, cerrojo);
        < resto >
    forever
end;
begin (* programa principal *)
    cerrojo := 0;
    parbegin
        P(1);
        P(2);
        ...
        P(n);
    parend
end.
```

(b) Instrucción Intercambiar

FIGURA 4.8 Soporte de hardware para la exclusión mutua

Propiedades de las soluciones con instrucciones de máquina

El uso de instrucciones especiales de la máquina para hacer cumplir la exclusión mutua tiene varias ventajas:

- Es aplicable a cualquier número de procesos en sistemas con memoria compartida, tanto de monoprocesador como de multiprocesador.
- Es simple y fácil de verificar.
- Puede usarse para disponer de varias secciones críticas; cada sección crítica puede definirse con su propia variable.

Algunas desventajas importantes son las siguientes:

- Se emplea espera activa. Así pues, mientras un proceso está esperando para acceder a la sección critica, continua consumiendo tiempo del procesador.
- Puede producirse inanición. Cuando un proceso abandona la sección crítica y hay más de un proceso esperando, la selección es arbitraria. Así pues, se podría denegar el acceso a algún proceso indefinidamente.
- Puede producirse interbloqueo. Supóngase la siguiente escena en un sistema monoprocesador. El proceso P1 ejecuta una instrucción especial (sea TS o Intercambiar) y entra en su sección critica. P1 es interrumpido para dar el procesador a P2, que tiene mayor prioridad. Si P2 intenta ahora usar el mismo recurso que P1, se le negará el acceso por el mecanismo de exclusión mutua. De este modo, P2 entrará en un bucle de espera activa. Sin embargo, P1 nunca será expedido porque su prioridad es menor que la de otro proceso listo, esto es, P2.

Debido a estos inconvenientes tanto de las soluciones por software como por hardware, es necesario buscar otros mecanismos.

4.4 SEMAFOROS

Se vuelven a considerar ahora los mecanismos que se usan en el sistema operativo y en los lenguajes de programación para dar soporte a la concurrencia. Esta sección comienza con los semáforos. En las dos siguientes secciones se discutirá sobre los monitores y el paso de mensajes.

El primer gran avance en la resolución de los problemas de procesos concurrentes llegó en *1965,* con el tratado de Dijkstra sobre la cooperación entre procesos secuénciales [DIJK65] Dijkstra estaba interesado en el diseño de un sistema operativo como un conjunto de proce**sos** secuénciales cooperantes y en el desarrollo de mecanismos eficientes y fiables para dar soporte a la cooperación. Los procesos de usuario podrán utilizar estos mecanismos en tanto que el procesador y el sistema operativo los hagan disponibles.

El principio fundamental es el siguiente: Dos o más procesos pueden cooperar por medio de simples señales, de forma que se pueda obligar a detenerse a un proceso en una posición determinada hasta que reciba una señal específica. Cualquier requisito complicado de coordinación puede satisfacerse por medio de la estructura de señales adecuada. Para la señalización, se usan variables especiales llamados *semáforos.* Para transmitir una señal por el Se-

máforo *s,* los procesos ejecutan la primitiva *signal(s).* Para recibir una señal del semáforo *s,* los procesos ejecutan la primitiva *wait(s);* si la señal correspondiente aún no se ha transmitido, el proceso es suspendido hasta que tenga lugar la transmisión[2].

Para lograr el efecto deseado, se pueden contemplar los semáforos como variables que tienen un valor entero sobre el que se definen las tres operaciones siguientes:

1. Un semáforo puede inicializarse con un valor no negativo.
2. La operación *wait* decrementa el valor del semáforo. Si el valor se hace negativo, el proceso que ejecuta el *wait* se bloquea.
3. La operación *signal* incrementa el valor del semáforo. Si el valor no es positivo, se desbloquea a un proceso bloqueado por una operación *wait.*

Aparte de estas tres operaciones, no hay forma de examinar o manipular los semáforos.

La figura 4.9 propone una definición más formal de las primitivas de los semáforos. Las primitivas *wait* y *signal* se suponen atómicas, es decir, no pueden ser interrumpidas y cada rutina puede considerarse como un paso indivisible. En la figura 4.10 se define una versión más limitada, conocida como *semáforo binario.* Un semáforo binario solo puede tomar los valores 0 y 1. En principio, los semáforos binarios son más sencillos de implementar y puede demostrarse que tienen la misma potencia de expresión que los semáforos generales (ver problema 4.14).

Tanto en los semáforos como en los semáforos binarios se emplea una cola para mantener los procesos esperando en el semáforo. La definición no dicta el orden en el que se quitan los procesos de dicha cola. La política más equitativa es la FIFO: el proceso que ha estado bloqueado durante más tiempo se libera de la cola. La única exigencia estricta es que un proceso no debe quedar retenido en la cola de un semáforo indefinidamente porque otros procesos tengan preferencia.

Exclusión mutua

La figura 4.11 muestra una solución sencilla al problema de la exclusión mutua por medio de un semáforo *s* (compárese con la figura 4.2). Sean *n* procesos, identificados por el array P(s). En cada proceso, se ejecuta un *wait(s)* justo antes de su sección crítica. Si el valor de *s* es negativo, se suspende el proceso. Si el valor es 1, se decrementa a 0 y el proceso entra inmediatamente en su sección critica; puesto que *s* ya no es positivo, no se permitirá a ningún otro proceso entrar en su sección critica.

El semáforo se inicializa a 1. De este modo, el primer proceso que ejecute un *wait* podrá entrar inmediatamente en la sección crítica, poniendo el valor de *s* a 0. Cualquier otro proceso que intente entras en la sección critica La encontrará ocupada y se bloqueara, poniendo el valor de *s* a -1. Cualquier número de procesos puede intentar entrar; cada uno de estos intentos infructuosos origina un nuevo decremento del valor de *s.*

Cuando el proceso que entró al principio en su sección critica salga, se incrementará *s* y se eliminará uno de los procesos bloqueados (si los hay) de la cola de procesos asociada al semáforo, poniéndolo en el estado listo. Cuando sea planificado de nuevo por el sistema operativo, podrá entrar en la sección critica.

---

[2] En el articulo original de Dijkstra y en gran parte de la bibliografía, se emplea la letra P para *wait* y la letra V para *signal:* estas son las iniciales de las palabras holandesas que significan comprobar *(proberen)* e incrementar *(verhogen).*

```
type semáforo = record
                    contador: entero;
                    cola: list of proceso
                end;
var s: semáforo;
wait(s):
    s.contador := s.contador – 1;
    if s.contador < 0
        then begin
            poner este proceso en s.cola;
            bloquear este proceso
        end;
signal(s):
    s.contador := s.contador + 1;
    if s.contador ≤ 0
        then begin
            quitar un proceso P de s.cola;
            poner el proceso P en la cola de listos
        end;
```

FIGURA 4.9 Una definición de las primitivas de los semáforos

```
type semáforo binario = record
                            valor: (0, 1);
                            cola: list of proceso;
                        end;
var s: semáforo binario;
waitB(s):
    if s.valor = 1
        then
            s.valor := 0
        else begin
            poner este proceso en s.cola;
            bloquear este proceso
        end;
signalB(s):
    if s.cola está vacia
        then
            s.valor := 1
        else begin
            quitar un proceso P de s.cola;
            poner el proceso P en la cola de listos
        end;
```

FIGURE 4.10 Una definición de las primitivas de los semáforos binarios

El algoritmo de exclusión mutua por medio de semáforos se puede ilustrar con el modelo del iglú (figura 4.12). Además de la pizarra, el iglú tiene ahora un potente congelador. Un proceso entra para realizar un *wait,* decrementa el valor de la pizarra en 1. Si el valor que hay ahora en la pizarra no es negativo, puede entrar en su sección critica. En otro caso, entra en hibernación en el congelador. Esto vacía el interior del iglú, permitiendo que otros procesos entren. Cuando un proceso ha completado su sección crítica, entra al iglú para realizar un *signal,* incrementando el valor de la pizarra en 1. Si el valor no es positivo, libera un proceso del congelador.El programa de la figura 4.11 puede manejar igual de bien el requisito de que más de un proceso pueda entrar en su sección crítica en un instante. Este requisito se satisface simplemente inicializando el semáforo con un valor específico. De este modo, en cualquier instante, el valor de s.contador puede interpretarse como sigue:

- s.contador _ 0: s.contador es el número de procesos que pueden ejecutar un *wait(s)* sin bloquearse (si no se ejecuta ningún *signal(s)* mientras tanto).
- s.contador <0: el valor absoluto de s.contador es el número de procesos bloqueados en *s.cola.*

Problema del productor/consumidor

Ahora se examinará uno de los problemas más habituales a los que se enfrenta la programación concurrente: el problema del productor/consumidor. Se seguirá el desarrollo dado por Ben-Ari [BEN82]. El planteamiento general es el siguiente: Uno o más productores generan cierto tipos de datos (registros, caracteres) y los sitúan en un buffer. Un único consumidor saca elementos del buffer de uno en uno. El sistema está obligado a impedir la superposición de operaciones sobre el buffer. Se considerarán varias soluciones a este problema para ilustrar tanto la potencia como los peligros de los semáforos.

```
program exclusion_mutua;
const n = …; (* número de procesos *); var s: semáforo (:= 1);
procedure P (i: entero);
begin
   repeat
       wait(s)
       <sección critica>;
       signal(s);
       <resto>
  forever
end;
begin (* programa principal *)
   parbegin
       P(1);
       P(2);
       …
       P(n);
   parend
end.
```

FIGURA 4.11 Exclusión mutua por medio de semáforos

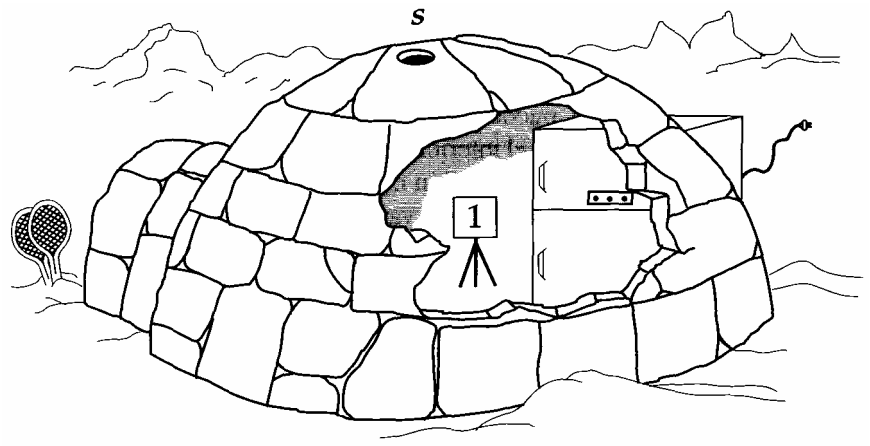


FIGURA 4.12 Un iglú semáforo

Para comenzar, supóngase que el buffer es ilimitado y que consiste en un vector lineal de elementos. En términos abstractos, se puede definir la función del productor como sigue:

productor:
repeat
    producir elemento v;
    b[ent] := v;
    ent := ent +1
forever;

Del mismo modo, la función del consumidor toma la siguiente forma:

consumidor:
repeat
    while ent ≤ sal do { nada };
    w := b[sal];
    sal := sal +1;
    consumir elemento w
forever;

La figura 4.13 muestra la estructura del buffer b. El productor puede generar elementos y almacenarlos en el buffer a su propio ritmo. En cada ocasión, se incrementa un índice (ent) en el buffer. El consumidor procede de forma similar, pero debe estar seguro de que no intenta leer de un buffer vació. Por tanto, el consumidor debe asegurarse de que el productor ha progresado por delante de él (ent> sal) antes de continuar.

Se implementará este sistema por medio de semáforos binarios. La figura 4.14 es el primer intento. En lugar de solucionarlo con los índices ent y sal, simplemente se guarda la cuenta del número de elementos del buffer por medio de la variable entera *n* = ent — sal). El semáforo *s* se usa para hacer cumplir la exclusión mutua; el semáforo *retraso* se usa para obligar al consumidor a esperar si el buffer está vació.

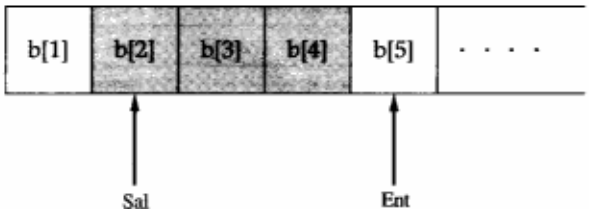


FIGURA 4.13 Buffer ilimitado en el problema del productor/consumidor

Esta solución parece demasiado sencilla. El productor es libre de añadir elementos al buffer en cualquier momento. Realiza un *waitB(s)* antes de añadir uno y un *signalB(s)* después para impedir que el consumidor o cualquier otro productor acceda a! buffer durante la operación. Además, mientras está en la sección critica, el productor incrementa el valor de *n.* Si $n = 1$, el buffer estaba vacío justo antes de añadir este elemento, por lo que el productor ejecutará un *signalB(retraso)* para avisar al consumidor de este hecho. El consumidor comienza esperando mediante *waitB(retraso)* a que se produzca el primer elemento. Entonces toma un elemento y decrementa *n* en su sección critica. Si el productor es capaz de adelantar al consumidor (una situación habitual), el consumidor se bloqueará rara vez en el semáforo retraso porque *n* será normalmente positivo. Así pues, tanto el productor como el consumidor ejecutarán tranquilamente.

Sin embargo, hay un defecto en este programa. Cuando el consumidor haya agotado el buffer, necesita restaurar el semáforo *retraso,* así que se ye obligado a esperar hasta que el productor haya colocado mas elementos en el buffer. Este es el propósito de la sentencia: if $n = 0$ **then** *waitB(retraso).* Considérese la situación representada en la tabla 4.2. En la línea 6, el consumidor falla al ejecutar la operación *waitB.* El consumidor en realidad agota el buffer y pone *n* a 0 (línea 4), pero el productor ha incrementado *n* antes de que el consumidor pueda comprobarlo en la línea 6. El resultado es un *signalB* que no se corresponde con ningún *waitB* anterior. El valor de -1 de *n* en la línea 9 significa que el consumidor ha consumido un elemento del buffer que no existe. No bastarla simplemente con mover la sentencia condicional dentro de la sección critica de consumidor porque esto podría llevar a interbloqueo (por ejemplo, en la línea 3).

Un arreglo al problema es introducir una variable auxiliar que pueda recibir un valor dentro de la sección crítica del consumidor para un uso posterior, como se muestra en la figura 4.15. Un recorrido cuidadoso de la lógica del programa convencerá de que no se puede producir interbloqueo.

Puede obtenerse una solución algo más elegante si se usan semáforos generales (también llamados *semáforos enteros),* tal y como se muestra en la figura 4.16. Ahora la variable *n* es un semáforo. Su valor sigue siendo igual al número de elementos en el buffer. Supóngase ahora que al transcribir este programa, se comete un error y se cambian de lugar las operaciones *signal(s)* y *signal(n).* Podría ser necesario que la operación *signal(s)* se realizara en la sección crítica del productor, sin interrupción del consumidor o de otro proceso. ¿Afectaría esto al programa? No, porque el consumidor debe esperar a ambos semáforos, en cualquier caso, antes de continuar.

Supóngase ahora que se invierten accidentalmente las operaciones *wait(n)* y *wait(s).* Esto producirla un error serio, incluso fatal. Si el consumidor entra en su sección critica cuando

```
program productor_consumidor;
var     n: entero;
        s: semáforo (*binario*) (:= 1);
        retraso: semáforo (*binario*) (:=0);
procedure productor;
begin
    repeat
        producir;
        waitB(s);
        añadir;
        n := n + 1;
        if n=1 then signalB(retraso);
        signalB(s)
    forever
end;
procedure consumidor;
begin
    repeat
        waitB(s);
        tomar;
        n := n – 1;
        signalB(s);
        consumir;
        if n=0 then waitB(retraso)
    forever
end;
begin (*programa principal*)
    n := 0;
    parbegin
        productor; consumidor
    parend
end.
```

FIGURA 4.14 Una solución incorrecta al problema del productor/consumidor con buffer ilimitado por medio de semáforos binarios

el buffer esté vacío (n.contador = 0), ningún productor podrá nunca añadir elementos al buffer y el sistema se ínterbloquea. Este es un buen ejemplo de la sutileza de los semáforos y de la dificultad de construir diseños correctos.

Por último, se va a introducir una restricción nueva y más realista al problema del productor/consumidor: que el buffer sea finito. El buffer se trata como un almacenamiento circular (figura 4.17) y los valores de los apuntadores pueden expresarse como el resto de la división por el tamaño del buffer. Las funciones del productor y del consumidor pueden expresarse como sigue (las variables *ent* y *sal* se inicializan a 0):

TABLA 4.2 Posible situación para el programa de la figura 4.15

	Acción	*n*	*Retraso*
1	Inicialmente	0	0
2	Productor: sección critica	1	1
3	Consumidor: waitB(retraso)	1	0
4	Consumidor: sección critica	0	0
5	Productor: sección critica	1	1
6	Consumidor: if *n* = 0 then waitB(retraso)	1	1
7	Consumidor: sección critica	0	1
8	Consumidor: if *n* = 0 then waitB(retraso)	0	0
9	Consumidor: sección critica	-1	0

```
productor:
repeat
    producir elemento v;
    while ((ent + 1) mod n = sal) do { nada };
    b[ent] := v;
    ent := (ent + 1) mod n
forever;
consumidor:
repeat
while ent = sal do { nada };
    w := b[sal];
    sal := (sal + 1) mod n;
    consumir elemento w
forever;
```

La figura 4.18 muestra una solución por medio de semáforos generales. El semáforo *e* se ha añadido para llevar la cuenta del número de huecos.

Implementación de los semáforos

Como se mencionó anteriormente, es imprescindible que las operaciones *wait y signal* sean implementadas como primitivas atómicas. Una forma obvia es hacerlo en el hardware o en el firmware. Si no se puede, se han propuesto varios esquemas. La esencia del problema es la exclusión mutua: solo un proceso puede manipular un semáforo en un instante, tanto con una operación *wait* como con *signal.* Así pues, se puede usar cualquier esquema de software, tales como los algoritmos de Dekker o Peterson, lo que supone una sobrecarga de proceso sustancial. Otra alternativa es usar uno de los esquemas de soporte del hardware para la exclusión mutua. Por ejemplo, la figura 4.19a muestra el uso de la instrucción Comparar y Fijar. En esta implementación, el semáforo está definido de nuevo con el tipo record, como en la figura 4.9, pero ahora incluye un nuevo componente entero, *s.señal.* Es verdad que esto implica espera activa, pero las operaciones *signal* y *wait* son relativamente cortas, con lo que la cantidad de espera activa que se obtiene será menor.

```
program productor_consumidor;
var     n: entero;
        s: semáforo (*binario*) (:= 1);
        retraso: semáforo (*binario*) (:=0);
procedure productor;
begin
    repeat
        producir;
        waitB(s);
        añadir;
        n := n + 1;
        if n=1 then signalB(retraso);
        signalB(s)
    forever
end;
procedure consumidor;
var m: entero; (*una variable local*);
begin
    waitB(retraso);
    repeat
        waitB(s);
        tomar;
        n := n – 1;
        m := n;
        signalB(s);
        consumir;
        if m=0 then waitB(retraso)
    forever
end;
begin (*programa principal*)
    n := 0;
    parbegin
        productor; consumidor
    parend
end.
```

FIGURA 4.15 Una solución correcta al problema del productor/consumidor con buffer ilimitado por medio de semáforos binarios

En sistemas con monoprocesador, se puede inhibir simplemente las interrupciones durante una operación *wait* o *signal,* como se propone en la figura 4. 19b. De nuevo, la duración relativamente corta de estas operaciones hace que esta solución sea razonable.

```
program productor_consumidor;
var     n: semáforo (:= 0);
        s: semáforo (:= 1);
procedure productor;
begin
    repeat
        producir;
        wait(s);
        añadir;
        signal(s);
        signal(n)
    forever
end;
procedure consumidor;
begin
    repeat
        wait(n);
        wait(s);
        tomar;
        signal(s);
        consumir
    forever
end;
begin (*programa principal*)
    parbegin
        productor; consumidor
    parend
end.
```

FIGURA 4.16 Una solución al problema del productor/consumidor con buffer ilimitado por medio de Semáforos

Problema de la Barbería

Como otro ejemplo del uso de semáforos en la concurrencia, se considera el simple problema de una barberia[3]. Este ejemplo es instructivo porque los problemas encontrados cuando se intenta ofrecer un acceso a medida a los recursos de una barbería son similares a los que se encuentran en un sistema operativo real.

La barbería tiene tres sillas, tres barberos, una zona de espera en la que se pueden acomodar cuatro clientes en un sofá y una sala de espera de pie para el resto de los clientes (figura 4.20). Las medidas contra incendios limitan el número total de clientes en la tienda a 20. En este ejemplo, se supondrá que la barbería procesará, finalmente, 50 clientes.

[3]Estoy en deuda con el profesor Ralph Hilzer, de la Universidad Estatal de California en Chico por proporcionarme este problema.

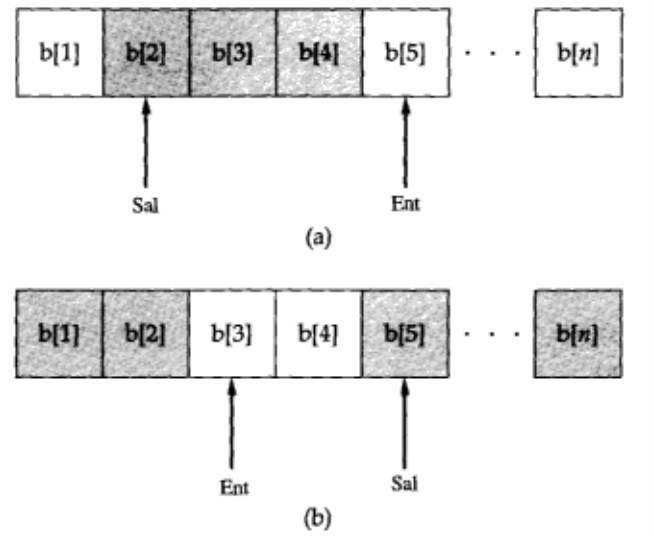


FIGURA 4.17 Buffer circular acotado para el problema del productor/consumidor

Un cliente no entrará en la tienda si su capacidad está al completo con otros clientes. Una vez dentro, el cliente toma asiento en el sofá o permanece de pie si el sofá está completamente ocupado. Cuando un barbero está libre, se atiende al cliente que ha estado más tiempo en el sofá y, si hay clientes de pie, el que ha entrado en la tienda hace más tiempo toma asiento. Cuando finaliza el corte de pelo de un cliente, cualquier barbero puede aceptar el pago, pero, debido a que solo hay una caja registradora, solo se acepta el pago de un cliente cada vez. Los barberos dividen su tiempo entre cortar el pelo, aceptar pagos y dormir en su silla esperando clientes.

Una barbería no equitativa

La figura 4.21 muestra una implementación en la que se emplean semáforos: se enumeran los tres procedimientos, uno junto a otro para ahorrar espacio. Se supone que las colas de todos los semáforos se gestionan con una política FIFO.

El cuerpo principal del programa activa *50* clientes, tres barberos y el proceso cajero. Considérese el propósito y la posición de los distintos operadores de sincronización:

- *Capacidad de la tienda y del sofá:* La capacidad de la tienda y la capacidad del sofá están gobernadas por los semáforos *max_capacidad* y *sofá,* respectivamente. Cada vez que un cliente intenta entrar en la tienda, se decrementa en uno el semáforo *max_capacidad;* cada vez que un cliente sale, se incrementa el semáforo. Si un cliente encuentra la tienda llena, el proceso de ese cliente se suspende en *max_capacidad* a través de la función *wait.* Del mismo modo, las operaciones *wait* y *signal* rodean a las acciones de sentarse y levantarse del sofá.
- *Capacidad de las sillas de barbero:* Hay tres sillas de barbero y se debe tener cuidado de usarlas adecuadamente. El semáforo *silla_barbero* se asegura de que más de tres clientes no intentan ser atendidos al mismo tiempo, intentado evitar la indecorosa situación de un cliente sentado en las rodillas de otro. Un cliente no se levantará del sofá hasta que este li-

```
program buffer_limitado;
const   tamaño_buffer = . . . ;
var     s: semáforo (:= 1);
        n: semáforo (:= 0);
        e: semáforo (:= Tamaño_buffer);
procedure productor;
begin
    repeat
        producir;
        wait(e);
        wait(s);
        añadir;
        signal(s);
        signal(n)
    forever
end;
procedure consumidor;
begin
    repeat
        wait(n);
        wait(s);
        tomar;
        signal(s);
        signal(e);
        consumir
    forever
end;
begin (*programa principal*)
    parbegin
        productor; consumidor
    parend
end.
```

FIGURA 4.18 Una solución al problema del productor/consumidor con buffer acotado por medio de semáforos

re al menos una silla[*wait*(*silla_barbero*)] y cada barbero indicará cuando un cliente deje una silla [*signal*(*silla_barbero*)]. Se garantiza un acceso equitativo a las sillas de barbero por la organización de las colas de semáforo.

El primer cliente que se bloquea es el primero al que se le permite dirigirse hacia una silla libre. Nótese que, en el procedimiento cliente, si se produce un *wait(silla barbero)* después de un *signal(sofá),* cada cliente solo podrá sentarse brevemente en el sofá y después permanecer en fila en las sillas de barbero, creando así un atasco y dejando a los barberos con poca libertad de acción.

- *Asegurarse de que los clientes están en la silla del barbero:* El semáforo *cliente listo* indica a un barbero que estuviera durmiendo que un cliente acaba de sentarse en una silla. Sin este semáforo, un barbero nunca domaría, sino que empezaría a cortar el pelo tan pronto como un cliente dejase la silla; si el cliente nuevo no llega a sentarse, el barbero cortaría el aire.

```
wait(s):
    repeat {nada} until TS(s.señal);
    s.contador := s.contador – 1;
    if s.contador < 0
        then begin
            poner este proceso en s.cola;
            bloquear este proceso (debe poner s.señal a 0)
        end
    else s.señal := 0;
signal(s):
    repeat {nada} until TS(s.señal);
    s.contador := s.contador + 1;
    if s.contador ≤ 0
        then begin
            quitar un proceso P de s.cola;
            poner P en cola de listos
        end;
    s.señal := 0;
```

(a) Instrucción TS

```
wait(s):
    inhabilitar interrupciones;
    s.contador := s.contador –1;
    if s.contador < 0
        then begin
            poner este proceso en s.cola;
            bloquear este proceso y habilitar interrupciones
        end
    else habilitar interrupciones;
signal(s):
    inhabilitar interrupciones;
    s.contador := s.contador + 1;
    if s.contador ≤ 0
        then begin
            quitar un proceso P de s.cola;
            poner P en cola de listos
        end;
    habilitar interrupciones;
```

(b) Interrupciones

FIGURA 4.19 Dos posibles implementaciones de los semáforos

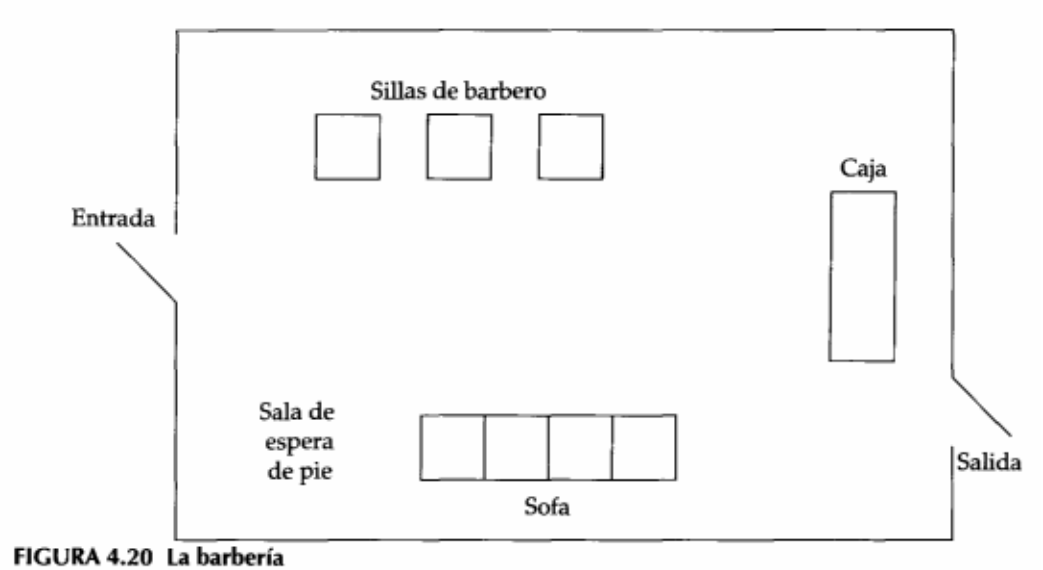


FIGURA 4.20 La barbería

Mantener los clientes en la si/la del barbero: Una vez sentado, un cliente permanece en la silla hasta que el barbero le indica que el corte está completo, por medio del semáforo *terminado.*

- *Limitarse a un cliente por cada si/la de barbero:* El semáforo *silla_barbero* limita el número de clientes en sillas de barbero a tres. Sin embargo, por si misma, la *silla_barbero* no lo conseguiría. Un cliente que no consigue obtener el procesador inmediatamente después de que su barbero ejecute *signal(terminado)* (es decir, entra en trance o se detiene a charlar con algún vecino) puede estar todavía en la silla cuando se le dé permiso para sentarse al siguiente. El semáforo *dejar silla b* corrige este problema, haciendo que el barbero no invite a un nuevo cliente a la silla hasta que el tardón haya anunciado su salida. En los ejercicios del final del capitulo, se vera que incluso esta precaución falla en detener a los clientes que estén deseosos por sentarse en las rodillas.
- *Pagar y tomar el recibo:* Naturalmente, conviene ser cuidadoso cuando hay dinero por medio. El cajero quiere asegurarse de que cada cliente paga antes de dejar la tienda y el cliente quiere una verificación de que se ha efectuado el pago **(un** recibo). Esto se lleva a cabo mediante una transferencia de dinero de persona a persona. Cada cliente, al levantarse de la silla de barbero, paga, después informa al cajero que se le ha dado el dinero [*signal*(pago)] y espera el recibo [*wait*(recibo)] cajero recibe pagos repetidamente: Espera que se le notifique un pago, acepta el dinero e indica su recepción. Aquí hay que evitar varios errores de programación. Si se produce un *signal(pago)* inmediatamente antes de la acción *pagar,* el cliente podría ser interrumpido después de tal señal; esto daría al cajero vía libre para aceptar pagos, incluso aunque no se le haya ofrecido ninguno. Un error más serio podría ser invertir las lineas *signal(pago)* y *wait(recibo).* Esto llevaría a un interbloqueo porque provocaría que todos los clientes y el cajero se bloquearan en sus operaciones *wait* respectivas.

```
program barbería1;
var   max_capacidad: semáforo (:= 20);
      sofá: semáforo (:= 4);
      silla_barbero, coord: semáforo (:= 3);
      cliente_listo, terminado, dejar_silla_b, pago, recibo: semáforo (:= 0);
procedure cliente;
begin
   wait(max_capacidad);
   entrar en tienda;
   wait(sofá);
   sentarse en sofá;
   wait(silla_barbero);
   levantarse del sofá;
   signal(sofá);
   sentarse en silla de barbero;
   signal(cliente_listo);
   wait(terminado);
   levantarse de silla de barbero;
   signal(dejar_silla_b);
   pagar;
   signal(pago);
   wait(recibo);
   salir de la tienda;
   signal(max_capacidad);
end;
procedure barbero;
begin
   repeat
      wait(cliente_listo);
      wait(coord);
      cotar pelo;
      signal(coord);
      signal(terminado);
      wait(dejar_silla_b);
      signal(silla_barbero);
   forever
end;
procedure cajero;
begin
   repeat
      wait(pago);
      wait(coord);
      aceptar pago;
      signal(coord);
      signal(recibo);
   forever
end;
begin (*programa principal*)
   parbegin
      cliente;...50 veces;...cliente;
      barbero; barbero; barbero;
      cajero
   parend
end.
```

FIGURA 4.21 Una barbería no equitativa

- *Coordinación de las funciones del cajero* y *del barbero:* Para ahorrarse un salario, esta barbería no utiliza un cajero independiente. Cada barbero debe realizar esta tarea cuando no esté cortando pelo. El semáforo *coord* garantiza que los barberos realizan solo una tarea a la vez.

La tabla 4.3 resume el uso de cada semáforo del programa.

El proceso cajero podría eliminarse incluyendo la función de pago en el procedimiento barbero. Cada barbero cortaría el pelo y después aceptarla pagos. Sin embargo, con una única caja registradora, hay que limitar el acceso a la función *aceptar pagos* a un solo barbero cada vez. Esto podría hacerse tratando esta función como una sección critica y protegiéndola con un semáforo.

TABLA 4.3 Función de los Semáforos de la figura 4.21

Semáforo	Operación *wait*	Operación *signal*
max_capacidad	El cliente espera hasta que haya sitio para entrar en a tienda.	El cliente que sale avisa a un cliente que está esperando para entrar.
sofá	El cliente espera asiento en el sofá.	El cliente que abandona el sofá avisa a un cliente que espera asiento.
silla_barbero	El cliente espera hasta que haya una silla de barbero vacía.	El barbero avisa cuando queda libre su silla.
cliente_listo	El barbero espera hasta que el cliente está en la silla.	El cliente avisa al barbero que ya está en la silla.
terminado	El cliente espera a que el corte de pelo esté completo.	El barbero avisa cuando termina el corte de pelo a un cliente.
dejar_silla_b	El barbero espera hasta que el cliente se levante de la silla.	El cliente avisa al barbero cuando se levanta de la silla.
pago	El cajero espera a que un cliente pague.	El cliente avisa al cajero que ya ha pagado.
recibo	El cliente espera para el recibo de haber pagado.	El cajero avisa al cliente que se ha aceptado el pago.
coord	Esperar a que un recurso de tipo barbero esté libre para cortar el pelo o para cobrar.	Indica que un recurso de tipo barbero está libre.

Una barbería equitativa

La figura 4.21 es una buena tentativa, pero aún quedan algunos problemas por resolver. Uno de ellos está resuelto en lo que resta de sección; los otros quedan como ejercicios para el lector (ver problema 4.20).

Hay un problema de temporización en la figura 4.21 que puede originar un trato no equitativo a los clientes. Supóngase que hay tres clientes sentados en las tres sillas de barbero. En tal caso, probablemente estarán bloqueados en *wait(terminado)* y, debido a la organización de la cola, serán liberados en el orden en que se sentaron en las sillas de barbero. Sin embargo, ¿qué ocurre **Si** uno de los barberos es muy rápido o uno de los clientes es bastante calvo? Si se libera a un cliente para que se siente en la silla, podría llegarse a una situación en la que un cliente es expulsado definitivamente de su asiento y obligado a pagar la tarifa completa por un corte de pelo parcial, mientras que a otro se le impide abandonar su asiento aunque su corte de pelo está completo.

El problema se resuelve con más semáforos, como se muestra en la figura 4.22. Se asigna un número único de cliente a cada uno; esto equivale a hacer que cada cliente tome un número al entrar a la tienda. El semáforo *exmut1* protege el acceso a la variable global *contador,* de forma que cada cliente reciba un número único. Se vuelve a definir el semáforo *terminado* para que sea un vector de 50 semáforos. Una vez que el cliente está sentado en una silla de barbero, ejecuta *wait*(*terminado*[*numcliente*]) para esperar en su propio semáforo; cuando el barbero termina con este cliente, ejecuta *signal* (*terminado*[*cliente_b*])para liberar el cliente apropiado.

Queda por especificar cómo los barberos conocen los números de clientes. Un cliente pone su número en la *cola_1* justo antes de avisar al barbero con el semáforo *cliente_listo.* Cuando un barbero está listo para cortar el pelo, *sacar_cola_1 (cliente listo)* retira el número más alto de cliente de la *cola1* y lo pone en la variable local del barbero *cliente_b.*

```
program barbería2;
var     max_capacidad: semáforo (:= 20);
        sofá: semáforo (:= 4);
        silla_barbero, coord: semáforo (:= 3);
        exmut1, exmut2: semáforo (:= 1);
        cliente_listo, dejar_silla_b, pago, recibo: semáforo (:= 0);
        terminado: array 1..50 of semáforo (:= 0);
        contador := entero;
procedure cliente;
var numcliente: entero
begin
    wait(max_capacidad);
    entrar en la tienda;
    wait(exmut1);
    contador := contador + 1;
    numcliente := contador;
    signal(exmut1);
    wait(sofá);
    sentarse en el sofá;
    wait(silla_barbero);
    levantarse del sofá;
    signal(sofá);
    sentarse en la silla del barbero;
    wait(exmut2);
    poner_cola_1([numcliente]);
    signal(cliente_listo);
    signal(exmut2);
    wait(terminadonumcliente);
    levantarse de la silla del barbero;
    signal(dejar_silla_b);
    pagar;
    signal(pago);
    wait(recibo);
    salir de la tienda;
    signal(max_capacidad);
end;
begin (*programa principal*)
    contador := 0;
    parbegin
       cliente;...50 veces;...cliente;
       barbero; barbero; barbero;
        cajero
    parend
end.

procedure barbero;
var cliente_b: entero;
begin
    repeat
        wait(cliente_listo);
        wait(exmut2);
        sacar_cola_1(cliente_b);
        signal(exmut2);
        wait(coord);
        cotar el pelo;
        signal(coord);
        signal(termina-
        do[cliente_b]);
        wait(dejar_silla_b);
        signal(silla_barbero);
    forever
end;

procedure cajero;
begin
    repeat
        wait(pago);
        wait(coord);
        aceptar pago;
        signal(coord);
        signal(recibo);
    forever
end;
```

FIGURA 4.22 Una barbería equitativa

4.5 MONITORES

Los semáforos son una herramienta básica, pero potente y flexible, para hacer cumplir la exclusión mutua y coordinar procesos. Sin embargo, como se propone en la figura 4.14, puede resultar difícil construir un programa correcto por medio de semáforos. La dificultad está en que las operaciones *wait y signal* deben distribuirse por todo el programa y no es fácil advertir el efecto global de estas operaciones sobre los semáforos a los que afectan.

Los monitores son estructuras de un lenguaje de programación que ofrecen una funcionalidad equivalente a La de los semáforos y que son más fáciles de controlar. El concepto fue definido formalmente por primera vez en [HOAR74]. La estructura de monitor se ha implementado en varios lenguajes de programación, incluido Pascal Concurrente, Pascal-plus, Modula-2 y Modula-3. Más recientemente, se han implementado como una biblioteca de programas. Esto permite poner cierres de monitor a objetos cualesquiera. En particular, para algo parecido a una lista enlazada, se puede querer bloquear todas las listas enlazadas con un solo cierre o bien tener un cierre para cada elemento de cada lista.

Se comenzará estudiando la versión de Hoare para después examinar una leve modificación.

Monitores con señales

Un monitor es un módulo de software que consta de uno o más procedimientos, una secuencia de inicialización y unos datos locales. Las características básicas de un monitor son las siguientes:

1. Las variables de datos locales están solo accesibles para los procedimientos del monitor y no para procedimientos externos.
2. Un proceso entra en el monitor invocando a uno de sus procedimientos.
3. Solo un proceso puede estar ejecutando en el monitor en un instante dado; cualquier otro proceso que haya invocado al monitor quedara suspendido mientras espera a que el monitor esté disponible.

Las dos primeras características recuerdan a las de los objetos del software orientado a objetos. En realidad, un sistema operativo o Lenguaje de programación orientado a objetos puede implementar un monitor fácilmente como un objeto con características especiales.

Si se cumple la norma de un proceso cada vez, el monitor puede ofrecer un servicio de exclusión mutua. Las variables de datos del monitor pueden ser accedidas solo por un proceso cada vez. Así pues, una estructura de datos compartida puede protegerse situándola dentro de un monitor. Si los datos del monitor representan a algún recurso, el monitor ofrecerá un servicio en exclusión mutua en el acceso a este recurso.

Para que resulten útiles en el proceso concurrente, los monitores deben incluir herramientas de sincronización. Por ejemplo, supóngase que un proceso llama a un monitor y, mientras está en el monitor, debe suspenderse hasta que se cumpla alguna condición. Hace falta un servicio para que el proceso no solo esté suspendido, sino que libere el monitor y otro proceso pueda entrar. Más tarde, cuando se cumpla la condición y el monitor está de nuevo disponible, el proceso puede reanudarse y tiene que permitírsele volver a entrar en el monitor en el punto de la suspensión.

Un monitor proporciona sincronización por medio de las variables de condición que se incluyen dentro del monitor y que son accesibles sólo desde dentro. Hay dos funciones para operar con las variables de condición:

wait(c): Suspende la ejecución del proceso llamado bajo la condición c. El monitor está ahora disponible para ser usado por otro proceso.

signal(c): Reanuda la ejecución de algún proceso suspendido después de un *wait* bajo la misma condición. Si hay varios procesos, elige uno de ellos; si no hay ninguno, no hace nada.

Nótese que las operaciones de monitor *wait/signal* son diferentes de las de los semáforos. Si un proceso de un monitor ejecuta un *signal* y no hay tareas esperando en la variable de condición, el *signal* se pierde.

La figura 4.23 ilustra la estructura de un monitor. Aunque un proceso puede entrar al monitor llamando a cualquiera de sus procedimientos, puede considerarse que el monitor tiene un único punto de entrada que está custodiado para que sólo un proceso pueda estar en el monitor en cada instante. Otros procesos que intenten entrar al monitor se añadirán a una cola de procesos suspendidos mientras esperan a que el monitor esté disponible. Una vez que un proceso está dentro del monitor, puede suspenderse a sí mismo temporalmente bajo la condición *x* ejecutando *wait(x);* entonces se sitúa en una cola de procesos que esperan volver a entrar al monitor cuando la condición cambie.

Si un proceso que está ejecutando en el monitor detecta un cambio en una variable de condición *x,* ejecuta *signal(x),* lo que avisa a la cola de condición correspondiente de que la condición ha cambiado. /-

Como ejemplo del uso de monitores, se va a retomar el problema del productor/consumidor con buffer acotado. La figura 4.24 muestra un solución con monitores. El módulo monitor, *buffer_acotado,* controla el buffer empleado para almacenar y retirar caracteres. El monitor incluye dos variables de condición: *no_lleno* es cierta cuando hay sitio para añadir al menos un carácter al buffer y *no_vacío* es cierta cuando hay al menos un carácter en el buffer.

Un productor sólo puede añadir caracteres al buffer por medio del procedimiento *añadir* del monitor; el productor no tiene acceso directo a *buffer.* El procedimiento comprueba primero la condición *no_lleno,* para determinar si hay espacio libre en el buffer. Si no lo hay, el proceso que está ejecutando el monitor se suspende en esa condición. Cualquier otro proceso (productor o consumidor) puede entrar ahora al monitor. Posteriormente, cuando el buffer ya no esté lleno, el proceso suspendido podrá ser retirado de la cola, reactivado y el procesamiento podrá reanudarse. Después de poner un carácter en el buffer, el proceso activa la condición *no_vacío.* Se puede hacer una descripción similar de la tarea del consumidor.

Este ejemplo señala la división de responsabilidades en los monitores comparados con los semáforos. En el caso de los monitores, la misma estructura del monitor garantiza la exclusión mutua: No es posible que el productor y el consumidor accedan simultáneamente al buffer. Sin embargo, el programador debe situar correctamente las primitivas *wait* y *sig-nal* en el monitor para impedir que los procesos depositen elementos en un buffer lleno o los extraigan de uno vacío. En el caso de los semáforos, tanto la exclusión mutua como la sincronización son responsabilidades del programador.

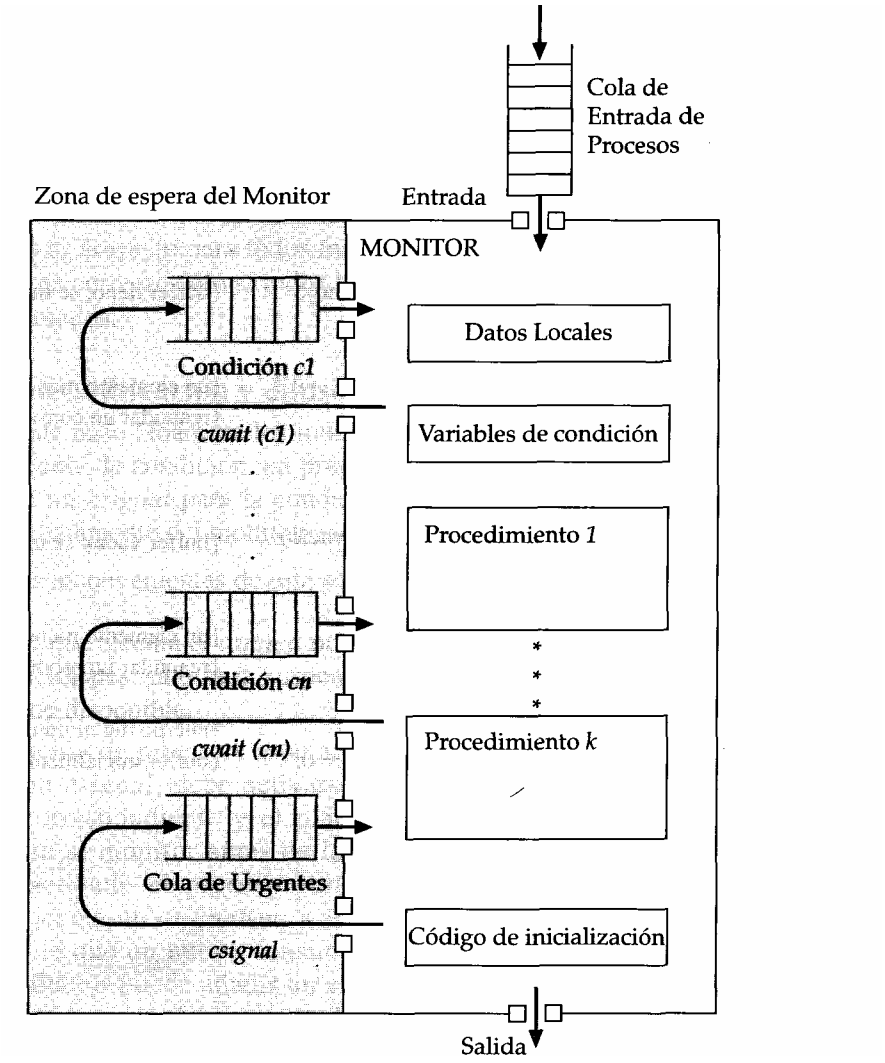


FIGURA 4.23 Estructura de un monitor

Nótese que, en la figura 4.24, un proceso sale del monitor inmediatamente después de ejecutar la función *signal.* Si *signal* no tiene lugar al final del procedimiento, entonces, según la propuesta de Hoare, el proceso que da la señal queda suspendido para dejar disponible el monitor y es puesto en una cola hasta que el monitor está libre. Una posibilidad al respecto sería situar el proceso suspendido en la cola de entrada para que compita por el acceso con los otros procesos que todavía no han entrado al monitor. Sin embargo, puesto que un proceso suspendido en una función *signal* ya ha realizado parcialmente su tarea en el monitor, tiene más sentido dar preferencia a este proceso sobre los procesos de nueva entrada, situándolo en una cola de urgentes por separado (figura 4.23). Un lenguaje que utiliza monitores, el Pascal Concurrente, obliga a que *signal* sólo pueda aparecer como la última operación ejecutada por un procedimiento del monitor.

Si no hay procesos esperando en la condición *x,* la ejecución de *signal(x)* no tiene efecto.

```
program productor_consumidor
    monitor buffer_acotado;
        buffer: array [0..N] of caracteres;          {espacio para N elementos}
        sigent, sigsal: entero;                      {apuntadores al buffer}
        contador: entero;                            {número de elementos en el buffer}
        no_lleno, no_vacío: condition;               {para sincronización}
var i: entero;
    procedure añadir (x: carácter);
        begin
            if contador=N then cwait (no_lleno);     {buffer lleno; se impide producir}
            buffer[sigent] := x;
            sigent := sigent + 1 mod N;
            contador := contador + 1;                {un elemento más en el buffer}
            csignal (no_vacío);                      {reanudar un consumidor en espera}
        end;
    procedure tomar (x: carácter);
        begin
            if contador=0 then cwait (no_vacío)      {buffer vacio; se impide consumir}
            x := buffer[sigsal];
            sigsal := sigsal +1 mod N;
            contador := contador –1;                 {un elemento menos en el buffer}
            csignal (no_lleno);                      {reanudar un productor en espera}
        end;
    begin                                            {cuerpo del monitor}
            sigent := 0; sigsal := 0; contador := 0; {buffer inicialmente vacío}
    end;
procedure productor;
var x: carácter;
begin
    repeat
        producir(x);
        añadir(x);
    forever
end;
procedure consumidor;
var x: carácter;
begin
    repeat
        tomar(x);
        consumir(x);
    forever
end;
begin (*programa principal*)
    parbegin
        productor; consumidor
    parend
end.
```

FIGURA 4.24 Una solución al problema del productor/consumidor con buffer acotado por medio de monitores

Como con los semáforos, es posible cometer errores en la sincronización de los monitores. Por ejemplo, si se omite cualquiera de las funciones *csignal* en el monitor *buffer_acotado,* los procesos que entran en la cola de la condición correspondiente permanecerán colgados permanentemente. La ventaja que tienen los monitores sobre los semáforos es que todas las funciones de sincronización están confinadas dentro del monitor. De este modo, es sencillo verificar que la sincronización se ha realizado correctamente y detectar los fallos. Es más, una vez que un monitor está correctamente programado, el acceso al recurso protegido es correcto para todos los procesos. Con los semáforos, en cambio, el acceso al recurso es correcto sólo si todos los procesos que acceden al recurso están correctamente programados.

Monitores con notificación y difusión

La definición que hace Hoare de los monitores [HOAR74] exige que, si hay al menos un proceso en una cola de condición, un proceso de dicha cola deberá ejecutar en cuanto otro proceso ejecute un *csignal* para la condición. Así pues, el proceso que ejecuta el *csignal* debe salir inmediatamente del monitor o suspenderse en el monitor.

Son varios los inconvenientes de esta solución:

1. Si el proceso que ejecuta el *csignal* no abandona el monitor, hacen falta dos cambios de contexto adicionales: uno para suspender el proceso y otro para reanudarlo cuando el monitor quede disponible.

2. La planificación de procesos asociada con las señales debe ser muy fiable. Cuando se ejecuta un *csignal,* debe activarse inmediatamente un proceso de la cola de la condición correspondiente y el planificador debe asegurarse de que ningún otro proceso entra al monitor antes de la activación. Si no es así, la condición bajo la que se ha activado el proceso podría cambiar. Por ejemplo, en la figura 4.24, cuando se ejecuta un *csignal(no_ vacío),* un proceso de la cola *no _ vació* debe activarse antes de que un nuevo consumidor entre al monitor. Otro ejemplo es que un proceso productor puede añadir un carácter a un buffer vacío y después fallar antes de dar la señal; cualquier proceso de la cola *no_vacío* permanecerá colgado para siempre.

Lampson y Redell desarrollaron una definición diferente de monitores para el lenguaje Mesa [LAMP80]. Su método supera los problemas comentados y ofrece algunas ampliaciones de utilidad. La estructura de monitores de Mesa se usa también en el lenguaje Modula-3 de programación de sistemas [CARD89, HARB90, NELS91]. En Mesa, la primitiva *csignal* se reemplaza por *cnotify,* con la siguiente interpretación:

Cuando un proceso que está en un monitor ejecuta *cnotify(x),* origina una notificación hacia la cola de la condición *x,* pero el proceso que da la señal puede continuar ejecutando. El resultado de la notificación es que el proceso de la cabeza de la cola de condición será reanudado en el futuro cercano, cuando el monitor esté disponible. Sin embargo, puesto que esto no garantiza que ningún otro proceso entre al monitor antes que el proceso que espera, el proceso que espera debe volver a comprobar la condición. Por ejemplo, los procedimientos en el monitor *buffer _ acotado* tendrían ahora el siguiente código:

```
procedure añadir (x: carácter);
begin
while contador=N do cwait (no _ lleno);       {buffer lleno; se evita el desbordamiento}
buffer[sigent] := x;
sigent := sigent + 1 mod N;
contador := contador +1;                      {un elemento más en el buffer}
    cnotify (no _ vació);                         {notifica a un consumidor esperando}
 end;
procedure tomar (x: carácter);
begin
while contador=0 do cwait (no _ vació);       {buffer vacío; se evita consumir}
x := buffer[sigsal];
sigsal := sigsal + 1 mod N;
contador := contador 1;                       {un elemento menos en el buffer}
cnotify (no _ lleno);                          { notifica a un productor esperando}
      end;
```

La sentencia **if** se reemplaza ahora por un bucle **while.** Así pues, esta modificación genera, al menos, una evaluación extra de la variable de condición. Como contrapartida, sin embargo, no hay cambios de contexto extra y no hay limitaciones sobre cuándo debe ejecutar el proceso que espera después del *cnotify.*

Una modificación útil que se puede asociar a la primitiva *cnotify* es establecer un temporizador de guarda asociado con cada primitiva sobre la condición. Un proceso que ha estado esperando durante el intervalo máximo de guarda será situado en el estado de listo independientemente de si se ha notificado a la condición. Cuando se active, el proceso comprueba la condición y continúa ejecutando si es que se cumple. El temporizador impide la inanición indefinida de un proceso en el caso de que algún otro proceso falle antes de señalar una condición.

Con la norma de notificar a los procesos en lugar de reactivarlos a la fuerza, es posible añadir una primitiva de difusión *cbroadcast* al repertorio. La difusión provoca que todos los procesos que están esperando por una condición se sitúen en el estado de listo. Esto es conveniente en las situaciones donde un proceso no sabe cuántos procesos deben reactivarse. Por ejemplo, en el programa del productor/consumidor, supóngase que tanto la función de añadir como la de tomar pueden aplicarse en bloques de caracteres de longitud variable. En este caso, si un productor añade un bloque de caracteres al buffer, no necesita saber cuantos caracteres está preparado para consumir cada consumidor en espera. El simplemente ejecuta un *cbroadcast y* todos los procesos en espera son avisados para que lo intenten de nuevo.

Además, la difusión puede emplearse en los casos en los que un proceso tenga dificultades para averiguar exactamente a qué otro proceso tiene que reactivar. Un buen ejemplo es el de un gestor de memoria. El gestor *tiene j* bytes libres; un proceso libera *k* bytes adicionales; pero no sabe que proceso en espera puede continuar con un total *de j +k* bytes; por tanto, utiliza *cbroadcast* y todos los procesos comprueban por sí mismos si hay suficiente memoria libre.

Una ventaja de los monitores de Lampson/Redell sobre los de Hoare es que el método de Lampson/Redell es menos propenso a errores. En el enfoque de Lampson/Redell, debido a que cada procedimiento comprueba la variable del monitor después de ser despertado, por medio de la instrucción *while,* un proceso puede realizar una señal o una difusión incorrectamente sin provocar un error en el programa que la recibe. El programa que recibe la señal comprobará la variable pertinente y, si no se cumple la condición deseada, continuará esperando.

Otra ventaja de los monitores de Lampson/Redell es que se prestan a un método más modular de construcción de programas. Por ejemplo, considérese la implementación de una asignación de buffers. Hay dos niveles de condición que deben satisfacerse para los procesos secuénciales cooperantes:

Nivel 1. Estructuras de datos consistentes. Esto significa que el monitor hace cumplir la exclusión mutua y concluye la operación de entrada o salida antes de permitir cualquier otra operación sobre el buffer.

Nivel 2. La misma condición del nivel 1 y, además, disponer de suficiente memoria para que este proceso pueda completar su solicitud de asignación.

En los monitores de Hoare, cada señal da a entender la condición del nivel 1, pero también conlleva el mensaje implícito "He liberado suficientes octetos para que funcione tu solicitud de asignación". Así pues, la señal lleva implícita la condición del nivel 2. Si el programador cambia más tarde la definición de la condición del nivel 2, será necesario reprogramar todos los procesos señalizadores. Si el programador cambia los supuestos realizados por un proceso determinado que esté esperando (es decir, esperando una invariante del nivel 2 ligeramente diferente), puede ser necesario reprogramar todos los procesos señalizadores. Esto no es modular y es probable que origine errores de sincronización (por ejemplo, despertarse por error) cuando se modifique el código. El programador tiene que acordarse de modificar todos los procesos cada vez que se haga un leve cambio en la condición del nivel 2. Ahora, con un monitor de Lampson/Redell, una difusión asegura la condición del nivel 1 y da una indicación de que puede que se cumpla la del nivel 2; cada proceso debe comprobar por sí mismo la condición del nivel 2. Si se hace algún cambio en la condición del nivel 2, tanto en uno que espera como en uno que señaliza, no hay posibilidad de despertar erróneo porque cada procedimiento chequea su propia condición del nivel 2. Por tanto, la condición del nivel 2 puede ocultarse en cada procedimiento. Con los monitores de Hoare, la condición del nivel 2 debe ser introducida por el que espera en el código de cada proceso señalizador, lo cual viola los principios de abstracción de datos y de modularidad entre procedimientos.

4.6 PASO DE MENSAJES

Cuando los procesos interactúan unos con otros, se deben satisfacer dos requisitos básicos:

la sincronización y la comunicación. Los procesos tienen que sincronizarse para cumplir la exclusión mutua; los procesos cooperantes pueden necesitar intercambiar información. Un método posible para ofrecer ambas funciones es el paso de mensajes. El paso de mensajes tiene la ventaja adicional de que se presta a ser implementado en sistemas distribuidos, así como en sistemas multiprocesador y monoprocesador de memoria compartida.

Hay varias formas de sistemas de paso de mensajes. En esta sección, se ofrecerá una introducción general que discute las características típicas encontradas en estos sistemas. La funcionalidad real del paso de mensajes se ofrece, normalmente, por medio de un par de primitivas:

• *sena (destino, mensaje)*

• *receive (origen, mensaje)*

Este es el conjunto mínimo de operaciones necesario para que los procesos puedan dedicarse al paso de mensajes. Un proceso envía información en forma de un *mensaje* a otro proceso designado como *destino.* Un proceso recibe información ejecutando la primitiva *receive,* que indica el proceso emisor *(origen)* y el *mensaje.*

En la tabla 4.4 se enumeran las cuestiones de diseño relativas a los sistemas de paso de mensajes; en el resto de esta sección se examinan, por orden, cada una de estas cuestiones.

Sincronización

La comunicación de un mensaje entre dos procesos implica cierto nivel de sincronización entre ambos. El receptor no puede recibir un mensaje hasta que sea enviado por otro proceso. Además, hace falta especificar qué le sucede a un proceso después de ejecutar una primitiva *sena o receive.*

Considérese en primer lugar la primitiva *send.* Cuando se ejecuta una primitiva *sena* en un proceso, hay dos posibilidades: O bien el proceso emisor se bloquea hasta que se recibe el mensaje o no se bloquea. Análogamente, cuando un proceso ejecuta una primitiva *receive,* hay dos posibilidades:

1. Si previamente se ha enviado algún mensaje, éste es recibido y continúa la ejecución.
2. Si no hay ningún mensaje esperando entonces, o bien (a) el proceso se bloquea hasta que llega un mensaje o (b) el proceso continúa ejecutando, abandonando el intento de recepción.

Así pues, tanto el emisor como el receptor pueden ser bloqueantes o no bloqueantes. Son habituales las siguientes tres combinaciones, aunque cualquier sistema concreto implementa sólo una o dos combinaciones:

• *Envío bloqueante, recepción bloqueante:* Tanto el emisor como el receptor se bloquean hasta que se entrega el mensaje; esta técnica se conoce como *rendezvous.* Esta combinación permite una fuerte sincronización entre procesos.

• *Envío no bloqueante, recepción bloqueante:* Aunque el emisor puede continuar, el receptor se bloquea hasta que llega el mensaje solicitado. Esta es, probablemente, la combinación más útil. Permite que un proceso envíe uno o más mensajes a varios destinos tan rápido como sea posible. Un proceso que debe recibir un mensaje antes de poder hacer alguna función útil tiene que bloquearse hasta que llegue el mensaje. Un ejemplo es el de un proceso servidor que ofrezca un servicio o un recurso a otros procesos.

• *Envío no bloqueante, recepción no bloqueante:* Nadie debe esperar.

El *send* no bloqueante es la forma más natural para muchas tareas de programación concurrente. Por ejemplo, si se usa para solicitar una operación de salida, tal como una impresión, permite al proceso solicitante realizar la solicitud en forma de mensaje y continuar. Un posible riesgo del *send* no bloqueante es que un error puede llevar a una situación en la que el proceso

TABLA 4.4 Características de diseño de sistemas de mensajes para la comunicación y sincronización entre procesos

Sincronización	Formato
Send	Contenido
bloqueante	Longitud
no bloqueante	fija
Receive	variable
bloqueante	
no bloqueante	**Disciplina de Cola**
comprobación de llegada	
	FIFO
Direccionamiento	Prioridades
Directo	
envío	
recepción	
explicita	
implícita	
Indirecto	
estático	
dinámico	
propiedad	

genere mensajes repetidamente. Como no hay bloqueo para hacer entrar en disciplina al proceso, esos mensajes pueden consumir recursos del sistema, incluido tiempo del procesador y espacio en buffer, en detrimento de otros procesos y del sistema operativo. Además, el *send* no bloqueante carga sobre el programador el peso de determinar qué mensaje se ha recibido: Los procesos deben emplear mensajes de respuesta para acusar La recepción de un mensaje.

Para la primitiva *receive,* la versión bloqueante parece ser la más natural para muchas tareas de programación concurrente. En general, un proceso que solicita un mensaje necesitará la información esperada antes de continuar. Sin embargo, si se pierde un mensaje, un proceso receptor puede quedarse bloqueado indefinidamente. Este problema puede resolverse por medio del *receive* no bloqueante. Sin embargo, el riesgo de esta solución es que si un mensaje se envía después de que un proceso haya ejecutado el correspondiente *receive,* el mensaje se perderá. Otro método posible consiste en permitir a un proceso comprobar si hay un mensaje esperando antes de ejecutar un *receive* y en permitir a un proceso especificar más de un origen en una primitiva *receive.* Esta última técnica resulta útil si un proceso espera mensajes de mas de un origen y puede continuar si llega cualquiera de estos mensajes.

Direccionamiento

Evidentemente, es necesario disponer de alguna forma de especificar en la primitiva *send* qué proceso va a recibir el mensaje. De forma similar, la mayoría de las implementaciones permiten a los procesos receptores indicar el origen del mensaje que se va a recibir.

Los distintos esquemas para hacer referencia a los procesos en las primitivas *sena* y *receive* se encuadran dentro de dos categorías: direccionamiento directo e indirecto. Con el **direccionamiento directo,** la primitiva *sena* incluye una identificación específica del proceso destino. La primitiva *receive* se puede gestionar de dos formas. Una posibilidad requiere que el proceso designe explícitamente un proceso emisor. Así pues, el proceso debe conocer de antemano de qué proceso espera un mensaje. Esto suele ser eficaz para procesos concurrentes y cooperantes. En otros casos, sin embargo, es imposible especificar el proceso de origen por anticipado. Un ejemplo es un proceso servidor de impresoras, que aceptará mensajes de solicitud de impresión de cualquier otro proceso. Para tales aplicaciones, una técnica más efectiva consiste en usar direccionamiento implícito. En este caso, el parámetro *origen* de la primitiva *receive* tendrá un valor de retomo cuando se haya realizado la operación de recepción.

El otro enfoque es el **direccionamiento indirecto.** En este caso, los mensajes no se envían directamente del emisor al receptor, sino a una estructura de datos compartida formada por colas que pueden guardar los mensajes temporalmente. Estas colas se denominan generalmente *buzones (mailboxes).* De este modo, para que dos procesos se comuniquen, uno envía mensajes al buzón apropiado y el otro los coge del buzón.

Una ventaja del direccionamiento indirecto es que se desacopla a emisor y receptor, permitiendo una mayor flexibilidad en el uso de los mensajes. La relación entre emisores y receptores puede ser uno a uno, de muchos a uno, de uno a muchos o de muchos a muchos. Una relación uno a uno permite que se establezca un enlace privado de comunicaciones entre dos procesos, lo que aísla su interacción de injerencias erróneas de otros procesos. Una relación de muchos a uno resulta útil para interacciones cliente/servidor, donde un proceso ofrece un servicio a un conjunto de procesos. En este caso, el buzón se denomina *puerto* (figura 4.25). Una relación uno a muchos permite un emisor y muchos receptores; es útil para aplicaciones en las que un mensaje o alguna información se difunda a un conjunto de procesos.

La asociación de procesos a buzones puede ser estática o dinámica. Los puertos suelen estar asociados estáticamente con algún proceso en particular, es decir, el puerto se crea y se asigna al proceso permanentemente. De forma similar, una relación uno a uno normalmente se define de forma estática y permanente. Cuando hay varios emisores, la asociación de un emisor a un buzón puede realizarse dinámicamente. Se pueden usar primitivas como *conectar* y *desconectar* con este propósito.

Una cuestión afín es la de la propiedad del buzón. En el caso de un puerto, normalmente pertenece y es creado por el proceso receptor. De este modo, cuando se destruye el proceso, también se destruirá el puerto. Para el caso general de los buzones, el sistema operativo puede ofrecer un servicio de creación de buzones. Estos buzones pueden ser considerados como propiedad del proceso creador, en cuyo caso se destruyen junto con el proceso o pueden ser considerados como propiedad del sistema operativo, en cuyo caso se necesita una orden explícita para destruir el buzón.

Formato de mensajes

El formato de los mensajes depende de los objetivos del servicio de mensajería y de si el servicio ejecuta en un ordenador independiente o en un sistema distribuido. Para algunos sistemas operativos, los diseñadores han elegido mensajes cortos y de tamaño

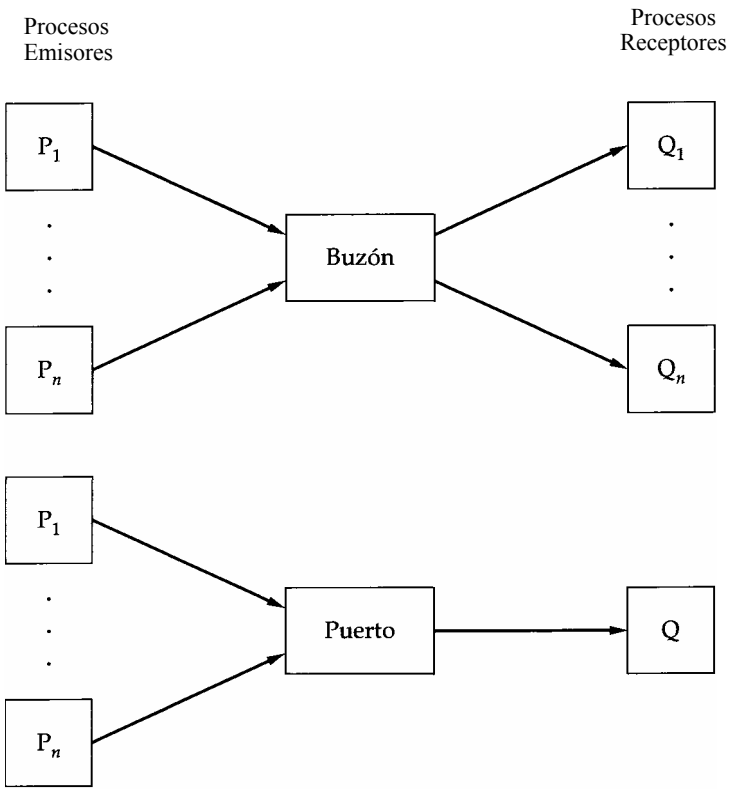


FIGURA 4.25 Comunicación indirecta entre procesos [BIC88]

fijo para minimizar el procesamiento y el coste de almacenamiento. Si se va a pasar una gran cantidad de datos, los datos pueden ponerse en un archivo y el mensaje simplemente hará referencia a este archivo. Una solución más flexible es permitir mensajes de longitud variable.

La figura 4.26 muestra un formato típico de mensajes para sistemas operativos que permiten mensajes de longitud variable. El mensaje se divide en dos partes: una cabecera, que alberga información sobre el mensaje y un cuerpo, que alberga el contenido real del mensaje. La cabecera puede contener una identificación del origen y el destino deseados, un campo de longitud y un campo de tipo para distinguir entre varios tipos de mensajes. Puede haber también información de control adicional, como un campo apuntador para poder crear una lista enlazada de mensajes; un número de secuencia, para guardar constancia del orden y número de mensajes pasados entre origen y destino; y un campo de prioridad.

Disciplina de cola

La disciplina de cola más simple es la de primero en llegar/primero en salir, pero ésta puede no ser suficiente si algunos mensajes son más urgentes que otros. Una alternativa es permitir la especificación de prioridades de los mensajes, en función del tipo de mensaje o por designación del emisor. Otra alternativa es permitir al receptor examinar la cola de mensajes y seleccionar el mensaje a recibir a continuación.

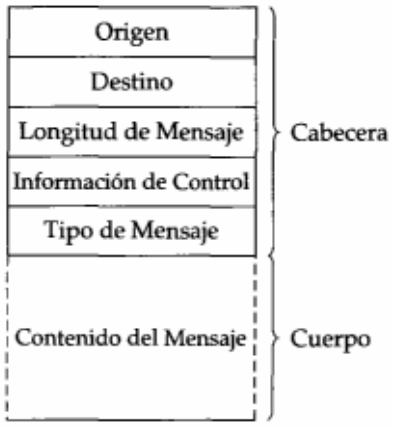


FIGURA 4.26 Formato típico de mensaje

Exclusión mutua

La figura 4.27 muestra una forma en que puede usarse el paso de mensajes para cumplir la exclusión mutua (compárense las figuras 4.2, 4.8 y 4.11). Supóngase que se usan primitivas *receive* bloqueantes y *send* no bloqueantes. Un conjunto de procesos concurrentes comparten un buzón, *exmut,* que puede ser usado por todos los procesos para enviar y recibir. El buzón contiene inicialmente un único mensaje, de contenido nulo. Un proceso que desea entrar en su sección critica intenta primero recibir el mensaje. Si el buzón está vacío, el proceso se bloquea. Una vez que un proceso ha conseguido el mensaje, ejecuta su sección crítica y, después, devuelve el mensaje al buzón. De este modo, el mensaje funciona como un testigo *(token)* que se pasa de un proceso a otro. Esta técnica supone que si hay más de un proceso ejecutando la acción *receive* concurrentemente, entonces:

- Si hay un mensaje, se entrega solo a uno de los procesos y los otros se bloquean.
- Si el buzón está vacío, todos los procesos se bloquean. Cuando haya un mensaje disponible, solo se activa y toma el mensaje uno de los procesos bloqueados.

Estas suposiciones son ciertas en prácticamente todos los servicios de paso de mensajes.

Como otro ejemplo del uso de paso de mensajes, en la figura 4.28 se ofrece una solución al problema del productor/consumidor con buffer acotado. Gracias a la característica básica de exclusión mutua del paso de mensajes, el problema puede resolverse con una estructura algorítmica similar a la de la figura 4.18. En cambio, el programa de la figura 4.28 tiene la ventaja de la capacidad del paso de mensajes para pasar datos además de señales. Se emplean dos buzones. A medida que el productor genera datos, los envía como mensajes a! buzón *puede consumir.* Con tal de que haya al menos un mensaje en dicho buzón, el consumidor podrá consumir. Por tanto, *puede _ consumir* hace de buffer; los datos en el buffer se organizan como una cola de mensajes. El "tamaño" del buffer viene determinado por la variable global *capacidad.* Inicialmente, el buzón *puede_producir* se rellena con un número de mensajes nulos igual a la capacidad del buffer. El número de mensajes de *puede _ producir* se reduce con cada producción y crece con cada consumo.

```
program exclusión_mutua;
const n = ...; (*número de procesos*);
procedure P(i: entero);
var msj: mensaje;
begin repeat
        receive (exmut, msj);
< sección crítica >;
send (exmut, msj);
< resto >
forever
end;
begin (*programa principal*)
crear_buzón (exmut);
    send (exmut, nuil);
    parbegin P(l);
        P(2);
        ...
        P(n)
    parend
end.
```

FIGURA 4.27 Exclusión mutua por medio de mensajes

Esta técnica es bastante flexible. Puede haber varios productores y consumidores siempre que todos tengan acceso a ambos buzones. El sistema puede ser incluso distribuido, con todos los procesos productores y el buzón *puede _producir* en un nodo y todos los consumidores y el buzón *puede _consumir* en otro.

4.7 PROBLEMA DE LOS LECTORES/ESCRITORES

En el diseño de los mecanismos de concurrencia y sincronización, resulta útil poder cotejar un problema dado con algún otro bien conocido y poder probar cualquier solución al problema en términos de su capacidad para resolver estos problemas conocidos. En la bibliografía, hay varios problemas que han adquirido importancia y que aparecen frecuentemente, debido a que son ejemplos de problemas de diseño habituales y a su valor educativo. Uno de estos problemas es el del productor/consumidor, que ya se ha estudiado. En esta sección, se considerará otro problema clásico: el problema de los lectores/escritores.

El problema de los lectores/escritores se define de la siguiente manera: Existe un área de datos compartida entre una serie de procesos. El área de datos puede ser un archivo, un bloque de memoria principal o incluso un banco de registros del procesador. Hay algunos procesos que sólo leen los datos (lectores) y otros que sólo escriben datos (escritores). Se deben satisfacer las siguientes condiciones:

1. Cualquier número de lectores puede leer el archivo simultáneamente.
2. Sólo puede escribir en el archivo un escritor en cada instante.
3. Si un escritor está accediendo al archivo, ningún lector puede leerlo.

Antes de continuar, se debe hacer una distinción entre este problema y otros dos: el problema general de exclusión mutua y el problema del productor/consumidor. En el problema de los lectores/escritores, ni los lectores escriben en el área de datos ni los escritores leen. Un caso más general, en el que se incluye éste, consiste en permitir a cualquiera de los procesos leer o escribir en los datos. En tal caso, se puede declarar cualquier parte del proceso que acceda a los datos como una sección crítica e imponer una solución general de exclusión

```
program buffer_acotado

const
capacidad = ...;                          {capacidad del buffer}
 null = ...;                              {mensaje vacío}
var i: entero;
procedure productor;
var msjp: mensaje;
begin
while cierto do begin
receive (puede_producir, msjp);
                  msjp := producir;
                 send (puede _ consumir, msjp)
            end
end;
procedure consumidor;
var msjp: mensaje;
begin
while cierto do begin
receive (puede_consumir, msjp);
consumir (msjp);
send (puede _ producir, null)
end
end;
(proceso padre}
begin
crear _ buzón (puede _ producir);
crear _ buzón (puede _ consumir);
for i := 1 to capacidad do send (puede _ producir, null);
parbegin
     productor;
     consumidor
parend
          end.
```

FIGURA 4.28 Una solución al problema del productor/consumidor por medio de mensajes [MILE92]

mutua. La razón de interesarse en el caso más restrictivo es que es posible crear soluciones más eficientes y que la solución menos eficiente para el problema general es inaceptablemente lenta. Por ejemplo, supóngase que el área compartida es el catálogo de una biblioteca. Los usuarios normales de la biblioteca consultan el catálogo para localizar un libro, mientras que los bibliotecarios deben poder actualizar el catálogo. En la solución general, todo acceso al catálogo deberá tratarse como una sección crítica y los usuarios estarían obligados a consultar el catálogo uno a uno. Esto redundaría, sin duda alguna, en retrasos intolerables. Al mismo tiempo, es importante impedir que los escritores interfieran unos con otros y es también necesario impedir las consultas mientras están llevándose a cabo modificaciones, para impedir el acceso a información incorrecta.

¿Puede considerarse al problema del productor/consumidor como simplemente un caso especial del problema de los lectores/escritores con un único escritor (el productor) y un único lector (el consumidor)? La respuesta es negativa. El productor no es sólo un escritor. Debe leer los punteros de la cola para determinar dónde escribir el siguiente elemento y debe determinar si el buffer está lleno. Análogamente, el consumidor no es sólo un lector porque debe ajustar los punteros de la cola para indicar que ha retirado una unidad del buífer.

A continuación se examinan dos soluciones al problema.

Prioridad a los lectores

La figura 4.29a es una solución que utiliza semáforos, mostrando un caso de un lector y otro de un escritor; la solución no cambia para el caso de varios lectores y escritores. El proceso escritor es sencillo. El semáforo *esem* se usa para respetar la exclusión mutua. Mientras que un escritor está accediendo a los datos compartidos, ningún otro escritor y ningún lector podrá acceder. El proceso lector también usa el semáforo *esem* para respetar la exclusión mutua. Sin embargo, para que se permitan varios lectores, hace falta que, cuando no haya ninguno, el primer lector que lo intente tenga que esperar en *esem.* Cuando ya hay al menos un lector, los lectores posteriores no necesitan esperar antes de entrar. La variable global *contlect* se utiliza para mantener el número de lectores y el semáforo *x* se utiliza para asegurar que *contlect* se actualiza correctamente.

Prioridad a los escritores

En la solución anterior, los lectores tienen prioridad. Una vez que un lector ha comenzado a acceder a los datos, es posible que los lectores mantengan el control del área de datos mientras que haya al menos uno leyendo. Por tanto, los escritores están sujetos a inanición.

La figura 4.29b muestra una solución que garantiza no permitir acceder a los datos a ningún nuevo lector una vez que, al menos, un escritor haya declarado su deseo de escribir. Para los escritores, se añaden los siguientes semáforos y variables al ya definido:

Un semáforo *Isem* que inhibe todas las lecturas mientras haya al menos un escritor que desee acceder a los datos.

- Una variable *contesc* que controla la activación de *Isem.*
- Un semáforo y que controla la actualización de *contesc.*

Para los lectores, se necesita un semáforo adicional. No debe permitirse que se construya una cola grande sobre *Isem,* pues los escritores no serían capaces de saltarla. Por tanto, sólo se permite a un lector ponerse en cola en *Isem* y todos los demás lectores deben ponerse en cola en un semáforo z inmediatamente antes de esperar en *Isem.* La tabla 4.5 resume las posibilidades.

```
program lectores_escritores;
var   contlect: entero;
x, esem: semáforo (:= 1);
procedure lector;
begin
      repeat
       wait(x);
            contlect := contlect + 1;
            if contlect = 1 then wait(esem);
        signal(x);
        LEER_UNIDAD;
        wait(x);
            contlect := contlect 1;
             if contlect = 0 then signal(esem);
         signal(x)
      forever
end;
procedure escritor;
begin
       repeat
wait(esem);
              ESCRIBIR_UNIDAD;
          signal(esem)
forever
end;
begin
contlect := 0;
   parbegin
       lector;
      escritor
   parend
end.
```

FIGURA 4.29a Una solución al problema de los lectores/escritores por medio de semáforos; los lectores tienen prioridad

Una solución alternativa, que da prioridad a los escritores y que se implementa mediante paso de mensajes, se muestra en la figura 4.30 y está basada en un algoritmo tomado de un trabajo de Theaker y Brookes [THEA83]. En este caso, hay un proceso controlador que tiene acceso al área de datos compartida. Los otros procesos que desean acceder a los datos envían un mensaje de solicitud al controlador, concediéndose el acceso mediante un mensaje de respuesta "OK" e indicándose la finalización del acceso con un mensaje "terminado".

El controlador dispone de tres buzones, uno para cada tipo de mensaje que debe recibir. El proceso controlador da servicio a los mensajes de solicitud de escritura antes que a las solicitudes de lectura, para dar prioridad a los escritores. Además, se respeta la exclusión mutua. Para hacerla cumplir, se emplea la variable *cont,* que se inicializa con algún número

```
program lectores_escritores;
var   contlect, contesc: entero;
            x, y, z, esem, Isem: semáforo (:= 1);
procedure lector;
      begin
          repeat
              wait(z);
                  wait(lsem);
                       wait(x);
                            contlect := contlect + 1;
                            if contlect = 1 then wait(esem);
                       signal(x);
                   signal(lsem);
               signal(z);
               LEER_UNIDAD;
               wait(x);
                    contlect := contlect 1;
                    if contlect = 0 then signal(esem);
               signal(x)
      forever
    end;
procedure escritor;
begin
       repeat
           wait(y);
              contesc := contesc + 1;
              if contesc = 1 then wait(lsem);
           signal(y);
           wait(esem);
                ESCRIBIR_UNIDAD;
            signal(esem);
            wait(y);
                  contesc := contesc 1;
                   if contesc = O then signal(lsem);
           signal(y);
    forever
   end;
begin
contlect, contesc := 0;
parbegin
      lector;
    escritor
    parend
end.
```

FIGURA 4.29b Una solución al problema de los lectores/escritores por medio de semáforos; los escritores tienen prioridad

mayor que el número máximo de lectores posible. En este ejemplo, se emplea un valor de 100. Las acciones del controlador pueden resumirse de la siguiente forma:

• Si *cont > 0,* no hay escritores esperando y puede haber o no lectores activos. Servir todos los mensajes "terminado" antes de eliminar los lectores activos. A continuación, servir las solicitudes de escritura y después las de lectura.

• Si *cont = 0,* la única solicitud pendiente es de escritura. Permitir al escritor continuar y esperar un mensaje "terminado".

• Si *cont < 0,* un escritor ha realizado una solicitud y se le ha hecho esperar para despejar todos los lectores activos. Por tanto, sólo deben servirse mensajes "terminado".

4.8 RESUMEN

Los temas centrales de los sistemas operativos modernos son la multiprogramación, el multipro-ceso y el proceso distribuido. Un punto fundamental en estos temas y en las tecnologías de diseño de sistemas operativos es la concurrencia. Cuando se ejecutan varios procesos concurrentemente, en el caso real de un sistema multiprocesador o en el caso virtual de un sistema monoprocesador multiprogramado, aparecen cuestiones de resolución de conflictos y de cooperación.

Los procesos concurrentes pueden interactuar de varias formas. Los procesos que no tienen conocimiento unos de otros pueden competir por recursos tales como el tiempo del procesador o los dispositivos de E/S. Los procesos pueden tener conocimiento indirecto de los otros porque comparten el acceso a unos objetos comunes, tales como un bloque de memoria principal o un archivo. Los procesos pueden tener un conocimiento directo de los otros y cooperar mediante intercambio de información. Los puntos clave que surgen en esta interacción son la exclusión mutua y el interbloqueo.

La exclusión mutua es una condición en la cual hay un conjunto de procesos concurrentes y sólo uno puede acceder a un recurso dado o realizar una función dada en cada instante de tiempo. Las técnicas de exclusión mutua pueden usarse para resolver conflictos, tales como

TABLA 4.5 Estado de las colas de procesos para el programa de Ia figura 4.29b

Sólo lectores en el sistema	• Activar *esem* • Sin colas
Sólo escritores en el sistema	• Activar *esem* y *Isem* • Los escritores se encolan en *esem*
Lectores y escritores con un lector primero	• *Esem* activado por un lector • *Isem* activado por un escritor • Todos los escritores se ponen en cola en *esem* • Un lector se pone en cola en *Isem* • Los otros lectores se ponen en cola en *z*
Lectores y escritores con un escritor primero	• Los escritores activan *esem* • Los lectores activan *Isem* • Todos los escritores se ponen en cola en esem • Un lector se pone en cola en *Isem* • Los otros lectores se ponen en cola en *z*

```
procedure lector_i;
var msjl: mensaje;
begin
    repeat
        msjl := i;
        send (solicitud_lectura, msjl);
        receive (buzon_i, msjl);
        LEER_UNIDAD;
        msjl := i;
        send (terminado, msjl)
    forever
end;
procedure escritor_j;
var msjl: mensaje;
begin
    repeat
        msjl := i;
        send (solicitud_escritura, msjl);
        receive(buzon_j, msje);
        ESCRIBIR_UNIDAD;
        msjl := i;
        send (terminado, msjl)
    forever
end;
```

```
procedure controlador;
var    msj: mensaje;
       cont: entero;
begin
    repeat
        if cont > 0 then begin
            if not vacío(terminado) then begin
                receive (terminado, msj);
                cont := cont + 1
            end
        else if not vacío(solicitud_escritura) then begin
            receive (solicitud_escritura, msj);
            escritor.id := msj.id;
            cont := cont –100
            end
        else if not vacío(solicitud_lectura) then begin
            receive (solicitud_lectura, msj);
            cont := cont – 1;
            send(msj.id, "OK")
            end
        end;
        if cont = 0 then begin
            send (escritor.id, "OK");
            receive (terminado, msj);
            cont := 100
        end;
        while cont < 0 do begin
            receive (terminado, msj);
            cont := cont + 1
            end
        forever
    end.
```

FIGURA 4.30 Una solución al problema de los lectores/escritores por medio de paso de mensajes

competencia por los recursos y para sincronizar procesos de modo que puedan cooperar. Un ejemplo de esto último es el modelo del productor/consumidor, en el que un proceso pone datos en un buffer y uno o más procesos los extraen.

Se han desarrollado varios algoritmos en software para ofrecer exclusión mutua, de los cuales el más conocido es el algoritmo de Dekker. Las soluciones por software suelen tener un alto coste y el riesgo de errores lógicos en el programa es también alto. Un segundo conjunto de métodos para soportar la exclusión mutua suponen el uso de instrucciones especiales de la máquina. Estos métodos reducen la sobrecarga, pero son aún ineficientes porque emplean espera activa.

Otro método para dar soporte a la exclusión mutua consiste en incluir las características dentro del sistema operativo. Dos de las técnicas más comunes son los semáforos y el paso de mensajes. Los semáforos se usan para la señalización entre procesos y pueden emplearse fácilmente para hacer respetar una disciplina de exclusión mutua. Los mensajes son útiles para el cumplimiento de la exclusión mutua y ofrecen también un medio efectivo de comunicación entre procesos.

4.9 LECTURAS RECOMENDADAS

A pesar de sus años, [BRIN73], que todavía se edita, ofrece uno de los mejores tratamientos de la concurrencia que se puede hallar en un libro de sistemas operativos. Tiene la ventaja adicional de incluir varios problemas resueltos. [BEN82] ofrece un estudio muy claro y entretenido de la concurrencia, la exclusión mutua, los semáforos y otros temas afines. Un estudio más formal, ampliado para incluir a los sistemas distribuidos, se encuentra en [BEN90]. [AXF088] es otro estudio ameno y útil; contiene también varios problemas resueltos. [RAYN86] es una colección clara y completa de algoritmos de exclusión mutua, que abarca soluciones por hardware y por software (por ejemplo, la de Dekker), además de semáforos y mensajes. [HOAR85] es un clásico muy ameno que introduce un método formal de definición de procesos secuenciales y concurrencia. [LAMP86] es un extenso tratamiento formal de la exclusión mutua. [RUD090] es una ayuda útil para comprender la concurrencia. [BAC093] es un estudio bien organizado de la concurrencia.

El artículo clásico sobre interbloqueo, [HOLT72], es todavía digno de ser leído, así como [COFF71], Otro buen estudio es [ISL080].

AXF088 AXFORD, T. *Concurrenty Programming: Fundamental Techniques for Real-Time ana Parallel Software Design.* Wiley, Nueva York, 1988.

BAC093 BACON, J. *Concurrent Systems.* Addison-Wesley, Reading, MA, 1993.

BEN82 BEN-ARI, M. *Principies of Concurrent Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

BEN90 BEN-ARI, M. *Principies of Concurrent and Distributed Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

BRIN73 BRINCH-HANSEN, P. *Operating System Principies* Prentice-Hall, Englewood Cliffs, NJ,1973.

COFF71 COPFMAN, E., ELPHICK, M. y SHOSHANI, A. "System Deadlocks". *Computing Surveys,* junio de 1971.

HOAR85 HOARE, C. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

HOLT72 HOLT, R. "Some Deadlock Properties of Computer Systems." *Computing Surveys,* septiembre de 1972.
ISLO80 ISLOOR, S. y MARSLAND, T. "The Deadlock Problem! An Overview". *Computer,* septiembre de 1980.
LAMP86 LAMPORT, L. "The Mutual Exclusion Problem". *Journal of the ACM,* abril de 1986.
RAYN86 RAYNAL, M. *Algorithms for Mutual Exclusion.* MIT Press, Cambridge, MA, 1986.
RUDO90 Rudolph, B. "Self-Assessment Procedure XXI: Concunency". *Communications oftheACM,* abril de 1990.

4.10

PROBLEMAS

4.1 Los procesos e hilos ofrecen una potente herramienta de estructuración para construir programas que serían mucho más complejos implementados como simples programas secuénciales. Una primera estructura que resulta instructivo examinar es la corrutina. El propósito de este problema es introducir las corrutinas y compararlas con los procesos. Considérese el siguiente problema de [CONW63]:

Leer tarjetas de 80 columnas e imprimirlas con 125 caracteres por línea, con los siguientes cambios. Después de cada tarjeta, introducir un blanco extra y cada pareja de asteriscos adyacentes (**) en la tarjeta se sustituye por el carácter _.

a) Desarrollar una solución a este problema como un programa secuencial ordinario. Se vera que el programa es difícil de escribir. Las interacciones entre los distintos elementos del programa son desiguales debido a la conversión de 80 a 125; es más, la longitud de la tarjeta, después de la conversión, variará dependiendo del número de dobles asteriscos. Una forma de mejorar la claridad y minimizar los errores potenciales, es escribir la aplicación como tres procedimientos separados. El primer procedimiento lee las tarjetas, rellena cada una con blancos y escribe una cadena de caracteres en un archivo temporal. Después de que se hayan leído todas las tarjetas, el segundo procedimiento lee el archivo temporal, realiza la sustitución de caracteres y escribe la salida en un segundo archivo temporal. El tercer procedimiento lee la cadena de caracteres del segundo archivo temporal e imprime líneas de 125 caracteres cada una.

b) La solución secuencial es poco atractiva debido a la sobrecarga de E/S y los archivos temporales. Conway propuso una nueva estructura de programa, la corrutina, que permite escribir una aplicación como tres programas conectados mediante buffers de un carácter (figura 4.31). En un procedimiento tradicional, hay una relación maestro/esclavo entre el procedimiento llamado y el llamador. El procedimiento llamador puede ejecutar una llamada desde cualquier punto; el procedimiento llamado comienza en su punto de entrada y retoma al procedimiento llamador en el punto de llamada. La corrutina expone una relación más simétrica. Cuando se hace cada llamada, la ejecución pasa al último punto activo en el procedimiento llamado. Puesto que no tiene sentido que el procedimiento llamador sea "mayor" que el llamado, no hay retorno. En cambio, cualquier corrutina puede pasar el control a cualquier otra con la orden reanudar. La primera vez que se llama a una corrutina, se "reanuda" en su punto de entrada. Posteriormente, la corrutina se reactiva en el punto de su última orden reanudar. Nótese que solo una corrutina puede estar en ejecución en un programa en cada instante y que los puntos de transición están definidos explícitamente en el código, por lo que esto no es un ejemplo de proceso concurrente. El funcionamiento del programa se explica en la figura 4.31.

```
var   rs, sp: caracter;
      bufent: array  of caracter;
      bufsal: array ~1.125J of caracter;
procedure leer;
begin
    repeat
      LEER_CARACTER(bufent);
      for i=l to 80 do
       begin
          rs := bufent[i];
          REANUDAR comprimir;
       end;
     rs :=" ";


      REANUDAR comprimir
      forever
    end;
    procedure imprimir;
    begin
       repeat
      for j=1 to 125
       begin
          bufsal~j} := sp;
          REANUDAR comprimir
       end;
     ESCRJBIR(bufsal)
  forever
end;
procedure comprimir;
begin
  repeat
    if rs <> "*" then
      begin
        sp := rs;
        REANUDAR imprimir;
      end
      else begin
       REANUDAR leer;
       if rs="*" then
         begin
           sp := "↑    "
           REANUDAR imprimir;
                end
             else begin
                sp := "*";
                REANUDAR imprimir;
                sp := rs;
                REANUDAR imprimir end
           end
           REANUDAR leer
         forever
      end.
```

FIGURA 4.31 Una aplicación de las corrutinas

c) El programa no tiene en cuenta la condición de terminación. Supóngase que la rutina de E/S LEER_TARJETA devuelve un valor cierto si ha situado una imagen de 80 caracteres en *bufent* y falso en cualquier otro caso. Modifíquese el programa para que tenga en cuenta este imprevisto. Nótese que la última línea impresa puede contener, por tanto, menos de 125 caracteres.

d) Reescribir la solución como un conjunto de tres procesos con semáforos.

4.2 Considérese un programa concurrente con los dos procesos, P y Q, que se muestran a continuación. A, B, C, D y E son sentencias arbitrarias atómicas (indivisibles). Supóngase que el programa principal (no mostrado) hace un **parbegin** de los dos procesos.

```
procedure P;        procedure Q;
begin               begin
A;                  D;
B;                  E;
C;                  end.
end.
```

Indicar todas las posibles intercalaciones de la ejecución de los dos procesos anteriores (indicarlo por medio de "trazas" de ejecución dadas en términos de sentencias atómicas).

4.3 Considérese el siguiente programa:

```
const n=50;
var cuenta: entero;
procedure total;
```

```
var cont: entero;
   begin
        for cont := ito n do cuenta := cuenta + 1; end;
begin (*programa principal*)
  cuenta := 0;
  parbegin
     total; total
  parend;
  writeln(cuenta)
end.
```

a) Determinar los límites inferior y superior adecuados para el valor final de la variable compartida *cuenta* escrita en la salida por este programa concurrente. Supóngase que los procesos pueden ejecutar a cualquier velocidad relativa y que un valor solo puede incrementarse después de que haya sido cargado en un registro por una instrucción separada de la máquina.

b) Supóngase que se permite ejecutar en paralelo a un número arbitrario de estos procesos bajo las suposiciones del apartado (a). ¿Qué efecto tendrá esta modificación en el rango de valores finales de *cuenta?*

4.4 ¿Es siempre la espera activa menos eficiente (en términos de utilización del procesador) que la espera bloqueante? Explíquese.

4.5 Considérese el siguiente programa:

```
var  bloqueado: array turno: [0..1] of booleano;
turno: 0..1;
procedure P(id: entero);
begin
  repeat
     bloqueado[id] := cierto;
     while tumo <>id do
       begin
          while bloqueado~ i _idi do {nada };
          turno := id
          end;
       <sección crítica>
       bloqueado[id] := falso;
       <resto>
    until falso
end;
begin
 bloqueado[0] := falso; bloqueado1] := falso; tumo
 := 0;
   parbegin
      P(0); P(1)
   parend

end.
```

Esta es una solución por software a! problema de la exclusión mutua propuesto en [HYMA66]. Encontrar un contraejemplo que demuestre que esta solución es incorrecta. Es interesante destacar que incluso en las *Comunicaciones de la ACM* se equivocaron con esta solución.

4.6 Probar la corrección del algoritmo de Dekker.

a) Probar que se respeta la exclusión mutua.
   Indicación: Demostrar que cuando Pi entra en su sección crítica, es cierta la siguiente expresión:

   Señal[i] and (not señal[1 - i])

b) Probar que se evita el interbloqueo.
   Indicación:
   Considérense los siguientes casos: (1) un único proceso intenta entrar en la sección crítica; (2) ambos procesos intentan entrar en la sección crítica y (2a) tumo = 0 y señal = falso y (2b) tumo = 0 y señal = cierto.

4.7 Considérese el algoritmo de Dekker, escrito para un número arbitrario de procesos, cambiando el protocolo de salida que se ejecuta cuando se abandona la sección crítica de

```
tumo := 1 - i
     (*es decir, P0 pone turno a i y P1 pone tumo a
   0*)
```

por

```
turno := (turno + 1) mod n
```

(* n = número de procesos *)

Evaluar el algoritmo cuando el número de procesos que ejecutan concurrentemente es mayor de dos.

4.8 El algoritmo de Peterson puede generalizarse para ofrecer exclusión mutua entre *N* procesos. Supónganse dos vectores globales q y *turno.* Los valores iniciales de los *N* elementos de q y de los *N* — 1 elementos de *turno* son todo ceros. Cada

```
repeat
1  for j := 1 to N – 1 do
2     q[i] := j;
3  turno[j] := i;
4  L: for k=1 to i – 1, i+1 do N
5        if ((q[k] ≤ j) and (turno[j] = i)) goto L;
6  q[i] = n;

sección crítica del proceso i;

7  q[i] := 0;

resto del proceso i
until falso;
```

proceso mantiene las variables privadas *j* y *k,* que se usan como índices del vector. El algoritmo para el proceso *i* es como sigue:

vectores globales enteros q[N], *turno[N*- 1]

Es conveniente considerar el valor de la variable local j como la "etapa" del algoritmo en la que ejecuta el proceso *i.* El vector global q indica la etapa de cada proceso. Cuando un proceso entra en su fase crítica, pasa a la etapa *N.* (Esto lo realiza la sentencia *q[i]*:= *N.* Realmente esta sentencia existe solo para simplificar el lenguaje usado en la demostración y es innecesaria para la corrección del algoritmo).

Si *q[x]* > q[y], se puede decir que el proceso x precede (está por delante) al proceso y.

Se busca demostrar que este algoritmo proporciona:

- Exclusión mutua
  No tiene interbloqueo
- No tiene inanición

Para hacerlo, demostrar los Siguientes lemas:

a) Lema 1: Un proceso que precede a todos los demás puede avanzar a! menos una etapa.

Indicación: Sea un proceso *i* tal que, al comienzo de la línea 4,

j = q[*i*] > *q[k]* para todo *k* - *i*

b) Lema 2: Cuando un proceso pasa de una etapa *j* a otra *j+1,* se cumple exactamente uno de los siguientes requisitos:

- Precede a todos los otros procesos o
- No es el único en la etapa j.

Indicación: Hacer que el proceso *i* avance hasta q[*i*] y considerar si la ecuación (1) es cierta **0** no.

c) Lema 3: Si hay (al menos) dos procesos en la etapa j, hay (a! menos) un proceso en cada etapa k, 1 ≤ *k*≤ *j*- **1.**

Indicación: Probarlo por inducción en j.

d) Lema 4: El máximo número de procesos que pueden estar en la etapa j es de N- j **+ 1, 1** ≤ *j*≤ *N* - **1.**

Indicación: Es simple aritmética.

e) Sobre la base de los lemas del 1 al 4, demostrar que el algoritmo ofrece exclusión mutua, está libre de interbloqueo y no presenta inanición.

4.9 Otro método de software para la exclusión mutua es el **algoritmo de la panadería** de Lamport [LAMP74I, llamado así porque está basado en la costumbre de las panaderías y de otras tiendas de que cada cliente recibe un número al llegar, lo que permite servirles por turnos. El algoritmo es como sigue:

```
var elección: array [0..n - 1] of booleano;
    número: array [0..n-n - 1] of entero;
repeat
  elección[i] := cierto;
  número[i] := 1 + max (número[0],
   número[l],..., número[n-lj);
  elección[i] := falso;
  for j := 0 to n- 1 do
  begin
        while elección[j] do (nada};
     while número[j] ≠ and (número[j], j)
     < (número[i], i) do (nada);
```

end;
<sección crítica>
número[i] := 0;
<resto>
forever

Los vectores *elección* y *número* se inicializan a falso y a 0, respectivamente. El *i*-ésimo elemento de cada vector puede ser leído y modificado por el proceso *i,* pero solo puede ser leído por otros procesos. La notación (a, *b) < (c, d)* se define como:
(a<c) or *(a=c and b<d)*
a) Describir el algoritmo con palabras.
b) Demostrar que este algoritmo evita al interbloqueo.
c) Demostrar que respeta la exclusión mutua.
d) Si hay siempre, por lo menos, un proceso con un número de ticket, mientras el mayor está siendo procesado, los valores de *número* se hacen indefinidamente largos. Modificar el algoritmo para eliminar este problema. Indicación: Hacer que el máximo valor de cualquier elemento del vector sea *2n* - 1.

4.10 Demostrar que los siguientes métodos de software para la exclusión mutua no dependen de la exclusión mutua básica del nivel del acceso a memoria.
a) El algoritmo de la panadería.
b) El algoritmo de Peterson.

4.11 Cuando se usa una instrucción especial de la máquina para ofrecer exclusión mutua, de la manera de la figura 4.12, no hay control sobre cuánto tiempo debe esperar un proceso antes de obtener acceso a su sección crítica. Idear Un algoritmo que use la instrucción TS pero que garantice que cualquier proceso esperando para entrar en su sección crítica lo hará dentro de *n* - 1 turnos, donde *n* es el número de procesos que pueden solicitar posiblemente acceso a la sección critica y un "tumo" es un suceso consistente en que un proceso abandona la sección critica y otro proceso obtiene el acceso.

4.12 Supóngase que, en el instante *5,* no se está usando ningún recurso del sistema, excepto el procesador y la memoria. Considérense, ahora, los siguiente sucesos:

En el instante 5, P1 ejecuta una orden de lectura de la unidad de disco 3.
En el instante 15,Termina La fracción de tiempo de P4.
En el instante 18,P7 ejecuta una orden de escritura en la unidad de disco 3.
En el instante 20,P3 ejecuta una orden de lectura de la unidad de disco 2.
En el instante 24,P4 ejecuta una orden de escritura en la unidad de disco 3.
En el instante 28,P2 solicita ejecutar una operación *wait* en un semáforo *Buf* con valor 2.
En el instante 33,se produce una interrupción de la unidad de disco 2; la lectura de P3 ha terminado.
En el instante 34,P3 solicita ejecutar una operación *wait* sobre el semáforo *Buf.*
En el instante 36,se produce una interrupción de la unidad de disco 3; la Lectura de P1 ha terminado.
En el instante 38,P8 termina.
En el instante 40,se produce una interrupción de la unidad de disco 3; La escritura de P4 ha terminado.
En el instante 44,P4 solicita ejecutar una operación *wait* sobre el semáforo *Cnt* con valor 0.
En el instante 48,se produce una interrupción de la unidad de disco 3; la escritura de P7 ha terminado.

Identificar los procesos conocidos que están en estado de espera por *wait* (no en estado Listo) y decir a qué suceso está esperando cada uno:
a) En el instante 22.
b) En el instante 37.
c) En el instante 47.

4.13 Considérese la siguiente definición de semáforo:

```
wait(s):
if s.cont> 0
  then
    s.cont := s.cont— 1
  else begin
    poner este proceso en s.cola;
    bloquearlo;
  end;
signal(s):

if hay al menos un proceso suspendido
  en el semáforo s
  then begin
    retirar un proceso P de s.cola;
    poner P en la cola de listos
  end
  else
      s.cont := s.cont + 1
              end;
```

Comparar este conjunto de instrucciones con las de la figura 4.13. Nótese una diferencia:

Con la definición anterior, un semáforo nunca puede tomar un valor negativo. ¿Hay alguna diferencia en el efecto de las dos definiciones cuando se usan en los programas? Es decir, Les posible sustituir una definición por la otra sin alterar el significado del programa?

4.14 Debe ser posible implementar semáforos generales por medio de semáforos binarios. Se pueden usar las operaciones *WaitB* y *SignaiB* y dos semáforos binarios *espera* y *exmut.* Considérese lo siguiente:

```
procedure Wait(var s: semaforo);
begin
  WaitB(exmut);
  S =5 - 1;

  if s <0 then begin
    SignalB(exmut); WaitB(espera)
  end;
  else SignalB(exmut)
end;
    procedure Signal(var s: semáforo);
    begin
      WaitB(exmut);
      s := S + 1;
      if s _ 0 then SignalB(espera);
      SignalB(exmut)
    end;
```

Inicialmente, s tiene el valor deseado para el semáforo. Cada operación *Wait* decrementa *s y* cada operación *Signal* lo incrementa. El semáforo binario *exmut,* inicializado a 1, asegura que hay exclusión mutua en las actualizaciones de s.

El semáforo binario *espera,* inicializado a 0, se usa para suspender los procesos.

Hay un defecto en el programa anterior. Encontrar el defecto y proponer un cambio que lo corrija. Indicación: Todo lo que se necesita es mover una línea del programa.

4.15 El siguiente problema se empleo una vez en Un examen:

Parque Jurásico está formado por un museo de dinosaurios y un parque para excursiones de safari. Hay *m* pasajeros y *n* coches monoplaza. Los pasajeros dan vueltas por el museo durante un tiempo y después se ponen en lila para dar un paseo en un coche de safari. Cuando el coche está disponible, carga un pasajero y recorre el parque durante un tiempo aleatorio. Si los *n* coches están todos dando vueltas a los pasajeros, los que quieren subir deben esperar; si un coche está listo para recoger pasajeros, pero no hay ninguno esperando, el coche debe esperar.

Usar semáforos para sincronizar los procesos de los *m* pasajeros con los de los *n* coches.

El siguiente esqueleto de código se encontró en un pedazo de papel en el suelo de la sala de exámenes. Determinar si es correcto. Ignorar la sintaxis y las declaraciones ausentes de variables.

```
\begin { literalmente}
recurso Parque_Jurasico()
  sem    coche vació:=O,tomar_coche:=0,coche_
  lleno: = 0, pasaj_libre:=0
  process pasajero(i: = 1 to num_pasajeros)
  do true → dormir(int(random(1000* tiempo_
    paseo)))
```

P(coche_vacío); V(tomar_coche); P(co-che_ lleno) P(pasaj_libre) od

end pasajero

process coche(j:=l to num_coches)

do true -» V(coche_libre); P(tomar_ coche); V(coche_lleno)
dormir(int(random(1000*tiempo_recorrido))
V(pasaj_libre)

od

end coche

end Parque_Jurasico

\end {literalmente}

4.16 Considérese la solución al problema del productor/consumidor con buffer ilimitado definido en la figura 4.19. Supóngase que se tiene el caso (habitual) en el que el productor y el consumidor están ejecutando aproximadamente a la misma velocidad. La situación podría ser como sigue:

- Productor: añadir; señalizar; producir; ...; añadir; señalizar; producir;...
- Consumidor: consumir; ...; tomar; esperar; consumir;...; tomar; esperar;...

El productor siempre añade un elemento al buffer y señaliza mientras el consumidor consume el elemento previo. El productor está siempre añadiendo a un buffer vacío y el consumidor está siempre tomando el único elemento del buffer. Aunque el consumidor nunca se bloquea en el semáforo, se realiza un gran número de llamadas al mecanismo de éste, originando una sobrecarga considerable.
Construir un nuevo programa que sea más eficiente bajo estas circunstancias. Indicación:
Permitir que *n* tome el valor -1, lo que significa que no sólo el buffer está vacío sino que el consumidor ha detectado este hecho y se va a bloquear hasta que el productor proporcione datos nuevos. La solución no necesita el uso de la variable local *m* de la figura 4.15.

4.17 Considérese la figura 4.18. ¿Cambiará el significado del programa si se intercambian las siguientes sentencias?
a) wait(e); wait(s)

b) signal(s); signal(n)

c) wait(n); wait(s)

d) signal(s); signal(e)

4.18 En el apartado sobre el problema del productor/consumidor, nótese que la definición permite como máximo *n* - 1 entradas en el buffer.

a) ¿Porqué?
a) Modificar el algoritmo para remediar esta deficiencia.

4.19 Responder a las siguientes cuestiones relativas a la barbería equitativa (figura 4.22):
a) ¿Requiere el código que cobre el pago de un corte de pelo a un cliente el mismo barbero que lo terminó?

b) ¿Usan los barberos siempre la misma silla?

4.20 Quedan por resolver una serie de problemas del barbero equitativo de la figura 4.22. Modificar el programa para corregir los siguientes problemas:
a) El cajero puede aceptar el pago de un cliente y liberar otro si hay dos o más esperando para pagar. Afortunadamente, una vez que un cliente ofrece el pago, no hay forma de retirarlo, así que, al final, en la caja registradora habrá la cantidad correcta de dinero. Sin embargo, es conveniente liberar el cliente correcto tan pronto como se haya aceptado su pago.
b) El semáforo *dejar_silla_b* impide supuestamente los accesos múltiples a un única silla de barbero. Desafortunadamente, este semáforo no lo consigue en todos los casos. Por ejemplo, supóngase que los tres barberos han terminado de cortar el pelo y están bloqueados en *wait(dejar_silla_b). Dos* de los clientes están en estado interrumpido inmediatamente antes de *abandonar la silla de barbero.* El tercer cliente abandona su silla y ejecuta *sig-nal(dejar_silla_b).* ¿Qué barbero se libera? Como la cola *dejar_silla_b* es del tipo primero en llegar/primero en salir, el primer barbero que se bloqueó será el que se libere. ¿Es éste el barbero que estaba cortando el pelo al cliente que dio la señal? Puede que sí o puede que no. Si no lo es, llegará un nuevo cliente y se sentará en la rodillas de un cliente que iba a levantarse en ese instante.

c) El programa exige que los clientes se sienten vacía. Es verdad que éste es un problema menor y que solucionarlo hace que el código resultante esté un poco (bastante) desordenado. A pesar de todo, conviene intentarlo,

4.21 Demostrar que los monitores y los semáforos tienen una funcionalidad equivalente:
a) Implementando un monitor por medio de semáforos. b) Implementando un semáforo por medio de monitores.

CAPITULO 5

Concurrencia: interbloqueo e inanición

Este capítulo continúa el estudio de la concurrencia considerando dos problemas que importunan todos los intentos de realizar procesamiento concurrente: el interbloqueo y la inanición. El capítulo comienza con una exposición de los principios básicos del interbloqueo y de un problema afín, la inanición. A continuación, se examinan las tres soluciones más comunes para hacer frente al interbloqueo: prevención, detección y predicción. Posteriormente se atiende a uno de los problemas clásicos empleados para ilustrar las cuestiones de sincronización e interbloqueo: el problema de la cena de los filósofos.

Al igual que el capítulo 4, este capítulo se limita a tratar la concurrencia y el interbloqueo en sistemas monoprocesador. Las medidas para hacer frente a los interbloqueos distribuidos se pueden apreciar en el capítulo 13.

PRINCIPIOS DEL INTERBLOQUEO

[MAEK87] define el interbloqueo como el bloqueo *permanente* de un conjunto de procesos que compiten por los recursos del sistema o bien se comunican unos con otros. A diferencia de otros problemas de la gestión concurrente de procesos, para el caso general no existe una solución eficiente. En esta sección, se examinará la naturaleza del problema del interbloqueo.

Todos los interbloqueos suponen demandas contradictorias de recursos por parte de dos o más procesos. La figura 5.1 ilustra este conflicto de forma abstracta en el caso de dos procesos y dos recursos. Los dos ejes del diagrama representan el avance de los dos procesos en términos de instrucciones ejecutadas. El avance conjunto de los dos procesos se representa entonces con una secuencia discreta de puntos en el espacio. Las líneas horizontales o verticales representan el intervalo de tiempo en el que sólo uno de los procesos está ejecutándose *(intercalado);* una línea diagonal significa ejecución simultánea (solapamiento). Supóngase que existe un punto en la ejecución de cada proceso en el que se requiere el uso exclusivo de ambos recursos, Rl y R2, para continuar. En el ejemplo, llega un punto en el que el proceso Pl ha adquirido el recurso Rl y el proceso P2 ha adquirido el recurso R2 y cada proceso necesita el otro recurso. Este es el punto de interbloqueo.

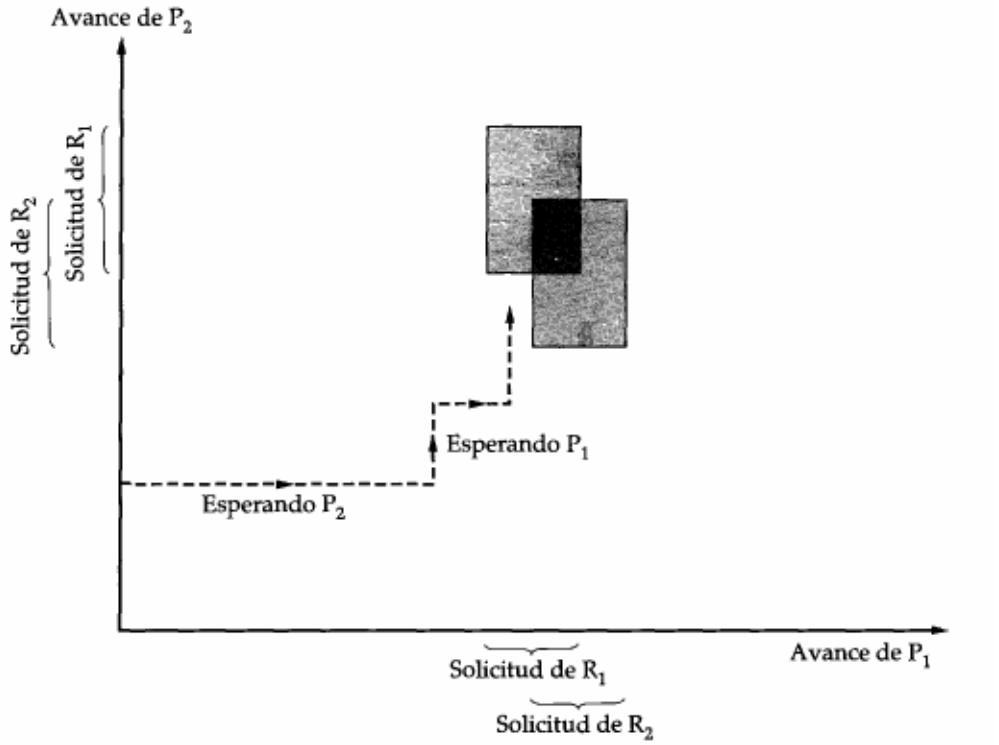


FIGURA 5.1 Avance conjunto de los procesos P1 y P2 [COFF71]

Recursos Reutilizables

Se pueden distinguir dos categorías generales de recursos: reutilizables y consumibles. Un recurso reutilizable es aquél que puede ser usado con seguridad por un proceso y no se agota con el uso. Los procesos obtienen unidades de recursos que liberan posteriormente para que otros procesos las reutilicen. Como ejemplos de recursos reutilizables se tienen los procesadores, canales de E/S, memoria principal y secundaria, dispositivos y estructuras de datos tales como archivos, bases de datos y semáforos.

Como ejemplo de interbloqueo con recursos reutilizables, considérense dos procesos que compiten por el acceso exclusivo a un archivo D del disco y a una unidad de cinta T. Los programas están dedicados a las siguientes operaciones repetitivas:

```
P1                          P2
repeat                      repeat
    Solicitar (D);              Solicitar (T);
    Solicitar (T);              Solicitar (D);
    Liberar (T);                Liberar (D);
    Liberar (D);            Liberar (T);
forever                     forever
```

El interbloqueo se produce si cada proceso retiene un recurso y solicita el otro. Puede parecer que es un error de programación en lugar de un error del diseño del sistema operativo. Sin embargo, se ha visto que el diseño de un programa concurrente entraña gran dificultad. Se producen interbloqueos como éste y la causa está frecuentemente en la compleja lógica del programa, lo que hace más difícil su detección. Una posible estrategia para resolver estos interbloqueos es imponer restricciones en el diseño del sistema sobre el orden en el que se solicitan los recursos.

Otro ejemplo de interbloqueo con un recurso reutilizable tiene que ver con las peticiones a memoria principal. Supóngase que el espacio disponible es de 200KB y se origina la siguiente secuencia de peticiones:

P1	**P2**
Solicitar 80K bytes;	Solicitar 70 K bytes;
Solicitar 60 K bytes;	Solicitar 80K bytes;

Se produce un interbloqueo si ambos procesos avanzan hasta su segunda petición. Si la cantidad de memoria que van a solicitar no se conoce con antelación, resulta difícil enfrentarse a este tipo de interbloqueo por medio de restricciones en el diseño del sistema. La mejor forma de resolver este problema en particular es, de hecho, eliminar la posibilidad, por medio de la memoria virtual, que se tratará en el capítulo 7.

Recursos Consumibles

Un recurso consumible es aquél que puede ser creado (producido) y destruido (consumido). Normalmente, no hay límite en el número de recursos consumibles de un tipo en particular. Un proceso productor que no está bloqueado puede liberar cualquier número de recursos consumibles. Cuando un proceso adquiere un recurso, éste deja de existir. Como ejemplos de recursos consumibles están las interrupciones, señales, mensajes, e información en buffers de E/S.

Como ejemplo de interbloqueo con recursos consumibles, considérese el siguiente par de procesos:

P1	**P2**
Recibir (P2, M);	Recibir (P1, Q);
Enviar (P2, N);	Enviar (P1, R);

El interbloqueo se produce si el receptor se bloquea. De nuevo, la causa del interbloqueo es un error de diseño. Estos errores pueden ser bastante sutiles y difíciles de detectar. Es más, puede darse una combinación de sucesos poco habitual que origine el interbloqueo; así pues, un programa puede funcionar durante un periodo de tiempo considerable, incluso años, antes de que el problema se manifieste.

No hay ninguna estrategia sencilla que pueda solucionar todas las clases de interbloqueo. La tabla 5.1 resume los elementos clave de los enfoques más importantes que se han tomado: detección, prevención y predicción. A continuación se examinará cada uno de ellos.

Condiciones de Interbloqueo

Deben darse tres condiciones para que pueda producirse un interbloqueo:

1. *Exclusión mutua:* Sólo un proceso puede usar un recurso simultáneamente.

TABLA 5.1 Resumen de los enfoques de detección, prevención y predicción del interbloqueo en los sistemas operativos [ISLO80]

Principio	Política de Reparto de Recursos	Esquemas	Ventajas Principales	Desventajas Principales
Prevención	Conservadora; los recursos están poco ocupados.	Solicitud de todos los recursos a la vez	• Funciona bien con procesos que realizan una sola ráfaga de actividad. • No es necesaria la apropiación.	• Ineficiencia. • Retrasos en el inicio de los procesos.
		Apropiación	• Conveniente cuando se aplica a recursos cuyo estado se puede salvar y restaurar fácilmente.	• Apropia más habitualmente de lo necesario. • Sujeto a reinicios cíclicos.
		Ordenación de recursos	• Aplicación factible por medio de chequeos durante la compilación. • No hace falta procesamiento durante la ejecución, pues el problema se resuelve durante el diseño del sistema.	• No permite las solicitudes incrementales de recursos. • Poco uso de la apropiación.
Detección	Muy liberal; los recursos solicitados se conceden cuando es posible.	Comprobación periódica del interbloqueo.	• Nunca retrasa el inicio de un proceso • Facilita el manejo en línea	• Pérdidas inherentes a la apropiación.
Predicción	A medio camino entre la detección y la prevención.	Manipulación para encontrar al menos un camino seguro.	• No es necesaria la apropiación.	• Deben conocerse las demandas futuras de recursos. • Los procesos pueden

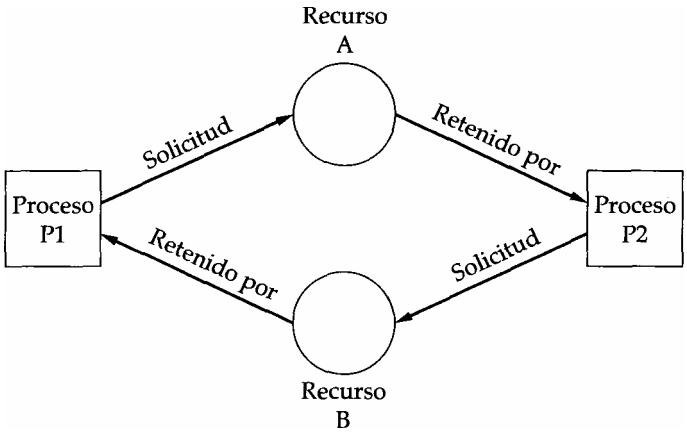


FIGURA *5.2* Espera circular

2. *Retención y esperar:* Un proceso puede retener unos recursos asignados mientras espera que se le asignen otros.

3. *No apropiación:* Ningún proceso puede ser forzado a abandonar un recurso que retenga.

En la mayoría de los casos, estas condiciones son bastante necesarias. Por ejemplo, la exclusión mutua hace falta para asegurar la consistencia de resultados y la integridad de la base de datos. De forma similar, la expulsión o apropiación no se puede aplicar arbitrariamente y, cuando se encuentran involucrados recursos de datos especialmente, debe estar acompañada de un mecanismo de recuperación y reanudación, que devuelva a un proceso y a sus recursos a un estado previo adecuado, desde el que el proceso pueda finalmente repetir sus acciones.

Puede existir interbloqueo con estas tres condiciones, pero puede no existir con *sólo* estas tres condiciones. Para que se produzca interbloqueo, se necesita una cuarta condición:

4. *Círculo vicioso de espera:* Existe una cadena cerrada de procesos, cada uno de los cuales retiene, al menos, un recurso que necesita el siguiente proceso de la cadena (véase la figura 5.2).

La tres primeras condiciones son necesarias, pero no suficientes, para que exista interbloqueo. La cuarta condición es, en realidad, una consecuencia potencial de las tres primeras. Es decir, dado que se producen las tres primeras condiciones, puede ocurrir una secuencia de eventos que desemboque en un círculo vicioso de espera irresoluble. Una círculo de espera irresoluble es, de hecho, la definición de interbloqueo. El círculo de espera de la condición 4 es irresoluble porque se mantienen las tres primeras condiciones. Es decir, las cuatro condiciones en conjunto constituyen una condición necesaria y suficiente para el interbloqueo. '

' Prácticamente todos los libros enumeran simplemente estas cuatro condiciones como condiciones necesarias para el interbioqueo, pero dicha exposición oculta algunas cuestiones sutiles [SHUB9]. La condición 4, la del círculo vicioso de espera, es muy diferente de las otras tres. Las condiciones de la 1 a la 3 son decisiones de política, mientras que la condición 4 es una circunstancia que podría darse en función de la secuencia de solicitudes y liberaciones de los procesos. Combinar el círculo vicioso de espera con las otras tres condiciones necesarias conduce a una distinción poco apropiada entre prevención y predicción.

5.2 PREVENCIÓN DEL INTERBLOQUEO

La estrategia de prevención del interbloqueo consiste, a grandes rasgos, en diseñar un sistema de manera que esté excluida, a priori, la posibilidad de interbloqueo. Los métodos para prevenir el interbloqueo son de dos tipos. Los métodos indirectos consisten en impedir la aparición de alguna de las tres condiciones necesarias, antes mencionadas (condiciones 1 a 3). Los métodos directos consisten en evitar la aparición del círculo vicioso de espera (condición 4). Se examinarán a continuación las técnicas relacionadas con cada una de las cuatro condiciones.

Exclusión Mutua

En general, la primera de las cuatro condiciones no puede anularse. Si el acceso a un recurso necesita exclusión mutua, el sistema operativo debe soportar la exclusión mutua. Algunos recursos, como los archivos, pueden permitir varios accesos para lectura, pero sólo accesos exclusivos para escritura. Incluso en este caso, se puede producir interbloqueo si más de un proceso necesita permiso de escritura.

Retención y Espera

La condición de retención y espera puede prevenirse exigiendo que todos los procesos soliciten todos los recursos que necesiten a un mismo tiempo y bloqueando el proceso hasta que todos los recursos puedan concederse simultáneamente. Esta solución resulta ineficiente por dos factores. En primer lugar, un proceso puede estar suspendido durante mucho tiempo, esperando que se concedan todas sus solicitudes de recursos, cuando de hecho podría haber avanzado con sólo algunos de los recursos. Y en segundo lugar, los recursos asignados a un proceso pueden permanecer sin usarse durante periodos considerables, tiempo durante el cual se priva del acceso a otros procesos.

No apropiación

La condición de no apropiación puede prevenirse de varias formas. Primero, si a un proceso que retiene ciertos recursos se le deniega una nueva solicitud, dicho proceso deberá liberar sus recursos anteriores y solicitarlos de nuevo, cuando sea necesario, junto con el recurso adicional. Por otra parte, si un proceso solicita un recurso que actualmente está retenido por otro proceso, el sistema operativo puede expulsar al segundo proceso y exigirle que libere sus recursos. Este último esquema evitará el interbloqueo sólo si no hay dos procesos que posean la misma prioridad.

Esta técnica es práctica sólo cuando se aplica a recursos cuyo estado puede salvarse y restaurarse más tarde de una forma fácil, como es el caso de un procesador.

Círculo Vicioso de Espera

La condición del círculo vicioso de espera puede prevenirse definiendo una ordenación lineal de los tipos de recursos. Si a un proceso se le han asignado recursos de tipo R, entonces sólo podrá realizar peticiones posteriores sobre los recursos de los tipos siguientes a R en la ordenación.

Para comprobar el funcionamiento de esta estrategia, se asocia un índice a cada tipo de recurso. En tal caso, el recurso R, antecede a R, en la ordenación si $i < j$. Entonces, supón-

gase que dos procesos A y B se interbloquean, porque A ha adquirido R, y solicitado Ry, mientras que B ha adquirido R;y solicitado R¿. Esta situación es imposible porque implica que *í<jyj<i.*

Como en la retención y espera, la prevención del círculo vicioso de espera puede ser ineficiente, retardando procesos y denegando accesos a recursos innecesariamente.

5.3

DETECCIÓN DEL 1NTERBLOQUEO

Las estrategias de prevención del interbloqueo son muy conservadoras; solucionan el problema del interbloqueo limitando el acceso a los recursos e imponiendo restricciones a los procesos. En el lado opuesto, las estrategias de detección del interbloqueo no limitan el acceso a los recursos ni restringen las acciones de los procesos. Con detección del interbloqueo, se concederán los recursos que los procesos necesiten siempre que sea posible. Periódicamente, el sistema operativo ejecuta un algoritmo que permite detectar la condición de círculo vicioso de espera descrita en el punto 4 anterior e ilustrada en la figura 5.2. Puede emplearse cualquier algoritmo de detección de ciclos en grafos dirigidos (véase [LEIB89]).

El control del interbloqueo puede llevarse a cabo tan frecuentemente como las solicitudes de recursos o con una frecuencia menor, dependiendo de la probabilidad de que se produzca el interbloqueo. La comprobación en cada solicitud de recurso tiene dos ventajas: Conduce a una pronta detección y el algoritmo es relativamente simple, puesto que está basado en cambios increméntales del estado del sistema. Por otro lado, tal frecuencia de comprobaciones consume un tiempo de procesador considerable.

Una vez detectado el interbloqueo, hace falta alguna estrategia de recuperación. Las técnicas siguientes son posibles enfoques, enumeradas en orden creciente de sofisticación:

1. Abandonar todos los procesos bloqueados. Esta es, se crea o no, una de las soluciones más comunes, si no la más común, de las adoptadas en un sistema operativo.

2. Retroceder cada proceso interbloqueado hasta algún punto de control definido previamente y volver a ejecutar todos los procesos. Es necesario que haya disponibles unos mecanismos de retroceso y reinicio en el sistema. El riesgo de esta solución radica en que puede repetirse el interbloqueo original. Sin embargo, el no determinismo del procesamiento concurrente asegura, en general, que esto no va a pasar.

3. Abandonar sucesivamente los procesos bloqueados hasta que deje de haber interbloqueo. El orden en el que se seleccionan los procesos a abandonar seguirá un criterio de mínimo coste. Después de abandonar cada proceso, se debe ejecutar de nuevo el algoritmo de detección para ver si todavía existe interbloqueo.

4. Apropiarse de recursos sucesivamente hasta que deje de haber interbloqueo. Como en el punto 3, se debe emplear una selección basada en coste y hay que ejecutar de nuevo el algoritmo de detección después de cada apropiación. Un proceso que pierde un recurso por apropiación debe retroceder hasta un momento anterior a la adquisición de ese recurso.

Para los puntos 3 y 4, el criterio de selección podría ser uno de los siguientes, consistentes en escoger el proceso con:

• La menor cantidad de tiempo de procesador consumido hasta ahora

- El menor número de líneas de salida producidas hasta ahora
- El mayor tiempo restante estimado
- El menor número total de recursos asignados hasta ahora
- La prioridad más baja

Algunas de estas cantidades son más fáciles de medir que otras. El tiempo restante estimado deja lugar a dudas, especialmente. Además, aparte de las medidas de prioridad, no existe otra indicación del "coste" para el usuario frente al coste para el sistema en conjunto.

5.4 PREDICCIÓN DEL INTERBLOQUEO

Otra forma de resolver el problema del interbloqueo, que se diferencia sutilmente de la prevención, es la predicción de interbloqueo.[2] En la prevención de interbloqueo, se obligaba a las solicitudes de recursos a impedir que sucediera, por lo menos, alguna de las cuatro condiciones de interbloqueo. Esto se hace indirectamente, impidiendo la aparición de una de las tres condiciones necesarias (exclusión mutua, retención y espera, no apropiación) o directamente, impidiendo la aparición de un círculo vicioso de espera. Se llega así a un uso ineficiente de los recursos y una ejecución ineficiente de los procesos. Con predicción del interbloqueo, por otro lado, se pueden alcanzar las tres condiciones necesarias, pero se realizan elecciones acertadas para asegurar que nunca se llega al punto de interbloqueo. La predicción, por tanto, permite más concurrencia que la prevención. Con predicción del interbloqueo, se decide dinámicamente si la petición actual de asignación de un recurso podría, de concederse, llevar potencialmente a un interbloqueo. La predicción del interbloqueo necesita, por tanto, conocer las peticiones futuras de recursos.

En este apartado se van a describir los dos siguientes enfoques para la predicción del interbloqueo:

- No iniciar un proceso si sus demandas pueden llevar a interbloqueo.
- No conceder una solicitud de incrementar los recursos de un proceso si esta asignación puede llevar a interbloqueo.

Negativa de Iniciación de Procesos

Considérese un sistema de *n* procesos y *m* tipos diferentes de recursos. Se definen los vectores y matrices siguientes:

$$\text{Recursos} = \begin{pmatrix} R1 \\ \\ Rm \end{pmatrix}$$

Cantidad total de cada recurso en el sistema

'El término predicción es un poco confuso. De hecho, se podría considerar a las estrategias tratadas en esta subsección como ejemplos de prevención del interbloqueo, puesto que, en realidad, previenen la aparición de un interbloqueo.

$$\text{Disponible} = \begin{pmatrix} AV_1 \\ \cdot \\ \cdot \\ \cdot \\ AV_m \end{pmatrix} \quad \text{Cantidad total de cada recurso sin asignar a los procesos}$$

$$\text{Demanda} = \begin{pmatrix} C_{11} \ldots C_{n1} \\ \cdot \\ \cdot \\ \cdot \\ C_{1m} \ldots C_{nm} \end{pmatrix} = (C_{1*} \cdots C_{n*}) \quad \text{Exigencias de recursos para cada proceso}$$

$$\text{Asignación} = \begin{pmatrix} A_{11} \ldots A_{n1} \\ \cdot \\ \cdot \\ \cdot \\ A_{1m} \ldots A_{nm} \end{pmatrix} = (A_{1*} \cdots A_{n*}) \quad \text{Asignación actual}$$

La matriz Demanda indica las exigencias máximas de recursos para cada proceso. Es decir, C_{ij} = demanda del recurso *j* por parte del proceso *i.* Esta información debe declararse por adelantado para que funcione la predicción de interbloqueo. De forma similar, A_{ij} = asignación actual del recurso *j* al proceso *i.* Se puede ver que se cumplen las siguientes relaciones:

1. $\sum_{k=1}^{n} A_{k*} + \text{Disponible} = \text{Recursos}$ — Todos los recursos están asignados o disponibles.
2. Para todo k, $C_{k*} \leq \text{Recursos}$ — Ningún proceso puede demandar más recursos que la cantidad total de recursos del sistema.
3. Para todo k, $A_{k*} \leq C_{k*}$ — Ningún proceso tiene asignados más recursos de cualquier tipo que los que ha declarado necesitar.

Con estas tres cantidades, se puede definir una política de predicción del interbloqueo que rechace iniciar un nuevo proceso si sus exigencias de recursos pueden conducir a un interbloqueo. Un nuevo proceso P^i comenzará sólo si:

$$\text{Recursos} \geq \sum_{k=1}^{n} C_{k*} + C_{(n+1)*}$$

Es decir, un proceso comenzará sólo si puede servirse la demanda máxima de todos los procesos actuales más la del nuevo proceso. Esta estrategia es poco óptima, puesto que asume el caso peor: que todos los procesos expresen su demanda máxima a la vez.

Negativa de Asignación de Recursos

La estrategia de negar la asignación de recursos, denominada **algoritmo del banquero,** fue propuesta por primera vez por Dijkstra [DIJK65].[3] Se comienza definiendo los conceptos de

[3] Dijkstra usó este nombre por la analogía de este problema con el de un banco cuando los clientes quieren obtener dinero prestado. Los clientes serian los procesos y el dinero a prestar, los recursos. Si se enuncia de esta manera, el banco tiene una reserva limitada de dinero para prestar y un conjunto de clientes con líneas de crédito. Un cliente puede elegir pedir dinero a cargo de la línea de crédito en un instante dado y no hay garantía de que el cliente realice ninguna reposición hasta después de sacar la cantidad máxima. El banquero puede rechazar un préstamo a un cliente si hay riesgo de que el banco no tenga fondos suficientes para hacer préstamos futuros que los clientes finalmente repondrán.

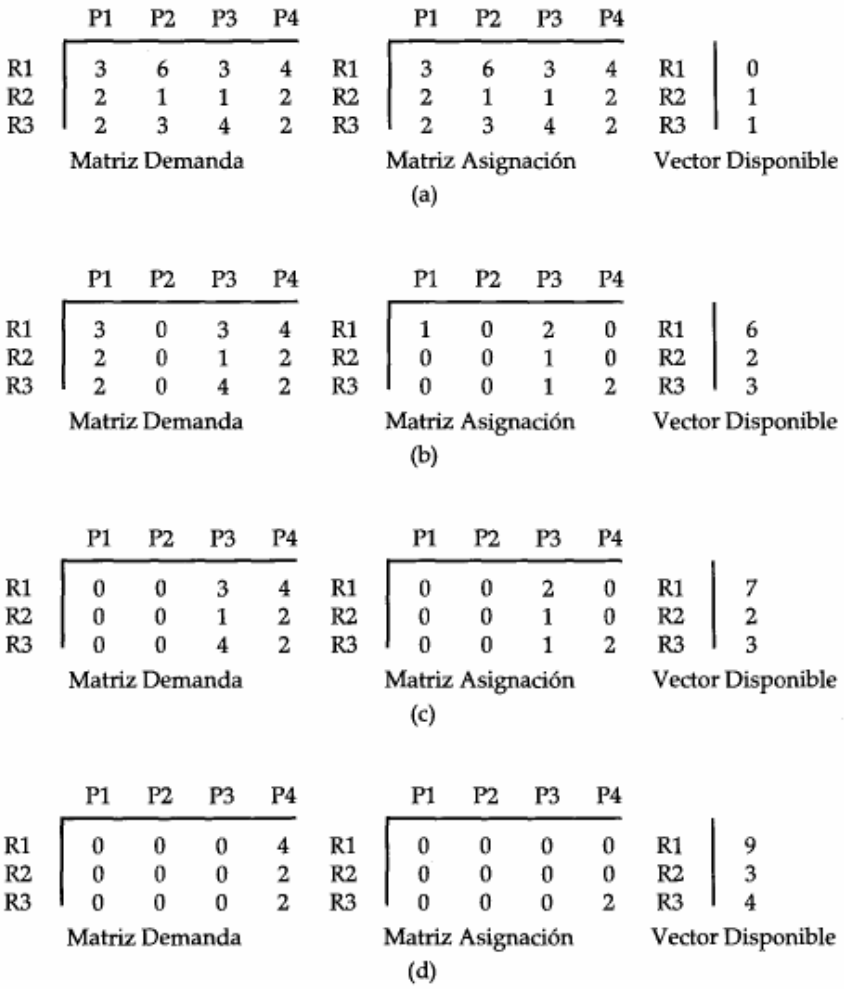


FIGURA 5.3 Determinación de un estado seguro

estado y de estado seguro. Considérese un sistema con un número fijo de procesos y un número fijo de recursos. En un instante dado, un proceso tendrá asignados cero o más recursos. El **estado** del sistema es, simplemente, la asignación actual de recursos a los procesos. Así pues, el estado estará formado por los dos vectores, Recursos y Disponible y las dos matrices, Demanda y Asignación, definidas anteriormente. Un **estado seguro** es un estado en el cual existe al menos un orden en el que todos los procesos pueden ejecutar hasta el final sin generar un interbloqueo. Un **estado inseguro** es, naturalmente, un estado que no es seguro.

El ejemplo siguiente ilustra estos conceptos. La figura 5.3a muestra el estado de un sistema que consta de cuatro procesos y tres recursos. La cantidad total de recursos Rl, R2 y R3 es 9, 3 y 6 unidades respectivamente. En el estado actual, se han asignado recursos a los cuatro procesos, quedando disponible 1 unidad del recurso 2 y 1 unidad del recurso 3. La pregunta es: ¿Es un estado seguro? Para resolver esta cuestión se plantea otra intermedia: ¿Pueden los cuatro

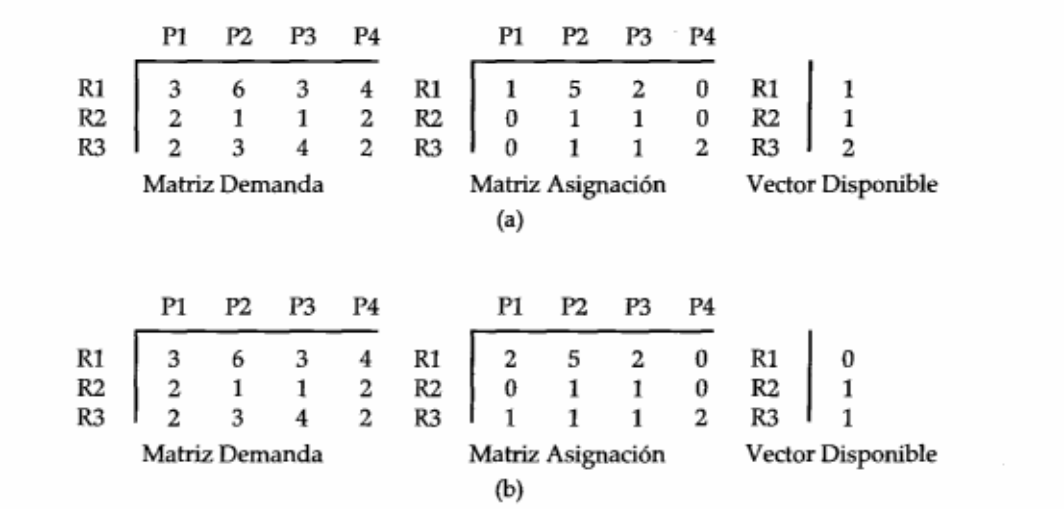


FIGURA 5.4 Determinación de un estado inseguro

procesos ejecutarse completamente con los recursos disponibles? Es decir, ¿Puede cubrirse la diferencia entre la asignación actual y la demanda máxima para cualquier proceso con los recursos disponibles? Sin duda, esto no es posible para Pl, que tiene sólo 1 unidad de Rl y necesita 2 unidades más de Rl, 2 unidades de R2 y 2 unidades de R3. Sin embargo, asignando una unidad de R3 al proceso P2, éste tendrá asignado el máximo de recursos necesarios y podrá ejecutarse hasta el final. Supóngase que esto sucede. Cuando P2 termina, sus recursos pueden volver a la reserva de recursos disponibles. El estado resultante se muestra en la figura 5.3b. Ahora se puede preguntar de nuevo si alguno de los procesos restantes puede terminar. En este caso, cada uno de los procesos restantes puede completarse. Supóngase que se elige Pl, se le asignan los recursos pedidos, finaliza Pl y todos sus recursos vuelven a la reserva disponible. Se continúa en el estado de la figura 5.3c. A continuación, se puede completar P3, dando como resultado el estado de la figura 5.3d. Por último, se puede completar P4. En este momento, todos los procesos han finalizado. Así pues, el estado definido por la figura 5.3a es un estado seguro.

Estas ideas sugieren una estrategia de predicción del interbloqueo, que consiste en asegurar que el sistema de procesos y recursos está siempre en un estado seguro. Para conseguir esto, se emplea la siguiente estrategia: Cuando un proceso realiza una solicitud de un conjunto de recursos, se supone que la solicitud se concede, se actualiza el estado del sistema y se determina si el resultado es un estado seguro. Si lo es, se concede la solicitud y, si no, se bloquea al proceso hasta que sea seguro conceder la solicitud.

Considérese el estado definido por la matriz de la figura 5.4a. Supóngase que P2 realiza una solicitud de 1 unidad de Rl y 1 unidad de R3. Si se supone que se concede la solicitud, el estado resultante es el de la figura 5.3a. Ya se ha visto que es un estado seguro; por tanto, resulta seguro satisfacer la solicitud. Sin embargo, volviendo al estado de la figura 5.4b, supóngase que Pl realiza una solicitud de una unidad de Rl y otra de R3. Ahora, suponiendo que se concede la solicitud, se tiene el estado de la figura 5.4b. La pregunta es: ¿Es un estado seguro? La respuesta es que no porque cada proceso necesitará al menos 1 unidad de Rl y no hay ninguna disponible. Así pues, con motivo de la predicción del interbloqueo, se debe denegar la solicitud de Pl y éste debe bloquearse.

Es importante señalar que la figura 5.4b no es un estado de interbloqueo. Simplemente tiene la posibilidad de un interbloqueo. Es posible, por ejemplo, que si Pl ejecutase desde ese estado, liberara posteriormente 1 unidad de Rl y 1 unidad de R3 antes de necesitar esos recursos de nuevo. Si esto sucediese, el sistema volvería a un estado seguro. Así pues, la estrategia de predicción del interbloqueo no predice el interbloqueo con certeza, sino que sólo anticipa la posibilidad de interbloqueo y asegura que nunca exista tal posibilidad.

La figura 5.5, basada en una versión de Krakowiak y Beeson, da una versión abstracta de la lógica de predicción del interbloqueo [KRAK88]. El algoritmo principal se muestra en la parte b de la figura. Con el estado del sistema definido por la estructura de datos *estado, solicitud[*]* es un vector que define los recursos pedidos por el proceso *i.* En primer lugar, se hace una comprobación para asegurar que la solicitud no excede la demanda inicial del proceso. Si la solicitud es válida, el paso siguiente es determinar si es posible satisfacer la solicitud, esto es, si hay suficientes recursos disponibles. Si no es posible, el proceso se suspende. Si es posible, el paso final es determinar si es seguro satisfacer la solicitud. Para hacer esto, se prueba a asignar los recursos al proceso *i* desde *nuevo _estado.* Después, se realiza un test de seguridad usando el algoritmo de la figura 5.5c.

La predicción del interbloqueo tiene la ventaja de que no es necesario expulsar y hacer retroceder procesos como en la detección del interbloqueo y es menos restrictiva que la prevención. Sin embargo, su uso supone una serie de desventajas y restricciones como las siguientes:

• Se debe presentar la máxima demanda de recursos por anticipado.

• Los procesos a considerar deben ser independientes; esto es, el orden en que ejecuten no debe estar forzado por condiciones de sincronización,

• Debe haber un número fijo de recursos a repartir y un número fijo de procesos.

Una Estrategia Integrada de Interbloqueo

Como se propone en la tabla 5.1, hay puntos fuertes y debilidades en todas las estrategias de solución del interbloqueo. En lugar de intentar diseñar un servicio del sistema operativo que emplee sólo una de las estrategias, puede ser más eficiente usar diferentes estrategias en diferentes situaciones. Silberschatz y Galvin sugieren este enfoque [SILB94]:

• Agrupar los recursos en un número de clases diferentes.

• Usar la estrategia de ordenación lineal definida anteriormente para la prevención de círculos viciosos de espera e impedir el interbloqueo entre clases de recursos.

• Dentro de cada clase de recursos, emplear el algoritmo más apropiado para dicha clase. Como ejemplo de esta técnica, considérense las siguientes clases de recursos:

• *Espacio intercambiable:* Bloques de memoria en almacenamiento secundario para el uso en el intercambio de procesos.

• *Recursos de procesos:* Dispositivos asignables, como unidades de cinta y archivos.

• *Memoria principal:* Asignable a los procesos en páginas o segmentos.

• *Recursos internos:* Como canales de E/S.

El orden en que se enumeran estas clases de recursos es el orden en el que se asignan. El orden es razonable, teniendo en cuenta la secuencia de pasos que un proceso debe seguir durante su vida. En cada clase, se pueden usar las siguientes estrategias:

```
type estado = record
                  recurso, disponible: array [0 ... m – 1] of entero;
                  demanda, asignación: array [0 ... n – 1, 0 ... m – 1] of entero;
              end
```

(a) Estructuras de datos globales

```
if asignación[i, *] + solicitud[*] > demanda[i, *] then
 <error>                  – solicitud total > demanda
else
 if solicitud[*] > disponible[*] then
     <suspender proceso>
 else                     – simular asignación
     <definir nuevo_estado como:
     asignación[i, *] := asignación [i, *] + solicitud[*]
     disponible[*] := disponible[*] – solicitud[*]
 end;
 if seguro(nuevo_estado) then
     <realizar asignación>
 else
     <restaurar estado original>;
     <suspender proceso>
 end
end.
```

(b) Algoritmo de asignación de recursos

```
function seguro (estado: S): boolean;
var  disponible_actual: array [0 ... m – 1] of entero;
     resto: set of procesos;
begin
disponible_actual := disponible;
 resto := {todos los procesos};
 posible := true;
 while posible do
     encontrar un P_k en resto tal que
         demanda[k, *] – asignación[k, *] < disponible_actual;
     if encontrado then          – simular ejecución de P
         disponible_actual := disponible_actual + asignación[k, *];
         resto := resto –{P_k}
     else
         posible := falso
     end
 end;
 seguro := (resto = null)
end.
```

(c) Algoritmo de comprobación de estado seguro (algoritmo del banquero)

FIGURA 5.5 Lógica de predicción del interbloqueo

• *Espacio intercambiable:* Puede aplicarse prevención de interbloqueos, pidiendo que todos los recursos sean asignados de una vez, como en la estrategia de prevención de retención y espera. Esta estrategia es razonable si se conocen los requisitos máximos de almacenamiento, lo que suele ser habitual. Otra posibilidad es la predicción de interbloqueos.
• *Recursos de procesos:* La predicción es a menudo efectiva en esta categoría, puesto que es razonable esperar que los procesos declaren por anticipado los recursos de esta clase que necesitarán. También es posible en esta clase la prevención mediante la ordenación de recursos.
• *Memoria principal:* La prevención con apropiación parece ser la estrategia más adecuada para la memoria principal. Cuando se expulsa a un proceso, simplemente es trasladado a la memoria secundaria, liberando espacio para resolver el interbloqueo.
• *Recursos internos:* Puede usarse la prevención por ordenación de recursos. 5.5

EL PROBLEMA DE LA CENA DE LOS FILÓSOFOS

Érase una vez cinco filósofos que vivían juntos. La vida de cada filósofo consistía principalmente en pensar y comer y, tras años de pensar, todos los filósofos se habían puesto de acuerdo en que la única comida que contribuía a sus esfuerzos pensadores eran los espaguetis.

Los preparativos de la comida eran simples (figura 5.6): una mesa redonda en la que había una gran fuente de espaguetis, cinco platos, uno para cada filósofo y cinco tenedores. Un filósofo que quiera comer irá a su lugar asignado en la mesa y, usando los dos tenedores de cada lado del plato, cogerá los espaguetis y se los comerá. El problema es el siguiente: Inventar un ritual (algoritmo) que permita comer a los filósofos. El algoritmo debe satisfacer la exclusión mutua (dos filósofos no pueden emplear el mismo tenedor a la vez), además de evitar el interbloqueo y la inanición (en este caso, este último término tiene significado literal además de algorítmico).

Este problema, propuesto por Dijkstra, puede no parecer importante o relevante por sí mismo. Sin embargo, sirve para ilustrar los problemas básicos del interbloqueo y la inanición. Es más, intentar desarrollar una solución revela muchas de las dificultades de la programación concurrente (véase, por ejemplo, [GING90]). Por consiguiente, este problema es un caso de prueba estándar para examinar soluciones a la sincronización.

La figura 5.7 sugiere una solución por medio de semáforos. Cada filósofo toma primero el tenedor de su izquierda y, después, el de su derecha. Cuando un filósofo termina de comer, devuelve los dos tenedores a la mesa. Esta solución, desafortunadamente, produce interbloqueo: Si todos los filósofos están hambrientos al mismo tiempo, todos se sientan, todos cogen el tenedor de su izquierda y todos intentan tomar el otro tenedor, que no estará. En esta situación informal, todos los filósofos pasan hambre.

Para superar el riesgo de interbloqueo, se podrían adquirir cinco tenedores adicionales (una solución más saludable) o enseñar a los filósofos a comer espaguetis con sólo un tenedor. Como otra solución posible, se podría considerar incorporar un sirviente que permita pasar sólo a cuatro filósofos a la vez al comedor. Con un máximo de cuatro filósofos sentados, al menos uno de los filósofos tendrá acceso a los dos tenedores. La figura 5.8 muestra esta solución, de nuevo con semáforos. Esta solución está libre de interbloqueo e inanición.

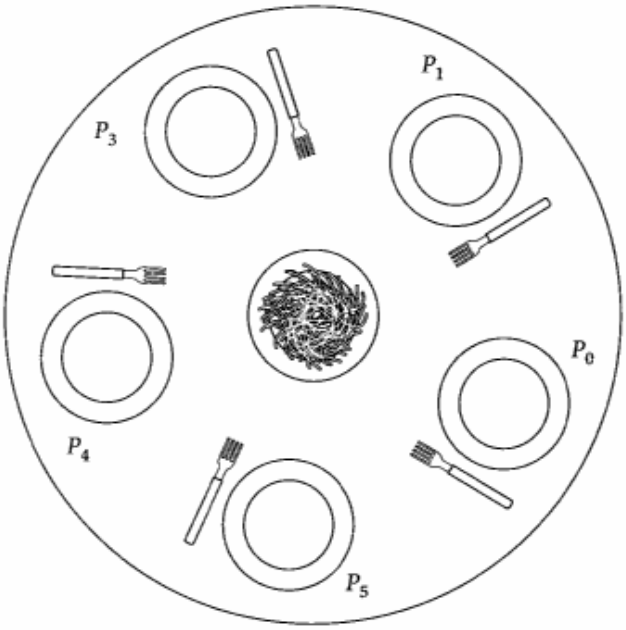


FIGURA 5.6 Preparativos de cena para filósofos

```
program cena_filósofos;
var tenedor: array [0 ... 4] of semáforo (:= 1);
    i: entero;
procedure filósofo (i: entero);
begin
 repeat
    pensar;
    wait (tenedor[i]);
    wait (tenedor[(i+1) mod 5]);
    comer;
    signal (tenedor[(i+1) mod 5]);
    signal (tenedor[i])
 forever
end;
begin
 parbegin
    filósofo (0);
    filósofo (1);
    filósofo (2);
    filósofo (3);
    filósofo (4);
 parend
end.
```

FIGURA 5.7 Una primera solución al problema de la cena de los filósofos

```
 program cena_filósofos;
var tenedor: array [O ... 4] of semáforo (:= 1);
      habitación: semáforo (:= 4);
      i: entero;
procedure filósofo (i: entero);
  begin repeat
      pensar;
      wait (habitación);
      wait (tenedor[i]);
      wait (tenedor[(i+l) mod 5]);
      comer;
      signal (tenedor[(i+l) mod 5]);
      signal (tenedor[i]);
signal (habitación) forever
end;
begin parbegin
      filósofo (0);
      filósofo (1);
      filósofo (2);
      filósofo (3);
     filósofo (4);
```

parend end.

FIGURA 5.8 Segunda solución al problema de la cena de los filósofos

5.6 SISTEMAS DE EJEMPLO

UNIX Sistema V

UNIX ofrece diversos mecanismos para la comunicación entre procesos y la sincronización. Aquí se mostrarán los más importantes:

- Tubos *(pipes)*
- Mensajes
- Memoria compartida
- Semáforos
- Señales

Los tubos, los mensajes y la memoria compartida proporcionan un medio de comunicación de datos entre procesos, mientras que los semáforos y las señales se usan para provocar acciones en otros procesos.

Tubos

Una de las contribuciones más significativas del UNIX al desarrollo de los sistemas operativos son los tubos. Inspirados en el concepto de corutinas [RITC84], un tubo es un buffer circular que permite a dos procesos comunicarse según el modelo productor/consumidor. Así pues, consiste en una cola primero en llegar/primero en salir en la que un proceso escribe y el otro lee.

Cuando se crea un tubo, se le da un tamaño fijo en bytes. Cuando un proceso intenta escribir en el tubo, la solicitud de escritura se ejecuta inmediatamente si hay suficiente espacio;

de otro modo, el proceso se bloquea. De forma similar, un proceso lector se bloquea si intenta leer más bytes de los que tiene el tubo en ese momento. El sistema operativo se encarga de la exclusión mutua, esto es, al tubo sólo puede acceder un proceso cada vez.

Hay dos tipos de tubos: con nombre y sin nombre. Sólo procesos afines pueden compartir tubos sin nombre, mientras que los procesos no afines sólo pueden compartir tubos con nombre.

Mensajes

Un mensaje es un bloque de texto con un tipo asociado. UNIX proporciona las llamadas al sistema *msgsnd* y *msgrcv* para que los procesos puedan dedicarse al paso de mensajes. Cada proceso tiene asociada una cola de mensajes, que funciona como un buzón de correos.

El emisor del mensaje especifica el tipo de mensaje en cada envío y el receptor puede usarlo como criterio de selección. El receptor puede recuperar los mensajes tanto en orden FIFO como por el tipo. Un proceso se suspenderá cuando intente leer de una cola vacía. Si un proceso intenta leer un mensaje de cierto tipo y falla, el proceso no se suspenderá.

Memoria Compartida

La forma más rápida de comunicación entre procesos que proporciona UNIX es la memoria compartida, que se trata de un bloque común de memoria virtual compartido por varios procesos. Los procesos leen y escriben en la memoria compartida usando las mismas instrucciones de la máquina que emplean para leer y escribir en otras partes de su espacio de direcciones virtual. Los permisos de un proceso son sólo lectura o lectura-escritura, según el proceso que sea. Las restricciones de exclusión mutua no forman parte del servicio de memoria compartida, sino que las debe suministrar el proceso que hace uso de la memoria compartida.

Semáforos

Las llamadas al sistema para semáforos en el UNIX Sistema V son una generalización de las primitivas *wait* y *signal* definidas en este capítulo, en las que se pueden realizar simultáneamente varias operaciones y los incrementos y decrementos pueden ser de valores mayores que 1. El núcleo ejecuta de forma atómica todas las operaciones solicitadas; ningún otro proceso puede acceder al semáforo hasta que todas las operaciones hayan terminado.

Un semáforo consta de los siguientes elementos:

- Valor actual del semáforo
- ID del último proceso que operó con el semáforo
- Número de procesos esperando a que el valor del semáforo sea mayor que su valor actual

• Número de procesos esperando a que el valor del semáforo sea cero

Asociadas con cada semáforo existen colas de procesos suspendidos.

Los semáforos, en realidad, se crean por conjuntos, donde un conjunto de semáforos consta de uno o más semáforos. Hay una llamada al sistema *semcti* que permite dar valores a todos los semáforos de un conjunto al mismo tiempo. Además, hay una llamada al sistema *semop* que toma como argumento una lista de operaciones sobre semáforos, cada una de ellas definida sobre uno de los semáforos de un conjunto. Cuando se genera esta llamada, el núcleo realiza las operaciones indicadas una a una. Para cada operación, se especifica la función real por medio del valor *sem_op.* Existen las siguientes posibilidades:

• Si *sem_op* es positivo, el núcleo incrementa el valor del semáforo y despierta a todos los procesos que esperaban a que el valor del semáforo se incrementase.

• Si *sem_op* es O, el núcleo comprueba el valor del semáforo. Si es O, continúa con las otras operaciones de la lista; en otro caso, incrementa el número de procesos esperando a que este semáforo sea O y suspende el proceso hasta que el valor del semáforo igual a 0.

• Si *sem_op* es negativo y su valor absoluto es menor o igual que el valor del semáforo, el núcleo suma *sem_op* (un número negativo) al valor del semáforo. Si el resultado es O, el núcleo despierta a todos los procesos que esperan a que el valor del semáforo sea igual a 0.

• Si *sem_op* es negativo y su valor absoluto es mayor que el valor del semáforo, el núcleo suspende al proceso, caso de que el valor del semáforo se incremente.

Esta generalización de los semáforos ofrece una flexibilidad considerable para llevar a cabo la sincronización y coordinación de procesos.

Señales

Una señal es un mecanismo de software que informa a un proceso del acontecimiento de un suceso asíncrono. Una señal es similar a una interrupción de hardware, pero no emplea prioridades. Es decir, todas las señales se tratan por igual; las señales que se producen en un mismo instante se presentan al proceso en el mismo instante, sin ninguna ordenación en particular.

Los procesos pueden enviarse señales unos a otros y el núcleo puede enviar señales internas. Una señal es emitida para actualizar un campo de la tabla de procesos del proceso al que se le envía. Puesto que cada señal se mantiene como un único bit, las señales de un tipo determinado no pueden ponerse en cola. Una señal se procesa justo después de que el proceso despierte para ejecutar o cada vez que el proceso esté dispuesto a volver de una llamada al sistema. Un proceso puede responder a una señal ejecutando alguna acción por omisión (por ejemplo, terminar), ejecutando una función de manejo de la señal o ignorando la señal.

La tabla 5.2 enumera las señales definidas en el UNIX Sistema V.

Windows NT

Windows NT ofrece sincronización entre los hilos como parte de la arquitectura de objetos. El mecanismo usado por el ejecutor de NT para implementar los servicios de sincronización es la familia de objetos de sincronización, que está formada por los siguientes:

• Proceso

• Hilo

• Archivo

- Suceso
- Pareja de sucesos
- Semáforo
- Temporizador
- Mulante

Los tres primeros tipos de objeto de la lista anterior tienen otros usos, pero también se pueden emplear para sincronización. El resto de los tipos de objeto están diseñados específicamente para respaldar la sincronización.

Cada caso de objeto de sincronización puede estar en estado señalizado o no señalizado. Un hilo puede estar suspendido por un objeto en estado no señalizado; el hilo es liberado cuando el objeto pasa a estado señalizado. El mecanismo es sencillo: un hilo genera una solicitud de espera al ejecutor de NT por medio del descriptor *(handie)* del objeto de sincronización. Cuando un objeto pasa a estado señalizado, el ejecutor de NT libera todos los objetos hilo que estaban esperando por dicho objeto de sincronización.

TABLA 5.2 Señales del UNIX

Valor	*Nombre*	*Descripción*
01	SIGHUP	colgar; se le envía a un proceso cuando el núcleo supone que el usuario de ese proceso no está realizando un trabajo útil
02	SIGINT	interrumpir
03	SIGQUIT	salir; la envía el usuario para provocar una parada del proceso y la generación de un volcado de memoria
04	SIGILL	instrucción ilegal
05	SIGTRAP	seguimiento de cepo *(trap)*; provoca la ejecución de un código de seguimiento del proceso
06	SIGIOT	instrucción IOT
07	SIGEMT	instrucción EMT
08	SIGFPT	excepción de coma flotante
09	SIGKILL	matar; terminación del proceso
10	SIGBUS	error de bus
11	SIGSEGV	violación de segmento; un proceso intenta acceder a una posición fuera de su espacio de direcciones virtual
12	SIGSYS	argumentos incorrectos en una llamada al sistema
13	SIGPIPE	escritura en un cauce que no tiene lectores asociados
14	SIGALARM	alarma de reloj; se emite cuando un proceso desea recibir una señal después de un periodo de tiempo
15	SIGTERM	terminación por software
16	SIGUSR1	señal 1 definida por el usuario
17	SIGUSR2	señal 2 definida por el usuario
18	SIGCLD	muerte de un hijo
19	SIGPWR	corte de energía

La tabla 5.3 resume los sucesos que hacen que cada tipo de objetos pase a estado señalizado y el efecto que tiene en los hilos que esperan. La pareja de sucesos es un objeto aso-

ciado con una interacción oliente-servidor y está accesible sólo para el hilo cliente y el hilo servidor. Tanto el cliente como el servidor pueden llevar al objeto al estado señalizado, haciendo que el otro hilo quede también señalizado. En todos los objetos de sincronización pueden esperar varios hilos.

El objeto mutante se utiliza para hacer cumplir la exclusión mutua en el acceso a un recurso, permitiendo que sólo un objeto hilo obtenga el acceso en cada instante. Por tanto, funciona como un semáforo binario. Cuando el objeto mutante pasa a estado señalizado, sólo se libera uno de los hilos que esperan. Para todos los demás objetos de sincronización que soporten espera de varios hilos, todos ellos se liberan cuando el objeto pasa a estado sincronizado.

MVS

MVS proporciona dos servicios para hacer cumplir la exclusión mutua en el uso de recursos:

las colas y los cierres. Las colas tienen que ver con los recursos controlados por el usuario, como los archivos, mientras que los cierres están relacionados con los recursos del sistema MVS.

ENQUE (colas)

El servicio ENQUE se usa para regular el acceso a recursos de datos compartidos. Los recursos solicitados pueden constituir todo un volumen de disco, uno o más archivos, registros dentro de un archivo, bloques de control de programa o cualquier área de trabajo dentro de la memoria principal. Un proceso (un espacio de direcciones) solicita control compartido si el recurso es de sólo lectura y control exclusivo si los datos pueden modificarse. La tabla 5.4 muestra las reglas de MVS para determinar la acción a seguir cuando se presenta una solicitud ENQUE. Básicamente, el esquema se adapta al modelo de los lectores/escritores con prioridad para los escritores.

Además de solicitudes definitivas de acceso, el usuario puede pedir que se compruebe la disponibilidad del recurso pero sin solicitar su control o bien puede pedir que se asigne el control del recurso al solicitante sólo si el recurso está libre en ese momento. Estas opciones pueden emplearse para prevenir el interbloqueo o para notificar al operador que hay un problema.

Cierres

Los cierres sirven para hacer valer la exclusión mutua en el acceso a los recursos del sistema, a través de rutinas de MVS. Un cierre es, simplemente, un área de la parte común del almacenamiento virtual y, normalmente, se mantiene en memoria principal permanentemente. Tiene varios bits para indicar si el cierre está en uso y, si procede, quién es el propietario del cierre. Los cierres se clasifican en dos clases y dos tipos. Las clases son:

• *Globales:* Definidos en todo el espacio de direcciones

• *Locales:* Definidos en todas las tareas de un solo espacio de direcciones Los tipos son:

• *Cierre de giro (spin lock):* El procesador ejecuta algún tipo de instrucción "comprobar y fijar" *(test-and-set)* sobre el cierre, con espera activa.

• *Cierres de suspensión:* La tarea en espera queda suspendida.

TABLA 5.3 Objetos de Sincronización de Windows NT

Tipo de Objeto	*Definición*	*Pasa a Estado Señalado Cuando*	*Efecto sobre los Hilos que Esperan*
Proceso	Invocación a un programa, incluyendo el espacio de direcciones y los recursos requeridos para ejecutar el programa	Termina el último hilo	Todos se liberan
Hilo	Entidad ejecutable dentro de un proceso	Termina el hilo	Todos se liberan
Archivo	Caso particular de un archivo abierto o de un dispositivo de E/S	Se completa la operación de E/S	Todos se liberan
Suceso	Anuncio de que ha ocurrido un suceso del sistema	Un hilo activa el suceso	Todos se liberan
Pareja de Sucesos	Notificación de que un hilo cliente dedicado ha copiado un mensaje del servidor Win32 o viceversa	El hilo cliente dedicado o el servidor activan el suceso	Se liberan otros hilos dedicados
Semáforo	Contador que regula el número de hilos que pueden emplear un proceso	El contador del semáforo llega a cero	Todos se liberan
Temporizador	Contador que registra el paso del tiempo	Llega el tiempo de activación o expira el intervalo de tiempo	Todos se liberan
Mutante	Mecanismo que proporciona exclusión mutua a los entornos Win32 y OS/2	El hilo propietario u otro hilo libera al mutante	Se libera un hilo

TABLA 5.4 Tabla de Decisión para el servicio ENQUE de MVS

	Estado del Recurso en el Momento de la Solicitud		
Tipo de Recurso	Libre	Compartido	Exclusivo
Compartido	Aceptada	Aceptada a menos que en la cola haya una solicitud exclusiva	En cola
Exclusivo	Aceptada	En cola	En cola

Los cierres de giro se emplean en secciones críticas que se ejecutan durante poco tiempo. Los cierres de suspensión dan a entender secciones críticas largas y la rentabilidad de suspender un proceso rechazado mientras se despacha a otro proceso. Los cierres locales son siempre de suspensión, mientras que los globales pueden ser tanto de suspensión como de giro.

Para prevenir interbloqueos, los cierres se disponen en una jerarquía; ésta es la estrategia discutida antes para prevenir la condición de espera circular. La tabla 5.5 muestra la jerarquía de cierres de MVS. Un procesador puede solicitar solamente cierres superiores a los que posee, según la jerarquía.

En [KATZ84] se puede encontrar un estudio detallado, aunque algo pasado de moda, de los cierres de MVS.

5.7 RESUMEN

El interbloqueo es el bloqueo de un conjunto de procesos que compiten por los recursos del sistema o bien se comunican unos con otros. El bloqueo es permanente hasta que el sistema operativo realice alguna operación extraordinaria, como puede ser matar uno o más procesos u obligar a uno o más procesos a retroceder en la ejecución. El interbloqueo puede involucrar a recursos reutilizables o consumibles. Un recurso consumible es aquél que se destruye al ser adquirido por un proceso; como ejemplos se incluyen los mensajes y la información de los buffers de E/S. Un recurso reutilizable es aquél que no se agota o se destruye por el uso, como un canal de E/S o una zona de memoria.

Los métodos generales para hacer frente al interbloqueo son tres: prevención, detección y predicción. La prevención del interbloqueo garantiza que no se producirá el interbloqueo asegurando que no se cumpla alguna de las condiciones necesarias para el interbloqueo. La detección del interbloqueo es necesaria si el sistema operativo está siempre dispuesto a conceder las peticiones de recursos; periódicamente, el sistema operativo debe comprobar la situación de interbloqueo y tomar medidas para deshacerlo. La predicción del interbloqueo supone el análisis de cada nueva petición de recursos para determinar si ésta puede conducir a un interbloqueo y concederlas sólo si no es posible llegar a un interbloqueo.

TABLA 5.5 Bloques de MVS

Nombre del bloqueo	*Categoría*	*Tipo*	*Descripción (ver nota 1)*
RSMGL	Global	de giro (*spin*)	Bloqueo global de gestión del almacenamiento real — publica los recursos globales RSM.
VSMFIX	Global	de giro (spin)	Bloqueo fijo de reservas para gestión del almacenamiento virtual — publica las colas globales VSM.
ASM	Global	de giro (spin)	Bloqueo de gestión del almacenamiento auxiliar — publica los recursos ASM a nivel del espacio de direcciones.
ASMGL	Global	de giro (spin)	Bloqueo global de gestión del almacenamiento auxiliar — publica los recursos ASM a nivel global.
RSMST	Global	de giro (spin)	Bloqueo de robo de gestión del almacenamiento real — publica los bloques de control RSM a nivel del espacio de direcciones cuando no se conoce qué cierres del espacio de direcciones se retienen actualmente.
RSMCM	Global	de giro (spin)	Bloqueo común de gestión del almacenamiento real — publica los recursos RSM de área común (como las entradas de una tabla de páginas)
RSMXM	Global	de giro (spin)	Bloqueo cruzado de memoria de gestión del almacenamiento real — publica los bloques de control RSM a nivel del espacio de direcciones cuando se necesita publicar un segundo espacio de direcciones.
RSMAD	Global	de giro (spin)	Bloqueo de espacio de direcciones de gestión del almacenamiento real — publica los bloques de control RSM a nivel del espacio de direcciones.
RSM	Global	de giro (spin)	Bloqueo (compartido/exclusivo) de gestión del almacenamiento real — publica las funciones y recursos RSM a nivel global.
VSMPAG	Global	de giro (spin)	Bloqueo de reservas paginables de gestión del almacenamiento virtual — publica el área de trabajo VSM para las reservas paginables VSM.
DISP	Global	de giro (spin)	Bloqueo de despachador global — publica el ASVT y la cola de despacho ASCB.
SALLOC	Global	de giro (spin)	Bloqueo de asignación de espacio — publica las rutinas receptoras que preparan al procesador para señales de emergencia o alertas de mal funcionamiento.
IOSYNCH	Global	de giro (spin)	Bloqueo de sincronización del supervisor de E/S — publica, por medio de una tabla de palabras de cierre, los recursos IOS.
IOSUCB	Global	de giro (spin)	Bloqueo de bloque de control de unidad del supervisor de E/S —publica los accesos y actualizaciones a las UCB. Hay un cierre IOSUCB por cada UCB.
SRM	Global	de giro (spin)	Bloqueo de gestión de los recursos del sistema — publica los bloques de control RSM y los datos asociados.
TRACE	Global	de giro (spin)	Bloqueo (compartido/exclusivo) de seguimiento — publica el buffer de trazado del sistema.
CPU	Global	de giro (spin)	Bloqueo de procesador — ofrece una inhabilitación (legal) reconocida por el sistema (ver nota 2).
CMSSMF	Global	de suspensión	Bloqueo cruzado de servicios de memoria para las funciones de gestión del sistema — publica las funciones SMF y los bloques de control (ver nota 3).

TABLA 5.5 (Continuación)

Nombre del Cierre	*Categoría*	*Tipo*	*Descripción (ver nota 1)*
CMSEQDQ	Global	de suspensión	Bloqueo cruzado de servicios de memoria ENQ/DEQ — publica las funciones ENQ/DEQ y los bloques de control (ver nota 3).
CMS	Global	de suspensión	Bloqueo general cruzado de servicios de memoria — publica en más de un espacio de direcciones, donde no se ofrece la publicación por parte de otro(s) cierre(s) global(es). El cierre CMS proporciona una publicación global cuando se necesita capacitar (ver nota 3).
CML	Local	de suspensión	Bloqueo de almacenamiento local — publica las funciones y el almacenamiento dentro de un espacio de direcciones distinto del espacio de direcciones propio (ver nota 4).
LOCAL	Local	de suspensión	Bloqueo de almacenamiento local — publica las funciones y el almacenamiento dentro de un espacio de direcciones local. Hay un cierre LOCAL por cada espacio de direcciones (ver nota 4).

Notas:

1. Todos los cierres se enumeran en orden jerárquico, siendo RSMGL el cierre superior de la jerarquía (véanse también notas 2, 3 y 4).
2. El cierre de CPU no guarda relación jerárquica con el resto de los cierres de giro. Sin embargo, una vez conseguido, no se pueden adquirir cierres de suspensión.
3. Los cierres cruzados de servicios de memoria (CMSSMF, CMSEQDQ y CMS) son todos de la misma jerarquía.
4. Los cierres CML y LOCAL son todos de la misma jerarquía.

5.8 LECTURAS RECOMENDADAS

El clásico artículo de interbloqueo, [HOLT72], es todavía digno de leer, como también lo es [COFF71]. Otro buen estudio es [ISL080].

COFF71 COFFMAN, E., ELPHICK, M., y SHOSHANI, A. "System Deadlocks". *Computing Surveys,* junio de 1971.

HOLT72 HOLT, R. "Some Deadlock Properties of Computer Systems." *Computing Surveys,* septiembre de 1972.

ISL080 ISLOOR, S., y MARSLAND, T. "The Deadlock Problem: An Overview". *Computer Surveys,* septiembre de 1980.

5.9

PROBLEMAS

5.1 Considérese la siguiente instantánea de un sistema. No hay solicitudes pendientes en la cola.

Disponible:

r1	r2	r3	r4
2	1	0	0

	asignación actual				demanda máxima				necesidad restante			
proceso	r1	r2	r3	r4	r1	r2	r3	r4	r1	r2	r3	r4
p1	0	0	1	2	0	0	1	2				
p2	2	0	0	0	2	7	5	0				
p3	0	0	3	4	6	6	5	6				
p4	2	3	5	4	4	3	5	6				
p5	0	3	3	2	0	6	5	2				

a) Calcular qué podría solicitar aún cada uno de los procesos y rellenar la columna "demanda restante".
b) ¿Está el sistema actualmente en un estado seguro o inseguro? ¿Por qué?
c) ¿Está el sistema actualmente bloqueado? ¿Por qué o por qué no?
d) ¿Qué procesos, si los hay, están o pueden llegar a estar interbloqueados?
e) Si llega de p3 una solicitud de (0,1,0,0), ¿podrá concederse inmediatamente esta solicitud con seguridad? ¿En qué estado (interbloqueado, seguro o inseguro) dejaría al sistema la concesión inmediata de la solicitud completa? ¿Qué procesos, si los hay, están o pueden llegar a estar interbloqueados si se concede inmediatamente la solicitud completa? 5.2 Evaluar si el algoritmo del banquero es de utilidad en la vida real

5.3 a) Tres procesos comparten 4 unidades de un recurso que se pueden reservar y liberar sólo una cada vez. Cada proceso necesita un máximo de 2 unidades. Demostrar que no puede haber interbloqueo.

b) *N* procesos comparten *M* unidades de un recurso que se pueden reservar y liberar sólo una cada vez. La demanda máxima de cada proceso no excede de M y la suma de todas las demandas máximas es menor que *M + N*. Demostrar que no puede haber interbloqueo.

5.4 Este ejercicio demuestra la sutileza del problema de la cena de los filósofos y la dificultad de escribir programas correctos usando semáforos. La solución siguiente al problema de la cena de los filósofos, escrita en C, se encuentra en el texto de Tanenbaum [TANE87]:

```
#define N                    5                    /* número de filósofos */
#define IZQUIERDA            (i -1) MOD N         /* número vecino izquierdo de i */
#define DERECHA              (i + 1) MOD N        /* número vecino derecho de i */
#define MEDITANDO            0                    /* el filósofo está meditando */
#define HAMBRIENTO           1                    /* el filósofo intenta tomar los tenedores */
#define COMIENDO             2                    /* el filósofo está comiendo */
typedef int semáforo;                             /* los semáforos son una clase de enteros */
int estado [N];                                   /* vector con el estado de cada uno */
semáforo mutex = 1;                               /* exclusión mutua en regiones críticas */
```

```
semáforo s[N];                                  /* un semáforo por filósofo */
filósofo (i)
int i;                                          /* número del filósofo, 0 a N–1  */
{
   while (TRUE) {                               /* bucle infinito */
         meditar ( );                           /* el filósofo está meditando */
         tomar_tenedores (i);                   /*toma dos tenedores o se bloquea */
         comer ( );                             /* ñam-ñam, espaguetis */
         dejar_tenedores (i);                   /* deja ambos tenedores en la mesa */
   }
}

tomar_tenedores (i)
int i;                                          /* número del filósofo, 0 a N–1 */
{
   wait (mutex);                                /* entra en región crítica */
   estado[i] = HAMBRIENTO;                      /* recuerda que filósofo i está hambriento */
   comprobar (i);                               /* intenta tomar 2 tenedores */
   signal (mutex);                              /* sale de región crítica */
   wait (s[i]);                                 /* se bloquea si no se tienen los tenedores */
}

dejar_tenedores (i)
int i;                                          /* número del filósofo, 0 a N–1 */
{
   wait (mutex);                                /* entra en región crítica */
   estado[i] = MEDITANDO;                       /* el filósofo i ha terminado de comer */
   comprobar (IZQUIERDA);                       /* ¿puede el de la izquierda comer ahora? */
   comprobar (DERECHA);                         /* ¿puede el de la derecha comer ahora? */
   signal (mutex);                              /* sale de región crítica */
}

comprobar (i)
int i;                                          /* número del filósofo, 0 a N–1 */
{
   if (estado[i] == HAMBRIENTO && estado[IZQUIERDA] != COMIENDO &&
       estado[DERECHA] != COMIENDO) {estado[i] = COMIENDO;
       signal (s[i]);
   }
}
```

a) Describir en pocas palabras el funcionamiento de esta solución.

b) Tanenbaum afirma que: "La solución [...] es correcta y permite el máximo paralelismo para un número arbitrario de filósofos". Aunque esta solución previene el interbloqueo, no es correcta porque la inanición es posible. Demuéstrese con un contraejemplo. Indicación: Considérese el caso de cinco filósofos. Supóngase que son filósofos glotones: No pasan apenas tiempo pensando. Tan pronto como un filósofo ha terminado un tumo de comida, le entra el hambre casi de inmediato. Considérese entonces una configuración en la que dos filósofos están comiendo y los otros tres bloqueados y hambrientos.

5.5 Supóngase que hay dos clases de filósofos. Los de una clase siempre cogen primero el tenedor izquierdo ("zurdos") y los de la otra clase siempre cogen primero el tenedor derecho ("diestros"). El comportamiento de un zurdo está definido en la figura 5.7. El comportamiento de un diestro es el siguiente:

```
begin repeat
pensar;
wait (tenedor[(i+l) mod 5]);
wait (tenedor[i])¡
comer;
signal (tenedor[i]);
    signal (tenedor[(i+l) mod 5]);
  forever end;
```

Probar las .siguientes afirmaciones:

a) Cualquier distribución de zurdos y diestros que tenga al menos uno de cada clase evita el interbloqueo.

b) Cualquier distribución de zurdos y diestros que tenga al menos uno de cada clase previniendo la inanición.

CAPITULO 6

Gestión de memoria

En un sistema monoprogramado, la memoria principal se divide en dos partes: una parte para el sistema operativo (monitor residente, núcleo) y otra parte para el programa que se ejecuta en ese instante. En un sistema multiprogramado, la parte de "usuario" de la memoria debe subdividirse aún más para hacer sitio a varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el sistema operativo y se conoce como **gestión de memoria.**

En un sistema multiprogramado resulta vital una gestión efectiva de la memoria. Si sólo hay unos pocos procesos en memoria, entonces la mayor parte del tiempo estarán esperando a la E/S y el procesador estará desocupado. Por ello, hace falta repartir eficientemente la memoria para meter tantos procesos como sea posible.

Este capítulo comienza con una ojeada a los requisitos que pretende satisfacer la gestión de memoria. A continuación, se plantea un enfoque de la tecnología de gestión de memoria que considera varios esquemas simples que se han venido usando. Se partirá de la necesidad de que un programa esté cargado en memoria principal para poder ser ejecutado. Esta discusión servirá para introducir algunos de los principios fundamentales de la gestión de memoria.

6.1

REQUISITOS DE LA GESTIÓN DE MEMORIA

Al realizar un estudio de los diversos mecanismos y políticas relacionadas con la gestión de memoria, vale la pena tener en mente los requisitos que se intentan satisfacer: [LIST88] propone cinco requisitos:

- Reubicación
- Protección
- Compartición
- Organización lógica
- Organización física

Reubicación

En un sistema multiprogramado, la memoria disponible se encuentra normalmente compartida por varios procesos. En general, el programador no puede conocer por adelantado qué otros programas residirán en memoria en el momento de la ejecución del programa. Además, se busca poder cargar y descargar los procesos activos en la memoria principal para maximizar el uso del procesador, manteniendo una gran reserva de procesos listos para ejecutar. Una vez que un programa haya sido descargado al disco, se limitará a declarar que, cuando vuelva a ser cargado, debe situarse en la misma región de memoria principal que antes.

De este modo, se sabe antes de tiempo dónde debe situarse un programa y hay que permitir que el programa pueda moverse en memoria principal como resultado de un intercambio. Esta situación plantea algunos asuntos técnicos relativos al direccionamiento, tal y como se muestra en la figura 6.1, que representa la imagen de un proceso. Por simplicidad, se supondrá que la imagen del proceso ocupa una región contigua de la memoria principal. Sin duda, el sistema operativo tiene que conocer la ubicación de la información de control del proceso y de la pila de ejecución, así como el punto de partida para comenzar la ejecución del programa para dicho proceso. Puesto que el sistema operativo gestiona

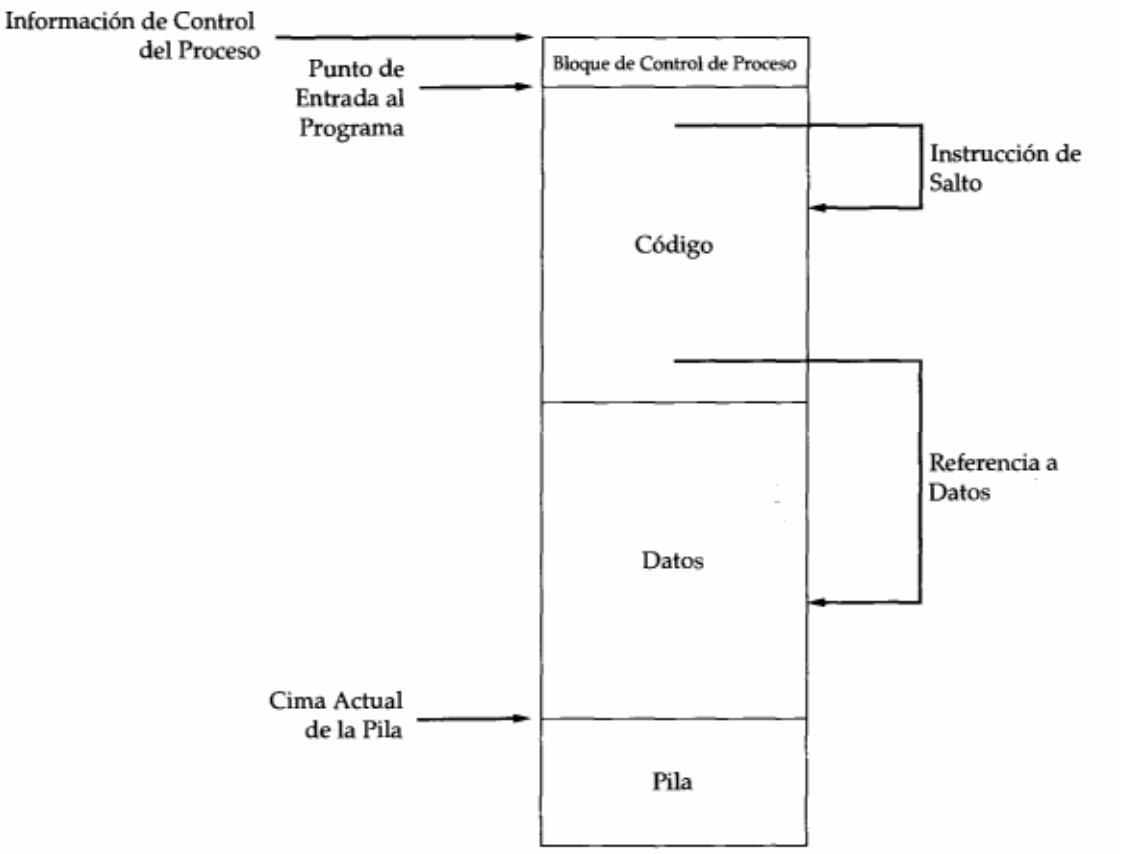


FIGURA 6.1 Requisitos de direccionamiento para un proceso

la memoria y es responsable de traer el proceso a memoria principal, estas direcciones deben ser fáciles de conseguir. Además, el procesador debe ocuparse de las referencias a memoria dentro del programa. Las instrucciones de bifurcación deben contener la dirección que haga referencia a la instrucción que se vaya a ejecutar a continuación. Las instrucciones que hagan referencia a datos deben contener la dirección del byte o de la palabra de datos referenciada. De algún modo, el hardware del procesador y el software del sistema operativo deben ser capaces de traducir las referencias a memoria encontradas en el código del programa a las direcciones físicas reales que reflejen la posición actual del programa en memoria principal.

Protección

Cada proceso debe protegerse contra interferencias no deseadas de otros procesos, tanto accidentales como intencionadas. Así pues, el código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos, con fines de lectura o escritura, sin permiso. Hasta cierto punto, satisfacer las exigencias de reubicación aumenta la dificultad de satisfacción de las exigencias de protección. Puesto que se desconoce la ubicación de un programa en memoria principal, es imposible comprobar las direcciones absolutas durante la compilación para asegurar la protección. Es más, la mayoría de los lenguajes de programación permiten el cálculo dinámico de direcciones durante la ejecución, generando, por ejemplo, un índice de un vector o un puntero a una estructura de datos. Por tanto, todas las referencias a memoria generadas por un proceso deben comprobarse durante la ejecución para asegurar que sólo hacen referencia al espacio de memoria destinado a dicho proceso. Afortunadamente, como se verá, los mecanismos que respaldan la reubicación también forman parte básica del cumplimiento de las necesidades de protección.

La imagen del proceso de la figura 6.1 ilustra las necesidades de protección. Normalmente, un proceso de usuario no puede acceder a ninguna parte del sistema operativo, tanto programa como datos. De nuevo, el programa de un proceso no puede en general bifurcar hacia una instrucción de otro proceso. Además, sin un acuerdo especial, el programa de un proceso no puede acceder al área de datos de otro proceso. El procesador debe ser capaz de abandonar tales instrucciones en el momento de la ejecución.

Nótese que, en los términos del ejemplo, las exigencias de protección de memoria pueden ser satisfechas por el procesador (hardware) en vez de por el sistema operativo (software). Esto es debido a que el sistema operativo no puede anticiparse a todas las referencias a memoria que hará un programa. Incluso si tal anticipación fuera posible, sería prohibitivo en términos de tiempo consumido el proteger cada programa por adelantado de posibles violaciones de referencias a memoria. Así pues, sólo es posible evaluar la tolerancia de una referencia a memoria (acceso a datos o bifurcación) durante la ejecución de la instrucción que realiza la referencia. Para llevar esto a cabo, el hardware del procesador debe poseer dicha capacidad.

Compartición

Cualquier mecanismo de protección que se implemente debe tener la flexibilidad de permitir el acceso de varios procesos a la misma zona de memoria principal. Por ejemplo, si una serie de procesos están ejecutando el mismo programa, resultaría beneficioso permitir a cada proceso que acceda a la misma copia del programa, en lugar de tener cada uno su pro-

pía copia aparte. Los procesos que cooperan en una tarea pueden necesitar acceso compartido a la misma estructura de datos. El sistema de gestión de memoria debe, por tanto, permitir accesos controlados a las áreas compartidas de la memoria, sin comprometer la protección básica. De nuevo, se verá que los mecanismos empleados para respaldar la reubicación forman parte básica de las capacidades de compartición.

Organización Lógica

De forma casi invariable, la memoria principal de un sistema informático se organiza como un espacio de direcciones lineal o unidimensional que consta de una secuencia de bytes o palabras. La memoria secundaria, a nivel físico, se organiza de forma similar. Si bien esta organización refleja fielmente el hardware de la máquina, no se corresponde con la forma en la que los programas están construidos habitualmente. La mayoría de los programas se organizan en módulos, algunos de los cuales no son modificables (sólo lectura, sólo ejecución) y otros contienen datos que se pueden modificar. Si el sistema operativo y el hardware del computador pueden tratar de forma efectiva los programas de usuario y los datos en forma de módulos de algún tipo, se conseguirá una serie de ventajas, tales como:

1. Los módulos pueden escribirse y compilarse independientemente, mientras que el sistema resuelve durante la ejecución todas las referencias de un módulo a otro.

2. Con un escaso coste adicional, pueden otorgarse varios grados de protección (sólo lectura, sólo ejecución) a los distintos módulos.

3. Es posible introducir mecanismos por medio de los cuales los procesos puedan compartir módulos. La ventaja de ofrecer compartición a nivel de módulo es que esto se corresponde con la visión del problema que üene el usuario y, por tanto, es fácil para el usuario especificar la compartición que desea.

La herramienta que más fácilmente satisface estas necesidades es la segmentación, que es una de las técnicas de gestión de memoria estudiadas en este capítulo.

Organización Física

Como se discutió en la sección 1.6, la memoria del computador se organiza en, al menos, dos niveles: memoria principal y memoria secundaria. La memoria principal ofrece un acceso rápido con un coste relativamente alto. Además, la memoria principal es volátil; esto es, no proporciona almacenamiento permanente. La memoria secundaria es más lenta y barata que la memoria principal y, normalmente, no es volátil. De este modo, una memoria secundaria de gran capacidad puede permitir un almacenamiento a largo plazo de programas y datos, al tiempo que una memoria principal pequeña mantiene los programas y datos de uso actual.

En este esquema a dos niveles, la organización del flujo de información entre la memoria principal y la secundaria tiene un gran interés en el sistema. La responsabilidad de este flujo podría asignarse al programador, pero esto es impracticable e indeseable, debido a dos razones:

1. La memoria principal disponible para un programa y sus datos puede ser insuficiente. En este caso, el programador debe emplear una práctica que se conoce como *superposición (overlaying),* en la cual el programa y los datos se organizan de tal forma que puede haber varios módulos asignados a la misma región de memoria, con un pro-

grama principal responsable del intercambio de los módulos según se necesite. Incluso con la ayuda de herramientas de compilación, la programación superpuesta malgasta el tiempo del programador.

2. En un entorno multiprogramado, el programador no conoce durante la codificación cuánto espacio habrá disponible o dónde estará este espacio.

Resulta claro entonces que la tarea de mover información entre los dos niveles de memoria debe ser responsabilidad del sistema. Esta tarea es la esencia de la gestión de memoria.

6.2 CARGA DE PROGRAMAS EN MEMORIA PRINCIPAL

La tarea central de cualquier sistema de gestión de memoria es traer los programas a memoria principal para su ejecución en el procesador. En casi todos los sistemas multiprogramados modernos, esta tarea supone un esquema sofisticado conocido como *memoria virtual*. La memoria virtual está, a su vez, basada en el uso de una de dos técnicas básicas; segmentación y/o paginación. Antes de ver estás técnicas de memoria virtual, se debe preparar el terreno considerando técnicas más simples que no requieren el uso de memoria virtual. Una de estas técnicas, la partición, se ha venido usando con distintas variantes en algunos sistemas operativos ahora obsoletos. Las otras dos técnicas, la paginación simple y la segmentación simple, no se usan en solitario. No obstante, el estudio de la memoria virtual resultará más sencillo si se consideran en primer lugar estas dos técnicas, sin tener en cuenta la memoria virtual. La tabla 6.1 adelanta los mecanismos cubiertos en este capítulo y en el siguiente.

Partición fija

En la mayoría de los esquemas de gestión de memoria, se puede suponer que el sistema operativo ocupa una parte fija de memoria principal y que el resto de la memoria está disponible para ser usado por varios procesos. El esquema más sencillo de gestión de la memoria disponible es dividirla en regiones con límites fijos.

Tamaños de Partición

En la figura 6.2 se ofrecen ejemplos de dos alternativas de partición fija. Una posibilidad es emplear particiones de igual tamaño. En este caso, cualquier proceso cuyo tamaño sea menor o igual que el tamaño de la partición puede cargarse en cualquier partición libre. Si todas las particiones están ocupadas y no hay procesos residentes en estado Listo o Ejecutando, el sistema operativo puede sacar un proceso de alguna de las particiones y cargar otro proceso de forma que haya trabajo para el procesador.

Las particiones fijas de igual tamaño plantean dos dificultades:

• Un programa puede ser demasiado grande para caber en la partición. En este caso, el programador debe diseñar el programa mediante superposiciones, para que sólo una parte del programa esté en memoria principal en cada instante. Cuando se necesita un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndose a los programas y datos que se encuentren en ella.

Tabla 6.1 Técnicas de Gestión de Memoria

Técnica	Descripción	Ventajas	Desventajas
Partición fija	La memoria principal se divide en un conjunto de particiones fijas durante la generación del sistema. Un proceso se puede cargar en una partición de mayor o igual tamaño.	Sencilla de implementar; poca sobrecarga del sistema operativo.	Empleo ineficiente de la memoria debido a la fragmentación interna; el número de procesos activos es fijo.
Partición Dinámica	Las particiones se crean dinámicamente, de forma que cada proceso se carga en una partición de exactamente el mismo tamaño que el proceso.	No hay fragmentación interna; uso más eficiente de la memoria principal.	Uso ineficiente del procesador debido a la necesidad de compactación para contrarrestar la fragmentación externa.
Paginación Simple	La memoria principal se divide en un conjunto de marcos de igual tamaño. Cada proceso se divide en una serie de páginas del mismo tamaño que los marcos. Un proceso se carga situando todas sus páginas en marcos libres pero no necesariamente contiguos.	No tiene fragmentación externa.	Hay una pequeña cantidad de fragmentación interna.
Segmentación Simple	Cada proceso se divide en una serie de segmentos. Un proceso se carga situando todos sus segmentos en particiones dinámicas que no tienen por qué ser contiguas.	No tiene fragmentación interna.	Necesita compactación.
Memoria Virtual Paginada	Como la paginación simple, excepto que no hace falta cargar todas las páginas de un proceso. Las páginas no residentes que se necesiten se traerán más tarde, de manera automática.	No hay fragmentación externa; alto grado de multiprogramación; gran espacio virtual para el proceso.	Sobrecarga por gestión compleja de memoria.
Memoria Virtual Segmentada	Como la segmentación simple, excepto que no es necesario cargar todos los segmentos de un proceso. Las segmentos no residentes que se necesiten se traerán más tarde, de forma automática.	No hay fragmentación interna; alto grado de multiprogramación; gran espacio virtual para el proceso; soporte de protección y compartición.	Sobrecarga por gestión compleja de memoria.

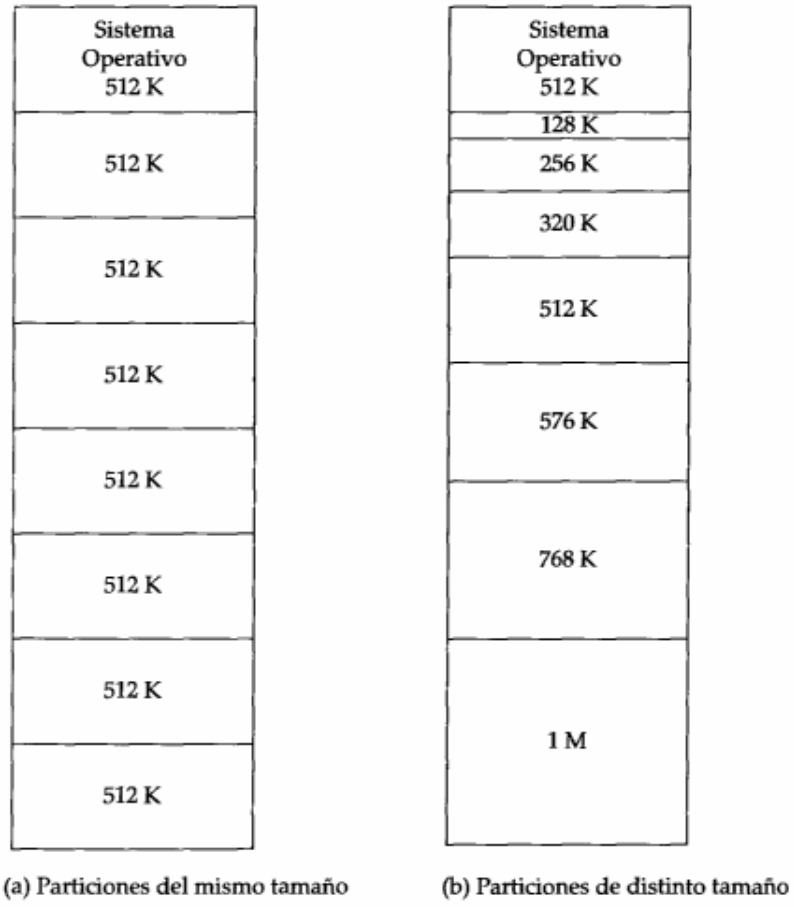


FIGURA 6.2 Ejemplo de partición estática de una memoria de 4Mb

• El uso de memoria principal es extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupará una partición completa. En el ejemplo, puede haber un programa que ocupe menos de 128Kb de memoria y, aún así, ocuparía una partición de 512Kb cada vez que se cargase. Este fenómeno, en el que se malgasta el espacio interno de una partición cuando el bloque de datos cargado sea más pequeño que la partición, se denomina **fragmentación interna.**

Pueden reducirse, aunque no solventarse, ambos problemas, por medio del empleo de particiones de tamaños distintos, como se muestra en la figura 5.2b. En este ejemplo, los programas de hasta IMb pueden alojarse sin superposición. Las particiones menores de 512K permiten alojar programas más pequeños con un desperdicio menor.

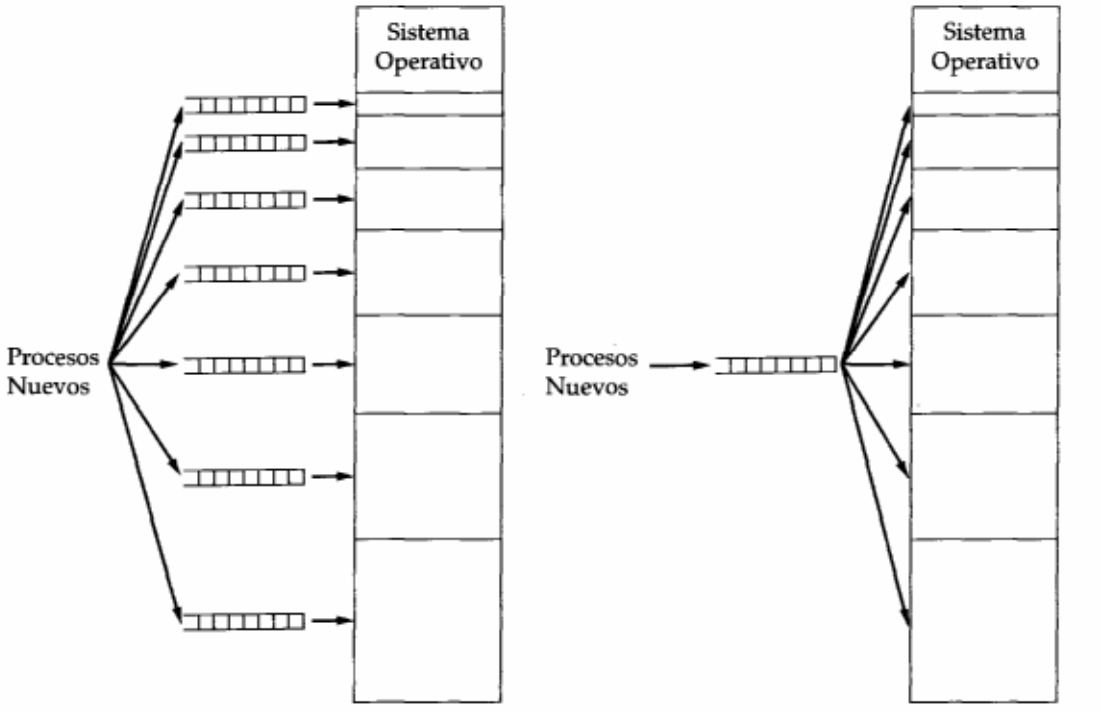
Algoritmo de Ubicación

Con particiones del mismo tamaño, la ubicación de un proceso en memoria es trivial. Mientras haya alguna partición libre, puede cargarse un proceso en esa partición. Puesto que todas las particiones son de igual tamaño, no importa la partición que se use. Si todas las par-

ticiones están ocupadas con procesos que no están listos para ejecutar, uno de esos procesos debe sacarse y hacer sitio para un nuevo proceso. Cuál debe expulsarse es una decisión de planificación; este punto se tratará en el capítulo 8.

Con particiones de distintos tamaños, hay dos maneras posibles de asignar los procesos a las particiones. La forma más simple es asignar cada proceso a la partición más pequeña en la que quepa'. En este caso, hace falta una cola de planificación para cada partición, que albergue los procesos expulsados cuyo destinado es dicha partición (figura 6.3a). La ventaja de este enfoque es que los procesos están siempre asignados de forma que se minimiza la memoria desperdiciada dentro de cada partición.

Sin embargo, aunque esta técnica parece óptima desde el punto de vista de una partición individual, no lo es desde el punto de vista del sistema global. Considérese el caso de la figura 6.2b, por ejemplo, donde no hay procesos con un tamaño comprendido entre 768K y 1M en un determinado instante. En este caso, la partición de 768K permanecerá sin usar, incluso aunque algún proceso más pequeño pudiera haber sido asignado a la misma. Así pues, una solución mejor sería emplear una única cola para todos los procesos (figura 6.3b). Cuando se va a cargar un proceso en memoria principal, se selecciona la partición más pequeña disponible que



(a) Una cola de procesos por partición (b) Cola única de procesos

FIGURA 6.3 Asignación de memoria con partición fija

[1] Se supone que se conoce la cantidad máxima de memoria que necesitará un proceso, lo que no siempre es cierto. Si no se sabe lo grande que puede llegar a ser un proceso, las únicas alternativas son la superposición o la memoria virtual.

pueda albergar al proceso. Si todas las particiones están ocupadas, se debe tomar una decisión de intercambio. Puede darse preferencia al intercambio de la partición más pequeña que pueda contener al proceso entrante. También es posible considerar otros factores, tales como prioridades y preferencia para descargar procesos bloqueados antes que procesos listos.

El uso de particiones de distinto tamaño proporciona cierto grado de flexibilidad a las particiones fijas. Además, ambos tipos de esquema de partición fija son relativamente simples y exigen un software del sistema operativo y una sobrecarga de procesamiento mínimos. Sin embargo, se plantean los problemas siguientes:

• El número de particiones especificadas en el momento de la generación del sistema limita el número de procesos activos (no suspendidos) del sistema.

• Puesto que los tamaños de partición se programan en el momento de la generación del sistema, los trabajos pequeños no hacen un uso eficiente del espacio de las particiones. En un entorno en el que los requisitos básicos de almacenamiento de todos los procesos se conocen de antemano, puede ser una técnica razonable, pero, en la mayoría de los casos, ineficiente.

El uso de la partición fija es casi nulo hoy día. Como ejemplo de un sistema operativo fructuoso que empleaba está técnica se tiene un antiguo sistema operativo de grandes computadores de IBM, el OS/MFT (Multiprogramación con un número Fijo de Tareas).

Partición Dinámica

Para superar algunas de las dificultades de la partición estática, se desarrolló una solución denominada *partición dinámica.* Otra vez, este enfoque ha sido superado de largo por técnicas de gestión de memoria más sofisticadas. Un sistema operativo importante que empleaba esta técnica fue el antiguo OS/MVT (Multiprogramación con un número Variable de Tareas), para grandes computadores de IBM.

Con la partición dinámica, las particiones son variables en número y longitud. Cuando se trae un proceso a memoria principal, se le asigna exactamente tanta memoria como necesita y no más. En la figura 6.4 se muestra un ejemplo que usa 1MB de memoria principal. Al principio, la memoria principal está vacía, exceptuando el sistema operativo (figura 6.4a). Se cargan los tres primeros procesos, empezando en donde acaba el sistema operativo y ocupando sólo un espacio suficiente para cada proceso (figuras 6.4b, c y d). Esto deja un "hueco" al final de la memoria demasiado pequeño para un cuarto proceso. En algún momento, ninguno de los procesos en memoria está listo. Así pues, el sistema operativo saca al proceso 2 (figura 6.4e), que deja sitio suficiente para cargar un nuevo proceso, el proceso 4 (figura 6.4f). Puesto que el proceso 4 es más pequeño que el proceso 2, se crea otro hueco pequeño. Más tarde, se alcanza un punto en el que ninguno de los procesos que están en memoria principal están listos y el proceso 2, que está en estado Listo, pero suspendido, está disponible. Puesto que no hay suficiente sitio en memoria para el proceso 2, el sistema operativo expulsa al proceso 1 (figura 6.4g) y carga de nuevo el proceso 2 (figura 6.4h).

Como se muestra en el ejemplo, este método comienza bien, pero, finalmente, desemboca en una situación en la que hay un gran número de huecos pequeños en memoria. Conforme pasa el tiempo, la memoria comienza a estar más fragmentada y su rendimiento decae. Este fenómeno se denomina **fragmentación externa** y se refiere al hecho de que la memoria externa a todas las particiones se fragmenta cada vez más, a diferencia de la fragmentación interna, que se comentó anteriormente.

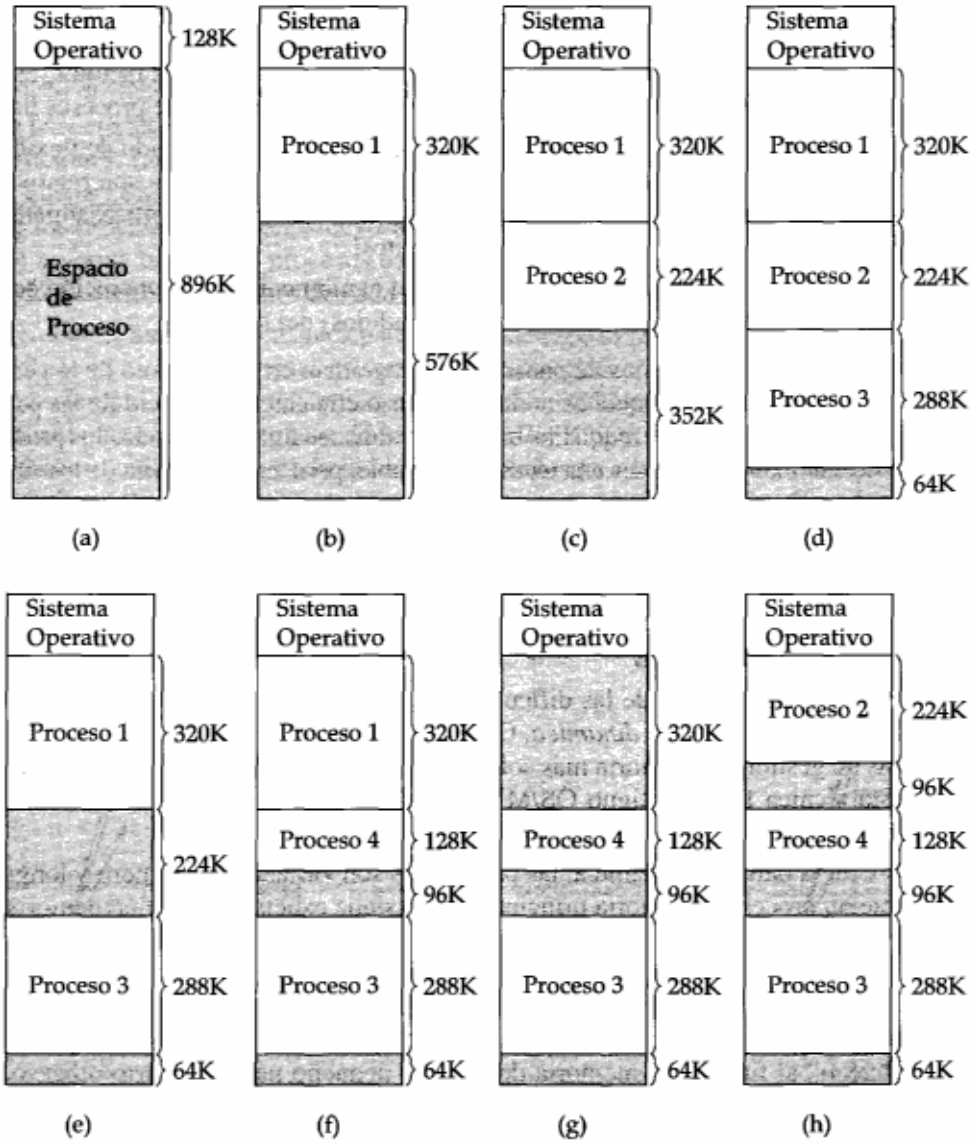


FIGURA 6.4 Efectos de la partición dinámica

Una técnica para superar la fragmentación externa es la **compactación:** De vez en cuando, el sistema operativo desplaza los procesos para que estén contiguos de forma que toda la memoria libre quede junta en un bloque. Por ejemplo, en la figura 6.4h, la compactación produce un bloque de memoria libre de 256K. Este hueco puede ser suficiente para cargar un proceso adicional. La dificultad de la compactación está en que es un procedimiento que consume tiempo, por lo que desperdicia tiempo del procesador. La compactación necesita de la capacidad de reubicación dinámica. Es decir, se debe poder mover un programa de una región a otra de memoria principal sin invalidar las referencias a memoria del programa (véase el Apéndice 6A).

Algoritmo de Ubicación

Puesto que la compactación de memoria consume tiempo, atañe al diseñador del sistema operativo decidir adecuadamente cómo asignar un proceso a memoria (como llenar los huecos). Cuando llega el momento de cargar o traer un proceso a memoria principal y, si hay libre más de un bloque de memoria de tamaño suficiente, el sistema operativo debe decidir cuál asignar.

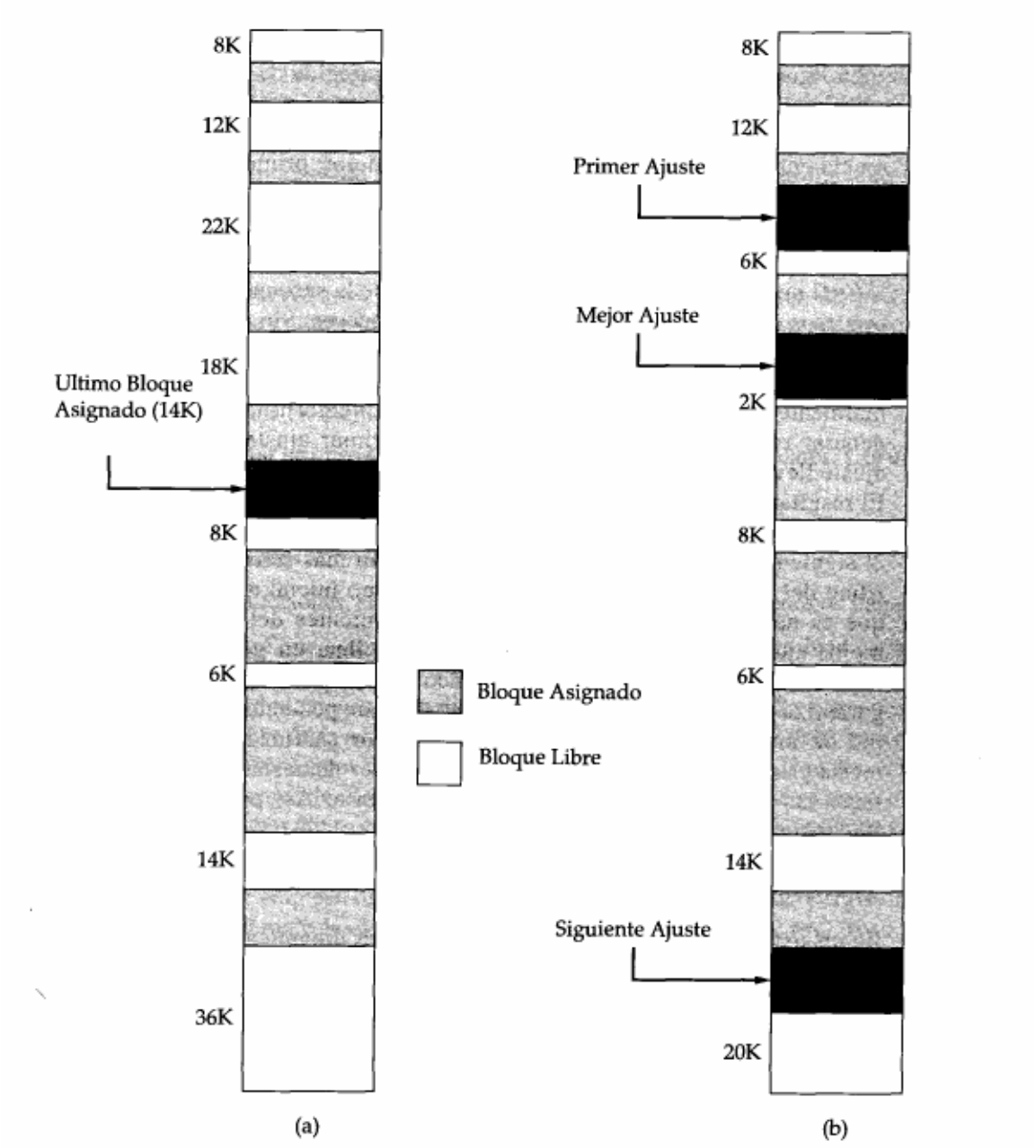


FIGURA 6.5 Ejemplo de una configuración de memoria antes y después de asignar un bloque de 16Kb

Los tres algoritmos de ubicación que se pueden considerar son el del mejor ajuste *(best-fit),* el del primer ajuste *(first-fit) y* el del siguiente ajuste *(next-fit).* Todos ellos se limitan a elegir entre los bloques de memoria libres que son mayores o iguales que el proceso a traer. El mejor ajuste elige el bloque de tamaño más parecido al solicitado. El primer ajuste comienza recorriendo la memoria desde el principio y escoge el primer bloque disponible que sea suficientemente grande. El siguiente ajuste recorre la memoria desde el lugar de la última ubicación y elige el siguiente bloque disponible que sea suficientemente grande.

La figura 6.5a muestra un ejemplo de configuración de la memoria después de cierto número de ubicaciones y operaciones de descarga de procesos. El último bloque usado fue de 22Kb, de donde se creó una partición de 14Kb. La figura 6.5b muestra la diferencia entre los algoritmos de ubicación del mejor, primer y siguiente ajuste para una solicitud de 16Kb. El mejor ajuste busca en la lista completa de bloques disponibles y emplea el hueco de 18Kb, dejando un fragmento de 2Kb. El primer ajuste genera un fragmento de 6Kb y el siguiente ajuste origina un fragmento de 20Kb.

Cuál de estos métodos es mejor dependerá de la secuencia exacta de intercambios de procesos que se den y del tamaño de estos procesos. Sin embargo, se pueden hacer algunos comentarios generales (pueden consultarse además [BAYS77], [BREN89] y [SHOR75]). El algoritmo del primer ajuste no sólo es el más sencillo, sino que normalmente es también el mejor y más rápido. El algoritmo del siguiente ajuste tiende a generar resultados algo peores que el del primer ajuste. El algoritmo del siguiente ajuste llevará frecuentemente a la asignación de bloques libres del final de la memoria. El resultado es que el bloque de memoria libre más grande, que suele aparecer al final del espacio de memoria, se divide rápidamente en fragmentos pequeños. Así pues, con el siguiente ajuste hará falta una compactación más frecuente. Por otro lado, el algoritmo del primer ajuste puede poblar el extremo inicial de pequeñas particiones libres que es necesario recorrer en las pasadas siguientes del algoritmo. El algoritmo del mejor ajuste, a pesar de su nombre, proporciona en general los peores resultados. Puesto que este algoritmo busca el hueco más pequeño que cumple con los requisitos, garantiza que el fragmento que se deja es lo más pequeño posible. Aunque cada solicitud de memoria desperdicia siempre la menor cantidad, el resultado es que la memoria principal se llena rápidamente de bloques demasiado pequeños como para satisfacer las solicitudes de asignación de memoria. Así pues, se debe compactar más frecuentemente que con los otros algoritmos.

Algoritmos de Reemplazo

En un sistema multiprogramado con particiones dinámicas, habrá algún momento en el que todos los procesos de memoria principal estén en estado bloqueado y la memoria sea insuficiente, incluso tras la compactación, para un proceso adicional. Para evitar desperdiciar el tiempo del procesador esperando a que un proceso activo se desbloquee, el sistema operativo expulsará uno de los procesos de memoria principal para hacer sitio a un proceso nuevo o un proceso Listo, pero suspendido. Por lo tanto, el sistema operativo debe elegir qué proceso reemplazar. Puesto que el tema de los algoritmos de reemplazo se cubre con mayor detalle en varios esquemas de memoria virtual, se aplazará la discusión de estos algoritmos hasta entonces.

Reubicación

Antes de considerar las formas de solucionar los defectos de las técnicas de partición, se debe aclarar un punto oscuro, que tiene relación con la ubicación de los procesos en memoria. Cuando se emplea el esquema de particiones fijas de la figura 6.3a, se puede esperar que un proceso sea asignado siempre a la misma partición. Es decir, la partición que se selecciona cuando se carga un nuevo proceso será la misma que se emplee siempre para devolver ese proceso a memoria, tras haber sido sacado. En este caso, se puede emplear un cargador sencillo, tal como se describe en el Apéndice 6A: Cuando el proceso se carga por primera vez, todas las referencias relativas a memoria en el código se reemplazan por direcciones absolutas de memoria principal, determinadas por la dirección base del proceso cargado.

En el caso de particiones de igual tamaño y en el caso de una cola única de procesos para particiones de distinto tamaño, un proceso puede ocupar diferentes particiones a lo largo de su vida. Cuando al principio se crea la imagen de un proceso, se cargará en alguna partición de memoria principal. Posteriormente, el proceso puede ser descargado; cuando, más tarde, vuelva a ser cargado, podrá asignársele una partición distinta de la anterior. Esto mismo se cumple con particiones dinámicas. Obsérvese en las figuras 6.4c y 6.4h que el proceso 2 ocupa dos regiones de memoria distintas en las dos ocasiones en las que se le trae a memoria. Es más, cuando se usa compactación, los procesos son desplazados durante su estancia en memoria principal.

Considérese ahora un proceso en memoria que incluya instrucciones y datos. Las instrucciones contendrán referencias a memoria de los dos tipos siguientes:

• Direcciones de elementos de datos, empleadas en instrucciones de carga, almacenamiento y en algunas instrucciones aritméticas y lógicas.

• Direcciones de instrucciones, empleadas para bifurcaciones e instrucciones de llamada.

Ahora se demostrará que estas instrucciones no son fijas, sino que cambian cada vez que se intercambia o desplaza un proceso. Para resolver este problema, se realiza una distinción entre varios tipos de direcciones. Una **dirección lógica** es una referencia a una posición de memoria independiente de la asignación actual de datos a memoria; se debe hacer una traducción a dirección física antes de poder realizar un acceso a memoria. Una **dirección relativa** es un caso particular de dirección lógica, en el cual la dirección se expresa como una posición relativa a algún punto conocido, habitualmente el comienzo del programa. Una **dirección física** o dirección absoluta, es una posición real en memoria principal.

Los programas que emplean direcciones relativas en memoria se cargan por medio de cargadores dinámicos durante la ejecución (véase el estudio del Apéndice 6A). Esto significa que todas las referencias a memoria en el proceso cargado son relativas al origen del programa. Así pues, se necesita en el hardware un medio para traducir las direcciones relativas a direcciones físicas en memoria principal en el momento de la ejecución de la instrucción que contiene la referencia.

/ La figura 6.6 muestra la forma en que se realiza normalmente esta traducción de direcciones.

Cuando un proceso pasa a estado Ejecutando, se carga con la dirección en memoria principal del proceso un registro especial del procesador, a veces denominado registro base. Existe también un registro límite que indica la posición final del programa; estos valores deben asignarse cuando se

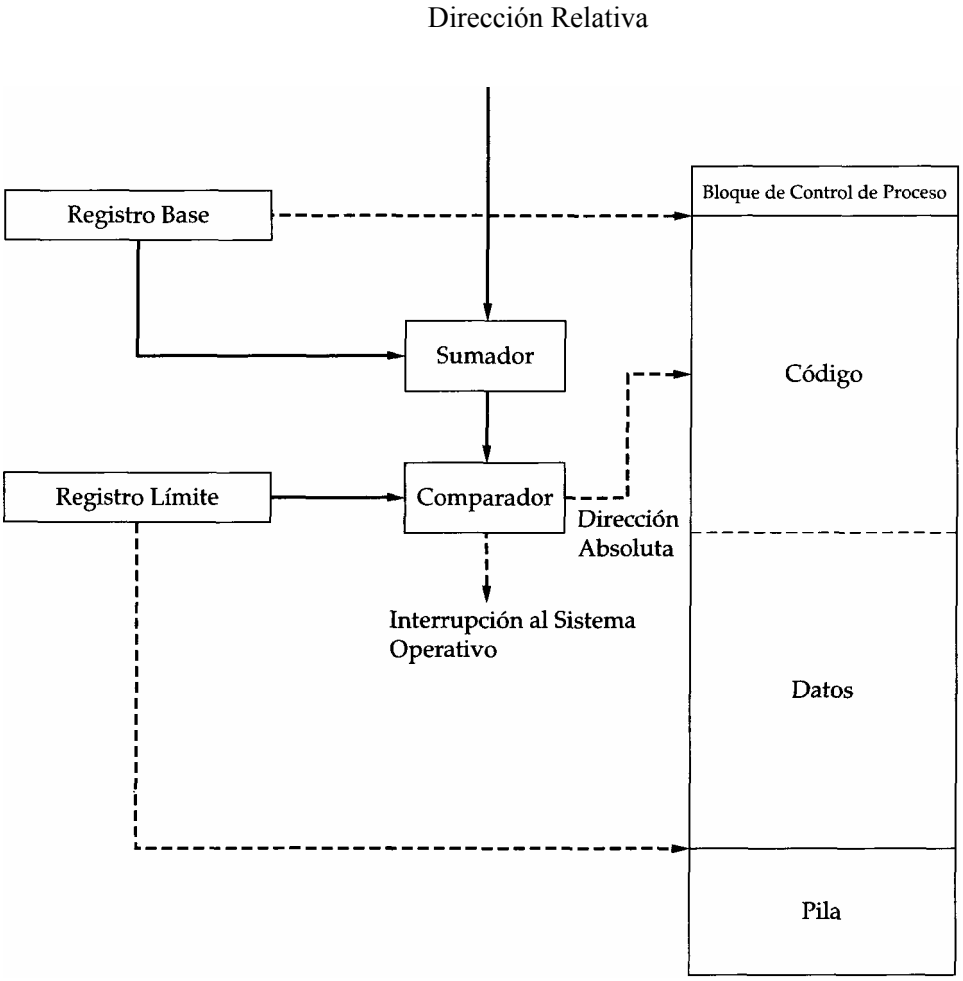


FIGURA 6.6 Soporte de hardware para la reubicación

carga el programa en memoria o cuando se carga la imagen del proceso. A lo largo de la ejecución del proceso se encuentran direcciones relativas. Estas direcciones incluyen los contenidos del registro de instrucción, las direcciones que aparecen en las instrucciones de bifurcación y llamada, y las direcciones de datos que aparecen en las instrucciones de carga y almacenamiento. Cada una de estas direcciones relativas pasa por dos etapas de manipulación en el procesador.

En primer lugar, se añade el valor del registro base a la dirección relativa para obtener una dirección absoluta. En segundo lugar, la dirección obtenida se compara con el valor del registro límite. Si la dirección está dentro de los omites, se puede proceder con la ejecución de la instrucción. En otro caso, se genera una interrupción al sistema operativo, que debe responder al error de algún modo.

El esquema de la figura 6.6 permite a los programas cargarse y descargarse de memoria a lo largo de la ejecución. También proporciona una medida de protección: Cada imagen de proceso está aislada por el contenido de los registros base y límite, y es segura contra accesos no deseados de otros procesos.

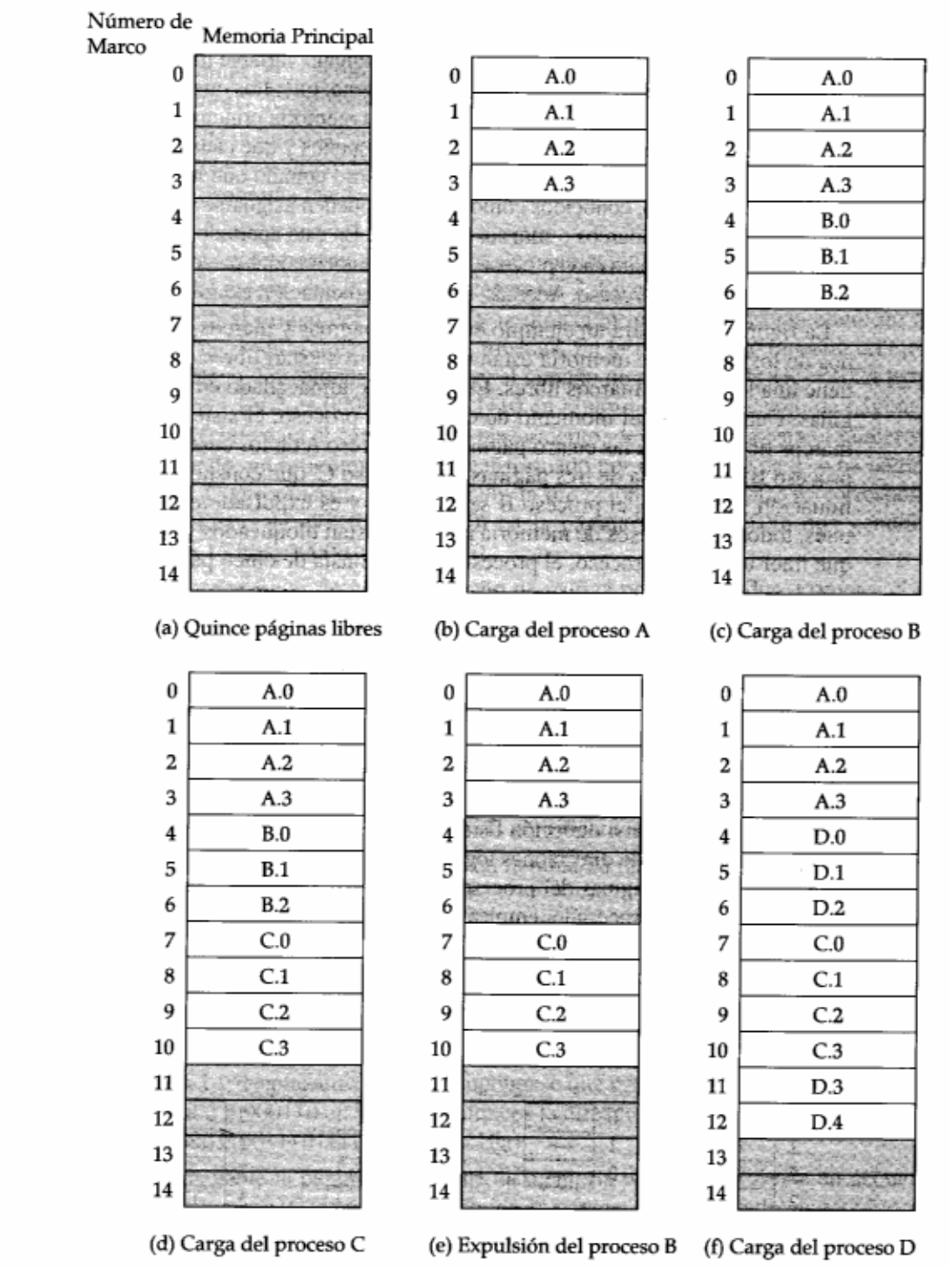


FIGURA 6.7 Asignación de páginas de procesos a marcos libres

Paginación Simple

Tanto las particiones de tamaño fijo como las de tamaño variable hacen un uso ineficiente de la memoria; las primeras generan fragmentación interna, mientras que las segundas originan fragmentación externa. Supóngase, no obstante, que la memoria principal se encuentra particionada en trozos iguales de tamaño fijo relativamente pequeños y que cada proceso está dividido también en pequeños trozos de tamaño fijo y del mismo tamaño que los de memoria. En tal caso, los trozos del proceso, conocidos como **páginas,** pueden asignarse a los trozos libres de memoria, conocidos como marcos o **marcos** de página. En este apartado se verá que el espacio malgastado en memoria para cada proceso por fragmentación interna consta sólo de una fracción de la última página del proceso. Además, no hay fragmentación externa.

La figura 6.7 muestra un ejemplo del uso de páginas y marcos. En un instante dado, algunos de los marcos de memoria están en uso y otros están libres. El sistema operativo mantiene una lista de los marcos libres. El proceso A, almacenado en disco, consta de cuatro páginas. Cuando llega el momento de cargar este proceso, el sistema operativo busca cuatro marcos libres y carga las cuatro páginas del proceso A en los cuatro marcos (figura 6.7b). El proceso B, que consta de tres páginas y el proceso C, que consta de cuatro, se cargan a continuación. Más tarde, el proceso B se suspende y es expulsado de memoria principal. Después, todos los procesos de memoria principal están bloqueados y el sistema operativo tiene que traer un nuevo proceso, el proceso D, que consta de cinco páginas.

Supóngase ahora, como en este ejemplo, que no hay suficientes marcos sin usar contiguos para albergar al proceso. ¿Impedirá esto al sistema operativo cargar D? La respuesta es negativa, puesto que se puede emplear de nuevo el concepto de dirección lógica. Ya no será suficiente con un simple registro base. En su lugar, el sistema operativo mantiene una **tabla de páginas** para cada proceso. La tabla de páginas muestra la posición del marco de cada página del proceso. Dentro del programa, cada dirección lógica constará de un número de página y de un desplazamiento dentro de la página. Recuérdese que, en el caso de la partición simple, una dirección lógica era la posición de una palabra relativa al comienzo del programa; el procesador realizaba la traducción a dirección física. Con paginación, el hardware del procesador también realiza la traducción de direcciones lógicas a físicas. Ahora, el procesador debe saber cómo acceder a la tabla de páginas del proceso actual. Dada una dirección lógica (número de página, desplazamiento), el procesador emplea la tabla de páginas para obtener una dirección física (número de marco, desplazamiento).

Continuando con el ejemplo, las cinco páginas del proceso D se cargan en los marcos 4, 5, 6, 11 y 12. La figura 6.8 muestra las distintas tablas de páginas en este instante. Cada tabla de páginas contiene una entrada por cada página del proceso, por lo que la tabla se indexa

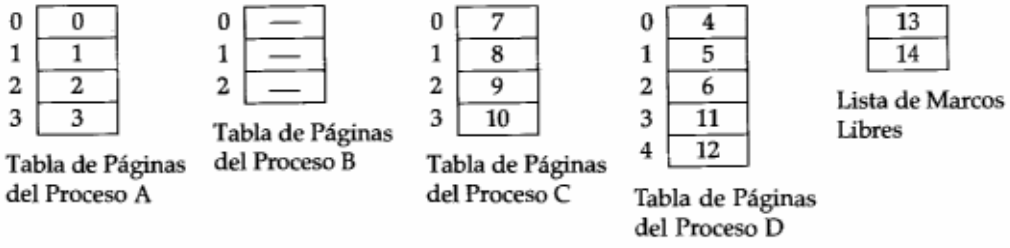


FIGURA 6.8 Estructuras de datos para el ejemplo de la figura 6.7 en el periodo de tiempo (f)

fácilmente por número de página (comenzando en la página 0). En cada entrada de la tabla de páginas se encuentra el número de marco en memoria, si lo hay, que alberga la página correspondiente. Además, el sistema operativo mantiene una lista de marcos libres con todos los marcos de memoria que actualmente están vacíos y disponibles para las páginas.

Así pues, se puede comprobar que la paginación simple, tal y como se describe, es similar a la partición estática. Las diferencias están en que, con paginación, las particiones son algo más pequeñas, un programa puede ocupar más de una partición y éstas no tienen por qué estar contiguas.

Para aplicar convenientemente este esquema de paginación, el tamaño de la página y, por tanto, el tamaño del marco, deben ser una potencia de 2. En este caso, la dirección relativa, definida en relación al origen del programa y la dirección lógica, expresada como un número de página y un desplazamiento, son las mismas. En la figura 6.9 se muestra un ejemplo, donde se emplean direcciones de 16 bits y el tamaño de página es de 1K = 1024 bytes. La dirección relativa 1502 es 0000010111011110 en binario. Con un tamaño de página de 1K, se necesitan 10 bits para el campo de desplazamiento, dejando 6 bits para el número de página. De este modo, un programa puede estar formado por un máximo de $2^6 = 64$ páginas de IKb cada una. Como muestra la figura 6.9b, la dirección relativa 1502 se corresponde con un desplazamiento de 478 (0111011110) en la página 1 (000001), lo que genera el mismo número de 16 bits, 0000010111011110.

Dos son las consecuencias de usar un tamaño de página potencia de dos. Primero, el esquema de direccionamiento lógico es transparente al programador, al ensamblador y al montador. Cada dirección lógica de un programa (número de página, desplazamiento) es idéntica a su dirección relativa. Segundo, resulta relativamente sencillo realizar una función hardware para llevar a cabo la traducción de direcciones dinámicas durante la ejecución. Considérese una dirección de $n + m$ bits, en la que los n bits más significativos son el número de página y los m bits menos significativos son el desplazamiento. En el ejemplo (figura 6.9b), n=6 y m=10. Para la traducción de direcciones hay que dar los siguientes pasos:

- Obtener el número de página de los n bits más significativos de la dirección lógica.
- Emplear el número de página como índice en la tabla de páginas del proceso para encontrar el número de marco k.
- El comienzo de la dirección física del marco es $k \times 2^m$ y la dirección física del byte referenciado es este número más el desplazamiento. No hace falta calcular esta dirección física, sino que se construye fácilmente concatenando el número de marco con el desplazamiento.

En el ejemplo, se tiene la dirección lógica 0000010111011110, que se corresponde con el número de página 1 y desplazamiento 478. Supóngase que esta página reside en el marco de memoria principal 6 = 000110 en binario. Entonces la dirección física es el marco 6, desplazamiento 478 = 0001100111011110 (figura 6.10a).

Resumiendo, mediante la paginación simple, la memoria principal se divide en pequeños marcos del mismo tamaño. Cada proceso se divide en páginas del tamaño del marco; los procesos pequeños necesitaran pocas páginas, mientras que los procesos grandes necesitarán más. Cuando se introduce un proceso en memoria, se cargan todas sus páginas en los marcos libres y se rellena su tabla de páginas. Esta técnica resuelve la mayoría de los problemas inherentes a la partición.

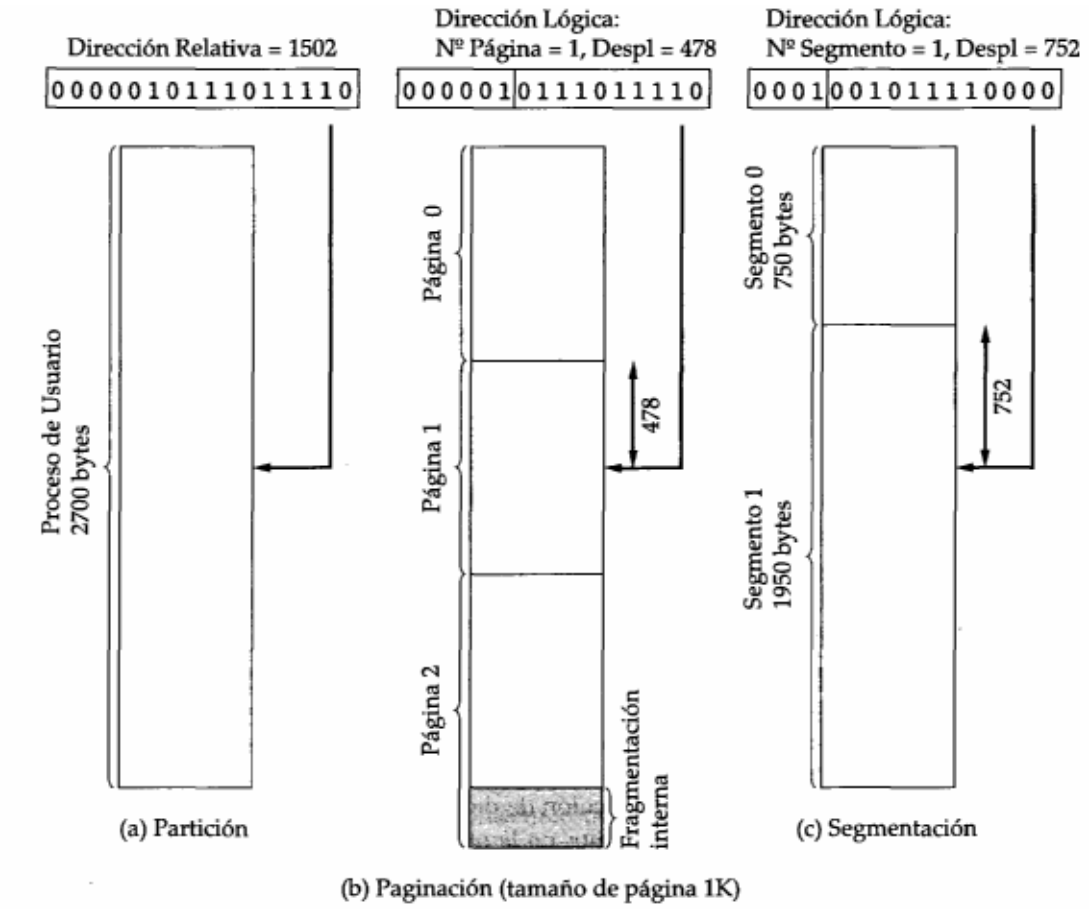


FIGURA 6.9 Direcciones Lógicas

Segmentación Simple

Otro modo de subdividir el programa es la segmentación. En este caso, el programa y sus datos asociados se dividen en un conjunto de **segmentos.** No es necesario que todos los segmentos de todos los programas tengan la misma longitud, aunque existe una longitud máxima de segmento. Como en la paginación, una dirección lógica segmentada consta de dos partes, en este caso un número de segmento y un desplazamiento.

Como consecuencia del empleo de segmentos de distinto tamaño, la segmentación resulta similar a la partición dinámica. En ausencia de un esquema de superposición o del uso de memoria virtual, sería necesario cargar en memoria todos los segmentos de un programa para su ejecución. La diferencia, en comparación con la partición dinámica, radica en que, con segmentación, un programa puede ocupar más de una partición y éstas no tienen por qué estar contiguas. La segmentación elimina la fragmentación interna, pero, como la partición dinámica, sufre de fragmentación externa. Sin embargo, debido a que los procesos se dividen en un conjunto de partes más pequeñas, la fragmentación externa será menor.

Mientras que la paginación es transparente al programador, la segmentación es generalmente visible y se proporciona como una comodidad para la organización de los programas y datos. Normalmente, el programador o el compilador asigna los programas y los datos a diferentes segmentos. En aras de la programación modular, el programa o los datos pueden ser divididos de nuevo en diferentes segmentos. El principal inconveniente de este servicio es que el programador debe ser consciente de la limitación de tamaño máximo de los segmentos.

Otra consecuencia del tamaño desigual de los segmentos es que no hay una correspondencia simple entre las direcciones lógicas y las direcciones físicas. De forma análoga a la paginación, un esquema de segmentación simple hará uso de una tabla de segmentos para cada proceso y una lista de bloques libres en memoria principal. Cada entrada de tabla de segmentos tendría que contener la dirección de comienzo del segmento correspondiente en memoria principal. La entrada deberá proporcionar también la longitud del segmento para asegurar que no se usan direcciones no válidas. Cuando un proceso pasa al estado Ejecutando, se carga la dirección de su tabla de segmentos en un registro especial del hardware de gestión de memoria. Considérese una dirección de $n + m$ bits, en la que los n bits más significativos son el número de segmento y los m bits menos significativos son el desplazamiento. En el ejemplo (figura 6.9c), $n=4$ y $m= 12$. Así pues, el máximo tamaño de segmento es $2^{12} = 4096$. Para la traducción de direcciones hay que dar los siguientes pasos:

- Extraer el número de segmento de los n bits más significativos de la dirección lógica.
- Emplear el número de segmento como índice en la tabla de segmentos del proceso para encontrar la dirección física de comienzo del segmento.
- Comparar el desplazamiento, expresado por los m bits menos significativos, con la longitud del segmento. Si el desplazamiento es mayor que la longitud, la dirección no es válida.
- La dirección física buscada es la suma de la dirección física de comienzo del segmento más el desplazamiento.

En el ejemplo, se emplea la dirección lógica 0001001011110000, que se corresponde con el número de segmento 1 y desplazamiento 752. Supóngase que dicho segmento está en memoria principal y comienza en la dirección física 0010000000100000. Entonces la dirección física será 0010000000100000 + 001011110000 = 0010001100010000.

En definitiva, con segmentación simple, un proceso se divide en varios segmentos que no tienen por qué ser del mismo tamaño. Cuando un proceso se introduce en memoria, se cargan todos sus segmentos en regiones de memoria libres y se rellena la tabla de segmentos.

6.3 RESUMEN

Una de las tareas más importantes y complejas de un sistema operativo es la gestión de memoria. La gestión de memoria implica tratar la memoria principal como un recurso que asignar y compartir entre varios procesos activos. Para un uso eficiente del procesador y de los servicios de E/S, resulta interesante mantener en memoria principal tantos procesos como

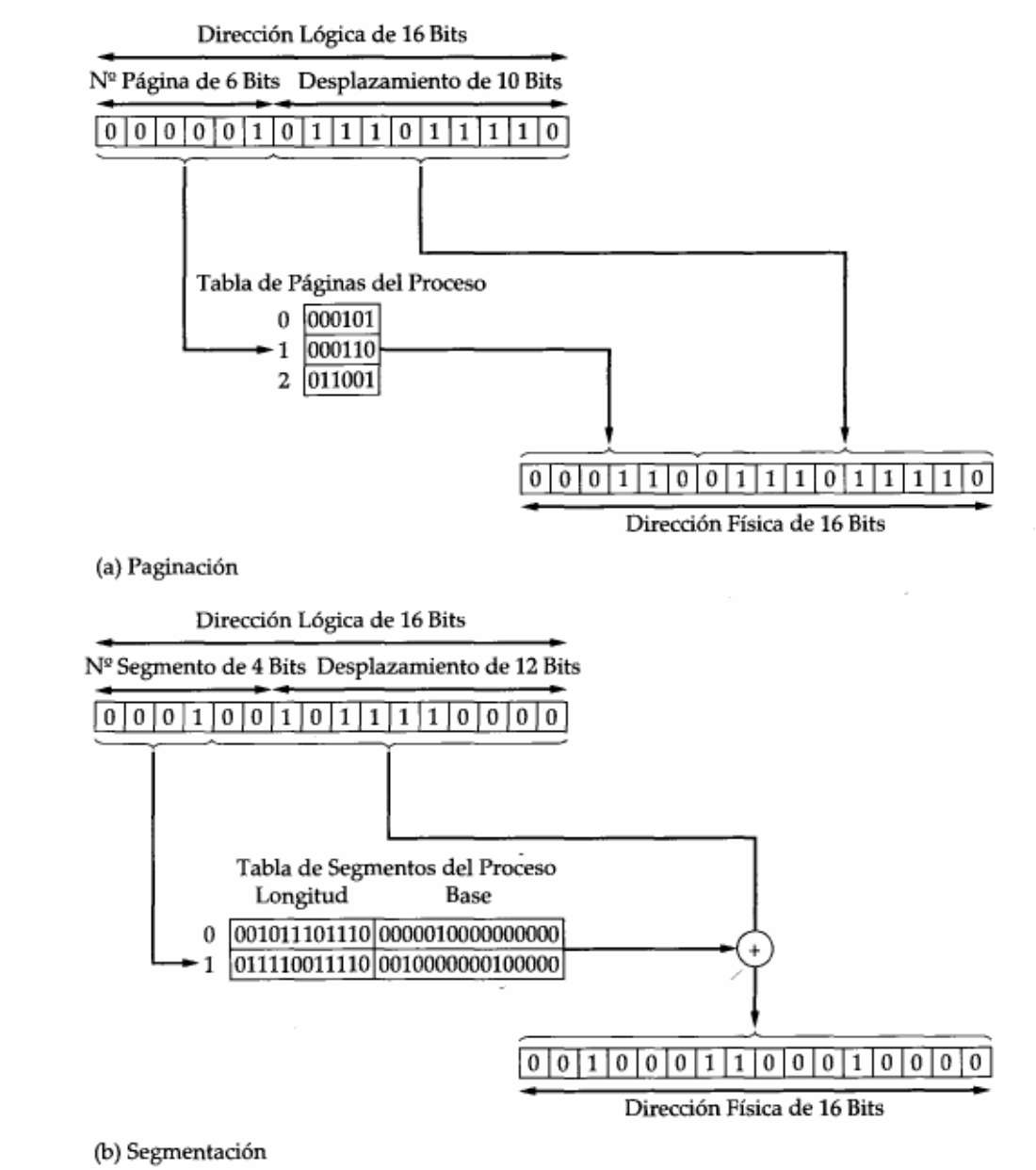


FIGURA 6.10 Ejemplos de traducción de dirección lógica a física

sea posible. Además, es deseable poder liberar a los programadores de las limitaciones de tamaño en el desarrollo de los programas.

Las herramientas básicas de la gestión de memoria son la paginación y la segmentación. En la paginación, cada proceso se divide en páginas de tamaño constante y relativamente pequeño. La segmentación permite el uso de partes de tamaño variable. También es posible combinar la segmentación y la paginación en un único esquema de gestión de memoria.

6.4 LECTURAS RECOMENDADAS

Todos los libros recomendados en la sección 2.6 también abarcan el tema de la gestión de memoria.

Puesto que la partición ha sido sustituida por las técnicas de memoria virtual, la mayoría de los libros ofrecen sólo un estudio superficial. Dos de los tratamientos más completos e interesantes están en [MILE92] y [HORN89]. Una discusión en profundidad de las estrategias de partición se encuentra en [KNUT73].

Los temas de montaje y carga están cubiertos en una gran variedad de libros de desarrollo de programas, arquitectura de computadores y sistemas operativos. [BECK90] ofrece un tratamiento especialmente detallado. La discusión del Apéndice 6A está organizada según las directrices de [SCHN85], que proporciona una introducción básica y clara. [KURZ84] examina el tema desde el punto de vista de las implicaciones para el sistema operativo y del diseño de la gestión de trabajos. [PINK89] proporciona un buen resumen, haciendo énfasis en las etapas que preceden al montaje y la carga en la creación de un módulo objeto. [DAVI87] ofrece una descripción detallada de las funciones de montaje y carga de MVS. El tema del montaje dinámico, con especial referencia al enfoque de Multics, queda cubierto en [BIC88] y [KRAK88].

BECK90 BECK, L. *System Software.* Addison-Wesley, Reading, MA, 1990.

BIC88 BIC, L. y SHAW, A. *The Logical Design of Operating Systems.* 2ª ed. Prentice Hall, Engle-woodCliffs,NJ,1988. DAVI87 DAVIS, W. *Operating Systems: A Systematic View.* 3ª ed. Addison-Wesley, Reading, MA, 1987.

HORN89 Homer, D. *Operating Systems: Concepts and Applications.* Glenview, IL: Scott, Fores-

man & Co., 1989. KNUT73 KNUTH, D. *The Art ofComputer Programming, Volunte 1 fundamental Algorithms.* 2ª ed. Addison-Wesley, Reading, MA, 1973.

KRAK88 KRAKOWIAK, S. y BEESON, D. *Principies of Operating Systems.* MIT Prcss, Cambrige, MA, 1988. /

KURZ84 KURZBAN, S. HEINES, T. y KURZBAN, S. *Operating Systems Principies.* 2ª ed. Van Nostrand Reinhold, Nueva York, 1984.

MILE92 MILENKOVIC, M. *Operating Systems: Concepts and Design.* McGraw-Hill, Nueva York, 1992.

PINK89 PINKERT, J. y WEAR, L. *Operating Systems: Concepts, Policies, and Mechanisms.* Prentice Hall, Englewood Cliffs, NJ, 1989.

SCHN85 SCHNEIDER, G. *The Principies ofComputer Organization.* Wiley, Nueva York, 1985.

6.5 PROBLEMAS

6.1 En la sección 2.3 se enumeran cinco objetivos de la gestión de memoria y, en la sección 6.1, cinco necesidades. Discutir cómo cada lista abarca todos los conceptos tratados en la otra.

6.2 Considérese un esquema de partición dinámica. Demostrar que, por término medio, la memoria contendrá la mitad de agujeros que de segmentos.

6.3 ¿Qué consejos se obtienen del resultado del problema 6.2 con respecto a una implementación eficiente de un algoritmo de ubicación para segmentos contiguos?

6.4 Durante el curso de la ejecución de un programa, el procesador incrementará el contenido del registro de instrucción (contador de pro-

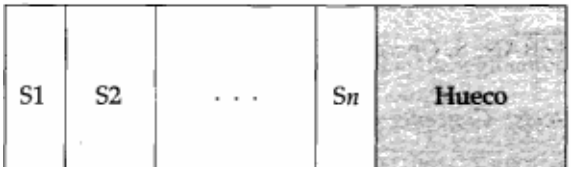
grama) en una palabra después de cada lectura de instrucción, pero cambiará el contenido de dicho registro si encuentra una instrucción de bifurcación o llamada que hace que la ejecución continué en otra parte del programa. Considérese la figura 6.6. Existen dos alternativas en las direcciones de instrucciones:

a) Mantener direcciones relativas en el registro de instrucción y realizar la traducción dinámica de direcciones empleando el registro de instrucción como entrada. Cuando se realiza con éxito una bifurcación o llamada, se carga en el registro de instrucción la dirección relativa generada por dicha bifurcación o llamada.

b) Mantener direcciones absolutas en el registro de instrucción. Cuando se realiza con éxito una bifurcación o llamada, se emplea la traducción dinámica de direcciones, almacenando el resultado en el registro de instrucción.

¿Qué método es preferible?

6.5 Considérese una memoria con los segmentos contiguos SI, S2, ..., *Sn* situados en orden de creación, desde un extremo a otro, como sugiere la figura



siguiente:

Cuando se crea el segmento Sn+1, se sitúa inmediatamente después del segmento Sn, aunque puedan haberse suprimido ya algunos de los segmentos SI, S2, … *Sn.* Cuando la frontera entre los segmentos (en uso o suprimidos) y el hueco alcanza el otro extremo de la memoria, se compactan los segmentos que están en uso.

a) Demostrar que la fracción *F* de tiempo consumida en la compactación obedece a la siguiente desigualdad:

$$F \geq \frac{1-f}{1+kf} \quad \text{siendo} \quad k = \frac{t}{2s} - 1,$$

donde

s = longitud media de un segmento, en palabras

t = tiempo de vida medio de un segmento, en referencias a memoria

f= fracción de memoria que está sin usar, en condiciones de equilibrio

Indicación: Búsquese la velocidad media en la que la frontera cruza la memoria y supóngase que la copia de una única palabra requiere al menos dos referencias a memoria.

b) Encontrar F para/= 0,2, *t* = 1000 y *s* = 50.

APÉNDICE 6A

CARGA Y MONTAJE

El primer paso para la creación de un proceso activo consiste en cargar un programa en memoria principal y crear una imagen del proceso (figura 6.11). La figura 6.12 representa una situación típica para la mayoría de los sistemas. La aplicación está formada por una serie de módulos compilados o ensamblados en forma de código objeto que se montan juntos para resolver las referencias entre los módulos. Al mismo tiempo, se resuelven las referencias a rutinas de biblioteca. Las rutinas de biblioteca pueden estar incorporadas en el programa o ser referenciadas como código compartido que debe suministrar el sistema operativo en el momento de la ejecución. En este apéndice, se resumirán las características clave de los montadores y cargadores. Por claridad de presentación, se comenzará con una descripción de la tarea de carga cuando se dispone de un único módulo de programa y no se necesita montaje.

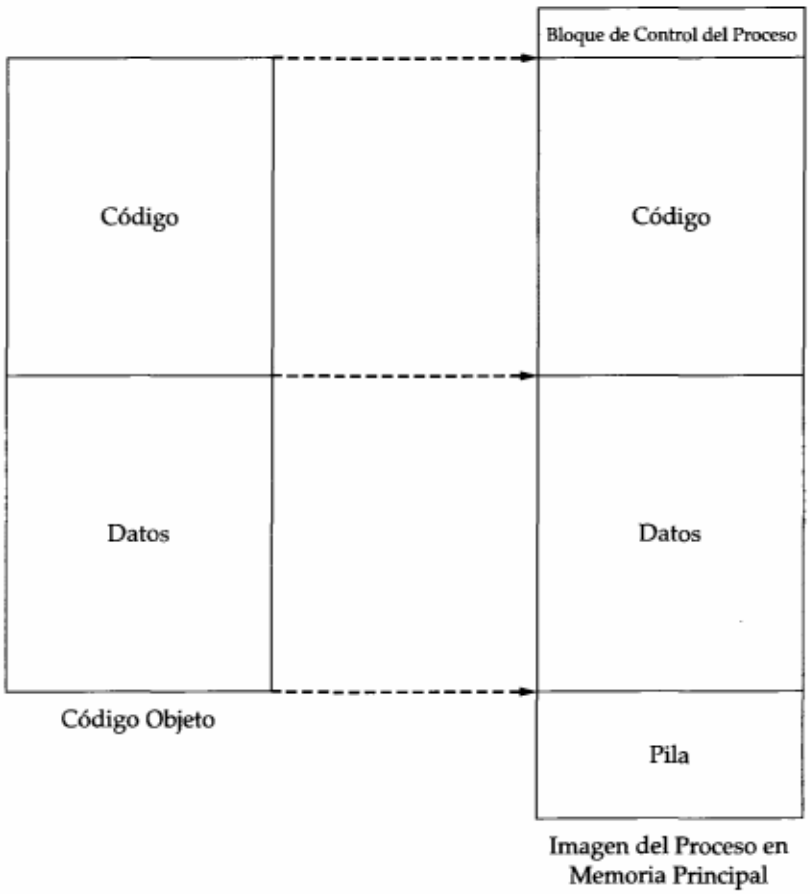


FIGURA 6.11 La función de carga

6A.1 Carga

En la figura 6.12, el cargador sitúa el módulo de carga en la memoria principal, comenzando en la posición *x*. En la carga del programa, se deben satisfacer las necesidades de direccionamiento mostradas en la figura 6.1. En general, se pueden aplicar tres métodos:

- Carga absoluta
- Carga reubicable
- Carga dinámica en tiempo de ejecución

Carga absoluta

La carga absoluta necesita que el módulo de carga ocupe siempre la misma posición de memoria principal. Así pues, todas las referencias del módulo de carga para el cargador deben ser direcciones específicas o absolutas en memoria principal. Por ejemplo, si *x* en la figura 6.12 es la posición 1024, la primera palabra del módulo de carga destinado a esa región de memoria tendrá la dirección 1024.

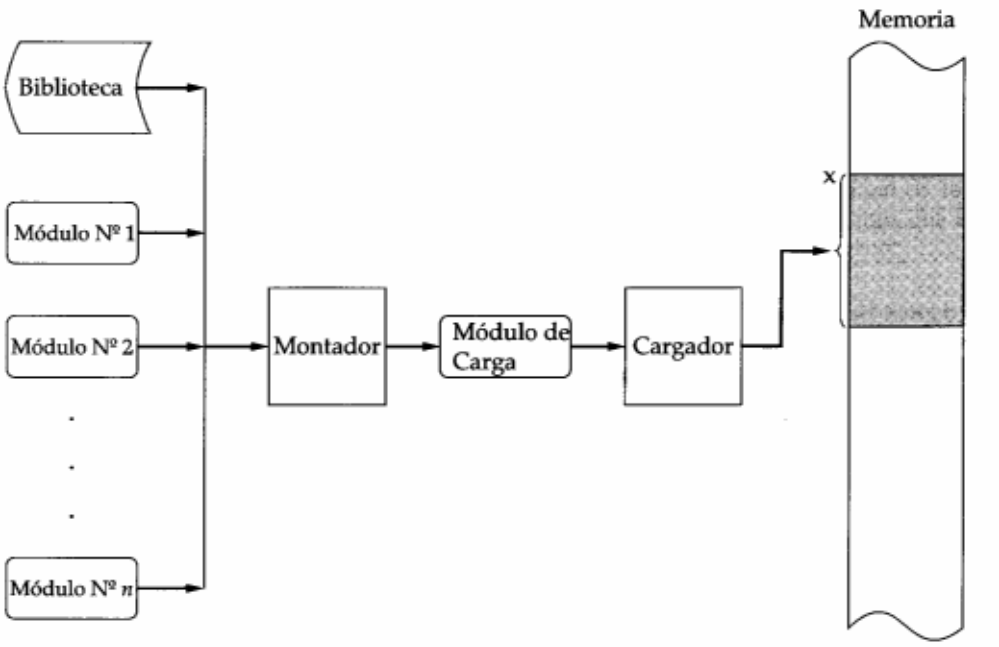


FIGURA 6.12 Proceso de carga

La asignación de direcciones específicas a las referencias a memoria de un programa puede ser realizada tanto por el programador como en tiempo de compilación o ensamblaje (tabla 6.2a). Con el primer método se tienen varias desventajas. En primer lugar, todos los programadores tendrán que conocer la estrategia de asignación deseada para situar los módulos en memoria principal. En segundo lugar, si se hace alguna modificación en el programa que suponga inserciones o borrados en el cuerpo del módulo, tendrán que cambiarse todas las direcciones. Por consiguiente, es preferible permitir que las referencias a memoria dentro de los programas se expresen simbólicamente y que se resuelvan en el momento de la compilación o el ensamblaje. Esto se ilustra en la figura 6.13b. Todas las referencias a una instrucción o elemento de datos se representan inicialmente por un símbolo. Cuando se prepara el módulo para la entrada a un cargador absoluto, el ensamblador o el compilador convertirán todas estas referencias a direcciones específicas (en este ejemplo, para cargar el módulo en la posición de comienzo 1024).

Carga reubicable

La desventaja de asociar las referencias a memoria a direcciones específicas previas a la carga es que el módulo de carga resultante sólo puede situarse en una región de memoria principal. Sin embargo, cuando varios programas comparten la memoria principal, puede no ser conveniente decidir por adelantado en qué región de memoria debe cargarse un módulo en particular. Es mejor tomar esa decisión en el momento de la carga. Así pues, se necesita un módulo de carga que pueda ubicarse en cualquier posición de la memoria principal.

Para satisfacer este nuevo requisito, el ensamblador o el compilador no generará direcciones reales de memoria principal (direcciones absolutas) sino direcciones relativas a algún

TABLA 6.2 Enlace de Direcciones

(a) Cargador

Tiempo de Enlace	Función
Tiempo de programación	El programador especifica directamente todas las direcciones físicas reales en el propio programa.
Tiempo de compilación o ensamblaje	El programa contiene referencias simbólicas a direcciones y el compilador o el ensamblador las convierten en direcciones físicas reales.
Tiempo de carga	El compilador o ensamblador genera direcciones relativas. El cargador traduce éstas a direcciones absolutas en el instante de la carga del programa.
Tiempo de ejecución	El programa cargado conserva direcciones relativas. El hardware del procesador las convierte dinámicamente en direcciones absolutas.

(b) Montador

Tiempo de Montaje	Función
Tiempo de programación	No se permiten referencias a programas o datos externos. El programador debe situar dentro del programa el código fuente de todos los subprogramas que sean referenciados.
Tiempo de compilación o ensamblaje	El ensamblador debe ir a buscar el código fuente de cada subrutina que sea referenciada y ensamblarlas como una unidad.
Creación del módulo de carga	Todos los módulos objeto se han ensamblado usando direcciones relativas. Estos módulos se montan juntos y todas las referencias se declaran de nuevo relativas al origen del módulo de carga final.
Tiempo de carga	Las referencias externas no se resuelven hasta que el módulo de carga sea ubicado en memoria principal. En ese momento, los módulos montados y referenciados dinámicamente se añaden al módulo de carga, enviando el paquete completo a la memoria principal o virtual.
Tiempo de ejecución	Las referencias externas no se resuelven hasta que el procesador ejecute las llamadas externas. En ese momento, se interrumpe el proceso y se monta el programa llamador con el módulo que se necesita.

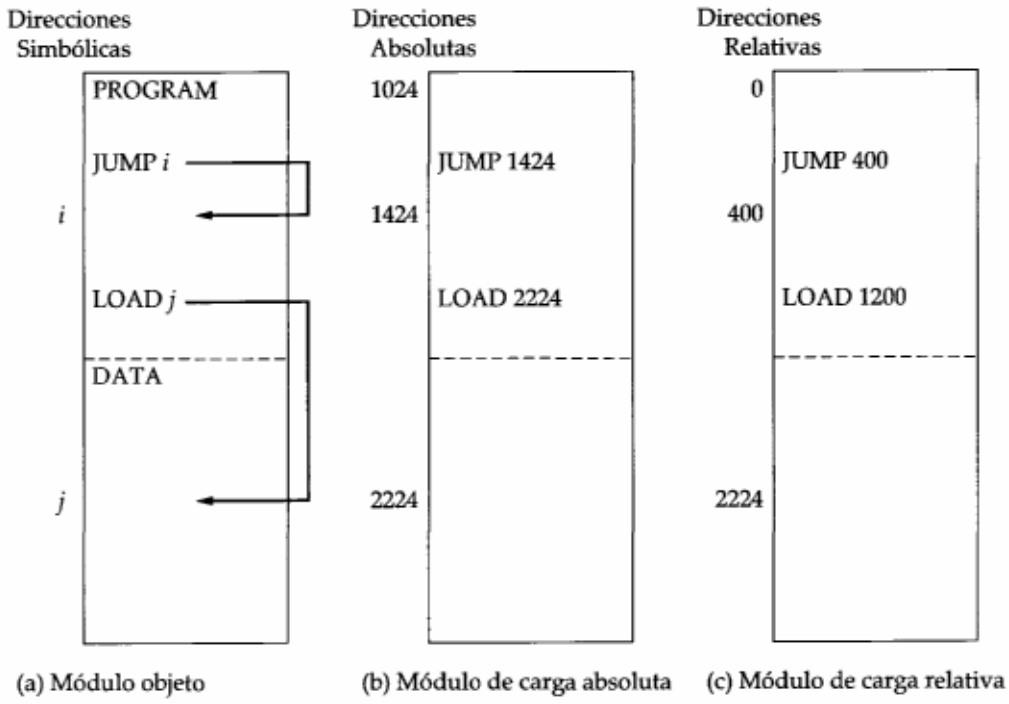


FIGURA 6.13 Módulos de carga absolutos y reubicables

punto conocido, tal como el comienzo del programa. Esta técnica se ilustra en la figura 6.13c. Al comienzo del módulo de carga se le asigna la dirección relativa O y todas las demás referencias dentro del módulo se expresan en relación al comienzo del módulo.

Con todas las referencias a memoria expresadas de forma relativa, situar los módulos en la posición deseada se convierte en una tarea sencilla para el cargador. Si el módulo va a ser cargado comenzando por la posición *x,* el cargador simplemente sumará *x* a cada referencia a memoria a medida que cargue el módulo en memoria. Para ayudar en esta tarea, el módulo de carga debe incluir información que indique al cargador dónde están las referencias a direcciones y cómo se interpretan (generalmente, de forma relativa al comienzo del programa, pero también es posible que sean relativas a algún otro punto del programa, como la posición actual). El compilador o el ensamblador prepara este conjunto de información que se conoce normalmente como *diccionario de reubicación.*

Carga dinámica en tiempo de ejecución

Los cargadores con reubicación son habituales y ofrecen ventajas obvias en relación con los cargadores absolutos. Sin embargo, en un entorno multiprogramado, incluso sin memoria virtual, el esquema de carga reubicable resulta inadecuado. Se ha hecho referencia a la necesidad de cargar y descargar las imágenes de procesos-de memoria principal para maximizar la utilización del procesador. Para maximizar el uso de la memoria principal, sería conveniente volver a cargar una imagen de un proceso en posiciones diferentes para diferentes instantes de tiempo. Así pues, un programa cargado puede ser expulsado a

disco y vuelto a cargar en una posición distinta. Este procedimiento resultaría imposible si las referencias a memoria hubieran estado asociadas a direcciones absolutas en el momento inicial de carga.

Una alternativa consiste en aplazar el cálculo de direcciones absolutas hasta que realmente se necesitan durante la ejecución. Con este propósito, el módulo de carga se trae a memoria con todas las referencias de forma relativa (figura 6.13c). La dirección absoluta no se calcula hasta que se ejecuta una instrucción. Para asegurar que esta función no degrada el rendimiento, debe realizarse por medio de un hardware especial del procesador, en vez de por software. Este hardware se describe en la sección 6.2.

El cálculo de direcciones dinámico proporciona una completa flexibilidad. Un programa puede cargarse en cualquier región de memoria principal. Más tarde, la ejecución del programa puede interrumpirse y el programa ser descargado de memoria principal para ser posteriormente cargado en una posición diferente.

6A.2 Montaje

La función de un montador consiste en tomar como entrada una colección de módulos objeto y generar un módulo de carga que conste de un conjunto integrado de módulos de programa y de datos para el cargador. En cada módulo objeto pueden haber referencias a direcciones situadas en otros módulos. Cada una de estas referencias puede expresarse sólo simbólicamente en un módulo objeto no montado. El montador crea un único módulo de carga que es la concatenación de todos los módulos objeto. Cada referencia interna de un módulo debe cambiarse de dirección simbólica a una referencia a una posición dentro del módulo de carga total. Por ejemplo, el módulo A de la figura 6.14a contiene una llamada a un procedimiento del módulo B. Cuando se combinan estos módulos en el módulo de carga, esta referencia simbólica al módulo B se cambia por una referencia específica a la posición del punto de entrada de B en el módulo de carga.

Editor de montaje

La esencia del montaje de direcciones dependerá del tipo de módulo de carga a crear y de cuándo se produzca el montaje (tabla 6.2b). Si, como suele ser el caso, se desea un módulo de carga reubicable, el montaje se realiza generalmente de la siguiente forma. Cada módulo objeto compilado o ensamblado se crea con referencias relativas al comienzo del módulo. Todos estos módulos se unen en un único módulo de carga reubicable, junto con todas las referencias relativas al origen del módulo de carga. Este módulo puede usarse como entrada para una carga reubicable o para una carga dinámica durante la ejecución.

Los montadores que generan módulos de carga reubicables se conocen a menudo como *editores de montaje.* La figura 6.14 muestra el funcionamiento de un editor de montaje.

Montador dinámico

Así como en la carga, es posible aplazar algunas funciones de montaje. El término *montaje dinámico* se emplea para referirse a la práctica de retrasar el montaje de algunos módulos externos hasta después de que el módulo de carga se haya creado. Así pues, el módulo de carga contiene referencias no resueltas a otros programas. Estas referencias pueden resolverse tanto en la carga como en la ejecución.

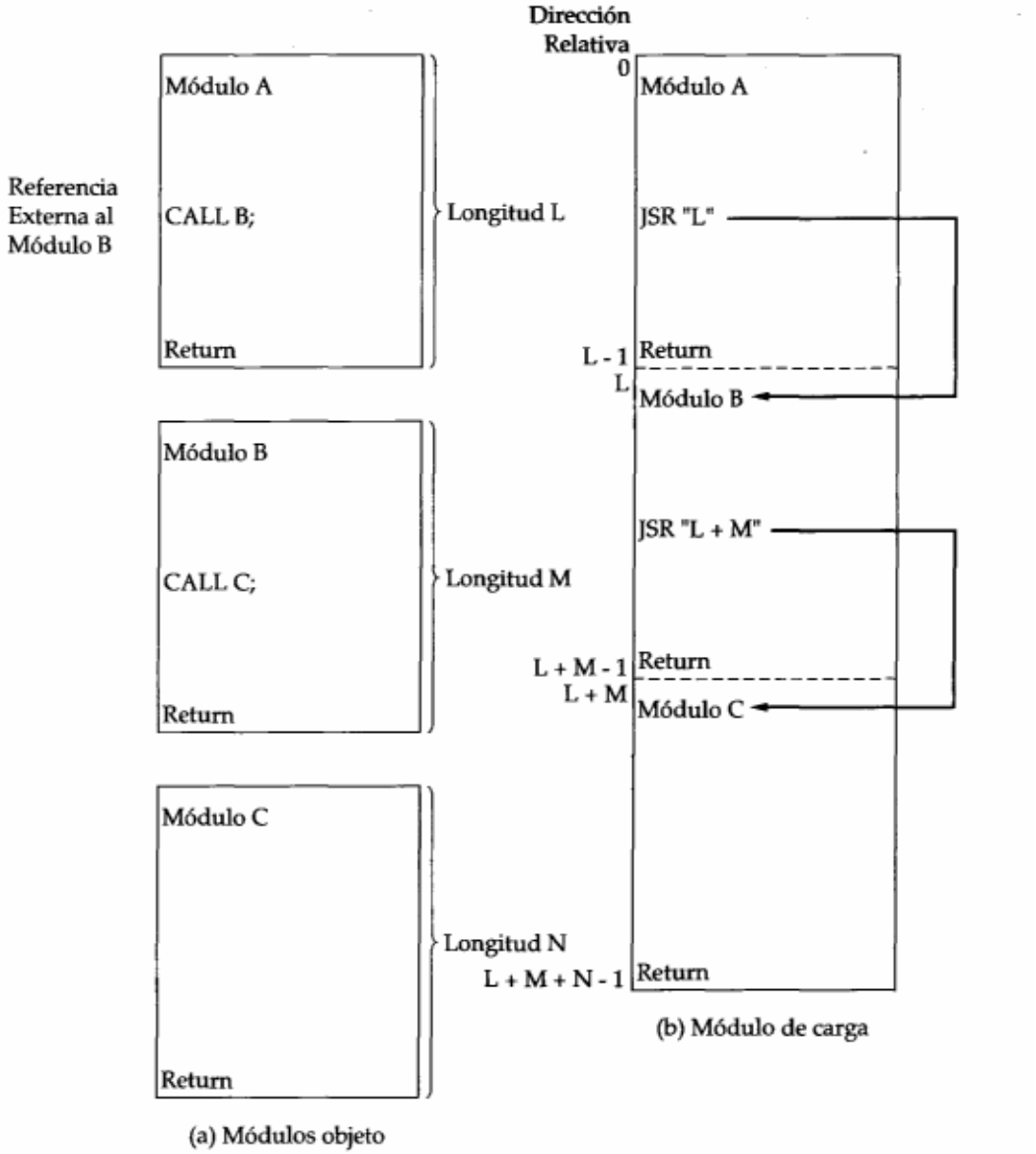


FIGURA 6.14 Funcionamiento del montaje

En el **montaje dinámico en tiempo de carga** se suceden las siguientes etapas. El módulo de carga (módulo de aplicación) se trae a memoria principal. Cualquier referencia a un módulo externo (módulo destino) hace que el cargador busque el módulo destino, lo cargue y modifique las referencias a direcciones relativas de memoria desde el comienzo del módulo de aplicación. Existen varias ventajas en este enfoque sobre el que podría llamarse carga estática, como son las siguientes:

- Resulta más fácil incorporar cambios o actualizar versiones del módulo destino, lo que puede constituir una utilidad del sistema operativo o alguna otra rutina de propósito gene-

ral. En el montaje estático, cualquier cambio en el módulo soporte requeriría volver a montar el módulo de aplicación por completo. Esto no sólo es ineficiente, sino que puede ser imposible en determinadas circunstancias. Por ejemplo, en el campo de los computadores personales, la mayoría del software comercial se entrega en forma de módulo de carga; no se entregan las versiones fuente y objeto.

• Tener el código destino en un fichero de montaje dinámico prepara el terreno para la compartición automática de código. El sistema operativo puede darse cuenta que más de una aplicación está empleando el mismo código destino, puesto que habrá cargado y montado dicho código. Esta información puede usarse para cargar una única copia del código destino y montarla en ambas aplicaciones, en vez de tener que cargar una copia para cada aplicación.

• Resulta más fácil para los productores independientes de software ampliar la funcionalidad de un sistema operativo muy empleado, como puede ser OS/2. Un productor de software puede proponer una nueva función que sea útil para varias aplicaciones y empaquetarla como un módulo de montaje dinámico.

En el **montaje dinámico en tiempo de ejecución,** parte del montaje se pospone hasta el momento de la ejecución. Las referencias externas a los módulos destino permanecen en el programa cargado.

Cuando se realiza una llamada a un módulo ausente, el sistema operativo localiza el módulo, lo carga y lo monta en el módulo llamador.

Se ha visto anteriormente que la carga dinámica permite desplazar un módulo de carga completamente; sin embargo, la estructura del módulo es estática y permanece sin cambios a lo largo de la ejecución del proceso y de una ejecución a la siguiente. No obstante, en algunos casos, no es posible determinar antes de la ejecución qué módulos objeto harán falta. Esta situación es la normal en las aplicaciones de proceso de transacciones, como el sistema de reservas de una compañía aérea o una aplicación bancaria. La naturaleza de la transacción dictamina qué módulos de programa se necesitan y estos se cargan de la forma apropiada y se montan con el programa principal. La ventaja de emplear un montador dinámico es que no hace falta reservar memoria para las unidades de programa a menos que se haga referencia a las mismas. Esta capacidad se emplea como soporte de los sistemas de segmentación.

Es posible un refinamiento adicional: Una aplicación no tiene por qué saber los nombres de todos los módulos o puntos de entrada a los que puede llamar. Por ejemplo, un programa de gráficos puede estar escrito para funcionar con una serie de trazadores gráficos, cada uno de los cuales es conducido por un paquete manejador distinto. La aplicación puede conocer el nombre del trazador instalado actualmente en el sistema a través de otro proceso o buscándolo en un archivo de configuración, lo que permite al usuario de la aplicación instalar un nuevo trazador que ni siquiera existía cuando se escribió la aplicación.

CAPÍTULO 7

Memoria virtual

En el capítulo 6 se realizó una introducción a los conceptos de paginación y segmentación y se analizaron sus deficiencias. Ahora se está en condiciones de trasladar la discusión a la memoria virtual. Este punto es complicado de analizar debido al hecho de que la gestión de memoria establece una interrelación estrecha y compleja con el hardware del procesador y el software del sistema operativo. Por consiguiente, el estudio se centrará en primer lugar en los aspectos de hardware de la memoria virtual, considerando el uso de paginación, segmentación y la combinación de ambas. Así se consigue estar en condiciones de estudiar los temas involucrados en el diseño de un servicio de memoria virtual para un sistema operativo. Esta discusión, como es habitual, vendrá seguida de un estudio de tres sistemas de ejemplo.

7.1 ESTRUCTURAS DE HARDWARE Y DE CONTROL

Cuando se compararon la paginación y segmentación simple por un lado, con la partición estática y dinámica por otro, se pusieron las bases de un avance fundamental en la gestión de memoria. Las claves de este avance son dos características de la paginación y la segmentación:

1. Todas las referencias a memoria dentro de un proceso son direcciones lógicas que se traducen dinámicamente a direcciones físicas durante la ejecución. Esto quiere decir que un proceso puede cargarse y descargarse de memoria principal de forma que ocupe regiones diferentes en instantes diferentes a lo largo de su ejecución.
2. Un proceso puede dividirse en varias partes (páginas o segmentos) y no es necesario que estas partes se encuentren contiguas en memoria principal durante la ejecución. Esto es posible por la combinación de la traducción dinámica de direcciones en tiempo de ejecución y el uso de una tabla de páginas o de segmentos.

Volviendo sobre el avance comentado, si *estas dos características están presentes, no será necesario que todas las páginas o todos los segmentos de un proceso estén en memoria durante la ejecución.* Si tanto la parte (página o segmento) que contiene la siguiente instrucción a leer como la parte que contiene los próximos datos a acceder están en memoria principal, la ejecución podrá continuar al menos por un tiempo.

A continuación, se verá cómo conseguirlo. Por ahora, se hablará en términos generales y se empleará el *término fragmento* para referirse tanto a páginas como a segmentos, dependiendo de si se emplea paginación o segmentación. Supóngase que se trae un proceso a memoria en un instante dado. El sistema operativo comienza trayendo sólo unos pocos fragmentos, incluido el fragmento que contiene el comienzo del programa. Se llamará **conjunto residente** del proceso a la parte de un proceso que está realmente en memoria principal. Cuando el proceso se ejecuta, todo irá sobre ruedas mientras todas las referencias a memoria estén en posiciones que pertenezcan al conjunto residente. A través de la tabla de páginas o de segmentos, el procesador siempre es capaz de determinar si esto es así. Si el procesador encuentra una dirección lógica que no está en memoria principal, genera una interrupción que indica un fallo de acceso a memoria. El sistema operativo pasa el proceso interrumpido al estado bloqueado y toma el control. Para que la ejecución de este proceso siga más tarde, el sistema operativo necesita traer a memoria principal el fragmento del proceso que contiene la dirección lógica que provocó el fallo de acceso. Para ello, el sistema operativo emite una solicitud de Lectura de E/S a disco. Después de haber emitido la solicitud de E/S, el sistema operativo puede despachar otro proceso para que se ejecute mientras se realiza la operación de E/S. Una vez que el fragmento deseado se ha traído a memoria principal y se ha emitido la interrupción de E/S, se devuelve el control al sistema operativo, que coloca el proceso afectado en el estado de Listo.

Ahora bien, esto puede llevar a cuestionarse la eficiencia de esta operación, en la cual un proceso puede estar siendo ejecutado y ser interrumpido sin otra razón que un fallo en la carga de todos los fragmentos necesarios para el proceso. Por ahora, se aplazará la discusión de esta cuestión con la seguridad de que es posible alcanzar esta eficiencia. En su lugar, se meditarán las implicaciones de esta nueva estrategia, que son dos, siendo la segunda más sorprendente que la primera. Ambas conducen a mejoras en la utilidad del sistema:

1. Se pueden conservar más procesos en memoria principal. Puesto que se van a cargar sólo algunos fragmentos de un proceso particular, habrá sitio para más procesos. Esto conduce a una utilización más eficiente del procesador, puesto que es más probable que, por lo menos, uno de los numerosos procesos esté en estado Listo en un instante determinado.
2. Es posible que un proceso sea más grande que toda la memoria principal. Se elimina así una de las limitaciones más notorias de la programación. Sin el esquema que se ha expuesto, un programador debe ser consciente de cuánta memoria tiene disponible. Si el programa que está escribiendo es demasiado grande, el programador debe idear formas de estructurar el programa en fragmentos que puedan cargarse de forma separada con algún tipo de estrategia de superposición. Con una memoria virtual basada en paginación o segmentación, este trabajo queda para el sistema operativo y el hardware. En lo que atañe al programador, se las arregla con una memoria enorme, dependiendo del tamaño de almacenamiento en disco. El sistema operativo cargará automáticamente en memoria principal los fragmentos de un proceso cuando los necesita.

Como los procesos se ejecutan sólo en memoria principal, a esta memoria se le llama **memoria real.** Pero un programador o usuario percibe en potencia una memoria mucho mayor, que está situada en el disco. Esta última se conoce por **memoria virtual.**

La memoria virtual permite una multiprogramación muy efectiva y releva al usuario de las rígidas e innecesarias restricciones de la memoria principal. La tabla 7.1 resume las características de la paginación y la segmentación, con y sin memoria virtual.

Memoria Virtual y Cercanía de Referencias

Las ventajas de la memoria virtual son importantes. La cuestión es: ¿Funcionará este esquema? En un primer momento, se produjo un gran debate sobre este punto, pero la experiencia con numerosos sistemas operativos demuestra más allá de cualquier duda que la memoria virtual funciona. Por consiguiente, ha llegado a ser un componente esencial de la mayoría de los sistemas operativos actuales.

Para comprender cuál el es elemento clave y por qué la memoria virtual generó tanto debate, se va a considerar de nuevo la labor del sistema operativo con respecto a la memoria virtual. Considérese un proceso grande formado por un programa largo y un conjunto de series de datos. Durante un corto periodo, la ejecución puede estar reducida a una pequeña sección del programa (por ejemplo, una subrutina) y acceder sólo a una o dos series de datos. Si esto es así, sería un claro desperdicio cargar docenas de fragmentos para el proceso cuando se van a usar sólo unos pocos antes de que pase a estar suspendido o se descargue. Se puede aprovechar mejor la memoria cargando tan sólo unos pocos fragmentos. Además, si el programa se bifurca a una instrucción o hace referencia a datos de un fragmento que no está en memoria, se producirá un fallo de página. Este fallo le dice al sistema operativo que traiga el fragmento deseado.

De este modo, en un instante dado, en memoria sólo se tienen unos pocos fragmentos de un proceso dado y, por tanto, se pueden mantener más procesos en memoria. Es más, se ahorra tiempo, porque los fragmentos que no se usan no se cargan ni se descargan de memoria. Sin embargo, el sistema operativo debe saber cómo gestionar este esquema. En un estado estable, prácticamente toda la memoria principal estará ocupada con fragmentos de procesos, por lo que el procesador y el sistema operativo tendrán acceso directo a la mayor cantidad de procesos posible. Así pues, cuando el sistema operativo traiga a memoria un fragmento, deberá expulsar otro. Si expulsa un fragmento justo antes de ser usado, tendrá que traer de nuevo el fragmento de manera casi inmediata. Demasiados intercambios de fragmentos conducen a lo que se conoce como **hiperpaginación** *(thrashing):* El procesador consume más tiempo intercambiando fragmentos que ejecutando instrucciones de usuario. Las formas de evitar la hiperpaginación fueron una de las áreas de investigación más importantes de los 70 y llevaron a un buen número de algoritmos complejos pero efectivos. En esencia, el sistema operativo intenta adivinar, en función de la historia reciente, qué fragmentos se usarán con menor probabilidad en un futuro próximo.

Los argumentos anteriores se basan en el principio de cercanía, que se introdujo en el capítulo 1 (ver el Apéndice 1A). Resumiendo, el principio de cercanía afirma que las referencias a los datos y al programa dentro de un proceso tienden a agruparse. Por tanto, es válida la suposición de que, durante cortos periodos, se necesitarán sólo unos pocos fragmentos de un proceso. Además, sería posible hacer predicciones inteligentes sobre qué fragmentos de un proceso se necesitarán en un futuro cercano y así evitar la hiperpaginación.

TABLA 7.1 Características de la Paginación y la Segmentación

Paginación Simple	Segmentación Simple	Memoria Virtual Paginada	Memoria Virtual Segmentada
La memoria principal está dividida en trozos pequeños de tamaño fijo llamados marcos. El compilador o el sistema gestor de memoria dividen los programas en páginas.	La memoria principal no está dividida. El programador especifica al compilador los segmentos del programa (es decir, el programador toma la decisión).	La memoria principal está dividida en trozos pequeños de tamaño fijo llamados marcos. El compilador o el sistema gestor de memoria dividen los programas en páginas.	La memoria principal no está dividida. El programador especifica al compilador los segmentos del programa (es decir, el programador toma la decisión).
Fragmentación interna en los marcos.	No hay fragmentación interna.	Fragmentación interna en los marcos.	No hay fragmentación interna.
No hay fragmentación externa.	Fragmentación externa.	No hay fragmentación externa.	Fragmentación externa.
El sistema operativo debe mantener una tabla de páginas para cada proceso, indicando en qué marco está cada página.	El sistema operativo debe mantener una tabla de segmentos para cada proceso, indicando la dirección de carga y la longitud de cada segmento.	El sistema operativo debe mantener una tabla de páginas para cada proceso, indicando en qué marco está cada página.	El sistema operativo debe mantener una tabla de segmentos para cada proceso, indicando la dirección de carga y la longitud de cada segmento.
El sistema operativo debe mantener una lista de marcos libres.	El sistema operativo debe mantener una lista de huecos libres en memoria principal.	El sistema operativo debe mantener una lista de marcos libres.	El sistema operativo debe mantener una lista de huecos libres en memoria principal.
El procesador emplea número de página y desplazamiento para calcular las direcciones absolutas.	El procesador emplea número de segmento y desplazamiento para calcular las direcciones absolutas.	El procesador emplea número de página y desplazamiento para calcular las direcciones absolutas.	El procesador emplea número de segmento y desplazamiento para calcular las direcciones absolutas.
Todas las páginas de un proceso deben estar en memoria principal para que el proceso ejecute, a menos que se use superposición.	Todos los segmentos de un proceso deben estar en memoria principal para que el proceso ejecute, a menos que se use superposición.	Para que el proceso ejecute, no hace falta que todas sus páginas estén en marcos de memoria principal. Las páginas se leerán cuando se necesiten. La carga de una página en memoria principal puede exigir descargar otra al disco.	Para que el proceso ejecute, no hace falta que todos sus segmentos estén en memoria principal. Los segmentos se leerán cuando se necesiten. La carga de un segmento en memoria principal puede requerir descargar uno o más segmentos al disco.

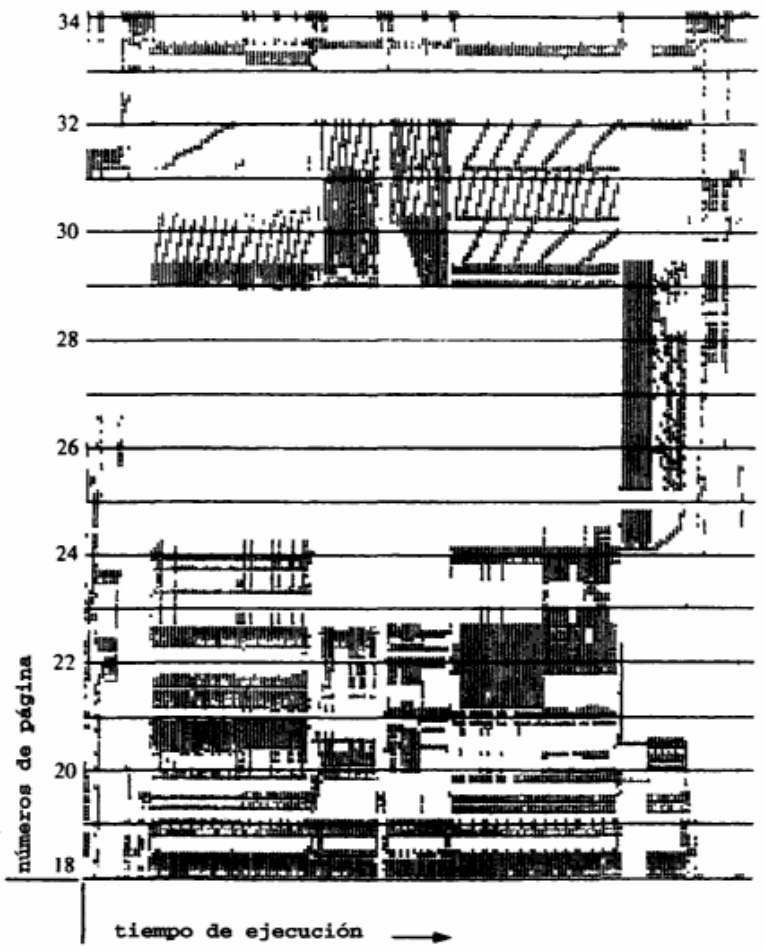


FIGURA 7.1 Comportamiento de la paginación

Una manera de confirmar el principio de cercanía es considerar el rendimiento de un proceso en un entorno de memoria virtual. En la figura 7.1 aparece un conocido diagrama que ilustra de manera espectacular el principio de cercanía [HATF72]. Nótese que, durante el tiempo de vida del proceso, las referencias se reducen a un subconjunto de páginas.

Así pues, se puede comprobar que el principio de cercanía sugiere que los esquemas de memoria virtual pueden funcionar. Para que la memoria virtual sea práctica y efectiva, se necesitan dos ingredientes. Primero, debe haber un soporte de hardware y, en segundo lugar, el sistema operativo debe incluir un software para gestionar el movimiento de páginas y/o segmentos entre memoria secundaria y memoria principal. En esta sección se examinan los aspectos de hardware, al tiempo que se consideran las estructuras de control necesarias que crea y mantiene el sistema operativo, pero usadas por el hardware de gestión de memoria. Los elementos de software se examinan en la siguiente sección.

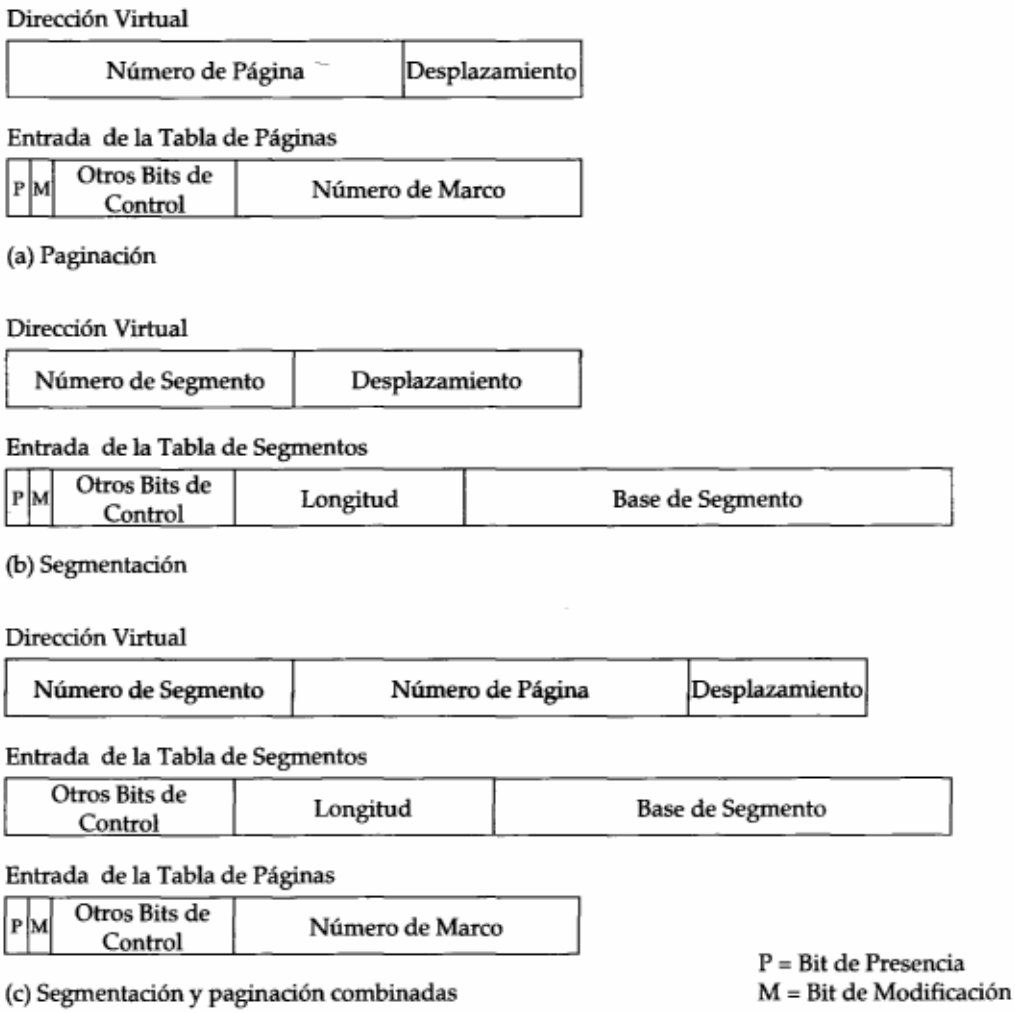


FIGURA 7.2 Formatos típicos de gestión de memoria

Paginación

El término *memoria virtual* se asocia normalmente con sistemas que emplean paginación, aunque también se puede usar memoria virtual basada en la segmentación, que se tratará después. El uso de la paginación en la memoria virtual fue presentado por primera vez en el computador Atlas [KILB62] y pronto alcanzó un uso comercial muy extendido.

En el estudio de la paginación simple se indicó que cada proceso tiene su propia tabla de páginas y que, cuando carga todas sus páginas en memoria principal, se crea y carga en memoria principal una tabla de páginas. Cada entrada de la tabla de páginas contiene el número de marco de la página correspondiente en memoria principal. Cuando se considera un esquema de memoria virtual basado en la paginación se necesita la misma estructura, una tabla de páginas. Nuevamente, es normal asociar una única tabla de páginas con cada proceso. En este caso,

sin embargo, las entradas de la tabla de páginas pasan a ser más complejas (figura 7.2a). Puesto que sólo algunas de las páginas de un proceso pueden estar en memoria principal, se necesita un bit en cada entrada de la tabla para indicar si la página correspondiente está presente (P) en memoria principal o no lo está. Si el bit indica que la página está en memoria, la entrada incluye también el número de marco para esa página.

Otro bit de control necesario en la entrada de la tabla de páginas es el bit de modificación (M), para indicar si el contenido de la página correspondiente se ha alterado desde que la página se cargó en memoria principal. Si no ha habido cambios, no es necesario escribir la página cuando sea sustituida en el marco que ocupa actualmente. Puede haber también otros bits de control. Por ejemplo, si la protección o la compartición se gestiona a nivel de página, se necesitarán más bits con tal propósito. Más adelante se verán varios ejemplos de entradas de tabla de páginas.

Estructura de la Tabla de Páginas

Así pues, el mecanismo básico de lectura de una palabra de memoria supone la traducción por medio de la tabla de páginas de una dirección virtual o lógica, formada por un número de página y un desplazamiento, a una dirección física que está formada por un número de marco y un desplazamiento. Puesto que la tabla de páginas es de longitud variable, en función del tamaño del proceso, no es posible suponer que quepa en los registros. En su lugar, debe estar en memoria principal para ser accesible. La figura 7.3 sugiere una implementación en hardware de este esquema. Cuando se está ejecutando un proceso en parti-

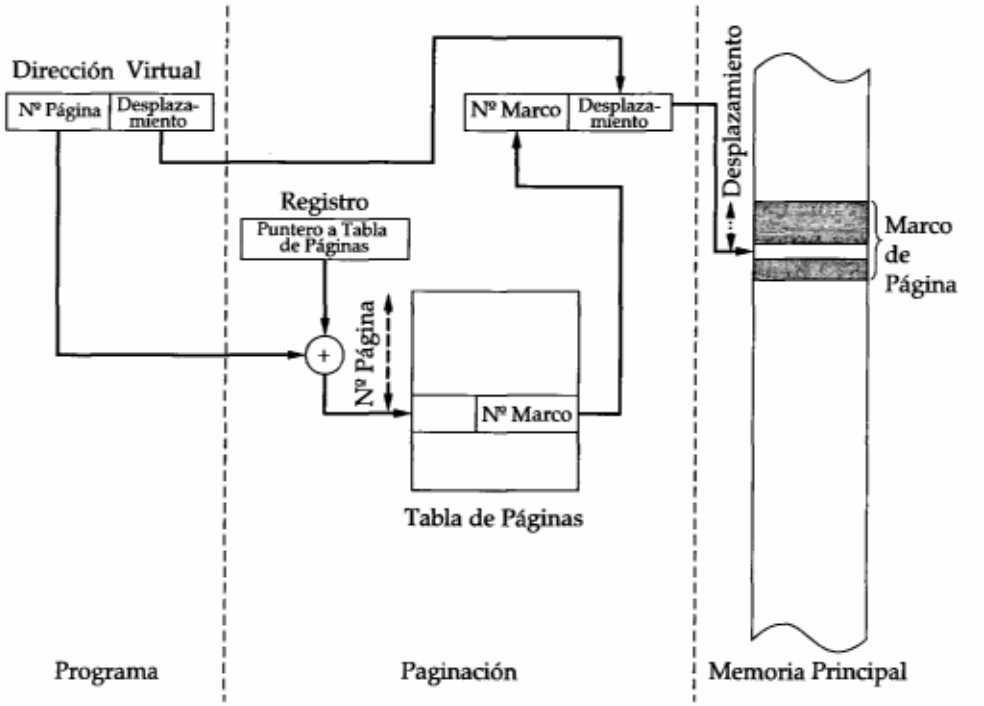


FIGURA 7.3 Traducción de direcciones en un sistema de paginación

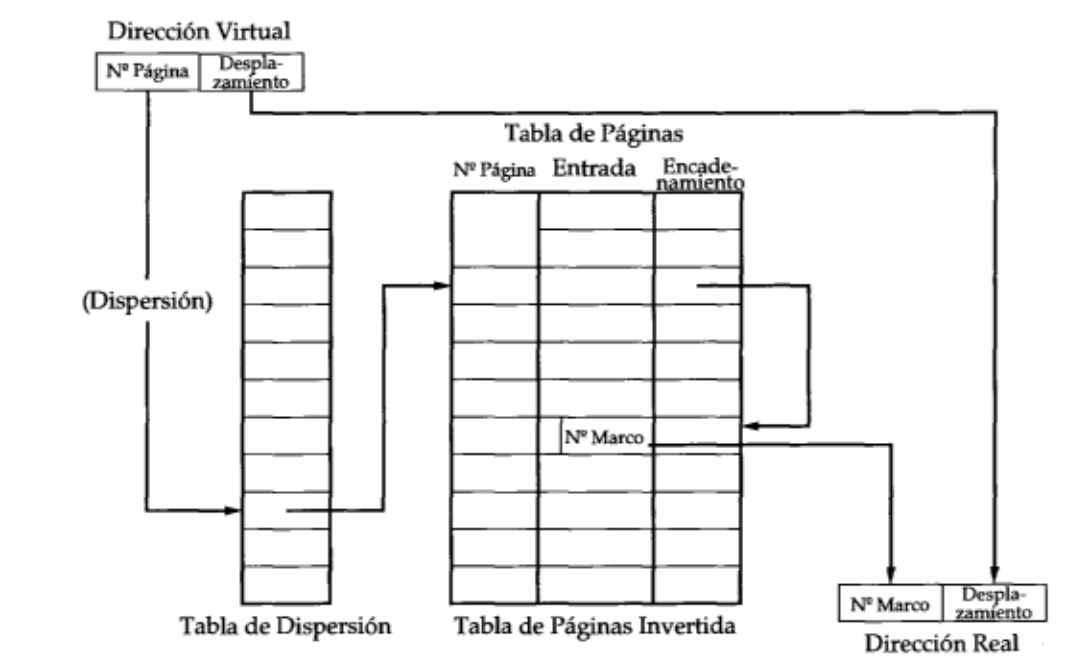


FIGURA 7.4 Estructura de la tabla de páginas invertida

cular, la dirección de comienzo de la tabla de páginas para este proceso se mantiene en un registro. El número de página de la dirección virtual se emplea como índice en esta tabla para buscar el número de marco correspondiente. Este se combina con la parte de desplazamiento de la dirección virtual para generar la dirección real deseada.

Considérese el número de entradas necesarias para la tabla de páginas. En la mayoría de los sistemas hay una tabla de páginas por proceso, pero cada proceso puede ocupar una cantidad enorme de memoria virtual. Por ejemplo, en la arquitectura del VAX, cada proceso puede necesitar una memoria virtual de 2^{31} = 2GB. Usando páginas de 2^9 =512 bytes, se necesitan al menos 2^{22} entradas en la tabla de páginas *por proceso.* Evidentemente, la cantidad de memoria dedicada sólo a la tabla de páginas puede ser inaceptable. Para solucionar este problema, la mayoría de los esquemas de memoria virtual almacenan las tablas de páginas en memoria virtual en vez de en memoria real. Esto significa que estas tablas de páginas están también sujetas a paginación, de la misma forma que las otras páginas. Cuando un proceso está ejecutando, al menos una parte de su tabla de páginas debe estar en memoria principal, incluyendo la entrada de la tabla de páginas para la página actualmente en ejecución. Algunos procesadores usan un esquema a dos niveles para organizar grandes tablas de páginas. En este esquema, hay un directorio de páginas en el que cada entrada señala a una tabla de páginas. Así pues, si la longitud del directorio de páginas es *X* y la longitud máxima de una tabla de páginas es *Y,* un proceso puede estar formado por hasta *X* x *Y* páginas. Normalmente, la longitud máxima de una tabla de páginas está limitada a una página. Se verá un ejemplo de esta solución a dos niveles cuando se estudie el Intel 80386 en este mismo capítulo.

Un enfoque alternativo al uso de tablas de páginas a uno o dos niveles consiste en el uso de una estructura de **tabla de páginas invertida** (figura 7.4). Este método se emplea en el AS/400 de IBM [CORA88, SCHL89] y en las estaciones de trabajo RISC Sistema/600 de IBM [CHAN88, CHAN90a, WATE86]. Esta técnica también la utiliza una implementación del sistema operativo Mach en el RT-PC [RASH88].

Con este método, la parte del número de página en una dirección virtual se contrasta en una tabla de dispersión por medio de una función de dispersión simple'. La tabla de dispersión contiene un puntero a la tabla de páginas invertida, que contiene a su vez las entradas de la tabla de páginas. Con esta estructura, hay una entrada en la tabla de dispersión y la tabla de páginas invertida por cada página de memoria real en lugar de una por cada página virtual. Así pues, se necesita una parte fija de la memoria real para las tablas, sin reparar en el número de procesos o de páginas virtuales soportados. Puesto que más de una dirección de memoria pueden corresponderse con la misma entrada de la tabla de dispersión, para gestionar las colisiones se emplea una técnica de encadenamiento. La técnica de dispersión genera normalmente cadenas cortas, de dos a tres entradas cada una.

Buffer de Traducción Adelantada

En principio, cada referencia a memoria virtual puede generar dos accesos a memoria: uno para obtener la entrada de la tabla de páginas correspondiente y otro para obtener el dato deseado. Así pues, un esquema sencillo de memoria virtual podría tener el efecto de doblar el tiempo de acceso a memoria. Para solucionar este problema, la mayoría de los esquemas de memoria virtual hacen uso de una cache especial para las entradas de la tabla de páginas, llamada generalmente **buffer de traducción adelantada** (TLB, *Translation Lookaside Buffer).* Esta cache funciona del mismo modo que una memoria cache (ver capítulo 1) y contiene aquellas entradas de la tabla de páginas usadas hace menos tiempo. La organización del hardware de paginación resultante se muestra en la figura 7.5. Dada una dirección virtual, el procesador examinará primero la TLB. Si la entrada de tabla de páginas buscada está presente (un acierto en la TLB), se obtiene el número de marco y se forma la dirección real. Si no se encuentra la entrada de la tabla de páginas buscada (un fallo de TLB), el procesador emplea el número de página para buscar en la tabla de páginas del proceso y examinar la entrada correspondiente de la tabla de páginas. Si se encuentra activo el bit de presencia, es que la página está en memoria principal y el procesador puede obtener el número de marco de la entrada de la tabla de páginas para formar la dirección real. El procesador, además, actualiza la TLB para incluir esta nueva entrada de la tabla de páginas. Por último, si el bit de presencia no está activo, es que la página buscada no está en memoria principal y se produce un fallo en el acceso a memoria, llamado **fallo de página.** En este punto, se abandona el ámbito del hardware y se invoca al sistema operativo, que carga la página necesaria y actualiza la tabla de páginas.

La figura 7.6 es un diagrama de flujo que representa el uso de la TLB. El diagrama muestra que si una página no está en memoria principal, una interrupción de fallo de página hace que se llame a la rutina de gestión de fallos de página. Para mantener la sencillez del diagrama, no se tiene en cuenta el hecho de que el sistema operativo puede pasar a ejecutar otro proceso mientras se realiza la operación de E/S a disco. Debido al principio de cercanía, la

' Véase el Apéndice 7A para un estudio sobre la dispersión.

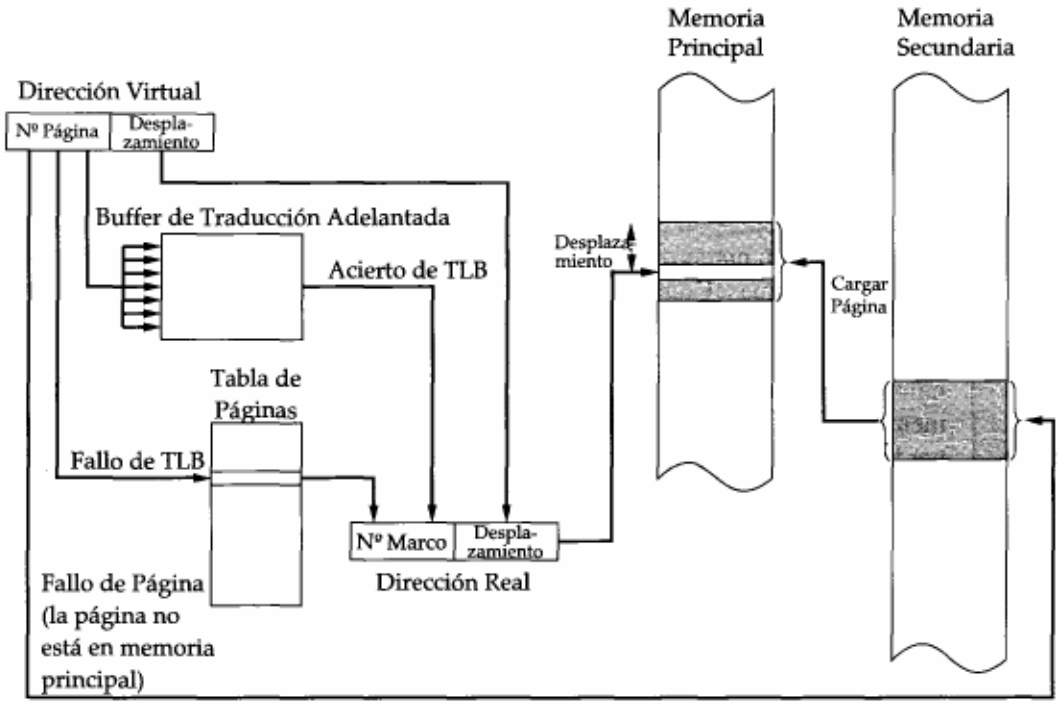


FIGURA 7.5 Empleo del buffer de traducción adelantada

mayoría de las referencias a memoria virtual se situarán en las páginas usadas recientemente. Por tanto, la mayoría de las referencias van a involucrar a las entradas de la tabla de páginas en la cache. Algunos estudios sobre la TLB del VAX han demostrado que este esquema puede incrementar significativamente el rendimiento [CLAR85, SATY81].

Existe una serie de detalles adicionales sobre la organización real de la TLB. Puesto que la TLB contiene sólo algunas de las entradas de la tabla de páginas completa, no se puede indexar simplemente la TLB por el número de página. En su lugar, cada entrada de la TLB debe incluir el número de página, además de la entrada completa a la tabla de páginas. El procesador estará equipado con hardware que permita consultar simultáneamente varias entradas de la TLB para determinar si hay una coincidencia en el número de página. Esta técnica llamada *correspondencia asociativa* y contrasta con la correspondencia directa, que se emplea para buscar en la tabla de páginas de la figura 7.7. El diseñador de la TLB debe tener también en cuenta la forma en que se organizan las entradas en la TLB y qué entrada reemplazar cuando se introduce una nueva. Estas cuestiones deben considerarse en cualquier diseño de cache de hardware. Este problema no se trata aquí; se debe consultar un tratado de diseño de caches para obtener más detalles (por ejemplo, [STAL93a]).

Por último, el mecanismo de memoria virtual debe interactuar con el sistema de cache (no con la cache TLB, sino con la cache de memoria principal). Esto se ilustra en la figura 7.8. Una dirección virtual tendrá normalmente un formato de número de página más desplazamiento. En primer lugar, el sistema de memoria consulta la TLB para ver si encuentra la en-

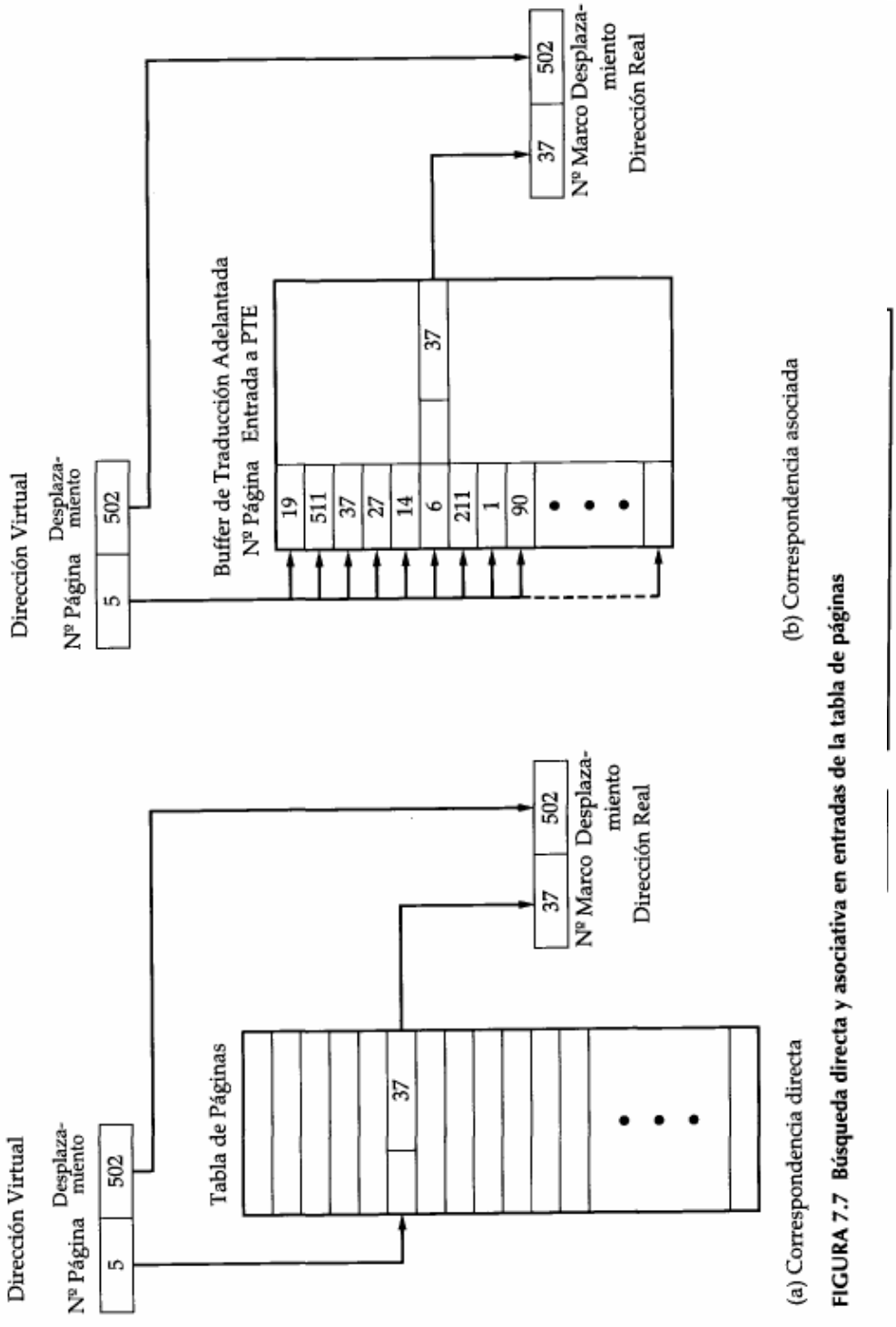


FIGURA 7.7 Búsqueda directa y asociativa en entradas de la tabla de páginas

FIGURA 7.6 Funcionamiento de la paginación con buffer de traducción adelantada (TLB) [FURH

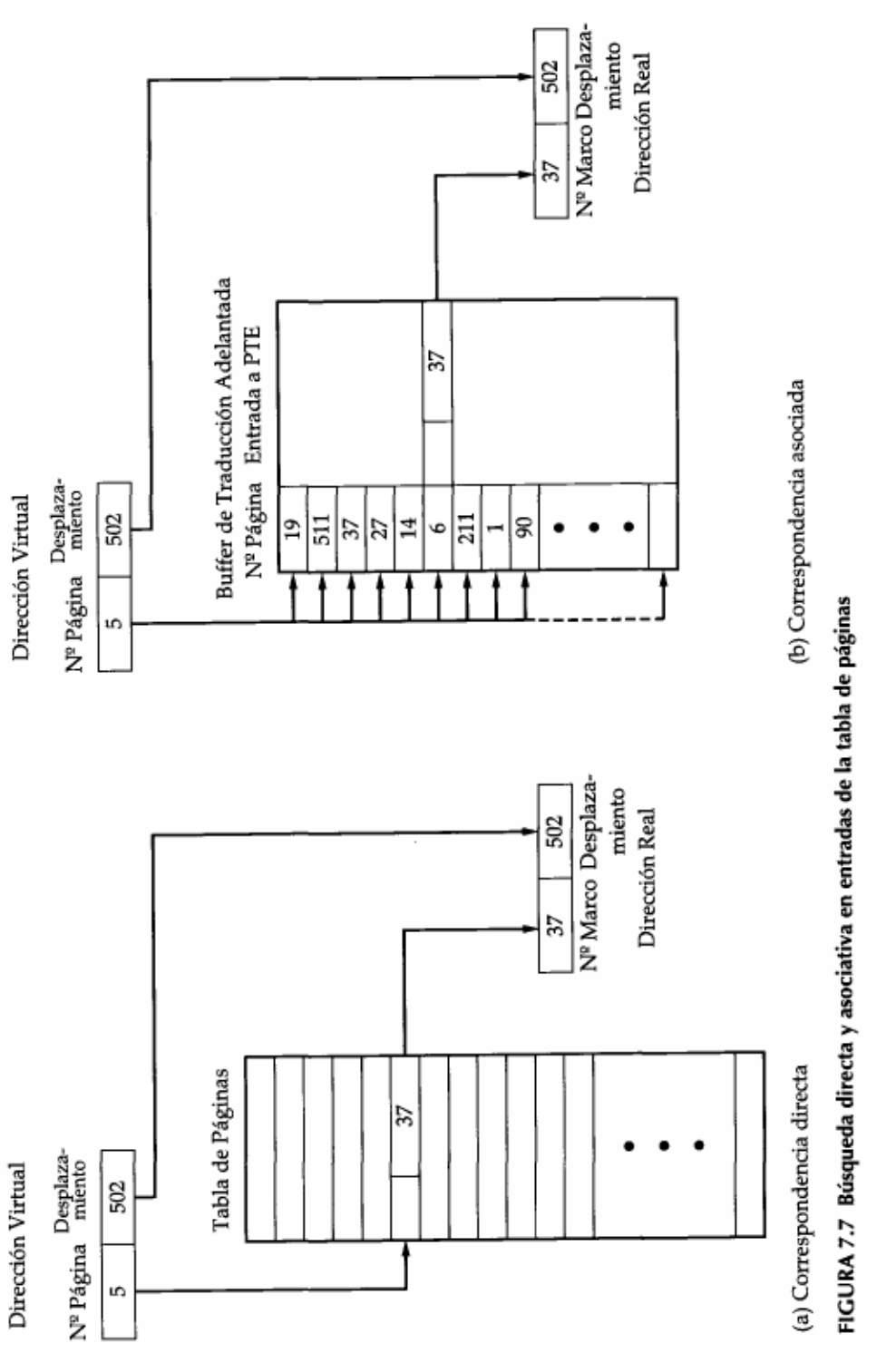


FIGURA 7.7 Búsqueda directa y asociativa en entradas de la tabla de páginas

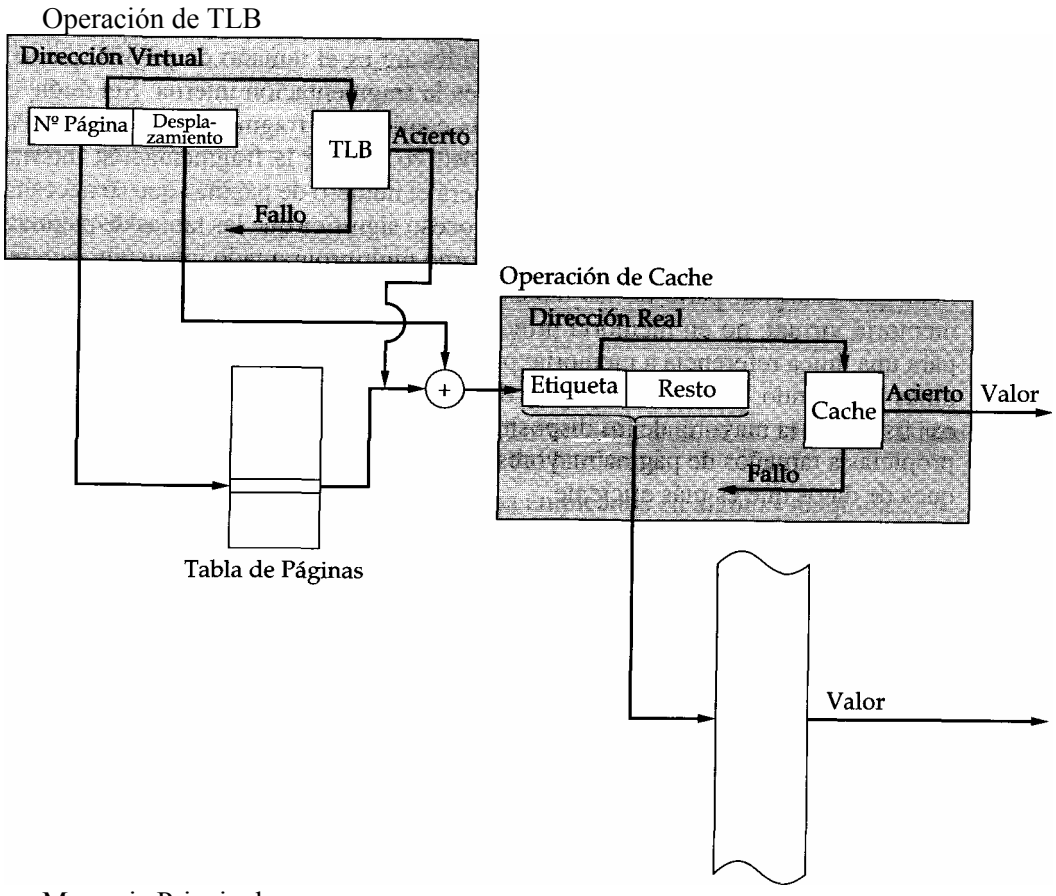


FIGURA 7.8 Funcionamiento del buffer de traducción adelantada y la cache

trada de la tabla de páginas correspondiente. Si es así, la dirección real (física) se genera combinando el número de marco con el desplazamiento. Si no, se accede a la entrada de la tabla de páginas. Una vez que se tiene la dirección real, que está en forma de una etiqueta[2] y un resto, se consulta la cache para ver si está presente el bloque que contiene dicha palabra. Si lo está, es devuelto a la CPU. Si no, se toma la palabra de memoria principal.

Se puede apreciar la complejidad del hardware de la CPU involucrado en una única referencia a memoria. Traducir la dirección virtual a una dirección real supone hacer referencia a una entrada de la tabla de páginas, que puede estar en la TLB, en memoria principal o en disco. La palabra referenciada puede estar en cache, en memoria principal o en disco. En este último caso, debe cargarse en memoria principal la página que contiene la palabra y su bloque en la cache. Además, se debe actualizar la entrada de la tabla de páginas para dicha página.

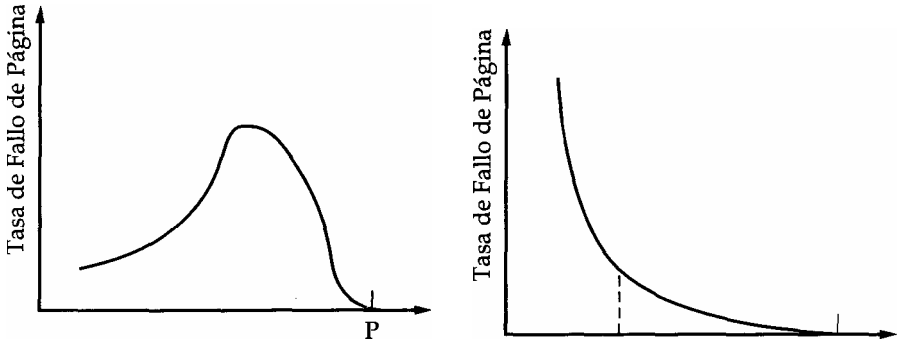
[2] Ver la figura 1.21. Normalmente, una etiqueta está formada por los bits más significativos de la dirección real De nuevo, para un estudio más detallado de las caches, véase [STAL93a].

Tamaño de Página

Una decisión importante de diseño del hardware es el tamaño de página que se va a usar. Hay varios factores que considerar. Uno es la fragmentación interna. Sin duda, cuanto menor sea el tamaño de página, menor será la cantidad de fragmentación interna. Para optimizar el uso de la memoria principal, es positivo reducir la fragmentación interna. Por otro lado, cuanto menor sea la página, mayor será el número de páginas que se necesitan por proceso. Un número mayor de páginas por proceso significa que las tablas de páginas serán mayores. Para programas grandes, en un entorno multiprogramado intensivo, esto puede significar que una gran parte de las tablas de páginas de los procesos activos deban estar en memoria virtual, no en memoria principal. Así pues, pueden suceder dos fallos de página para una única referencia a memoria: primero, para traer la parte necesaria de la tabla de páginas y, segundo, para traer la página del proceso. Otro factor radica en que las características físicas de la mayoría de los dispositivos de memoria secundaria, que son de rotación, son propicias a tamaños de página mayores, puesto que así se obtiene una transferencia por bloques de datos que es más eficiente.

Para complicar la cuestión se tiene el efecto que tiene el tamaño de página en el porcentaje de fallos de página. Este comportamiento se representa, en líneas generales, en la figura 7.9a y se basa en el principio de cercanía. Si el tamaño de página es muy pequeño, normalmente estarán disponibles en memoria principal un gran número de páginas para cada proceso. Después de un tiempo, todas las páginas en memoria contendrán parte de las referencias más recientes del proceso. Así pues, la tasa de fallos de página será menor. Cuando se incrementa el tamaño de la página, cada página individual contendrá posiciones cada vez más distantes de cualquier referencia reciente. Así pues, se atenúa el efecto del principio de cercanía y comienza a aumentar la tasa de fallos de página. Sin embargo, la tasa de fallos de página comenzará a bajar cuando, finalmente, el tamaño de página se aproxime al tamaño de todo el proceso (punto P del diagrama). Cuando una única página abarca todo el proceso, no hay fallos de página.

Una dificultad más es que la tasa de fallos de página viene determinada también por el número de marcos asignados a un proceso. La figura 7.9b demuestra que, para un tamaño de página fijo, la tasa de fallos baja conforme sube el número de páginas contenidas en memo-



(a) Tamaño de pagina W N (b) Número de marcos de página asignados

FIGURA 7.9 Comportamiento típico de la paginación en un programa

TABLA 7.2 Tamaños de página de ejemplo

Computadora	Tamaño de Página	Unidad
Atlas	512	palabra de 48 bits
Honeywell-Multics	1024	palabra de 36 bits
Familia IBM 370	2048 ó 4096	Byte de 8 bits
IBM 370/XA y 370/ESA	4096	Byte de 8 bits
Familia VAX	512	Byte de 8 bits
IBM AS/400	512	Byte de 8 bits
Intel 486 (IBM PC)	4096	Byte de 8 bits
Motorola 68040 (Macintosh)	4096	Byte de 8 bits

ria principal. Así pues, una política del software (la cantidad de memoria asignada a cada proceso) afectará a una decisión de diseño del hardware.

La tabla 7.2 indica los tamaños de página empleados en algunas máquinas.

Por último, el diseño del tamaño de página está relacionado con el tamaño de la memoria física principal. Las implementaciones de memoria virtual en la mayoría de los sistemas actuales fueron diseñadas bajo el supuesto de que la memoria física no es muy grande, normalmente comprendida en un rango de 4M a 256M bytes. Sin embargo, en los próximos años se pueden esperar que aparezcan tamaños de memoria física mayores en estaciones de trabajo y grandes máquinas.

Al mismo tiempo que la memoria principal se hace mayor, el espacio de direcciones que emplean las aplicaciones también crece. Esta tendencia es más evidente en los computadores personales y estaciones de trabajo, donde las aplicaciones se hacen cada vez más complejas. Es más, las técnicas de programación empleadas actualmente en los programas grandes tienden a disminuir la cercanía de las referencias dentro de un proceso [HUCK93]. Por ejemplo:

• Las técnicas de orientación a objetos fomentan el uso de varios módulos pequeños de programas y datos con referencias dispersas por un número de objetos relativamente grande en un periodo corto de tiempo.

• Las aplicaciones con varios hilos dan como resultado cambios bruscos en el flujo de instrucciones y referencias a memoria dispersas.

Para un tamaño dado de TLB, a medida que crece el tamaño de los procesos en memoria y decrece su cercanía, disminuye el porcentaje de acierto en accesos a la TLB. En estas circunstancias, la TLB puede llegar a constituir un cuello de botella del rendimiento (por ejemplo, véase [CHEN92]).

Una forma de incrementar el rendimiento de la TLB es usar TLB mayores y con más entradas. Sin embargo, el tamaño de la TLB influye en otros aspectos del diseño del hardware, como la cache de memoria principal y el número de accesos a memoria por ciclo de instrucción [TALL92]. El resultado es que no es probable que crezca el tamaño de la TLB tan rápidamente como el de la memoria principal. Una alternativa consiste en usar tamaños de página grandes de forma que cada entrada a la tabla de páginas de la TLB haga referencia a un bloque de memoria mayor. No obstante, ya se ha visto que la elección de tamaños de página grandes pueden acabar en una degradación del rendimiento.

Por consiguiente, varios diseñadores han investigado el uso de varios tamaños de página [TALL92, KHAL93] en diversas arquitecturas de microprocesador que lo permiten, incluyendo el R4000, Alpha y SuperSPARC. Por ejemplo, el R4000 permite siete tamaños de página (desde 4K hasta 16M bytes). La posibilidad de tener varios tamaños de página ofrece la flexibilidad necesaria para un uso eficaz de la TLB. Por ejemplo, regiones grandes que están contiguas en el espacio de direcciones de un proceso, como las instrucciones de un programa, pueden asignarse a un pequeño número de páginas grandes, en vez de a un gran número de páginas pequeñas, mientras que las pilas de los hilos pueden asignarse por medio de páginas de pequeño tamaño.

Segmentación

Consecuencias de la Memoria Virtual

La segmentación permite al programador contemplar la memoria como si constara de varios espacios de direcciones o segmentos. Con memoria virtual, el programador no necesita preocuparse de las limitaciones de memoria impuestas por la memoria principal. Los segmentos pueden ser de distintos tamaños, incluso de forma dinámica. Las referencias a memoria constan de una dirección de la forma (número de segmento, desplazamiento).

Esta organización ofrece al programador varias ventajas sobre un espacio de direcciones no segmentado:

1. Simplifica el manejo de estructuras de datos crecientes. Si el programador no conoce *a priori* cuan larga puede llegar a ser una estructura de datos determinada, es necesario suponerlo a menos que se permitan tamaños de segmento dinámicos. Con memoria virtual segmentada, a cada estructura de datos se le puede asignar a su propio segmento y el sistema operativo expandirá o reducirá el segmento cuando se necesite. Si un segmento necesita expandirse en memoria y no dispone de suficiente sitio, el sistema operativo puede mover el segmento a un área mayor de la memoria principal, si la hay disponible, o descargarlo. En este último caso, el segmento agrandado será devuelto a memoria en la siguiente ocasión.
2. Permite modificar y recompilar los programas independientemente, sin que sea necesario recompilar o volver a montar el conjunto de programas por completo. Esto se puede llevar nuevamente a cabo gracias al uso de varios segmentos.
3. Se presta a la compartición entre procesos. Un programador puede situar un programa de utilidades o una tabla de datos en un segmento que pueda ser referenciado por otros procesos.
4. Se presta a la protección. Puesto que un segmento puede ser construido para albergar un conjunto de procedimientos y datos bien definido, el programador o el administrador del sistema podrá asignar los permisos de acceso de la forma adecuada.

Organización

En el estudio de la segmentación simple, se llegó a la conclusión de que cada proceso tiene su propia tabla de segmentos y que, cuando todos los segmentos se encuentran en memoria principal, la tabla de segmentos del proceso se crea y carga en memoria. Cada entrada de la tabla de segmentos contiene la dirección de comienzo del segmento correspondiente en memoria principal, así como su longitud. La misma estructura, una tabla de segmentos, se necesitara al hablar de un esquema de memoria virtual basado en segmentación. Nuevamente, es normal asociar una única tabla de segmentos a cada proceso. En este caso, sin embargo, las entradas de la tabla de segmentos pasan

a ser más complejas (figura 7.2b). Puesto que sólo algunos de los segmentos de un proceso estarán en memoria principal, se necesita un bit en cada entrada de la tabla de segmentos para indicar si el segmento correspondiente está presente en memoria principal. Si el bit indica que el segmento está en memoria, la entrada incluye también la dirección de comienzo y la longitud del segmento.

Otro bit de control necesario en la entrada de la tabla de segmentos es un bit de modificación que indique si el contenido del segmento correspondiente ha sido modificado desde que se cargó por última vez en memoria principal. Si no ha habido cambios, no será necesario escribir a disco el segmento cuando llegue el momento de reemplazarlo en el marco que ocupa actualmente. Puede haber también otros bits de control. Por ejemplo, si se gestiona la protección o la compartición a nivel del segmento, se necesitarán bits con tal propósito. Más adelante se verán varios ejemplos de entradas de tablas de segmentos.

Así pues, el mecanismo básico para leer una palabra de memoria supone la traducción de una dirección virtual o lógica, formada por un número de segmento y un desplazamiento, a una dirección física, mediante una tabla de segmentos. Puesto que la tabla de segmentos tiene longitud variable, en función del tamaño del proceso, no es posible suponer que quepa en registros. En su lugar, debe estar en memoria principal para que sea accesible. La figura 7.10 sugiere una implementación en hardware de este esquema. Cuando un proceso determinado está ejecutando, la dirección de comienzo de la tabla de segmentos de este proceso se guardará en un registro. El número de segmento de la dirección virtual se emplea como índice de la tabla para buscar la dirección de memoria principal correspondiente al comienzo del segmento. Esta se añade a la parte de desplazamiento de la dirección virtual para generar la dirección real deseada.

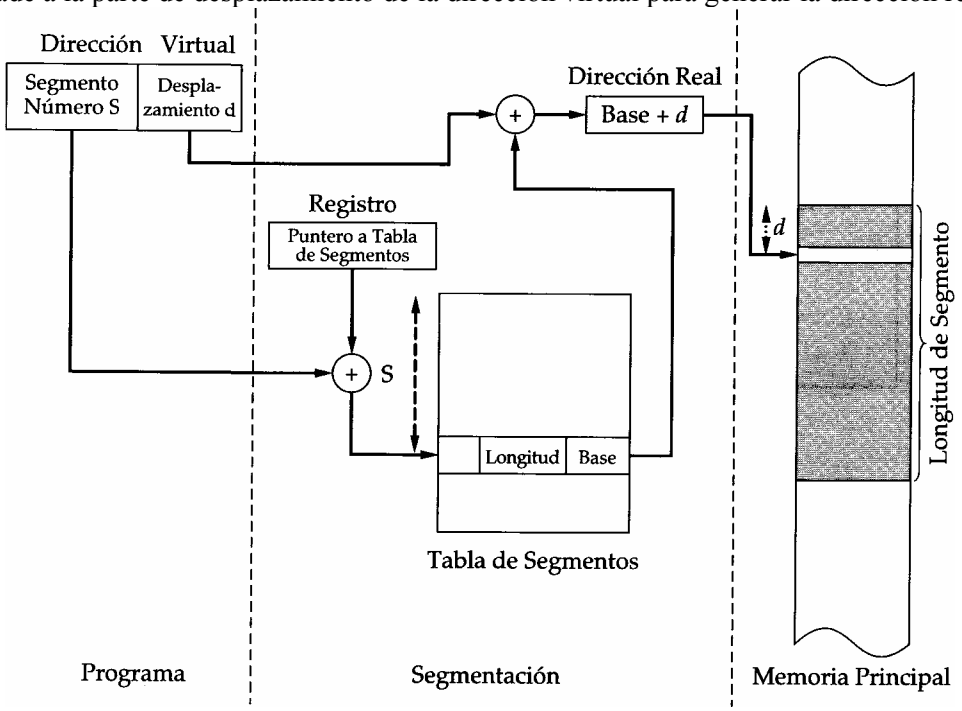


FIGURA 7.10 Traducción de direcciones en un sistema con segmentación

Paginación y Segmentación Combinadas

Tanto la paginación como la segmentación tienen sus ventajas. La paginación, que es transparente al programador, elimina la fragmentación externa y, de este modo, aprovecha la memoria principal de forma eficiente. Además, puesto que los fragmentos que se cargan y descargan de memoria principal son de tamaño constante e igual para todos, es posible construir algoritmos de gestión de memoria sofisticados que aprovechen mejor el comportamiento de los programas, tal y como se verá. La segmentación, que es visible para el programador, tiene las ventajas antes citadas, incluida la capacidad de manejar estructuras de datos que puedan crecer, la modularidad y el soporte de la compartición y la protección. Para combinar las ventajas de ambas, algunos sistemas están equipados con hardware del procesador y software del sistema operativo que las permiten.

En un sistema con paginación y segmentación combinadas, el espacio de direcciones de un usuario se divide en varios segmentos según el criterio del programador. Cada segmento se vuelve a dividir en varias páginas de tamaño fijo, que tienen la misma longitud que un marco de memoria principal. Si el segmento tiene menor longitud que la página, el segmento ocupará sólo una página. Desde el punto de vista del programador, una dirección lógica también está formada por un número de segmento y un desplazamiento en el segmento. Desde el punto de vista del sistema, el desplazamiento del segmento se ve como un número de página dentro del segmento y un desplazamiento dentro de la página.

La figura 7.11 propone una estructura para soportar la combinación de paginación y segmentación. Asociada con cada proceso existe una tabla de segmentos y varias tablas de páginas, una por cada segmento del proceso. Cuando un proceso determinado está ejecután-

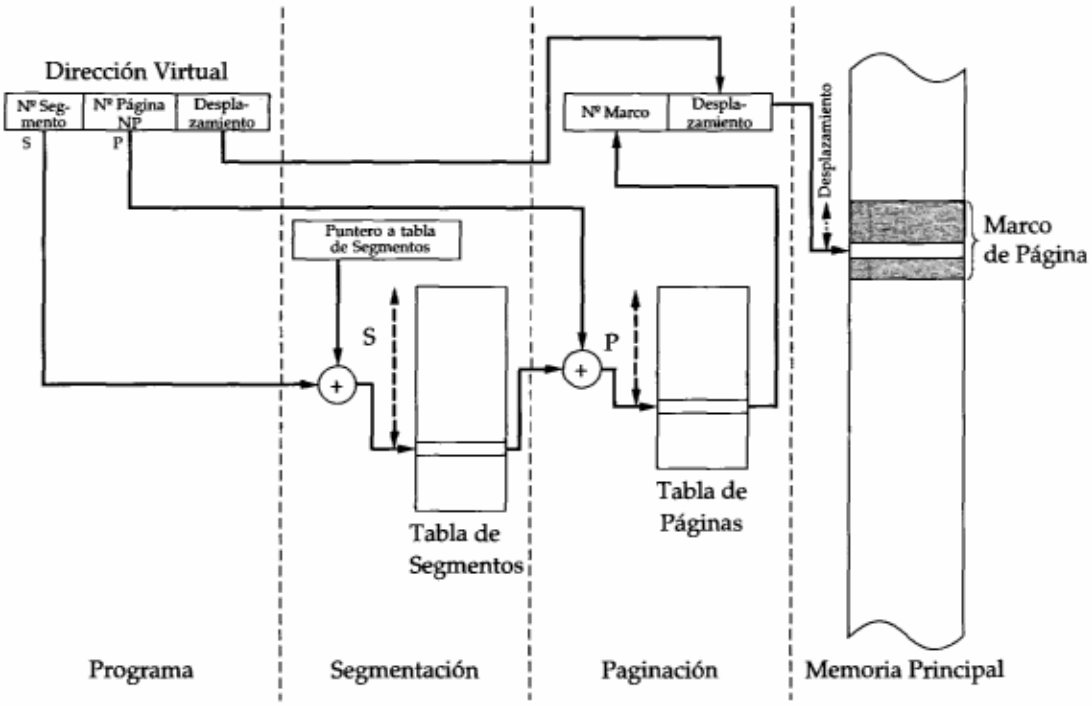


FIGURA 7.11 Traducción de direcciones en un sistema con segmentación y paginación

dose, un registro contendrá la dirección de comienzo de la tabla de segmentos para ese proceso. Dada una dirección virtual, el procesador emplea la parte de número de segmento como índice en la tabla de segmentos del proceso para encontrar la tabla de páginas de dicho segmento. Entonces, la parte de número de página de la dirección virtual se usará como índice en la tabla de páginas para localizar el número de marco correspondiente. Este se combina con la parte de desplazamiento de la dirección virtual para generar la dirección real deseada.

La figura 7.2a propone los formatos de la entrada de la tabla de segmentos y de la entrada de la tabla de páginas. Como antes, la entrada de la tabla de segmentos contiene la longitud del segmento. También contiene un campo base, que ahora se refiere a una tabla de páginas. Los bits de presencia y modificación no son necesarios, puesto que estos elementos se manejan a nivel de página. Pueden usarse otros bits de control para compartición y protección. La entrada de la tabla de páginas es, básicamente, la misma que se usa en un sistema de paginación pura. Cada número de página se convierte en el número de marco correspondiente si la página está presente en memoria. El bit de modificación indica si se necesita escribir la página a disco cuando se asigna el marco a otra página. Además, pueden haber otros bits de control para ocuparse de la protección y de otros aspectos de la gestión de memoria.

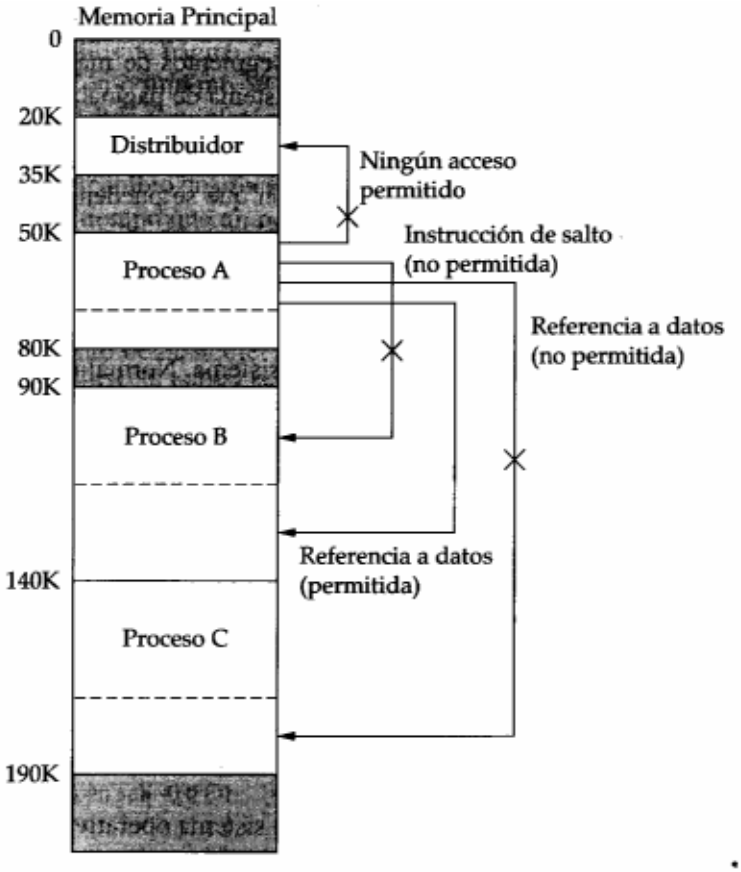
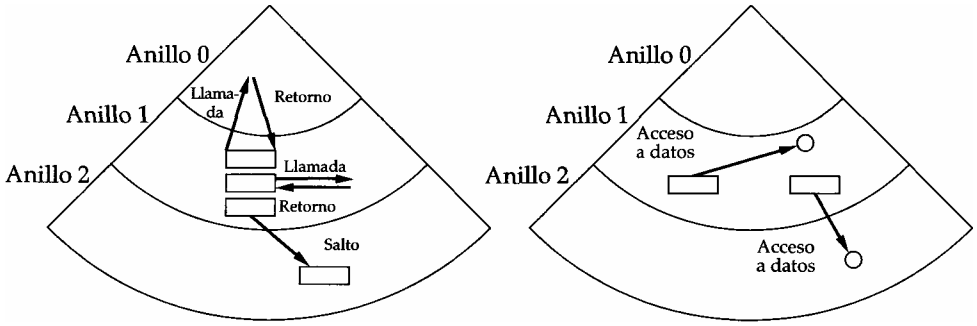


FIGURA 7.12 Relaciones de protección entre segmentos



(a) Transferencia de control entre programas (b) Accesos a datos

FIGURA 7.13 Convenios de protección por anillos [FURH87]

Protección y Compartición

La segmentación se presta a la implementacióni de políticas de protección y compartición. Puesto que cada entrada de la tabla de segmentos incluye la longitud, además de la dirección base, un programa no podrá acceder por descuido a una posición de memoria principal más allá de los límites de un segmento. Para conseguir la compartición, es posible que un segmento se referencie en las tablas de segmentos de más de un proceso. Este mismo mecanismo es, por supuesto, válido en un sistema de paginación. Sin embargo, es este caso, la estructura de páginas de programas y datos no es visible al programador, haciendo que la especificación de los requisitos de protección y seguridad sea más difícil. La figura 7.12 indica el tipo de relaciones de protección que se pueden definir entre segmentos.

Se pueden ofrecer también mecanismos más sofisticados. Un esquema habitual consiste en usar una estructura de anillo de protección como la que se explicó en el capítulo 3. Con este esquema, los anillos más interiores o con números menores gozan de mayores privilegios que los anillos externos o con números mayores. La figura 7.13 ilustra los tipos de protección que pueden aplicarse en cada sistema. Normalmente, el anillo O está reservado para las funciones del núcleo del sistema operativo y las aplicaciones están situadas en un nivel más alto. Algunas utilidades o servicios del sistema operativo pueden ocupar un anillo intermedio. Los principios básicos del sistema de anillos son los siguientes:

1. Un programa puede acceder sólo a datos que estén en el mismo anillo o en un anillo de menor privilegio.
2. Un programa puede hacer llamadas a servicios que residan en el mismo anillo o en anillos más privilegiados.

7.2 SOFTWARE DEL SISTEMA OPERATIVO

El diseño del gestor de memoria en un sistema operativo depende de tres áreas fundamentales de decisión:

- Si se emplean o no técnicas de memoria virtual

• El uso de paginación, segmentación o ambas

• Los algoritmos empleados para los diversos problemas de la gestión de memoria

Las decisiones tomadas en las dos primeras áreas dependen de la plataforma de hardware disponible. De este modo, las primeras implementaciones de UNIX no ofrecían memoria virtual, porque los procesadores en los que ejecutaba el sistema operativo no ofrecían paginación ni segmentación. Ninguna de esas técnicas resulta práctica sin un soporte de hardware para la traducción de direcciones y otras funciones básicas.

Dos comentarios adicionales acerca de los dos primeros puntos de la lista anterior: Primero, con la excepción de los sistemas operativos para algunos computadores personales antiguas, tales como MS-DOS y para sistemas especializados, todos los sistemas operativos importantes ofrecen memoria virtual. Por tanto, se hará un mayor hincapié en este capítulo en el tema de la memoria virtual. Segundo, los sistemas de segmentación pura se han convertido en algo cada vez menos común. Cuando se combina la segmentación con paginación, la mayoría de los problemas de gestión de memoria con las que se enfrenta el diseñador del sistema operativo están en el área de la paginación. Así pues, esta sección se centrará en los elementos asociados con la paginación.

Las decisiones del tercer punto (los algoritmos) entran en el dominio del software del sistema operativo y son el objeto de esta sección. La tabla 7.3 enumera los elementos clave de diseño que se van a examinar. En cada caso, el punto clave es el rendimiento: Se busca minimizar el porcentaje de fallos de página. Los fallos de página originan una considerable sobrecarga en el software. Como mínimo, el sistema operativo debe tomar la decisión sobre qué páginas residentes se deben reemplazar y sobre el intercambio de E/S involucrado. Después, el sistema operativo debe planificar otro proceso para ejecutar durante la E/S de páginas, lo que provoca un intercambio de procesos. Por consiguiente, se pretenden organizar las cosas para que, durante el tiempo que un proceso está ejecutándose, la probabilidad de que haga referencia a una palabra de una página ausente sea mínima. En todas las áreas referidas en la tabla 7.3, no hay una política definitiva que funcione mejor que las demás. Como se verá, la tarea del gestor de memoria en un entorno de paginación es extremadamente compleja. Es más, el rendimiento de cualquier conjunto particular de políticas depende del tamaño de la memoria principal, la velocidad relativa de memoria principal y secundaria, el

TABLA 7.3 Políticas del Sistema Operativo sobre Memoria Virtual

Políticas de lectura	**Gestión del conjunto residente**
Por demanda	Tamaño del conjunto residente
Paginación previa	Fijo
Políticas de ubicación	Variable
Políticas de reemplazo	Alcance del reemplazo
Algoritmos básicos	Global
Óptimo	Local
Usada hace más tiempo (LRU)	**Políticas de vaciado**
Primera en llegar/primera en salir (FIFO)	Por demanda
Reloj	Vaciado previo
Memoria intermedia de páginas	**Control de carga**
	Grado de multiprogramación

tamaño y número de procesos que compiten por los recursos y el comportamiento en ejecución de los programas individuales. Esta última característica depende de la naturaleza de la aplicación, el lenguaje de programación y el compilador empleado, el estilo del programador que lo escribió y, para un programa interactivo, el comportamiento dinámico del usuario. De este modo, no se pueden esperar respuestas definitivas aquí o en algún otro sitio. Para sistemas pequeños, el diseñador del sistema operativo intentará elegir un conjunto de políticas que parezcan "buenas" entre un amplio rango de condiciones, en función del estado actual del conocimiento. Para sistemas mayores, especialmente *mainframes,* el sistema operativo estará equipado con herramientas de supervisión y control que permiten al administrador de la instalación configurar el sistema operativo para obtener "buenos" resultados en función de las condiciones de dicha instalación.

Políticas de lectura

La política de lectura *(fetch)* está relacionada con la decisión de cuándo se debe cargar una página en memoria principal. Las dos alternativas más comunes son la paginación por demanda y la paginación previa. Con **paginación por demanda,** se trae una página a memoria principal sólo cuando se hace referencia a una posición en dicha página. Si los otros elementos de la política de gestión de memoria funcionan adecuadamente, debe ocurrir lo siguiente. Cuando un proceso se ejecute por primera vez, se producirá un aluvión de fallos de página. A medida que se traigan a memoria más páginas, el principio de cercanía hará que la mayoría de las futuras referencias estén en páginas que se han cargado hace poco. Así pues, después de un tiempo, la situación se estabilizará y el número de fallos de página disminuirá hasta un nivel muy bajo.

Con **paginación previa,** se cargan otras páginas distintas a las demandadas debido a un fallo de página. El principal atractivo de esta estrategia está en las características de las mayoría de los dispositivos de memoria secundaria, como los discos, que tienen un tiempo de búsqueda y una latencia de giro. Si las páginas de un proceso se cargan secuencialmente en memoria secundaria, es más eficiente traer a memoria un número de páginas contiguas de una vez que ir trayéndolas de una en una durante un periodo largo de tiempo. Por supuesto, esta política no es efectiva si la mayoría de las páginas extra que se traen no se referencian.

La política de paginación previa puede emplearse incluso cuando un proceso se ejecuta por primera vez, en cuyo caso, el programador tendría que designar de algún modo las páginas deseadas o hacerlo cada vez que se produzca un fallo de página. Esta última forma parece ser preferible, puesto que es invisible al programador. Sin embargo, la utilidad de la paginación previa no ha sido demostrada [MAEK87].

Una última observación sobre la paginación previa: La paginación previa no debe confundirse con el intercambio. Cuando un proceso se descarga de memoria y pasa al estado suspendido, todas sus páginas residentes se llevan también fuera. Cuando se reanuda el proceso, todas las páginas que estaban antes en memoria principal se devuelven a la misma. Esta política es la seguida por la mayoría de los sistemas operativos.

Políticas de ubicación

La política de ubicación tiene que ver con determinar dónde va a residir una parte de un proceso en memoria principal. En un sistema de segmentación puro, la política de ubicación es un aspecto importante del diseño; como posibles alternativas se tienen las

políticas del mejor ajuste, el primer ajuste y otras, que se trataron en el capítulo 6. Sin embargo, para un sistema que usa tanto paginación pura como paginación combinada con segmentación, la ubicación carece normalmente de importancia, puesto que el hardware de traducción de direcciones y el hardware de acceso a memoria principal pueden desarrollar sus funciones para cualquier combinación de marco de página con idéntica eficiencia.

Hay un campo en el que la ubicación llega a tener interés y ha sido objeto de recientes investigaciones y desarrollos. En los llamados multiprocesadores de acceso a memoria no uniforme (NUMA, *Non-Uniform Memory Access),* la memoria compartida, distribuida de la máquina puede referenciarse desde cualquier procesador, pero el tiempo de acceso a una dirección física en particular varía con la distancia entre el procesador y el módulo de memoria. Así pues, el rendimiento depende en gran medida de hasta qué punto los datos están cerca del procesador que los usa [BOL089, COX89, LAR092]. Para los sistemas NUMA, es conveniente una estrategia de ubicación automática que asigne las páginas a los módulos de memoria que proporcionen el mejor rendimiento.

Políticas de reemplazo

En la mayoría de los textos de sistemas operativos, el tratamiento de la gestión de memoria incluye una sección titulada "políticas de reemplazo", que trata de la selección de la página a reemplazar en memoria principal cuando se debe cargar una nueva página. Este punto resulta difícil de explicar a veces porque se encuentran involucrados varios conceptos interrelacionados, tales como los siguientes:

- El número de marcos de página a asignar a cada proceso activo
- Si el conjunto de páginas candidatas para el reemplazo debe limitarse a las del proceso que provocó el fallo de página o abarcará a todos los marcos de página situados en memoria principal
- De entre el conjunto de páginas candidatas, la página que debe elegirse en particular para el reemplazo

Se hará referencia al primer concepto como *gestión del conjunto residente,* que se tratará en el apartado siguiente y, se reserva el término *política de reemplazo* para el tercer concepto, que se va a tratar en este apartado.

El tema de la política de reemplazo es probablemente la más estudiada dentro de la gestión de memoria en los últimos veinte años. Cuando todos los marcos de memoria principal están ocupados y es necesario traer una nueva página para atender a un fallo de página, la política de reemplazo se encarga de seleccionar la página a reemplazar de entre las que están actualmente en memoria. Todas las políticas tienen como objetivo que la página a reemplazar sea la que tenga una menor posibilidad de ser referenciada en un futuro cercano. Debido al principio de cercanía, hay una alta correlación entre la historia de referencias recientes y las pautas de futuras referencias. Así pues, la mayoría de las políticas intentan predecir el comportamiento futuro en función del comportamiento pasado. Una regla que debe tenerse en cuenta es que cuanto más elaborada y sofisticada es la política de reemplazo, mayor es la sobrecarga de hardware y software necesaria para implementarla.

Bloqueo de marcos

Antes de atender a los diversos algoritmos, es necesario mencionar una restricción de la política de reemplazo: Algunos de los marcos de memoria principal pueden estar bloqueados. Cuando un marco está bloqueado, la página cargada actualmente en este marco no puede ser reemplazada. La mayoría del núcleo del sistema operativo, así como las estructuras clave de control, se albergan en marcos bloqueados. Además, los buffers de E/S y otras áreas críticas en tiempo pueden bloquearse en marcos de memoria. El bloqueo se consigue asociando un bit de bloqueo a cada marco. Este bit puede guardarse en una tabla de marcos o estar incluido en la tabla de páginas actual.

Algoritmos básicos

Además de la estrategia de gestión del conjunto residente (discutida en el siguiente apartado), existen ciertos algoritmos básicos que se emplean para la selección de una página a reemplazar. Entre las políticas de los algoritmos de reemplazo que se han considerado en la bibliografía se incluyen:

- Óptima
- Usada hace más tiempo (LRU, *Least Recently U sed)*
- Primera en entrar, primera en salir (FIFO, *First-In, First-Out)*
- **De** reloj

La política **óptima** selecciona para reemplazar la página que tiene que esperar una mayor cantidad tiempo hasta que se produzca la referencia siguiente. Se puede demostrar que esta política genera el menor número de fallos de página [BELA66]. Sin duda, este algoritmo resulta imposible de implementar, puesto que requiere que el sistema operativo tenga un conocimiento exacto de los sucesos futuros. Sin embargo, sirve como un estándar con el que comparar los otros algoritmos.

La figura 7.14 ofrece un ejemplo de la política óptima. El ejemplo supone una asignación constante de tres marcos para el proceso. La ejecución del proceso hace referencia a cinco páginas distintas. La cadena de referencias a las páginas durante la ejecución del programa es:

232 152453252

lo que significa que la primera referencia es a la página 2, la segunda a la 3 y así sucesivamente. El algoritmo óptimo origina tres fallos de página después de haber llenado los marcos asignados.

La política de la **usada hace más tiempo** (LRU, *Least Recently U sed)* reemplaza la página de memoria que no ha sido referenciada desde hace más tiempo. Debido al principio de cercanía, ésta debería ser la página con menor probabilidad de ser referenciada en un futuro cercano. De hecho, la política LRU afina casi tanto como la política óptima. El problema de este método es su dificultad de implementación. Una solución sería etiquetar cada página con el instante de su última referencia; esto tendría que hacerse para cada referencia a memoria, tanto para instrucciones como datos. Incluso si el hardware respaldara este esquema, la sobrecarga resultaría tremenda. Como alternativa, se podría mantener una pila de referencias a página, aunque también con un coste elevado.

La figura 7.14 muestra un ejemplo del comportamiento del LRU, utilizando la misma cadena de referencias a páginas del ejemplo de la política óptima. En este caso, se producen cuatro fallos de página.

La política de **primera en entrar, primera en salir (FIFO)** trata los marcos asignados a un proceso como un buffer circular y las páginas se suprimen de memoria según la técnica de espera circular *(round-robiri).* Todo lo que se necesita es un puntero que circule a través de los marcos del proceso. Esta es, por tanto, una de las políticas de reemplazo más sencillas de implementar. La lógica que hay detrás de esta elección, además de su sencillez, es reemplazar la página que ha estado más tiempo en memoria: Una página introducida en memoria hace mucho tiempo puede haber caído en desuso. Este razonamiento será a menudo incorrecto, porque habrá regiones de programa o de datos que son muy usadas a lo largo de la vida de un programa. Con el algoritmo FIFO, estas páginas se cargarán y expulsarán repetidas veces.

Continuando con el ejemplo de la figura 7.14, la política FIFO genera seis fallos de página. Nótese que la LRU se da cuenta de que las páginas 2 y 5 se referencian más frecuentemente que las otras, mientras que la FIFO no lo hace.

Si bien la política LRU se acerca mucho a la política óptima, es difícil de implementar e impone una sobrecarga significativa. Por otro lado, la política FIFO es muy simple de implementar, pero su rendimiento es relativamente pobre. A lo largo de los años, los diseñadores de sistemas operativos han probado una serie de algoritmos para aproximarse al rendimiento de la LRU sin introducir mucha sobrecarga. Muchos de esos algoritmos son variantes de un esquema denominado **política del reloj.**

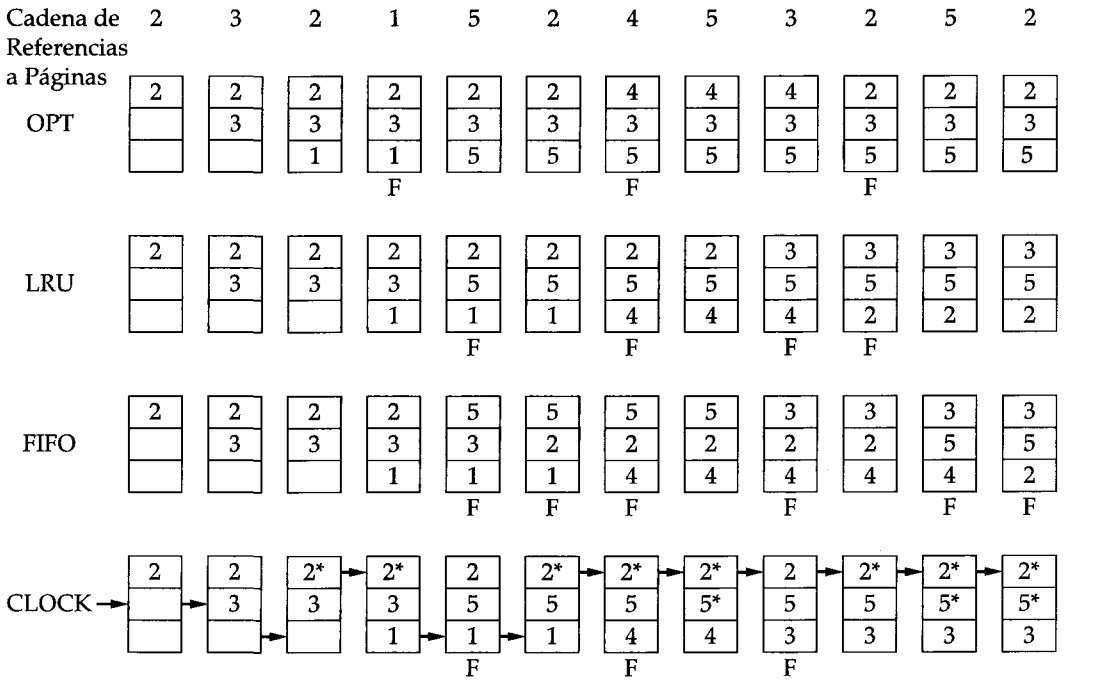


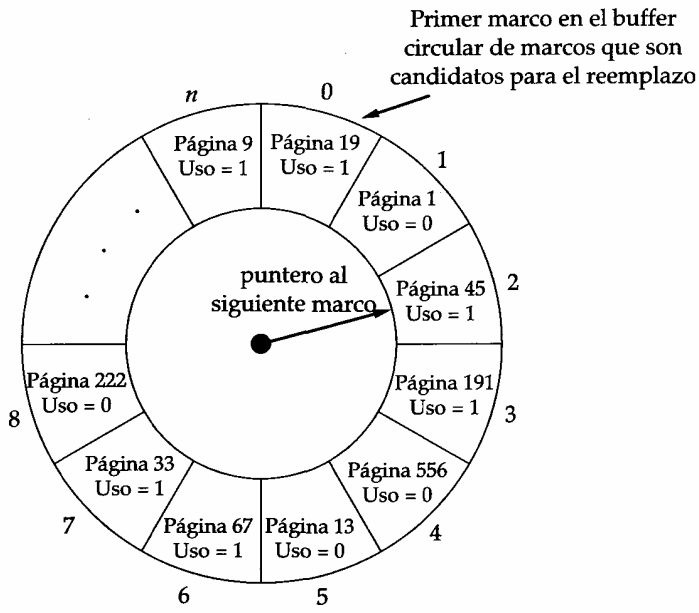
FIGURA 7.14 Comportamiento de cuatro algoritmos de reemplazo de páginas (basado en [HAYE88])

La forma más simple de la política del reloj requiere asociar un bit adicional a cada marco, denominado *bit de uso*. Cuando se carga una página por primera vez en un marco de memoria, el bit de uso de dicho marco se pone a cero. Cuando se hace referencia a la página posteriormente (después de la referencia que generó el fallo de página), el bit de uso se pone a 1. Para el algoritmo de reemplazo de páginas, el conjunto de marcos candidatos a ser reemplazados se considera como un buffer circular con un puntero asociado. El alcance es local si los candidatos son de un solo proceso; el alcance es global si los candidatos provienen de toda la memoria principal. Al reemplazar una página, se hace que el puntero señale al siguiente marco del buffer. Cuando llega el momento de reemplazar una página, el sistema operativo recorre el buffer buscando un marco con el bit de uso a 0. Cada vez que se encuentra un marco con el bit de uso a 1, lo pone a 0. Si algún marco del buffer tiene el bit de uso a 0 al comienzo de la búsqueda, se elige para reemplazar el primero que se haya encontrado. Si todos los marcos tienen el bit de uso puesto a 1, el puntero dará una vuelta completa al buffer, poniendo todos los bits a 0 y se detendrá en la posición inicial, reemplazando la página de dicho marco. Es posible comprobar que esta política es similar a la FIFO, excepto que cualquier marco con el bit de uso a 1 se descarta en el algoritmo. La política se denomina política del reloj porque se pueden imaginar los marcos dispuestos en círculo. Varios sistemas operativos han empleado una variación de esta simple política, como por ejemplo Multics [CORB68].

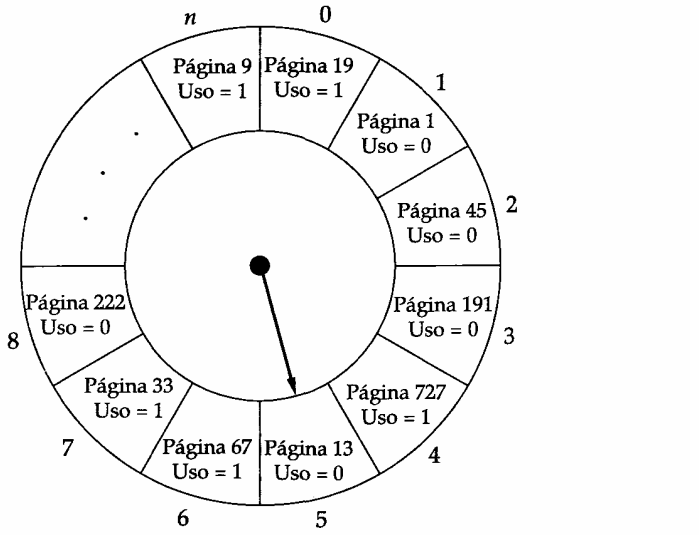
La figura 7.15 muestra un ejemplo del mecanismo de la política del reloj. Hay disponible un buffer circular de *n* marcos de memoria principal para reemplazo de páginas. Justo antes de reemplazar una página del buffer con la 727, el puntero al siguiente marco señala al marco 2, que contiene la página 45. Se aplica ahora la política del reloj. Puesto que el bit de uso para la página 45 en el marco 2 es igual a 1, no se reemplaza está página. En su lugar, el bit de uso se pone a O y el puntero avanza. Análogamente, no se reemplaza la página 191 en el marco 3; su bit de uso se pone a O y avanza el puntero. En el siguiente marco, el 4, el bit de uso está a 0. Por tanto, se reemplaza la página 556 con la página 727. El bit de uso se pone a uno para este marco y el puntero avanza al marco 5, completando el procedimiento de reemplazo de páginas.

El comportamiento de la política del reloj se muestra en la figura 7.14. La presencia de un asterisco indica que el bit de uso correspondiente es igual a 1 y la flecha indica la posición actual del puntero. Así pues, en la tercera unidad de tiempo, se hace referencia a la página 2 y se activa su bit de uso. En la cuarta unidad de tiempo, se carga una página en el tercer marco y el puntero se sitúa en el primero. Sin embargo, puesto que el bit de uso de este marco está activo, no se usará como marco de reemplazo en el quinto instante de tiempo. Nótese que la política del reloj es adecuada para proteger a los marcos 2 y 5 del reemplazo. De hecho, en este ejemplo, la política del reloj supera el rendimiento del LRU. En general, no es éste el caso.

La figura 7.16 muestra el resultado de un experimento expuesto en [BAER80], que compara los cuatro algoritmos que se han estudiado; se supone que el número de marcos asignado a un proceso es fijo. Los resultados están basados en la ejecución de $0{,}25 \times 10^6$ referencias en un programa en FORTRAN usando un tamaño de página de 256 palabras. Baer realizó la prueba con asignaciones de 6, 8, 10, 12 y 14 marcos. Las diferencias entre las cuatro políticas son más notorias con asignaciones pequeñas. Con FIFO se rondaba un factor de 2 veces peor que el óptimo. [FINK88] ha expuesto resultados casi idénticos, que



(a) Estado del buffer justo antes del reemplazo de página



(b) Estado del buffer justo después del siguiente reemplazo de página

FIGURA 7.15 Ejemplo de funcionamiento de la política del reloj

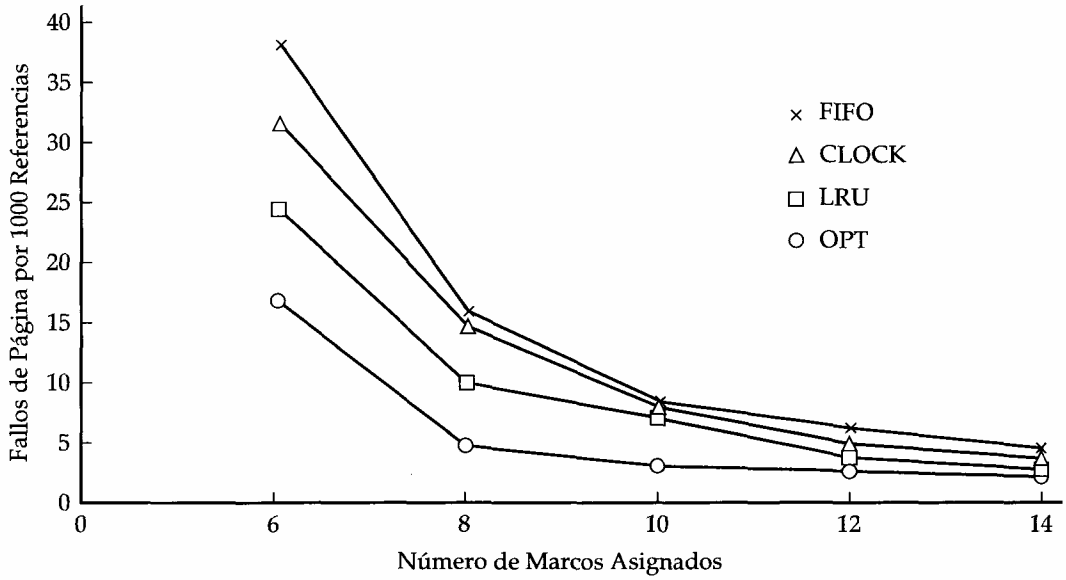


FIGURA 7.16 ***Comparación de algoritmos de asignación fija y reemplazo local***

también demuestran una dispersión máxima cercana a un factor de 2. El método de Finkel era simular los efectos de varias políticas en una cadena de 10.000 referencias artificiales a páginas, seleccionadas de un espacio virtual de 100 páginas. Para aproximarse a los efectos del principio de cercanía, se impuso una distribución exponencial para la probabilidad de hacer referencia a una página en particular. Finkel observa que podría concluirse que casi no vale la pena elaborar algoritmos de reemplazo de páginas cuando sólo está en juego un factor de 2. Pero también observa que esta diferencia tendrá un efecto notable tanto en las exigencias de memoria principal (para evitar la degradación del rendimiento del sistema operativo) como en el rendimiento del sistema operativo (para evitar que haga falta más memoria principal).

El algoritmo del reloj también se ha comparado con los demás algoritmos cuando se emplea asignación variable y un alcance del reemplazo tanto global como local. (Ver el estudio sobre políticas de reemplazo situado más adelante con el título de "Alcance del Reemplazo") [CARR81, CARR84]. Nuevamente, se determinó que el rendimiento del algoritmo del reloj se aproxima al del LRU.

El algoritmo del reloj puede hacerse más potente incrementando el número de bits empleados[3]. En todos los procesadores que ofrecen paginación, se asocia un bit de modificación con cada página en memoria principal y, por tanto, con cada marco. Este bit es necesario para que, cuando se modifica una página, no se reemplace hasta volverla a escribir en memoria secundaria. Es posible aprovechar este bit en el algoritmo del reloj del siguiente modo. Si se tienen en cuenta el bit de uso y el de modificación, cada marco estará en una de las siguientes cuatro categorías:

[3] Por otro lado, si se reduce el número de bits empleados a cero, el algoritmo del reloj degenera a PIFO.

• No accedido recientemente y sin modificar *(u = 0; m* = 0)

• Accedido recientemente y sin modificar (*u* = 1; *m* = 0)

• No accedido recientemente y modificado (w = 0; *m* = 1)

• Accedido recientemente y modificado *(u* = 1; *m* = 1) Con esta clasificación, el algoritmo del reloj se comporta de la forma siguiente:

1. Comenzando en la posición actual del puntero, recorrer el buffer de marcos. Durante este recorrido, no cambiar el bit de uso. Se selecciona para reemplazo el primer marco encontrado con (u = 0; *m* = 0).

2. Si falla el paso 1, recorrer de nuevo buscando marcos con *(u* = 0; *m* = 1). Se selecciona para reemplazar el primer marco encontrado. Durante este recorrido, se pone a O el bit de uso de cada marco por el que se pasa.

3. Si falla el paso 2, el puntero habrá retomado a su posición original y todos los marcos del conjunto tendrán el bit de uso a 0. Repetir el paso 1. Esta vez, se encontrará un marco para reemplazar.

En resumen, el algoritmo de reemplazo de páginas recorre todas las páginas del buffer buscando una que no se haya modificado desde que fue cargada y a la que no se haya accedido recientemente. Tal página es una buena candidata para su reemplazo y tiene la ventaja de que, como no está modificada, no hace falta volver a escribirla en memoria secundaria. Si no se encuentra ninguna candidata en la primera vuelta, el algoritmo recorre el buffer de nuevo buscando una página modificada a la que no se haya accedido recientemente. Incluso aunque dicha página deba escribirse para reemplazarla, debido al principio de cercanía, no será necesaria de nuevo durante algún tiempo. Si esta segunda pasada falla, todos los marcos del buffer no habrán tenido accesos recientes y se llevará a cabo una tercera pasada.

Esta estrategia es la usada en el esquema de memoria virtual del Macintosh [GOLD89] y se muestra en la figura 7.17. La ventaja de este algoritmo sobre el algoritmo simple del reloj es que las páginas que no han sido cambiadas tienen preferencia para reemplazarse. Puesto que una página modificada debe escribirse al disco antes de ser reemplazada, se produce un ahorro de tiempo evidente.

Almacenamiento Intermedio de Páginas

A pesar de que las políticas LRU y del reloj son superiores a la FIFO, ambas poseen una complejidad y sobrecarga que ésta última no padece. Además, el coste de reemplazar una página que ha sido modificada es mayor que el de una que no lo ha sido, porque la primera debe volver a escribirse en memoria secundaria.

Una estrategia interesante que puede mejorar el rendimiento de la paginación y permitir el uso de una política de reemplazo de páginas más sencilla es el almacenamiento intermedio de páginas. Un método representativo es el del VAX/VMS. VMS emplea una estrategia de asignación variable y reemplazo local (véase más adelante el estudio sobre el alcance del reemplazo). El algoritmo de reemplazo de páginas es simplemente un FIFO. Para mejorar el rendimiento, no se pierde la pista de la página reemplazada, sino que se asigna a una de las dos listas siguientes: la lista de páginas libres, si la página no ha sido modificada o la lista de páginas modificadas, si lo ha sido. Nótese que la página no se mueve físicamente de la me-

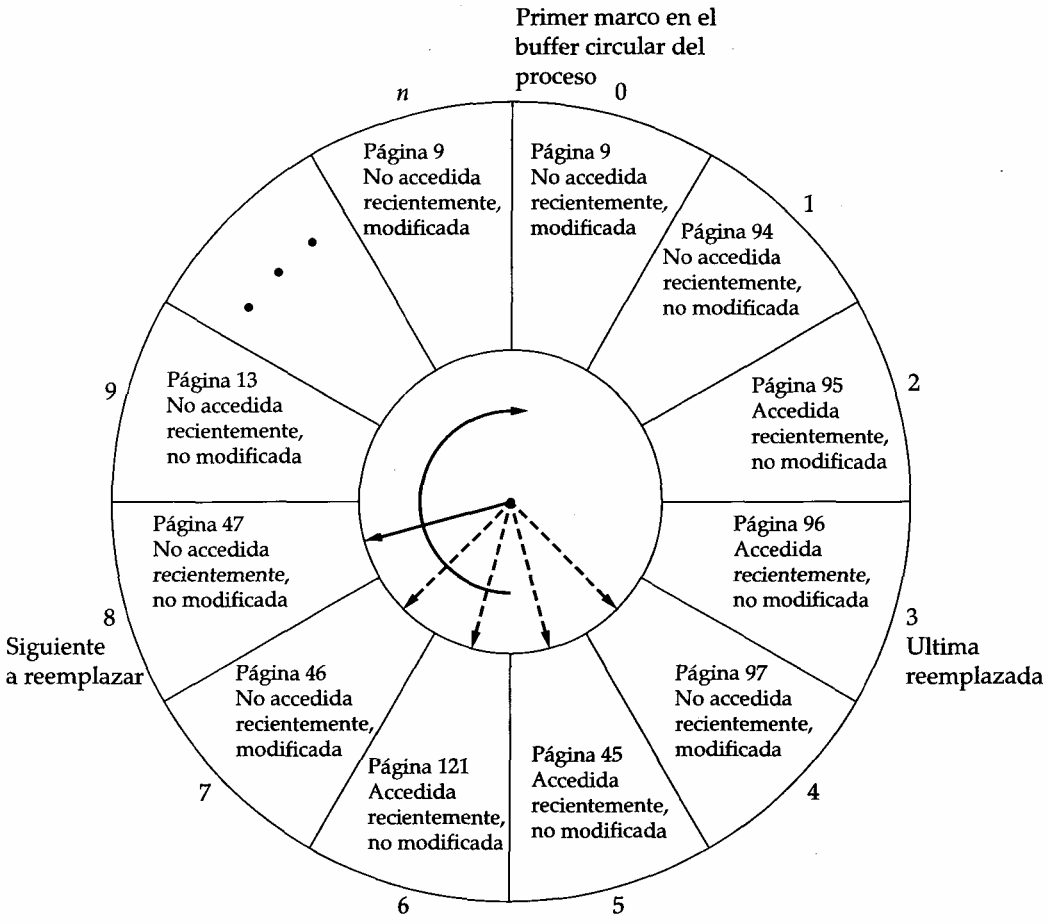


FIGURA 7.17 Algoritmo del reloj para reemplazo de páginas [GOLD89]

moría principal; en su lugar, se suprime su entrada en la tabla de páginas y se pone en la lista de páginas libres o modificadas.

La lista de páginas libres es una lista de marcos disponibles para cargar páginas. VMS intenta mantener un pequeño número de marcos libres en todo instante. Cuando se va a leer una página, se emplea el marco del principio de la lista, acabando con la página que estaba allí. Cuando se va a reemplazar una página no modificada, ésta permanece en memoria y su marco se añade al final de la lista de páginas libres. Del mismo modo, cuando se va a reescribir y reemplazar una página modificada, su marco se añade al final de la lista de páginas modificadas.

Lo importante de estas operaciones es que la página a reemplazar permanece en memoria. Así pues, si el proceso hace referencia a dicha página, se devuelve al conjunto residente del proceso con un coste pequeño. En realidad, las listas de páginas libres y modificadas actúan como una cache de páginas. La lista de páginas modificadas tiene otra función provechosa: Las páginas modificadas son reescritas por bloques, en vez de una a una. Esto reduce significativamente el número de operaciones de E/S y, por tanto, la cantidad de tiempo de acceso a disco.

Una versión sencilla del almacenamiento intermedio de páginas es la implementada en el sistema operativo Mach [RASH88]. En este caso, no se hace distinción entre páginas modificadas y sin modificar.

Políticas de reemplazo y tamaño de cache

Como se discutió anteriormente, el tamaño de la memoria es cada vez más grande y la cercanía en las aplicaciones cada vez menor. Para compensar, el tamaño de las caches es cada vez mayor. Actualmente, son alternativas de diseño factibles grandes tamaños de cache, incluso de varios megabytes [BORG90]. Con una cache grande, el reemplazo de páginas de memoria virtual puede tener un gran impacto en el rendimiento. Si el marco seleccionado para reemplazar está en la cache, entonces se pierde el bloque de cache, así como la página que contiene.

En sistemas que emplean alguna forma de almacenamiento intermedio de páginas, es posible mejorar el rendimiento de la cache sustituyendo la política de reemplazo de páginas por una política de ubicación de páginas en el buffer de páginas. La mayoría de los sistemas operativos sitúan las páginas seleccionando un marco arbitrario del buffer de páginas; normalmente, se emplea una disciplina FIFO. Un estudio expuesto en [KESS92] muestra que una estrategia cuidadosa de reemplazo de páginas puede dar como resultado de un 10 a un 20% menos de fallos de cache que la ubicación arbitraria.

En [KESS92] se examinan varios algoritmos de ubicación cuidadosa de páginas. Los detalles están más allá del alcance de este libro y dependen de los detalles de la estructura de la cache y de las políticas. La esencia de estas estrategias consiste en traer páginas consecutivas a memoria principal de forma que se minimice el número de marcos que se asignan a un mismo hueco de la cache.

Gestión del conjunto residente

Tamaño del conjunto residente

Con memoria virtual paginada no resulta necesario y, de hecho, puede no ser posible, traer todas las páginas de un proceso a memoria principal para preparar su ejecución. Así pues, el sistema operativo debe decidir cuántas páginas traer, es decir, cuánta memoria principal asignar a un determinado proceso. Aquí entran en juego varios factores:

- Cuanto menor es la cantidad de memoria asignada a un proceso, mayor es el número de procesos que pueden estar en memoria principal en cualquier instante. Esto aumenta la probabilidad de que el sistema operativo encuentre al menos un proceso Listo en cualquier instante dado y, por tanto, reduzca el tiempo perdido en el intercambio.
- Si en memoria principal hay un número relativamente pequeño de páginas de un proceso, entonces, a pesar del principio de cercanía, el porcentaje de fallos de página será algo mayor (ver figura 7.9b).
- Por encima de un determinado tamaño, la asignación de memoria adicional a un proceso en particular no tendrá efectos notables en el porcentaje de fallos de página para ese proceso, debido al principio de cercanía.

Con estos factores en mente, en los sistemas operativos actuales se pueden encontrar dos tipos de políticas. La política de **asignación fija** otorga a cada proceso un número fijo de pá-

ginas en las que ejecutar. Dicho número se decide en el instante de carga inicial (instante de creación del proceso) y puede estar determinado por el tipo de proceso (interactivo, por lotes, tipo de aplicación) o en función de las directrices del programador o del administrador del sistema. Con una política de asignación fija, cada vez que se produce un fallo de página en la ejecución de un proceso, se debe reemplazar una de las páginas de dicho proceso por la página que se necesite.

La política de **asignación variable** permite que el número de marcos asignados a un proceso cambie a lo largo de su vida. En el mejor de los casos, un proceso que está sufriendo de forma permanente un alto número de fallos de página, demostrando que el principio de cercanía apenas se cumple para él, recibirá marcos adicionales para reducir el porcentaje de fallos de página, mientras que un proceso con una tasa de fallos excepcionalmente baja, que demuestra que el proceso se comporta bastante bien desde el punto de vista de la cercanía, verá reducida su asignación, con la esperanza de que esto no aumentará demasiado su tasa de fallos. El uso de una política de asignación variable está relacionado con el concepto de alcance del reemplazo, como se explica más adelante.

La política de asignación variable parece ser la más potente. Sin embargo, la dificultad de este método está en que requiere que el sistema operativo evalúe el comportamiento de los procesos activos. Esto produce inevitablemente la necesidad de más software en el sistema operativo y es dependiente de los mecanismos de hardware ofrecidos por la plataforma del procesador.

Alcance del Reemplazo

El alcance de un reemplazo puede clasificarse en global o local. Ambos tipos de políticas son activadas por un fallo de página que se produce cuando no hay marcos libres. Para seleccionar la página a reemplazar, una **política de reemplazo local** escoge únicamente de entre las páginas residentes del proceso que originó el fallo de página. Una **política de reemplazo global** considera todas las páginas en memoria como candidatas para reemplazar, independientemente del proceso al que pertenezcan. Aunque las políticas locales son más fáciles de analizar, no hay ninguna evidencia de que se comporten mejor que las políticas globales, las cuales son atrayentes por su simplicidad de implementación y su mínimo coste [CARR84, MAEK87].

Existe una relación entre el alcance del reemplazo y el tamaño del conjunto residente (tabla 7.4). Un conjunto residente fijo implica una política de reemplazo local: Para mantener constante el tamaño del conjunto residente, una página suprimida de memoria principal debe reemplazarse por otra del mismo proceso. Una política de asignación variable puede, sin duda, emplear una política de reemplazo global: El reemplazo de una página de un proceso en memoria principal por otra provoca que la asignación de un proceso crezca en una página y la de otro disminuya también en una. También se verá que la asignación variable y el reemplazo local es una combinación válida. En el resto de este apartado, se examinarán estas tres combinaciones.

Asignación fija y alcance local

En este caso, se tiene un proceso que ejecuta en memoria principal con un número fijo de páginas. Cuando se produce un fallo de página, el sistema operativo debe elegir la página a

TABLA 7.4 Gestión del conjunto residente

	Reemplazo Local	Reemplazo Global
Asignación Fija	El número de marcos asignados a un proceso es fijo. La página a reemplazar se elige de entre los marcos asignados al proceso.	No es posible.
Asignación Variable	El número de marcos asignados a un proceso puede cambiar de un momento a otro para mantener su conjunto de trabajo. La página a reemplazar se elige de entre las páginas asignadas al proceso.	La página a reemplazar se elige de entre todos los marcos disponibles en memoria principal; esto hace que cambie el tamaño del conjunto residente de los procesos.

reemplazar entre las de dicho proceso que están actualmente en memoria. Se pueden usar algoritmos de reemplazo como los discutidos en el apartado anterior.

Con una política de asignación fija, es necesario decidir por anticipado la cantidad de memoria asignada a un proceso. Esta decisión puede hacerse en función del tipo de aplicación y de la cantidad solicitada por el programa. Las desventajas de esta solución son dos: Si la asignación tiende a ser demasiado pequeña, se producirá un alto porcentaje de fallos de página, haciendo que el sistema multiprogramado al completo funcione lentamente. Si la asignación tiende a ser innecesariamente grande, habrá muy pocos programas en memoria principal y el procesador estará desocupado un tiempo considerable o bien se consumirá un tiempo importante en intercambio.

Asignación variable y alcance global

Esta combinación es quizá la más sencilla de implementar y ha sido adoptada por un buen número de sistemas operativos. En un instante dado, en memoria principal habrá varios procesos, cada uno de ellos con un cierto número de marcos asignados. Normalmente, el sistema operativo también mantiene una lista de marcos libres. Cuando se produce un fallo de página, se añade un marco libre al conjunto residente del proceso y se carga la página. Así pues, los procesos que producen fallos de página incrementan gradualmente su tamaño, lo que ayuda a reducir el número global de fallos de página en el sistema.

La dificultad de este método está en la elección del reemplazo. Cuando no hay marcos libres, el sistema operativo debe elegir una página que esté en memoria para reemplazar. La selección se realiza entre todas las páginas de memoria, excepto los marcos bloqueados, como los del núcleo. Usando cualquiera de las políticas tratadas en el apartado anterior, la página elegida puede pertenecer a cualquier proceso residente; no hay ninguna disciplina que determine el proceso que debe perder una página de su conjunto residente. Es más, la selección del proceso que sufre la reducción en el conjunto residente puede que no sea óptima.

Una forma de contrarrestar los problemas potenciales de rendimiento de una política de asignación variable y alcance global es el almacenamiento intermedio de páginas. De esta manera, la elección de la página a reemplazar se hace menos significativa, ya que la página puede ser recuperada si se produce una referencia antes de que se escriba el bloque con el siguiente grupo de páginas.

Asignación variable y alcance local

Otro método de asignación variable intenta superar los problemas de la estrategia de alcance global. La estrategia de asignación variable y alcance local puede resumirse como sigue:

1. Cuando se carga un nuevo proceso en memoria, se le asigna cierto número de marcos en función del tipo de aplicación, las necesidades del programa u otros criterios. La asignación puede cubrirse tanto con paginación previa como con paginación por demanda.
2. Cuando se produce un fallo de página, se selecciona la página a reemplazar de entre las del conjunto residente del proceso que sufre el fallo.
3. De vez en cuando, se vuelve a evaluar la asignación otorgada al proceso y se aumenta o disminuye para mejorar el rendimiento global.

Con esta estrategia, se meditará la decisión de aumentar o disminuir el tamaño del conjunto residente y ésta se hará en función de una valoración de las posibles demandas futuras de los procesos activos. Debido a esta evaluación, la estrategia es más compleja que la simple política de reemplazo global. Sin embargo, puede obtenerse un rendimiento mejor.

Los elementos clave de la asignación variable con alcance local son los criterios empleados para determinar el tamaño del conjunto residente y el momento de los cambios. Una estrategia específica que ha recibido mucha atención en la bibliografía se conoce como **estrategia del conjunto de trabajo.** Si bien una estrategia pura del conjunto de trabajo sería difícil de implementar, puede ser útil como base para las comparaciones.

El conjunto de trabajo es un concepto presentado y popularizado por Denning [DENN68, DENN70, DENN80b]; este concepto ha tenido un gran impacto en el diseño de la gestión de la memoria virtual. El conjunto de trabajo de un proceso en un instante virtual t y con parámetro A, denotado por $W(t, \text{A})$, es el conjunto de páginas a las que el proceso ha hecho referencia en las últimas A unidades de tiempo virtual. Se usa el término "tiempo virtual" para indicar el tiempo que transcurre mientras que el proceso está realmente en ejecución. Se puede considerar el tiempo virtual medido por ciclos de instrucción, donde cada instrucción ejecutada es igual a una unidad de tiempo.

A continuación se examinan cada una de las dos variables de W. La variable A es una ventana de tiempo para la observación del proceso. El tamaño del conjunto de trabajo será una función no decreciente del tamaño de la ventana. La figura 7.18 ilustra estos resultados y muestra una secuencia de referencias a páginas para un proceso. Los puntos indican unidades de tiempo en las que no cambia el conjunto de trabajo. Nótese que, cuanto mayor es el conjunto de trabajo, menos frecuentes son los fallos de página. Este resultado puede expresarse mediante la siguiente relación:

$$W(t, \Delta + 1) \supseteq W(t, \Delta)$$

El conjunto de trabajo es también función del tiempo. Si un proceso ejecuta durante Δ unidades de tiempo y emplea sólo una página, entonces $|W(t, \Delta)| = 1$. Si se referencian rápidamente muchas páginas diferentes y si el tamaño de la ventana lo permite, el conjunto de trabajo puede también crecer hasta el número de páginas n del proceso. Así pues,

$$1 \leq |W(t, \Delta)| \leq \min(\Delta, n)$$

Secuencia de Referencias a Página	Tamaño de Ventana 2	3	4	5
24	24	24	24	24
15	15 24	15 24	15 24	15 24
18	18 15	18 15 24	18 15 24	18 15 24
23	23 18	23 18 15	23 18 15 24	23 18 15 24
24	24 23	24 23 18	•	•
17	17 24	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17	18 17 24	•	•
24	24 18	•	•	•
18	18 24	•	•	•
17	17 18	•	•	•
17	17	•	•	•
15	15 17	15 17 18	15 17 18 24	•
24	24 15	24 15 17	•	•
17	17 24	•	•	•
24	24 17	•	•	•
18	18 24	18 24 17	•	•

FIGURA 7.18 Conjunto de trabajo de un proceso definido por el tamaño de la ventana [BACH86]

La figura 7.19 muestra cómo puede variar el tamaño del conjunto de trabajo a lo largo del tiempo para un valor fijo de A. En muchos programas, se alternan periodos con un tamaño relativamente estable del conjunto de trabajo con otros periodos de cambios rápidos. Cuando un proceso comienza a ejecutarse por primera vez, construirá gradualmente un conjunto de trabajo, conforme hace referencias a páginas nuevas. Finalmente, por el principio de cercanía, el proceso debería estabilizarse en un determinado conjunto de páginas. Periodos transitorios posteriores reflejarán un desplazamiento del programa a una nueva ubicación. Durante la fase de transición, algunas de las páginas de la antigua ubicación permanecerán en la ventana A, originando un pico en el tamaño del conjunto de trabajo conforme se referencian páginas nuevas. A medida que la ventana se desplaza más allá de estas referencias, el tamaño del conjunto de trabajo desciende hasta contener únicamente las páginas de la nueva posición.

El concepto de conjunto de trabajo puede motivar la siguiente estrategia para el tamaño del conjunto residente:

1. Supervisar el conjunto de trabajo de cada proceso.
2. Eliminar periódicamente del conjunto residente de un proceso aquellas páginas que no pertenezcan a su conjunto de trabajo.
3. Un proceso puede ejecutarse sólo si su conjunto de trabajo está en memoria principal, esto es, si su conjunto residente incluye a su conjunto de trabajo.

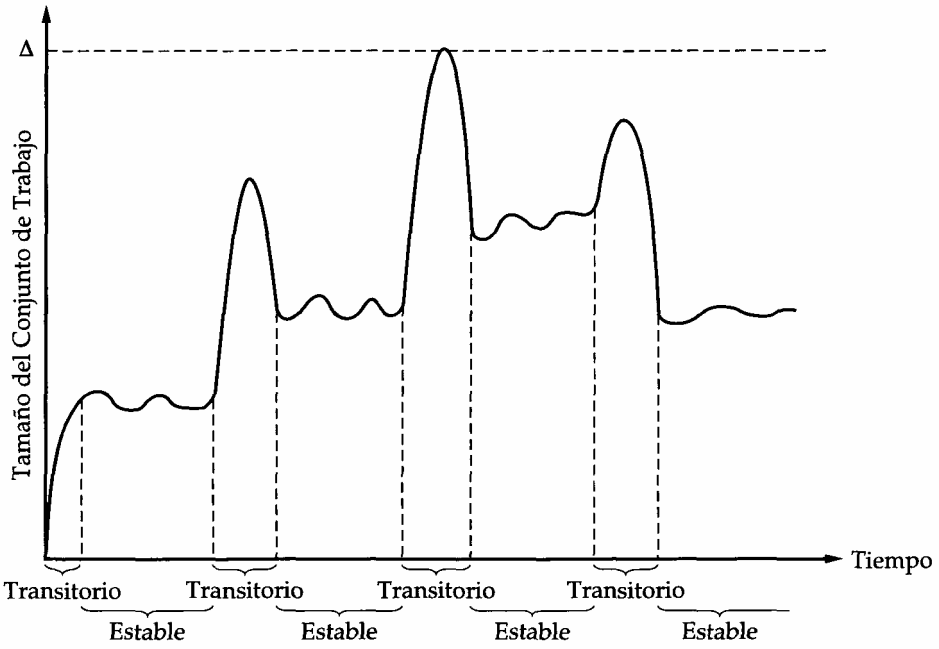


FIGURA 7.19 Gráfica típica del tamaño real de un conjunto de trabajo [MAEK87]

Esta estrategia resulta atractiva porque tiene en consideración un principio reconocido, el principio de cercanía y lo aprovecha para obtener una estrategia de gestión de memoria que permita minimizar los fallos de página. Por desgracia, la estrategia del conjunto de trabajo presenta una serie de problemas, como los siguientes:

1. El pasado no siempre predice el futuro. Tanto el tamaño como el contenido del conjunto de trabajo cambiarán con el tiempo (por ejemplo, véase la figura 5.28).
2. Es impracticable una medida real del conjunto de trabajo para cada proceso. Sería necesario marcar cada referencia de página de cada proceso con su tiempo virtual y mantener una cola ordenada en el tiempo de las páginas de cada proceso.
3. El valor óptimo de A es desconocido y, en cualquier, caso variable.

No obstante, es válido el espíritu de esta estrategia y, de hecho, un buen número de sistemas operativos intentan aproximarse a ella. Una forma de hacerlo es no centrándose en las referencias exactas a páginas, sino en el porcentaje de fallos de página de un proceso. Como ilustra la figura 7.9b, la tasa de fallos de página decrece a medida que aumenta el tamaño del conjunto residente. El tamaño del conjunto de trabajo debería decrecer en el punto de la curva marcado con una *W* en la figura. Por tanto, en lugar de supervisar directamente el tamaño del conjunto de trabajo, se pueden conseguir resultados equiparables supervisando la tasa de fallos de página. La línea de razonamiento es la siguiente: Si la tasa de fallos de página de un proceso es inferior a un umbral mínimo, el sistema al completo puede beneficiarse asignando un tamaño menor para el conjunto residente del proceso (puesto que hay disponibles más marcos de página para otros procesos) sin perjudicar al proceso (haciendo que incremente sus fallos de página). Si la tasa de fallos de página de un proceso es superior a un umbral máximo, el proceso puede utilizar un conjunto residente mayor (y tener menos fallos) sin degradar el sistema.

El algoritmo de **frecuencia de fallos de página** (PFF, *Page Fault Frequency)* sigue esta estrategia [CHU72, GUPT78]. El algoritmo exige que se asocie un bit de uso a cada página de memoria. El bit se pone a 1 cuando se accede a la página. Cuando se produce un fallo de página, el sistema operativo anota el tiempo virtual transcurrido desde el último fallo de página para ese proceso; esto puede hacerse, por ejemplo, manteniendo un contador de referencias a página. Se define un umbral *F.* Si el tiempo transcurrido desde el último fallo de página es menor que *F,* la página se añade al conjunto residente del proceso. En otro caso, se descartan todas las páginas con bit de uso a O y, en consecuencia, se reduce el conjunto residente. Al mismo tiempo, se restaura a O el valor del bit de uso en las páginas restantes. La estrategia puede mejorarse con el empleo de dos umbrales: uno superior que se usa para activar el crecimiento del tamaño del conjunto residente y otro inferior, que se usa para provocar una reducción.

El tiempo transcurrido entre fallos de página es el inverso de la tasa de fallos de página. Aunque podría parecer mejor mantener un promedio de la tasa de fallos de página, el empleo de una única medida de tiempo es un compromiso razonable que permite que las decisiones sobre el tamaño del conjunto residente se basen en la tasa de fallos de página. Si se complementa esta estrategia con el almacenamiento intermedio de páginas, se puede obtener un rendimiento bastante bueno.

No obstante, hay un defecto importante en el método PFF y es que no rinde de un modo adecuado durante los periodos transitorios, cuando se produce un desplazamiento a una nueva ubicación. Con PFF, ninguna página se retira del conjunto residente antes de hayan transcurrido *F* unidades de tiempo virtual desde su última referencia. Durante las transiciones entre ubicaciones, la rápida sucesión de fallos de página provoca que el conjunto residente de un proceso se infle antes de que se expulsen las páginas de la anterior ubicación; los picos repentinos de demanda de memoria pueden producir desactivaciones y reactivaciones innecesarias de procesos, con el correspondiente e indeseable coste por intercambio y descarga de procesos.

Un método que intenta solucionar el fenómeno de las transiciones entre ubicaciones con una pequeña sobrecarga, similar a la del PFF, es la política de **conjunto de trabajo muestreado en intervalos variables** (VSWS, *Variable-interval Sampled Working Set)* [FERR83]. La política VSWS evalúa el conjunto de trabajo de un proceso tomando muestras en función del tiempo virtual transcurrido. Al comienzo de un intervalo de muestreo, se restauran los bits de uso de todas las páginas residentes del-proceso. Al final, sólo las páginas que han sido referenciadas durante el intervalo tendrán sus bits de uso activos; estas páginas se mantienen en el conjunto residente del proceso durante el intervalo siguiente, descartándose el resto. Así pues, el tamaño del conjunto residente puede disminuir sólo al final de un intervalo. Durante cada intervalo, cualquier página fallida se añade al conjunto residente; de este modo, el conjunto residente permanece fijo o crece durante el intervalo.

La política del VSWS está gobernada por los siguientes parámetros:

M Duración mínima del intervalo de muestreo

L Duración máxima del intervalo de muestreo

Q Número de fallos de página permitidos entre cada par de muestras

La política del VSWS es la siguiente:

1. Si el tiempo virtual transcurrido desde la última muestra alcanza *L,* suspender el proceso y explorar los bits de uso.

2. Si, antes de que transcurra un tiempo virtual *L*, se producen *Q* fallos de página:

a) Si el tiempo virtual desde la última muestra es menor que *M*, esperar hasta que el tiempo virtual transcurrido alcance *M* para suspender el proceso y explorar los bits de uso.

b) Si el tiempo virtual desde la última muestra es mayor o igual que *M*, suspender el proceso y explorar los bits de uso.

Los valores de los parámetros deben seleccionarse de forma que el muestreo se active normalmente por el acontecimiento del Q-ésimo fallo de página después de la última exploración (caso 2b). Los otros dos parámetros (M y L) ofrecen unos límites para protección ante condiciones excepcionales. La política del VSWS intenta reducir los picos de demanda de memoria provocados por transiciones bruscas entre ubicaciones, incrementando la frecuencia de muestreo y, por tanto, la proporción de páginas no usadas que abandonan el conjunto residente cuando aumenta la tasa de paginación. Algunos experimentos realizados con esta técnica en el sistema operativo GCOS 8 de los computadores Bull indican que este método es tan sencillo de implementar como el PFF, pero más efectivo [PIZZ89].

Políticas de Vaciado

Una política de vaciado es la contraria a una política de lectura; se preocupa de determinar el momento en que hay que escribir en memoria secundaria una página modificada. Las dos alternativas más habituales son el vaciado por demanda y el vaciado previo. Con **vaciado por demanda,** una página se escribirá en memoria secundaria sólo cuando haya sido elegida para reemplazarse.

Una política de **vaciado previo** escribe las páginas modificadas antes de que se necesiten sus marcos, de forma que las páginas puedan escribirse por lotes.

Existe un peligro si se sigue estrictamente cualquiera de las dos políticas. Con el vaciado previo, una página se escribe pero permanece en memoria principal hasta que el algoritmo de reemplazo de páginas diga que se suprime. El vaciado previo permite escribir las páginas por lotes, pero tiene poco sentido escribir cientos o miles de páginas para encontrarse con que la mayoría de ellas han sido de nuevo modificadas antes de ser reemplazadas. La capacidad de transferencia de la memoria secundaria es limitada y no debe malgastarse con operaciones de vaciado innecesarias.

Por otro lado, en el vaciado por demanda, la escritura de una página modificada es anterior a la lectura de una nueva página. Esta técnica puede minimizar las escrituras de páginas, pero hace que un proceso que sufra un fallo de página pueda tener que esperar dos transferencias de página antes de desbloquearse. Esto puede disminuir el aprovechamiento del procesador.

Una solución mejor es incorporar almacenamiento intermedio de páginas, lo que permite la adopción de la siguiente política: Vaciar sólo las páginas que es posible reemplazar, pero desconectar las operaciones de vaciado y reemplazo. Con almacenamiento intermedio de páginas, las páginas reemplazadas pueden situarse en dos listas: modificadas y no modificadas. Las páginas de la lista de modificadas pueden escribirse periódicamente por lotes y trasladarse a la lista de no modificadas. Una página de la lista de no modificadas puede reclamarse, si se le hace de nuevo referencia o perderse, cuando se asigna su marco a otra página.

Control de carga

El control de carga consiste en determinar el número de procesos que pueden estar en memoria principal, lo que ha venido en llamarse grado de multiprogramación. La política de control de carga es crítica para la efectividad de la gestión de memoria. Si, en un instante dado, hay pocos procesos residentes en memoria, habrá muchas ocasiones en las que todos los procesos estén bloqueados y se consumirá mucho tiempo en el intercambio. Por otro lado, si hay demasiados procesos residentes, el tamaño medio del conjunto residente de cada proceso no será el adecuado y se producirán frecuentes fallos de página. El resultado es una situación conocida como *hiperpaginación (thrashing)*.

Grado de multiprogramación

El fenómeno de la hiperpaginación se ilustra en la figura 7.20. Cuando el grado de multiprogramación supera un pequeño valor, se podría esperar que la utilización del procesador aumentara, puesto que hay menos posibilidades de que todos los procesos estén bloqueados. Sin embargo, se alcanza un punto en el que el conjunto residente en promedio no es adecuado. En este punto, el número de fallos de página se eleva drásticamente y la utilización del procesador se desploma.

Hay varias formas de abordar este problema. Los algoritmos del conjunto de trabajo o de frecuencia de fallos de página incorporan implícitamente el control de carga. Sólo pueden ejecutar aquellos procesos cuyos conjuntos residentes sean suficientemente grandes. Dado un tamaño exigido para el conjunto residente de cada proceso activo, la política determina automática y dinámicamente el número de programas activos.

Otro método, propuesto por Denning y sus colaboradores [DENN80b], se conoce como el "criterio $L = 5$"', que ajusta el grado de multiprogramación de forma que el tiempo medio entre fallos sea igual al tiempo medio exigido para procesar un fallo de página. Algunos es-

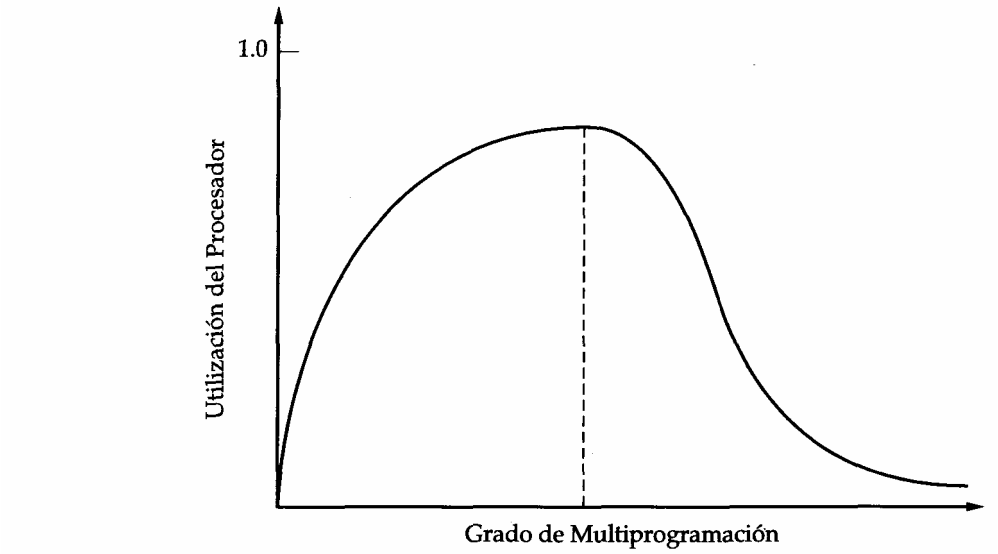


FIGURA 7.20 Efectos de la multiprogramación

tudios de rendimiento indican que éste es el punto en el que el uso del procesador alcanza un máximo. Una política con un efecto similar, propuesta en [LER076], es el "criterio del 50%", que intenta mantener el uso del dispositivo de paginación aproximadamente a un 50%. Otros estudios de rendimiento también indican que éste es un punto de máximo aprovechamiento del procesador.

Otro método consiste en adaptar el algoritmo del reloj para reemplazo de páginas descrito anteriormente (figura 7.17). [CARR84] describe una técnica de alcance global que incorpore una supervisión de la velocidad en la que el puntero recorre el buffer circular de marcos. Si la velocidad está por debajo de un umbral inferior determinado, es que se produce una de estas situaciones o ambas a la vez:

1. Se producen pocos fallos de página, lo que genera pocas peticiones de avance del puntero.
2. Para cada petición, el puntero explora un número medio de marcos pequeño, lo que indica que hay muchas páginas de memoria sin referenciar y que están listas para reemplazar.

En ambos casos, el grado de multiprogramación puede aumentarse con seguridad. Por otro lado, si la velocidad de exploración del puntero excede un umbral superior, quiere decirse que la tasa de fallos de página es alta o que es difícil encontrar páginas para reemplazar, lo que implica que el grado de multiprogramación es demasiado alto.

Suspensión de procesos

Si se va a reducir el grado de multiprogramación, deben suspenderse (descargarse) uno o más procesos actualmente residentes. [CARR84] enumera las seis posibilidades siguientes:

- *Procesos con la prioridad más baja:* Esta alternativa toma una decisión de política de planificación y no tiene que ver con las cuestiones de rendimiento.
- *Procesos con fallos de página:* El razonamiento es que hay una gran probabilidad de que las tareas que provocan fallos no tengan residente su conjunto de trabajo y, suspendiéndolas, el rendimiento pierda lo menos posible. Además, esta elección tiene resultados inmediatos, ya que bloquea un proceso que está a punto de ser bloqueado de cualquier modo y elimina así el coste de un reemplazo de página y de una operación de E/S.
- *Último proceso activado:* Este es el proceso con menos posibilidades de tener su conjunto de trabajo residente.
- *Proceso con el conjunto residente más pequeño:* Este es el proceso que necesita el menor esfuerzo futuro para volver a cargar el conjunto residente. Sin embargo, penaliza a los programas con ubicaciones pequeñas.
- *El proceso mayor:* Esta alternativa obtiene la mayor cantidad de marcos libres en una memoria muy ocupada, haciendo poco probables más desactivaciones en breve.
- *Procesos con la mayor ventana de ejecución restante:* En la mayoría de los esquemas de planificación del procesador, un proceso puede ejecutarse sólo durante cierto espacio de tiempo antes de ser interrumpido y puesto al final de la cola de Listos. Este es un enfoque parecido a la disciplina de planificación de "primero el proceso más corto".

Como en muchos otros campos del diseño de sistemas operativos, la política a elegir es una cuestión de juicio y depende de muchos otros factores de diseño del sistemas operativo, así como de las características de los programas que se ejecuten.

7.3

EJEMPLOS DE GESTIÓN DE MEMORIA

Sistema/370 y MVS

Estructura del espacio de direcciones

El Sistema/370 de IBM emplea una estructura de memoria a dos niveles y se refiere a estos dos niveles como segmentos y páginas, si bien el método de segmentación carece de muchas de las características descritas anteriormente en este capítulo. En la arquitectura 370 básica, el tamaño de página puede ser de 2KB o 4KB y el tamaño del segmento es fijo, de 64KB o 1MB. En las arquitecturas 370/XA y 370/ESA, el tamaño de la página es de 4KB y el del segmento de 1MB. Este estudio hará referencia principalmente a la estructura XA-ESA.

La figura 7.21a representa el formato de una dirección virtual. El índice de byte especifica un byte dentro de los 4KB de una página, el índice de página especifica una de las 256 páginas dentro de un segmento y el índice de segmento identifica uno de los 2048 segmentos visibles para el usuario. Así pues, el usuario contempla un espacio de direcciones virtual de 2^{31} bytes (2GB) de almacenamiento.

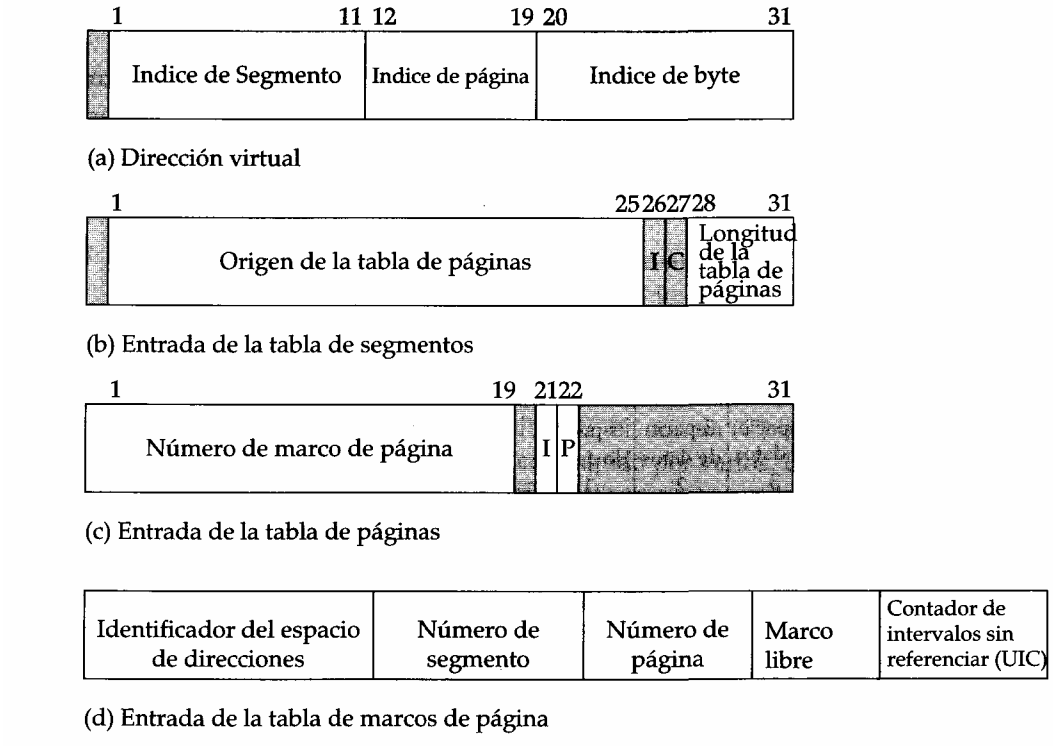


FIGURA 7.21 Formatos de la gestión de memoria de los sistemas IBM 370/XA y 370/ESA

La figura 7.22 muestra la evolución de la estructura del espacio de direcciones en la arquitectura 370. La familia 370 original, que da soporte al sistema operativo MVS original, emplea direcciones de 24 bits, permitiendo un espacio de direcciones virtual de 16MB. Así se pueden asignar 16MB de memoria virtual a cualquier trabajo o aplicación. Parte del espacio de direcciones se reserva para programas y estructuras de datos de MVS. Dentro de un espacio de direcciones dado, pueden estar activas una o más tareas. Como se mencionó anteriormente, esto se corresponde con el concepto de múltiples hilos dentro de un mismo proceso.

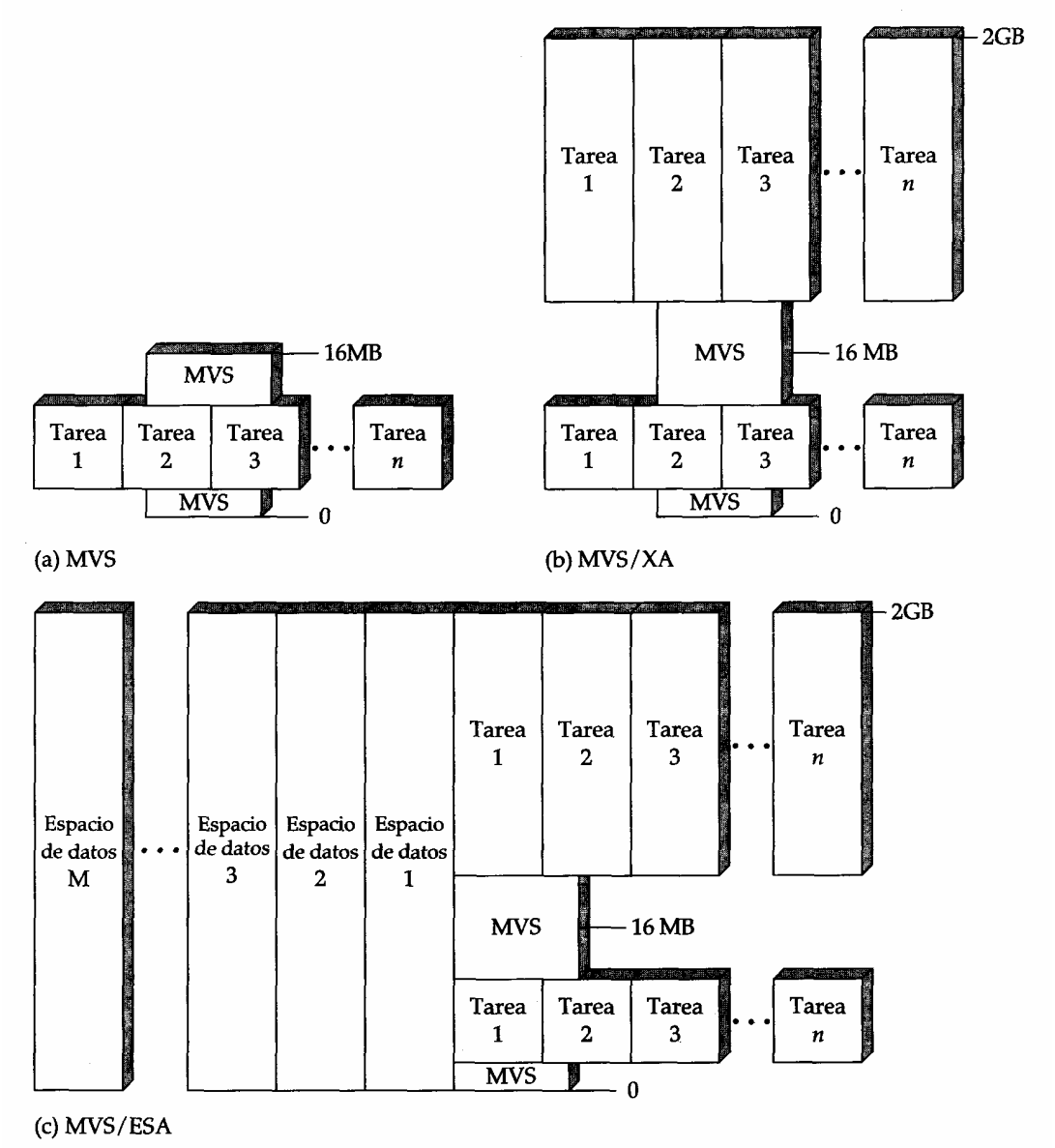


FIGURA 7.22 Estructura del espacio de direcciones de MVS

Finalmente, IBM descubrió que había dos razones por las que la estructura de direcciones de 24 bits no era adecuada para muchos clientes y aplicaciones. En primer lugar, las aplicaciones comenzaban a ser más complejas y necesitaban emplear estructuras de datos mayores. En segundo lugar, el crecimiento de MVS fue tal que llegó a ocupar de la mitad a tres cuartas partes del espacio de direcciones virtual del usuario[4]. El resultado fue que el espacio virtual de direcciones disponible se reducía a medida que se expandían las exigencias del usuario. Así pues, IBM modificó la arquitectura básica para ofrecer direcciones extendidas (XA, *extended addresing)*. La arquitectura resultante se denomina Sistema 370/XA, que da soporte al MVS/XA. Esta arquitectura soporta el esquema de direcciones de 31 bits representado en la figura 7.21a. Cada aplicación o cada trabajo tiene ahora 2GB disponibles de espacio de direcciones virtual. Este espacio se divide en dos regiones separadas por la barrera de las 16MB. El espacio dedicado a MVS permanece por debajo de esta barrera y el resto está disponible para uno o más programas. Por encima, también se reserva algún espacio adicional para MVS, con el resto disponible para los programas cuyas exigencias excedan de la cantidad disponible por debajo de la barrera. De este modo, una única tarea puede emplear espacio de usuario por encima y por debajo de la barrera.

La última versión de la arquitectura del Sistema 370 y el MVS correspondiente se denomina Arquitectura de Sistemas Empresariales (ESA, *Enterprise System Architecture)* [PLAM89], que representa un paso adelante de IBM para satisfacer las necesidades aparentemente insaciables de almacenamiento virtual de las aplicaciones a gran escala. El Sistema 370/ESA emplea la misma estructura de direcciones que XA, ofreciendo un espacio de direcciones de 2GB para dar soporte a programas y datos de MVS y a programas de usuario. Además, un programa puede direccionar hasta 15 espacios adicionales de 2GB sólo para datos, obteniéndose un espacio total de direcciones virtual para una única aplicación de 16 Terabytes (TB) (2^{44} bytes).

Estructuras de Control y Hardware

Para gestionar el almacenamiento, se emplean tablas de segmentos y de páginas, como se indica en la figura 7.11. Hay una tabla de segmentos por cada espacio de direcciones virtual. Así pues, cada espacio de direcciones virtual contiene varios segmentos, con una entrada para cada uno en la tabla de segmentos.

El formato de una entrada de la tabla de segmentos se muestra en la figura 7.21b. Los 25 bits de origen de la tabla de páginas señalan a la tabla de páginas para ese segmento; al campo de 25 bits se añaden seis bits puestos a 0 para formar una dirección real de 31 bits. La longitud de la tabla de páginas (LPT) indica el número de entradas en la tabla (= 16 x LPT). El bit de segmento invalidado (I) se activa cuando se produce algún funcionamiento incorrecto o error irrecuperable e indica que el segmento no está disponible. El bit común (C) indica si el segmento es privado para un espacio de direcciones o está compartido por varios. En este último caso, las entradas de las tablas de páginas y de segmentos relativas a este segmento pueden permanecer en el buffer de traducción adelantada después de descargar el espacio de direcciones.

Cada entrada de la tabla de páginas (figura 7.2lc) incluye un bit de no validez (I) que indica si la página correspondiente está en memoria real. Si es así, el número de marco proporciona la ubicación de la página; para formar la dirección real se añaden 12 bits a la derecha de este número. El bit de protección (P), cuando está activo, impide las operaciones de Escritura, pero permite las de Lectura de la página correspondiente.

[4] Esto fue así incluso a pesar de que la mayor parte de MVS no estaba ubicado en el espacio virtual de direcciones del usuario.

En el 370/XA, la traducción de direcciones se lleva a cabo de la forma representada en la figura 7.11. Además, un buffer de traducción adelantada mantiene los segmentos referenciados recientemente y las entradas de la tabla de páginas.

En el 370/ESA hace falta una lógica adicional. En el conjunto de instrucciones del 370, las referencias a memoria virtual se hacen a través de registros de propósito general (GPR, *General Purpose Registers).* Hay un registro de acceso (AR, *Address Register)* asociado a cada uno de los 16 registros de propósito general de 32 bits del 370. Cuando se usa un GPR como registro base para la ubicación de un operando, el AR correspondiente especifica el espacio de direcciones en el cual se encuentra el operando. De este modo, es posible asociar distintos espacios de direcciones virtuales con cada GPR. Cambiando el contenido de un AR, puede accederse a muchos espacios de direcciones virtuales diferentes.

La opción ESA se activa por medio de un bit en la palabra de estado del programa. Cuando está utilizándose, hacen falta dos tipos de traducción para calcular una dirección real. La figura 7.23 ilustra este mecanismo. En el ejemplo se emplea un tipo de instrucción del 370 para indexar. El primer operando es un registro. El segundo operando es una posición virtual referenciada por la suma de un desplazamiento de 12 bits más el contenido de un registro base y un registro índice, ambos procedentes de la reserva de registros de propósito general. El registro base de designación (B2) de 4 bits también se usa para hacer referencia a uno de los 16 registros de espacio de direcciones. El contenido de este registro de espacio de direcciones se emplea como entrada a un proceso de traducción que involucra a varias tablas y genera la dirección real de la tabla de segmentos para un espacio de direcciones en particular. Esta tabla, más la dirección virtual, sirve como entrada al mecanismo de traducción dinámica de direcciones (figura 7.11), que genera por último la dirección real.

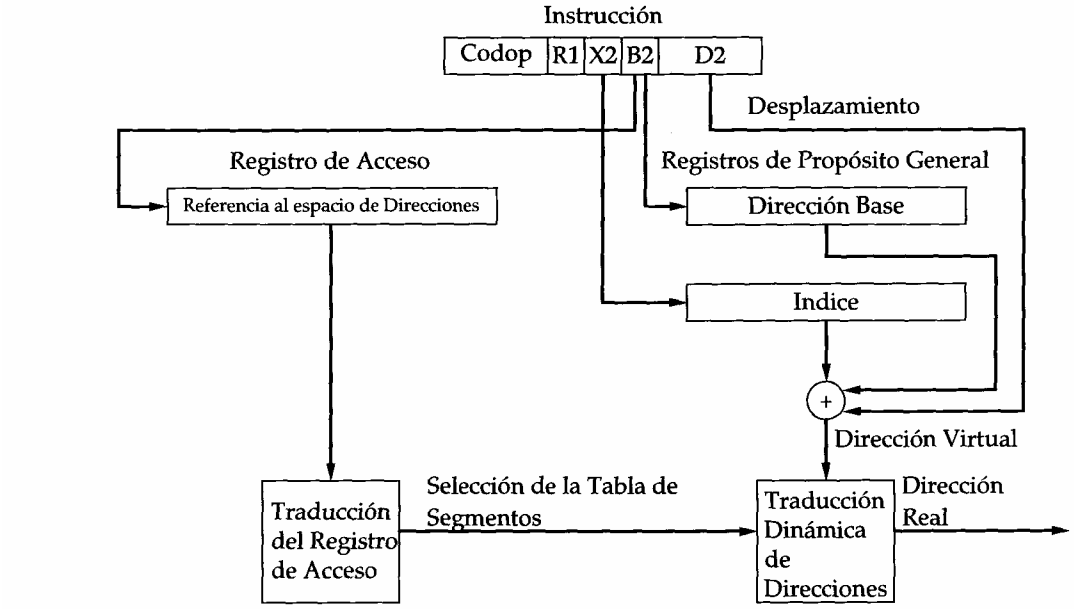


FIGURA 7.23 Traducción de direcciones del 370/ESA [PLAM89]

Consideraciones del sistema operativo

MVS hace uso de las características del hardware descritas para la gestión de memoria. MVS emplea una estrategia global de reemplazo de páginas. El sistema mantiene una lista de marcos de página disponibles. Cuando se carga una nueva página, se usa un marco de esta lista. Cuando el número de marcos disponibles está por debajo de un determinado umbral, el sistema operativo lleva a cabo una operación de robo de página, convirtiendo una serie de páginas activas en páginas disponibles. La decisión de robar una página en particular se toma en función del historial de actividad de cada página que reside actualmente en un marco de almacenamiento real. Las páginas a las que no se ha accedido durante un tiempo relativamente largo son buenas candidatas para el robo de página. Básicamente, se sigue la política de la usada hace más tiempo.

Para implementar esta política, MVS hace uso de dos fuentes de información: una clave de almacenamiento asociada a cada marco y una tabla de marcos. La clave de almacenamiento es un campo de control asociado con cada marco de memoria real. Dos bits de dicha clave que son relevantes para el reemplazo de páginas son el bit de referencia y el de modificación. El bit de referencia se pone a 1 cuando se accede a alguna dirección dentro del marco para Lectura o Escritura y se pone a 0 cuando se carga una página nueva en el marco. El bit de modificación se pone a 1 cuando se realiza una operación de Escritura en cualquier posición dentro del marco. La tabla de marcos contiene una entrada por cada marco de memoria real. Cada entrada contiene los siguientes campos (figura 7.21d):

• *Identificador del espacio de direcciones:* Identifica el espacio de direcciones propietario del marco.

• *Número de segmento, número de página:* Página virtual que ocupa el marco.

• *Marco disponible:* Indica si el marco está disponible u ocupado.

• *Contador de intervalos sin referenciar (UIC, Unreferenced Interval Count):* Indica cuánto tiempo ha pasado desde que un programa hizo referencia al marco.

Una vez por segundo, MVS comprueba el bit de referencia para cada marco de la memoria real. Si el bit de referencia no está activo (el marco no ha sido referenciado), el sistema incrementa el UIC de dicho marco. Si el bit de referencia está activo, MVS restaura el bit de referencia y pone el UIC a 0. Cuando hace falta robar páginas, MVS selecciona para reemplazar aquellos marcos cuyo UIC sea mayor. Esta política de reemplazo puede modificarse por medio de una técnica conocida como *aislamiento de almacenamiento,* que exige que cada espacio de direcciones mantenga un número mínimo de páginas específico. Así pues, puede ser que no se robe un marco de un espacio de direcciones si el robo provoca que el número de páginas caiga por debajo del mínimo de ese espacio de direcciones.

Windows NT

Windows NT fue diseñado para funcionar en varios procesadores. Una de las plataformas más importantes para Windows NT es la Intel 486, que es parte de una familia de procesadores que se han utilizado en los PC de IBM y compatibles. Windows adopta un tamaño de página de 4KB que es el usado en el 486 como base del esquema de memoria virtual. En este apartado, se examinarán las características de memoria virtual disponibles en el 486.

Espacios de direcciones

El 80386 incorpora hardware tanto para segmentación como para paginación. Ambos mecanismos pueden inhabilitarse, lo que permite al usuario elegir entre cuatro modelos distintos de la memoria:

• *Memoria no paginada y no segmentada:* En este caso, la dirección virtual es la misma que la dirección física, lo cual es útil, por ejemplo, en aplicaciones de control de poca complejidad y alto rendimiento.

• *Memoria paginada no segmentada:* La memoria se ve como un espacio de direcciones lineal paginado. La paginación lleva a cabo la protección y la gestión de memoria. Este modelo de memoria es el elegido por algunos sistemas operativos, como UNIX.

• *Memoria segmentada y no paginada:* Ahora la memoria se contempla como un conjunto de espacios de direcciones lógicas. Las ventajas de este modelo sobre un método de paginación consisten en que proporciona protección a nivel de byte, si se necesita. Es más, a diferencia de la paginación, garantiza que la tabla de traducción necesaria (la tabla de segmentos) estará en el chip del procesador cuando el segmento esté en memoria. Por tanto, la memoria segmentada no paginada genera tiempos de acceso predecibles.

• *Memoria paginada y segmentada:* La segmentación se usa para definir particiones lógicas de memoria sometidas a control de acceso y la paginación se usa para gestionar la asignación de memoria dentro de las particiones. Algunos sistemas operativos, como OS/2, se han decantado por este modelo.

Segmentación

Cuando se emplea segmentación, cada dirección virtual (llamada dirección lógica en la documentación del 80386) está formada por una referencia de 16 bits a un segmento y un desplazamiento de 32 bits. Dos bits de la referencia al segmento se encargan del mecanismo de protección, dejando 14 bits para especificar un segmento en particular. Así pues, con memoria no segmentada, la memoria virtual del usuario es de 2^{32} = 4GB. Con memoria segmentada, el espacio total de memoria virtual visible por el usuario es de 2^{46} = 64TB. El espacio de direcciones físicas emplea direcciones de 32 bits, con un máximo de 4GB.

La cantidad de memoria virtual puede ser en realidad mayor de 64TB: La interpretación de una dirección virtual que hace el 80386 depende de qué proceso está activo en un momento dado. La mitad del espacio virtual de direcciones (8K segmentos x 4GB) es global, compartida por todos los procesos; el resto es local y distinta para cada proceso.

Asociado a cada segmento hay dos formas de protección: el nivel de privilegio y el atributo de acceso. Hay cuatro **niveles de privilegio,** desde el más protegido (nivel 0) al menos protegido (nivel 3). El nivel de privilegio asociado a un segmento de datos es su *clasificación;* el nivel de privilegio asociado a un segmento de programa es su *acreditación.* Un programa en ejecución puede acceder a un segmento de datos si su acreditación es menor (más privilegiado) o igual (el mismo privilegio) que el nivel de privilegio del segmento de datos.

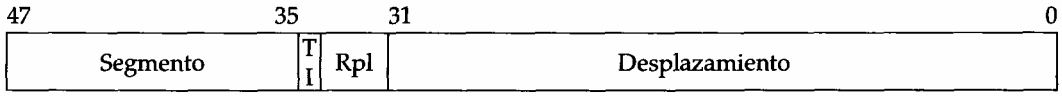
El hardware no dicta cómo se van a usar estos niveles de privilegio; su uso depende del diseño y la implementación del sistema operativo. Se pretendió que la mayor parte del sistema operativo usara el nivel de privilegio 1 y una pequeña parte destinada a la gestión de memoria, protección y control de acceso usara el nivel 0, dejando dos niveles para las aplicaciones. En muchos sistemas, las aplicaciones residen en el nivel 3, mientras que el nivel 2 no se

usa. Unos buenos candidatos para el nivel 2 son algunos subsistemas especializados de aplicación que deben estar protegidos debido a que implementan sus propios mecanismos de seguridad. Algunos ejemplos son los sistemas gestores de bases de datos, sistemas de automatización de oficinas y entornos de ingeniería del software.

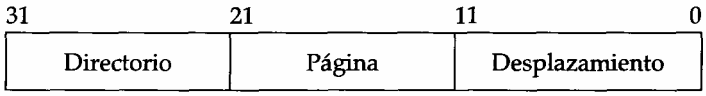
Además de regular el acceso a los segmentos de datos, los mecanismos de privilegio limitan el uso de ciertas instrucciones. Algunas instrucciones, como aquellas que se ocupan de los registros de gestión de memoria, pueden ejecutarse sólo en nivel 0. Las instrucciones de E/S sólo pueden ejecutarse hasta un determinado nivel designado por el sistema operativo; normalmente, éste es el nivel 1.

El **atributo de acceso** de un segmento de datos determina si se permiten accesos de sólo Lectura o de Lectura/Escritura. Para los segmentos de programa, el atributo de acceso especifica acceso de sólo Lectura o de Lectura/Ejecución.

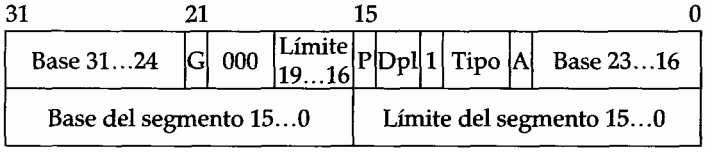
El mecanismo de traducción de direcciones de la segmentación implica la transformación de una **dirección virtual** en lo que se denomina una **dirección lineal.** El formato de la dirección virtual (figura 7.24a) incluye los siguientes campos:



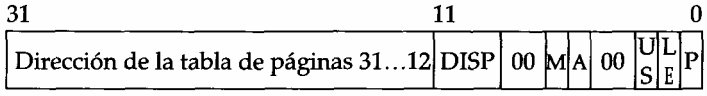
(a) Dirección Virtual



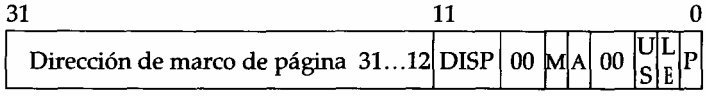
(b) Dirección lineal



(c) Entrada de la tabla de segmentos



(d) Entrada de directorio de la tabla de páginas



(e) Entrada de la tabla de páginas

FIGURA 7.24 Formatos de la gestión de memoria del Intel 486

• *Indicador de tabla (TI, Table Indicator):* Indica si debe utilizarse la tabla de segmentos global o una local en la traducción.

• *Número de segmento:* Sirve como índice en la tabla de segmentos.

• *Desplazamiento:* El desplazamiento del byte direccionado dentro del segmento.

• *Nivel solicitado de privilegio (RPL, Requested Privilege Level):* El nivel de privilegio pedido para el acceso.

Cada entrada de la tabla de segmentos consta de 64 bits, como se muestra en la figura 5.33c. Los campos se definen en la tabla 7.5.

Paginación

La segmentación es un servicio opcional y puede inhabilitarse. Cuando se usa segmentación, las direcciones del programa son direcciones virtuales y se convierten en direcciones lineales, tal y como se ha descrito. Cuando no se usa segmentación, el programa emplea direcciones lineales. En ambos casos, el paso siguiente consiste en convertir estas direcciones lineales en direcciones reales de 32 bits.

Para comprender la estructura de las direcciones lineales, hace falta saber que el mecanismo de paginación del 80386 es en realidad una operación de búsqueda a dos niveles en una tabla. El primer nivel es un **directorio de páginas,** que puede contener hasta 1024 entradas. Este directorio divide el espacio lineal de memoria de 4GB en 1024 grupos de páginas, cada uno con su propia **tabla de páginas** y con 4MB de longitud. Cada tabla de páginas contiene hasta 1024 entradas; cada entrada corresponde a una única página de 4KB. La ges-

TABLA 7.5 Parámetros de gestión de memoria del Intel 486

Entrada de la Tabla de Segmentos
Límite Define el tamaño del segmento. El procesador interpreta el campo límite de una de dos formas, dependiendo del bit de granularidad: en unidades de un byte, hasta un límite de 1MB o en unidades de 4KB, hasta un límite de 4GB.
Base Define la dirección de comienzo del segmento dentro del espacio de direcciones lineal de 4GB.
Bit de acceso (A) Activo si se ha accedido al segmento. Un sistema operativo que use memoria segmentada y no paginada puede usar este bit para supervisar la frecuencia de uso de segmentos con el fin de gestionar la memoria. En un sistema paginado, este bit se ignora.
Tipo Distingue entre varias clases de segmentos e indica los atributos de acceso.
Nivel de privilegio del descriptor (DPL, Descriptor Privilege Level) Especifica el nivel de privilegio del segmento al que se refiere la entrada de la tabla de segmentos (descriptor del segmento).
Bit de presencia del segmento (P) Se usa en sistemas no paginados. Indica si el segmento está presente en memoria principal. Para sistemas paginados, este bit está siempre a 1.
Bit de granularidad (G) Indica si el campo Límite se interpreta en unidades de 1 byte o de 4KB.

TABLA 7.5 Parámetros de gestión de memoria del Intel 486 (continuación)

Entrada al Directorio de la Tabla de Páginas y Entrada a la Tabla de Páginas
Dirección del Marco de Página Proporciona la dirección física de la página en memoria si el bit de presencia está activo. Puesto que los marcos están alineados en límites de 4K, los 12 bit inferiores están a cero y sólo los 20 bits superiores están incluidos en la entrada.
Dirección de la Tabla de Páginas Proporciona la dirección física de una tabla de páginas en memoria si está activo el bit de presencia.
Bit de presencia (P) Indica si la tabla de páginas o la página están en memoria principal.
Bit de acceso (A) El procesador pone a 1 este bit en los dos niveles de las tablas de páginas cuando se produce una operación de Lectura o Escritura sobre la página correspondiente.
Bit de modificación (M) El procesador pone a 1 este bit cuando se produce una operación de Escritura sobre la página correspondiente.
Bit de usuario/supervisor (US) Indica si la página está disponible sólo para el sistema operativo (nivel supervisor) o si lo está para el sistema operativo y las aplicaciones (nivel de usuario).
Bit de lectura/escritura (LE) Para páginas del nivel de usuario, indica si la página tiene acceso sólo para Lectura o para Lectura/Escritura.
Bits disponibles (DISP) Disponibles para los programadores de sistemas.

tión de memoria tiene las posibilidades de usar un mismo directorio de páginas para todos los procesos, un directorio de páginas para cada proceso o una combinación de ambas. El directorio de páginas para la tarea actual siempre está en memoria principal. Las tablas de páginas pueden estar en memoria virtual.

La figura 7.24 muestra el formato de las entradas de los directorios y de las tablas de páginas; los campos se definen en la tabla 7.5. Nótese que los mecanismos de control de acceso pueden darse en función de una página o un grupo de páginas.

El 80386 hace uso de un buffer de traducción adelantada en el que caben 32 entradas de la tabla de páginas. Cada vez que se cambia el directorio de páginas, se borra el buffer.

La figura 7.25 ilustra la combinación de los mecanismos de segmentación y paginación. Por claridad, no se muestran los mecanismos del buffer de traducción adelantada y de la cache de memoria.

Sistema UNIX, versión V

Puesto que UNIX se ideó para ser independiente de la máquina, su esquema de gestión de memoria varía de un sistema a otro. Las primeras versiones de UNIX simplemente empleaban particiones variables sin ningún esquema de memoria virtual. La mayor parte de las implementaciones actuales utilizan memoria virtual paginada.

Para la memoria virtual paginada, UNIX V hace uso de una serie de estructuras de datos que, con pequeñas modificaciones, son independientes de la máquina (figura 7.26 y tabla 7.6):

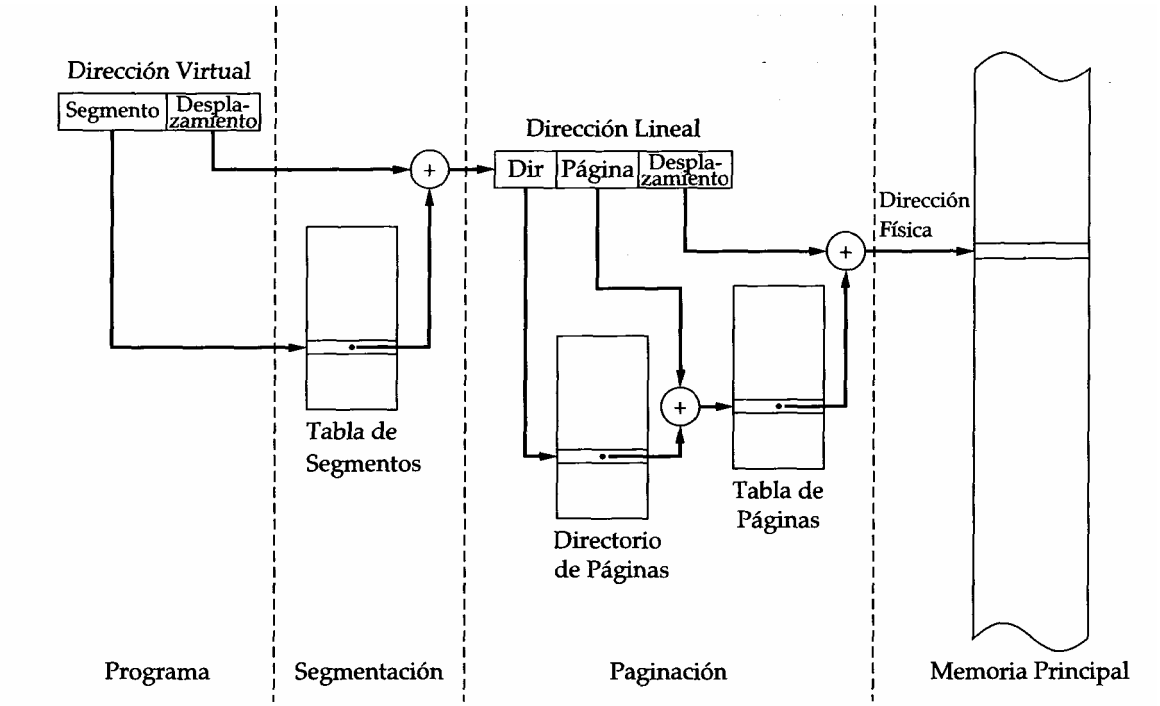


FIGURA 7.25 Mecanismo de traducción de direcciones de memoria del Intel 486

• *Tabla de páginas:* Normalmente, hay una tabla por proceso, con una entrada para cada página de la memoria virtual del proceso.

• *Descriptor de bloques de disco:* Asociado a cada página de un proceso hay una entrada en la tabla que describe la copia en disco de la página virtual.

• *Tabla de marcos de página:* Describe cada marco de la memoria real y está indexada por el número de marco.

• *Tabla de intercambios:* Hay una tabla de utilización del intercambio por cada dispositivo de intercambio, con una entrada para cada página de dicho dispositivo.

La mayoría de los campos definidos en la tabla 7.6 son autoexplicativos. Hace falta alguna explicación adicional. El campo Envejecimiento de la entrada de la tabla de páginas es similar al campo UIC empleado por MVS. Sin embargo, el número de bits y la frecuencia de actualización de este campo son dependientes de la implementación. Por tanto, en UNIX no hay un uso universal de este campo para la política de reemplazo de páginas.

El campo Tipo de Almacenamiento del descriptor de bloques de disco es necesario por la razón siguiente. Cuando un archivo ejecutable se usa por vez primera en la creación de un nuevo proceso, sólo podrá cargarse en memoria real una parte del programa y los datos del archivo. Después, cuando se produzca un fallo de página, se cargarán nuevos fragmentos del programa y los datos. Sólo se crean páginas de memoria virtual y se asignan a posiciones de alguno de los dispositivos de intercambio en el instante de la primera carga.

TABLA 7.6 Parámetros de gestión de memoria del Sistema UNIX, versión V

Entrada de la Tabla de Páginas

Número de marco de página
Se refiere a un marco en la memoria real.

Envejecimiento
Indica cuánto tiempo ha estado la página en memoria sin ser referenciada. La longitud y el contenido de este campo son dependientes del procesador.

Copia en escritura
Activo cuando la página es compartida por más de un proceso. Si uno de los procesos escribe en la página, debe hacerse primero una copia de la página para todos los demás procesos que la comparten. Esta característica permite aplazar la operación de copia hasta que sea necesario y evitarla en caso de que no resulte necesario.

Modificación
Indica que la página ha sido modificada.

Referencia
Indica que la página ha sido referenciada. Este bit se pone a cero cuando la página se carga por primera vez y puede ser restaurado periódicamente por el algoritmo de reemplazo de página.

Validez
Indica que la página está en memoria principal.

Protección
Indica si está permitida la operación de Escritura.

Descriptor de Bloques de Disco

Número de dispositivo de intercambio
Número del dispositivo lógico secundario que contiene la página correspondiente. Se puede usar más de un dispositivo para intercambio.

Número de bloque del dispositivo
Posición del bloque que contiene la página en el dispositivo de intercambio.

Tipo de almacenamiento
El almacenamiento puede ser la unidad de intercambio o un archivo ejecutable. En este último caso, hay un indicador de si debe borrarse primero la memoria virtual a asignar.

Entrada de la Tabla de Marcos de Página

Estado de la página
Indica si el marco está disponible o tiene una página asociada. En este último caso, se especifica la situación de la página: en el dispositivo de intercambio, en un archivo ejecutable o en proceso de DMA.

Contador de referencias
Número de procesos que hacen referencia a la página.

Dispositivo lógico
Dispositivo lógico que contiene una copia de la página.

Número de bloque
Posición del bloque con la copia de la página en el dispositivo lógico.

Puntero a marco de página
Puntero a otra entrada en la tabla de marcos para mantener una lista de páginas libres y una cola de dispersión de páginas.

Entrada de la Tabla de Utilización del Intercambio

Contador de referencias
Número de entradas de la tabla de páginas que señalan a páginas situadas en el dispositivo de intercambio.

Número de página/unidad de almacenamiento
Identificador de la página en la unidad de almacenamiento.

Número de Marco	Envejecimiento	Copia en escritura	Modificación	Referencia	Validez	Protección

(a) Entrada de la tabla de páginas

Número de Dispositivo de Intercambio	Número de Bloque de Dispositivo	Tipo de Almacenamiento

(b) Descriptor de bloques de disco

Estado de la Página	Contador de Referencias	Dispositivo Lógico	Número de Bloque	Puntero a Marco de Página

(c) Entrada de la tabla de marcos de página

Contador de Referencias	Número de Página/Unidad de almacenamiento

(a) Entrada de la tabla de intercambios

FIGURA 7.26 Formatos de gestión de memoria del Sistema UNIX, versión V

En ese momento, se le informa al sistema operativo si hace falta limpiar (poner a 0) las posiciones en el marco de página antes de cargar por primera vez un bloque de programa o de datos.

La tabla de marcos de página se utiliza en el reemplazo de páginas. Se emplean varios punteros para crear listas dentro de esta tabla. Todos los marcos disponibles se encadenan en una lista de marcos libres utilizada para traer páginas. Cuando el número de páginas disponibles baja de un cierto umbral, el núcleo robará un conjunto de páginas para compensar. Todas las páginas que están en el mismo dispositivo de intercambio y en el mismo bloque del dispositivo se encadenan juntas en una lista denominada *cola de dispersión.* El término *dispersión* se usa para referirse al hecho de que el identificador de la lista se obtiene directamente del número de dispositivo y de bloque. Esta lista es útil para bloquear operaciones de E/S de páginas.

7.4 RESUMEN

Para un aprovechamiento eficiente del procesador y de los servicios de E/S es conveniente mantener tantos procesos en memoria principal como sea posible. Además, conviene liberar a los programadores de las limitaciones de tamaño en el desarrollo de programas.

La forma de abordar ambos problemas es por medio de la memoria virtual. Con memoria virtual, todas las referencias a direcciones son referencias lógicas que se traducen a direcciones reales durante la ejecución. Esto permite a los procesos situarse en cualquier posición de memoria principal y cambiar de ubicación a lo largo del tiempo. La memoria virtual permite también dividir un proceso en fragmentos. Estos fragmentos no tienen por qué estar situados de forma contigua en la memoria principal durante la ejecución y no es ni siquiera necesario que todos los fragmentos del proceso estén en memoria durante la ejecución.

Los dos enfoques básicos de memoria virtual son la paginación y la segmentación. Con paginación, cada proceso se divide en páginas de tamaño fijo y relativamente pequeño. La segmentación permite el uso de fragmentos de tamaño variable. También es posible combinar segmentación y paginación en un único esquema de gestión de memoria.

Un esquema de gestión de memoria virtual exige un soporte tanto de hardware como de software. El soporte de hardware lo proporciona el procesador. Este soporte incluye la traducción dinámica de direcciones virtuales a direcciones físicas y la generación de interrupciones cuando una página o segmento referenciado no están en memoria principal. Estas interrupciones activan el software de gestión de memoria del sistema operativo.

Una serie de cuestiones de diseño relativas a los sistemas operativos dan soporte a la gestión de memoria virtual:

- *Políticas de lectura:* Las páginas de los procesos pueden cargarse por demanda o se puede usar una política de paginación previa; esta última agrupa las actividades de entrada cargando varias páginas a la vez.
- *Políticas de ubicación:* En un sistema de segmentación pura, un segmento entrante debe encajar en un espacio de memoria disponible.
- *Políticas de reemplazo:* Cuando la memoria está llena, debe tomarse la decisión de qué página o páginas serán reemplazadas.
- *Gestión del conjunto residente:* El sistema operativo debe decidir cuánta memoria principal ha de asignar a un proceso en particular cuando se carga. Puede hacerse una asignación estática en el momento de la creación del proceso o bien puede cambiar dinámicamente.
- *Políticas de vaciado:* Las páginas modificadas de un proceso pueden escribirse al disco en el momento del reemplazo o bien puede aplicarse una política de paginación previa; esta última agrupa las actividades de salida escribiendo varias páginas de una vez.
- *Control de carga:* El control de carga determina el número de procesos residentes que habrá en memoria principal en un momento dado.

7.5 *LECTURAS RECOMENDADAS*

Como se podía esperar, la memoria virtual recibe un amplio estudio en la mayoría de los libros de sistemas operativos. [MAEK87] ofrece un buen resumen de varias áreas de investigación. [CARR84] proporciona un excelente examen en profundidad del rendimiento, [KUCK78] y [BAER80] ofrecen algunos resultados analíticos y de simulación interesantes. Todavía merece la pena leer el texto clásico, [DENN70]. [DEWA90] ofrece una presentación detallada de las características de la memoria virtual del 80386.

Es una experiencia interesante leer [IBM86], que ofrece un informe detallado de las herramientas y opciones disponibles para que un administrador optimice las políticas de memoria virtual de MVS. El documento muestra la complejidad del problema.

BAER80 BAER, J. *Computer Systems Architecture.* Computer Science Press, Rockville, MD, 1980. CARR84 CARR, R. *Virtual Memory Management.* UMI Research Press, Ann Arbor, MI, 1984. DENN70 DENNING, P. "Virtual Memory." *Computing Surveys,* septiembre de 1970.

DEWA90 DEWAR, R. y SMOSNA, M. *Microprocessors: A Programmer's View.* McGraw-Hill, Nueva York, 1990.

IBM86 IBM NATIONAL TECHNICAL SUPPORT, LARGE SYSTEMS. *Múltiple Virtual Storage (MVS) Virtual Storage Tuning Cookbook.* Dallas Systems Center Technical Bulletin G320-0597, junio de 1986.

KUCK78 KUCK, D. *The Structure ofComputers and Computations.* Wiley, Nueva York, 1978.

MAEK87 MAEKAWA, M., Oidehoeft, A. y Oideohoeft, R. *Operating Systems: Advanced Con-cepts.* Benjamin Cummings, Memo Park, CA, 1987.

7.6

PROBLEMAS

7.1 Supóngase que la tabla de páginas del proceso que está ejecutándose actualmente en el procesador es como la siguiente. Todos los números son decimales, todos están numerados empezando en el cero y todas las direcciones son de bytes de memoria. El tamaño de página es de 1024 bytes.

N.º de Página Virtual	Bit de Validez	Bit de Referencia	Bit de Modificación	N.º de Marco de Página
0	1	1	0	4
1	1	1	1	7
2	0	0	0	—
3	1	0	0	2
4	0	0	0	—
5	1	0	1	0

a) Describir exactamente cómo se traduce en general una dirección virtual generada por la CPU a una dirección física de memoria principal.

b) ¿Qué dirección física, si la hay, se correspondería con cada una de las siguientes direcciones virtuales? (No debe intentarse manejar los fallos de página, si los hay).

(i) 1052
(ii) 2221
(iii) 5499

7.2 Un proceso tiene asignados 4 marcos de página. (Todos los números siguientes son decimales y todos están numerados empezando por cero). A continuación se muestra el instante de la última carga de páginas en cada marco de memoria, el instante del último acceso a la página en cada marco, el de cada marco y los bits de referencia (R) y modificación (M) (los instantes se dan en pulsos de reloj del procesador desde el instante O hasta el suceso, no el número de pulsos desde el suceso hasta el instante actual).

N.º de Página Virtual	Marco de Referencia	Instante de Carga	Instante de Referencia	Bit *R*	Bit *M*
2	0	60	161	0	1
1	1	130	160	0	0
0	2	26	162	1	0
3	3	20	163	1	1

Se ha producido un fallo en la página virtual 4. ¿Qué marco reemplazará su contenido para cada una de las siguientes políticas de gestión de memoria? Expliqúese por qué en cada caso.

a) FIFO (primera en entrar, primera en salir).

b) LRU (usada hace más tiempo).

c) NRU (no usada recientemente). (Si se necesita saber como surge esta configuración en particular, hay que emplear los bits *R* y *M* de un modo razonable: todos los bits de referencia se borraron inmediatamente después del pulso de reloj 161).

d) Reloj.

e) MIN Óptimo de Belady (Emplear la cadena de referencias situada debajo).

f) Dado el estado de memoria anterior, inmediatamente antes del fallo de página, considérese la siguiente cadena de referencias a páginas virtuales:

4,0,0,0,2,4,2,1,0,3,2

¿Cuántos fallos de página se producirán si se emplea la política de conjunto de trabajo con un tamaño de ventana de cuatro en vez de con asignación fija? Muéstrese claramente cuándo se produce cada fallo de página.

7.3 Un proceso hace referencia a cinco páginas. A, B, C, D y E, en el siguiente orden:

A; B; C; D; A; B; E; A; B; C; D; E

Supóngase que el algoritmo de reemplazo es el de primera en entrar/primera en salir y determínese el número de transferencias de páginas durante esta secuencia de referencias, comenzando con la memoria principal vacía con 3 y 4 marcos de página.

Id. de segmento	número de página	desplazamiento

7.4 El Sistema RISC/6000 de IBM emplea el siguiente formato de direcciones virtuales:

Para acceder a la tabla de páginas, se calcula un valor de dispersión a partir realizando un XOR de los campos Identificador de segmento y número de página. ¿Qué efectividad se puede suponer que ofrece el algoritmo para generar una tabla de dispersión con un encadenamiento mínimo?

7.5 Construir un diagrama similar al de la figura 7.8 que muestre el funcionamiento conjunto de segmentación, paginación, buffer de traducción adelantada y cache.

7.6 En el VAX, las tablas de páginas de usuario se encuentran en direcciones virtuales del espacio del sistema. ¿Cuál es la ventaja de tener las tablas de páginas de usuario en memoria virtual en lugar de en memoria principal? ¿Cuál es la desventaja?

7.7 Supóngase que la sentencia de programa **for i in 1... n do** A[i] := B[i] + C[i] **endfor**

se ejecuta en una memoria con un tamaño de página de 1000 palabras. Sea $n = 1000$. Por medio de una máquina que posee un rango completo de instrucciones de registro a registro y emplea registros de índice, escribir un programa en un hipotético lenguaje de máquina para implementar la sentencia anterior. A continuación, mostrar la secuencia de referencias a páginas durante la ejecución.

7.8 Para implementar los diversos algoritmos de ubicación tratados en la sección de partición dinámica, debe mantenerse una lista de bloques libres de memoria. Para cada uno de los tres métodos expuestos (mejor ajuste, primer ajuste, siguiente ajuste), ¿Cuál es la longitud media de la búsqueda.

7.9 Otro algoritmo de ubicación para particiones dinámicas se conoce como el del *peor ajuste.* En este caso, el bloque de memoria más grande que esté libre se usa para cargar el proceso. Discutir los pros y los contras de este método, comparado con el del mejor ajuste, el del primer ajuste y el del siguiente ajuste. ¿Cuál es la longitud media de la búsqueda para el peor ajuste?

7.10 El método de segmentación empleado en el Sistema/370 parece carecer de muchas de las ventajas potenciales de la segmentación. ¿De qué ventajas carece? ¿Cuál es el mayor beneficio de la segmentación del 370?

7.11 Para la gestión de memoria del Sistema/370, proponer un método que determine cuál es el marco usado hace más tiempo (LRU) haciendo uso solamente del bit de referencia asociado a cada marco de la memoria real.

7.12 Supóngase que se tiene un computador con una instrucción de 3 direcciones para sumar el contenido de dos posiciones de memoria y almacenar el resultado en una tercera posición. En ensamblador, la instrucción se asemejaría a:

ADD A, B, C /* (A) + (B) —•(C) */

Si la instrucción ocupa tres palabras y si todas las direcciones son directas (es decir, la dirección efectiva forma parte de la propia instrucción), cuál sería el número mínimo de marcos que necesita esta instrucción para garantizar su ejecución correcta. Explicar la respuesta.

7.13 Una clave del rendimiento de la política VSWS de gestión del conjunto residente es el valor de *Q*. Algunos experimentos han demostrado que, con un valor fijo de *Q* para un proceso, existen diferencias considerables en la frecuencia de fallos de página para distintas etapas de la ejecución. Es más, si se usa un único valor de *Q* para procesos diferentes, se obtienen diferencias drásticas en la frecuencia de fallos de página. Estas diferencias indican que un mecanismo que ajustara dinámicamente el valor de *Q* durante la vida de un proceso podría mejorar el comportamiento del algoritmo. Proponer un mecanismo sencillo con tal propósito.

7.14 Obtener la fórmula del encadenamiento separado de la tabla 7.7

7.15 Cuando se implemento la memoria virtual por primera vez, las memorias principales eran muy pequeñas en comparación con las actuales y las aplicaciones eran, con frecuencia, mayores que la memoria principal. La tecnología de memoria ha cambiado drásticamente desde entonces, siendo comunes los sistemas de varios megabytes en los computadores personales. A medida que el tamaño de la memoria principal crezca en un rango de gigabytes en un futuro no muy distante, sin que los tiempos de acceso al disco muestren un incremento similar, ¿seguirá siendo la memoria virtual un esquema viable de gestión de memoria? Expóngase por qué o por qué no.

7.16 Exponer los requisitos de soporte de hardware para la estructura de tabla de páginas inversa. ¿Cómo afecta este método a la compartición?

APÉNDICE 7A

TABLAS DE DISPERSIÓN

Considérese el siguiente problema. Un conjunto de N elementos debe almacenarse en una tabla. Cada elemento está formado por una etiqueta y alguna información adicional, a la que se llamará el valor del elemento. Sería conveniente poder realizar algunas operaciones comunes sobre la tabla, tales como inserción, eliminación y búsqueda de un elemento dado por la etiqueta.

Si las etiquetas de los elementos son numéricas, en el rango de 0 a *M,* una solución sencilla podría ser emplear una tabla de longitud *M.* Un elemento con etiqueta;' puede insertarse en la tabla en la posición *i.* Mientras que los elementos sean de longitud fija, las búsquedas en la tabla son triviales y requieren una indexación de la tabla en función de la etiqueta numérica del elemento. Es más, no es necesario almacenar la etiqueta de un elemento en la tabla, puesto que está implícita en la posición del elemento. Esta tabla se conoce como **tabla de acceso directo.**

Si las etiquetas no son numéricas, aún es posible utilizar un método de acceso directo. Se hará referencia a los elementos como A[l],... , *A[N].* Cada elemento *A[i]* está formado por una etiqueta o clave k_i y un valor v_i. Se define una función de correspondencia *I(k)* que toma

un valor entre 1 y *M* para todas las claves, con $I(k_i) \approx I(k_j)$ para *i y j* cualesquiera. En este caso, puede usarse también una tabla de acceso directo con longitud igual a *M.*

La primera dificultad de estos esquemas se produce si *M » N.* En tal caso, la proporción de entradas no utilizadas de la tabla es muy elevada, lo que origina un empleo ineficiente de la memoria. Una alternativa podría ser usar una tabla de longitud *N* y almacenar *N* elementos (etiquetas y valores) en las *N* entradas de la tabla. Con este esquema, la cantidad de memoria se minimiza, pero se produce una carga de procesamiento para las búsquedas en la tabla. Se presentan las siguientes posibilidades:

• *Búsqueda secuencial:* Este método de fuerza bruta consume mucho tiempo para tablas grandes.

• *Búsqueda asociativa:* Con el hardware apropiado, se puede buscar en todos los elementos de la tabla simultáneamente. Este método no es de propósito general y no puede aplicarse a todas las tablas.

• *Búsqueda binaria:* Si las etiquetas o los números correspondientes a las etiquetas están dispuestos en la tabla por orden ascendente, una búsqueda binaria es mucho más rápida que una búsqueda secuencial (tabla 7.7) y no necesita un hardware especial.

La búsqueda binaria parece prometedora para la búsqueda en tablas. El mayor inconveniente de este método es que añadir nuevos elementos no es un proceso sencillo y requiere la reordenación de las entradas. Por tanto, la búsqueda binaria se usa generalmente sólo para tablas bastante estáticas, que rara vez cambian.

Sería conveniente evitar la penalización de memoria impuesta por el método de acceso directo y la carga de procesamiento de las alternativas que se acaban de enunciar. El método usado con mayor frecuencia para alcanzar este compromiso se denomina **dispersión** *(hashing).* La dispersión, desarrollada en los años 50, es sencilla de implementar y tiene dos ventajas. En primer lugar, puede encontrar la mayor parte de los elementos en una única búsqueda, como en el acceso directo y, en segundo lugar, las inserciones y eliminaciones puede gestionarse sin ninguna complejidad añadida.

La función de dispersión puede definirse como sigue. Supóngase que se desea almacenar un máximo de *N* elementos en una **tabla de dispersión** de longitud *M,* siendo *M* & *N,* pero no mucho mayor que *N.* Para insertar un elemento en la tabla, hay que dar los siguientes pasos:

11. Convertir la etiqueta del elemento en un número *n* pseudoaleatorio entre O y *M* - 1. Por ejemplo, si la etiqueta es numérica, una función de correspondencia habitual es dividir la etiqueta por *M* y tomar el resto como valor de *n.*

12. Usar *n* como índice en la tabla de dispersión.

a) Si la entrada correspondiente de la tabla está vacía, almacenar el elemento (etiqueta y valor) en dicha entrada.

b) Si la entrada ya está ocupada, almacenar el elemento en un área de desbordamiento, como se explica más adelante.

Para llevar a cabo la búsqueda en la tabla de un elemento cuya etiqueta es conocida, puede emplearse la siguiente rutina:

Ll. Convertir la etiqueta del elemento en un número *n* pseudoaleatorio entre O y *M* - 1, usando la misma función de correspondencia que para la inserción.

L2. Usar *n* como índice en la tabla de dispersión.

a) Si la entrada correspondiente de la tabla está vacía, el elemento no se ha almacenado en la tabla con anterioridad.

b) Si la entrada ya está ocupada y las etiquetas coinciden, se puede sacar el valor.

c) Si la entrada ya está ocupada y las etiquetas no coinciden, continuar la búsqueda en el área de desbordamiento.

Los esquemas de dispersión difieren en la manera de tratar el desbordamiento. Una técnica habitual se denomina técnica de **encadenamiento abierto** y es muy usada en los compiladores. Con esta técnica, la anterior regla I2(b) pasa a ser:

• Si la entrada ya está ocupada, poner $n = n + 1$ (mod M) y volver al paso I2(a) La regla L2(c) se modifica de manera análoga.

En la figura 7.27a se muestra un ejemplo. En este caso, las etiquetas de los elementos a almacenar son numéricas y la tabla de dispersión tiene ocho posiciones (M = 8). La función de correspondencia va a tomar el resto de la división por 8. La figura supone que los elementos se insertaron en orden numérico ascendente, aunque esto no es obligatorio. Así pues, los elementos 50 y 51 se corresponden con las posiciones 2 y 3, respectivamente; como estas posiciones están vacías, se insertaron ahí mismo. El elemento 74 también se corresponde con la posición 2, pero como ésta no está vacía, se prueba con la posición 3. Esta también está ocupada, por lo que finalmente se usa la posición 4.

No es fácil determinar la longitud media de la búsqueda de un elemento en una tabla de dispersión con encadenamiento abierto, debido al efecto de agrupación. Schay y Spruth [SCHA62] obtuvieron una fórmula aproximada:

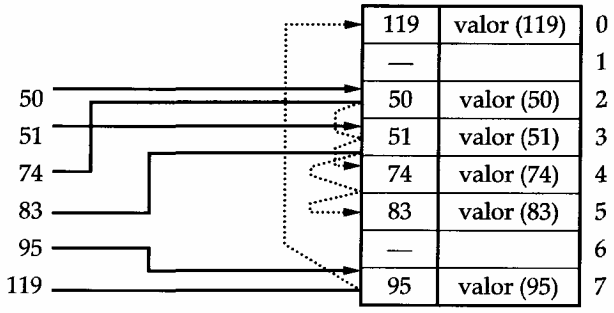
$$\text{Longitud media de la búsqueda} = \frac{2 - r}{2 - 2r}$$

donde $r = N / M$. Nótese que el resultado es independiente del tamaño de la tabla y depende sólo de lo llena que esté. Lo sorprendente del resultado es que con la tabla ocupada al 80%, la longitud media de la búsqueda todavía es cercana a 3.

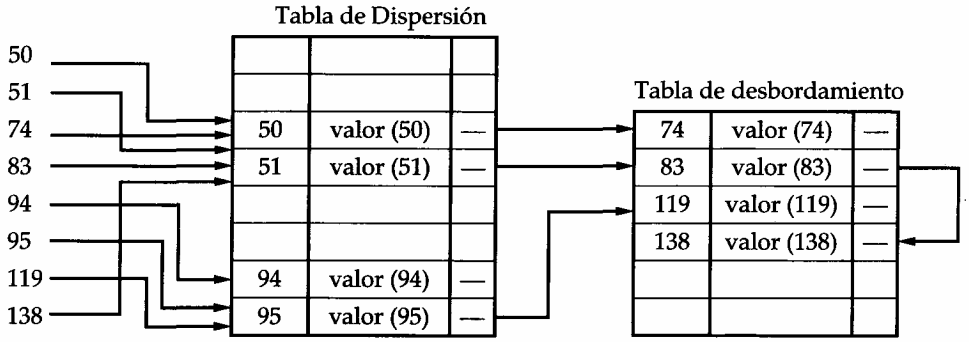
Incluso así, una longitud de búsqueda de 3 puede considerarse larga y la tabla de dispersión con encadenamiento abierto presenta el problema adicional de que no es fácil eliminar

TABLA 7.7 Longitud media de la búsqueda de un elemento de entre N elementos de una tabla de longitud M

Técnica	Longitud de la Búsqueda
Directa	1
Secuencial	$\frac{M + 1}{2}$
Binaria	$\log_2 M$
Dispersión (encadenamiento abierto)	$\frac{2 - \frac{N}{M}}{2 - \frac{2N}{M}}$
Dispersión (Encadenamiento separado)	$1 + \frac{N - 1}{2M}$



(a) Encadenamiento abierto



(b) Encadenamiento separado

FIGURA 7.27 Dispersión

elementos. Un método más atractivo, que realiza búsquedas más cortas (tabla 7.7) y permite tanto eliminaciones como inserciones, es el **encadenamiento separado.** Esta técnica está ilustrada en la figura 7.27b. En este caso, hay una tabla separada en la que se insertan las entradas de desbordamiento. Esta tabla incluye punteros que enlazan la cadena de entradas asociadas con una posición de la tabla de dispersión. En tal caso, la longitud media de una búsqueda, suponiendo una distribución de datos aleatoria, es:

$$\text{Longitud media de la búsqueda} = 1 + \frac{N-1}{2M}$$

Para valores mayores de N y M, este valor se aproxima a 1,5 para $N = M$. Así pues, esta técnica ofrece un almacenamiento compacto con rapidez en la búsqueda.

CAPÍTULO 8

Planificación de monoprocesadores

En un sistema multiprogramado, la memoria principal contiene varios procesos. Cada proceso alterna entre usar el procesador y esperar que se realice una operación de E/S o que ocurra algún otro suceso. El procesador o los procesadores se mantienen ocupados ejecutando un proceso mientras los demás esperan.

La clave de la multiprogramación está en la planificación. De hecho, son cuatro las clases de planificación que entran en juego normalmente (tabla 8.1). Una de ellas, la planificación de E/S, se aborda de una forma más adecuada en el capítulo 10, donde se habla de E/S. Las otras tres clases de planificación, que son de planificación del procesador, son objeto de estudio en este capítulo y en el siguiente.

El capítulo comienza con un examen de los tres tipos de planificación del procesador y muestra la forma en que se relacionan. Se verá que las planificaciones a largo y medio plazo están conducidas hacia el objetivo principal de los aspectos de rendimiento relativos al grado de multiprogramación. Estos puntos se tratan, hasta cierto punto, en el capítulo 3 y, con un mayor detalle, en los capítulos 6 y 7. Así pues, el resto del capítulo se centrará en la planificación a corto plazo, limitándose a considerar la planificación de sistemas de un solo procesador. Puesto que la presencia de varios procesadores añade una complejidad adicional, es preferible centrarse en el caso del monoprocesador en primer lugar, de forma que las diferencias entre los algoritmos de planificación se vean con claridad.

La sección 8.2 muestra los diversos algoritmos que pueden usarse en la toma de decisiones de planificación a corto plazo.

8.1 TIPOS DE PLANIFICACIÓN

El afán de la planificación del procesador consiste en asignar los procesos al procesador o los procesadores para que sean ejecutados en algún momento, de forma que se cumplan objetivos del sistema tales como el tiempo de respuesta, la productividad y la eficiencia del procesador. En muchos sistemas, la actividad de planificación se divide en tres funciones in-

TABLA 8.1 TIPOS DE PLANIFICACIÓN

Planificación a largo plazo	Decisión de añadir procesos a la reserva de procesos a ejecutar
Planificación a medio plazo	Decisión de añadir procesos al conjunto de procesos que se encuentran parcial o completamente en memoria
Planificación a corto plazo	Decisión sobre qué proceso disponible será ejecutado en el procesador
Planificación de E/S	Decisión sobre qué solicitud de E/S pendiente será tratada por un dispositivo de E/S disponible

dependientes: planificación a largo, medio y corto plazo. Los nombres hacen referencia a la frecuencia relativa con la que son ejecutadas estas funciones.

La figura 8.1 relaciona las funciones de planificación con el diagrama de transición de estados de un proceso. La planificación a largo plazo se lleva a cabo al crear un proceso nuevo. La creación de un nuevo proceso parte de la decisión de si añadir un proceso al conjunto de procesos activos. La planificación a medio plazo forma parte del proceso de intercambio y tiene como origen la decisión de añadir un proceso a los que se encuentran, al menos parcialmente, en memoria principal y, por tanto, disponibles para ejecutar. La planificación a corto plazo es la decisión de qué proceso en estado Listo será el que ejecute a continuación. En la figura 8.2 se reorganiza el diagrama de transición de estados para representar gráficamente el anidamiento de las funciones de planificación.

La planificación afecta al rendimiento del sistema, pues determina qué proceso esperará y qué proceso continuará. Este punto de vista es el que se presenta en la figura 8.3, que muestra las colas involucradas en las transiciones de estado de un proceso. Fundamentalmente, la planificación no es sino una gestión de dichas colas que minimice la espera y optimice el rendimiento del entorno.

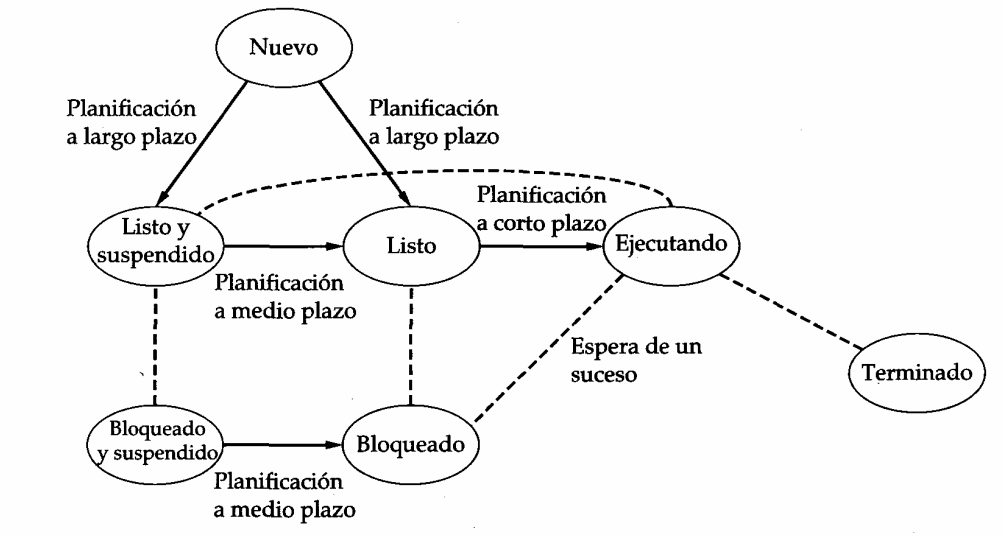


FIGURA 8.1 Planificación y transiciones de estado de los procesos

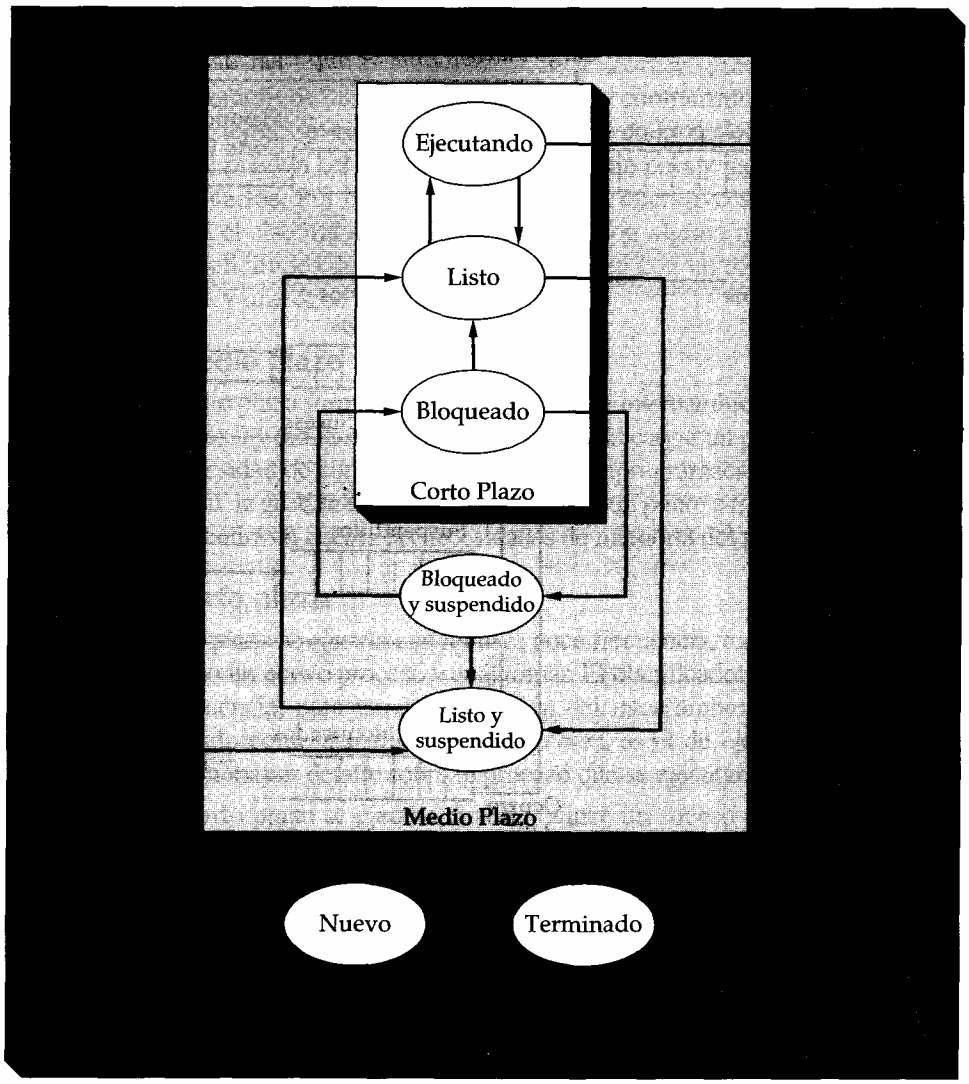


FIGURA 8.2 Niveles de planificación

Planificación a largo plazo

La planificación a largo plazo determina cuáles son los programas admitidos en el sistema. De este modo, se controla el grado de multiprogramación. Una vez admitido, un trabajo o un programa de usuario se convierte en un proceso y es añadido a la cola del planificador a corto plazo. En algunos sistemas, un proceso recién creado comienza en situación de descargado de la memoria principal, en cuyo caso se añade a la cola del planificador a medio plazo.

En un sistema de proceso por lotes o bien en la parte de proceso por lotes de un sistema operativo de propósito general, los procesos recién incorporados se encaminan hacia el disco y permanecen detenidos en una cola de procesamiento por lotes. El planificador a

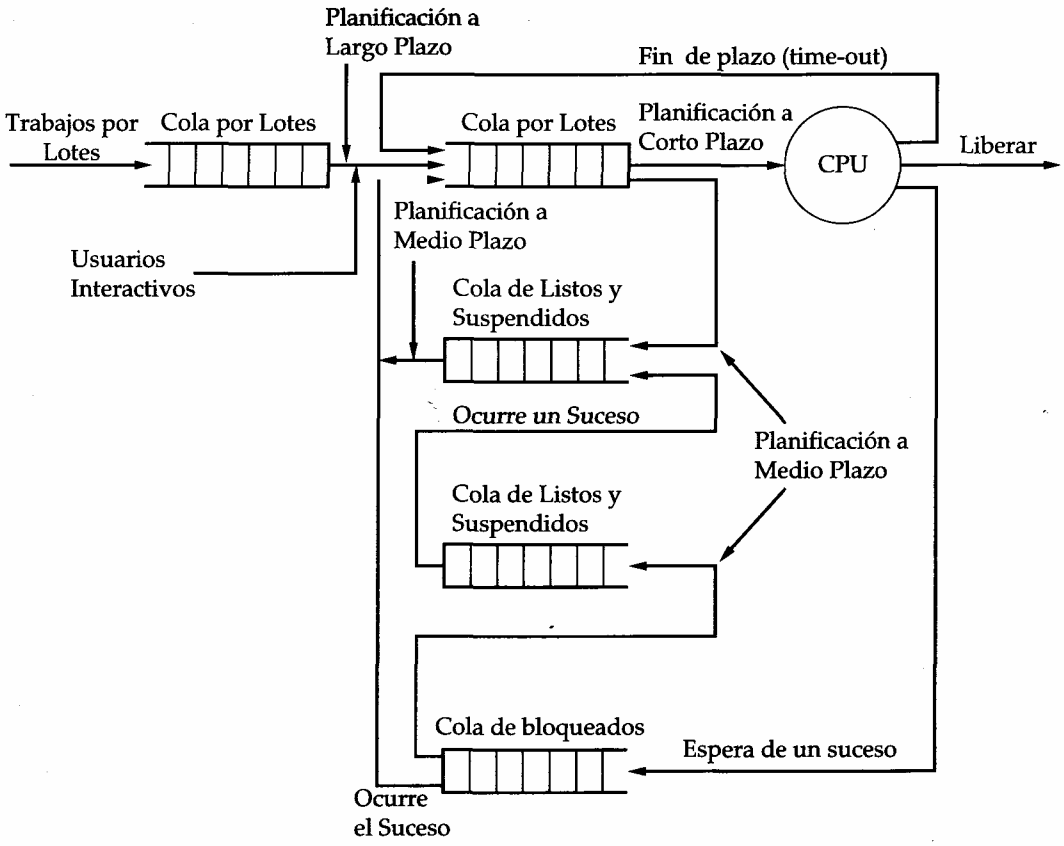


FIGURA 8.3 Diagrama de colas de planificación

largo plazo creará procesos a partir de la cola cuando sea posible. Dos decisiones entran en juego en este sistema. Primero, el planificador debe decidir si el sistema operativo puede acoger algún proceso más. Segundo, el planificador debe decidir qué trabajos son aceptados y se convierten en procesos. Considérense un momento estas dos decisiones.

La decisión de cuándo crear un nuevo proceso viene dada, en general, por el grado de multiprogramación. Cuantos más procesos se crean, menor es el porcentaje de tiempo en el que cada proceso puede ejecutar. Así pues, el planificador a largo plazo puede limitar el grado de multiprogramación para ofrecer un servicio satisfactorio al conjunto de procesos actual. Cada vez que finaliza un trabajo, el planificador puede tomar la decisión de añadir uno o más trabajos" nuevos. Además, si la fracción de tiempo que el procesador está desocupado excede un cierto umbral, se puede volver a invocar al planificador a largo plazo.

La decisión de cuál va a ser el siguiente proceso a admitir puede basarse en un simple algoritmo primero en llegar/primero en servirse (FCFS, *First-come, First-served) o* bien puede basarse en alguna herramienta de gestión de la actividad del sistema. El criterio empleado puede tener en cuenta prioridades, tiempos de ejecución esperados y exigencias de E/S. Por ejemplo, si se encuentra disponible la información, el planificador puede intentar mantener una combinación de procesos con mayor carga de procesador y con mayor carga

de E/S[1]. Además, en un intento de equilibrar el uso de la E/S, la decisión puede tomarse dependiendo del recurso de E/S que se solicite.

Para programas interactivos en un sistema de tiempo compartido, cuando un usuario intenta conectarse al sistema, se genera una solicitud de crear un proceso. Los usuarios de tiempo compartido no pueden ser puestos simplemente en cola y hacerles esperar hasta que el sistema pueda aceptarlos. Por el contrario, el sistema operativo acepta todas las llegadas autorizadas hasta que el sistema se sature de acuerdo con alguna medida predefinida. Llegado este punto, las solicitudes de conexión se responden con un mensaje que indica que el sistema está completo y que debe intentarse más tarde.

Planificación a medio plazo

La planificación a medio plazo forma parte de la función de intercambio. Los temas relacionados se tratan en los capítulos 3 y 6. Generalmente, la decisión de cargar un proceso en memoria principal se basa en la necesidad de controlar el grado de multiprogramación. En un sistema que no emplee memoria virtual, la gestión de memoria también es un punto a tratar. Así pues, la decisión de carga en memoria tendrá en cuenta las necesidades de memoria del proceso descargado.

Planificación a corto plazo

El planificador a largo plazo se ejecuta con relativa poca frecuencia, tomando una primera decisión sobre si tomar o no un nuevo proceso y cuál tomar. El planificador a medio plazo se ejecuta con algo más de frecuencia, para tomar la decisión del intercambio. El planificador a corto plazo, también conocido como *distribuidor (dispatcher), es* el de ejecución más frecuente y toma decisiones con un mayor detalle sobre el proceso que se ejecutará a continuación.

El planificador a corto plazo se ejecuta cuando ocurre un suceso que puede conducir a la interrupción del proceso actual o que ofrece la oportunidad de expulsar de la ejecución al proceso actual en favor de otro. Como ejemplos de estos sucesos se tienen:

- Interrupciones del reloj
- Interrupciones de E/S
- Llamadas al sistema operativo
- Señales

8.2 ALGORITMOS DE PLANIFICACIÓN

Criterios de la planificación a corto plazo

El principal objetivo de la planificación a corto plazo es repartir el tiempo del procesador de forma que se optimicen algunos puntos del comportamiento del sistema. Generalmente, se fija un conjunto de criterios con los que evaluar las diversas estrategias de planificación.

[1] Un proceso se considera con mayor carga de procesador si, más que nada, desempeña un trabajo de computación y utiliza la E/S sólo ocasionalmente. Un proceso se considera con mayor carga de E/S si consume más tiempo utilizando dispositivos de E/S que usando el procesador.

El criterio más empleado establece dos clasificaciones. En primer lugar, se puede hacer una distinción entre los criterios orientados al usuario y los orientados al sistema. Los criterios **orientados al usuario** se refieren al comportamiento del sistema tal y como lo perciben los usuarios o los procesos individuales. Un ejemplo posible es el tiempo de respuesta de un sistema interactivo. El *tiempo de respuesta* es el periodo de tiempo transcurrido desde que se emite una solicitud hasta que la respuesta aparece en la salida. Este valor es perceptible para el usuario, para quien tiene un interés evidente. Sería conveniente disponer de una política de planificación que ofrezca un "buen" servicio a diversos usuarios. En el caso del tiempo de respuesta, un umbral puede estar en, por ejemplo, 2 segundos. Entonces, el objetivo del mecanismo de planificación debería ser maximizar el número de usuarios que reciben un tiempo de respuesta medio de 2 segundos o menos.

Otros criterios están **orientados al sistema,** esto es, se centran en el uso efectivo y eficiente del procesador. Un ejemplo puede ser la *productividad,* es decir, el ritmo con el que los procesos terminan. La productividad es una medida muy válida del rendimiento del sistema y que sería deseable maximizar. Sin embargo, está enfocado hacia el rendimiento del sistema, en lugar de hacia el servicio ofrecido al usuario. Así pues, es interesante para el administrador del sistema, pero no para el conjunto de usuarios normales.

Mientras que los criterios orientados a usuario son importantes prácticamente en todos los sistemas, los criterios orientados al sistema tienen, en general, menor importancia en los sistemas monousuario. En los sistemas monousuario, probablemente no es importante obtener un gran aprovechamiento del procesador o una gran productividad siempre que la capacidad de respuesta del sistema a una aplicación sea aceptable.

Otra forma de clasificación es considerar los criterios relativos al rendimiento del sistema y los que no lo son. Los criterios **relativos al rendimiento** son cuantitativos y, en general, pueden evaluarse fácilmente. Algunos ejemplos son el tiempo de respuesta y la productividad. Los criterios **no relativos al rendimiento** son, en cambio, cualitativos y no pueden ser evaluados o analizados fácilmente. Un ejemplo de estos criterios es la previsibilidad. Sería conveniente que el servicio ofrecido a los usuarios tenga las mismas características en todo momento, independientemente de la existencia de otros trabajos ejecutados por el sistema. Hasta cierto punto, este criterio puede ser evaluado con el cálculo de las varianzas en función de la carga de trabajo. Sin embargo, este método no es tan sencillo de evaluar como la productividad o el tiempo de respuesta en función de la carga de trabajo.

La tabla 8.2 resume los criterios clave de planificación. Estos criterios son dependientes entre sí y es imposible optimizar todos de forma simultánea. Por ejemplo, obtener un buen tiempo de respuesta puede exigir un algoritmo de planificación que alterne entre los procesos con frecuencia, lo que incrementa la sobrecarga del sistema y reduce la productividad. Por tanto, en el diseño de una política de planificación entran en juego compromisos entre requisitos opuestos; el peso relativo que reciben los distintos requisitos dependerá de la naturaleza y empleo del sistema.

Una observación final es que, en la mayoría de los sistemas operativos interactivos, bien sean de un solo usuario o de tiempo compartido, el requisito fundamental es un tiempo de respuesta adecuado. Por la importancia de este requisito y, debido a que la definición de "adecuado" cambia de una aplicación a otra, este tema será explorado con mayor profundidad en un apéndice de este capítulo (páginas 375-377).

TABLA 8.2 CRITERIOS DE PLANIFICACIÓN

	Criterios orientados al usuario, Criterios de rendimiento
Tiempo de respuesta	Para un proceso interactivo, es el intervalo de tiempo transcurrido desde que se emite una solicitud hasta que se comienza a recibir la respuesta. A menudo, un proceso empieza a generar alguna salida para el usuario mientras que continúa procesando la solicitud. Así pues,ésta es una medida mejor que el tiempo de retorno desde el punto de vista del usuario. La disciplina de planificación debe intentar alcanzar un tiempo de respuesta bajo y maximizar el número de usuarios interactivos que reciben un tiempo de respuesta aceptable.
Tiempo de retorno	Es el intervalo de tiempo transcurrido entre el lanzamiento de un proceso y su finalización. Es la suma del tiempo de ejecución real y el tiempo consumido en la espera por los recursos, incluido el procesador. Ésta es una medida apropiada para trabajos por lotes.
Plazos	Cuando se pueden especificar plazos de terminación de un proceso, la disciplina de planificación debe subordinar otras metas a la maximización del porcentaje de plazos cumplidos.
	Criterios orientados al usuario, Otros criterios
Previsibilidad	Un determinado trabajo debe ejecutar aproximadamente en el mismo tiempo y con el mismo coste sin importar la carga del sistema. Las variaciones elevadas del tiempo de respuesta o del tiempo de retorno resultan molestas para el usuario. Puede indicar una descompensación importante en la carga del sistema o la necesidad de ajustar el sistema para remediar la inestabilidad.
	Criterios orientados al sistema, Criterios relativos al rendimiento
Productividad	La política de planificación debe intentar maximizar el número de procesos terminados por unidad de tiempo. Esta medida indica la cantidad de trabajo que se está realizando. Depende de la longitud media de cada proceso, pero también está influida por la política de planificación, que puede influir en el uso del procesador.
Utilización del procesador	Es el porcentaje de tiempo en el que el procesador está ocupado. En un sistema compartido caro, éste es un criterio importante. En sistemas monousuario y en otros sistemas, como los de tiempo real, este criterio tiene menor importancia que otros.
	Criterios orientados al sistema, Otros criterios
Equidad	En ausencia de directrices de usuario o de otras directrices ofrecidas por el sistema, los procesos deben ser tratados de igual forma y ningún proceso debe sufrir inanición.
Prioridades	Cuando se asignan prioridades a los procesos, la política de planificación debe favorecer a los de mayor prioridad.
Equilibrio de recursos	La política de planificación debe mantener ocupados los recursos del sistema. Se debe favorecer a los procesos que no utilicen recursos sobrecargados. Este criterio también afecta a la planificación a medio y largo plazo.

Se van a examinar seguidamente algunas políticas de planificación. En el resto de la sección, se supondrá que sólo existe un procesador.

Uso de prioridades

Una cuestión importante en la planificación es el uso de prioridades. En muchos sistemas, cada proceso tiene una prioridad asignada y el planificador seleccionará siempre a un proceso de mayor prioridad antes que a los de menor prioridad.

La figura 8.4 muestra el uso de prioridades. Para mayor claridad, el diagrama de colas está simplificado, ignorando la existencia de varias colas de bloqueo y del estado suspendido (compárese con la figura 3.6a). En vez de una simple cola de Listos, se ofrece un conjunto de colas en orden de prioridad descendente: RQ0, RQ1,..., RQn, donde prioridad [RQ*i*] > prioridad[RQ*j*] para i < *j*. Cuando se vaya a realizar una selección de planificación, el planificador comenzará por la cola de Listos de mayor prioridad (RQ0). Si hay uno o más procesos en esta cola, se selecciona uno mediante alguna política de planificación. Si RQ0 esta vacía, se examina RQ1, y así sucesivamente.

Un problema de los esquemas puros de planificación por prioridades es que los procesos de prioridad más baja pueden sufrir inanición. Este problema ocurre si siempre hay un flujo continuo de procesos listos de alta prioridad. Para superar este problema, la prioridad de un proceso puede cambiar en función de su edad o su historial de ejecución. Más tarde se verá un ejemplo.

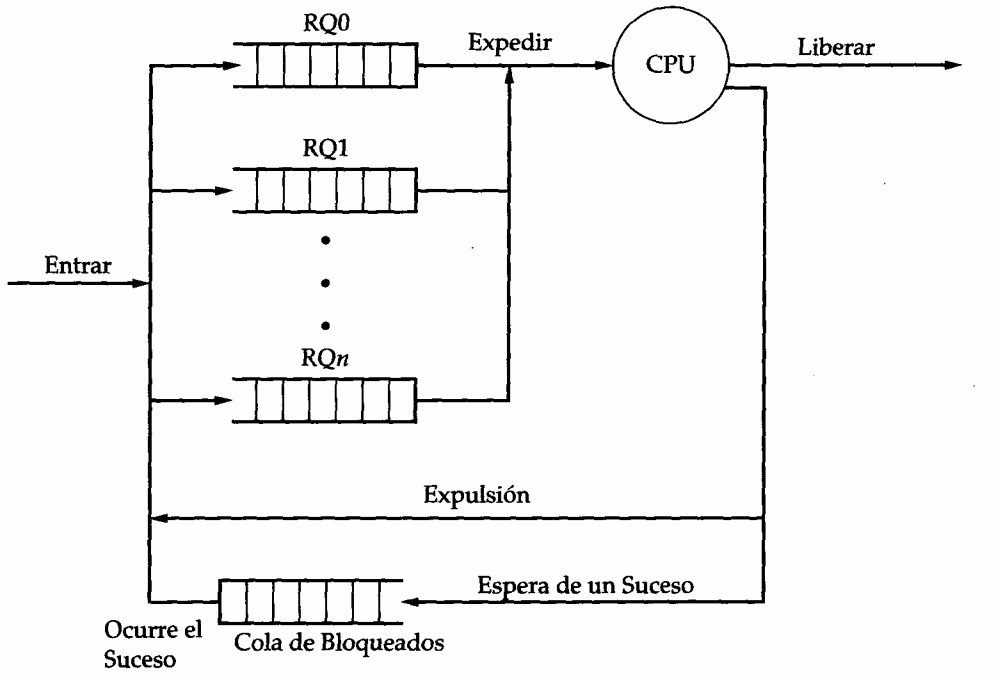


FIGURA 8.4 Planificación por prioridades

Otras políticas de planificación

La tabla 8.3 presenta información resumida sobre las distintas políticas de planificación que se examinan en este apartado. Las dos primeras características se basan en la idea propuesta en [RUSC77], que clasifica las políticas de planificación según su función de selección y su modo de decisión. La **función de selección** determina qué proceso, de entre los listos, se elige para ejecutar a continuación. La función puede estar basada en prioridades, necesidades de recursos o en las características de ejecución de los procesos. En el último caso, los valores significativos son tres:

w = tiempo consumido hasta el momento en el sistema, esperando y ejecutando
e = tiempo consumido hasta el momento en ejecución
s = tiempo total de servicio exigido por el proceso, incluido e

Por ejemplo, una función de selección $f(w) = w$ indica una disciplina de primero en entrar/primero en salir (FIFO).

El **modo de decisión** especifica los instantes de tiempo en que se aplica la función de selección. Hay dos categorías generales:

No apropiativo: En este caso, una vez que el proceso pasa a estado de ejecución, continúa ejecutando hasta que termina o hasta que se bloquea en espera de una E/S o al solicitar algún servicio del sistema.

Apropiativo: El proceso que se está ejecutando actualmente puede ser interrumpido y pasado al estado de Listos por parte del sistema operativo. La decisión de apropiarse de la CPU puede llevarse a cabo cuando llega un nuevo proceso, cuando se produce una interrupción que lleva a un proceso Bloqueado al estado Listo o periódicamente, en función de una interrupción del reloj.

Las políticas apropiativas suponen un mayor coste que las no apropiativas pero dan un servicio mejor al conjunto de todos los procesos, puesto que evitan que un proceso pueda monopolizar el procesador durante mucho tiempo. Además, el coste de la apropiación puede mantenerse relativamente bajo por medio de mecanismos eficientes de cambio de contexto (con tanta ayuda del hardware como sea posible) y usando mucha memoria principal para que el porcentaje de programas en memoria sea grande.

Para describir las diversas políticas de planificación, se utilizará como ejemplo el siguiente conjunto de procesos en ejecución:

Proceso	Instante de Llegada	Tiempo de Servicio
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Se puede considerar a estos procesos como trabajos por lotes, en los que el tiempo de servicio es el tiempo total de ejecución. En otras ocasiones, se pueden considerar como procesos en curso que se alternan en el uso del procesador y de la E/S de modo repetitivo. En este caso, los tiempos de servicio representan el tiempo de procesador requerido en un ciclo. En ambos casos, en términos del modelo de colas (ver Apéndice 8A), esta cantidad corresponde al tiempo de servicio.

TABLA 8.3 CARACTERÍSTICAS DE VARIAS POLÍTICAS DE PLANIFICACIÓN

	FCFS	Turno rotatorio	SPN	SRT	HRRN	Realimentación
Función de selección	max [w]	constante	min [s]	min [$s - e$]	max $\left(\frac{w+s}{s}\right)$	(ver texto)
Modo de decisión	No apropiativo	Apropiativo (en los cuantos de tiempo)	No apropiativo	Apropiativo (en la llegada)	Apropiativo	Apropiativo (en los cuantos de tiempo)
Productividad	No relevante	Puede ser baja si el cuanto es muy pequeño	Alta	Alta	Alta	No relevante
Tiempo de respuesta	Puede ser alto, especialmente si varían mucho los tiempos de ejecución	Ofrece un buen tiempo de respuesta para procesos cortos	Ofrece un buen tiempo de respuesta para procesos cortos	Ofrece un buen tiempo de respuesta	Ofrece un buen tiempo de respuesta	No relevante
Sobrecarga	Mínima	Baja	Puede ser alta	Puede ser alta	Puede ser alta	Puede ser alta
Efecto sobre los procesos	Penaliza los procesos cortos; penaliza los procesos con carga de E/S	Trato equitativo	Penaliza los procesos largos	Penaliza los procesos largos	Buen equilibrio	Puede favorecer a los procesos con carga de E/S
Inanición	No	No	Posible	Posible	No	Posible

w = tiempo consumido hasta el momento en el sistema, esperando y ejecutando

e = tiempo consumido hasta el momento en ejecución

s = tiempo total de servicio exigido por el proceso, incluido e

Primero en Llegar, Primero en ser Servido

La política más simple de planificación es la de primero en llegar/primero en servirse (FCFS, *First-come, First-served),* también llamada primero en entrar/primero en salir (FIFO, *First-in, First-out).* Cada vez que un proceso esté listo para ejecutar, se incorpora a la cola de Listos. Cuando el proceso actual cesa su ejecución, se selecciona el proceso más antiguo de la cola.

La figura 8.5 muestra las pautas de ejecución del ejemplo propuesto para un ciclo y la tabla 8.4 indica algunos resultados importantes. Primero, se determina el tiempo de finalización de cada proceso. A partir de él, es posible determinar el tiempo de retorno. En términos del modelo de colas, el tiempo de retorno es el tiempo en cola o tiempo total que el elemento consume en el sistema (tiempo de espera más tiempo de servicio). Un valor más útil es el tiempo de retorno normalizado, que es la razón entre el tiempo de retorno y el tiempo de servicio. Este valor indica el retardo relativo experimentado por un proceso. Normalmente, cuanto mayor es el tiempo de ejecución, mayor es el retardo absoluto que puede tolerarse. El valor mínimo para esta proporción es de 1.0; los valores mayores corresponden a niveles decrecientes del servicio.

FCFS rinde mucho mejor con procesos largos que con procesos cortos. Considérese el siguiente ejemplo, basado en uno propuesto en [FINK88]:

Proceso	Instante de Llegada	Tiempo de Servicio (T_s)	Instante de Comienzo	Instante de Finalización	Tiempo de Retorno (T_q)	T_q/T_s
A	0	1	0	1	1	1
B	1	100	1	101	100	1
C	2	1	101	102	100	100
D	3	100	102	202	199	1,99
Media					100	26

El tiempo de espera normalizado para el proceso C no es tolerable: El tiempo total que pasa en el sistema es 100 veces el tiempo necesario de procesamiento. Tan larga espera tiene lugar cada vez que un proceso corto llega justo después de uno largo. Por otro lado, incluso en este ejemplo extremo, a los procesos largos no les va del todo mal. El proceso D obtiene un tiempo de retorno que es aproximadamente el doble que el de C, pero su tiempo de espera normalizado está por debajo de 2,0.

Otro problema del FCFS es que tiende a favorecer a los procesos con carga de CPU frente a los que tienen carga de E/S. Supóngase un conjunto de procesos, uno de los cuales usa principalmente la CPU y los otros se dedican a hacer E/S. Cuando un proceso con carga de CPU está ejecutando, todos los que tienen carga de E/S deben esperar. Algunos de ellos pueden estar en colas de E/S (estado bloqueado) pero puede ser que regresen a la cola de Listos mientras el de la carga de CPU todavía está ejecutando. Llegado este momento, todos o la mayoría de los dispositivos de E/S estarán ociosos, a pesar de que, posiblemente, haya trabajo para ellos. Cuando el proceso que está actualmente en ejecución abandone el estado Ejecutando, los procesos Listos con carga de E/S pasaran rápidamente por el estado de Ejecución y volverán a bloquearse por sucesos de E/S. Si el proceso con carga de CPU también está bloqueado, el procesador pasa a estar desocupado. Así pues, FCFS puede dar como resultado un uso ineficiente tanto del procesador como de los dispositivos de E/S.

FCFS no es una alternativa atractiva por sí mismo para un sistema monoprocesador. Sin embargo, se combina a menudo con un esquema de prioridades para obtener un planificador

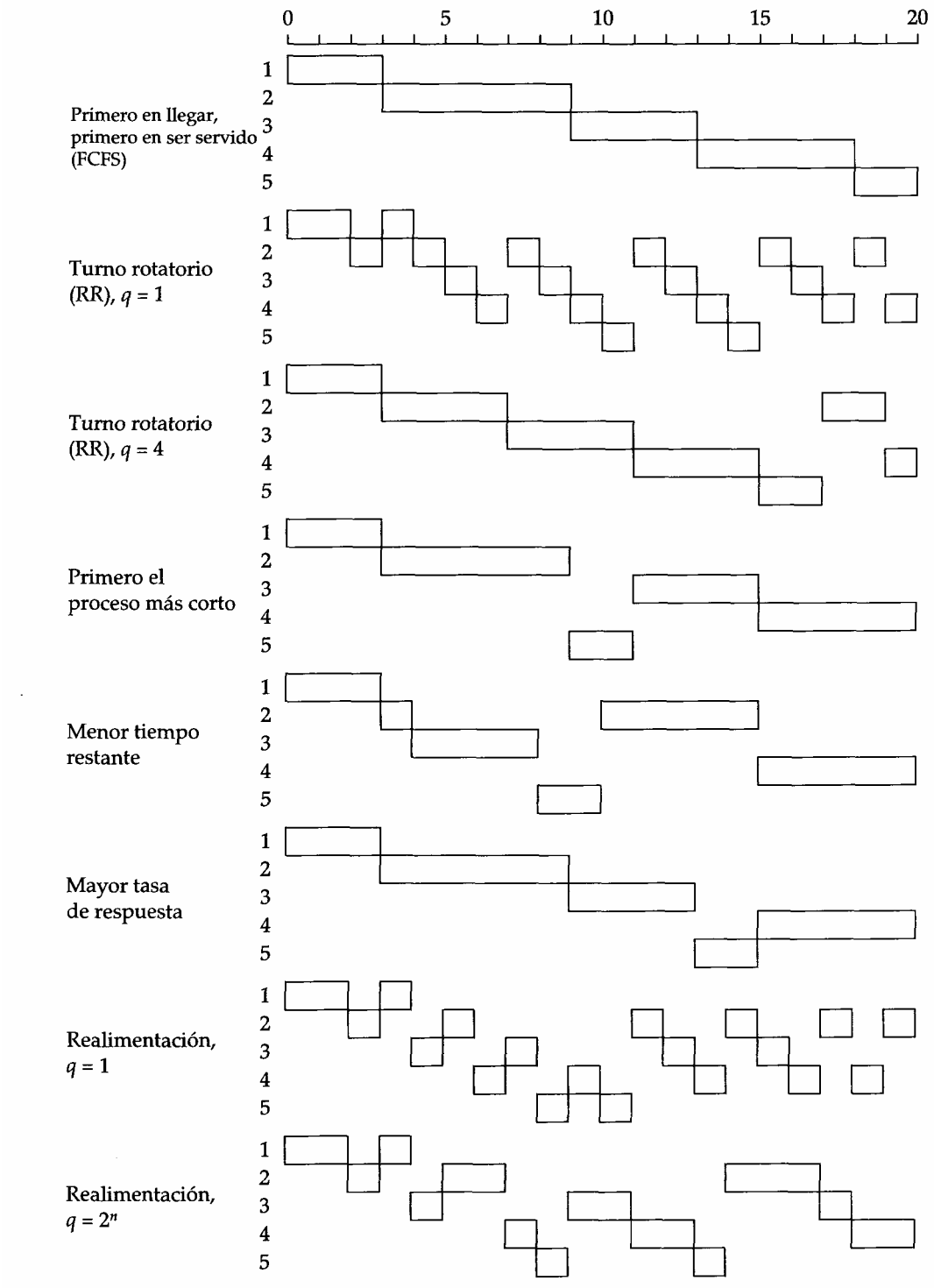


FIGURA 8.5 Una comparativa de las políticas de planificación

TABLA 8.4 UNA COMPARATIVA DE LAS POLÍTICAS DE PLANIFICACIÓN

							Media
	Proceso	1	2	3	4	5	
	Hora de Llegada	0	2	4	6	8	
	Tiempo de Servicio	3	6	4	5	2	
FCFS	Tiempo Finalización	3	9	13	18	20	
	Tiempo Retorno (T_q)	3	7	9	12	12	8,60
	T_q/T_s	1,00	1,17	2,25	2,40	6,00	2,56
RR q=1	Tiempo Finalización	4	19	17	20	15	
	Tiempo Retorno (T_q)	4	17	13	14	7	11,00
	T_q/T_s	1,33	2,83	3,25	2,80	3,50	2,74
RR q=4	Tiempo Finalización	3	19	11	20	17	
	Tiempo Retorno (T_q)	3	17	7	14	9	10,00
	T_q/T_s	1,00	2,83	1,75	2,80	4,50	2,58
SPN	Tiempo Finalización	3	9	15	20	11	
	Tiempo Retorno (T_q)	3	7	11	14	3	7,60
	T_q/T_s	1,00	1,17	2,75	2,80	1,50	1,84
SRT	Tiempo Finalización	3	15	8	20	10	
	Tiempo Retorno (T_q)	3	13	4	14	2	7,20
	T_q/T_s	1,00	2,17	1,00	2,80	1,00	1,59
HRRN	Tiempo Finalización	3	9	13	15	20	
	Tiempo Retorno (T_q)	3	7	9	9	12	8,00
	T_q/T_s	1,00	1,17	2,25	1,80	6,00	2,44
RETRO $q = 1$	Tiempo Finalización	4	20	16	19	11	
	Tiempo Retorno (T_q)	4	18	12	13	3	10,00
	T_q/T_s	1,33	3,00	3,00	2,60	1,50	2,29
RETRO $q = 2^n$	Tiempo Finalización	4	17	18	20	14	
	Tiempo Retorno (T_q)	4	15	14	14	6	10,60
	T_q/T_s	1,33	2,50	3,50	2,80	3,00	2,63

efectivo. Así pues, el planificador puede mantener un conjunto de colas, una para cada nivel de prioridad y expedir cada cola con un algoritmo primero en llegar/primero en servirse. Más tarde se planteará un ejemplo de este sistema, en el apartado "Planificación realimentada".

Turno rotatorio

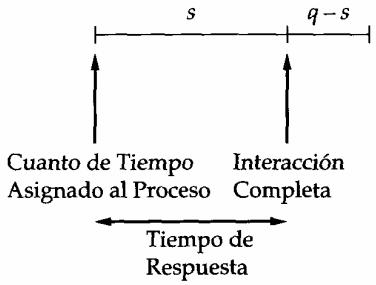
Un modo sencillo de reducir la penalización que sufren los trabajos cortos con FCFS es considerar apropiación dependiente de un reloj. La más simple de estas políticas se denomina planificación por tumo rotatorio (RR, *Round-robin).* Periódicamente, se genera una interrupción de reloj. Cuando se genera la interrupción, el proceso que está en ejecución se sitúa en la cola de Listos y se selecciona el siguiente trabajo, según un FCFS. Esta técnica se conoce también como **fracciones de tiempo,** puesto que cada proceso recibe una fracción de tiempo antes de ser expulsado.

Con la política del turno rotatorio, la cuestión principal de diseño es la longitud del cuanto de tiempo o fracción que se va a usar. Si el cuanto es muy pequeño, los procesos cortos pa-

san por el sistema rápidamente. Por otro lado, se produce una sobrecarga en la gestión de las interrupciones del reloj y en la ejecución de las funciones de planificación y expedición. Por tanto, se deben evitar los cuantos pequeños. Una referencia útil consiste en que el cuanto debe ser ligeramente mayor que el tiempo necesario para una interacción. Si es menor, la mayoría de los procesos necesitarán al menos dos cuantos. La figura 8.6 ilustra el efecto que esta referencia tiene en el tiempo de respuesta. En el caso límite de un cuanto mayor que el mayor tiempo de ejecución, el turno rotatorio degenera en FCFS.

La figura 8.5 y la tabla 8.4 muestran los resultados de aplicar al ejemplo unos cuantos de 1 y 4 unidades de tiempo, respectivamente. Nótese que el proceso 5, el trabajo más corto, experimenta una mejora significativa en ambos casos. La figura 8.7 muestra los resultados en función del tamaño del cuanto. En este caso particular, el tamaño del cuanto de tiempo tiene poca incidencia. Sin embargo, en general, se debe tener cuidado al seleccionar este valor.

El turno rotatorio es particularmente efectivo en sistemas de propósito general y de tiempo compartido o de proceso de transacciones. Una desventaja del turno rotatorio es el tratamiento que hace de los procesos con carga de procesador y con carga de E/S. Generalmente, un proceso con carga de E/S tiene ráfagas de procesador (cantidad de tiempo consumido ejecutando entre dos operaciones de E/S) más cortas que un proceso con carga de procesador. Si hay una mezcla de procesos con carga de procesador y con carga de E/S, ocurrirá lo siguiente: Un proceso con carga de E/S utiliza el procesador durante un periodo corto y después se bloquea en la E/S; espera a que se complete la operación de E/S y entonces vuelve a la cola de Listos. Por otro lado, un proceso con carga de procesador generalmente hace uso de un cuanto de tiempo completo cuando ejecuta e, inmediatamente, retorna a la cola de Listos. Así pues, los procesos con carga de procesador tienden a recibir una porción desigual de tiempo de procesador, lo que



(a) Cuanto de tiempo mayor que la interacción normal.

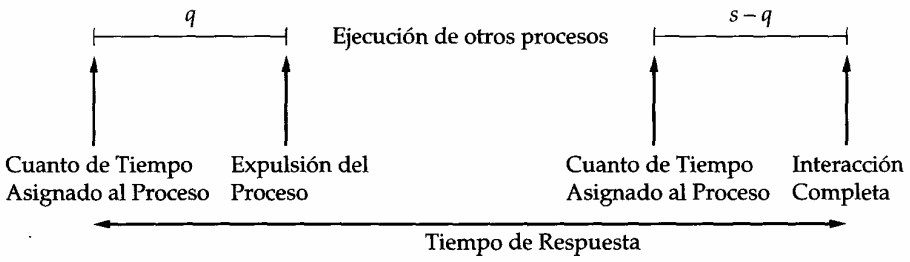


FIGURA 8.6 Efecto del tamaño del cuanto de tiempo de apropiación

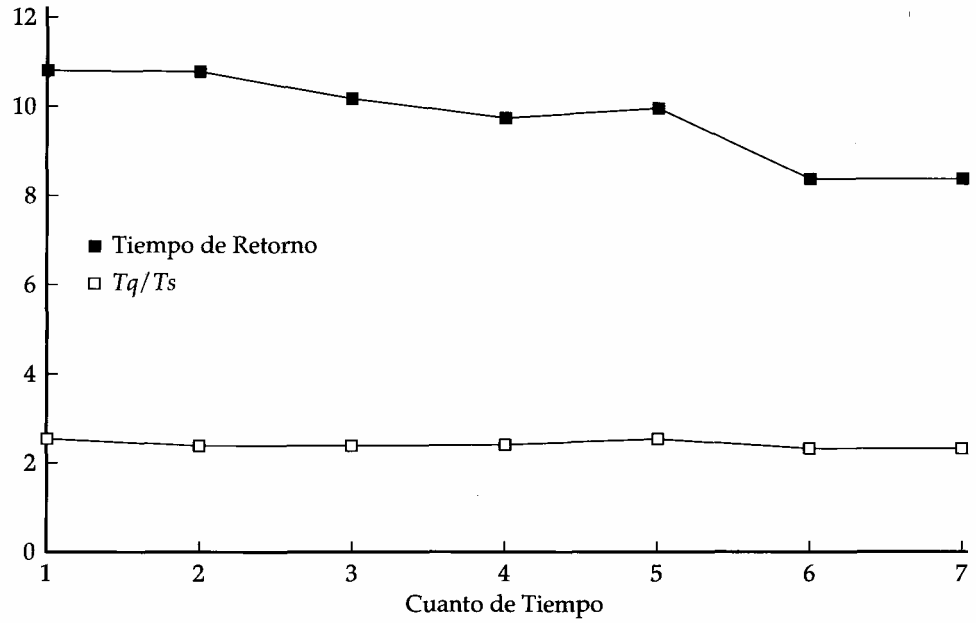


FIGURA 8.7 Rendimiento de turno rotatorio (*round-robin*)

origina un rendimiento pobre de los procesos con carga de E/S, un mal aprovechamiento de los dispositivos de E/S y un incremento de la variabilidad del tiempo de respuesta.

En [HALD91] se propone una modificación del tumo rotatorio que se denomina tumo rotatorio virtual (VRR, *VirtualRound-robin)* y que evita esta desigualdad. El esquema se muestra en la figura 8.8. Los nuevos procesos llegados se unen a la cola de Listos, que se gestiona según un FCFS. Cuando un proceso termina su cuanto de ejecución, vuelve a la cola de Listos. Cuando un proceso se bloquea por una E/S, se añade a la cola de E/S. Hasta ahora, el tratamiento es el normal. La nueva característica consiste en una cola FCFS auxiliar a la que se desplazan los procesos una vez que son liberados de la espera por E/S. Al tomar una decisión sobre el siguiente proceso a expedir, los procesos de la cola auxiliar tienen preferencia sobre los de la cola principal de Listos. Cuando se expide un proceso desde la cola auxiliar, no puede ejecutar más que un tiempo igual al cuanto básico menos el tiempo total de ejecución consumido desde que fue seleccionado por última vez en la cola de Listos. Algunos estudios de rendimientos realizados por los autores indican que este método es superior al tumo rotatorio en lo que se refiere a equidad.

Primero el proceso más corto

Otra forma de reducir el sesgo favorable al proceso más largo inherente al FCFS es la política de primero el proceso más corto (SPN, *Shortest Process Next),* una política no apropiativa en la que se selecciona el proceso con menor tiempo esperado de ejecución. Así pues, un proceso corto saltará a la cabeza de la cola, sobrepasando a trabajos largos.

La figura 8.5 y la tabla 8.4 muestran los resultados para el ejemplo. Nótese que el proceso 5 recibe servicio mucho antes que con FCFS. La mejora del rendimiento global es significativa en términos de tiempo de respuesta. Sin embargo, se incrementa la variabilidad de los tiempos de respuesta, especialmente para procesos largos, reduciendo así la previsibilidad.

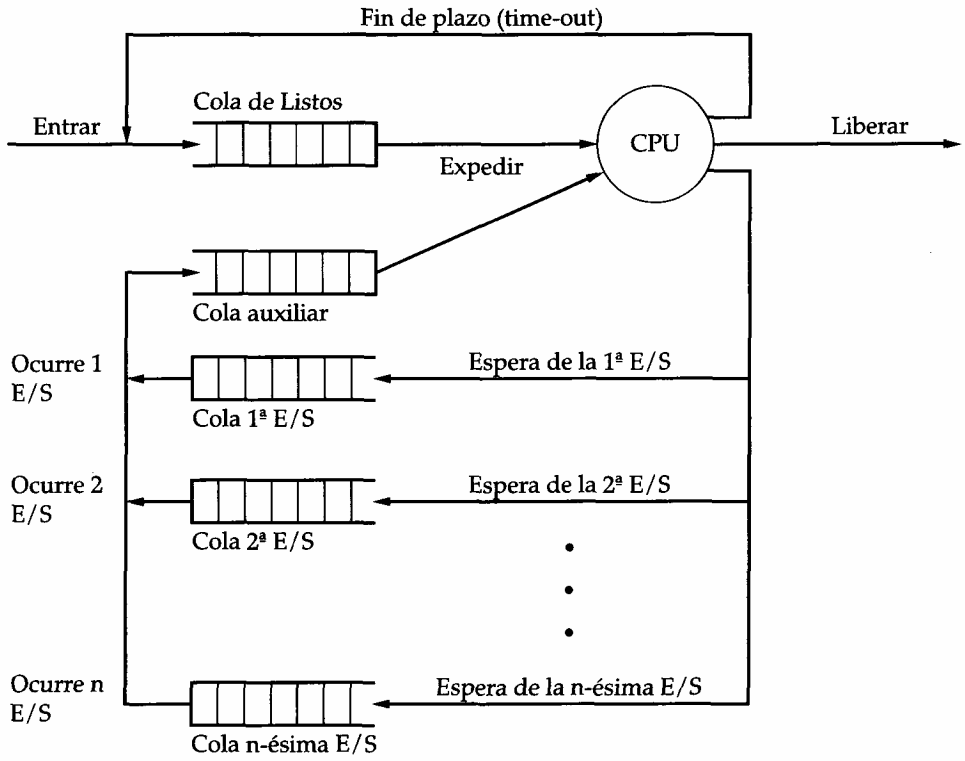


FIGURA 8.8 Diagrama de colas del planificador por turno rotatorio virtual (round-robin)

Una dificultad que plantea la política SPN es la necesidad de conocer o, por lo menos, estimar, el tiempo exigido por cada proceso. Para trabajos por lotes, el sistema puede solicitar al programador que estime el valor y se lo proporcione al sistema operativo. Si la estimación del programador está considerablemente por debajo del tiempo de ejecución real, el sistema puede abandonar el trabajo. En un entorno real de producción, se ejecutan frecuentemente los mismos trabajos y se pueden calcular estadísticas. Para los procesos interactivos, el sistema operativo puede mantener calculada una media para las ráfagas de cada proceso. El cálculo más sencillo podría ser el siguiente:

$$S_{n+1} = \frac{1}{n} \sum_{i=0}^{n} T_i \qquad (1)$$

donde

T_i = tiempo de ejecución en el procesador para el i-ésimo caso del proceso (tiempo total de ejecución para un trabajo por lotes; tiempo de ráfaga de procesador para trabajos interactivos)

S_i = valor pronosticado para el caso i-ésimo

S_0 = valor pronosticado para el primer caso; no calculado

$$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n \quad (2)$$

Nótese que esta fórmula da el mismo peso a todos los casos. Normalmente, resulta más conveniente dar un peso mayor a los casos más recientes, ya que es más probable que reflejen el comportamiento futuro. Por ello, una técnica habitual de predicción de los valores futuros a partir de una serie de valores pasados es usar un *promediado exponencial.*

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n \quad (3)$$

Compárese esta ecuación con la (2). Empleando un valor constante de a ($0 < \alpha < 1$), independiente del número de observaciones pasadas, se llega a una situación en la que se tienen en cuenta todos los valores pasados, pero los más distantes reciben un peso menor. Para verlo con más claridad, considérese el siguiente desarrollo de la ecuación (3):

$$S_{n+1} = \alpha T_n + (1 - \alpha)\alpha T_{n-1} + \cdots + (1 - \alpha)^i \alpha T_{n-i} + \cdots + (1 - \alpha)^n S_0 \quad (4)$$

Como α *y* $(1 - \alpha)$ son menores que uno, cada uno de los sucesivos términos de la ecuación es más pequeño. Por ejemplo, para $\alpha = 0{,}8$, la ecuación (4) se convierte en:

$$S_{n+1} = 0{,}8\, T_n + 0{,}16\, T_{n-1} + 0{,}032\, T_{n-2} + 0{,}0064\, T_{n-3} + \cdots$$

Cuanto más antigua es la observación, su peso cuenta menos en la media.

El tamaño del coeficiente, en función de su posición en el desarrollo, se muestra en la figura 8.9. Cuanto mayor es el valor de a, mayor es el peso dado a las observaciones más recientes. Para $\alpha = 0{,}8$, prácticamente todo el peso lo reciben las cuatro observaciones más recientes, mientras que para $\alpha = 0{,}5$, la media se distribuye sobre los valores de las ocho observaciones más recientes, más o menos. La ventaja de emplear un valor a cercano a 1 es que la media reflejará rápidamente los cambios repentinos en la cantidad observada. La desventaja es que si se produce un breve aumento en los valores observados y después se vuelve a estabilizar en algún valor medio, el empleo de un valor grande de a generará cambios bruscos en la media.

En la figura 8.10 se comparan el promediado simple con el exponencial (para dos valores diferentes de a). En la parte (a) de la figura, el valor observado comienza siendo 1, crece gradualmente hasta 10 y se estabiliza. En la parte (b), el valor observado parte de 20, desciende gradualmente hasta 10 y se estabiliza. En ambos casos, se parte de un valor estimado de $So = 0$. Esto otorga mayor prioridad a los procesos nuevos. Nótese que el promediado exponencial marca los cambios en el comportamiento de los procesos más rápidamente que el promediado simple y que el valor mayor de a genera una reacción rápida a los cambios del valor observado.

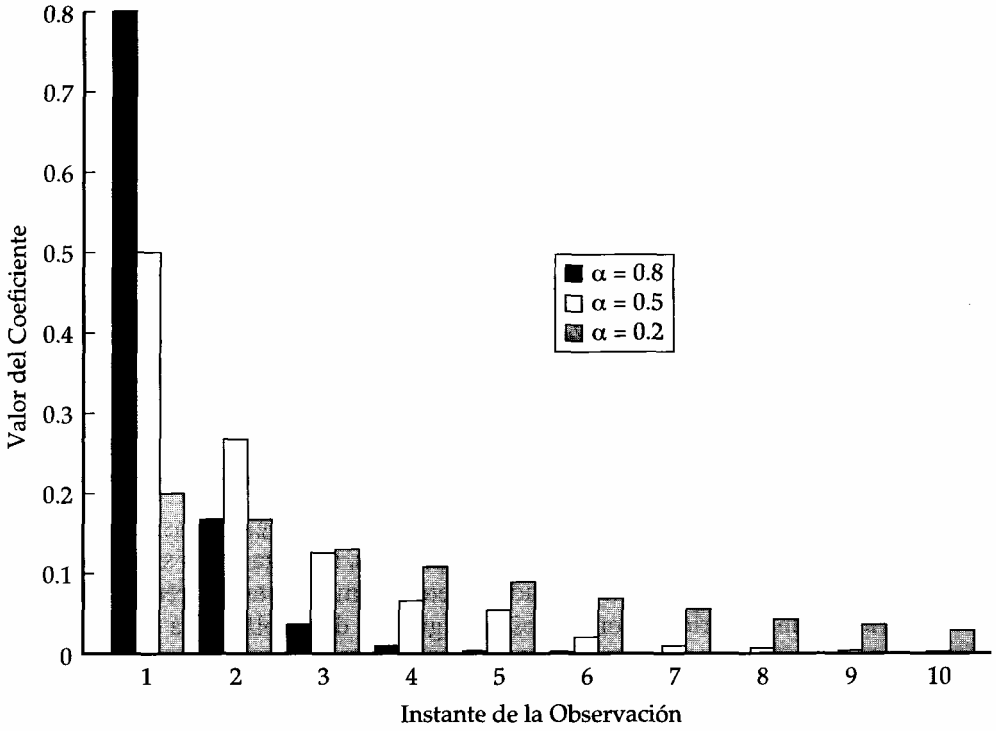


FIGURA 8.9 Coeficientes de suavizado exponencial

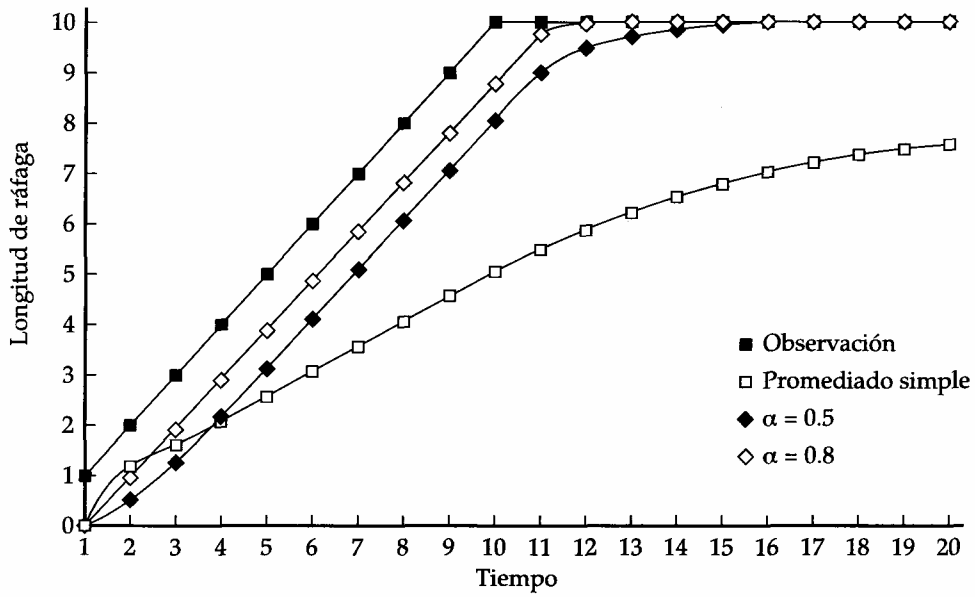
Un riesgo que existe en SPN es la posibilidad de inanición para los procesos largos mientras exista un flujo continuo de procesos más cortos. Por otro lado, aunque SPN reduce el sesgo favorable a los procesos largos, no es conveniente para entornos de tiempo compartido o de procesamiento de transacciones, debido a la ausencia de apropiación. Volviendo al análisis del peor caso, descrito en el estudio del FCFS, los procesos A, B, C y D ejecutarían también en el mismo orden, penalizando en exceso al proceso corto C.

Menor tiempo restante

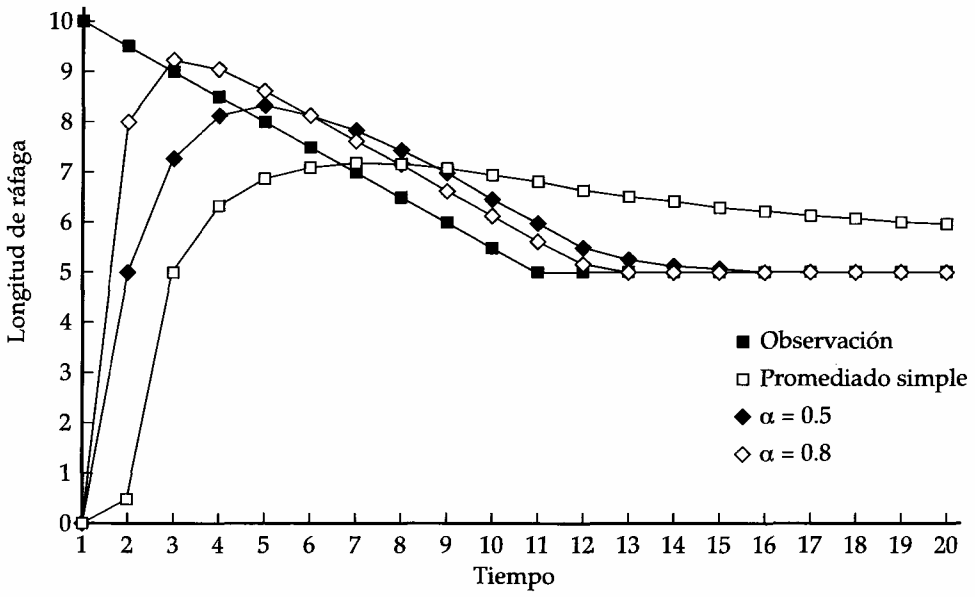
La política del menor tiempo restante (SRT, *Shortest Remaining Time) es* una versión apropiativa del SPN, en la que el planificador siempre elige al proceso que le queda menos tiempo esperado de ejecución. Cuando se añade un nuevo proceso a la cola de Listos, puede quedarle un tiempo esperado de ejecución menor que al proceso que está ejecutándose en ese momento. Por consiguiente, el planificador puede apropiarse del procesador siempre que un proceso nuevo esté listo. Como en el SPN, el planificador debe disponer de una estimación del tiempo de proceso para poder llevar a cabo la función de selección, existiendo el riesgo de inanición para procesos largos.

El SRT no presenta el sesgo favorable a los procesos largos del PCFS. Al contrario que el turno rotatorio, no se generan interrupciones adicionales y, así, el coste se ve reducido. Por contra, se deben los tiempos de servicio transcurridos, lo que contribuye a la sobrecarga. El SRT también debe producir unos tiempos de retorno mejores que los del SPN, puesto que los trabajos cortos reciben una atención inmediata y preferente a los trabajos largos.

Nótese que, en el ejemplo, los tres procesos más cortos reciben un servicio inmediato, obteniendo cada uno un tiempo de retorno normalizado de 1,0.



(a) Función creciente



(b) Función decreciente

FIGURA 8.10 Utilización de medias exponenciales

Primero el de mayor tasa de respuesta

En la tabla 8.4, se ha empleado el tiempo de retomo normalizado, que es la razón entre el tiempo de retomo y el tiempo real de servicio, como valor a destacar. Para cada proceso individual, se desea minimizar esta razón, así como minimizar el valor medio de todos los procesos. Aunque ésta es una medida a posteriori, es posible aproximarla a una medida a priori, como el criterio de selección de un planificador no apropiativo. En concreto, considérese la siguiente tasa de respuesta (RR, *Response Ratio):*

$$RR = \frac{w + s}{s}$$

donde

w = tiempo consumido esperando al procesador

s = tiempo de servicio esperado

Si el proceso con este valor se expide inmediatamente, RR es igual al tiempo de retomo normalizado. Nótese que el valor mínimo de RR es 1,0, alcanzado cuando un proceso entra por primera vez en el sistema.

Hasta ahora, la regla de planificación ha sido: Cuando el proceso actual termina o se bloquea, se elige el proceso listo con un valor mayor de RR. Este método es atractivo porque tiene en cuenta la edad del proceso. Aunque se favorece a los trabajos más cortos (un denominador menor produce una razón mayor), el envejecimiento sin que haya servicio incrementa el valor de la razón, de forma que los procesos más largos pasen finalmente primero, en competición con los más cortos.

El tiempo esperado de servicio debe estimarse antes de emplear la técnica de la mayor tasa de respuesta (HRRN, *Highest Response Ratio Next*), como ya ocurría con SRT y SPN.

Realimentación

Si no se dispone de ninguna información sobre la longitud relativa de los diversos procesos, no se puede emplear ninguno de los algoritmos anteriores (SPN, SRT y HRRN). Otra forma de dar preferencia a los trabajos más cortos consiste en penalizar a los trabajos que han estado ejecutando por más tiempo. Dicho de otro modo, si no es posible utilizar como base el tiempo de ejecución restante, se empleará el tiempo de ejecución consumido hasta el momento.

La forma de llevar a cabo lo anterior es la siguiente. La planificación es apropiativa y se emplea un mecanismo dinámico de prioridades. Cuando un proceso entra por primera vez en el sistema, se sitúa en RQ0 (ver figura 8.4). Cuando vuelve al estado de Listo, después de su primera ejecución, se incorpora a RQ1. Después de cada ejecución siguiente, se le degradará al nivel inmediatamente inferior de prioridad. Un proceso corto terminará rápidamente, sin descender demasiado en la jerarquía de las colas de Listos. Un proceso largo será gradualmente llevado hacia abajo. Así pues, se favorece a los procesos más nuevos y cortos antes que a los más viejos y largos. Se usará un simple mecanismo de FCFS dentro de cada cola, excepto en la de menor prioridad. Una vez en la cola de menor prioridad, un proceso no puede descender, sino que vuelve a la misma cola repetidamente hasta completar su ejecución. Por tanto, esta cola se trata con tumo rotatorio.

La figura 8.11 ilustra el mecanismo de planificación con realimentación, mostrando el camino que sigue un proceso a través de las colas[2] Este método se denomina realimentación multinivel (FB, *Feedback),* lo que quiere decir que el sistema operativo asigna el procesador a un proceso y, cuando el proceso se bloquea o es expulsado, lo devuelve a una determinada cola de prioridad.

Existe un gran número de variantes a este esquema. Una versión simple consiste en hacer apropiación de la misma forma que en el tumo rotatorio, es decir, a intervalos periódicos. En el ejemplo se aplica (figura 8.5 y tabla 8.4) con un cuanto de una unidad de tiempo. Nótese que, en este caso, el comportamiento es similar al del tumo rotatorio con un cuanto de tiempo de uno.

Un problema que presenta el sencillo esquema anterior es que el tiempo de retomo de los procesos mayores puede alargarse de forma alarmante. En realidad, puede ocurrir inanición si llegan regularmente nuevos trabajos al sistema. Para compensar, se puede variar el tiempo de apropiación en función de la cola: Un proceso planificado para RQ1 dispone de 1 unidad de tiempo de ejecución hasta ser expulsado; a un proceso planificado para RQ2 se le permite ejecutar durante 2 unidades de tiempo y así sucesivamente. En general, un proceso planificado para RQ puede ejecutar 2 unidades de tiempo antes de ser expulsado. En la figura 8.5 y la tabla 8.4 se ilustra este esquema.

Incluso permitiendo una mayor asignación de tiempo a las prioridades menores, los procesos largos también pueden sufrir inanición. Un posible remedio consiste en promocionar un proceso a una cola de mayor prioridad después de que esté esperando servicio en su cola actual durante un cierto tiempo.

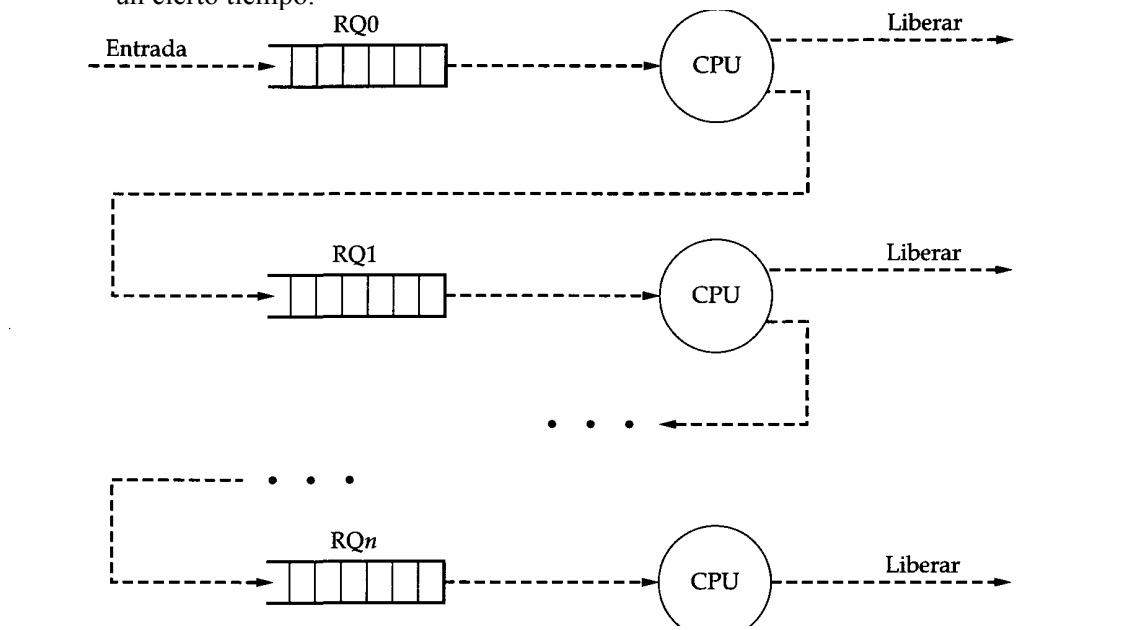


FIGURA 8.11 Planificación con realimentación

[2] *Las líneas de puntos se emplean para hacer énfasis en que se trata de un diagrama de tiempos y no de una representación estática de las transiciones posibles, como en la figura 8.4.*

Comparativa de rendimiento

Ciertamente, el rendimiento de las distintas políticas de planificación es un factor crítico en la elección de una política. Sin embargo, es imposible hacer una comparación definitiva porque el rendimiento relativo depende de una gran variedad de factores, incluyendo la distribución de probabilidad de los tiempos de servicios de los procesos, la eficiencia de la planificación y de los mecanismos de cambio de contexto, la naturaleza de las peticiones de E/S y el rendimiento del subsistema de E/S. No obstante, a continuación se intentarán esbozar algunas conclusiones generales.

Análisis de colas

En esta sección se hará uso de fórmulas básicas de la teoría de colas, haciendo la suposición habitual de que las llegadas siguen una Poisson y los tiempos de servicio una exponencial. Puede encontrarse un resumen de estos conceptos en el Apéndice 8A.

En primer lugar, se hará la observación de que cualquier disciplina de planificación que elija el siguiente elemento a servir independientemente del tiempo de servicio cumple la siguiente relación:

$$\frac{t_q}{s} = \frac{1}{1-p}$$

donde

t_q = tiempo de retomo; tiempo total en el sistema, espera más ejecución
s = tiempo medio de servicio; tiempo medio consumido en el estado de Ejecución
p = utilización del procesador

En particular, un planificador basado en prioridades, en el que la prioridad de cada proceso se asigna independientemente del tiempo esperado de servicio, proporciona el mismo tiempo medio de retomo y tiempo medio de retomo normalizado que un simple FCFS. Es más, la presencia o ausencia de apropiación no marca diferencias entre las medias.

Con la excepción del tumo rotatorio y del FCFS, las diversas disciplinas de planificación vistas hasta ahora hacen elecciones en función del tiempo esperado de servicio. Por desgracia, resulta bastante difícil construir modelos analíticos fiables para estas disciplinas. Sin embargo, es posible hacerse una idea del rendimiento relativo de dicho algoritmo de planificación en comparación con el FCFS, considerando una planificación por prioridades donde la prioridad está en función del tiempo de servicio.

Si la planificación se hace en función de prioridades y si los procesos se asignan a una clase de prioridad según su tiempo de servicio, entonces aparecen diferencias. La tabla 8.5 muestra las fórmulas resultantes con dos clases de prioridad, con tiempos de servicio diferentes para cada clase. Estos resultados se pueden generalizar a cualquier número de clases de prioridad (por ejemplo, véase [MART72] para un resumen de dichas fórmulas). Nótese que las fórmulas son diferentes para la planificación no apropiativa y apropiativa. En el último caso, se supone que un proceso de menor prioridad es interrumpido inmediatamente cuando uno de mayor prioridad está listo.

TABLA 8.5 FÓRMULAS PARA COLAS DE DOS PRIORIDADES CON UN SOLO SERVIDOR

Suposiciones:
1. La llegada sigue una Poisson.
2. Los elementos de prioridad j se sirven antes que los de prioridad $(j + 1)$.
3. Los elementos no son interrumpidos durante el servicio.
4. Los elementos de igual prioridad se expiden según FIFO.
5. Ningún elemento abandona la cola (se retrasan las perdidas).

(a) Fórmulas Generales

$$\lambda = \lambda_1 + \lambda_2$$

$$\rho_1 = \lambda_1 s_1;\ \rho_2 = \lambda_2 s_2$$

$$\rho = \rho_1 + \rho_2$$

$$s = \frac{\lambda_1}{\lambda} s_1 + \frac{\lambda_2}{\lambda} s_2$$

$$t_q = \frac{\lambda_1}{\lambda} t_{q1} + \frac{\lambda_2}{\lambda} t_{q2}$$

(b) Sin interrupciones; tiempo de servicio exponencial

$$t_{q1} = 1 + \frac{\rho_1 t_{s1} + \rho_2 t_{s2}}{1 - \rho_1}$$

$$t_{q2} = 1 + \frac{t_{q1} - 1}{1 - \rho}$$

(c) Disciplina de colas de reanudación apropiativa tiempo de servicio exponencial

$$t_{q1} = 1 + \frac{\rho_1 t_{s1}}{1 - \rho_1}$$

$$t_{q2} = 1 + \frac{1}{1 - \rho_1}\left(\rho_1 t_{s1} + \frac{\rho t_s}{1 - \rho}\right)$$

Como ejemplo, considérese el caso de dos clases de prioridad, con igual número de llegadas de procesos a cada clase y con tiempo medio de servicio para la clase de inferior prioridad cinco veces mayor al de la clase de mayor prioridad. De este modo, se escoge dar preferencia a los procesos cortos. La figura 8.12 muestra el resultado global. Si se da preferencia a los procesos más cortos, se mejora el tiempo medio de retomo normalizado. Como podría esperarse, la mejora es mayor cuando se emplea apropiación. Nótese, sin embargo, que el rendimiento total no se ve muy afectado.

Sin embargo, surgen diferencias significativas cuando se consideran las dos clases de prioridad de forma separada. La figura 8.13 muestra los resultados para los procesos más cortos y de prioridad superior. Para comparar, la línea superior del gráfico indica que no se emplean las prioridades, sino simplemente se observa el rendimiento relativo de la mitad de los procesos que presentan menor tiempo de procesamiento. Las otras dos líneas indican que a estos procesos se les asigna una prioridad mayor. Cuando el sistema ejecute con planificación no apropiativa por prioridades, las mejoras son significativas y son, incluso mayores cuando se emplea apropiación.

La figura 8.14 muestra el mismo análisis para los procesos largos y de prioridad inferior. Como se podía esperar, dichos procesos sufren una degradación del rendimiento cuando se utiliza planificación por prioridades.

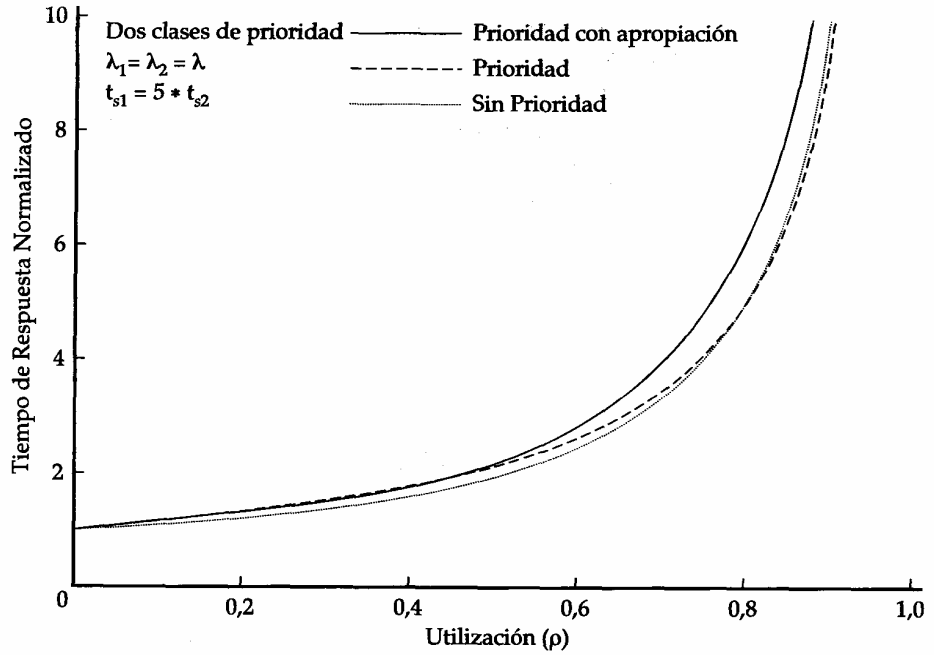


FIGURA 8.12 Tiempos totales de respuesta normalizados

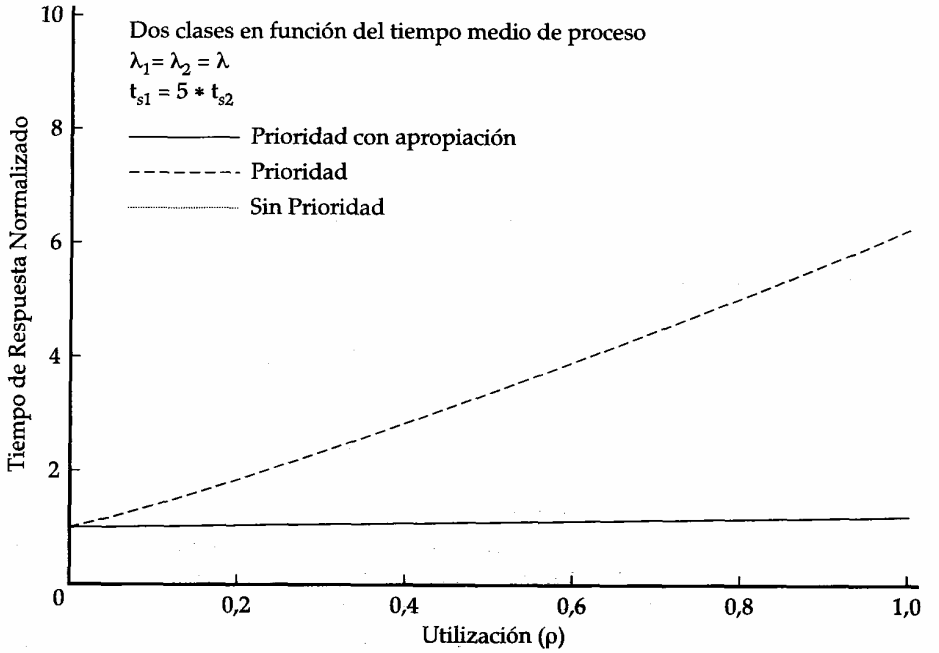


FIGURA 8.13 Tiempos de respuesta normalizados para procesos más cortos

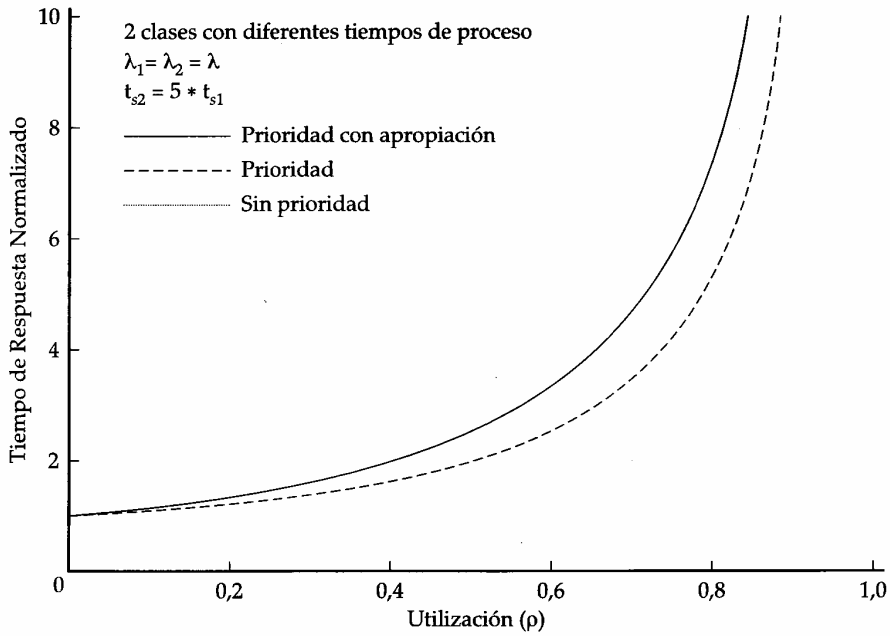


FIGURA 8.14 Tiempos de respuesta normalizados para procesos más largos

Modelos de Simulación

Algunas de las dificultades de los modelos analíticos son superadas mediante simulación discreta de sucesos, lo que permite modelar un amplio número de políticas. La desventaja de la simulación radica en que los resultados de una determinada ejecución son aplicables sólo a un conjunto de procesos en particular y bajo un conjunto particular de supuestos. No obstante, se pueden obtener revelaciones útiles.

Los resultados de un estudio al respecto se encuentran en [FINK88]. La simulación comprendía 50.000 procesos con una tasa de llegada de X = 0,8 y un tiempo de servicio medio de *s* = 1. Así pues, se supone que la utilización del procesador es de p = Xí = 0,8. Se debe destacar que sólo se mide la utilización en un momento dado.

Para presentar los resultados, los procesos se agruparon en percentiles del tiempo de servicio, cada uno de los cuales tenía 500 procesos. Por tanto, los 500 procesos con menor tiempo de servicio estaban en el primer percentil; sin contar estos, los 500 procesos restantes con menor tiempo de servicio estaban en el segundo percentil y así sucesivamente. Este agrupamiento permitió contemplar los efectos sobre los procesos de varias políticas en función de la longitud de los procesos.

La figura 8.15 muestra el tiempo de retomo normalizado y la figura 8.16 ofrece el tiempo medio de espera. Observando el tiempo de retomo, se puede comprobar que el rendimiento del FCFS es muy desfavorable, habiendo una tercera parte de los procesos con un tiempo de retorno normalizado superior a 10 veces el tiempo de servicio. Es más, estos son los procesos más cortos. Por otro lado, el tiempo absoluto de espera es uniforme, como se esperaba, puesto

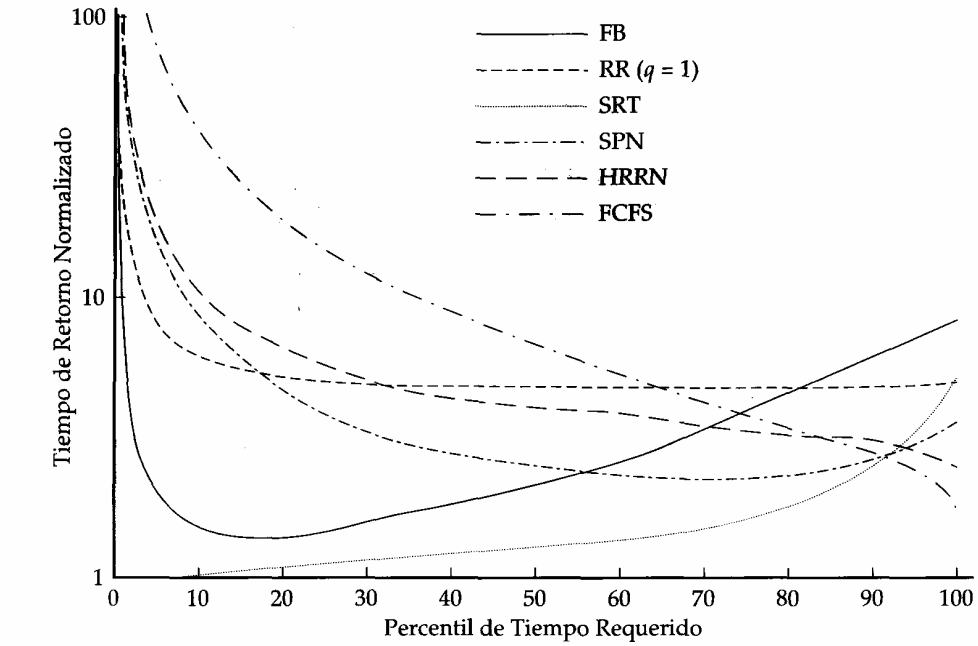


FIGURA 8.15 Resultados de la simulación para el tiempo de retorno normalizado

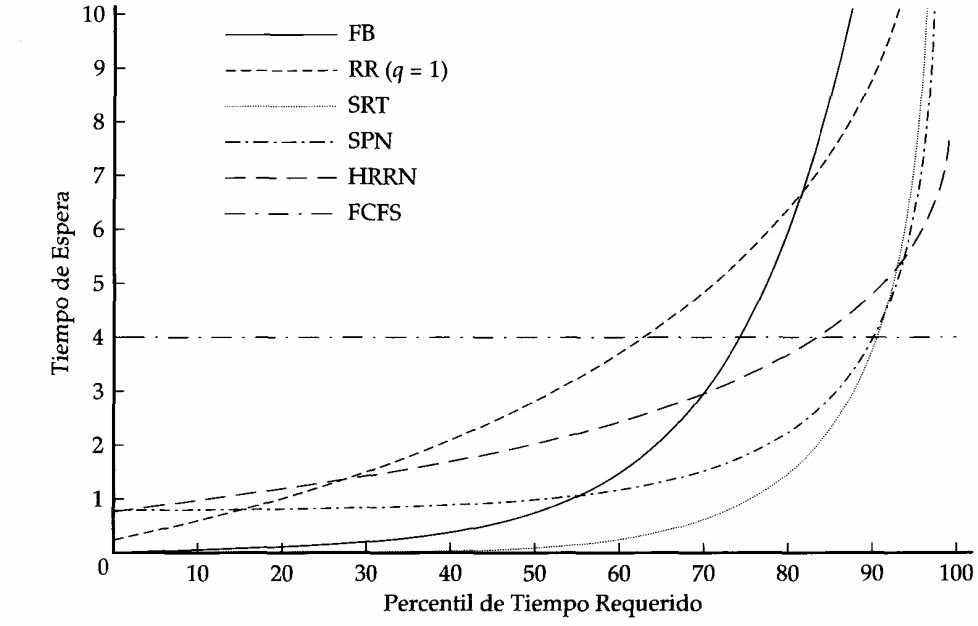


FIGURA 8.16 Resultados de la simulación para el tiempo de espera

que la planificación es independiente del tiempo de servicio. Las figuras muestran un tumo rotatorio con un cuanto de 1 unidad de tiempo. Excepto para los procesos más cortos, que ejecutan en menos de 1 cuanto, el RR produce un tiempo de retomo normalizado en tomo a 5 para todos los procesos, tratando a todos por igual. El algoritmo de primero el proceso más corto (SPN) obtiene mejor rendimiento que el tumo rotatorio, excepto para los procesos más cortos. El del menor tiempo restante (SRT), que es la versión apropiativa del SPN, da un rendimiento mejor que el SPN, excepto para el 7% de los procesos (los más largos). Se ha visto que, con políticas no apropiativas, FCFS favorece a los procesos largos y SPN a los cortos. La política de primero la mayor tasa de respuesta (HRRN) intenta establecer un compromiso entre ambos, lo que queda confirmado en las figuras. Por último, la figura muestra la planificación con realimentación con un cuanto uniforme y fijo para cada cola. Como se esperaba, la realimentación (FB) funciona bastante bien para los procesos cortos.

Planificación por reparto equitativo

Todos los algoritmos de planificación expuestos hasta ahora tratan el conjunto de procesos Listos como una única reserva de procesos de donde se selecciona el que pasará a estar Ejecutando. Esta reserva puede desglosarse por prioridades pero es, normalmente, homogénea.

Sin embargo, en un sistema multiusuario, si las aplicaciones o los trabajos de los usuarios pueden organizarse en forma de varios procesos (o hilos), se dispone de una estructura para el conjunto de procesos que no se identifica con ningún planificador tradicional. Desde el punto de vista del usuario, el interés no está en cómo se comporta un proceso en particular, sino en cómo se comporta el conjunto de procesos de usuario que constituyen una aplicación. Así pues, sería interesante poder tomar decisiones de planificación en función de estos grupos de procesos. Este método se conoce generalmente como *planificación por reparto equitativo* (FSS, *Fair-share Scheduling).* Es más, el concepto puede ampliarse a grupos de usuarios, incluso si cada usuario está representado por un solo proceso. Por ejemplo, en un sistema de tiempo compartido, sería conveniente considerar a todos los usuarios de un determinado departamento como miembros del mismo grupo. Las decisiones de planificación podrían intentar dar a cada grupo un servicio similar. Es decir, si se conecta al sistema un gran número de personas de un departamento, sería bueno comprobar que la degradación en el tiempo de respuesta afecta principalmente a los miembros de dicho departamento, en lugar de afectar a usuarios de otros departamentos.

El término **reparto equitativo** hace referencia a la filosofía del planificador. Cada usuario tiene asignado algún tipo de ponderación, que indica la parte de los recursos del sistema para el usuario como una fracción de la utilización total de dichos recursos. En particular, cada usuario dispone de una parte del procesador. Este esquema debe funcionar de una forma más o menos lineal, por lo que si un usuario A tiene un peso dos veces mayor que el de un usuario B, entonces, a la larga, el usuario A debe poder hacer el doble de trabajo que B. El objetivo de un planificador por reparto equitativo es supervisar el uso, de forma que se asignen menos recursos a los usuarios que han consumido más de lo que les corresponde y más recursos a los que han consumido menos de lo que le corresponde.

Se han realizado un buen número de propuestas de planificadores por reparto equitativo [HENR84, KAY88, LARM75, WOOD86]. En esta sección, se describirá el esquema propuesto por [HENR84], que se encuentra implementado en varios sistemas UNIX.

En las decisiones de planificación, la FSS tiene en cuenta el historial de ejecución de un

grupo afín de procesos, junto con el historial de ejecución individual de cada proceso. El sistema divide la comunidad de usuarios en un conjunto de grupos equitativos y reserva un parte del procesador para cada grupo. Así pues, podría haber cuatro grupos, donde cada uno dispusiera de un 25% de utilización del procesador. De hecho, cada grupo equitativo dispone de un sistema virtual que ejecuta proporcionalmente más lento que el sistema completo.

La planificación se lleva a cabo por prioridades, teniendo en cuenta la prioridad básica del proceso, su utilización reciente de la CPU y la utilización reciente de la CPU por parte del grupo al que pertenece. Cuanto mayor es el valor numérico de la prioridad, menor es ésta. Las fórmulas siguientes se aplican al proceso *j* del grupo *k:*

$$P_j(i) = Base_j + \frac{CPU_j(i-1)}{2} + \frac{GCPU_k(i-1)}{4 \times W_k}$$

$$CPU_j(i) = \frac{U_j(i-1)}{2} + \frac{CPU_j(i-1)}{2}$$

$$GCPU_k(i) = \frac{GU_k(i-1)}{2} + \frac{GCPU_k(i-1)}{2}$$

donde

$P(i)$ = Prioridad del proceso *j* al principio del intervalo *i*

Base *j* = Prioridad de base del proceso

$Uj(i)$ = Utilización de CPU del proceso *j* en el intervalo *i*

$GU_K\ (i)$ = Utilización total de CPU de todos los procesos del grupo *k* en el intervalo *i*

CPUj(i) = Media ponderada exponencial de la utilización de CPU del proceso *j* en el intervalo *i*

$GCPU_k(i)$ = Media ponderada exponencial de la utilización total de CPU del proceso y en el intervalo *i*

W_k = Peso asignado al grupo *k,* con la restricción de $0 \leq W_k \leq 1$ y $\Sigma_k W_k = 1$.

Cada proceso tiene asignada una prioridad de base que desciende a medida que el proceso y el grupo al que pertenece utilizan la CPU. En ambos casos, se calcula una media de utilización de la CPU mediante un promediado exponencial, con a = 0,5. En el caso de la utilización del grupo, la media se normaliza dividiendo por el peso del grupo. Cuanto mayor es el peso asignado al grupo, menos afecta su utilización a la prioridad.

La figura 8.17 es un ejemplo en el que el proceso A está en un grupo y los procesos B y C en un segundo grupo, donde cada grupo tiene un peso de 0,5. Supóngase que todos los procesos tienen carga de CPU y normalmente están listos para ejecutar. Todos los procesos tienen una prioridad de base de 60. La utilización de la CPU se mide como sigue: Se interrumpe al procesador 60 veces por segundo; durante cada interrupción, se incrementa el campo de utilización de CPU del proceso en ejecución, así como

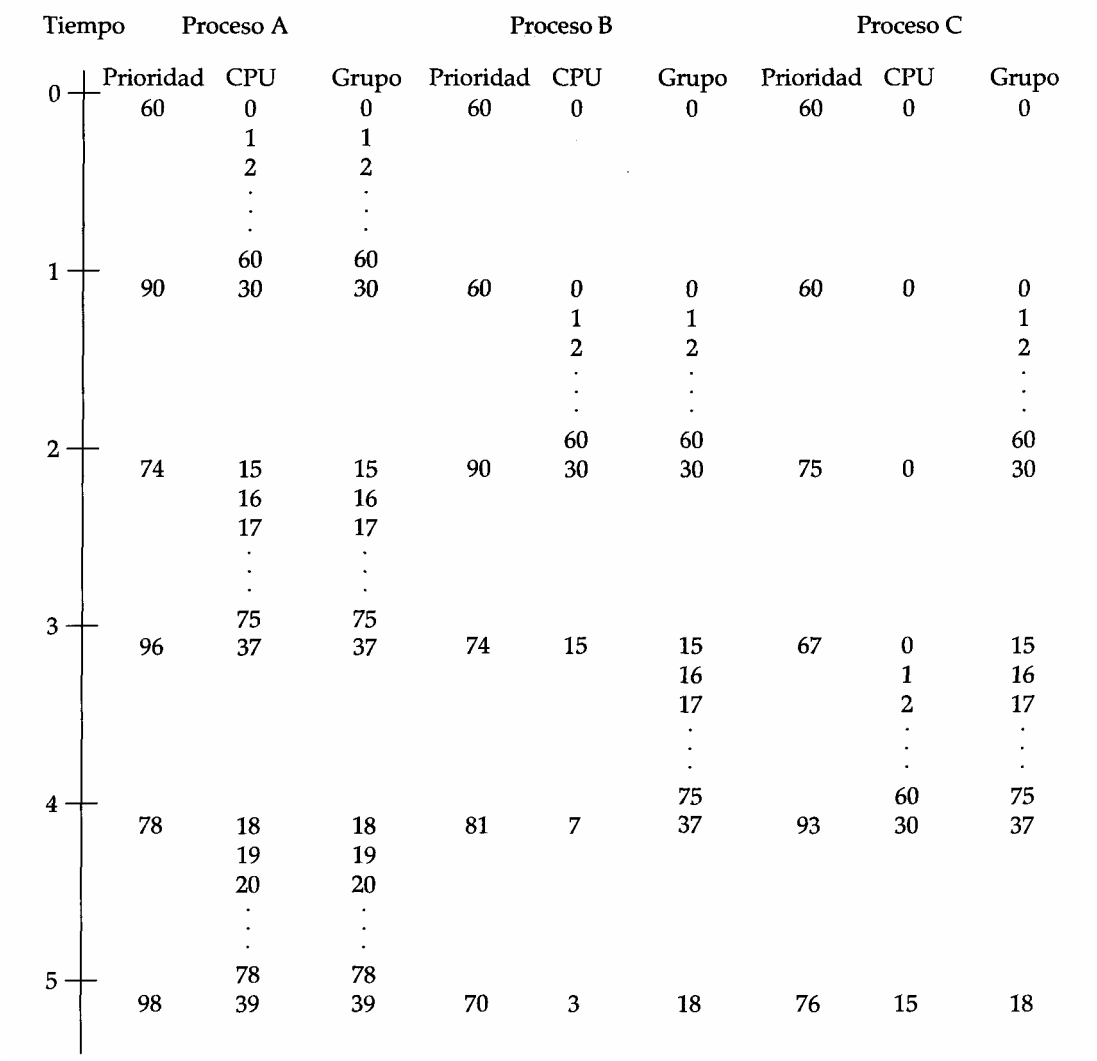


FIGURA 8.17 Ejemplo de planificador por reparto equitativo (Tres procesos, dos grupos [BACH86])

el campo del grupo correspondiente. Las prioridades se vuelven a calcular una vez por segundo.

En la figura, se planifica en primer lugar el proceso A. Al terminar el primer segundo, es expulsado. Los procesos B y C tienen ahora mayor prioridad y pasa a ejecutar el proceso B. Al final de la segunda unidad de tiempo, el proceso A tiene la mayor prioridad. Nótese que se repite la secuencia: El núcleo planifica los procesos en el orden: A, B, A, C, A, B y así sucesivamente.

De este modo, el 50% del tiempo de la CPU es asignado al proceso A, que constituye un grupo y el 50% a los procesos B y C, que forman otro grupo.

3.8 RESUMEN

El sistema operativo puede tomar tres tipos de decisiones de planificación que afectan a la ejecución de los procesos. La planificación a largo plazo determina cuándo se admiten nuevos procesos en el sistema. La planificación a medio plazo forma parte de la función de intercambio y determina cuándo se lleva parcial o totalmente un proceso a memoria principal para que pueda ser ejecutado. La planificación a corto plazo determina cuál de los procesos listos será ejecutado a continuación por el procesador. Este capítulo se centra en los asuntos relativos a la planificación a corto plazo.

En el diseño de un planificador a corto plazo se emplean un gran variedad de criterios. Algunos de estos criterios hacen referencia al comportamiento del sistema tal y como lo percibe el usuario (orientados a usuario), mientras que otros consideran la efectividad total del sistema para satisfacer las necesidades de todos los usuarios (orientados al sistema). Algunos de los criterios se refieren concretamente a medidas cuantitativas del rendimiento, mientras que otros son de tipo cualitativo. Desde el punto de vista del usuario, la característica más importante de un sistema es, en general, el tiempo de respuesta, mientras que desde el punto de vista del sistema es más importante la productividad o la utilización del procesador.

Se ha desarrollado una gran variedad de algoritmos para tomar las decisiones de planificación a corto plazo entre los procesos listos. Entre estos se incluyen:

- *Primero en llegar I primero en servirse:* Selecciona el proceso que lleva más tiempo esperando servicio.
- *Turno rotatorio:* Emplea un fraccionamiento del tiempo para hacer que los procesos se limiten a ejecutar en ráfagas cortas de tiempo, rotando entre los procesos listos.
- *Primero el proceso más corto:* Selecciona el proceso con menor tiempo esperado de ejecución, sin apropiarse de la CPU.
- *Menor tiempo restante:* Selecciona el proceso al que le queda menos tiempo esperado de ejecución en el procesador. Un proceso puede ser expulsado cuando otro proceso está listo.
- *Primero la mayor tasa de respuesta:* La decisión de planificación se basa en una estimación del tiempo de retomo normalizado.
- *Realimentación:* Establece un conjunto de colas de planificación y sitúa los procesos en las colas, teniendo en cuenta, entre otros criterios, el historial de ejecución.

La elección de un algoritmo de planificación dependerá del rendimiento esperado y de la complejidad de la implementación.

8.4 LECTURAS RECOMENDADAS

Prácticamente todos los libros de texto sobre sistemas operativos incluyen la planificación. En [CONW67], [KLEI76] y [STUC85] se presenta un análisis de colas riguroso de varias políticas de planificación.

CONW67 CONWAY, R., MAXWELL, W. y MILLER, L. *Theory of Scheduling.* Addison-Wesley, Reading, MA, 1967.
KLEI76 KLEINROCK, L. *Queuing Systems, Volume II: Computer Applications.* Wiley, Nueva York, 1976.
STUC85 STUCK, B. y ARTHURS, E. A *Computer and Communications Network Performance Analysis Primer.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

8.5

PROBLEMAS

8.1 Considérese el siguiente conjunto de procesos:

Nombre de Proceso	Instante de Llegada	Tiempo de Proceso
1	0	3
2	1	5
3	3	2
4	9	5
5	12	5

Realizar con este conjunto el mismo análisis que el representado en la tabla 8.4 y en la figura 8.5.

8.2 Repetir el problema 8.1 para el siguiente conjunto:

Nombre de Proceso	**Instante de Llegada**	**Tiempo de Proceso**
A	**0**	**1**
B	**1**	**100**
C	**2**	**1**
D	**3**	**100**

8.3 Probar que, entre los algoritmos de planificación no apropiativos, SPN proporciona el menor tiempo medio de espera.

8.4 Supóngase la siguiente secuencia de ráfagas de un proceso: 6,4,6,4,13,13,13 y supóngase que la estimación inicial es 10. Generar una gráfica similar a la de la figura 8.10

8.5 Considérese el siguiente par de ecuaciones como alternativa a la ecuación (3):

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n$$

$$X = \min(Slim, \max(Ilim, (\beta S_{n+i}))$$

donde *Slim* y *Ilim* son los límites superior e inferior elegidos previamente para el valor estimado de *T.* El valor de *X se* usa en lugar del valor de 5_{n+1} en el algoritmo de primero el proceso más corto. ¿Qué función realizan a y (i y cuál es el efecto de los valores superior e inferior de cada uno?

8.6 En un sistema monoprocesador no apropiativo, la cola de Listos contiene tres trabajos en un instante *t,* inmediatamente después de terminar un trabajo. Estos trabajos llegaron en los instantes t_j, t_j y t_j con tiempos estimados de ejecución r_j, r_j y r, respectivamente. La figura 8.18 muestra el incremento lineal de sus tasas de respuesta en el tiempo. Emplear este ejemplo para encontrar una variante de la planificación por tasas de respuesta, conocida como planificación por **tasa de respuestas minimax,** que minimiza la tasa de respuesta máxima para un lote de trabajos dado, ignorando las siguientes llegadas (Indicación:
Decidir primero el trabajo que debe planificarse el último).

8.7 Probar que el algoritmo por tasas de respuesta minimax del problema anterior minimiza la máxima tasa de respuesta para un lote de trabajos dado. (Indicación: Centrar la atención en el trabajo que obtenga la mayor tasa de respuesta y todos los trabajos ejecutados antes que él. Considérese el mismo subconjunto de trabajos planificados en otro orden y observar la tasa de respuesta del trabajo que se ejecuta el último. Nótese que este

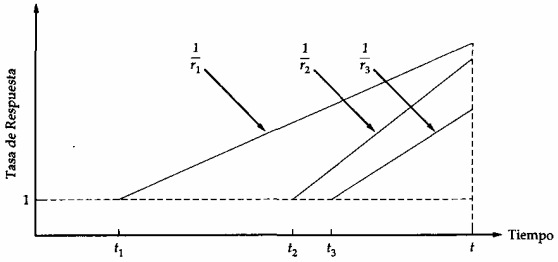


FIGURA 8.18 Tasa de respuesta en función de tiempo

subconjunto puede combinarse ahora con otros trabajos del conjunto total).

8.8 Si se define el tiempo de respuesta *R* como el tiempo medio total que un proceso pasa esperando y en servicio, demostrar que, para un FIFO con tiempo medio de servicio *s, se* cumple que $R = s / (1 - p)$.

8.9 Un procesador se multiplexa a una velocidad infinita entre todos los procesos presentes en una cola de Listos sin sobrecarga. (Este es un modelo idealizado de la planificación por tumo rotatorio entre procesos Listos que emplea fracciones de tiempo muy pequeñas en comparación con el tiempo de servicio). Demostrar que, para una fuente infinita de entrada según una distribución de Poisson, con tiempos de servicio exponenciales, el tiempo medio de respuesta R_x de un proceso con tiempo de servicio *x* viene dado por $R_x = x / (1 - p)$. (Indicación: Revisar las ecuaciones de colas del Apéndice 8A. Considerar después el tamaño medio de la cola *q* en el sistema a la llegada de los procesos).

8.10 En un sistema de colas, los trabajos nuevos deben esperar un momento antes de ser servidos. Mientras un trabajo espera, su prioridad se incrementa linealmente con el tiempo, a partir de cero y según una tasa a. Un trabajo espera hasta que su prioridad alcanza la de los trabajos en servicio; entonces, comienza a compartir el procesador equitativamente con los otros trabajos en servicio mediante un tumo rotatorio, mientras su prioridad continúa aumentando según una tasa menor ß. El algoritmo se denomina tumo rotatorio egoísta porque los trabajos en servicio tratan inútilmente de monopolizar el procesador incrementando su prioridad continuamente. Usar la figura 8.19 para demostrar que el tiempo medio de respuesta R_x para un trabajo con tiempo de servicio *x* viene dado por:

$$R_x = \frac{s}{1-\rho} + \frac{x-s}{1-\rho'}$$

donde

$$\rho = \lambda s \quad \rho' = \rho\left(1 - \frac{\beta}{\alpha}\right) \qquad 0 \le \beta \le \alpha$$

suponiendo que los tiempos de llegada y servicio están distribuidos exponencialmente con medias 1/Á. y *s,* respectivamente. (Indicación:

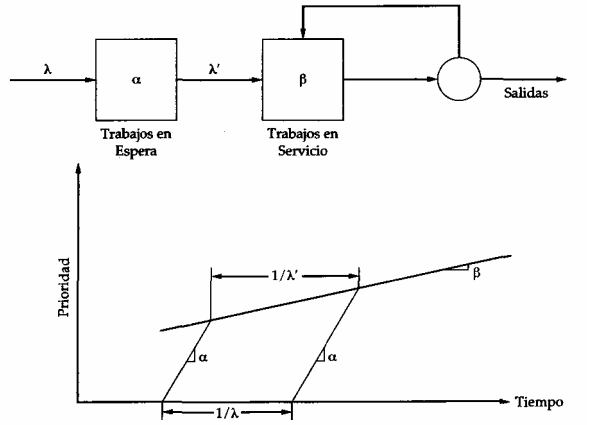


FIGURA 8.19 Round-robin egoísta

Considerar el sistema total y los dos subsistemas por separado).

8.11 Un sistema interactivo que emplea planificación por turno rotatorio e intercambio intenta dar respuestas seguras a las solicitudes triviales, de la forma siguiente: Después de completar un ciclo del tumo rotatorio entre todos los procesos Listos, el sistema determina la fracción de tiempo que asignar en el siguiente ciclo a cada proceso Listo dividiendo el tiempo máximo de respuesta por el número de procesos que solicitan servicio. ¿Es ésta un política práctica?

8.12 ¿Qué tipo de procesos se ve favorecido, en general, por un planificador por colas multinivel con realimentación, los procesos con carga de procesador o los procesos con carga de E/S? Explicar brevemente por qué.

8.13 En una planificación de procesos basada en prioridades, el planificador da el control a un proceso en particular sólo si no hay ningún proceso de mayor prioridad en la cola de Listos. Supóngase que no se emplea ninguna otra información en la decisión de planificación. Supóngase también que las prioridades de los procesos se establecen en el momento de la creación del proceso y que no cambian. En un sistema operativo con estas suposiciones, ¿Por qué seria "peligrosa" la solución de Dekker al problema de la exclusión mutua? Expliqúese comentando *qué* suceso no deseado podría ocurrir y *cómo* podría ocurrir.

APÉNDICE 8A

TIEMPO DE RESPUESTA

El tiempo de respuesta es el tiempo que tarda el sistema en reaccionar ante una entrada determinada. En una transacción interactiva, puede definirse como el tiempo transcurrido entre la última pulsación de tecla por parte del usuario y el comienzo de la visualización de resultados en el computador. Para diferentes tipos de aplicación, hacen falta definiciones ligeramente diferentes. En general, es el tiempo que tarda el sistema en responder a una solicitud de realizar una tarea en particular.

Idealmente, sería conveniente que el tiempo de respuesta de una aplicación fuese corto. Sin embargo, un tiempo de respuesta corto exige casi siempre un gran coste. Este coste proviene de dos fuentes:

- *Potencia computacional:* Cuanto más rápida es el computador, menor es el tiempo de respuesta. Por supuesto, incrementar la potencia de proceso significa incrementar el coste.
- *Requisitos contrapuestos:* Ofrecer tiempos de respuesta rápidos para unos procesos puede penalizar a otros.

Así pues, el valor de obtener un determinado tiempo de respuesta debe ser confrontado con el coste necesario para conseguirlo.

La tabla 8.6, basada en [MART88], enumera seis rangos de tiempo de respuesta. Las dificultades de diseño se presentan cuando se necesita un tiempo de respuesta inferior a 1 segundo. Las necesidades de tiempos inferiores a un segundo provienen de sistemas que controlan o, de algún modo, interactúan, con alguna actividad externa interrumpida, como puede ser una cadena de montaje. En este caso, los requisitos son fáciles de cumplir. Cuando se considera la interacción humana, como en una aplicación de entrada de datos, se trabaja en un rango de tiempos de respuestas conversacionales. En este caso, sigue existiendo la necesidad de un tiempo de respuesta bajo, pero puede resultar difícil calcular un tiempo aceptable.

Varios estudios [GUYN88, SHNE84, THAD81] han confirmado que un tiempo de respuesta rápido es la clave de la productividad de las aplicaciones interactivas. Estos trabajos demuestran que, cuando el computador y el usuario interactúan a un ritmo que asegura que ninguno debe esperar al otro, la productividad se incrementa de forma significativa, el coste del trabajo hecho por el computador disminuye y la calidad tiende a mejorar. Para la mayoría de las aplicaciones interactivas solían darse por válidos tiempos de respuesta relativamente lentos, de hasta 2 segundos, desde el momento en que la persona va a estar pensando en la siguiente tarea [MILL68]. Sin embargo, parece que la productividad aumenta cuando se consigue un tiempo de respuesta más rápido.

Los resultados aportados sobre tiempos de respuesta se apoyan en un análisis de transacciones interactivas. Una transacción consta de una orden del usuario desde un terminal y la respuesta del sistema. Esta es la unidad básica de trabajo para los usuarios de sistemas interactivos y puede dividirse en dos secuencias temporales (figura 8.20):

- *Tiempo de respuesta del sistema:* El intervalo de tiempo transcurrido entre el momento en que el usuario introduce una orden y el momento en que se muestra una respuesta completa en el terminal.
- *Tiempo de respuesta de usuario:* El intervalo de tiempo transcurrido entre el momento en que el usuario recibe una respuesta completa a una orden e introduce la orden siguiente. Este tiempo suele conocerse como *tiempo para pensar.*

TABLA 8.6 RANGOS DE TIEMPOS DE RESPUESTA [MART88]

Más de 15 sg.

Se descarta la interacción conversacional. Para determinados tipos de aplicaciones, algunos usuarios pueden conformarse con sentarse delante de un terminal esperando durante más de 15 sg. una respuesta a una simple petición. Sin embargo, para personas ocupadas, un cautiverio de más de 15 sg. resulta intolerable. Si se producen estos retardos, el sistema debe diseñarse para que el usuario pueda acudir a otras actividades y solicitar la respuesta algo más tarde.

Más de 4 sg.

Generalmente, son demasiado largos para una conversación, obligando al operador a retener la información en una memoria a corto plazo (memoria del operador, no del computador). Dichos retardos pueden inhibir las tareas de solución de problemas y frustrar las tareas de entrada de datos. Sin embargo, después de la toma de contacto, se pueden tolerar retardos entre 4 y 15 sg.

De 2 a 4 sg.

Un retardo de más de 2 sg. puede inhibir las operaciones del terminal que necesiten un alto nivel de concentración. Un retardo de 2 a 4 sg. en un terminal puede parecer sorprendentemente grande cuando el usuario está absorto y comprometido emocionalmente a terminar lo que está haciendo. Un retardo dentro de este rango puede ser aceptable tras haberse producido una mínima toma de contacto.

Menos de 2 sg.

Cuando el usuario de un terminal tiene que recordar la información de varias respuestas, el tiempo de respuesta debe ser corto. Cuanto más detallada sea la información a recordar, mayor es la necesidad de respuestas que están por debajo de 2 sg. Para realizar tareas en el terminal, los 2 sg. representan un límite considerable de tiempo de respuesta.

Inferiores a *1* sg.

Cierto tipo de trabajos de razonamiento intensivo, especialmente con aplicaciones gráficas, necesitan tiempos de respuesta muy cortos para mantener el interés y la atención del usuario durante largos periodos.

Décimas de segundo

La respuesta al pulsar una tecla y ver el carácter en la pantalla o al señalar un objeto en la pantalla con el ratón deben de ser casi instantáneas, de menos de 0,1 sg. tras de la acción. La interacción con el ratón debe ser extremadamente rápida si al diseñador no se le permite utilizar sintaxis extrañas (con órdenes, mnemónicos, símbolos de puntuación, etc.)

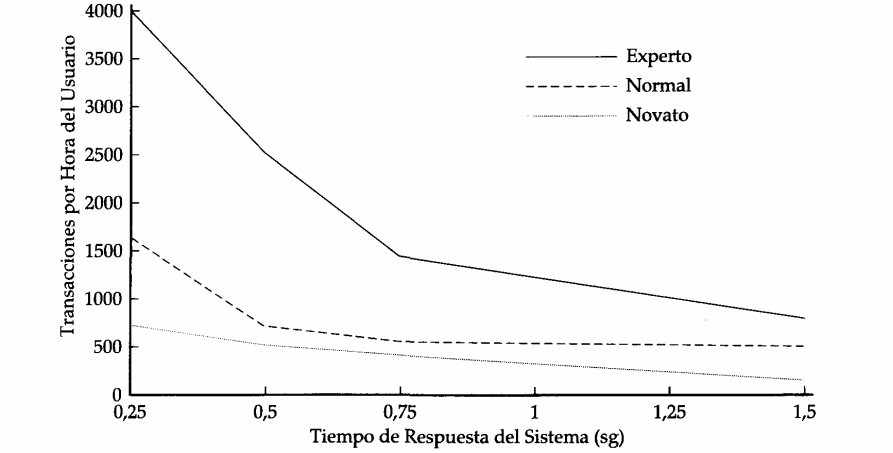


FIGURA 8.20 Resultados del tiempo de respuesta para gráficos de alto rendimiento

Como ejemplo del efecto de reducir el tiempo de respuesta del sistema, la figura 8.21 muestra el resultado de un estudio llevado a cabo con ingenieros que emplean programas gráficos de diseño asistido por computador para el diseño de placas y circuitos integrados [SMIT83]. Cada transacción consta de una orden del ingeniero que modifica de alguna forma la imagen gráfica de la pantalla. Los resultados muestran que el porcentaje de transacciones aumenta a medida que el tiempo de respuesta del sistema baja y asciende drásticamente una vez que el tiempo de respuesta cae por debajo de 1 sg. Lo que ocurre es que, como el tiempo de respuesta del sistema baja, también lo hace el tiempo de respuesta del usuario, resultado del efecto de la memoria a corto plazo y la duración de la atención humana.

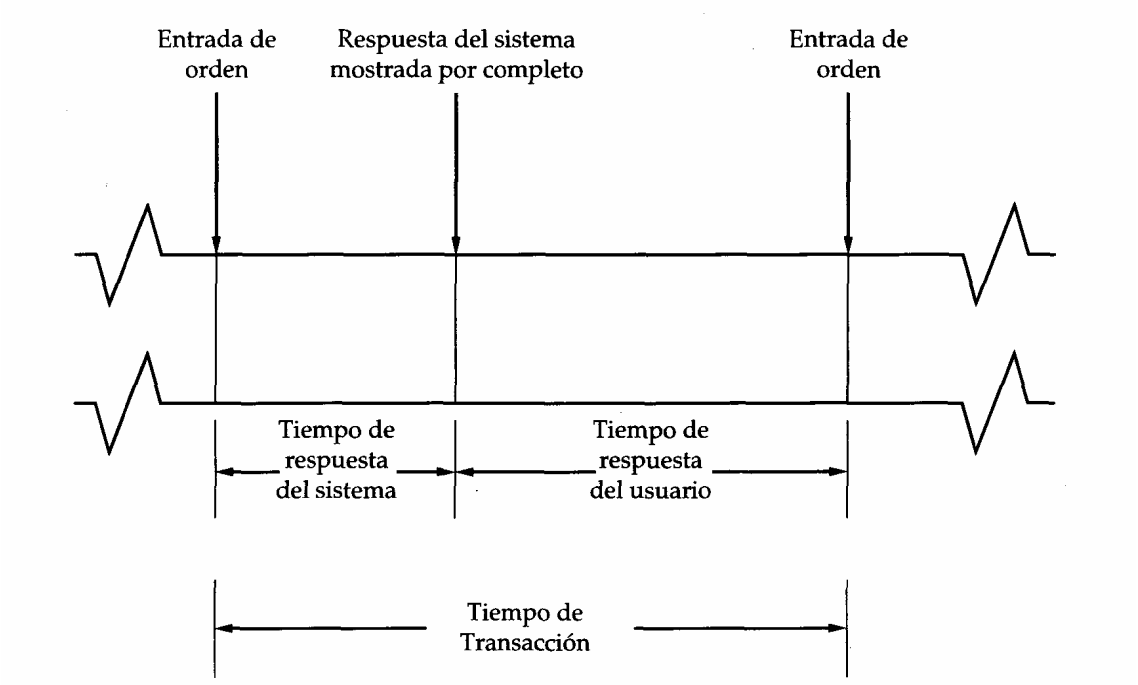


FIGURA 8.21 Elementos de una transacción interactiva

CAPÍTULO 9

Planificación de multiprocesadores y en tiempo real

Este capítulo continúa el recorrido por la planificación de procesos. El capítulo comienza con un examen de los puntos planteados por la existencia de más de un procesador. Se estudiarán varios aspectos del diseño. A continuación se considerará la planificación de procesos en un sistema multiprocesador. Después, se examinarán los aspectos de diseño que son algo diferentes para planificación de hilos de multiprocesadores. La segunda sección de este capítulo trata sobre la planificación en tiempo real. La sección comienza con una discusión de las características de los procesos de tiempo real para después atender a la esencia de la tarea de planificación. Se examinan dos métodos de planificación en tiempo real, la planificación por plazos *(deadline scheduling)* y la planificación monótona en frecuencia *(rate monotonic scheduling).*

9.1

PLANIFICACIÓN DE MULTIPROCESADORES

Cuando un sistema informático tiene más de un único procesador, aparecen varios elementos nuevos en el diseño de la tarea de planificación. Se va a comenzar con una breve introducción a los multiprocesadores y, después, se discutirá sobre lo que hay que tener en cuenta para llevar a cabo la planificación al nivel de los procesos o al nivel de los hilos.

Los sistemas multiprocesador pueden clasificarse de la siguiente manera:

- *Multiprocesador débilmente acoplado:* Consta de un conjunto de sistemas relativamente autónomos, donde cada procesador tiene su propia memoria principal y sus propios canales de E/S. Este tipo de configuración se aborda en el capítulo 12.
- *Procesadores especializados:* Similares a los procesadores de E/S. En este caso, hay un procesador principal, de propósito general; los procesadores especializados están controlados por el procesador principal y le ofrecen servicios. Los aspectos relativos a los procesadores de E/S se abordan en el capítulo 10.

- *Multiprocesador fuertemente acoplado:* Consta de un conjunto de procesadores que comparten una memoria principal común y se encuentran bajo el control integrado de un sistema operativo.

En esta sección, el interés se dirige a la última categoría y, especialmente, a los aspectos relacionados con la planificación.

Evolución de los multiprocesadores fuertemente acoplados

La arquitectura de los multiprocesadores ha estado al acecho durante muchos años. El primer sistema comercial fue el Burroughs D825, disponible en 1960, que incluía hasta cuatro procesadores. Hasta hace poco, el objetivo principal de los procesadores múltiples era ofrecer un rendimiento mejorado y fiabilidad en la multiprogramación:

- *Rendimiento:* Un único multiprocesador ejecutando en un sistema operativo multiprogramado ofrecerá mejor rendimiento que un sistema monoprocesador equivalente y puede ser más efectivo que varios sistemas monoprocesador.
- *Seguridad:* En un sistema fuertemente acoplado, si los procesadores funcionan por parejas, el fallo de un procesador sólo produce una degeneración del rendimiento en vez de la pérdida completa de servicio.

Aunque, en principio, era posible ejecutar en un multiprocesador aplicaciones que realizaban procesamiento paralelo, este modo de ejecución no es el habitual. En cambio, el punto fuerte está en ofrecer un sistema que sea lógicamente equivalente a un monoprocesador multiprogramado con un mayor rendimiento y seguridad. Este tipo de sistemas es aún la forma dominante de multiprocesadores comerciales.

En los últimos años, se ha incrementado el interés en el desarrollo de aplicaciones en que participen varios procesos o hilos; estas aplicaciones se conocen como **aplicaciones paralelas.** Una ventaja de esta idea, como se discutió en el capítulo 3, es que el diseño del software puede simplificarse si se compara a la construcción de la aplicación como un programa secuencial. Otra ventaja es que, en un multiprocesador, es posible ejecutar ciertas partes de la aplicación simultáneamente en distintos procesadores, para conseguir un mejor rendimiento. Este uso de los multiprocesadores complica considerablemente, como se verá, la función de planificación.

Granularidad

Una buena forma de caracterizar los multiprocesadores y situarlos en el contexto de otras arquitecturas es considerar la granularidad de la sincronización o frecuencia de sincronización entre los procesos de un sistema. Es posible distinguir cinco categorías de paralelismo que difieren en el grado de granularidad. Estas categorías se encuentran resumidas en la tabla 9.1, que está extraída de [GEHR87] y [WOOD89].

Paralelismo independiente

Con paralelismo independiente, no existe sincronización explícita entre los procesos. Cada uno representa una aplicación o trabajo separado e independiente. Un uso clásico de este tipo de paralelismo se da en sistemas de tiempo compartido. Cada usuario está ejecutando una aplicación en particular, como un procesador de texto o una hoja de cálculo. El multiprocesador ofrece el mismo servicio que un monoprocesador multiprogramado. Como hay más de un procesador disponible, el tiempo medio de respuesta de los usuarios será menor.

Tabla 9.1 Procesos y granularidad de la sincronización

Tamaño de Grano	Descripción	Intervalo de Sincronización (Instrucciones)
Fino	Paralelismo inherente en un único flujo de instrucciones	<20
Medio	Procesamiento paralelo o multitarea dentro de una aplicación individual	20-200
Grueso	Multiprocesamiento de procesos concurrentes en un entorno multiprogramado	200-2000
Muy Grueso	Proceso distribuido por los nodos de una red para formar un solo entorno de computación	2000-1M
Independiente	Varios procesos no relacionados	(N/A)

Es posible alcanzar un aumento similar de rendimiento proporcionando a cada usuario un computador personal o una estación de trabajo. Si van a compartirse archivos o alguna información, entonces se deben conectar los sistemas individuales en un sistema distribuido apoyado en una red. Este método se examina en el capítulo 12. Por otro lado, un único sistema multiprocesador compartido ofrece, en muchos casos, un coste mejor que un sistema distribuido, pudiendo así mejorar los discos y otros periféricos.

Paralelismo de grano grueso y muy grueso

Con el paralelismo de grano grueso y muy grueso, existe una sincronización entre los procesos pero a un nivel muy burdo. Este tipo de situación se maneja fácilmente con un conjunto de procesos concurrentes ejecutando en un monoprocesador multiprogramado y puede verse respaldado por un multiprocesador con escasos cambios o incluso ninguno en el software del usuario.

Un ejemplo simple de una aplicación que puede aprovechar la presencia de un multiprocesador es el presentado en [WOOD89], Woodbury y otros han desarrollado un programa que recibe una especificación de los archivos que necesitan ser recompilados para reconstruir una parte del software y determina cuales de estas compilaciones (normalmente todas) pueden ejecutarse simultáneamente. El programa crea entonces un proceso para cada compilación paralela. Los resultados de Woodbury y otros indican que la aceleración en un multiprocesador supera realmente la que podría esperarse añadiendo simplemente el número de procesadores usados. Estos resultados vienen dados por las sinergias en las caches del disco (un concepto tratado en el capítulo 10) y por la compartición del código del compilador, que se carga en memoria sólo una vez.

En general, cualquier conjunto de procesos concurrentes que necesiten comunicarse o sincronizarse pueden aprovechar el uso de arquitecturas de multiprocesadores. Un sistema distribuido puede ofrecer un soporte adecuado en caso de interacciones poco frecuentes entre los procesos. Sin embargo, si la interacción es algo más frecuente, la sobrecarga de comunicaciones a través de la red puede anular parte de la posible aceleración. En este caso, la organización del multiprocesador ofrece el soporte más efectivo.

Paralelismo de grano medio

En el capítulo 3 se estudiaron las aplicaciones que pueden implementarse de hecho como un conjunto de hilos, en vez de en un solo proceso. En este caso, el programador debe especificar explícitamente el posible paralelismo de la aplicación. Normalmente, hará falta un grado

más alto de coordinación e interacción entre los hilos de la aplicación, lo que lleva a un nivel de sincronización de grano medio.

Mientras que el paralelismo independiente, de grano grueso y de grano muy grueso puedan verse respaldados tanto en un monoprocesador multiprogramado como en un multiprocesador con poco o ningún impacto sobre la planificación, se debe revisar la planificación cuando se afronta la planificación de hilos. Puesto que los diversos hilos de una aplicación interactúan de forma muy frecuente, las decisiones de planificación que involucren a un hilo pueden afectar al rendimiento de la aplicación completa. Se volverá sobre este punto más adelante en esta sección.

Paralelismo de grano fino

El paralelismo de grano fino significa un uso del paralelismo mucho más complejo que el que se consigue con el uso de hilos. Si bien gran parte del trabajo se realiza en aplicaciones muy paralelas, este es un campo, hasta el momento, muy especializado y fragmentado, con varias soluciones diferentes. En [ALMA89] se presenta un buen estudio.

Elementos de diseño

En la planificación de un multiprocesador se deben considerar tres puntos interrelacionados:

- La asignación de procesos a los procesadores
- El uso de multiprogramación en los procesadores individuales
- La expedición real de los procesos

Considerando estos tres puntos, es importante tener en cuenta que el método a emplear dependerá, en general, del grado de granularidad de la aplicación y del número de procesadores disponibles.

Asignación de procesos a los procesadores

Si se supone que la arquitectura del multiprocesador es uniforme, en el sentido de que ningún procesador posee ninguna ventaja física en el acceso a memoria principal o a dispositivos de E/S, el método más simple consiste en tratar a los procesadores como un recurso reservado y asignar los procesos a los procesadores por demanda. La cuestión ahora es si la asignación debe ser estática o dinámica.

Si un proceso es asignado a un procesador de forma permanente, desde su activación hasta su terminación, entonces debe mantenerse una cola dedicada a corto plazo para cada procesador. Una ventaja de este método es que la función de planificación tiene un coste menor, puesto que la asignación al procesador se realiza una sola vez y para siempre. Además, el uso de procesadores dedicados permite una estrategia conocida como *planificación por grupos apandillas,* que se trata más adelante.

Una desventaja de la asignación estática es que un procesador puede estar desocupado, con su cola vacía, mientras que otro procesador tiene trabajos pendientes. Para prevenir esta situación, se puede usar una cola común. Todos los procesos van a una cola global y son ejecutados en cualquier procesador que esté disponible. De este modo, durante la vida de un proceso, éste puede ejecutarse en diferentes procesadores en momentos diferentes. Con acoplamiento muy fuerte y una arquitectura de memoria compartida, la información de contexto de todos los procesos se en-

cuentra disponible para todos los procesadores y, por tanto, el coste de la planificación de un proceso será independiente de la identidad del procesador en el que fue planificado.

Sin reparar en si los procesos están dedicados a los procesadores, hace falta una forma de asignar los procesos a los procesadores. Se pueden utilizar dos métodos: arquitectura maestro/esclavo y arquitecturas simétricas. En la arquitectura maestro/esclavo, las funciones clave del núcleo del sistema operativo siempre se ejecutan en un determinado procesador. Los otros procesadores pueden ejecutar sólo programas de usuario. El maestro es el responsable de la planificación de trabajos. Una vez que un proceso está activo, si el esclavo necesita un servicio (por ejemplo, una llamada de E/S), debe enviar una solicitud al maestro y esperar a que el servicio se lleve a cabo. Esta solución es bastante simple y exige una pequeña mejora en el sistema operativo multiprogramado del monoprocesador. La resolución de conflictos se simplifica puesto que un procesador tiene el control de toda la memoria y todos los recursos de E/S. Las desventajas de esta solución son dos: (1) un fallo en el maestro hace caer todo el sistema y (2), el maestro puede llegar a ser un cuello de botella del rendimiento.

En una arquitectura simétrica, el sistema operativo puede ejecutarse en cualquier procesador y cada procesador se autoplanifica con el conjunto de procesos disponibles. Esta solución complica el sistema operativo. El sistema operativo debe asegurarse de que dos procesadores no seleccionan el mismo proceso y que los procesos no se pierden de la cola. Las técnicas deben emplearse para resolver y sincronizar las solicitudes concurrentes de recursos.

Por supuesto, hay un amplio abanico de soluciones entre estos dos extremos. Unas hacen que un subconjunto de los procesadores se dediquen a ejecutar el núcleo en lugar de uno sólo. Otro método consiste simplemente en controlar las diferencias entre las necesidades de los procesos del núcleo y de otros procesos en función de prioridades y el historial de ejecución.

Uso de la multiprogramación en procesadores individuales

Cuando cada proceso se asigna estáticamente a un procesador durante todo su tiempo de vida, surge una nueva cuestión: ¿Puede estar multiprogramado dicho procesador? La primera reacción puede ser preguntarse por qué hace falta responder a la cuestión; puede parecer demasiado dispendioso vincular a un procesador con un único proceso cuando dicho proceso puede bloquearse frecuentemente esperando una E/S o por motivos de concurrencia o sincronización.

En los multiprocesadores tradicionales, con sincronización de grano grueso o independiente (ver tabla 9.1), está claro que cada procesador individual podría alternar entre varios procesos para conseguir una alta utilización y, por tanto, un mejor rendimiento. Sin embargo, cuando se trabaja con aplicaciones de grano medio ejecutándose en muchos procesadores, la situación es menos clara. Cuando hay varios procesadores disponibles, no resulta tan importante que cada procesador esté ocupado al máximo. En su lugar, se debe tratar de obtener el mejor rendimiento, en promedio, para las aplicaciones. Una aplicación que conste de varios hilos puede rendir poco a menos que todos sus hilos estén disponibles para ejecutar simultáneamente.

Expedición de un proceso

El último punto de diseño relacionado con la planificación de multiprocesadores es la selección real del proceso a ejecutar. Se ha visto que en un sistema monoprocesador multiprogramado el uso de prioridades o algoritmos de planificación sofisticados basados en la utiliza-

ción anterior pueden mejorar la simple estrategia FIFO de decisión. Cuando se considera un multiprocesador, esta complejidad puede ser innecesaria o incluso contraproducente. En el caso de la planificación tradicional de procesos, una técnica sencilla puede ser más efectiva y aportar menos sobrecarga. En los desarrollos recientes de planificación de hilos, entran en juego nuevos elementos que pueden ser más importantes que las prioridades o el historial de ejecución. Cada uno de esos temas se abordará en su momento.

Planificación de procesos

En la mayoría de los sistemas multiprocesador tradicionales, los procesos no se asignan a los procesadores de forma dedicada. En su lugar, hay una cola única para todos los procesadores o, si se utiliza algún tipo de esquema de prioridades, existirán varias colas, según la prioridad, alimentando todas a una reserva común de procesadores. En cualquier caso, es posible contemplar el sistema como una arquitectura de colas multiservidor.

Considérese el caso de un sistema dual de procesadores en el que cada procesador del sistema tiene la mitad de tasa de proceso que un sistema de procesador único. [SAUE81] realiza un análisis de colas que compara la planificación FCFS con el tumo rotatorio y con el algoritmo del menor tiempo restante. En el caso del tumo rotatorio, se supone que el cuanto de tiempo es grande comparado con la sobrecarga por cambio de contexto y pequeño comparado con el tiempo medio de servicio. Los resultados dependen de la variabilidad que se produzca en los tiempos de servicio. Una medida habitual de la variabilidad es el coeficiente de variación, C_s que se define como:

$$C_s = \frac{\sigma_s}{s}$$

donde

σ_s = desviación típica del tiempo de servicio
s = tiempo medio de servicio

Una razón de 1 correspondería a un tiempo de servicio exponencial. Sin embargo, la distribución de tiempo de servicio del procesador es normalmente más variable; no es extraño encontrar valores de 10 o más para C_s.

En la figura 9.1a se comparan la productividad del tumo rotatorio con la del FCFS en función de C_s. Nótese que la diferencia en los algoritmos de planificación es mucho menor en el caso del procesador dual. Con dos procesadores, un proceso único con un tiempo de servicio largo trastorna mucho menos en el caso del FCFS; los otros procesos pueden usar el procesador restante. En la figura 9. Ib se muestran resultados similares.

El estudio de [SAUE81] repite este análisis bajo una serie de supuestos sobre el grado de multiprogramación, la combinación de procesos con carga de E/S y con carga de CPU y el uso de prioridades. La conclusión general es que el algoritmo específico de planificación es mucho menos importante con dos procesadores que con uno.

Resulta evidente que esta conclusión se hace más sólida si se aumenta el número de procesadores. De este modo, una disciplina FCFS simple o el uso de un esquema FCFS con prioridades estáticas puede bastar en un sistema multiprocesador.

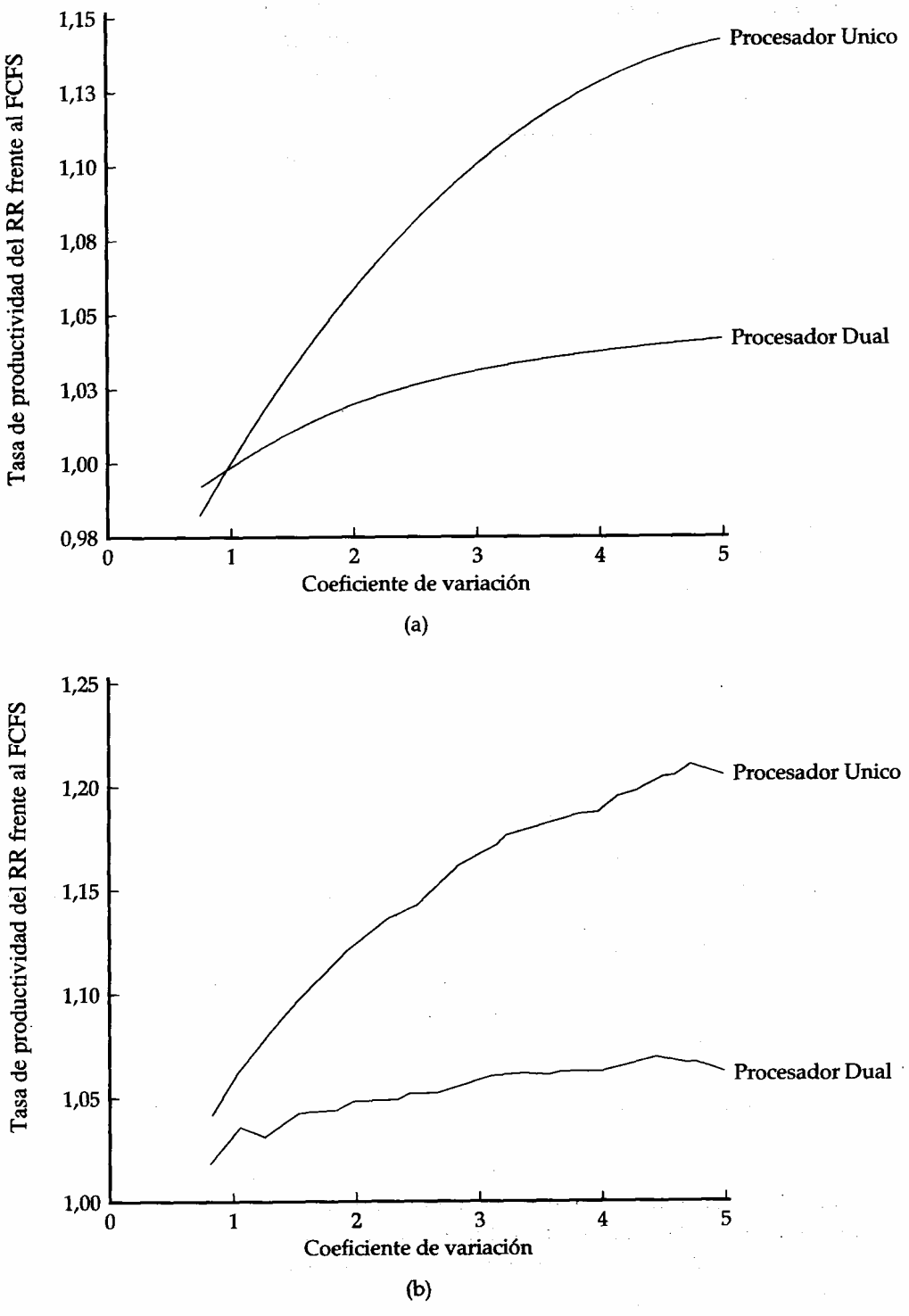


FIGURA 9.1 Comparación del rendimiento de la planificación para uno y dos procesadores

Planificación de hilos

Los hilos han llegado a ser cada vez más habituales en los nuevos sistemas operativos y lenguajes. Como se ha visto, con los hilos el concepto de ejecución se separa del resto de la definición de un proceso. Una aplicación puede implementarse como un conjunto de hilos que cooperan y ejecutan concurrentemente en el mismo espacio de direcciones.

En un monoprocesador, los hilos pueden usarse como una ayuda a la estructuración de un programa y para superponer la E/S y el procesamiento. Puesto que la penalización por el intercambio de hilos es mínima, comparada con el intercambio de procesos, estos beneficios se llevan a cabo a bajo coste. Sin embargo, toda la potencia de los hilos se hace más evidente en un sistema multiprocesador donde pueden emplearse hilos para obtener un paralelismo real en las aplicaciones. Si los diversos hilos de una aplicación se ejecutan simultáneamente en distintos procesadores, se posibilita un aumento drástico del rendimiento. Sin embargo, puede demostrarse que, en aplicaciones que exijan una interacción considerable entre los hilos (paralelismo de grano medio), pequeñas diferencias en la planificación y gestión de hilos pueden tener un importante significado en el rendimiento [ANDE89]. Actualmente se está investigando en el campo de la planificación de hilos en multiprocesadores. La discusión ofrecida aquí da una idea de los elementos y métodos clave.

Entre las diversas propuestas de planificación de hilos de multiprocesadores y de asignación de procesadores, destacan los siguientes métodos:

- *Compartición de carga:* Los procesos no se asignan a un procesador en particular. Se mantiene una cola global de hilos listos y cada procesador, cuando está ocioso, selecciona un hilo de la cola. El termino *compartición de carga* se emplea para distinguir esta estrategia del esquema de carga equilibrada en el que el trabajo se reparte de una forma más permanente (por ejemplo, véase [FEIT90a]).[1]
- *Planificación por grupos:* Se planifica un conjunto de hilos afines para su ejecución en un conjunto de procesadores al mismo tiempo, según una relación uno a uno.
- *Asignación dedicada de procesadores:* Es el enfoque opuesto a la autoplanificación y ofrece una planificación implícita definida por la asignación de hilos a los procesadores. En el tiempo que se ejecuta un programa, a cada programa se le asigna un número de procesadores igual al número de hilos que posea. Cuando el programa finaliza, los procesadores retoman a la reserva general para posibles asignaciones a otros programas.
- *Planificación dinámica:* El número de hilos en un programa puede cambiar en el curso de una ejecución.

Compartición de carga

La compartición de carga es quizá el método más simple y el que se puede traducir más fácilmente desde un entorno monoprocesador. Tiene las siguientes ventajas:

- La carga se distribuye uniformemente entre los procesadores, asegurando que ningún procesador esté ocioso mientras haya trabajo disponible.

' Determinados textos sobre este tema se refieren a este método como autoplanificación, ya que cada procesador se planifica a sí mismo, sin importar los demás. Sin embargo, este término también se usa en la bibliografía referente a los programas escritos en un lenguaje que permite al programador determinar la planificación (por ejemplo, véase [FOST91])

• No es necesario un planificador centralizado. Cuando un procesador está libre, la rutina de planificación del sistema operativo se ejecuta en este procesador para seleccionar un nuevo hilo.

• La cola global puede organizarse y se puede acceder a ella por medio de uno de los esquemas tratados en la sección 6.2, incluyendo los esquemas basados en prioridades y los que tienen en cuenta el historial de ejecución o las solicitudes de proceso por adelantado.

• [LEUT90] analiza tres versiones diferentes de compartición de carga:

• *Primero en llegar/primero en servirse (FCFS):* Cuando llega un trabajo, cada uno de sus hilos se sitúa consecutivamente al final de la cola compartida. Cuando un procesador pasa a estar ocioso, toma el siguiente hilo listo y lo ejecuta hasta que finalice o se bloquee.

• *Primero el de menor número de hilos:* La cola de Listos compartida se organiza como una cola de prioridades, en la que la prioridad más alta se asigna a los hilos de los trabajos con el menor número de hilos sin planificar. Los trabajos de la misma prioridad se ordenan según el orden de llegada. Como con FCFS, un hilo planificado ejecuta hasta que finaliza o se bloquea.

• *Primero el de menor número de hilos, con apropiación:* La mayor prioridad se da a los trabajos con el menor número de hilos sin terminar. La llegada de un trabajo con un número de hilos menor que un trabajo en ejecución expulsará los hilos del trabajo planificado.

Por medio de modelos de simulación, Leutenegger y Vemon informaron que, sobre un amplio rango de características de los trabajos, la FCFS es superior a las otras dos políticas. Además, encontraron que algunas formas de planificación por grupos, tratadas en el siguiente apartado, son, en general, superiores a la compartición de carga.

Algunas desventajas de la compartición de carga son:

• La cola central ocupa una región de memoria a la que se debe acceder con exclusión mutua. Así pues, si muchos procesadores buscan trabajo al mismo tiempo, puede convertirse en un cuello de botella. Cuando hay un número pequeño de procesadores, es poco probable que éste sea un problema evidente. Sin embargo, cuando el multiprocesador está formado por docenas o incluso cientos de procesadores, la posibilidad de un cuello de botella es real.

• Es improbable que los hilos expulsados reanuden su ejecución en el mismo procesador. Si cada procesador dispone de una caché local, el uso de la caché será menos eficiente.

• Si todos los hilos son tratados como una reserva común de hilos, es improbable que todos los hilos de un programa consigan acceder a los procesadores al mismo tiempo. Si se exige un alto grado de coordinación entre los hilos de un programa, los intercambios de procesos pueden comprometer seriamente el rendimiento.

A pesar de las desventajas potenciales de la autoplanificación, éste es uno de los esquemas usados más habitualmente en los multiprocesadores actuales.

El sistema operativo Mach [BLAC90] emplea una variante de la técnica de compartición de carga. El sistema operativo mantiene una cola de ejecución local para cada procesador y una cola de ejecución global compartida. La cola de ejecución local la emplean los hilos que han sido asociados temporalmente a un procesador específico. Cada procesador examina en primer lugar la cola de ejecución local, para dar preferencia absoluta a los hilos asociados sobre los no asociados. Como ejemplo del uso de los hilos asociados, uno o más procesado-

res podrían dedicarse a ejecutar procesos que forman parte del sistema operativo. Otro ejemplo: Los hilos de una aplicación particular podrían distribuirse entre una serie de procesadores; con el software apropiado, esta estrategia da soporte a la planificación por grupos, que se tratará a continuación.

Planificación por grupos

El concepto de planificar un conjunto de procesos simultáneamente en un conjunto de procesadores es anterior al concepto de hilo. [JONE80] se refiere a dicho concepto como *planificación por grupos* y cita las siguientes ventajas:

- Si procesos relativamente próximos se ejecutan en paralelo, pueden reducirse los bloqueos por sincronización, pueden hacer falta menos intercambios de procesos y se incrementará el rendimiento.
- La sobrecarga de planificación puede reducirse debido a que una sola decisión afecta a varios procesadores y procesos al mismo tiempo.

En el multiprocesador Cm*, se emplea el término *coplanificación* [GEHR87]. La coplanificación se basa en el concepto de planificación de un conjunto de tareas afines, llamado *cuerpo de tareas.* Los elementos individuales de un *cuerpo de tareas* tienden a ser pequeños y son, por tanto, parecidos a la idea de hilo.

El término *planificación por pandillas* se ha aplicado a la planificación simultánea de hilos que forman parte de un único proceso [FEIT90b]. Es necesaria para aplicaciones paralelas de grano medio a fino cuyo rendimiento se degrada seriamente cuando alguna parte de la aplicación no está ejecutando mientras otras partes están listas para ejecutar. Es también beneficiosa para cualquier aplicación paralela, incluso aquellas que no son lo bastante sensibles al rendimiento. La necesidad de planificación por pandillas está muy reconocida y existen implementaciones en una gran variedad de sistemas operativos de multiprocesadores.

Una razón obvia por la que la planificación por grupos mejora el rendimiento de una sola aplicación es que se minimizan tos intercambios de procesos. Supóngase que un hilo de un proceso está ejecutando y alcanza un punto en el cual debe sincronizarse con otro hilo del mismo proceso. Si ese otro hilo no está ejecutando pero está listo para ejecutar, el primer hilo se queda colgado hasta que se puede realizar un intercambio en otro procesador para traer el hilo que se necesita. En una aplicación con una coordinación estrecha entre sus hilos, tales intercambios reducen drásticamente el rendimiento. La planificación simultánea de los hilos cooperantes puede, además, ahorrar tiempo en el reparto de recursos. Por ejemplo, varios hilos planificados por grupos pueden acceder a un archivo sin el coste adicional del bloqueo durante la operación de Búsqueda, Lectura o Escritura.

El empleo de planificación por grupos origina un requisito de asignación del procesador. Una posibilidad es la siguiente. Supóngase que se tienen N procesadores y M aplicaciones, cada una de las cuales tiene N hilos o menos. Entonces, si se fracciona el tiempo, cada aplicación podría tener $1/M$ del tiempo disponible de los N procesadores. [FEIT90a] indica que esta estrategia puede ser ineficiente. Considérese un ejemplo en el que hay dos aplicaciones, una con cuatro hilos y otra con un hilo. Mediante reparto uniforme del tiempo se derrocha el 37,5% de los recursos de procesamiento, debido a que, cuando la aplicación de un sólo hilo ejecuta, hay tres procesadores ociosos (ver figura 9.2). Si hay varias aplicaciones de un único hilo, éstas podrían unirse para aumentar la utilización del procesador. Si esta opción

no está disponible, una alternativa a la planificación uniforme es la planificación ponderada por el número de hilos. Así pues, la aplicación de cuatro hilos podría obtener cuatro quintas partes del tiempo y la de un hilo obtener sólo una quinta parte del tiempo, reduciendo el malgasto de procesador al 15%.

Asignación dedicada de procesadores

Una forma extrema de planificación por grupos, propuesta en [TUCK89], consiste en dedicar un grupo de procesadores a una aplicación, mientras dure la aplicación. Es decir, cuando se planifica una aplicación, se asigna cada uno de sus hilos a un procesador que permanece dedicado a ese hilo hasta que la aplicación termine su ejecución.

Este método podría parecer demasiado dispendioso en tiempo de procesador. Si un hilo de una aplicación se bloquea en espera de una E/S o por sincronización con otro hilo, el procesador de dicho hilo quedará desocupado, pues no hay multiprogramación de procesadores. Se pueden plantear varias observaciones en defensa de esta estrategia:

1. En un procesador masivamente paralelo, con decenas o cientos de procesadores, cada uno de los cuales representa una pequeña parte del coste del sistema, la utilización del procesador no es tan importante como medida de la efectividad o el rendimiento.
2. La anulación total del intercambio de procesos durante el tiempo de vida de un programa dará como resultado una aceleración sustancial del programa

Tanto los análisis de [TUCK89] como de [ZAH090] dan soporte a la segunda afirmación. La figura 9.3 muestra los resultados de un experimento [TUCK89]. Zahorjan y McCann ejecutaron dos aplicaciones, una multiplicación de matrices y un cálculo de la transformada rápida de Fourier (FFT, *Fast Fourier Transform),* en un sistema con 16 procesadores. Cada aplicación dividía el problema en una serie de tareas, que se organizaban en hilos que ejecutaban la aplicación. Los programas se escribieron de forma que se permitiera variar el número de hilos empleados. Básicamente, cada aplicación define una serie de tareas y las pone en cola. Se toman las tareas de la cola y se hacen corresponder con los hilos disponibles. Si hay menos hilos que tareas, entonces las tareas restantes permanecen en la cola y son recogidas por hilos que hayan completado su tarea asignada. Evidentemente, no todas las aplicaciones se pueden estructurar de esta forma, pero muchos problemas numéricos y algunas otras aplicaciones pueden resolverse de este modo.

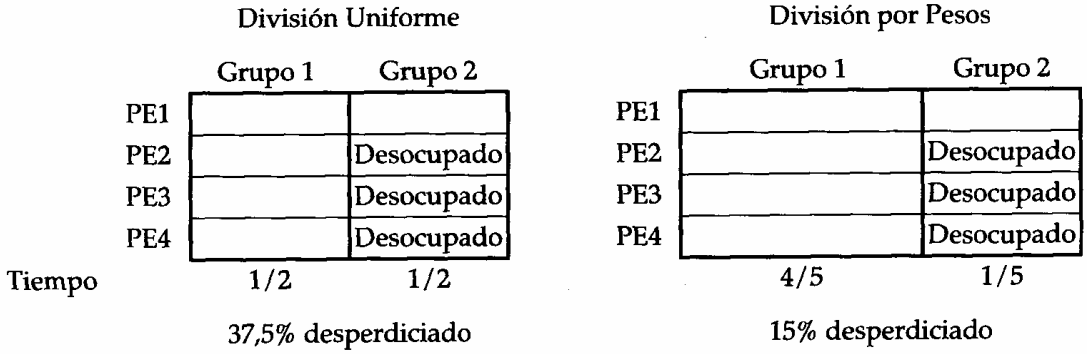


FIGURA 9.2 Ejemplo de planificación por grupos con uno y cuatro hilos [FEIT90a]

La figura 9.3 muestra la aceleración de una aplicación a medida que el número de hilos que están ejecutando las tareas de cada aplicación varía desde uno hasta 24. Por ejemplo, se ve que cuando se da inicio a ambas aplicaciones simultáneamente con 24 hilos cada una, la aceleración que se obtiene, tomando como base el uso de un único hilo por aplicación, es de 2,8 para la multiplicación de matrices y 2,4 para la FFT. La figura muestra que el rendimiento de ambas aplicaciones empeora considerablemente cuando el número de hilos en cada aplicación excede de 8 y, por tanto, el número total de procesos en el sistema supera al de procesadores. Es más, cuanto mayor es el número de hilos, peor rendimiento se obtiene, debido a que aumenta la frecuencia de expulsión de hilos y de replanificación. Esta expulsión excesiva resulta ineficiente en muchos sentidos, incluyendo el tiempo consumido esperando a que un hilo suspendido abandone una sección crítica, el tiempo malgastado en intercambio de procesos y el comportamiento ineficiente de la caché.

[ZAH090] termina diciendo que una estrategia eficaz consiste en limitar el número de hilos activos al número de procesadores del sistema. Si la mayoría de las aplicaciones tienen un único hilo o pueden usar la estructura de cola de tareas, se obtendrá un aprovechamiento de los recursos del procesador efectivo y razonablemente eficiente.

Tanto la asignación dedicada de procesadores como la planificación por grupos abordan el problema de la planificación haciendo frente a la asignación del procesador. Se puede observar que el problema de asignación del procesador en un multiprocesador se asemeja más al problema de asignación de memoria que al problema de planificación en un monoprocesador. La clave está en cómo se asignan muchos procesadores a un programa en un instante dado, que es análogo a cómo se asignan varios marcos de página a un proceso en un instante dado. [GEHR87] propone el concepto de *conjunto de trabajo de actividades,* análogo al conjunto de trabajo en memoria virtual, como el número mínimo de actividades (hilos) que deben ser planificadas simultáneamente en los procesadores para que la aplicación tenga un progreso aceptable. Como en los esquemas de gestión de memoria, el fallo al planificar todos los elementos de un conjunto de trabajo de actividades puede conducir a una *hiperpla-*

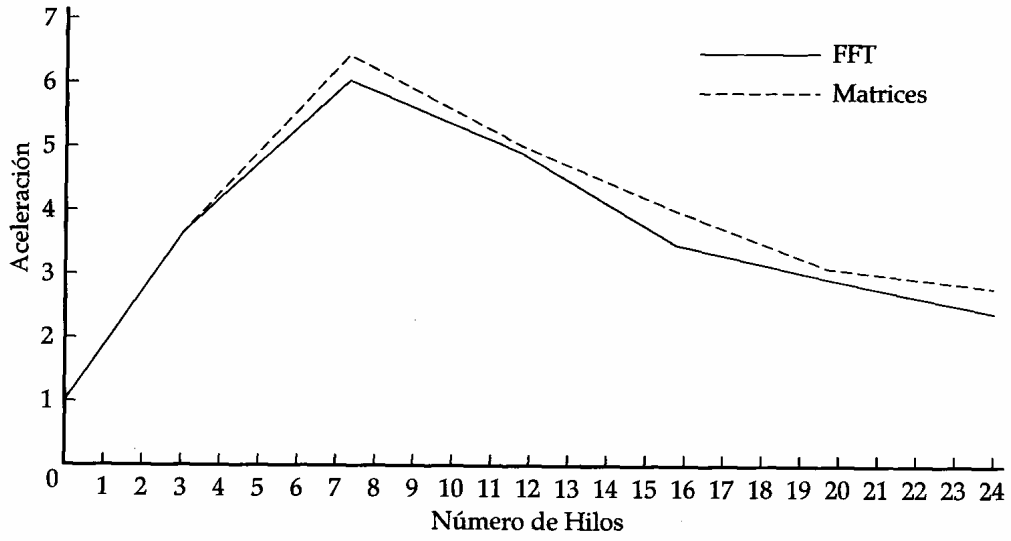


FIGURA 9.3 Aceleración de una aplicación en función del número de procesos [TUCK89]

nificación del procesador (processor thrashing), que se produce cuando la planificación de hilos cuyos servicios son necesarios induce la desplanificación de otros hilos cuyos servicios se necesitarán pronto. De forma similar, *la fragmentación del procesador* se refiere a la situación en la que sobran algunos procesadores mientras otros están asignados y los procesadores sobrantes son insuficientes en número o están mal organizados para dar soporte a los requisitos de las aplicaciones que esperan. La planificación por grupos y la asignación dedicada de procesadores intentan evitar esos problemas.

Planificación dinámica

En algunas aplicaciones, es posible emplear un lenguaje y unas herramientas del sistema que permitan cambiar dinámicamente el número de hilos de un proceso para permitir al sistema operativo ajustar la carga y mejorar así la utilización.

[ZAH090] propone un método en el que tanto el sistema operativo como la aplicación están involucrados en la toma de decisiones de planificación. El sistema operativo es responsable de repartir los procesadores entre los trabajos. Cada trabajo emplea los procesadores de su partición para procesar un subconjunto de sus tareas ejecutables, organizando estas tareas en hilos. A las aplicaciones individuales se les deja (quizá mediante una biblioteca de rutinas de tiempo de ejecución) la decisión sobre el subconjunto a ejecutar, además de a qué hilo suspender cuando se expulsa un proceso. Este método puede no resultar conveniente para todas las aplicaciones. Sin embargo, algunas aplicaciones podrían limitarse a un único hilo, mientras que otras podrían ser programadas para sacar partido de esta característica del sistema operativo.

Con este método, la responsabilidad de planificación del sistema operativo se limita sobre todo a la asignación del procesador y actúa según la siguiente política. Cuando un trabajo solicita uno o más procesadores (tanto si el trabajo llega por primera vez como si cambian sus necesidades):

1. Si hay procesadores desocupados, se usan para satisfacer la petición.
2. En otro caso, si el trabajo que realiza la petición está recién llegado, se le asigna un procesador individual quitándoselo a algún proceso que tenga más de un procesador asignado.
3. Si no se puede satisfacer alguna parte de la petición, queda pendiente hasta que un procesador pase a estar disponible o hasta que el trabajo anule la petición (por ejemplo, si ya no hacen falta más procesadores).

Al liberar uno o más procesadores (incluyendo la salida del trabajo):

4. Explorar la cola de peticiones de procesador no satisfechas. Asignar un solo procesador a cada trabajo de la lista que no tiene aún procesador (es decir, a todas las nuevas llegadas que estén en espera). Después, recorrer la lista de nuevo, asignando el resto de procesadores según un FCFS.

Algunos análisis expuestos en [ZAH090] y [MAJU88] proponen que, para las aplicaciones que pueden sacar partido de la planificación dinámica, este método es superior a la planificación por grupos o la asignación dedicada de procesadores. Sin embargo, el coste de este método puede anular la aparente mejora de rendimiento. Para probar la validez de la planificación dinámica hace falta más experiencia con los sistemas reales.

9.2

PLANIFICACIÓN EN TIEMPO REAL

Antecedentes

En los últimos años, el procesamiento en tiempo real ha pasado a ser una disciplina emergente en informática e ingeniería. El sistema operativo y, en particular, el planificador, es quizá el componente más importante de un sistema de tiempo real. Los ejemplos de aplicaciones de tiempo real actuales incluyen control de experimentos de laboratorio, plantas de control de procesos, robótica, control del tráfico aéreo, telecomunicaciones y sistemas de control y mando militar. Los sistemas de la próxima generación serán los vehículos autónomos todo terreno, controladores de robots articulados, sistemas de fabricación inteligente, estaciones espaciales y exploraciones submarinas.

El procesamiento en tiempo real puede definirse como un tipo de procesamiento en el que la exactitud del sistema no depende sólo del resultado lógico de un cálculo sino también del instante en que se produzca el resultado. Un sistema de tiempo real se puede definir indicando lo que significa proceso de tiempo real o tarea[2]. En general, en un sistema de tiempo real, algunas de las tareas son tareas de tiempo real y, por ello, poseen cierto grado de urgencia. Dichas tareas intentan controlar o reaccionar ante sucesos que tienen lugar en el mundo exterior. Puesto que estos sucesos se producen en "tiempo real", una tarea de tiempo real debe poder ir al paso de los sucesos de los que se ocupa. Así pues, normalmente es posible asociar un plazo a una tarea en particular, donde el plazo especifica el instante de comienzo o el de finalización. Dichas tareas pueden clasificarse en rígidas o flexibles. Una **tarea rígida de tiempo real** debe cumplir el plazo; en otro caso producirá daños no deseados o un error fatal en el sistema. Una **tarea flexible de tiempo real** tiene un plazo asociado, que es conveniente, pero no obligatorio; aunque haya vencido el plazo, aún tiene sentido planificar y completar la tarea.

Otra característica de las tareas de tiempo real es la posibilidad de ser periódicas o aperiódicas. Una **tarea aperiódica** debe comenzar o terminar en un plazo o bien puede tener una restricción tanto para el comienzo como para la finalización. En el caso de una **tarea periódica,** el requisito se puede enunciar como "una vez por cada periodo T" o "exactamente cada *T* unidades".

Características de los sistemas operativos de tiempo real

Los sistemas operativos de tiempo real se pueden caracterizar por presentar requisitos especiales en cinco áreas generales [MORG92]:

[2] Como de costumbre, la terminología plantea un problema ya que palabras diversas se usan en la literatura con significados distintos. Esto es común para un proceso que ocurre en tiempo real y con las limitaciones de una naturaleza repetitiva. Esto es, el proceso dura desde hace mucho tiempo y durante ese tiempo realiza alguna función repetitiva en respuesta a los sucesos que ocurren en tiempo real. No dejaremos, en esta sección, de referirnos a una función individual como una tarea. Así, el proceso puede considerarse como una progresión mediante una sucesión de tareas. En cualquier tiempo determinado, el proceso se compromete en una tarea única, y es el proceso/tarea la que debe programarse.

- Determinismo
- Sensibilidad
- Control del usuario
- Fiabilidad
- Tolerancia a fallos

Un sistema operativo es **determinista** si realiza las operaciones en instantes fijos y predeterminados o en intervalos de tiempo predeterminados. Cuando compiten varios procesos por los recursos y por el tiempo del procesador, ningún sistema será completamente determinista. En un sistema operativo de tiempo real, las solicitudes de servicio de los procesos vienen dictadas por sucesos y temporizaciones externas. El punto hasta el cual un sistema operativo puede satisfacer las peticiones de forma determinista depende, en primer lugar, de la velocidad con la que pueda responder a las interrupciones y, en segundo lugar, de si el sistema posee suficiente capacidad para gestionar todas las peticiones en el tiempo exigido.

Una medida útil de la capacidad de un sistema operativo para operar de forma determinista es el retardo máximo que se produce desde la llegada de una interrupción de alta prioridad de un dispositivo hasta que comienza el servicio. En un sistema operativo que no es de tiempo real, este retardo puede estar en un rango de decenas a cientos de milisegundos, mientras que en un sistema operativo de tiempo real, este retardo puede tener un límite superior que va desde unos pocos microsegundos hasta un milisegundo.

Una característica afín pero diferente es la **sensibilidad.** El determinismo hace referencia a cuánto tiempo tarda un sistema operativo en reconocer una interrupción. La sensibilidad se refiere a cuánto tiempo tarda un sistema operativo en dar servicio a la interrupción, después de reconocerla. Las características de la sensibilidad incluyen:

1. La cantidad de tiempo necesario para iniciar la gestión de la interrupción y comenzar la ejecución de su rutina de tratamiento (ISR, *Interrupt-service Routine).* Si la ejecución exige un cambio de contexto, la espera será mayor que si la ISR puede ejecutarse dentro del contexto del proceso actual.
2. La cantidad de tiempo necesario para ejecutar la ISR. Generalmente, depende de la plataforma de hardware.
3. El efecto del anidamiento de interrupciones. El servicio se retrasará si una ISR puede ser interrumpida por la llegada de otra interrupción.

El determinismo y la sensibilidad forman conjuntamente el tiempo de respuesta a sucesos externos. Los requisitos en tiempo de respuesta son críticos en los sistemas de tiempo real, ya que cada sistema debe cumplir los requisitos de tiempo impuestos por los individuos, dispositivos y flujos de datos externos al sistema.

El **control del usuario** es generalmente mucho mayor en un sistema operativo de tiempo real que en un sistema operativo ordinario. En un sistema operativo típico que no sea de tiempo real, el usuario no tiene control sobre la planificación del sistema operativo o únicamente puede proporcionar directrices a grandes rasgos, tales como agrupar a los usuarios en más de una clase de prioridad. En un sistema de tiempo real, sin embargo, resulta esencial permitir al usuario un control preciso sobre la prioridad de las tareas. El usuario debe poder distinguir entre tareas rígidas y flexibles y especificar prioridades relativas dentro de cada clase. Un sistema de tiempo real también permitirá al usuario especificar características

tales como el uso de paginación o intercambio de procesos, qué procesos deben estar siempre residentes en memoria principal, los algoritmos de transferencia de disco que se emplearán, qué factores disponen a los procesos en varias bandas de prioridad, etc.

La **fiabilidad** es normalmente mucho más importante en sistemas de tiempo real que en los que no lo son. Un fallo transitorio en un sistema que no sea de tiempo real puede resolverse simplemente volviendo a iniciar el sistema. Un fallo de un procesador en un multiprocesador que no sea de tiempo real produce una reducción del nivel de servicio hasta que se repara o sustituye el procesador averiado. Pero un sistema de tiempo real responde y controla sucesos en tiempo real. Las pérdidas o degradaciones del rendimiento pueden tener consecuencias catastróficas, que pueden ir desde pérdidas financieras hasta daños importantes en equipos e incluso pérdidas de vidas.

Como en otros campos, la diferencia entre un sistema operativo de tiempo real y otro que no lo sea es notable. Un sistema de tiempo real debe diseñarse para responder incluso ante varias formas de fallo. La **tolerancia a fallos** es una característica que hace referencia a la capacidad de un sistema de conservar la máxima capacidad y los máximos datos posibles en caso de fallo. Por ejemplo, un sistema UNIX típico, cuando detecta datos corruptos en el núcleo, genera un mensaje de error en la consola del sistema, vuelca el contenido de la memoria a disco para un análisis posterior y finaliza la ejecución del sistema. Por el contrario, un sistema de tiempo real intentará corregir el problema o minimizar sus efectos mientras continúa la ejecución. Normalmente, el sistema notificará a un usuario o a un proceso de usuario que se debe intentar una acción correctora y después continuará la operación con un nivel de servicio posiblemente reducido. En caso de que sea necesario apagar el sistema, se intentara mantener la consistencia de los archivos y los datos.

Un aspecto importante de la tolerancia a fallos es la *estabilidad.* Un sistema de tiempo real es estable si, en los casos en los que es imposible cumplir todos los plazos de ejecución de las tareas, el sistema cumplirá los plazos de las tareas más críticas y de mayor prioridad, incluso si no se cumplen los de alguna tarea menos crítica.

Para cumplir los requisitos anteriores, los sistemas operativos actuales de tiempo real incluyen normalmente las siguientes características:

- Cambios de contexto rápidos
- Pequeño tamaño (con una mínima funcionalidad asociada)
- Capacidad de responder rápidamente a interrupciones externas
- Multitarea con herramientas de comunicación entre procesos, como semáforos, señales y sucesos
- Uso de archivos secuenciales especiales que puedan acumular datos a alta velocidad
- Planificación apropiativa basada en prioridades
- Reducción de los intervalos en los que están inhabilitadas las interrupciones
- Primitivas para demorar tareas durante un tiempo fijo y para detenerlas y reanudarlas
- Alarmas especiales y temporizadores

El corazón de un sistema de tiempo real es el planificador de tareas a corto plazo. En el diseño de tales planificadores no son importantes la equidad y la reducción del tiempo medio de respuesta. Lo que resulta importante es que todas las tareas rígidas de tiempo real acaben (o comiencen) en su plazo y que también acaben (o comiencen) en su plazo tantas tareas flexibles de tiempo real como sea posible.

La mayoría de los sistemas operativos actuales de tiempo real son incapaces de trabajar directamente con plazos. En su lugar, se han diseñado para ser tan sensibles como sea posible a las tareas de tiempo real, de forma que, cuando se aproxime un plazo, se pueda planificar rápidamente la tarea. Desde este punto de vista, las aplicaciones de tiempo real normalmente necesitan tiempos de respuesta deterministas en un rango de varios milisegundos a fracciones de milisegundo, bajo un amplio conjunto de condiciones; las aplicaciones al límite, como los simuladores de aviones militares, por ejemplo, presentan a menudo restricciones en un rango de 10 a 100 microsegundos (μs) [ATLA89].

La figura 9.4 muestra el abanico de posibilidades. En un planificador apropiativo que emplea una planificación simple por tumo rotatorio, una tarea de tiempo real se añadiría a la cola de Listos para esperar su siguiente fracción de tiempo, como se ilustra en la figura 9.4a. El tiempo de planificación normalmente sería inaceptable para aplicaciones de tiempo real. Por otro lado, en un planificador no apropiativo, se puede emplear un mecanismo de planificación por prioridades, dando a las tareas de tiempo real la prioridad más alta. Una tarea de tiempo real que estuviera lista sería planificada tan pronto como el proceso actual se bloquee o termine su ejecución (figura 9.4b). Esto podría acarrear un retraso de varios segundos si una tarea lenta de baja prioridad estaba ejecutando en un momento crítico. Este método tampoco es aceptable. Una idea más prometedora consiste en combinar las prioridades con interrupciones del reloj. Los instantes de apropiación se originan a intervalos regulares. Cuando llegue un instante de apropiación, se expulsará la tarea que esté ejecutando en ese momento si es que una tarea de prioridad superior está esperando, incluyendo la expulsión de tareas que son parte del núcleo del sistema operativo. Estos retardos pueden ser del orden de varios milisegundos (ms) (figura 9.4c). Si bien este último método puede ser adecuado para algunas aplicaciones de tiempo real, no es suficiente para las aplicaciones más exigentes, en cuyo caso, se emplea la denominada *apropiación inmediata.* En la apropiación inmediata, el sistema operativo responde a las interrupciones casi inmediatamente, a menos que el sistema se encuentre ejecutando una sección de código crítico. Los retrasos de planificación de una tarea de tiempo real pueden verse reducidos a 100 μs, o menos.

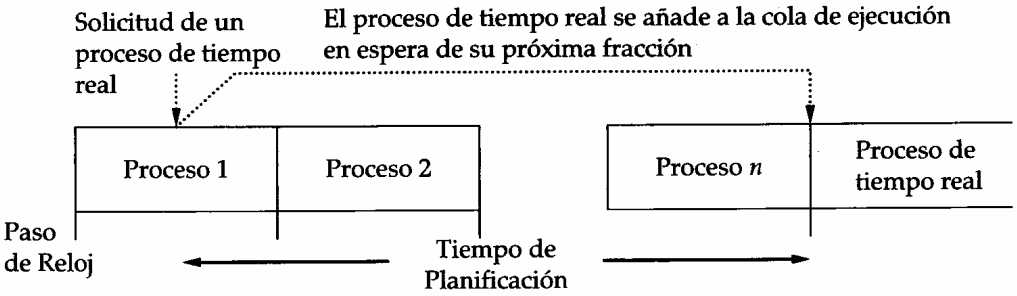
Planificación en tiempo real

La planificación en tiempo real es uno de los campos de investigación más activos en la informática. En este apartado, se ofrecerá una introducción a los distintos métodos de planificación en tiempo real y se estudiarán dos algoritmos de planificación clásicos.

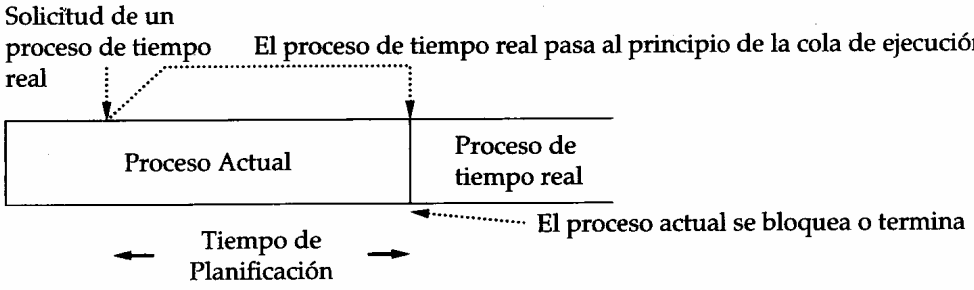
En un estudio de los algoritmos de planificación en tiempo real, [RAMA94] observa que los distintos métodos de planificación dependen de (a) si el sistema lleva a cabo un análisis de planificación; (b) en caso afirmativo, si se realiza estática o dinámicamente; y (c) si el resultado del análisis genera un plan con respecto al cual se expiden las tareas durante la ejecución. Con base en estas consideraciones, [RAMA94] identifica las siguientes clases de algoritmos:

• *Métodos con tablas estáticas:* Realizan un análisis estático de las planificaciones posibles. El resultado del análisis es una plan que determina, durante la ejecución, cuándo debe comenzar la realización de una tarea.

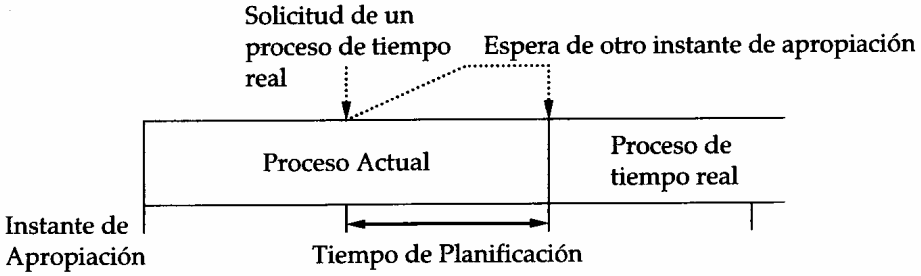
• *Métodos apropiativos con prioridades estáticas:* También se realiza un análisis estático, pero no se traza ningún plan. En cambio, se usa dicho análisis para asignar prioridades a tareas, de forma que se puede emplear un planificador convencional apropiativo con prioridades.



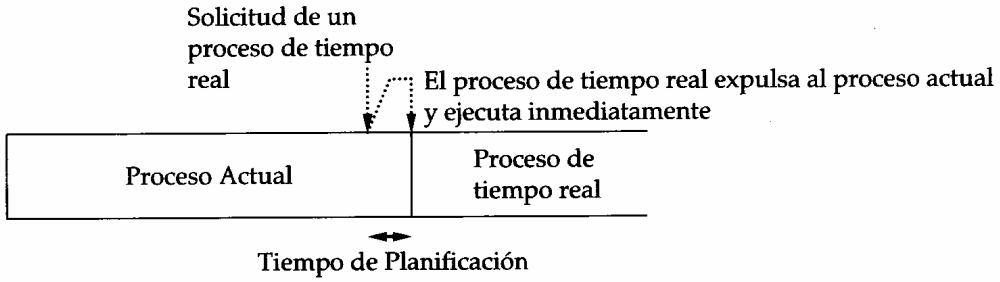
(a) Planificador apropiativo Round-robin



(b) Planificador no apropiativo controlado por prioridades



(c) Planificador apropiativo controlado por prioridades con instantes de apropiación



(d) Planificador con apropiación inmediata

• *Métodos dinámicos de planificación:* Se determina la viabilidad durante la ejecución (dinámicamente) en vez de antes de empezar la ejecución (estáticamente). Se acepta una nueva tarea para ejecutar sólo si es factible cumplir sus restricciones de tiempo. Uno de los resultados del análisis de viabilidad es un plan o proyecto empleado para decidir cuándo se expide cada tarea.

• *Métodos dinámicos del mejor resultado:* No se realiza ningún análisis de viabilidad. El sistema intenta cumplir todos los plazos y abandona cualquier proceso ya iniciado y cuyo plazo no se haya cumplido.

La **planificación con tablas estáticas** es aplicable a tareas periódicas. La entrada del análisis consta del tiempo periódico de llegada, el tiempo de ejecución, el plazo periódico de finalización y la prioridad relativa de cada tarea. El planificador intenta trazar un plan que le permita cumplir las exigencias de todas las tareas periódicas. Este es un método predecible, pero también es inflexible, puesto que cualquier cambio en las exigencias de una tarea requiere que se trace de nuevo el plan. Son típicos de esta categoría los algoritmos de planificación de primero el plazo más próximo u otras técnicas periódicas de plazos (discutidas más adelante).

La **planificación apropiativa con prioridades estáticas** hace uso del mecanismo de planificación apropiativa con prioridades, habitual en la mayoría de los sistemas multiprogramados que no son de tiempo real. En un sistema que no sea de tiempo real, puede emplearse una diversidad de factores para determinar la prioridad. Por ejemplo, en un sistema en tiempo compartido, la prioridad de un proceso cambia dependiendo de si tiene carga de procesador o de E/S. En un sistema de tiempo real, la asignación de prioridades se encuentra relacionada con las restricciones de tiempo asociadas a cada tarea. Un ejemplo de este método es el algoritmo monótono en frecuencia (discutido más adelante), que asigna prioridades estáticas a las tareas en función de sus periodos.

En la **planificación dinámica,** después de que una tarea llega al sistema, pero antes de comenzar a ejecutarla, se intenta crear un plan que contenga las tareas previamente planificadas, así como la recién llegada. Si la recién llegada puede planificarse de forma que se cumplan sus plazos y que no se pase ningún plazo de las tareas ya planificadas, se revisa el plan para hacer sitio a la nueva tarea.

La **planificación dinámica del mejor resultado** es la técnica que se usa en la mayoría de los sistemas de tiempo real comercializados en la actualidad. Cuando llega una tarea, el sistema le asigna una prioridad en función de sus características. Normalmente, se emplea alguna forma de planificación por plazos, como puede ser la de primero el plazo más próximo. En general, las tareas son aperiódicas, por lo que no es posible un análisis estático de planificación. Con este tipo de planificación, no se sabe si se va a cumplir una restricción de tiempo hasta que vence el plazo o la tarea termina. Esta es la mayor desventaja de esta forma de planificación. Su ventaja está en la facilidad de implementación.

Planificación por plazos

La mayoría de los sistemas operativos actuales de tiempo real se diseñaron con el objetivo de dar inicio a las tareas de tiempo real tan rápidamente como sea posible y, por tanto, hacen hincapié en una rápida gestión de interrupciones y expedición de tareas. De hecho, no hay medidas particularmente útiles en la evaluación de los sistemas operativos de tiempo real. En las aplicaciones de tiempo real generalmente no preocupa la velocidad absoluta, sino

completar (o iniciar) las tareas en el momento más apropiado, ni antes ni después, a pesar de las peticiones dinámicas de recursos y los conflictos, la sobrecarga de proceso y los fallos de hardware o software. De esto se deduce que las prioridades son una herramienta burda y que no captan el requisito de "finalización (o inicio) en el momento más apropiado".

En los últimos años, ha habido un buen número de propuestas de métodos más potentes y apropiados para la planificación de tareas de tiempo real. Todos ellos se basan en disponer de información adicional sobre cada tarea. En general, se debería disponer de la siguiente información de cada tarea:

- *Instante en que está cada lista:* El instante en que la tarea pasa a estar lista para ejecución. Fin el caso de una tarea repetitiva o periódica, es en realidad una secuencia de instantes conocidos con anterioridad. En el caso de una tarea aperiódica, este tiempo puede ser conocido con anterioridad o bien el sistema operativo puede tener conocimiento de él cuando la tarea ya se encuentre lista.
- *Plazo de comienzo:* Instante en el que la tarea debe comenzar.
- *Plazo de finalización:* Instante en el que la tarea debe terminar. Las aplicaciones típicas de tiempo real tienen normalmente un plazo de comienzo o un plazo de finalización pero no ambos.
- *Tiempo de proceso:* El tiempo necesitado para ejecutar una tarea hasta su Finalización. En algunos casos, este tiempo es facilitado, pero, en otros, el sistema operativo calcula una media exponencial. En otros sistemas de planificación, no se usa esta información.
- *Exigencias de recursos:* El conjunto de recursos (además del procesador) que necesita una tarea durante su ejecución.
- *Prioridad:* mide la importancia relativa de la tarea. Las tareas rígidas de tiempo real pueden hacer una prioridad "absoluta", produciéndose un fallo del sistema si un plazo se pierde. Si el sistema continúa ejecutando pase lo que pase, tanto las tareas rígidas de tiempo real como las flexibles recibirán una prioridad relativa como guía para el planificador.
- *Estructura de subtareas:* Una tarea puede descomponerse en una subtarea obligatoria y otra subtarea opcional. Solo la subtarea obligatoria tiene un plazo rígido.

Cuando se consideran plazos, hay varios factores a tener en cuenta en la planificación en tiempo real: qué tarea se planifica a continuación y qué tipo de apropiación se permite. Puede demostrarse que para una estrategia de apropiación dada, tanto si se emplean plazos de inicio como de terminación, usar la política de planificación de la tarea con el plazo mas próximo minimiza la proporción de tareas que no cumplen sus plazos [HONG89, PANWK88]. Esta conclusión es válida tanto para configuraciones monoprocesador como multiprocesador.

El otro punto crítico del diseño es la apropiación. Cuando se especifican unos plazos de comienzo, tiene sentido un planificador no apropiativo. En este caso. Sería responsabilidad de la tarea de tiempo real bloquearse a sí misma después de completar la parle obligatoria o crítica de su ejecución, permitiendo satisfacer otros plazos de comienzo de tiempo real.

Esto se ajusta al patrón de la figura 9.4b. Para un sistema con plazos de terminación, es más apropiada una estrategia apropiativa (figuras 9.4c y 9.4d). Por ejemplo, si la tarea X está ejecutando y la tarea *Y* está lista, puede haber circunstancias en que la única forma de per-

mitir que tanto *X* como *Y* cumplan sus plazos de terminación sea expulsando a X, ejecutando Y hasta el final y, después, reanudando *X* hasta su finalización.

Como ejemplo de planificación periódica de tareas con plazos de terminación, considérese un sistema que recoge y procesa datos de dos sensores. A y B. El plazo para tomar los datos del sensor A debe ser cada 20 ms y, del sensor B, cada 50 ms. Se tarda 10 ms, incluyendo la sobrecarga del sistema operativo, para procesar cada muestra de datos de A y 25 ms para procesar cada muestra de B. En resumen:

Proceso	Instante de Llegada	Tiempo de Ejecución	Plazo Final
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
•			
•			
•			
B(1)	25	50	
B(2)	75	100	
•			
•			
•			

El computador es capaz de tomar una decisión de planificación cada 10 ms. Supóngase, bajo estas circunstancias, que se intenta usar un esquema de planificación por prioridades. Los dos primeros diagramas de tiempo de la figura 9.5 muestran el resultado. Si A tiene mayor prioridad, el primer caso de la tarea B recibe sólo 20 ms de tiempo de procesador, en dos fases de 10 ms. tiempo en que vence su plazo y. por tanto, falla. Si B recibe mayor prioridad, entonces A no cumplirá su primer plazo. El diagrama final de tiempos muestra el uso de la planificación por el plazo más próximo. En el instante *t = 0* llegan tanto A1 como B1. Puesto que A1 tiene más próximo el plazo, se planifica en primer lugar. Cuando A1 termina. B1 recibe el procesador. En el instante *t = 20* llega A2. Puesto que A2 tiene el plazo más próximo que B1, se interrumpe B1 para que A2 pueda ejecutar hasta su finalización. A continuación, se reanuda B1 en el instante *t = 30.* En el instante *t = 40,* llega A3. Sin embargo. Bl tiene su plazo final mas próximo y se le permite ejecutar hasta el final, en el instante *t = 45.* A3 recibe el procesador y termina en el instante *t = 55*.

En este ejemplo, pueden cumplirse todos los requisitos del sistema por medio de una planificación que da prioridad, en los instantes de apropiación, a la tarea que tenga el plazo más cercano. Puesto que las tareas son periódicas y predecibles, se usa un método de planificación con tablas estáticas.

Considérese ahora un esquema que trate con tareas aperiódicas con plazos de inicio. La parte superior de la figura 9.6 muestra los instantes de llegada y los plazos de inicio para un ejemplo que consta de cinco tareas, cada una de las cuales con un tiempo de ejecución de 20 ms. Como resumen se puede acudir a la tabla de la pagina 402.

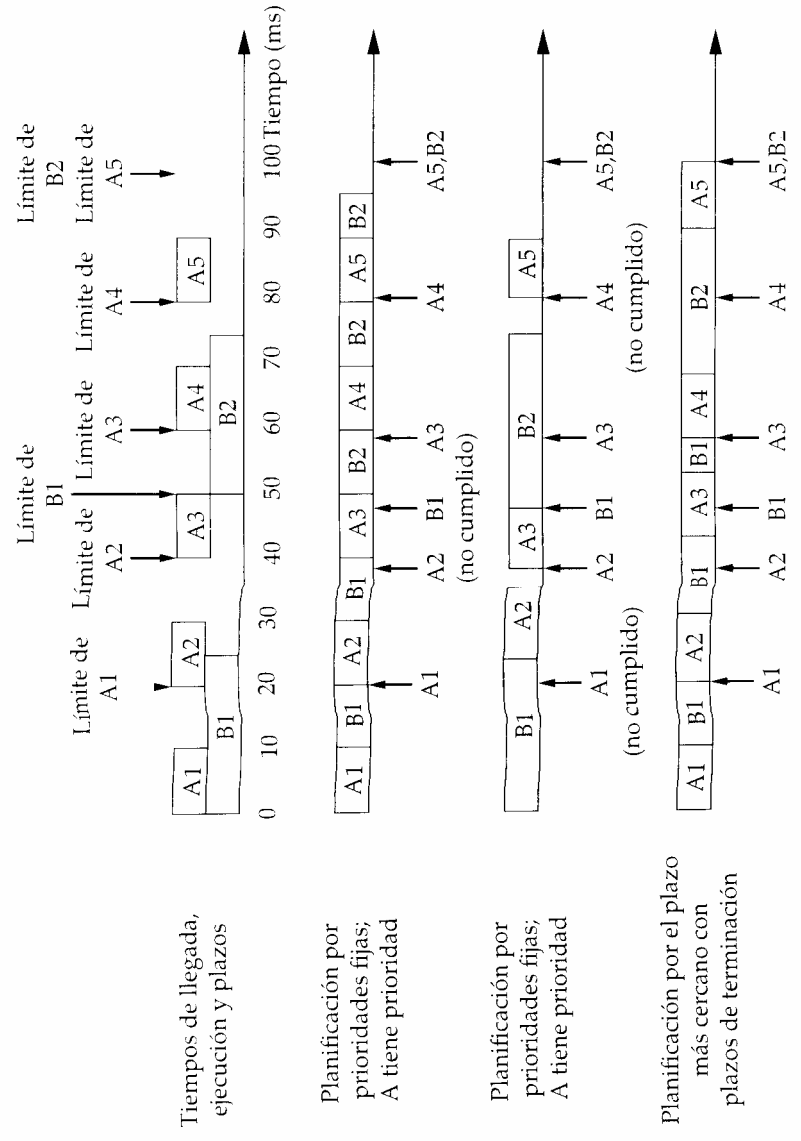


FIGURA 9.5 Planificación de tareas periódicas de tiempo real con plazos de terminación

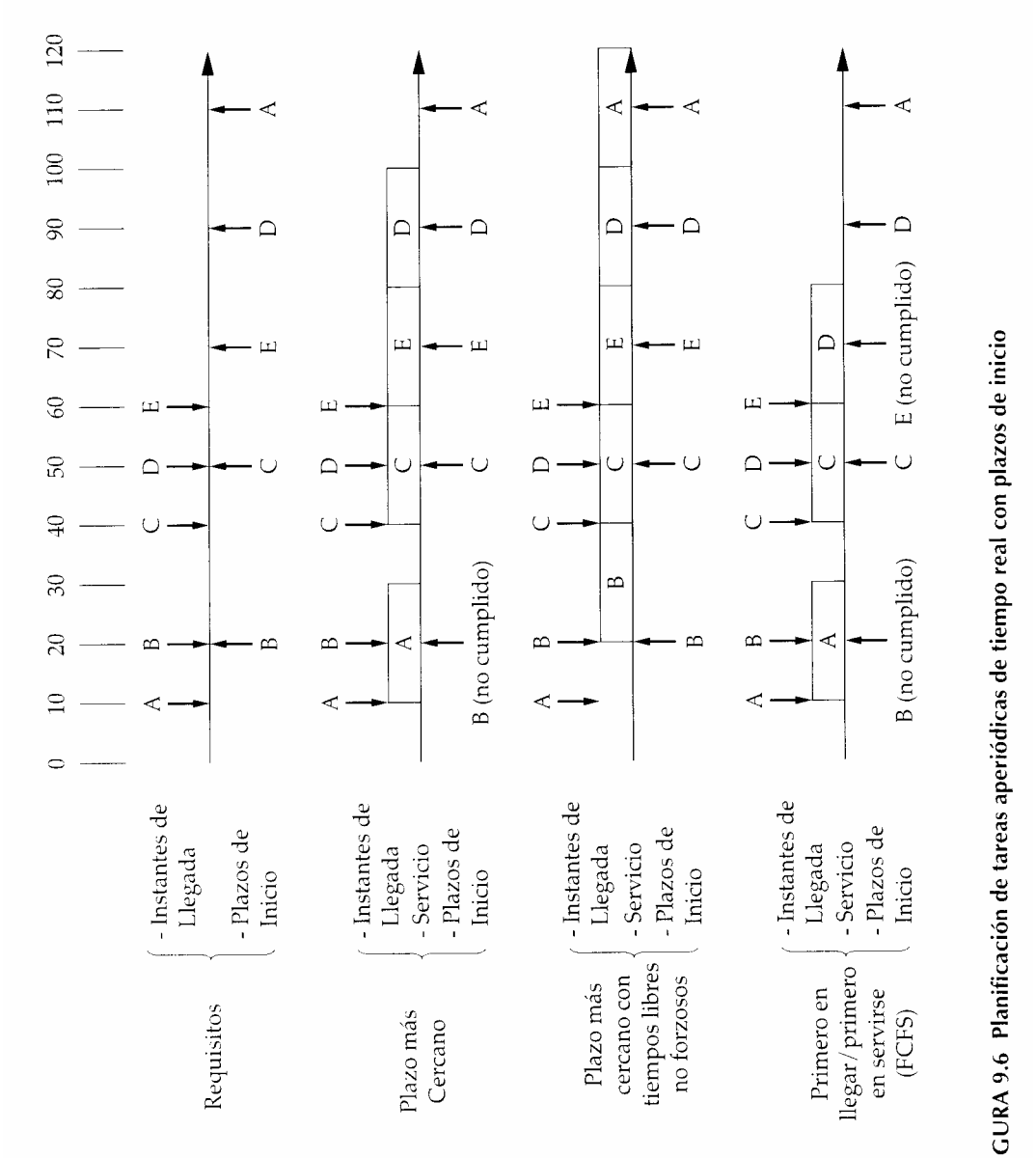


FIGURA 9.6 Planificación de tareas aperiódicas de tiempo real con plazos de inicio

Proceso	Instante de Llegada	Tiempo de Ejecución	Plazo de Inicio
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

Un método sencillo consiste en planificar la tarea lista con el plazo más próximo y dejar que la tarea ejecute hasta el final. Cuando se usa este método en el ejemplo de la figura 9.6, puede observarse que, aunque la tarea B requiere un servicio inmediato, se le niega. Este es el riesgo de tratar con las tareas aperiódicas, especialmente con plazos de inicio. Un refinamiento de la política mejorara el rendimiento si los plazos se conocen antes del instante en que la tarea esté lista. Esta política, conocida como la del plazo más próximo con tiempos libres no forzosos, funciona de la siguiente forma: siempre se planifica la tarea elegible con el plazo más próximo y se deja ejecutar hasta que finalice. Una tarea elegible puede no estar lista y esto puede originar que el procesador quede libre aunque haya tareas listas. En el ejemplo, el sistema se abstiene de planificar la tarea A aunque sea la única tarea lista. El resultado es que, aunque el procesador no se utiliza con la máxima eficiencia, se cumplen lodos los requisitos de planificación. Por último y como comparación se muestra la política FCFS. En este caso, las tareas B y E no cumplen sus plazos.

Planificación monótona en frecuencia

Uno de los métodos más prometedores de resolución de conflictos de la planificación multitarea con tareas periódicas es la planificación monótona en frecuencia **(RMS,** *Rate Monotonic Schedulin).* El esquema fue propuesto por primera vez, en [LIU73], pero ha ganado popularidad recientemente. RMS asigna prioridades a las tareas en función de sus periodos.

La figura 9.7 muestra los parámetros relevantes de las tareas periódicas. El periodo *T* de las tareas es el tiempo que transcurre entre una llegada de la tarea y la siguiente llegada de la misma tarea. La frecuencia de una tarea (en Hertzios) es, simplemente, la inversa de su periodo (en segundos). Por ejemplo, una tarea con un periodo de 50 ms tiene una frecuencia de 20 Hz. Normalmente, el final del periodo de una tarea es también el plazo rígido de la tarea, aunque algunas tareas pueden tener plazos anteriores. El tiempo *C* de ejecución (o computación) es el tiempo de procesamiento de cada acontecimiento de una tarea. Debe quedar claro que, en un sistema monoprocesador, el tiempo de ejecución no debe ser mayor que el periodo ($T \leq C$). Si una tarea periódica ejecuta siempre hasta el final, es decir, si nunca se niega el servicio a la tarea por insuficiencia de recursos, la utilización del procesador por parte de la tarea es $U = C/T$. Por ejemplo, si una tarea tiene un periodo de 80 ms y un tiempo de ejecución de 55 ms. su utilización del procesador será de 55/80 = 0,6X75.

En RMS, la tarea de más alta prioridad es la del periodo más corto, la segunda tarea de mayor prioridad es la del segundo periodo más corto y así sucesivamente. Cuando más de una tarea se encuentran disponibles para ejecutar, se da servicio primero a la que tenga el periodo más corto. Si se dibuja la prioridad de las tareas en función de sus frecuencias, el resultado es una función monótona creciente (figura 9.8); de aquí viene el nombre de planificación monótona en frecuencia.

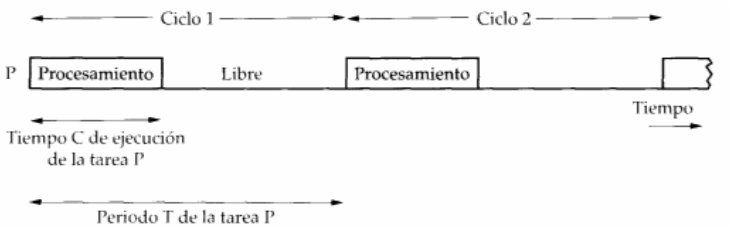


FIGURA 9.7 Diagrama de tiempos de tareas periódicas

Una medida de la efectividad de un algoritmo de planificación periódico está en ver si garantiza o no que se cumplen todos los plazos rígidos de las tareas. Supóngase que se dispone de *n* tareas, cada una de las cuales tiene un periodo y un tiempo de ejecución fijos. Entonces, para que sea posible cumplir todos los plazos, debe cumplirse que:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq 1 \qquad (9.1)$$

La suma de las utilizaciones del procesador por parte de las tareas individuales no puede exceder un valor de 1, que corresponde a la utilización total del procesador. La ecuación (9.1) proporciona un límite para el número de tareas que puede planificar con éxito un algoritmo de planificación perfecto. Para un algoritmo en particular, el límite puede ser inferior. Para el RMS, se puede demostrar que se cumple la siguiente desigualdad:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq n(2^{1/n} - 1) \qquad (9.2)$$

La tabla 9.2 ofrece algunos valores para este límite superior. A medida que aumenta el número de tareas, el límite de planificación converge a ln2 ≈ 0,693. Como ejemplo, considérese el caso de tres tareas periódicas, donde $U_i = C_i / T_i$:

- Tarea P_1: $C_1 = 20$; $T_1 = 100$; $U_1 = 0{,}2$
- Tarea P_2: $C_1 = 40$; $T_1 = 150$; $U_1 = 0{,}267$
- Tarea P_3: $C_1 = 100$; $T_1 = 350$; $U_1 = 0{,}286$

La utilización total de estas tres tareas es 0,2 + 0,267 + 0,268 = 0,753. El límite superior de la planificación de estas tres tareas mediante un RMS es:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} < 3(2^{1/3} - 1) = 0{,}779 \qquad (9.3)$$

Como la utilización total requerida por las tres tareas es menor que el límite superior del RMS (0,753 < 0,779), se sabe que, si se emplea un RMS, se planificarán con éxito todas las tareas.

También puede demostrarse que el límite superior de la ecuación (9.1) se cumple para la planificación por plazos más cercanos. Así pues, es posible alcanzar un aprovechamiento to-

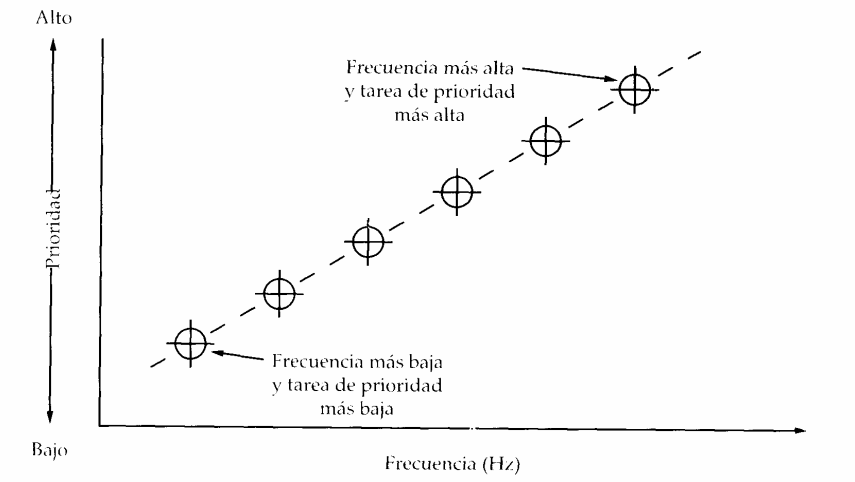


FIGURA 9.8 Una tarea con RMS [WARR91]

tal del procesador mayor y, además, tener espacio para más tareas periódicas planificadas según el plazo más cercano. No obstante, el RMS ha sido muy utilizado en aplicaciones industriales. |SHA91] ofrece la siguiente justificación:

1. La diferencia de rendimiento en la práctica es pequeña. El límite superior de la ecuación (9.2) es conservador y, en la práctica, se consiguen a menudo utilizaciones del *90%.*

2. La mayoría de los sistemas rígidos de tiempo real tienen también componentes flexibles, tal como ciertas visualizaciones no críticas y autochequeos incorporados que pueden ejecutar con niveles de prioridad menores, para absorber el tiempo del procesador no empleado por el RMS en las tareas rígidas de tiempo real.

TABLA 9.2 Valor del Límite Superior RMS

n	$n(2^{1/n} - 1)$
1	1,0
2	0,828
3	0,779
4	0,756
5	0,743
6	0,734
•	•
•	•
•	•
∞	$\ln 2 \approx 0{,}693$

3. La estabilidad se consigue fácilmente con RMS. Cuando un sistema no puede cumplir todos los plazos, debido a la sobrecarga o a errores transitorios, es necesario poder garantizar que se cumplirán los plazos de las tareas fundamentales, dado por supuesto que este subconjunto de tareas se puede planificar. En el método de asignación estática de prioridades, basta con asegurar que las tareas fundamentales reciben prioridades relativamente altas. Esto se hace en el RMS estructurando las tareas fundamentales para que tengan periodos cortos o modificando las prioridades de RMS para que tengan en cuenta a dichas tareas. Con la planificación por plazos más cercanos, la prioridad de una tarea periódica cambia de un periodo a otro. Esto hará más difícil asegurar que las tareas cumplan sus plazos.

9.3 SISTEMAS DE EJEMPLO

Sistema UN IX, **versión V**

El planificador de UNIX emplea realimentación multinivel con turno rotatorio en cada una de las colas de prioridad. El sistema lleva a cabo la apropiación cada segundo. Es decir, si un proceso en ejecución no se bloquea ni termina en 1 seg, será expulsado. La prioridad está basada en el tipo de proceso y en el historial de ejecución. Se aplican las siguientes fórmulas:

$$P_j(i) = \text{Base}_j + \frac{CPU_j(i-1)}{2} + \text{nice}_j$$

$$CPU_j(i) = \frac{U_j(i)}{2} + \frac{CPU_j(i-1)}{2}$$

donde

$P_j(i)$ = Prioridad del proceso j al comienzo del intervalo i; un número menor indica mayor prioridad

Base_j = Prioridad de base del proceso j

$U_j(i)$ = Utilización de CPU del proceso j en el intervalo i

$CPU_j(i)$ = Media ponderada exponencial de la utilización de CPU del proceso j en el intervalo i

nice_j = Factor de ajuste controlado por el usuario

La prioridad de cada proceso se recalcula una vez cada segundo, instante en que se toma una nueva decisión de planificación. El propósito de la prioridad de base es dividir todos los procesos en bandas fijas de niveles de prioridad. Los componentes *nice* y CPU están limitados a impedir que un proceso salga fuera de la banda asignada (según el nivel de prioridad de base). Estas bandas se usan para optimizar el acceso a los dispositivos de bloques (por

ejemplo, los discos) y para permitir al sistema operativo responder rápidamente a las llamadas al sistema. Las bandas son, en orden de prioridad decreciente:

- Intercambio a disco
- Control de dispositivos de E/S de bloques
- Gestión de archivos
- Control de dispositivos de E/S de caracteres
- Procesos de usuario

Esta jerarquía debe proporcionar el aprovechamiento más eficiente de los dispositivos de E/S. En la banda de procesos de usuario, el empleo del historial de ejecución tiende a penalizar los procesos con carga de CPU a costa de los procesos con carga de E/S. Esto también

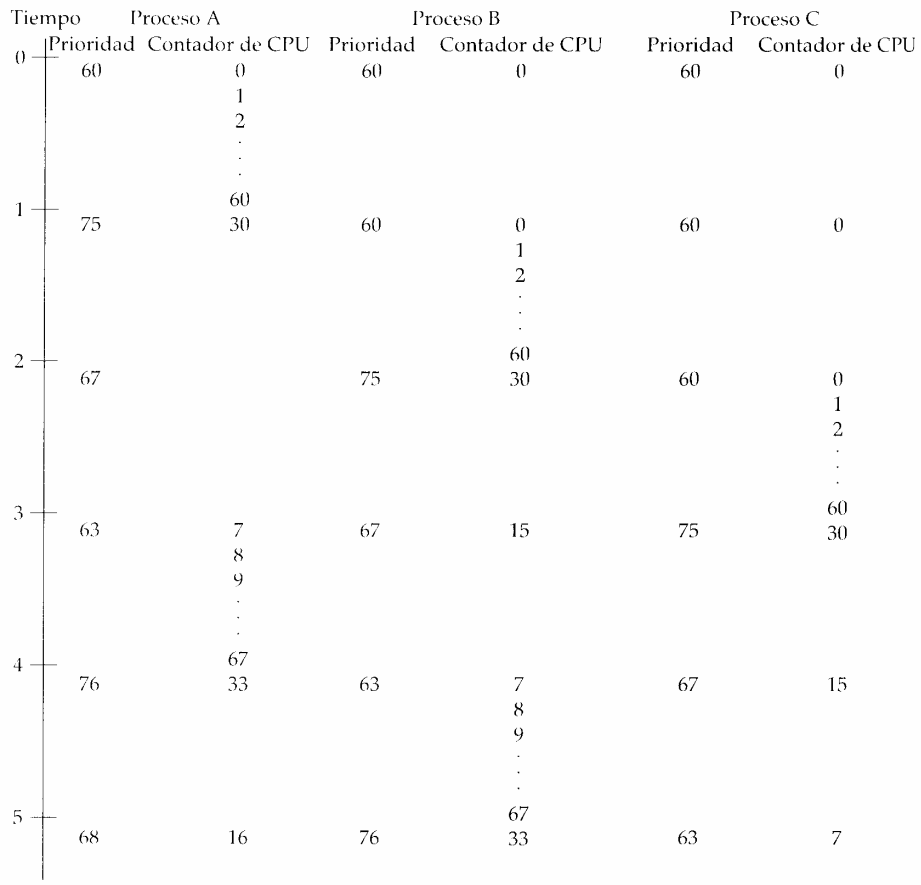


FIGURA 9.9 Ejemplo de planificación de procesos en UN1X [BACH86]

mejorará la eficiencia. Conjuntamente con el esquema apropiativo por turno rotatorio, la estrategia de planificación está bien preparada para satisfacer los requisitos de tiempo compartido de propósito general.

En la figura 9.9 se muestra un ejemplo de planificación de procesos. Los procesos A, B y C se crean en el mismo instante y tienen unas prioridades base de 60 (se ignorará el valor *nice*). El reloj interrumpe al sistema 60 veces por segundo e incrementa un contador del proceso que esté en ejecución. El ejemplo supone que ninguno de los procesos se bloquea a sí mismo y que no hay más procesos listos para ejecutar. Compárese con la figura 8.17.

Windows NT

Los objetivos de planificación de Windows NT son diferentes de los de UNIX. UNIX está interesado en dar un servicio equitativo y sensible a una comunidad de usuarios que comparten el sistema. Windows NT fue diseñado para ser tan sensible como sea posible a las necesidades de un único usuario en un entorno muy interactivo o en el papel de un servidor. Como en UNIX, Windows NT implementa un planificador apropiativo con varios niveles de prioridad. En el caso de Windows NT, se aplica un sistema flexible de niveles de prioridad que incluye planificación por turno rotatorio dentro de cada nivel y. para algunos niveles, variación dinámica de prioridades en función de la actividad de sus hilos.

Prioridades de procesos e hilos

Las prioridades en Windows NT se organizan en dos bandas o clases: tiempo real y variable. Cada una de estas bandas consta de 16 niveles de prioridad. Los hilos que requieren atención inmediata están en la clase de tiempo real, que incluye funciones tales como comunicaciones y tareas de tiempo real.

Como NT utiliza un planificador apropiativo con prioridades, los hilos con prioridades de tiempo real tienen precedencia sobre los otros hilos. En un monoprocesador, cuando un hilo cuya prioridad es mayor que la del que ejecuta en ese momento pasa a estar listo, el hilo de menor prioridad es expulsado y se asigna el procesador al de mayor prioridad.

Las prioridades se gestionan de forma algo diferente en las dos clases (figura 9.10). En la clase de prioridad de tiempo real, lodos los hilos tienen una prioridad fija que no cambia nunca. Todos los hilos activos en un nivel de prioridad dado están en una cola de turno rotatorio. En la clase de prioridad variable, la prioridad de un hilo parte de algún valor inicial asignado y puede cambiar, subir o bajar, durante la vida del hilo. De este modo, hay una cola FIFO en cada nivel de prioridad, pero un proceso puede emigrar a una de las otras colas dentro de la clase de prioridad variable. Sin embargo, un hilo de nivel de prioridad 15 no puede promocionarse a nivel 16 o a ningún otro nivel de la clase de tiempo real.

La prioridad inicial de un hilo en la clase de prioridad variable viene determinada por dos valores: la prioridad de base del proceso y la prioridad de base del hilo. Uno de los atributos del objeto proceso es la *prioridad de base del proceso,* que puede tomar valores de 0 a 15. Cada objeto hilo asociado con un objeto proceso tiene un atributo *prioridad de base del hilo,* que indica la prioridad de base del hilo, relativa a la del proceso. La prioridad de base del hilo puede ser igual que la de su proceso o dos niveles por encima o por debajo de la del proceso. Así, por ejemplo, si un proceso tiene una prioridad base igual a 4 y uno de sus hilos tiene una prioridad base igual a -1, la prioridad inicial del hilo será 3.

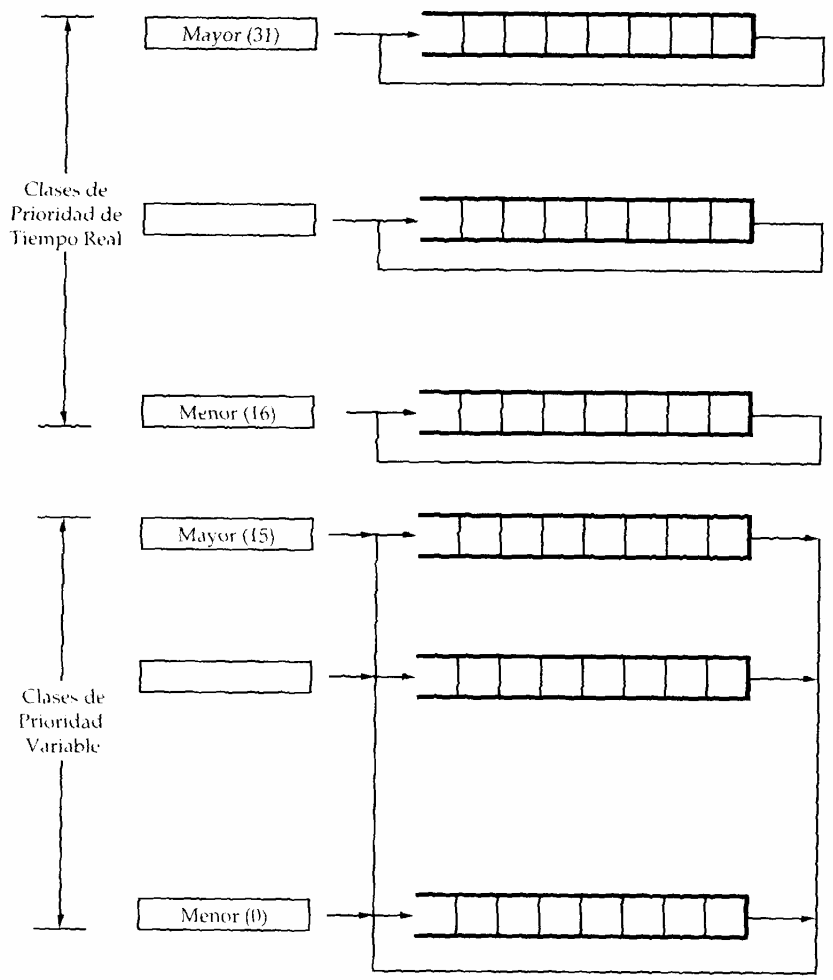


FIGURA 9.10 Prioridades de expedición de hilos en Windows NT

Una vez que ha sido activado un hilo en la clase de prioridad variable, su prioridad actual, llamada prioridad dinámica del hilo, puede fluctuar entre unos límites determinados. La prioridad dinámica nunca puede caer por debajo del valor inferior de prioridad de base del hilo y nunca puede superar 15. La figura 9.11 muestra un ejemplo. El objeto proceso tiene un atributo prioridad de base de 4. Cada objeto hilo asociado con ese objeto proceso debe tener una prioridad inicial entre 2 y 6. La prioridad dinámica para cada hilo puede fluctuar en un rango de 2 a 15. Si se interrumpe un hilo porque ha concluido su cuanto de tiempo actual, el ejecutor de NT disminuye su prioridad. Si se interrumpe un hilo porque espera una E/S, el

ejecutor de NT incrementa su prioridad. Así pues, los hilos con carga de procesador tienen tendencia hacia prioridades inferiores y los hilos con carga de E/S tienen tendencia hacia prioridades más altas. En el caso de los hilos con carga de E/S, el ejecutor eleva la prioridad por esperas interactivas (por ejemplo, esperas por teclado o pantalla) más que por otro tipo de E/S (por ejemplo, E/S a disco). De este modo, los hilos interactivos tienden hacia las prioridades más altas dentro de la clase de prioridad variable.

Planificación del multiprocesador

Cuando NT ejecuta en un único procesador, el hilo de mayor prioridad siempre está activo, a menos que se encuentre esperando un suceso. Si hay más de un hilo con máxima prioridad, el procesador está compartido, por turno rotatorio, entre todos los hilos de ese nivel de prioridad. En un sistema multiprocesador con *N* procesadores, siempre están activos los *(N* — 1) hilos de mayor prioridad, ejecutando de forma exclusiva en uno de los (N — 1) procesadores extra. El resto, los hilos de menor prioridad, comparten el único procesador que queda. Por ejemplo, si hay tres procesadores, los dos hilos de mayor prioridad ejecutan en dos procesadores, mientras que el resto de los hilos ejecutan en el procesador que queda.

La disciplina anterior se ve afectada por un atributo del hilo, la afinidad de procesador. Si un hilo está listo para ejecutar, pero el único procesador disponible no está en su conjunto de procesadores afines, el hilo es obligado a esperar y el ejecutor planifica el siguiente hilo disponible.

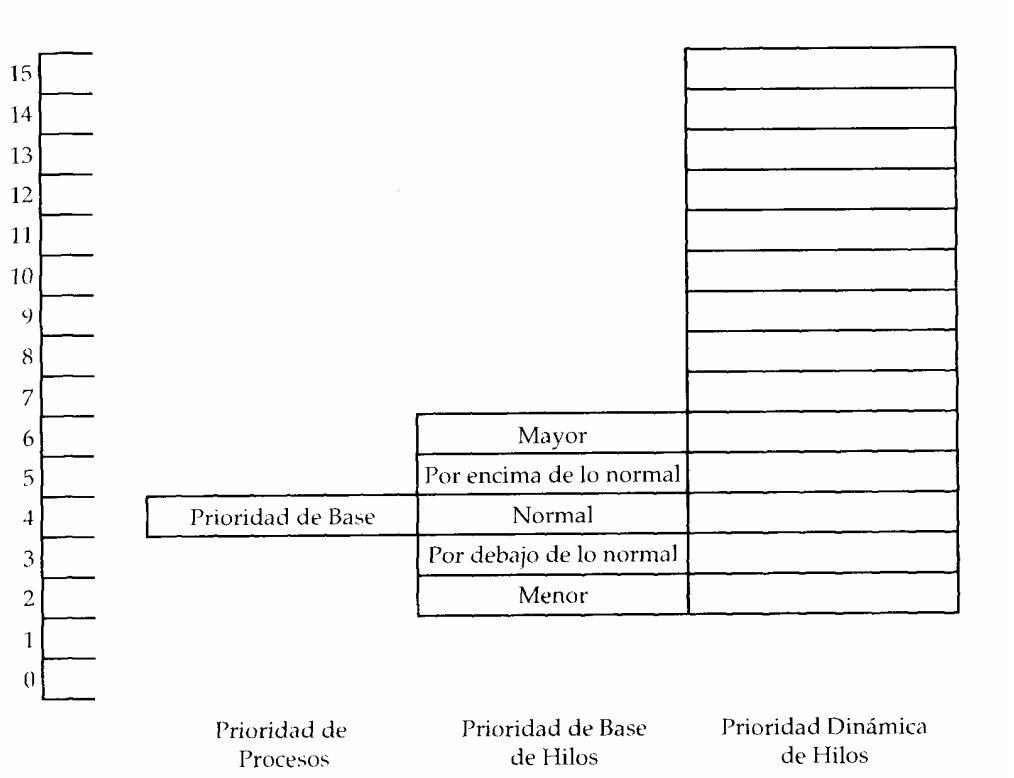


FIGURA 9.11 Ejemplo de relaciones entre prioridades en Windows NT

MVS

Se debe recordar de la sección 3.5 que el entorno de procesos de MVS está formado por las siguientes entidades (ver figura 3.21):

- *Bloque de petición de servicio global (SRB global):* Empleado para gestionar una larca del sistema que no ejecuta en el espacio de direcciones del usuario. Un SRB global representa a una tarea del sistema envuelta en operaciones entre espacios de direcciones.
- *Bloque de control del espacio de direcciones (ASCB) y bloque de extensión del espacio de direcciones (ASXB):* Empleado para gestionar un espacio de direcciones que corresponde aproximadamente a una única aplicación o trabajo. El trabajo dentro de cada espacio de direcciones está formado por un conjunto de tareas del sistema y de larcas de usuario.
- *Bloque de petición de servicio (SRB):* Empleado para gestionar una tarea del sistema que ejecuta dentro de un espacio de direcciones de usuario, que se corresponde con una petición de servicio de una de las tareas del espacio de direcciones.
- *Bloque de control de tarea (TCB):* Representa una de las tareas del espacio de direcciones.

Las tareas representadas por los SRB no son apropiativas; si se interrumpe una tarea, esta recibe el control después de procesar la interrupción. Por el contrario, una tarea controlada por un TCB es apropiativa. Si se interrumpe, cuando se complete la gestión de la interrupción, el control pasará al distribuidor de tareas; el distribuidor puede entonces seleccionar alguna otra tarea para ejecutar.

Los SRB globales se mantienen en una cola de prioridad descendente en el área de colas del sistema, que es una región de memoria principal que no puede ser intercambiada a disco. De este modo, los SRB globales están siempre disponibles en la memoria principal. Cuando hay un procesador disponible, MVS busca primero un SRB listo en esta cola. Si no encuentra ninguno, busca en un espacio de direcciones que tenga al menos un TCB o un SRB listo. Con tal fin, se mantiene una cola de ASCB en el área de colas del sistema. Se asigna un nivel de prioridad completo a cada uno de los espacios de direcciones. Una vez que se ha seleccionado un espacio de direcciones. MVS puede trabajar con las estructuras de este espacio de direcciones, que se conocen como área local de colas del sistema. MVS selecciona liara expedir al SRB de mayor prioridad o, a falta de este, al TCB de mayor prioridad.

Las prioridades de expedición tienen 256 niveles en MVS y los SRB globales se expiden por encima del nivel más alto definido. Dentro de un nivel dado de prioridades de expedición, los espacios de direcciones están en una cola FCFS. Las prioridades de expedición se organizan en conjuntos o bandas, de 16 niveles cada una. Normalmente, un espacio de direcciones que es asignado a una banda permanece en esa banda. En cada banda, los 6 niveles superiores se pueden especificar individualmente como **prioridades fijas,** mientras que los 10 niveles inferiores forman un **grupo de tiempo medio de espera** (MTTW. *mean-time-to-wait*)**.** Dentro del grupo MTTW, las prioridades de los espacios de direcciones se determinan por el tiempo medio entre esperas en cada espacio de direcciones. Aquellas esperas de intervalos más cortos (normalmente para finalización de una E/S) tienen prioridades altas: aquellas con tramos largos de uso de CPU sin intervención de actividades de E/S tienen prioridades menores.

Cuando se crea un espacio de direcciones, es asignado a un **grupo de rendimiento,** que determina la prioridad que disfrutará ese espacio de direcciones. Un espacio de direcciones en un grupo de rendimiento se considera que pasa por una serie de **periodos de rendimiento.** Los periodos de rendimiento permiten a una transacción o a cualquier trabajo, gestionarse de forma.

referente según su edad. La medida de tal envejecimiento es el servicio acumulado y la manera de cambiar los parámetros de gestión es pasar de un periodo de rendimiento al siguiente. Cuando se define un grupo de rendimiento, pueden especificarse uno o más periodos de rendimiento. Cada periodo de rendimiento se caracteriza por una duración y una prioridad. Un espacio de direcciones comienza en el periodo 1 y permanece en el nivel de prioridad especificado para ese periodo hasta que consuma el tiempo de proceso del periodo. Entonces pasa a una prioridad menor en el periodo 2 por la duración de este periodo. La duración de cada periodo de rendimiento se especifica en términos de unidades de servicio. Una unidad de servicio es un intervalo de tiempo de CPU distinto para cada modelo de CPU básico, número de procesadores y entorno de sistema operativo; el rango está entre 1 y 11 ms de tiempo de procesador.

Así pues, la carga de trabajo total se divide, en primer lugar, en grupos de rendimiento, en función de tipos similares (transacciones, por lotes, subsistemas) o espacios de direcciones y, en segundo lugar, acorde a las prioridades establecidas por los distintos componentes de la carga de trabajo. Entonces, en un espacio de direcciones, se realiza una descomposición en función de la duración esperada de las transacciones. Por ejemplo, un espacio de direcciones de tiempo compartido puede definirse como sigue:

- Periodo 1: La prioridad más alta y una duración de 1000 unidades de servicio
- Periodo 2: La siguiente prioridad más alta y una duración de 4000 unidades de servicio
- Periodo 3: La prioridad más baja y duración ilimitada

Con esta estructura, es posible esperar que las órdenes triviales terminen en el periodo I, obteniéndose el mejor servicio con la probabilidad más alta de acceso al sistema. Las órdenes más largas agotarían la cuota del periodo 1 y ejecutan hasta terminar en el periodo 2, dando todavía un servicio razonablemente bueno. Las órdenes de esta categoría serán algo más complejas que las órdenes más cortas y pocas serán ejecutadas concurrentemente. Por último, las órdenes más largas, corno puede ser el lanzamiento de una compilación o de un programa de aplicación, ejecutarán con un servicio aún más bajo.

9.4 RESUMEN

En un multiprocesador fuertemente acoplado, varios procesadores tienen acceso a la misma memoria principal. Con esta configuración, la estructura de planificación es algo más compleja. Por ejemplo, se puede asignar un determinado proceso al mismo procesador durante toda su vida o se puede expedir hacia un procesador distinto cada vez que alcance el estado Ejecutando. Algunos estudios de rendimiento proponen que las diferencias entre los diversos algoritmos de planificación son menos significativas en un sistema multiprocesador.

Un proceso o tarea de tiempo real es aquél que se ejecuta en conexión con algún proceso, función o conjunto de sucesos externos al sistema informático y que debe cumplir uno o más plazos para interactuar de forma correcta y eficiente con el entorno exterior. Un sistema operativo de tiempo real es aquél que gestiona procesos de tiempo real. En este contexto, no son aplicables los criterios tradicionales de selección de algoritmos de planificación. En su lugar, el factor clave está en cumplir los plazos. Son apropiados en este contexto los algoritmos que dependen mucho de la apropiación y de reaccionar a los plazos relativos.

9.5 LECTURAS RECOMENDADAS

[WEND89| es un tratado interesante sobre los métodos de planificación de multiprocesadores.

Las siguientes recopilaciones recientes de artículos contienen trabajos importantes sobre la planificación y los sistemas operativos de tiempo real: [KRIS94], [LEE93], [STAN93] y [T1LB9I]

En [BACH86, [POWE93] y [SAMS90] se pueden hallar buenas descripciones de las estrategias de planificación del sistema UNIX versión V, Windows NT y MVS, respectivamente.

BACH86 BACH, M. *The Design of the UNIX Operating System.* Prentice-Hall. Englewood Cliffs, NJ,1986.

KRIS94 KRISHNA, C. y LEE, Y., eds. "Special Issue on Real-Time Systems". Proccedings of the IEEE, enero de 1994.

LEE93 LEE, Y. y KRISHNA. C., eds. *Readings in Real-Time Systems.* IEEE Computer Society Press, Los Alamitos. CA, 1993.

POWE93 POWELL, J. M*ultitask Windows NT.* Waite Group Press, Corte Madera, CA, 1993.

SAMS90 SAMSON. S. *MVS Performance Management.* McGraw-Hill, Nueva York, 1990.

STAN93 STANKOVIC, J. y RAMAMRITHAM, K., eds. *Advances in Real-Time Systems.* IEEE Computer Society Press, Los Alamitos, CA, 1993.

T1LB91 TILBORG, A. y KOOB, G., eds. *Foundations of Real-Time Computing Scheduling and Resource Management.* Kluwer Academic Publishers, Boston, 1991.

WEND89 WENDORF, J., WENDORF, R. y TOKUDA, H. "Scheduling Operating System Processing on Small-Scale Microprocessors." *Procedings. 2nd Annual Hawaii International Conference on System Science,* enero de 1989.

CAPÍTULO 10

Administración de la Entrada/Salida y planificación de discos

Tal vez el aspecto más contuso en el diseño de los sistemas operativos sea la entrada/salida (E/S). Dada la amplia variedad de dispositivos y sus muchas aplicaciones, resulta difícil construir una solución general y consistente.

Este capítulo comienza con una discusión breve sobre los dispositivos y la organización de las funciones de E/S. Dichos temas, que generalmente caen dentro del campo de estudio de la arquitectura de computadores, constituyen las bases para la observación de la E/S desde el punto de vista de los sistemas operativos.

La sección siguiente examina los aspectos de diseño de los sistemas operativos, incluyendo los objetivos de diseño y la manera en que se pueden estructurar las funciones de E/S. A continuación se examinará un área clave de la E/S, como es el almacenamiento intermedio *(buffering).* Uno de los servicios básicos de E/S provistos por los sistemas operativos es el almacenamiento intermedio, que permite mejorar el rendimiento del sistema en conjunto.

La última parte del capítulo está dedicada a la E/S con discos magnéticos. En los sistemas actuales, esta forma de E/S es la más importante y es la clave del rendimiento que el usuario puede percibir. Se va a comenzar por construir un modelo del rendimiento de la E/S con los discos para, posteriormente, introducir varias técnicas que pueden aplicarse a la mejora del rendimiento.

10.1 DISPOSITIVOS DE ENTRADA/SALIDA

Los dispositivos externos que tienen que hacer E/S con los computadores pueden clasificarse, básicamente, en tres categorías:

- *Dispositivos legibles por los humamos:* apropiados para la comunicación con el usuario. Como ejemplo se tienen los terminales de vídeo, que constan de un teclado, una pantalla y, quizá, otros dispositivos como un ratón o una impresora.

• *Dispositivos legibles por la máquina:* adecuados para comunicarse con equipos electrónicos, como discos, unidades de cinta, sensores, controladores e impulsores.

• *Dispositivos de comunicaciones:* apropiados para comunicarse con dispositivos lejanos. Por ejemplo, adaptadores de líneas digitales y módems.

Existen grandes diferencias entre las clases de dispositivos y éstas son, incluso, sustanciales, dentro de cada clase. Las siguientes son las diferencias principales:

• *Velocidad de los datos:* Puede haber una diferencia de varios órdenes de magnitud en las velocidades de transmisión de datos. La tabla 10.1 ofrece varios ejemplos.

• *Aplicaciones:* La utilidad que se le da a un dispositivo tiene una gran influencia en el software y en las políticas del sistema operativo y de las utilidades de apoyo. Por ejemplo, un disco que almacena archivos necesita el soporte de un software de gestión de archivos. En cambio, un disco usado como almacén de páginas de un sistema de memoria virtual dependerá del uso que se haga del hardware y el software de memoria virtual. Además, estas aplicaciones tendrán su impacto en los algoritmos de planificación del disco (discutidos más adelante en este capítulo). Como ejemplo adicional, un terminal puede valer para un usuario normal o para el administrador del sistema. El uso que se le dé exigirá diferentes niveles de privilegio y, quizá, diferentes prioridades en el sistema operativo.

• *Complejidad del control:* Una impresora necesita una interfaz de control relativamente simple. En cambio, un disco es mucho más complejo. El efecto de estas diferencias en el sistema operativo es filtrado, hasta cierto punto, por la complejidad del módulo de E/S que controla al dispositivo, como se discute en la sección siguiente.

• *Unidad de transferencia:* Los datos pueden transmitirse como flujos de bytes o caracteres (por ejemplo, en un terminal) o en bloques mayores (por ejemplo, con un disco).

Dispositivo	Comportamiento	Interacción	Velocidad de Transmisión
Teclado	Entrada	Humano	0,01
Ratón	Entrada	Humano	0,02
Micrófono	Entrada	Humano	0,02
Escáner	Entrada	Humano	200
Altavoces	Salida	Humano	0,6
Impresora de línea	Salida	Humano	1
Impresora láser	Salida	Humano	1 00
Pantalla gráfica	Salida	Humano	30.000
CPU a buffer	Salida	Humano	200
Terminal de red	Entrada/Salida	Máquina	0,05
Adaptador de LAN	Entrada/Salida	Máquina	200
Disco óptico	Almacenamiento	Máquina	500
Cinta magnética	Almacenamiento	Máquina	2.000
Disco magnético	Almacenamiento	Máquina	2.000

• *Representación de los datos:* En diferentes dispositivos se emplean diferentes esquemas de codificación de datos, incluidas las diferencias en los códigos de caracteres y los convenios de paridad.
• *Condiciones de error:* La naturaleza de los errores, la manera en que se informa sobre ellos, sus consecuencias y el rango disponible de respuestas difieren ampliamente de un dispositivo a otro.

Esta diversidad conduce hacia un enfoque consistente y uniforme de la E/S, que es difícil de alcanzar, tanto desde el punto de vista del sistema operativo como de los procesos de usuario.

10..2

ORGANIZACIÓN DE LAS FUNCIONES DE E/S

La sección 1.7 resumía tres técnicas para realizar la E/S :

• *E/S programada:* El procesador emite una orden de E/S de parte de un proceso a un módulo de E/S; el proceso espera entonces a que termine la operación, antes de seguir.
• *E/S dirigida por interrupciones:* El procesador emite una orden de E/S de parle de un proceso, continúa la ejecución de las instrucciones siguientes y es interrumpido por el módulo de E/S cuando este ha completado su trabajo. Las instrucciones siguientes pueden ser del mismo proceso, si no es necesario para este esperar la terminación de la E/S. En otro caso, el proceso se ve suspendido a la espera de la interrupción, mientras se realiza otro trabajo.
• *Acceso directo u memoria (DMA):* Un módulo de DMA controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador envía una petición de transferencia de un bloque de datos al módulo de DMA y se ve interrumpido sólo cuando el bloque entero se haya transferido.

La tabla 10.2 indica la relación entre estas tres técnicas. En la mayoría de los sistemas informáticos, el DMA es la forma dominante de transferencia ofrecida por el sistema operativo. En esta sección se van a ampliar algunas ideas sobre el uso del DMA.

Evolución de las Funciones de la E/S

A medida que los sistemas informáticos han evolucionado, se ha producido una tendencia creciente en la complejidad y sofisticación de cada componente individual. En ningún caso se hace esto más evidente que en las funciones de la E/S. Las etapas de su evolución pueden resumirse como sigue:

1. El procesador controla directamente los dispositivos periféricos. Esto se puede ver en dispositivos simples controlados por microprocesadores.
2. Se añade un controlador o módulo de E/S. El procesador utiliza E/S programada sin interrupciones. En este punto, el procesador parece aislarse de los detalles específicos de las interfaces con dispositivos externos.
3. Se considera la misma configuración del punto 2, pero empleándose interrupciones. Ahora el procesador no tiene que desperdiciar tiempo esperando a que se realice una operación de E/S, incrementando así la eficiencia.

TABLA 10.2 Técnicas de E/S

	Sin interrupciones	Con interrupciones
Transferencia de E/S a memoria a través del procesador	E/S programada	E/S dirigida por interrupciones
Transferencia de E/S directa a memoria		Acceso directo a memoria (DMA)

4. El módulo de E/S recibe control directo de la memoria, a través de DMA. Ahora puede mover un bloque de datos a la memoria o desde la misma sin que intervenga el procesador, excepto al principio y al final de la transferencia.
5. El módulo de E/S es mejorado para constituir un procesador separado con un conjunto de instrucciones especializado para realizar E/S. El procesador central (CPU) ordena al procesador de E/S la ejecución de los programas de E/S en la memoria principal. El procesador de E/S va en busca de estas instrucciones y las ejecuta si la intervención de la CPU. Esto permite a la CPU precisar que una secuencia de actividades de E/S se vea interrumpida sólo cuando haya terminado la secuencia entera.
6. El módulo de E/S posee su memoria local y es, de hecho, un computador independiente. Con esta arquitectura se pueden controlar un gran número de dispositivos de E/S con una participación mínima de la CPU. Un uso muy común de tal arquitectura ha sido el control de las comunicaciones con terminales interactivos. El procesador de E/S se encarga de la mayoría de las tareas implicadas en el control de los terminales.

A medida que se sigue en esta evolución, una mayor parte de las funciones de E/S se realiza sin la participación de la CPU. El procesador central se ve liberado cada vez más de las tareas relacionadas con la E/S, mejorando así el rendimiento. En las dos últimas etapas (5 y 6) se produce un cambio sustancial con la introducción del concepto de módulo de E/S capaz de ejecutar programas.

Una indicación sobre la terminología: Para todos los módulos descritos en los pasos 4, 5 y 6, el término "acceso directo a memoria" (DMA) es apropiado porque todos contemplan un control directo de la memoria principal por parte del módulo de E/S. Además, el módulo de E/S de la etapa 5 es a menudo denominado **canal de** E/S, mientras que al de la etapa 6 se le llama **procesador de** E/S. Sin embargo, cada término se aplica, en algunos casos, a ambas situaciones. En la parte siguiente de esta sección se empleará el término *canal de E/S* para referirse a ambos tipos de módulos.

Acceso Directo a Memoria

La figura 10.1 muestra, en líneas generales, la lógica del DMA. La unidad de DMA es capaz de imitar a la CPU y, de hecho, es capaz de relevar a la CPU en el control del sistema para transferir los datos con la memoria por el bus del sistema. Normalmente, el módulo de DMA debe usar el bus sólo cuando la CPU no lo necesite, o debe forzar a la CPU a que suspenda temporalmente su operación. Esta última técnica es más común y se denomina *robo de ciclos* porque la unidad de DMA debe robar un ciclo del bus.

La figura 10.2 muestra dónde puede suspenderse a la CPU dentro del ciclo de una instrucción. En cada caso, la CPU se ve suspendida justo antes de que necesite usar el bus.

Entonces, la unidad de DMA transfiere una palabra de memoria y devuelve el control a la CPU. Nótese que esto no es una interrupción; la CPU no tiene que guardar el contexto y hacer otra cosa. Más bien, la CPU espera un ciclo del bus. El efecto final es que la CPU ejecuta más lentamente. Sin embargo, el DMA es mucho más eficiente que la E/S programada o dirigida por interrupciones para una transferencia de varias palabras.

El mecanismo de DMA puede configurarse de muchas formas. Algunas posibilidades se muestran en la figura 10.3. En el primer ejemplo, todos los módulos comparten el mismo bus del sistema. El módulo de DMA, actuando como una CPU suplente, realiza E/S programada para intercambiar datos entre la memoria y el módulo de E/S a través del módulo de DMA. Esta configuración, aunque puede ser barata, es claramente ineficiente. Como con la E/S programada, cada transferencia de una palabra consume dos ciclos del bus.

El número de ciclos de bus requeridos puede ser acortado sustancialmente mediante la integración de las funciones del DMA y de la E/S. Como muestra la figura 10.3b, esto significa que debe haber un camino entre el módulo de DMA y uno o más módulos de E/S que no pasen por el bus del sistema. La lógica del DMA puede formar parte del módulo de E/S, o puede constituir un módulo separado que controle uno o más módulos de E/S. Esta idea puede llevarse un paso más allá si se conectan los módulos de E/S al módulo de DMA mediante un bus de E/S (figura 10.3c). Esto reduce a una el número de

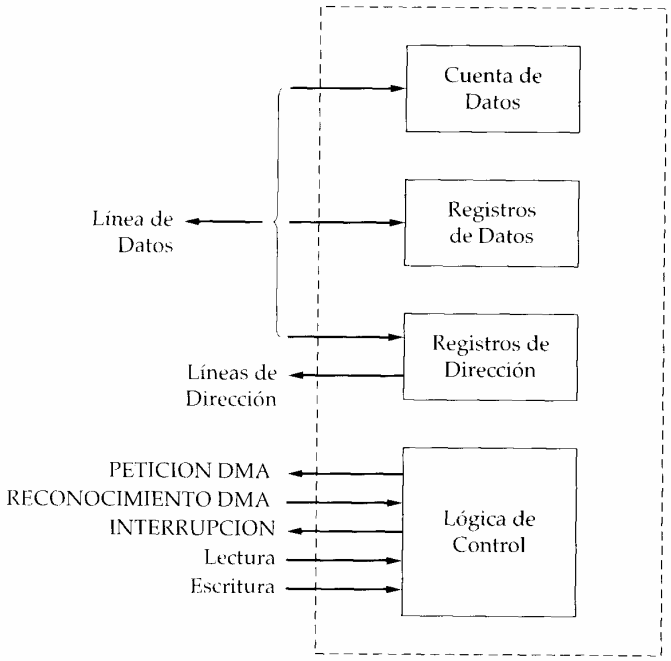


FIGURA 10.1 Diagrama de bloques de un DMA típico

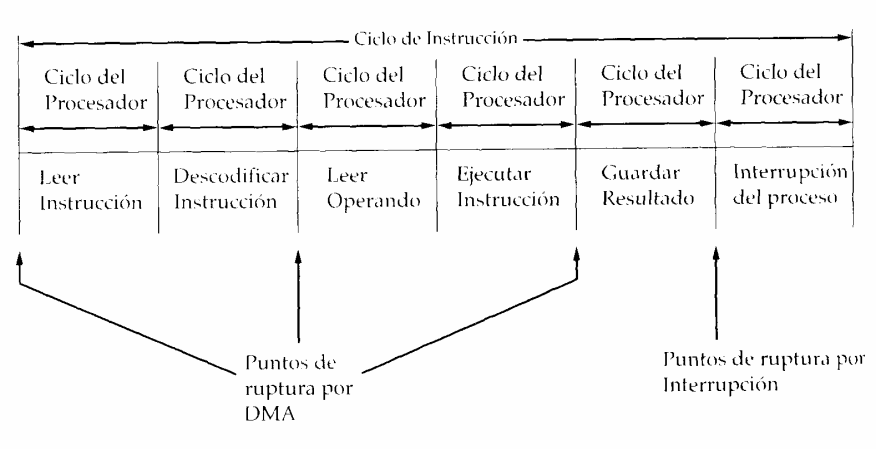


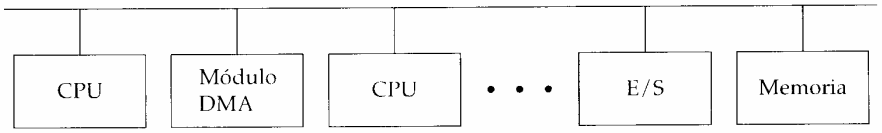
FIGURA 10.2 Puntos de ruptura por DMA e interrupciones en un ciclo de instrucción

interfaces de E/S en el módulo de DMA y proporciona una configuración fácilmente ampliable. En todos los casos (figuras 10.3b y 10.3c), el bus del sistema que el módulo de DMA comparte con la CPU y la memoria principal es utilizado por el módulo de DMA sólo para intercambiar datos con la memoria y para intercambiar señales de control con la CPU. El intercambio de datos entre el módulo de DMA y el de E/S tiene lugar fuera del bus del sistema.

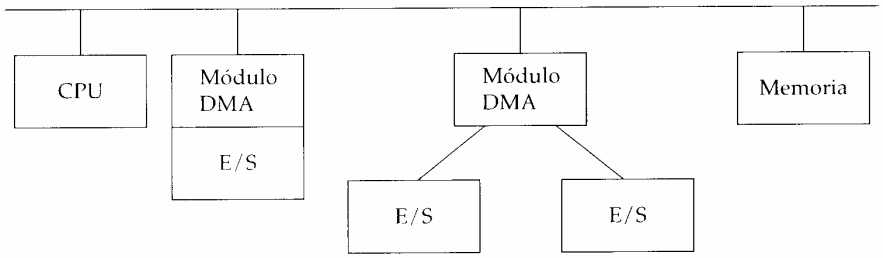
Características de los Canales de E/S

El canal de E/S es una extensión del concepto de DMA. Un canal de E/S tiene la capacidad de ejecutar instrucciones de E/S. lo que le da un control total sobre las operaciones de E/S. En un sistema informático que conste de tales dispositivos, las instrucciones de E/S se almacenan en la memoria principal y serán ejecutadas por un procesador de propósito específico en el mismo canal de E/S. Así, la CPU inicia una transferencia de E/S ordenando al canal de E/S que ejecute un programa en la memoria. El programa designará a el (los) dispositivo(s), la(s) zona(s) de memoria para lectura o escritura, la prioridad y las acciones a tomar bajo ciertas condiciones de error.

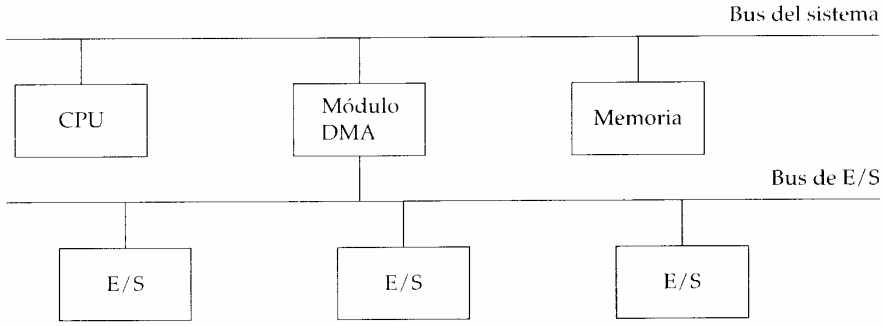
Hay dos tipos comunes de canales de E/S, como se ilustra en la figura 10.4. Un **canal selector** controla varios dispositivos de alta velocidad y se dedica a la transferencia de dalos con estos dispositivos, cada ve/ con uno. De esta forma, el canal de E/S elige un dispositivo y realiza la transferencia. Un **controlador** o módulo de E/S muy parecido a los antes descritos, maneja un dispositivo o un pequeño conjunto de dispositivos. Así, el canal de E/S actúa en ve/, de la CPU manejando estos controladores. Un **canal multiplexor** puede manejar la E/S con varios dispositivos al mismo tiempo. Para dispositivos de baja velocidad, un **multiplexor de bytes** recibe o transmite caracteres de/a varios dispositivos tan rápido como sea posible. Por ejemplo, el flujo de caracteres resultante para tres dispositivos con velocidades diferentes de datos y con flujos individuales $A_1A_2A_3A_4$…., $B_1B_2B_3B_4$..., $C_1C_2C_3C_4$..., podría ser $A_1B_1C_1A_2C_2A_3B_2C_3A_4$, y así sucesivamente. Para dispositivos de alta velocidad, un **multiplexor de bloques** puede mezclar los bloques de dalos de varios dispositivos.



(a) DMA independiente de bus sencillo



(b) Integración DMA-E/S de bus sencillo



(c) Bus de E/S

FIGURA 10.3 Configuraciones posibles de DMA

10.3 ASPECTOS DE DISEÑO EN LOS SISTEMAS OPERATIVOS

Objetivos de Diseño

Hay dos objetivos primordiales en el diseño de la E/S: eficiencia y generalidad. La **eficiencia** es importante porque las operaciones de E/S constituyen a menudo un cuello de botella en los sistemas informáticos. Si se observa de nuevo la tabla 10.1, se puede comprobar que la mayoría de los dispositivos de E/S son extremadamente lentos en comparación con la memoria principal y el procesador. Una manera de abordar este problema es el uso de la multiprogramación, que, como se ha visto, hace que algunos procesos esperen en operaciones de E/S mientras otro proceso se está ejecutando. Sin embargo, a pesar del enorme tamaño de

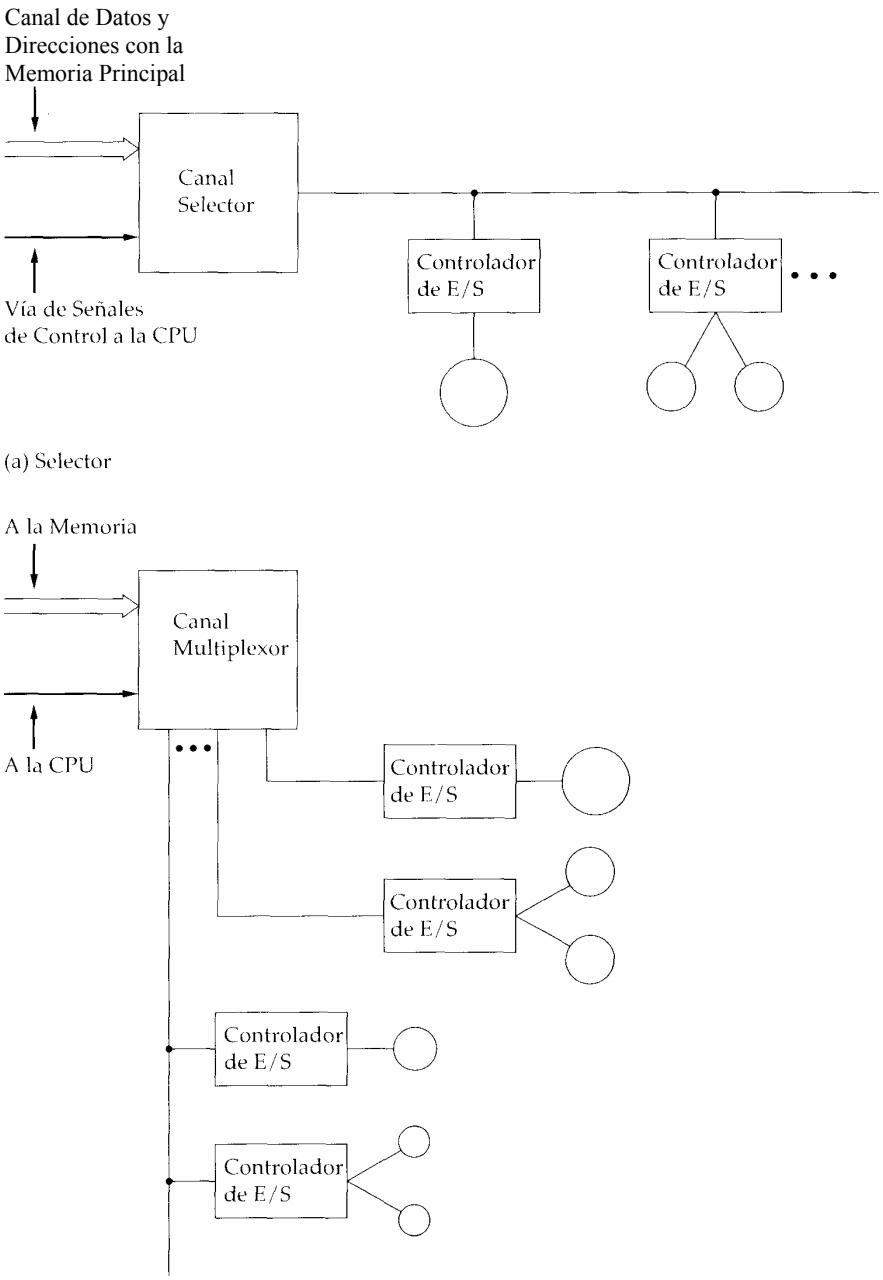


FIGURA 10.4 Arquitectura de un canal de E/S

la memoria principal en los computadores actuales, seguirá dándose el caso de que la E/S no mantenga el nivel con la actividad del procesador. Se puede utilizar el intercambio para introducir más procesos listos para ejecución y mantener así el procesador ocupado, pero ésta es una operación de E/S en sí misma. De este modo, ha habido un gran esfuerzo en el diseño de esquemas de E/S para mejorar la eficiencia. El área que ha recibido una mayor atención, debido a su importancia, ha sido la E/S a disco, estando una gran parle de este capítulo dedicada al estudio de la eficiencia de la E/S a disco.

El segundo gran objetivo es la **generalidad.** En interés de la simplicidad y la exención de errores, será deseable manejar lodos los dispositivos de una manera uniforme. Esta afirmación se aplica tanto a la manera en que los procesos contemplan a los dispositivos de E/S como a la forma en que el sistema operativo gestiona los dispositivos de E/S y las operaciones. Debido a la diversidad de características de los dispositivos, en la práctica es difícil conseguir una generalidad verdadera. Lo que puede hacerse es emplear un enfoque jerárquico y modular para el diseño de las funciones de E/S. Este proceder ocultará la mayoría de los detalles de la E/S con dispositivos en rutinas de bajo nivel, de forma que los procesos y los niveles superiores del sistema operativo contemplen a los dispositivos en términos de funciones generales, como la lectura, escritura, apertura, cierre, bloqueo y desbloqueo. A continuación, la dedicación será para discutir este enfoque.

Estructura Lógica de las Funciones de E/S

En el capítulo 2, al hablar de la estructura del sistema, se hizo énfasis en la naturaleza jerárquica de los sistemas operativos modernos. La filosofía jerárquica propone que las funciones del sistema operativo deben separarse de acuerdo a su complejidad, sus rangos característicos de tiempo y su nivel de abstracción. Seguir este enfoque conduce a una organización del sistema operativo en un conjunto de niveles. Cada nivel realiza una parte afín de las funciones necesarias del sistema operativo. Cada nivel cuenta con el nivel inferior para realizar funciones más básicas y ocultar los detalles de éstas últimas. Asimismo, cada nivel ofrece servicios al nivel superior. En el mejor de los casos, los niveles deben definirse de forma que los cambios en un nivel no provoquen más cambios en otros niveles. De este modo, el problema se ha descompuesto en una serie de subproblemas más manejables.

En general, los niveles inferiores hacen frente a un rango de tiempos mucho menor. Algunas partes del sistema operativo deben interactuar directamente con el hardware del computador, donde los sucesos pueden ocurrir en una escala de tiempos del orden de unos pocos nanosegundos. En el otro extremo del espectro, algunas partes del sistema operativo se comunican con el usuario, que emite órdenes a un ritmo mucho más pausado, como puede ser una cada pocos segundos. El empleo de un conjunto de niveles se adapta bien a este entorno.

La aplicación específica de esta filosofía a la E/S conduce a la clase de organización sugerida por la figura 10.5 (compárese con la tabla 2.4). Los detalles de la organización dependen del tipo de dispositivo y de la aplicación. En la figura se presentan las tres estructuras lógicas más importantes. Por supuesto, puede que un sistema operativo no se ajuste exactamente a estas estructuras. Sin embargo, los principios generales son válidos y la mayoría de los sistemas operativos enfocan la E/S más o menos de esta manera.

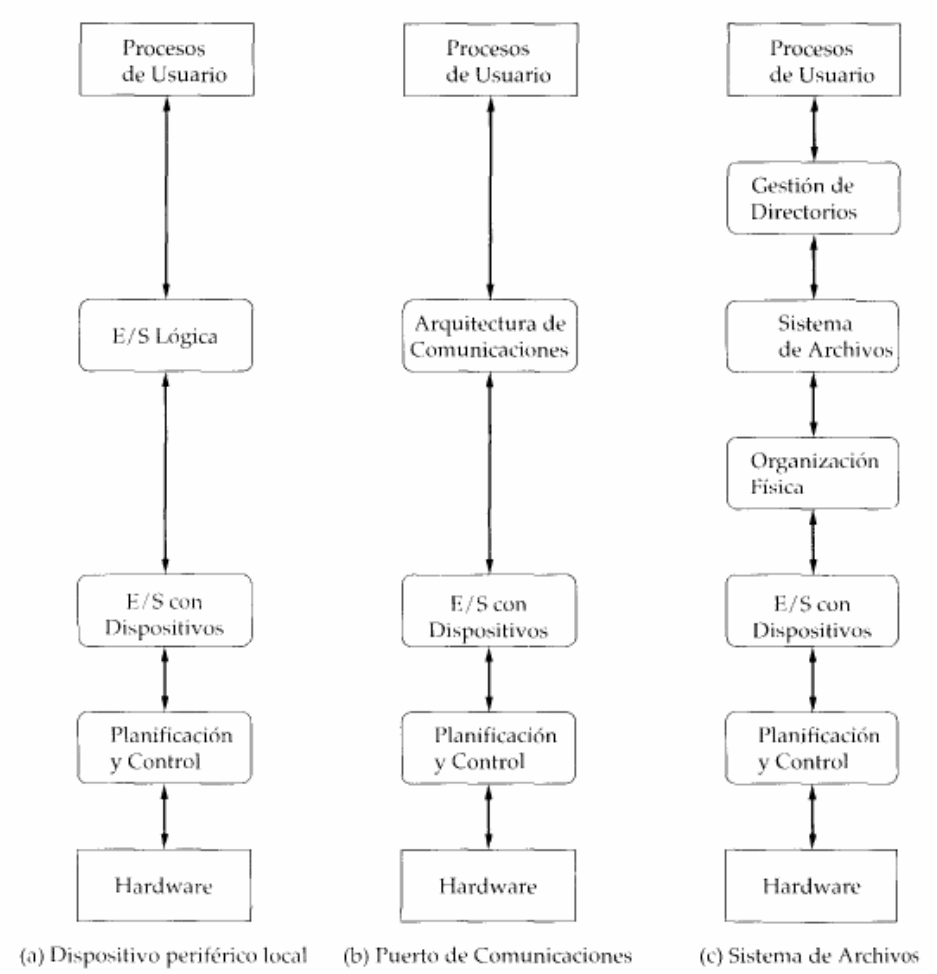


FIGURA 10.5 Un modelo de organización de E/S

Considérese el caso más simple, el primero, de un dispositivo periférico local que se comunica de una manera sencilla, como un flujo de bytes o de registros (figura 10.5a). Los niveles implicados son los siguientes:

• *E/S lógica:* El módulo de E/S lógica trata al dispositivo como un recurso lógico y no se preocupa de los detalles de control real del dispositivo. El módulo de E/S lógica se ocupa de la gestión de funciones generales de E/S pedidas por los procesos de usuario, permitiéndoles manejar el dispositivo mediante un identificador y órdenes simples como Abrir, Cerrar, Leer y Escribir.

• *E/S con dispositivos:* Las operaciones pedidas y los datos (caracteres almacenados, registros. etc.) se convierten en secuencias adecuadas de instrucciones de E/S, comandos para el canal y órdenes al controlador. Se pueden utilizar técnicas de almacenamiento intermedio para mejorar el uso.

• *Planificación* y *control:* La planificación y encolado de las operaciones de E/S ocurren en este nivel, así como el control de las operaciones. Las interrupciones se manejan en este nivel, así como se averigua e informa sobre el estado de la E/S. Este es el nivel del software que realmente interacciona con el módulo de E/S y, por tanto, con el hardware del dispositivo.

Para un dispositivo de comunicaciones, la estructura de E/S (figura 10.5b) se parece mucho a la ya descrita. La diferencia principal es que el módulo de E/S lógica se reemplaza por una arquitectura de comunicaciones, que puede constar, asimismo, de varios niveles. Por ejemplo, la conocida arquitectura de interconexión de sistemas abiertos (OSI) consta de siete niveles. Las arquitecturas de comunicaciones se discutirán en el capítulo 12.

La figura 10.5c muestra la estructura representativa de gestión de E/S en un dispositivo de almacenamiento secundario que soporta un sistema de archivos. Los tres niveles que no han sido descritos antes son los siguientes:

• *Gestión de directorios:* En este nivel se traducen los nombres simbólicos de archivos a identificadores que referencian directamente al archivo o indirectamente, a través de un descriptor de archivo o índice en una tabla. Este nivel se ocupa también de las operaciones del usuario que afectan al directorio de archivos, como Añadir, Borrar y Reorganizar.

• *Sistema de archivos:* Este nivel se encarga de la estructura lógica de los archivos y las operaciones que pueden especificar los usuarios, como Abrir. Cerrar, Leer y Escribir. En este nivel también se gestionan los derechos de acceso.

• *Organización física:* Del mismo modo que las direcciones virtuales de memoria deben convertirse en direcciones tísicas de la memoria principal, teniendo en cuenta la estructura segmentada y paginada, las referencias lógicas a los archivos y registros deben convertirse en direcciones físicas del almacenamiento secundario, teniendo en cuenta la pista física y la estructura en sectores del archivo. La asignación de espacio de almacenamiento secundario y de buffers de almacenamiento principal también se trata generalmente en este nivel.

Debido a la importancia del sistema de archivos, se va a dedicar un tiempo mayor a observar sus diversos componentes, tanto en este capítulo como en el siguiente. La discusión en este capítulo se centra en los tres niveles inferiores, mientras que los dos superiores se examinarán en el capítulo 11.

10.4 ALMACENAMIENTO INTERMEDIO DE E/S

Supóngase que un proceso de usuario desea leer bloques de datos de una cinta, uno cada vez, siendo cada bloque de 100 bytes. Los datos van a ser leídos en una zona de datos del proceso de usuario situada en las direcciones virtuales 1000 a 1009. La forma más sencilla de hacerlo sería emitir una orden de E/S (parecida a "Leer Bloque[1000, cinta]") a la unidad de cinta y esperar a que los datos estén disponibles. La espera podría ser activa (comprobando continuamente el estado del dispositivo) o, de manera más práctica, suspender al proceso en espera de una interrupción.

Hay dos problemas con este enfoque. En primer lugar, el programa se queda colgado esperando a que la relativamente lenta operación de E/S termine. El segundo problema es que este método de E/S dificulta las decisiones de intercambio del sistema operativo. Las ubicaciones virtuales 1000 a 1009 deben permanecer en memoria principal durante el curso de la transferencia del bloque. De lo contrario, parte de los datos se perderán. Si se está utilizando paginación, la página que contenga dichas direcciones virtuales, por lo menos, debe permanecer en memoria principal. De este modo, aunque algunas partes del proceso puedan ser expulsadas a disco, es imposible expulsar al proceso por completo, aunque el sistema operativo lo desee. Nótese también que hay riesgo de interbloqueo de un solo proceso. Si un proceso emite una orden de E/S, queda suspendido a la espera del resultado, se le expulsa antes de comenzar la operación y se bloquea esperando a que la operación termine. Mientras tanto, la operación de E/S queda bloqueada esperando a que el proceso vuelva a memoria. Para evitar este interbloqueo, la memoria de usuario implicada en la operación de E/S debe quedar fija en la memoria principal, inmediatamente después de emitir la petición de E/S, incluso aunque la operación de E/S se encole y pueda no ejecutarse por algún tiempo.

Las mismas consideraciones pueden aplicarse a las operaciones de salida. Si se transfiere un bloque desde el área de un proceso de usuario hacia un módulo de E/S directamente, el proceso se bloqueará durante la transferencia y no puede ser expulsado.

Para evitar esta carga e incapacidad, a veces es conveniente llevar a cabo las transferencias de entrada por adelantado a las peticiones y realizar las transferencias de salida un tiempo después de hacer la petición. Esta técnica se conoce con el nombre de almacenamiento intermedio *(buffering).* En esta sección se van a considerar algunos esquemas de almacenamiento intermedio ofrecidos por los sistemas operativos para mejorar el rendimiento del sistema.

A la hora de discutir los distintos métodos de almacenamiento intermedio, es a veces importante hacer una distinción entre dos tipos de dispositivos: dispositivos de bloques y dispositivos de flujo. Los **dispositivos de bloques** almacenan la información en bloques, normalmente de un tamaño fijo, siendo las transferencias de un bloque cada vez. Generalmente, es posible referirse a los bloques por un número de bloque. Los discos y las cintas son ejemplos de dispositivos de bloques. Los **dispositivos de flujo** transfieren los datos como flujos de bytes; no poseen estructura de bloques. Terminales, impresoras, puertos de comunicación, ratones y otros dispositivos apuntadores y la mayoría de los dispositivos restantes que no son de almacenamiento secundario son dispositivos de flujo.

Buffer Sencillo

La clase de apoyo más simple que el sistema operativo puede ofrecer es el buffer sencillo (figura 10.6b). Cuando un proceso de usuario realiza una petición de E/S, el sistema operativo le asigna a la operación un buffer en la parte del sistema de la memoria principal.

Para los dispositivos de bloques, el esquema del buffer sencillo puede describirse como sigue. Las transferencias de entrada se realizan al buffer del sistema. Cuando se ha completado la transferencia, el proceso mueve el bloque al espacio del usuario y pide otro bloque inmediatamente. Esta técnica se llama *lectura por adelantado o entrada anticipada:* se realiza esperando que el bloque se necesite más adelante. Para muchos tipos de operaciones, ésta suposición es razonable la mayoría de las veces. La lectura del bloque será innecesaria sólo al final de una secuencia de procesamiento.

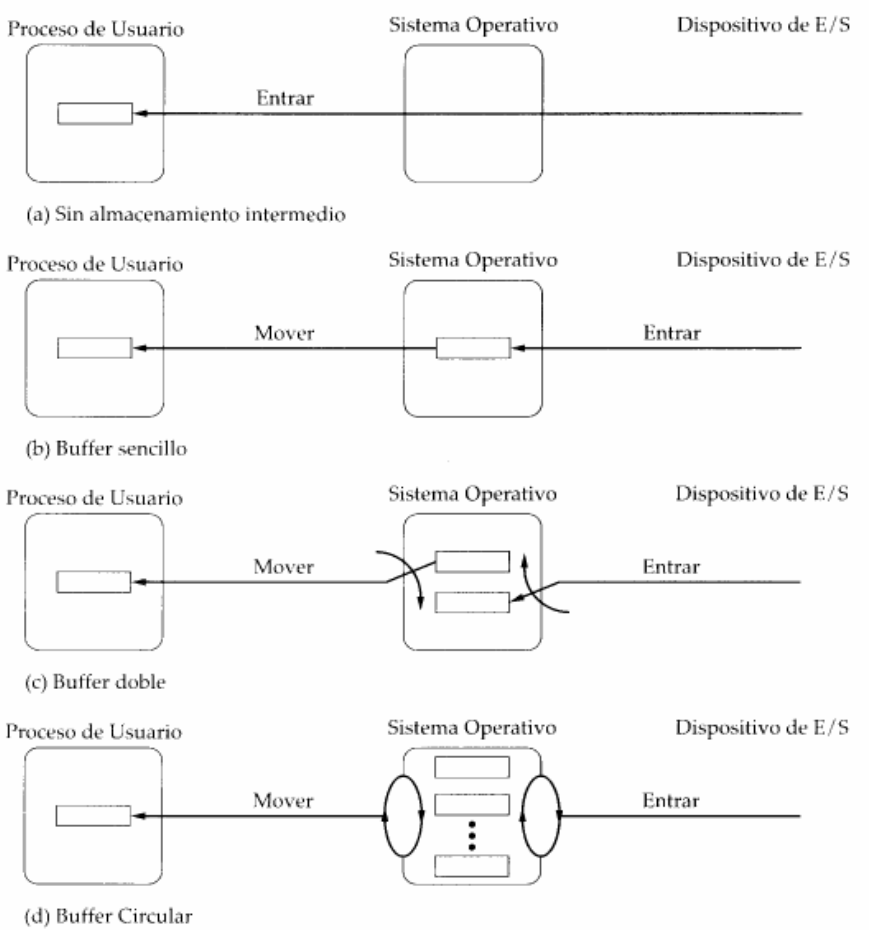


FIGURA 10.6 Esquemas de almacenamiento intermedio de E/S **(entrada)**

Generalmente, este método proporciona una mayor velocidad en comparación con la ausencia de almacenamiento intermedio en el sistema. El proceso de usuario puede procesar un bloque de datos mientras se está leyendo el siguiente. El sistema operativo será capaz de expulsar al proceso porque la operación de entrada tiene lugar dentro de la memoria del sistema en vez de en la memoria de usuario del proceso. Sin embargo, esta técnica complica la lógica del sistema operativo. El sistema operativo debe guardar constancia de las asignaciones de buffers del sistema a los procesos de usuario. La lógica de intercambio también se ve afectada: Si la operación de E/S implica al mismo disco que se usa para intercambio, apenas importa encolar las escrituras al disco necesarias para expulsar al proceso. Este intento de expulsar al proceso y liberar memoria principal no comenzará hasta que la operación de E/S finalice, momento en que la expulsión del proceso al disco puede no ser ya apropiada.

Se pueden aplicar consideraciones similares a la salida con dispositivos de bloques. Cuando se transmiten datos a un dispositivo, deben copiarse primero del espacio de usuario a un buffer del sistema, desde donde serán finalmente escritos. El proceso que realizó la petición podrá entonces continuar o ser expulsado si es necesario.

[KNUT73] sugiere una tosca pero informativa comparación de rendimiento entre el buffer sencillo y la ausencia de buffer. Supóngase que *T* es el tiempo necesario para realizar la entrada de un bloque y C es el tiempo que dura la operación que sucede entre dos peticiones de entrada. Sin almacenamiento intermedio, el tiempo de ejecución por bloque es, esencialmente, 7' + *C.* Con un buffer sencillo, el tiempo es *max(C, T) + M.* donde *M* es el tiempo necesario para mover los datos del buffer del sistema a la memoria de usuario. En la mayoría de los casos, esta última cantidad es sustancialmente menor que la primera.

Para la E/S con dispositivos de flujo, el esquema del buffer sencillo puede aplicarse por líneas o por bytes. La operación línea a línea es adecuada para terminales con desplazamiento (scroll) vertical (a veces llamados terminales "tontos"). Con este tipo de terminales, la entrada del usuario se realiza por líneas, marcadas con un retorno de carro al final de la misma. La salida al terminal es similar, línea a línea. Las impresoras de línea constituyen otro ejemplo de tales dispositivos. La operación por bytes se utiliza en terminales de pantalla completa, donde cada tecla pulsada tiene su significado, así como para otros periféricos, como sensores y controladores.

En el caso de la E/S por líneas, se puede emplear el buffer para guardar una sola línea. El proceso de usuario quedará suspendido durante la entrada, esperando la llegada de la línea completa. Para la salida, el proceso de usuario puede colocar una línea de salida en el buffer y seguir procesando. No será suspendido a menos que llegue una segunda línea para enviar antes de que se vacíe el buffer de la primera operación de salida. En el caso de la E/S por bytes, la interacción entre el sistema operativo y el proceso de usuario sigue el modelo del productor/consumidor discutido en el capítulo 4.

Buffer Doble

Se puede realizar una mejora del buffer sencillo asignando dos buffers del sistema a cada operación (figura l0.6c). De esta forma, un proceso puede transfiere datos hacia (o desde) un buffer mientras que el sistema operativo vacía (o rellena) el otro. Esta técnica se conoce como **buffer doble o intercambio de buffers.**

Para las transferencias de bloques, se puede hacer una estimación aproximada del tiempo de transferencia como el máximo de *C* y *T.* Por tanto, es posible que el dispositivo de bloques funcione a su máxima velocidad si C < *T.* Por otro lado, si *C* > *T.* el buffer doble asegura que el proceso no tendrá que esperar en la E/S. En cualquier caso se consigue una mejora con respecto al buffer sencillo. Sin embargo, esta mejora sufre el coste del incremento de la complejidad.

En la entrada de flujos, se afronta de nuevo el problema de las dos alternativas de operación. Para la E/S de líneas, el proceso de usuario no tiene que ser suspendido para entrada o salida a menos que el proceso se adelante al buffer doble. Para la operación con bytes, el buffer doble no ofrece ninguna ventaja con respecto a un buffer sencillo de doble tamaño. En ambos casos, se seguirá el modelo del productor/consumidor.

Buffer Circular

El esquema del buffer doble debería solucionar el flujo de datos entre un dispositivo de E/S y un proceso. Si preocupa el rendimiento de un proceso determinado, sería deseable que las operaciones de E/S fueran capaces de ir al ritmo del proceso. El buffer doble puede ser inapropiado si el proceso lleva a cabo rápidas ráfagas de E/S. En este caso, el problema puede mitigarse usando más de dos buffers.

Cuando se emplean más de dos, el conjunto de buffers se conoce con el nombre de *buffer circular* (figura 10.6d). Cada buffer individual constituye una unidad del buffer circular. Este es, sencillamente, el modelo del productor/consumidor con un buffer limitado, estudiado en el capítulo 4.

La Utilidad del Almacenamiento Intermedio

El almacenamiento intermedio es una técnica que soluciona los problemas de "horas punta" en la demanda de E/S. Sin embargo, no existe un tamaño de los buffers que asegure a un dispositivo de E/S ir al mismo ritmo que un proceso cuando la demanda media del proceso es mayor que la que el dispositivo puede admitir. Incluso si se dispone de varios buffers, al final todos se llenarán y el proceso tendrá que quedarse esperando tras operar con una determinada cantidad de datos. Sin embargo, en un entorno de multiprogramación, con la variedad de actividades de E/S y de procesamiento que hay que realizar, el almacenamiento intermedio es una herramienta que puede incrementar la eficiencia del sistema operativo y el rendimiento de los procesos individuales.

10.5 ENTRADA/SALIDA A DISCO

En los últimos 30 años, el crecimiento en velocidad de los procesadores y la memoria principal ha dejado muy atrás el de los accesos a disco. La velocidad del procesador y de la memoria se ha incrementado en dos órdenes de magnitud con respecto al disco. El resultado es que, actualmente, los discos son, por los menos, cuatro veces más lentos que la memoria principal. Este avance se espera que continúe en el futuro inmediato. De este modo, el rendimiento de los subsistemas de almacenamiento en disco es de una importancia vital y se han realizado muchas investigaciones sobre maneras de mejorar dicho rendimiento. En esta sección se realzarán los aspectos clave y los métodos más importantes. Como el rendimiento del disco está estrechamente relacionado con cuestiones de diseño, la discusión continuará en el capítulo 11.

Parámetros de Rendimiento de Discos

Los detalles reales de las operaciones de E/S con los discos dependen del computador, el sistema operativo y la naturaleza del canal de E/S y el hardware controlador de disco. En la figura 10.7 se muestra un típico diagrama de tiempos de la E/S a disco.

Cuando la unidad de disco está operando, el disco gira a una velocidad constante. Para leer o escribir, la cabeza debe posicionarse en la pista deseada, al comienzo del sector pertinente. Si el sistema es de cabezas móviles, hay que mover la cabeza para elegir la pista. Si el

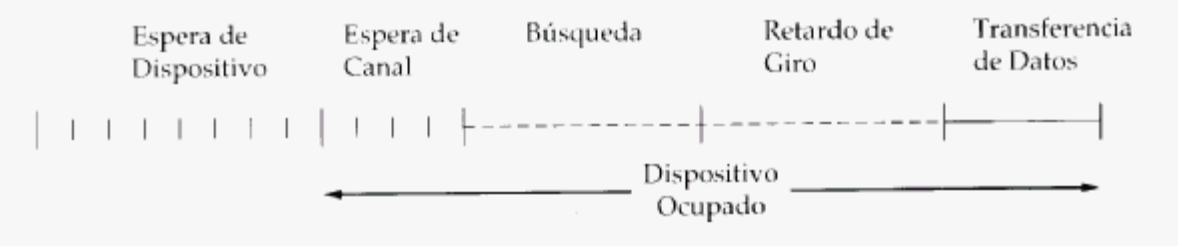


FIGURA 10.7 Medida del tiempo de una transferencia de E/S a disco

sistema es de cabezas fijas, habrá que seleccionar electrónicamente una de ellas. En un sistema de cabezas móviles, el tiempo que se tarda en ubicar la cabeza en la pista se llama **tiempo de búsqueda.** En cualquier caso, una vez que se ha seleccionado la pista, el controlador del disco esperará hasta que el sector apropiado se alinee con la cabeza en su rotación. El tiempo que tarda el comienzo del sector en llegar hasta la cabeza se conoce como **retardo de giro,** o latencia de giro. La suma del tiempo de búsqueda y el retardo de giro es el **tiempo de** acceso, es decir, el tiempo que se tarda en llegar a la posición de lectura o escritura. Una vez que la cabeza está ubicada, se puede llevar a cabo la operación de Lectura o Escritura a medida que el sector se mueve bajo la cabeza; esta es la parte de transferencia real de datos de la operación.

Además del tiempo de acceso y del tiempo de transferencia, en una operación de E/S intervienen algunos retardos. Cuando un proceso emite una petición de E/S, primero debe esperar en una cola a que el dispositivo esté disponible. En ese momento, el dispositivo queda asignado al proceso. Si el dispositivo comparte un único canal de E/S o un conjunto de canales con otras unidades de disco, puede producirse una espera adicional hasta que el canal esté disponible. En ese punto se realizará la búsqueda con que comienza el acceso al disco.

En algunos sistemas grandes se emplea una técnica conocida como *detección posicional de giro (RPS).* Esta técnica funciona como se explica seguidamente. Cuando se ejecuta la orden de búsqueda, se libera el canal para que pueda realizar otras operaciones de E/S. Cuando la búsqueda termine, el dispositivo debe averiguar el instante en que los datos van a pasar bajo la cabeza. A medida que el sector se aproxima a la cabeza, el dispositivo intenta restablecer la vía de comunicaciones con el computador central. Si la unidad de control o el canal están ocupados con otra operación de E/S, el intento de reconexión no tendrá éxito y el dispositivo debe dar una vuelta completa antes de intentar la reconexión, lo que se denomina una falta *de RPS.* Esta componente extra del retardo debe añadirse al diagrama de tiempos de la figura 10.7.

Tiempo de Búsqueda

El tiempo de búsqueda es el tiempo necesario para mover el brazo del disco hasta la pista solicitada. Esta cantidad resulta difícil de concretar. El tiempo de búsqueda consta de dos componentes clave: el tiempo de arranque inicial y el tiempo que se tarda en recorrer los cilindros, una vez que el brazo haya cogido velocidad. Por desgracia, el tiempo de recorrido no es una función lineal con el número de pistas. Se puede aproximar el tiempo de búsqueda con la fórmula lineal:

$T_s = m \times n + s$

donde
T = tiempo de búsqueda estimado

n = número de pistas recorridas

m = constante que depende de la unidad de disco

s = tiempo de arranque

Por ejemplo, un disco *Winchester* económico en un computador personal podría tener, aproximadamente, m = 0,3 ms y s = 20 ms, mientras que uno más grande y más caro podría tener m = 0,1 ms y x = 3 ms.

Retardo de Giro
Los discos, excepto los Flexibles, giran normalmente a 3600 rpm, es decir, una revolución cada 16,7 ms. Por tanto, el retardo medio de giro será de 8,3 ms. Los discos flexibles giran mucho más lentamente, generalmente entre 300 y 600 rpm. Por tanto, el retardo medio estará entre 100 y 200 ms.

Tiempo de Transferencia El tiempo de transferencia con el disco depende de la velocidad de rotación de la forma siguiente:

$T = b/(rN)$ donde

T = tiempo de transferencia

b = número de bytes a transferir

N = número de bytes por pista

r = velocidad de rotación en revoluciones por segundo

Por tanto, el tiempo medio de acceso total puede expresarse como

$T_a = T_s+(1/2r)+(b/rN)$ donde T_s es el tiempo medio de búsqueda.

Comparativa de Tiempos

Habiendo definido los parámetros anteriores, se va a atender a continuación a dos operaciones de E/S que muestran el peligro de confiar en los valores medios. Considérese un disco típico con un tiempo medio de búsqueda conocido de 20 ms, velocidad de transferencia de 1 Mb/sg y sectores de 512 bytes, habiendo 32 sectores por pista. Supóngase que se desea leer un archivo que consta de 256 sectores para formar un total de 128 Kb. Así podría estimarse el tiempo total que dura la transferencia.

En primer lugar, supóngase que el archivo se almacena en el disco de la forma más compacta posible. Es decir, el archivo ocupará todos los sectores de ocho pistas adyacentes (8 pistas 32 sectores/pista = 256 sectores). Esta disposición se conoce como *organización secuencial.* En tal caso, el tiempo que dura la lectura de la primera pista es el siguiente:

Búsqueda media	20,0 ms
Retardo de giro	8,3 ms
Lectura de 32 sectores	16,7 ms
	45,0 ms

Supóngase que las pistas restantes pueden leerse ahora sin tiempo de búsqueda alguno. Es decir, la operación de E/S puede llevar el ritmo del flujo de datos que llega del disco. En tal caso hace falta considerar, como mucho, el retardo de giro de las pistas sucesivas. Por tanto, cada pista consecutiva se puede leer en 8,3 + 16,7 = 25 ms. Para leer el archivo por completo:

Tiempo total = 45 + 7 x 25 = 220 ms = 0,22 sg.

Se calcula ahora el tiempo necesario para leer los mismos datos utilizando acceso aleatorio en vez de acceso secuencial: esto es, el acceso a los sectores se distribuye aleatoriamente por el disco. Para cada sector, se tiene:

Búsqueda media	20,0 ms
Retardo de giro	8,3 ms
Lectura de 1 sector	0,5 ms
	28,8 ms

Tiempo total = 256 x 28,8 = 7373 ms = 7,37 sg.

Está claro que el orden en que se leen los sectores del disco tiene un efecto inmenso en el rendimiento de la E/S. En el caso de los accesos a archivos en los que se leen varios sectores, se puede ejercer algún control sobre la manera en que se distribuyen los sectores de datos. En el próximo capítulo se comentará algo al respecto. Sin embargo, incluso en el caso del acceso a un archivo en un entorno de multiprogramación, existirán varias solicitudes de E/S que compitan por el mismo disco. Por tanto, merece la pena investigar alguna manera más de mejorar el rendimiento de la E/S a disco, sobre todo del conseguido con un acceso puramente aleatorio. En el resto de la sección se van a examinar dos de las estrategias más conocidas: la planificación del disco y la memoria intermedia *(caché)* de disco. En el capítulo siguiente se estudiará la organización de los archivos y las cuestiones de almacenamiento que influyen en el rendimiento.

Políticas de Planificación de Discos

Si se observa el ejemplo de la sección anterior, se puede apreciar que la razón de la diferencia en rendimiento puede encontrarse en el tiempo de búsqueda. Si las solicitudes de acceso a un sector necesitan de la selección de pistas aleatorias, el rendimiento del sistema de E/S a disco será muy pobre. Para mejorarlo, hay que reducir el tiempo medio gastado en las búsquedas.

Considérese una situación normal de un entorno de multiprogramación, en el que el sistema operativo mantiene una cola de peticiones para cada dispositivo de E/S. De este modo, para un disco sencillo, en la cola habrá peticiones de E/S (lecturas y escrituras) procedentes de varios procesos. Si se eligen los elementos de la cola en un orden aleatorio, se puede esperar que las pistas recorridas sigan también un orden aleatorio, obteniéndose el peor rendimiento posible. Esta **planificación aleatoria** es útil como medida comparativa para evaluar otras técnicas.

La manera más sencilla de planificación sería la de "Primero en entrar, primero en salir" (FIFO), lo que significa que los elementos se procesan de la cola en un orden secuencial. Esta estrategia tiene la ventaja de ser justa porque las peticiones son servidas en el orden en que llegaron. La figura 10.8a representa el movimiento del brazo del disco con FIFO en comparación con otras tres políticas. En este ejemplo, se dispone de un disco de 200 pistas y se supone que las peticiones llegan aleatoriamente a la cola del disco. Las pistas solicitadas, en el orden recibido, son: 55, 58, 39, 18, 90, 160. 150. 38, 184. La tabla 10.3a refleja los resultados.

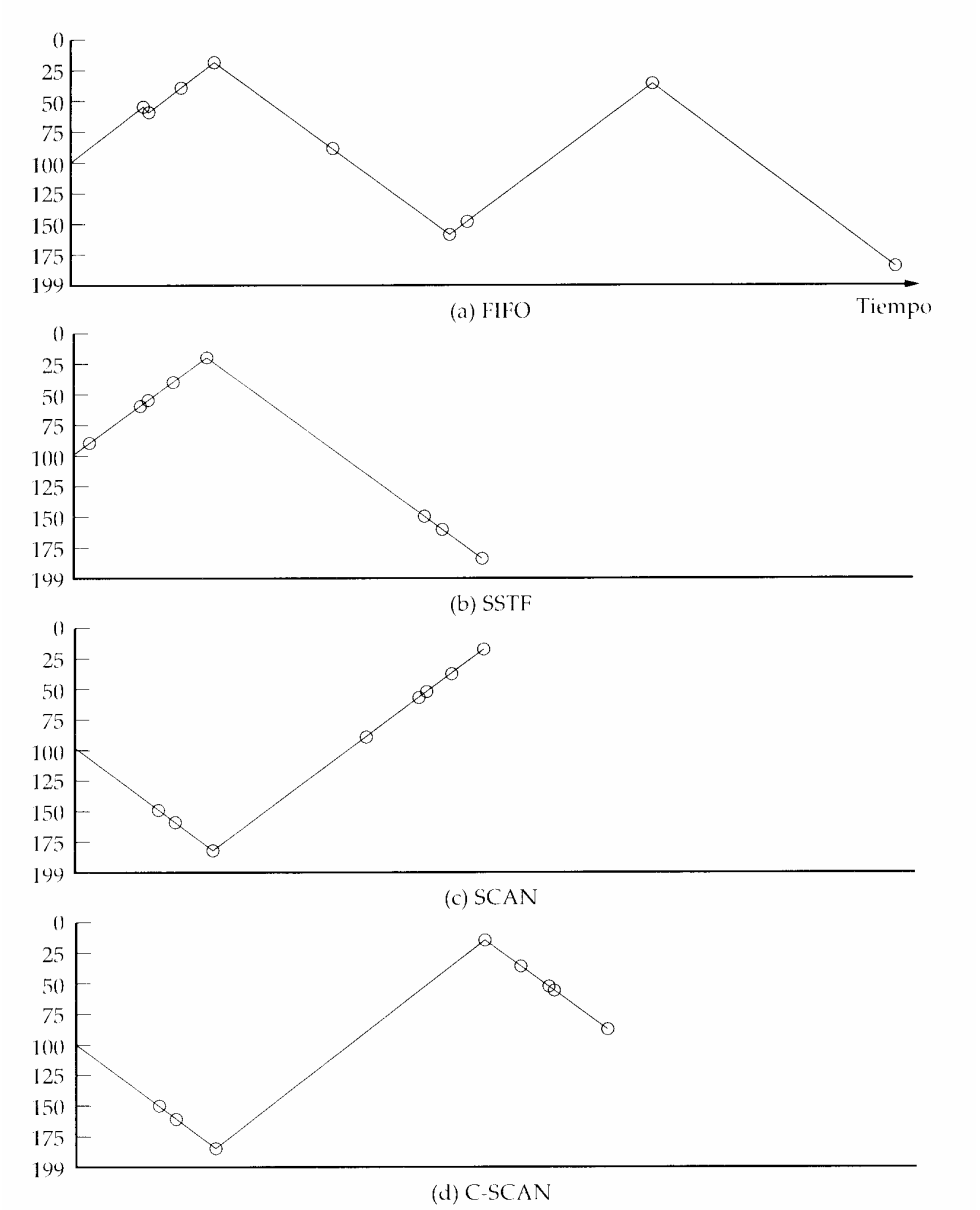


FIGURA 10.8 Comparación de algoritmos de planificación del disco (ver tabla 10.3)

Con la técnica FIFO. si hay pocos procesos que requieren acceso y si muchas de las peticiones son a sectores agrupados de un archivo, se puede esperar un buen rendimiento. Sin embargo, el rendimiento de esta técnica se parece a menudo al de la planificación aleatoria si hay muchos procesos compitiendo por el disco. Así, puede ser más beneficioso considerar una política de planificación mas sofisticada. En la tabla 10.4 se relatan algunas de ellas, consideradas a continuación.

TABLA 10.3 Comparación de Algoritmos de Planificación del Disco

(a) FIFO (comenzando en la pista 100)		(b) SSTF (comenzando en la pista 100)		(c) SCAN (comenzando en la pista 100, en direcciones crecientes de número de pista)		(d) C-SCAN (comenzando en la pista 100, en direcciones crecientes de número de pista)	
Siguiente pista accedida	Número de pistas recorridas	Siguiente pista accedida	Número de pistas recorridas	Siguiente pista accedida	Número de pistas recorridas	Siguiente pista accedida	Número de pistas recorridas
55	45	909	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Tiempo medio de búsqueda:	55,3	Tiempo medio de búsqueda:	27,5	Tiempo medio de búsqueda:	27,8	Tiempo medio de búsqueda:	35,8

Prioridad

Con un sistema de prioridades (PRI), el control de la planificación queda aislado del control del software gestor del disco. Este enfoque no persigue la optimización del uso del disco, sino cumplir con otros objetivos del sistema operativo. Los trabajos por lotes que sean cortos y los trabajos interactivos reciben frecuentemente una prioridad más alta que trabajos mayores que realizan largas operaciones. Esta práctica permite que el sistema haga salir más rápidamente a muchos trabajos cortos y pueda proporcionar un buen tiempo de respuesta interactiva. Sin embargo, los trabajos mayores pueden tener que esperar excesivamente. Más aún, esta política podría conducir a contramedidas por parte de los usuarios, que pueden dividir sus trabajos en trozos más pequeños para explotar el sistema. Este tipo de política tiende a ser poco favorable para sistemas de bases de datos.

Ultimo en Entrar, Primero en Salir

Sorprendentemente, la política de tomar siempre la petición más reciente tiene alguna virtud. En los sistemas de proceso de transacciones, conceder el dispositivo al último usuario acarrea pocos o nulos movimientos del brazo al recorrer un fichero secuencia!. El provecho de esta cercanía mejora la productividad y reduce la longitud de las colas. A medida que un trabajo utiliza de forma activa el sistema de archivos, va procesándose tan rápido como es posible. Sin embargo, si el disco está ocupado con una carga de trabajo larga, existe la posibilidad inconfundible de inanición. Una vez que un trabajo ha lanzado una petición de E/S a la cola y haya abandonado la cabeza, no podrá volver a ganar la cabeza a menos que se vayan todos los que estén por delante.

La política FIFO, la de prioridades y el esquema LIFO (último en entrar, primero en salir) se basan únicamente en las propiedades de la cola o del proceso demandante. Si la posición de la pista actual es conocida por el planificador, puede emplearse una planificación en función del elemento demandado. A continuación se examinarán dichas políticas.

TABLA 10.4 Algoritmos de Planificación de Discos [WIED87]

Nombre	Descripción	Comentarios
Selección en función del demandante:		
RSS	Planificación Aleatoria	Para análisis y simulación
FIFO	Primero en entrar, primero en salir	El más justo de todos
PRI	Prioridad del proceso	El control se lleva aparte de la gestión de la cola del disco
LIFO	Ultimo en entrar, primero en salir	Maximiza la utilización de recursos y aprovecha la cercanía
Selección en función del elemento solicitado:		
SSTF	Primero el más corto	Gran aprovechamiento y colas pequeñas
SCAN	Recorrer el disco de un lado a otro	Mejor distribución del servicio
C-SCAN	Recorrer el disco en un solo sentido	Menor variabilidad en el servicio
SCAN de N	SCAN de N registros a la vez	Garantía de servicio
FSCAN	SCAN de N pasos, con N = longitud de la cola al comienzo del ciclo del SCAN	Sensible a la carga

Primero el más corto

La política de "primero el más corto" (SSTF) es elegir la solicitud de E/S a disco que requiera el menor movimiento posible del brazo del disco desde su posición actual. De este modo, siempre se elige procurando el mínimo tiempo de búsqueda. Por supuesto, la elección siempre del menor tiempo de búsqueda no garantiza que sea mínimo el tiempo medio de búsqueda de entre una serie de movimientos. Sin embargo, esta elección debe ofrecer un rendimiento mejor que el del FIFO. Como el brazo puede moverse en ambos sentidos, se puede usar un algoritmo aleatorio de desempate para resolver los casos de igualdad de distancias.

La figura 10.8b y la tabla 10.3b muestran el rendimiento del SSTF con el mismo ejemplo que se empleó para el FIFO.

SCAN

Con la excepción del FIFO, todas las políticas descritas hasta ahora pueden dejar alguna petición incumplida hasta que se vacíe la cola entera. Es decir, pueden llegar siempre nuevas peticiones que se elegirán antes que una petición existente. Una alternativa simple que previene este tipo de inanición es el algoritmo SCAN.

Con el SCAN. el brazo sólo se puede mover en un sentido, resolviendo todas las peticiones pendientes de su ruta, hasta que alcance la última pista o hasta que no haya más peticiones en esa dirección. Esta última puntualización se conoce a veces como la política de LOOK. Se cambia entonces la dirección de servicio y el rastreo sigue en sentido opuesto, volviendo a recoger todas las peticiones en orden.

La figura 10.Se y la tabla 10.3c ilustran la política del SCAN. Como puede verse, esta política se comporta de manera muy parecida a la SSTF. De hecho, si se supone que, al principio del ejemplo, el brazo se mueve en direcciones decrecientes de números de pista, el modelo de planificación sería idéntico para SSTF y SCAN. Sin embargo, éste es un ejemplo estático en el que no se añaden nuevos elementos a la cola. Incluso cuando la cola cambia dinámicamente, el SCAN es muy similar al SSTF, a menos que el tipo de peticiones no sea muy común.

Nótese que la política del SCAN no es imparcial con la zona que acaba de recorrerse, pues no aprovecha tan bien la cercanía como el SSTF o incluso un **LIFO.**

No es difícil comprobar que la política del SCAN favorece a los trabajos con peticiones de pistas cercanas a los cilindros más interiores y exteriores, así como a los últimos trabajos en llegar. El primer problema se puede evitar con la política del C-SCAN, mientras el segundo puede abordarse con el SCAN de N pasos.

C-SCAN

La política del C-SCAN restringe el rastreo a una sola dirección. Así, cuando se haya visitado la última pista en un sentido, el brazo vuelve al extremo opuesto del disco y comienza a recorrerlo de nuevo, lo que reduce el retardo máximo sufrido por las nuevas peticiones. Con el SCAN. si / es el tiempo esperado de un recorrido desde la pista más interior a la más exterior, entonces el intervalo de servicio esperado para los sectores de los extremos es *1t.* Con el C-SCAN. el intervalo es del orden de $t + s_{max}$ donde s_{max} es el tiempo de búsqueda máximo.

La figura 10.8d y la tabla 10.3d ilustran el comportamiento del C-SCAN.

SCAN de N pasos y FSCAN

Con SSTF, SCAN y C-SCAN, es posible que el brazo no se mueva durante un tiempo considerable. Por ejemplo, si uno o varios procesos realizan una alta proporción de accesos a una pista, pueden monopolizar el dispositivo entero por medio de peticiones repetidas a dicha pista. Los discos de alta densidad con múltiples superficies son más propensos a verse afectados por esta característica que los de baja densidad y/o los discos con sólo una o dos caras. Para evitar esta "pegajosidad" del brazo, la cola de peticiones del disco puede dividirse en segmentos, procesándose un segmento por completo cada vez. Dos ejemplos de este método son el SCAN de *N* pasos y el FSCAN.

La política del SCAN de *N* pasos divide la cola de peticiones del disco en subcolas de longitud *N*. Las subcolas se procesan una a una mediante un SCAN. Mientras se procesa una cola, se añadirán nuevas peticiones a las otras. Si hay menos de *N* peticiones disponibles al final del rastreo, entonces todas serán procesadas en el siguiente recorrido. Para valores grandes de *N*, el rendimiento del SCAN de *N* pasos se aproxima al del SCAN; con un valor de $N = 1$, se está adoptando la política FIFO.

La política FSCAN emplea dos subcolas. Cuando comienza un rastreo, todas las peticiones están en una de las colas y la otra permanece vacía. Durante el recorrido, todas las peticiones nuevas se colocan en la cola que inicialmente estaba vacía. De este modo, el servicio de nuevas peticiones se retrasará hasta que se hayan procesado las viejas.

En la tabla 10.4 se resumen los distintos algoritmos de planificación del disco.

Caché de Disco

En la sección 1.7 y el apéndice 1A se resumen los principios de la memoria caché. El término *memoria caché* se aplica normalmente a una memoria más pequeña y más rápida que la memoria principal y que se sitúa entre ésta y el procesador. Este tipo de memoria caché reduce el tiempo medio de acceso a memoria aprovechándose del principio de cercanía.

El mismo principio puede aplicarse a la memoria de disco. Más en concreto, una caché de disco es un buffer para sectores de disco situado en la memoria principal. La caché contiene una copia de algunos sectores del disco. Cuando se hace una petición de E/S para un sector específico, se comprueba si el sector está en la caché del disco. Si es así, la petición se cumple con la caché. Si no, se lee el sector pedido del disco y se coloca en la caché. Debido a la cercanía de referencias, cuando se traiga un bloque de datos a la caché para satisfacer una sola petición de E/S, será probable que se produzcan referencias futuras al mismo bloque.

Consideraciones de Diseño

Son de interés varias cuestiones de diseño. En primer lugar, cuando una petición de E/S se sirve con la caché, los datos de la misma deben ser enviados al proceso que los solicitó. El envío puede hacerse por una transferencia en memoria del bloque de datos, desde la caché del disco a la memoria asignada al proceso de usuario o, simplemente, usar la posibilidad de memoria compartida y pasar un puntero a la entrada apropiada de la caché del disco. Este último método ahorra el tiempo de la transferencia interna en memoria y, además, permite el acceso compartido de otros procesos que puedan seguir el modelo de los lectores/escritores descrito en el capítulo 4.

Una segunda cuestión de diseño tiene que ver con la estrategia de reemplazo. Cuando se trae un nuevo sector a la caché del disco, uno de los bloques existentes debe ser sustituido. Este problema es idéntico al presentado en el capítulo 6, donde la exigencia era para un algoritmo de reemplazo de páginas. Se han probado un buen número de algoritmos. El algoritmo más común es el de "el usado hace más tiempo" (LRU), en el que el bloque que ha permanecido sin referencias en la caché por más tiempo es reemplazado. Lógicamente, la caché constará de una pila de bloques, estando situado el más reciente en la cima de la pila. Cuando se referencia un bloque de la caché, se le mueve de su posición a la cima de la pila. Cuando se trae un bloque de la memoria secundaria, se elimina el bloque que está en el fondo de la pila, colocando al recién llegado en la cima de la pila. Naturalmente, no es necesario mover estos bloques por la memoria: puede asociarse una pila de punteros a la caché.

Otra posibilidad es el algoritmo de "la menos usada" (LFU), donde se sustituye el bloque de la caché que ha sufrido un menor número de referencias. El algoritmo LFLJ podría im-plementarse asociando un contador a cada bloque. Cuando se trae un bloque, se le asigna un valor de 1: con cada referencia al bloque, se incrementa el contador en una unidad. Cuando hace falta un reemplazo, se selecciona el bloque con menor valor del contador. Intuitivamente, podría parecer que el LFU es más adecuado que el LRU porque se emplea más información de cada bloque en el proceso de selección.

El sencillo algoritmo de LFU tiene el siguiente problema. Puede ser que ciertos bloques se referencien poco frecuentemente, pero cuando lo son, se produzcan intervalos cortos de referencias repetidas, debido a la cercanía, obteniéndose así grandes valores del contador de referencias. Tras este intervalo, el valor del contador de referencias puede ser engañoso y no reflejar la probabilidad de que el bloque sea referenciado nuevamente. De este modo, el efecto de la cercanía puede originar que el algoritmo LFU realice malas elecciones en el reemplazo.

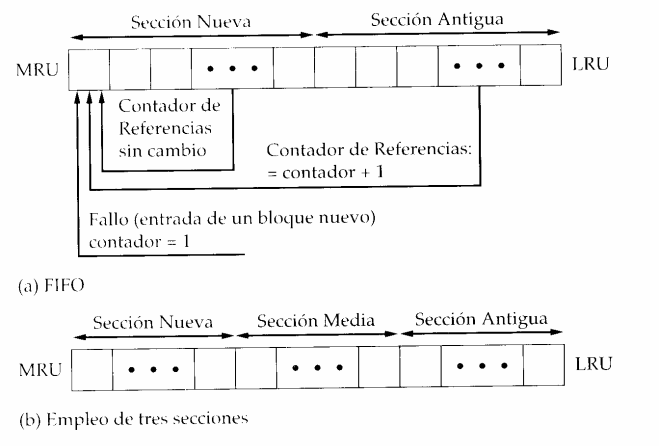


FIGURA 10.9 Reemplazo en función de la frecuencia

Para superar esta dificultad del LFU, en [ROBI90] se propone una técnica conocida como *reemplazo en función de la frecuencia.* Para mayor claridad, considérese una versión simplificada, ilustrada en la figura 10.9a. Los bloques están organizados lógicamente en una pila, como en el algoritmo LRU. Una parte determinada de la cima de la pila es reservada como una *sección nueva.* Cuando se hace blanco en la caché, el bloque referenciado es trasladado a la cima de la pila. Si el bloque ya estaba en la sección nueva, su contador de referencias no se incrementará: en otro caso, se incrementa en 1. Con una sección nueva suficientemente grande, el resultado de este procedimiento es que el contador de los bloques referenciados repetidamente en un corto intervalo de tiempo permanece inalterado. Si se produce una falta, se elegirá para reemplazar el bloque con el menor valor del contador de referencias que no esté en la sección nueva; en caso de empate, se elegirá el usado hace más tiempo.

Los autores comentan que con esta estrategia sólo se consiguió una leve mejora sobre el LRU. El problema consiste en lo siguiente:

1. Cuando se produzca una falta de caché, se traerá un nuevo bloque a la sección nueva, con un contador de uno.
2. El contador permanece a uno mientras el bloque se quede en la sección nueva.
3. El bloque envejece y, finalmente, sale de la sección nueva con su contador todavía a uno.
4. Si el bloque no vuelve a ser referenciado rápidamente, es muy probable que sea reemplazado porque, forzosamente, posee el menor contador de todos los bloques que no están en la sección nueva. En otras palabras, no parece haber un intervalo suficientemente grande para que los bloques que salen de la sección nueva aumenten sus contadores, incluso si han sido referenciados más o menos frecuentemente.

Una refinación adicional dirigida a este problema, dividir la pila en tres secciones: nueva, media y vieja (figura 10.9b). Como antes, las cuentas de referencias no se incrementan en los bloques de la sección nueva. Sin embargo, sólo los bloques de la sección vieja serán candidatos para el reemplazo. Disponiendo de una sección media suficientemente grande, a los bloques referenciados más o menos frecuentemente se les da la oportunidad de aumentar sus contadores de referencias antes de ser candidatos al reemplazo. Los estudios de simulación relatados en [ROBI90] indican que esta política mejorada es significativamente mejor que un simple LRU o LFU.

Sin considerar la estrategia de reemplazo particular, la sustitución puede llevarse a cabo bajo demanda o puede ser planificada previamente. En el primer caso, los sectores se sustituyen sólo cuando se necesita su entrada de la tabla. En el último caso, cada vez se liberan un conjunto de entradas. La razón de ser de este método está relacionada con la necesidad de volver a escribir los sectores. Si se trae un sector a la caché y sólo es leído, no será necesario volver a escribirlo al disco cuando sea reemplazado. Sin embargo, si el sector es actualizado, entonces sí será necesario volver a escribirlo antes de reemplazarlo. En este último caso, tiene sentido agrupar las escrituras y ordenarlas para minimizar el tiempo de búsqueda.

Consideraciones de Rendimiento

Aquí se pueden aplicar las mismas consideraciones sobre el rendimiento discutidas en el apéndice 1A. El tema del rendimiento de la caché se ve reducido a la cuestión de si se puede alcanzar una determinada tasa de faltas. Esto dependerá de la cercanía de las referencias al disco, el algoritmo de reemplazo y otros factores de diseño. Sin embargo, la tasa de faltas es principalmente función del tamaño de la caché de disco. La figura 10.10 resume los resulta-

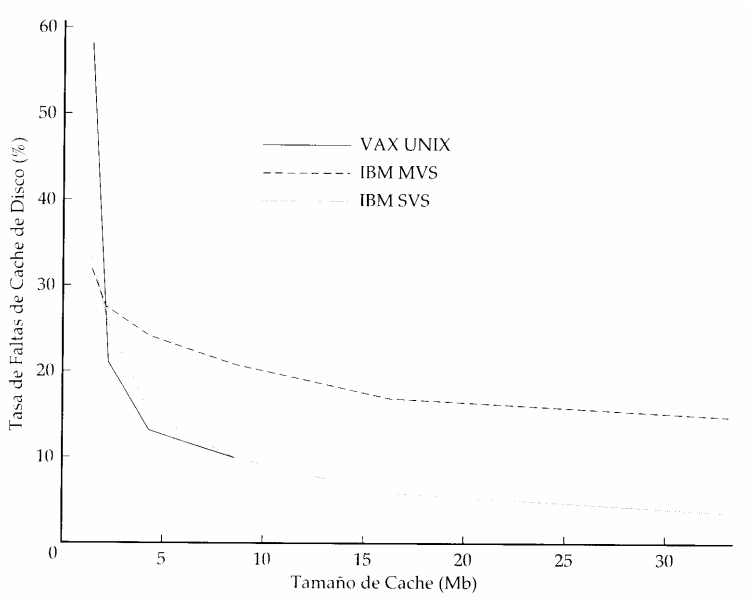


FIGURA 10.10 Resultados del rendimiento de la caché de disco usando LRU

dos de diversos estudios que utilizaron LRU, uno para un sistema UNIX ejecutando en un VAX [OUST85] y otro para los sistemas operativos de los grandes computadores de IBM [SMIT85J. En la figura 10.11 se aprecian los resultados de estudios de simulación sobre los algoritmos de reemplazo en función de la frecuencia. La comparación de ambas gráficas señala uno de los riesgos de este tipo de valoraciones del rendimiento. Las figuras parecen mostrar que el algoritmo LRU supera al de reemplazo en función de la frecuencia. Sin embargo, cuando se comparan pautas de referencia idénticas, que empleen la misma estructura de caché, el algoritmo de reemplazo en función de la frecuencia es superior.

De este modo, la secuencia exacta de pautas de referencia, unida a cuestiones relativas al diseño, como puede ser el tamaño de bloque, tendrán una enorme influencia en el rendimiento conseguido.

10.6 SISTEMAS DE EJEMPLO

Unix Sistema V

En UNIX. cada dispositivo particular de E/S tiene asociado un *archivo especial,* gestionado por el sistema de archivos, del que se lee y se escribe de la misma forma que los archivos de

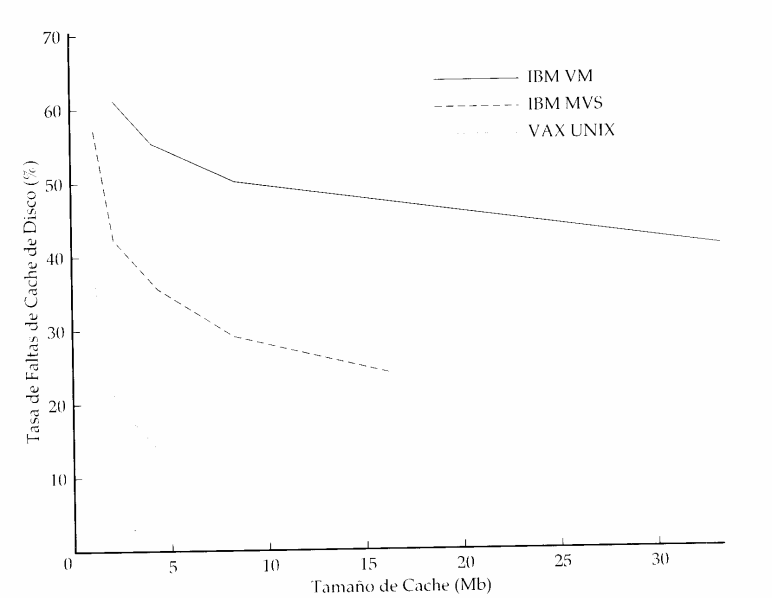


FIGURA 10.11 Rendimiento de la caché de disco usando reemplazo en función de la frecuencia [ROBI90I

datos del usuario. Así se ofrece una interfaz uniforme y bien definida con los usuarios y los procesos. Para leer o escribir en un dispositivo, se realizarán peticiones de lectura o escritura al archivo especial asociado con el dispositivo.

En la figura 10.12 se puede observar la estructura lógica del sistema de E/S. El subsistema de archivos realiza la gestión de los archivos de los dispositivos de almacenamiento secundario. Además, a los procesos les sirve de interfaz con los dispositivos, ya que estos se tratan como si fueran archivos.

En UNIX hay dos tipos de E/S: amortiguada y no amortiguada. La E/S amortiguada aprovecha los buffers del sistema, mientras que la no amortiguada utiliza DMA, realizándose directamente la transferencia entre el módulo de E/S y la zona de E/S del proceso. Con E/S amortiguada se pueden usar dos clases de buffers: buffers del sistema y colas de caracteres.

Caché de Buffers

La caché de buffers en UNIX es, básicamente, una caché de disco. Las operaciones de E/S con el disco se manejan a través de la caché de buffers. La transferencia de datos entre la caché de buffers y el espacio de usuario del proceso siempre se produce mediante DMA. Como la caché de buffers y la zona de E/S del proceso residen ambas en memoria principal, se usará DMA para llevar a cabo una copia de memoria a memoria. Esta acción no gastará ningún ciclo del procesador, pero consumirá ciclos del bus.

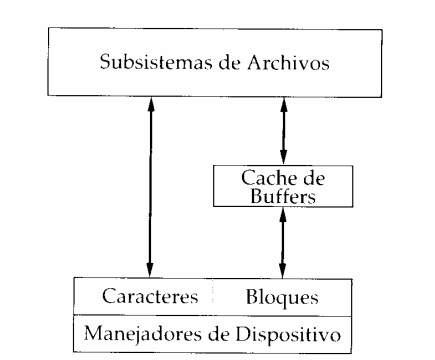


FIGURA 10.12 Estructura de la E/S en UNIX

Para administrar la caché de buffers se van a mantener tres listas:

- *Lista de libres:* Lista de todas las entradas de la caché que están disponibles para asignación (en UNIX, una "entrada" se refiere a un buffer; cada entrada almacena un sector de disco).
- *Lista de dispositivos:* Lista de todos los buffers que están asociados actualmente a cada disco.
- *Cola de E/S del manejador:* Lista de buffers que se someten o esperan por la E/S con un dispositivo determinado.

Cada uno de los buffers debería pertenecer a la lista de libres o a la cola de E/S del mane-jador. Una vez que un buffer se asocia a un dispositivo, permanecerá asociado al mismo, incluso si está en la lista de libres, hasta que se utilice de nuevo y se le asocie a otro dispositivo. Estas listas se mantienen como punteros asociados con cada buffer más que como listas físicamente separadas.

Cuando se hace una referencia a un número de bloque físico de un dispositivo particular, el sistema operativo comprueba primero si el bloque está en la caché de buffers. Para minimizar el tiempo de búsqueda, la lista de dispositivos se organiza como una tabla de dispersión *(hash)* por medio de una técnica similar al encadenamiento separado discutido en el apéndice 7A (figura 7.27b). La figura 10.13 representa la organización general de la caché de buffers. Existe una tabla de dispersión de tamaño fijo que contiene punteros a la caché de buffers. Cada referencia de la forma (n^{o} dispositivo, n" bloque) se traduce en una entrada particular de la tabla. El puntero asociado a dicha entrada apunta al primer buffer de la cadena. Un puntero de dispersión asociado a cada buffer apunta al siguiente buffer de la cadena para dicha entrada de la tabla. De este modo, para todas las referencias del tipo (n^{o} dispositivo, n" bloque) que se traduzcan en una misma entrada de la tabla, si el bloque correspondiente está en la caché de bloques, entonces dicho buffer estará en la cadena asociada. Así, la longitud de la búsqueda en la caché de buffers se reduce en un factor del orden de *N,* donde *N* es el tamaño de la tabla de dispersión.

Para el reemplazo de bloques se utiliza un algoritmo LRU: Después de que se haya asignado un buffer a un bloque de disco, no podrá ser usado por otro bloque hasta que todos los demás buffers se hayan usado. La lista de libres es la que mantiene este orden LRU.

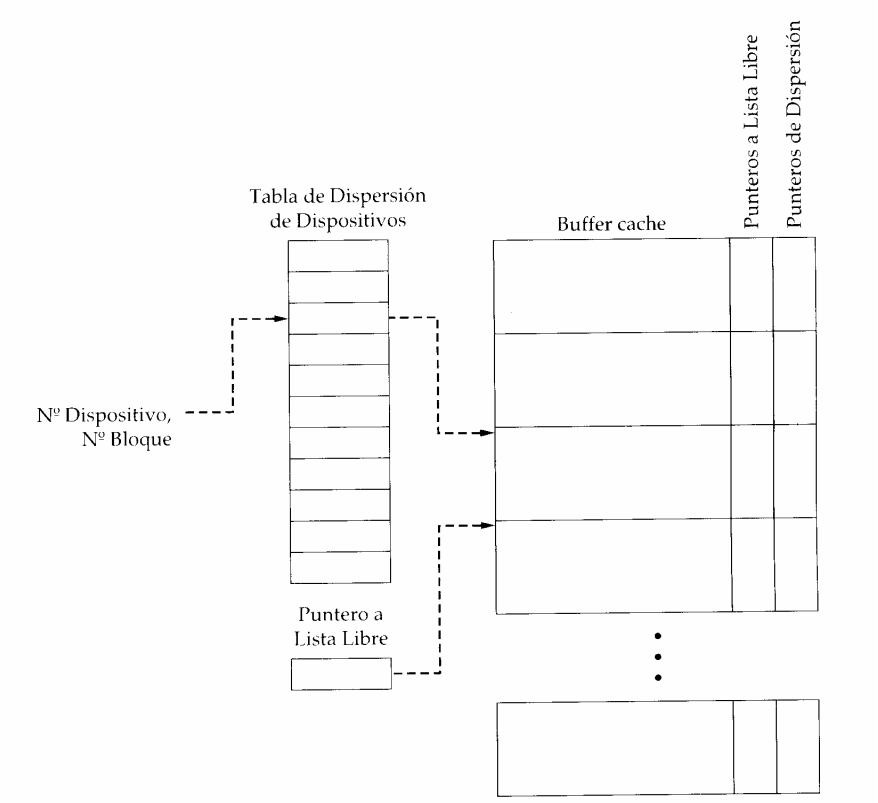


FIGURA 10.13 Organización de buffer caché en UN1X

Cola de caracteres

Los dispositivos de bloques, como los discos y las cintas, pueden ser tratados a través de la caché de buffers de una forma eficaz. Pero hay una forma diferente de almacenamiento intermedio, más adecuada para dispositivos de caracteres, como terminales e impresoras. El dispositivo de E/S escribe en una cola de caracteres, de la que lee el proceso o, también, el proceso escribe y el dispositivo lee de ella. En ambos casos se utilizará el modelo del productor/consumidor presentado en el capítulo 4. De esta manera, las colas de caracteres sólo podrán ser leídas una vez; a medida que se lee cada carácter, éste es destruido. Este mecanismo es distinto al de la caché de buffers, donde se puede leer varias veces y, por tanto, se sigue el modelo de los lectores/escritores (también discutido en el capítulo 4).

E/S no amortiguada

La E/S no amortiguada, que es un simple DMA entre el dispositivo y el espacio del proceso, es siempre el método más rápido de realizar E/S para un proceso. Los procesos que realizan

E/S no amortiguada quedan bloqueados en memoria principal y no pueden ser expulsados a disco. Esta condición reduce las oportunidades de expulsión inmovilizando parte de la memoria principal, reduciendo por tanto el rendimiento global del sistema. Además, el dispositivo de E/S se paraliza junto al proceso mientras dure la transferencia, quedando inasequible para otros procesos.

Dispositivos de UNIX

UNIX reconoce las cinco clases de dispositivos siguientes:

- Unidades de disco
- Unidades de cinta
- Terminales
- Líneas de comunicación
- Impresoras

La tabla 10.5 muestra los tipos de E/S a que se ajusta cada clase de dispositivo. Las unidades de disco son muy empleadas en UNIX, son dispositivos de bloques y ofrecen una productividad razonablemente alta. Por tanto, la E/S con estos dispositivos tiende a ser no amortiguada por una c-iche. Las unidades de cinta son funcionalmente similares a las de disco y emplean esquemas similares de E/S.

Como los terminales realizan un intercambio de caracteres relativamente lento, la E/S con ellos hace normalmente uso de las colas de caracteres. De forma similar, las líneas de comunicación requieren el procesamiento en serie de bytes de datos para entrada o salida y se manejan mejor mediante colas de caracteres. Por último, el tipo de E/S empleado para las impresoras depende generalmente de su velocidad. Las impresoras lentas emplean normalmente colas de caracteres, mientras que las rápidas pueden utilizar E/S no amortiguada. Se puede usar una caché de buffers para una impresora rápida. Sin embargo, como los datos dirigidos a una impresora nunca se van a utilizar de nuevo, no es necesaria la caché de buffers.

MVS

MVS fue diseñado para ofrecer un sistema de E/S estructurado en niveles que permitiera a los programadores ignorar los múltiples detalles de las operaciones de E/S, así como saltarse o detallar determinadas fases de cada operación. La figura 10.14 ofrece la estructura lógica de la E/S en MVS. En una secuencia típica de E/S entran en Juego los siguientes pasos:

Dispositivo	E/S sin buffer	Buffer cache	Colas de Caracteres
Unidad de disco	X	X	
Unidad de cinta	X	X	
Terminales			X
Líneas de comunicación			X
Impresoras	X		X

TABLA 10.5 E/S con dispositivos en UNIX

El programa de usuario da comienzo a una operación de E/S enviando una macroinstrucción OPEN a un dispositivo de E/S y solicitando después una entrada o una salida por medio de otra macro como GET, PUT, READ o WRITE. La macroinstrucción de E/S invoca a un servicio del sistema operativo conocido como **método de acceso.** El método de acceso interpreta el comando y determina los recursos del sistema que se necesitan. El usuario puede saltarse el método de acceso, pero esto haría que el programa de usuario tuviera que tratar con la operación de E/S con mucho mayor detalle y a un nivel de control más fino.

Los métodos de acceso de MVS se dividen en tres categorías: métodos de acceso convencionales, métodos de acceso de telecomunicación y métodos de acceso al almacenamiento virtual (VSAM). La tabla 10.6 resume los métodos de acceso disponibles en MVS. Con un método de acceso, el programa se aisla de los detalles de la E/S y sólo debe preocuparse de emplear el método apropiado para satisfacer sus necesidades.

Para solicitar el traslado de los datos, bien el método de acceso, bien el programa de usuario, presentan información sobre la operación al procesador EXCP (programa ejecutor de canales). El EXCP traduce esta información a un formato inteligible por el subsis-

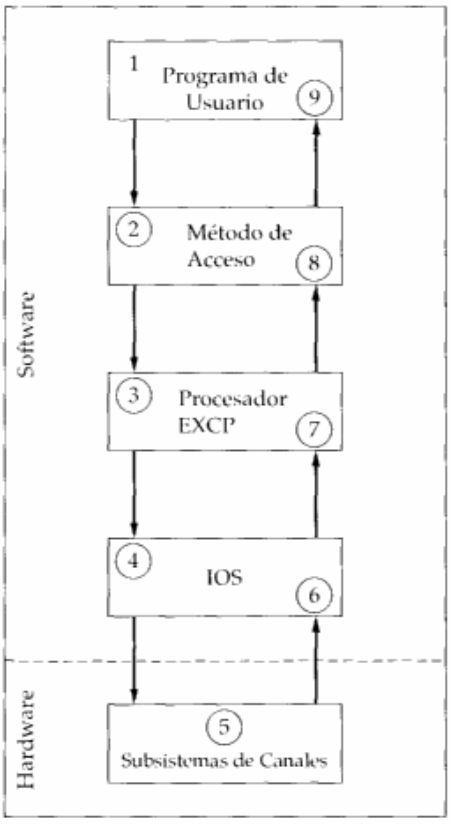


FIGURA 10.14 Servicios de E/S de MVS

TABLA 10.6 Métodos de Acceso de MVS

Métodos de Acceso Convencionales

Método de Acceso Secuencial Básico (BSAM)

Los registros de un archivo están organizados secuencialmente. BSAM ofrece la capacidad de leer y escribir sólo registros tísicos y sólo en secuencia.

Método de Acceso Secuencial con Colas (QSAM)

Los registros de un archivo están organizados secuencialmente. Con QSAM, los registros lógicos pueden agruparse y almacenarse en registros físicos mayores (bloques). QSAM maneja la agrupación y desagrupación de registros lógicos y se encarga de la F./S amortiguada.

Método de Acceso Directo Básico (BDAM)

BDAM permite la entrada y salida de bloques individuales especificando la pista física y el número del registro.

Método de Acceso Secuencial Indexado (ISAM)

Uno o más campos de un registro, llamados claves, identifican unívocamente al registro. Puede accederse a los registros proporcionando una clave directa o secuencialmente por el orden de la clave.

Método de Acceso Particionado Básico (BPAM)

El archivo se agrupa en conjuntos de registros independientes, llamados *miembros.* Todos los registros de un mismo miembro poseen las mismas propiedades, como el tamaño del registro lógico y el tamaño del bloque. En un directorio se guarda el nombre y la ubicación de cada miembro. BPAM mantiene el directorio y accede al mismo. Una vez que un miembro es ubicado por BPAM, se accede a los registros mediante BSAM o QSAM.

Métodos de Acceso al Almacenamiento Virtual

Método de Acceso al Almacenamiento Virtual (VSAM)

Este método de acceso está diseñado especialmente para aprovechar el hardware de memoria virtual y el software de memoria virtual de MVS. Ofrece tanto acceso directo como secuencia! y proporciona un rendimiento y flexibilidad mayores que otros métodos de acceso a archivos.

Métodos de Acceso de Telecomunicación

Método de Acceso Básico de Telecomunicación (BTAM)

BTAM proporciona una sencilla capacidad de transmisión de datos en forma de mensajes con terminales remotos.

Método de Acceso de Telecomunicación (TCAM)

Soporta una variedad más amplia de terminales que BTAM. Permite a las aplicaciones realizar su propio encaminamiento de mensajes, edición de mensajes y comprobación de errores.

Método de Acceso Virtual de Telecomunicación (VTAM)

VTAM es el método de acceso usado como soporte a la arquitectura de red de sistemas IBM (SNA). Proporciona una potente capacidad de manejo de terminales en el contexto de una arquitectura completa de comunicaciones.

tema de canales e invoca al supervisor de ES (IOS). Básicamente, EXCP es un programa que crea un bloque del supervisor de E/S (IOSB) para que el IOS lo utilice. El IOSB contiene las instrucciones para el IOS y las direcciones de memoria principal involucradas en la transferencia.

4. El IOS ubica la petición de E/S en la cola del dispositivo elegido y da inicio al subsistema de canales. El subsistema de canales se inicia con una orden que hace referencia

al programa del canal en memoria principal. La CPU queda entonces disponible para realizar otra labor hasta que el subsistema de canales indique que la operación de E/S ha terminado.

5. El subsistema de canales es un procesador separado que puede leer y ejecutar órdenes de canal o instrucciones en memoria principal. El subsistema elige el mejor camino para la transmisión de datos entre memoria principal y el dispositivo y controla el traslado de los datos. Cuando finaliza la E/S, el subsistema realiza una interrupción a la CPU).
6. El IOS evalúa la interrupción y devuelve el control al EXCP.
7. El EXCP actualiza varias tablas para indicar el resultado de la operación de E/S y pasa el control al distribuidor (*dispatcher*).
8. El distribuidor reactiva el método de acceso para responder a la terminación de la petición de E/S.
9. El método de acceso devuelve el control al programa de usuario, junto con alguna información de estado requerida.

Gran parte de la actividad de E/S implica solamente al subsistema de canales y no al procesador principal. De este modo, la arquitectura de E/S de MVS ofrece un servicio que es tan potente como eficaz.

10.7 RESUMEN

La interfaz del computador con el mundo exterior es la arquitectura de E/S. Esta arquitectura está diseñada para ofrecer un medio sistemático de controlar la interacción con el mundo exterior y proporcionar al sistema operativo la información que necesita para administrar la actividad de E/S de una manera eficaz.

Las funciones de E/S se dividen generalmente en un conjunto de niveles, donde los más bajos se encargan de los detalles cercanos a las funciones físicas a realizar y los superiores tratan con la E/S desde un punto de vista lógico y general. El resultado es que los cambios en los parámetros del hardware no afecten necesariamente a la mayor parte del software de E/S.

Un aspecto clave de la E/S es el empleo de buffers controlados por utilidades de E/S más que por los procesos de aplicación. El almacenamiento intermedio sirve para igualar las diferencias de velocidades internas del computador y las velocidades de los dispositivos de E/S. El uso de buffers también permite desacoplar las transferencias reales de E/S del espacio de direcciones del proceso de aplicación, lo que permite al sistema operativo una mayor flexibilidad en la realización de las funciones de gestión de memoria.

El área de la E/S que tiene un impacto mayor en el rendimiento global del sistema es la E/S a disco. Por consiguiente, se han realizado más investigaciones e invertido más esfuerzos de diseño en este punto que en cualquier otro de la E/S. Dos de los métodos usados más frecuentemente para mejorar el rendimiento de la E/S a disco son la planificación y la caché de disco.

En un momento dado puede haber una cola de peticiones de E/S al mismo disco. Es una labor de la planificación del disco el satisfacer estas peticiones de forma que se minimice el

tiempo de búsqueda mecánica del disco y, por tanto, se mejore el rendimiento. Aquí entran en juego la disposición física de las peticiones pendientes, así como consideraciones sobre la cercanía de las mismas.

Una caché de disco es un buffer, normalmente en memoria principal, que funciona como una caché de bloques de disco entre la memoria del disco y el resto de la memoria principal. Por el principio de cercanía, el empleo de una caché de disco debe reducir sustancialmente el número de transferencias de E/S de bloques entre la memoria principal y el disco.

10.8 LECTURAS RECOMENDADAS

Un estudio valioso sobre la tecnología de los discos es [SIER90]. |WIED87] contiene una discusión excelente sobre aspectos de rendimiento del disco, incluyendo los relativos a la planificación. Pueden encontrarse discusiones más generales de la E/S en la mayoría de los libros de arquitectura de computadores, como |STAL93a] y [HENN90].

HENN90 HENNESSY, I. y PATTERSON, D. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, San Mateo, CA, 1990.

SIER90 SIERRA, H. *An Introduction to Direct Acccess Storage Devices.* Academic Press, Boston, MA, 1990.

STAL93a STALLINGS, W. *Computer Organization and Architecture,* 3ª ed. Macmillan, Nueva York, 1993.

WE1D87 WEIDERHOLD, G. *File Organization for Database Design.* McGraw-Hill, Nueva York, 1987.

10.9 PROBLEMAS

10.1 Realice el mismo tipo de análisis de la tabla 10.3 para la siguiente secuencia de peticiones de pistas: 27, 129. 110, 186, 147. 41, 10, 64. 120. Supóngase que la cabeza del disco está ubicada inicialmente sobre la pista 100 y se está moviendo en direcciones decrecientes de números de pista. Haga el mismo análisis, pero suponga ahora que la cabeza del disco está moviéndose en direcciones crecientes de números de pista.

10.2 Considérese un disco de *N* pistas numeradas de O a (N - I) y supóngase que los sectores pedidos están distribuidos aleatoriamente y de manera uniforme por todo el disco. Se desea calcular el número medio de pistas recorridas en una búsqueda.

a) Primero, calcular la probabilidad de una búsqueda de longitud j cuando la cabeza está ubicada sobre la pista t. Indicación: Es cuestión de determinar el número total de combinaciones, identificando que todas las posiciones de pista destino de la búsqueda son igualmente probables.

b) Seguidamente, calcular la probabilidad de una búsqueda de longitud *K.* Indicación:
Esto implica sumar todas las posibles combinaciones de movimientos de *K* pistas.

c) Calcular el número medio de pistas atravesadas por una búsqueda, usando la siguiente fórmula para el valor esperado:

$$E[x] = \sum_{i=0}^{N} i \times P_r[x = i]$$

d) Para valores grandes de *N*. mostrar el número medio de pistas atravesadas por una búsqueda de *N*/3.

10.3 En el capítulo 5 se introduce el concepto de almacenamiento intermedio de páginas, que consiste, simplemente, en una estrategia de caché para las páginas de memoria virtual. En un sistema que emplee almacenamiento intermedio de páginas, ¿haría falta una caché de disco como la descrita en este capítulo? ¿Y al contrario?

10.4 Se propone la ecuación siguiente, tanto para la memoria caché como para la caché de disco:

$$T = T + M \times T_D$$

Generalizar esta ecuación para una jerarquía de memoria de N niveles en lugar de sólo dos.

10.5 Para el algoritmo de reemplazo en función de la frecuencia, definir F_{nueva} F_{MEDIA}, y $F_{antigua}$, como la fracción de caché comprendida por las secciones nueva, inedia y antigua, respectivamente. Evidentemente, F_{nueva}, + F_{media}, + $F_{antigua}$, = 1. Caracterizar dicha política cuando se cumpla:

a) $F_{antigua} = 1 - F_{nueva}$

b) $F_{antigua}$ = l/(tamaño de caché)

10.6 ¿Cuál es la velocidad de transferencia de una unidad de cinta magnética de 9 pistas cuya velocidad es de 120 pulgadas por segundo y cuya densidad es de 1600 bits lineales por pulgada?

10.7 Supóngase un rollo de cinta de 2400 pies con un salto entre registros de 0.6 pulgadas, donde la cinta se detiene a medio camino entre las lecturas. La tasa de incrementos o disminuciones de la velocidad de la cinta durante estos saltos es lineal. El resto de características de la cinta son las mismas que en el problema 10.6. Los datos de la cinta se organizan en registros físicos que contienen un número fijo de unidades definidas por el usuario, llamadas registros lógicos.

a) ¿Cuánto se tardará en leer una cinta completa con registros lógicos de 120 bytes distribuidos en grupos de 10 por cada registro físico?

b) ¿Y si son grupos de 30?

c) ¿Cuántos registros lógicos podrá guardar la cinta para los factores de agrupación de cada una de las situaciones anteriores?

d) ¿Cuál es la velocidad efectiva de transferencia global para cada factor de agrupación de (a) y (b)?

e) ¿Cuál es la capacidad de la cinta?

10.8 Calcular la cantidad de espacio en disco (en sectores, pistas y superficies) necesaria para almacenar los registros lógicos del problema 10.7b si el disco es fijo con 512 bytes por sector, 96 sectores por pista, 110 pistas por superficie y ocho superficie útiles. Ignorar los registros de cabecera de archivos y los índices de pista, suponiendo que los registros no pueden extenderse a dos sectores.

10.9 Considerar el sistema de disco descrito en el problema 10.8 y suponer que el disco gira a 360 rpm. El procesador lee un sector del disco mediante E/S dirigida por interrupciones, con una interrupción por cada byte. Si se tarda 2,5 microsegundos en procesar cada interrupción, ¿que porcentaje del tiempo gastará el procesador en gestionar la E/S (no considerar el tiempo de búsqueda)?

10.10 Repetir el problema 10.9 usando DMA y suponiendo que se produce una interrupción por sector.

10.11 Un computador de 32 bits posee dos canales selectores y un canal multiplexor. Cada canal selector da soporte a dos discos magnéticos y dos unidades de cinta magnética. El canal multiplexor tiene conectadas dos impresoras, dos lectores de tarjetas y diez terminales VDT. Supónganse las siguientes velocidades de transferencia:

Unidad de disco:	800 Kb/sg
Unidad de cinta magnética:	200 Kb/sg
Impresora:	6,6 Kb/sg
Lector de tarjetas:	1,2 Kb/sg
VDT:	1 Kb/sg

Estimar la velocidad máxima agregada de transferencia E/S en este sistema.

CAPITULO 11

Gestión de archivos

En la mayoría de las aplicaciones, el archivo es el elemento central. Cualquiera que sea la finalidad de la aplicación, implicará la generación y uso de información. Con la excepción de las aplicaciones de tiempo real y otras aplicaciones especializadas, la entrada a la aplicación se hace por medio de archivos y, casi en todas las aplicaciones, la salida se guarda en archivos para su almacenamiento a largo plazo y para accesos posteriores por parte del usuario y de otros programas.

Los archivos tienen vida fuera de cualquier aplicación individual que los utilice para entrada y salida. Los usuarios desean poder acceder a los archivos, guardarlos y mantener la integridad de su contenido. Como ayuda a estos objetivos, virtualmente todos los sistemas de computadores proporcionan sistemas específicos de gestión de ficheros. Normalmente, cada sistema dispone de programas de utilidad que se ejecutan como aplicaciones privilegiadas. Sin embargo, un sistema de gestión de archivos necesita como mínimo algunos servicios especiales del sistema operativo. Como máximo, el sistema de gestión de archivos por completo se considerará parte del sistema operativo. De este modo, es apropiado considerar por lo menos en el libro los elementos básicos de la gestión de archivos.

En este capítulo se van a examinar estos elementos básicos. Se comenzará con una visión general de los archivos y de los sistemas de gestión de archivos. A continuación se sigue con una visión de las alternativas de organización de los archivos. Aunque la organización de los archivos se sale generalmente del alcance del sistema operativo, es esencial tener una compresión general de las alternativas con el objeto de apreciar algunos aspectos de diseño implicados en la gestión de archivos. El resto del capítulo se dedica a otros aspectos de la gestión de archivos, como los que se perfilan en la sección 11.1.

11.1 INTRODUCCIÓN

Archivos

Cuando se habla de archivos, se utilizan cuatro términos comunes:

• Campo

• Registro
• Archivo
• Base de datos

Un **campo** es el elemento de datos básico. Un campo individual contiene un valor único, como el apellido de un empleado, una fecha o el valor leído por un sensor. Se caracteriza por su longitud y por el tipo de datos (por ejemplo, una cadena ASCI1 o un número decimal). El contenido de un campo es proporcionado por un usuario o por un programa. Dependiendo del diseño del archivo, los campos pueden ser de tamaño fijo o variable. En el último caso, el campo consta a menudo de dos o tres subcampos: el valor real a almacenar, el nombre del campo y, en algunos casos, la longitud del campo. En otros casos de campos de longitud variable, la longitud del campo se indica usando símbolos de referencia especiales entre los campos. La mayoría de los sistemas de archivos no soportan campos de longitud variable.

Un **registro** es una colección de campos relacionados que pueden tratarse como una unidad en algunos programas de aplicación. Por ejemplo, un registro "empleado" contendría campos tales como nombre, número de DNI, categoría salarial, fecha de contratación, etc. También dependiendo del diseño, los registros pueden ser de longitud fija o variable. Un registro es de longitud variable si algunos de sus campos son de longitud variable o si el número de campos puede variar. En el último caso, cada campo se acompaña normalmente de un nombre de campo. En cualquier caso. el registro entero incluye un campo de longitud.

Un **archivo** es una colección de registros similares. Los usuarios y las aplicaciones tratan al archivo como una entidad única y se refieren a él por un nombre. Los archivos tienen nombres únicos y pueden crearse y borrarse. Las restricciones al control de accesos suelen aplicarse a nivel de archivo. Es decir, en un sistema compartido, el acceso de los usuarios y los programas se garantiza o deniega a archivos completos. En sistemas mas sofisticados, dicho control se aplica a los registros o incluso a los campos.

Una **base de datos** es una colección de datos relacionados. El aspecto fundamental de una base de datos es que las relaciones que existen entre los elementos de datos son explícitas y que la base de datos está diseñada para ser usada por varias aplicaciones diferentes. Una base de datos puede contener toda la información relativa a una organización o proyecto, como un estudio científico o de mercado. La base de dalos consta de una o más clases de archivos. Generalmente, se dispone de un sistema de gestión de bases de datos separado, aunque dichos sistemas puedan hacer uso de algunos programas de gestión de archivos.

Los usuarios y las aplicaciones hacen uso de los archivos. Las operaciones típicas que deben aportarse incluyen las siguientes |LIVA90|:

• *Recuperar_Todo:* Recuperar todos los registros de un archivo. Esta operación será necesaria para una aplicación que deba procesar toda la información del archivo de una sola ve/. Por ejemplo, una aplicación que produzca un resumen de la información del archivo necesitaría recuperar todos los registros. Esta operación se considera equivalente al término *proceso secuencial.* porque se accede a todos los registros secuencialmente.
• *Recuperar _U 110:* Esta operación implica la recuperación de un único registro. Las aplicaciones interactivas de transacciones necesitan esta operación.
• *Recuperar _Siguiente:* Esta operación implica la recuperación del registro "siguiente", según una secuencia lógica, al recuperado hace menos tiempo. Algunas

aplicaciones interactivas, como el rellenado de formularios, pueden necesitar una operación como ésta. Una programa que realice búsquedas puede usar también esta operación.

- *Recuperar_Previo:* Similar a la recuperación del siguiente, pero en este caso se recupera el registro anterior al que se está accediendo en el momento actual.
- *Insertar_Uno:* Inserta un registro nuevo en el archivo. Puede ser necesario que el registro nuevo deba ajustarse a una posición determinada para mantener el secuenciamiento del archivo.
- *Borrar_Uno:* Suprimir un registro existente. Se puede necesitar actualizar ciertos enlaces u otras estructuras de datos para mantener el secuenciamiento del archivo.
- *Actualizar_Uno:* Recuperar un registro, actualizar uno o más de sus campos y volver a escribir el registro actualizado en el archivo. De nuevo, puede ser necesario mantener el secuenciamiento al usar esta operación. Si la longitud del registro ha cambiado, la operación de actualización es, en general, más difícil que si se mantiene la longitud.
- *Recuperar_Varios:* Recuperar un número determinado de registros. Por ejemplo, una aplicación o un usuario puede desear recuperar todos los registros que satisfagan unos ciertos criterios.

La naturaleza de las operaciones que se realizan más frecuentemente sobre un archivo influye en la manera en que se éste se organiza, como se discute en la sección 11.2.

Sistemas de Gestión de Archivos

Un sistema de gestión de archivos es aquel sistema software que proporciona a los usuarios y aplicaciones unos servicios relativos al empleo de archivos. Normalmente, la única forma en que un usuario o aplicación puede acceder a los archivos es mediante el sistema de gestión de archivos. Esto acaba con la necesidad, para el usuario o programador, de desarrollar software de propósito específico para cada aplicación y proporciona al sistema un medio de controlar su ventaja más importante. [GROS86] sugiere los siguientes objetivos para un sistema de gestión de archivos:

- Cumplir con las necesidades de gestión de datos y con los requisitos del usuario, que incluyen el almacenamiento de datos y la capacidad de realizar las operaciones antes expuestas.
- Garantizar, en la medida de lo posible, que los datos de los archivos son válidos.
- Optimizar el rendimiento, tanto desde el punto de vista del sistema, en términos de productividad global, como desde el punto de vista del usuario, en términos de tiempo de respuesta.
- Ofrecer soporte de E/S para la variedad de tipos de dispositivos de almacenamiento.
- Minimizar o eliminar la posibilidad de pérdida o destrucción de datos.
- Ofrecer un conjunto estándar de rutinas de interfaz de E/S.
- Proporcionar soporte de E/S para múltiples usuarios en el caso de sistemas multiusuario.

Con respecto al primer punto, sobre el cumplimiento de los requisitos del usuario, el alcance de dichos requisitos depende de la variedad de aplicaciones y el entorno en que el sistema va a utilizarse. Para un sistema interactivo, de propósito general, los siguientes requisitos mínimos deben cumplirse [WATS70|:

Figura 11.1 Arquitectura del software del sistema de archivos [GROS86]

1. Cada usuario debe ser capaz de crear, borrar y cambiar los archivos.
2. Cada usuario puede tener acceso controlado a los archivos de otros usuarios.
3. Cada usuario puede controlar qué tipos de acceso estarán permitidos a sus archivos.
4. Cada usuario debe poder reestructurar sus archivos de manera adecuada al problema.
5. Cada usuario debe ser capaz de mover datos entre los archivos.
6. Cada usuario debe ser capaz de guardar copia de reserva y recuperar sus archivos en caso de que haya desperfectos.
7. Cada usuario debe ser capaz de acceder a sus archivos mediante un nombre simbólico.

Estos objetivos y requisitos deben tenerse en cuenta en toda nuestra discusión de los sistemas de gestión de archivos.

Arquitectura de los Sistemas de Archivos

Una manera de hacerse una idea del alcance de la gestión de archivos es observar una representación de una organización típica del software, como se muestra en la figura 11.1. Por supuesto, sistemas diferentes se organizan de forma diferente, pero esta organización es razonablemente representativa. En el nivel más bajo, los **manejadores de dispositivos** *(device drivers)* se comunican directamente con los dispositivos periféricos o sus controladores o canales. Cada manejador de dispositivo es responsable de comenzar las operaciones de E/S en un dispositivo y procesar la terminación de una petición de E/S. En operaciones con archivos, los dispositivos típicos controlados son discos y unidades de cinta. Los manejadores de dispositivos son considerados generalmente como parte del sistema operativo.

El siguiente nivel es conocido con el nombre de **sistema de archivos básico** o nivel de E/S **física,** que constituye la interfaz primaria con el entorno exterior al computador. Este ni-

vel trata con bloques de datos que son intercambiados con sistemas de disco o cinta. De este modo, se preocupa de ubicar dichos bloques en el dispositivo de almacenamiento secundario y del almacenamiento intermedio de los mismos en memoria principal. Este nivel no comprenderá el contenido de los datos o la estructura de los archivos implicados. El sistema de archivos básico se considera a menudo parte del sistema operativo.

El **supervisor básico de E/S** es el responsable de la iniciación y terminación de toda la E/S con archivos. En este nivel se mantienen unas estructuras de control que se encargan de la E/S con los dispositivos, la planificación y el estado de los archivos. El supervisor básico de E/S se ocupa de la selección del dispositivo donde va a realizarse la E/S con los archivos, dependiendo del archivo seleccionado. También se ocupa de la planificación de los accesos a disco y cinta para optimizar el rendimiento. En este nivel se asignan los buffers de E/S y se reserva la memoria secundaria. El supervisor básico de E/S es parte del sistema operativo.

La **E/S lógica** es la parte del sistema de archivos que permite a usuarios y aplicaciones acceder a los registros. Así, mientras el sistema de archivos básico trabaja con bloques de datos, el módulo de E/S lógica lo hace con registros. La E/S lógica ofrece una capacidad de E/S de registros de propósito general y mantiene unos datos básicos sobre los archivos.

Finalmente, el nivel del sistema de archivo más cercano al usuario es, generalmente, el **método de acceso**. Cada método proporciona una interfaz estándar entre las aplicaciones y los sistemas de archivos y dispositivos que guarden datos. Los diferentes métodos de acceso reflejan las distintas estructuras de archivos y las formas diferentes de acceder y procesar los datos. Algunos métodos de acceso muy conocidos se muestran en la figura 11.1 y se describen brevemente en la sección 11.2.

Funciones de la Gestión de Archivos

Otra manera de contemplar las funciones de un sistema de archivos es la de la figura 11.2. Examínese este diagrama de izquierda a derecha. Los usuarios y los programas de aplicación interactúan con el sistema de archivos por medio de órdenes de creación y eliminación de archivos y realizando operaciones sobre los archivos. Antes de realizar ninguna operación, el sistema de archivos debe identificar y ubicar el archivo en cuestión. Esto requiere el uso de algún tipo de directorio que describa la ubicación de todos los archivos y sus atributos. Además, la mayoría de los sistemas compartidos aplican algún control de acceso de los usuarios: Sólo a los usuarios autorizados se les permite acceder de una forma determinada a determinados archivos. Las operaciones básicas que un usuario o aplicación puede realizar sobre un archivo tienen lugar a nivel de registros. El usuario o la aplicación contempla al archivo con una estructura que organiza los registros, como una estructura secuencia! (por ejemplo, registros personales almacenados alfabéticamente por apellido). De este modo, para traducir las órdenes del usuario a órdenes específicas de manipulación de archivos, debe emplearse el método de acceso apropiado para esta estructura de archivo.

Mientras que los usuarios y las aplicaciones se ocupan de los registros, la E/S se lleva a cabo con bloques. De esta forma, los registros de un archivo deben bloquearse para salida y desbloquearse tras la entrada. Para respaldar la E/S de bloques se necesitan varias funciones. Debe gestionarse el almacenamiento secundario. Esto incluye la asignación de archivos a los bloques libres de memoria secundaria y la gestión del espacio libre, de manera que se conozca qué bloques están disponibles si se crean archivos nuevos o crecen los archivos existentes. Además, deben planificarse las peticiones de bloques individuales; esta

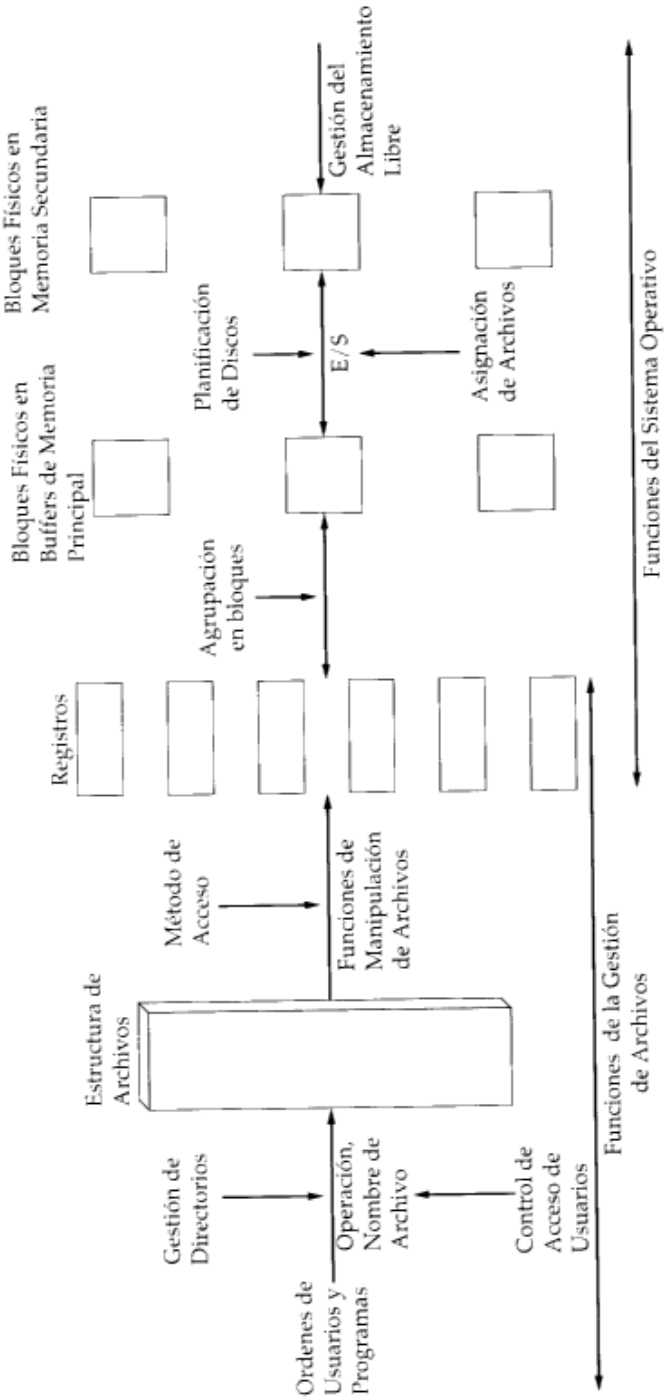


FIGURA 11.2 Elementos de la Gestión de Archivos

cuestión fue tratada en el capítulo 10. Tanto la planificación del disco como la asignación de archivos se preocupan de optimizar el rendimiento. Como podía esperarse, estas funciones necesitan considerarse en conjunto. Además, la optimización dependerá de la estructura de los archivos y de los tipos de acceso. De acuerdo a esto, el desarrollo de un sistema de gestión de archivos óptimo desde el punto de vista del rendimiento es una tarea sumamente complicada.

La figura 11.2 propone una división entre lo que debe considerarse asunto del sistema de gestión de archivos como utilidad separada del sistema y los asuntos del sistema operativo, siendo el punto de intersección el proceso de registros. Esta división es arbitraria y en distintos sistemas se adoptan métodos diferentes. Como se ha mencionado, la planificación del disco fue tratada en el capítulo 10. En el resto de este capítulo se verán algunas cuestiones de diseño propuestas en la figura 11.2. Se comenzará por una discusión de la organización de archivos y los métodos de acceso. Aunque este punto se escapa del alcance de lo que generalmente se considera preocupación del sistema operativo, es imposible abordar las otras cuestiones de diseño relativas a los archivos sin tener una apreciación de la organización y el acceso a los archivos. Seguidamente se va a ver el concepto de directorios de archivos. gestionados a menudo por el sistema operativo, en nombre del sistema de gestión de archivos. Los puntos restantes tratan sobre los aspectos físicos de E/S de la gestión de archivos y son tratados estrictamente como aspectos de diseño de sistemas operativos. Lina cuestión clave es la forma en que los registros lógicos se organizan en bloques físicos. Finalmente se consideran cuestiones relacionadas con la asignación de archivos en la memoria secundaria y la gestión del espacio libre.

11.2 ORGANIZACIÓN Y ACCESO A ARCHIVOS

Un archivo consiste en una colección de registros. Uno de los elementos clave de diseño del sistema de archivos es la forma en que estos registros se organizan o estructuran. En esta sección se emplea el término *organización de archivos* para referirse a la estructuración lógica de los registros determinada por la forma en que se accede a ellos. La organización física de un archivo en memoria secundaria depende de la estrategia de agrupación y de la estrategia de asignación de archivos, cuestiones tratadas más tarde en este mismo capítulo.

Diversos criterios son importantes en la elección de una organización de archivos:

- Acceso rápido para la recuperación eficaz de información
- Facilidad de actualización para ayudar a mantener la información al día
- Economía de almacenamiento para reducir costes
- Mantenimiento sencillo para reducir costes y la posibilidad de errores
- Fiabilidad para asegurar la confianza en los datos

La prioridad relativa de estos criterios dependerá de las aplicaciones que usará el archivo. Por ejemplo, si un archivo va a procesarse solamente por lotes *(batch),* accediendo cada vez a todos los registros, entonces el acceso rápido para la recuperación de un único registro es una preocupación mínima. Un archivo guardado en CD-ROM nunca será actualizado y la facilidad de actualización no se considera.

El número de organizaciones alternativas de archivos que se han implementado o propuesto es inmanejable, incluso para un libro dedicado a los sistemas de archivos. En este breve estudio se esbozan cinco organizaciones fundamentales. La mayor parte de las estructuras empleadas en los sistemas reales se encuadran en una de estas categorías o puede im-plementarse con una combinación de estas organizaciones. Las cinco organizaciones, representadas en la figura 11.4, son las siguientes :

- Pilas
- Archivos secuenciales
- Archivos secuenciales indexados
- Archivos indexados
- Archivos directos o de dispersión *(hash)* La tabla 11.1 resume las cuestiones de rendimiento relativo de estas cinco organizaciones.

Pilas

La forma menos complicada de organización de archivos puede denominarse *pila.* Los datos se recogen en el orden en que llegan. Cada registro consta de una ráfaga de datos. La finalidad de la pila es simplemente acumular una masa de datos y guardarla. Los registros pueden tener campos diferentes o pueden tener campos similares en un orden distinto. Así, cada campo debe ser autodescriptivo, incluyendo tanto un campo de nombre como el valor. La longitud de cada campo debe indicarse implícitamente mediante delimitadores, explicitados como un subcampo más o conocidos por omisión para el tipo de campo.

Como no hay una estructura para el archivo de la pila, el acceso a registros se hace por búsqueda exhaustiva. Es decir, si se quiere encontrar un registro que contiene un campo particular con un valor determinado, es necesario examinar cada registro de la pila hasta que se encuentre el registro deseado o se haya recorrido el archivo completo. Si se quieren encontrar todos los registros que contienen un campo particular o que tienen un valor determinado para ese campo, debe buscarse en el archivo entero.

Los archivos de pila se aplican cuando los datos se recogen y almacenan antes de procesarlos o cuando no son fáciles de organizar. Esta clase de archivos aprovecha bien el espacio cuando los datos almacenados varían en tamaño y estructura. Los archivos de pila son muy adecuados para búsquedas exhaustivas y son fáciles de actualizar. Sin embargo, fuera de estos usos limitados, este tipo de archivos no se adapta a la mayoría de las aplicaciones.

Archivos Secuenciales

La forma más común de estructura de archivo es el archivo secuencia!. En esta clase de archivos se emplea un formato fijo para los registros. Todos los registros son de la misma longitud y constan del mismo número de campos de tamaño fijo en un orden determinado. Como se conocen la longitud y la posición de cada campo, sólo se necesita almacenar los valores de cada campo; el nombre del campo y la longitud de cada uno son atributos de la estructura del archivo.

Un campo particular, generalmente el primero de cada registro, es conocido como el **campo clave.** El campo clave identifica unívocamente al registro; así, los valores de la clave para registros diferentes son siempre diferentes. Además, los registros se almacenan en secuencia por la clave: orden alfabético para una clave de texto y orden numérico para una clave numérica.

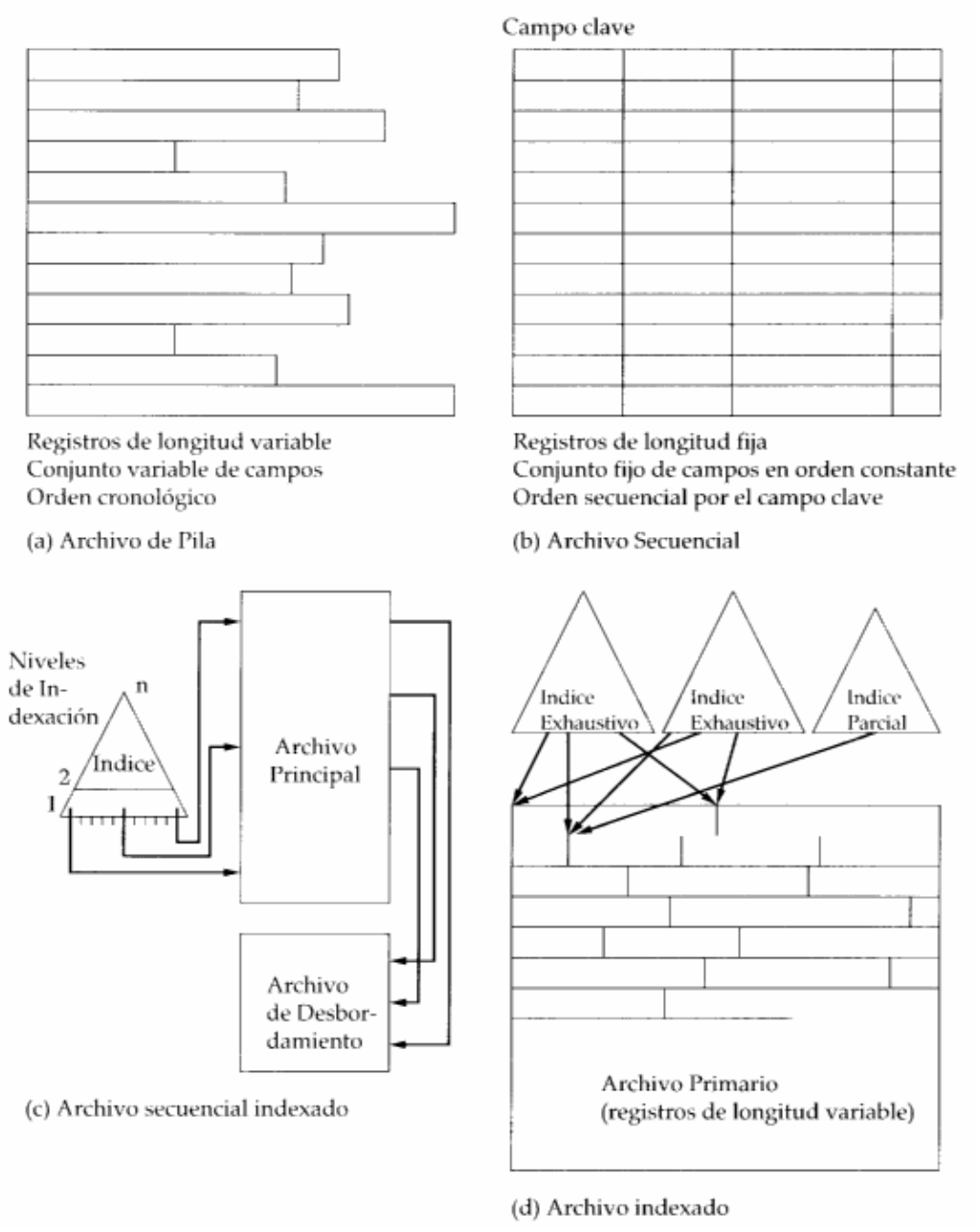


FIGURA 11.3 Organizaciones comunes de archivos

Los archivos secuenciales se utilizan normalmente en aplicaciones de proceso por lotes y, generalmente, son óptimos para dichas aplicaciones si se procesan todos los registros (por ejemplo, aplicaciones de facturación o nóminas). La organización secuencial de archivos es la única que se puede guardar tanto en cinta como en disco.

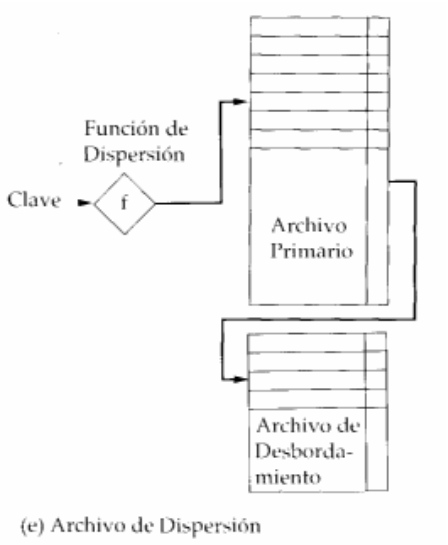


FIGURA 11.3 (continuación)

Para las aplicaciones interactivas que incluyen peticiones o actualizaciones de registros individuales, los archivos secuenciales ofrecen un rendimiento pobre. El acceso requiere la búsqueda secuencial en el archivo de una correspondencia con la clave. Si el archivo entero o una gran parte pueden traerse a memoria principal de una sola vez, es posible aplicar técnicas de búsqueda más eficientes. Sin embargo, al acceder a un registro de un archivo secuencial grande, se produce un proceso adicional y un retardo considerable. Las adiciones al archivo también presentan problemas. Normalmente, un archivo secuencial se almacena en bloques, en un orden secuencial simple de los registros. Es decir, la organización física del archivo en una cinta o disco se corresponde exactamente con la organización lógica del archivo. En este caso, el procedimiento habitual es ubicar los nuevos registros en un archivo de pila separado, llamado *archivo de registro (log file) o archivo de transacciones.* Periódicamente, se realiza una actualización por lotes que mezcla el archivo de registro con el archivo maestro para producir un nuevo archivo en secuencia correcta de claves.

Una alternativa es organizar físicamente el archivo secuencial como una lista enlazada. En cada bloque físico se almacena uno o más registros. Cada bloque del disco contiene un puntero al siguiente bloque. La inserción de registros nuevos conlleva la manipulación de punteros, pero no requiere que los nuevos registros ocupen una posición particular de bloque físico. De este modo, se obtiene una ventaja más al coste del procesamiento adicional.

Archivos Secuenciales Indexados

El método más popular para solventar las desventajas de los archivos secuenciales es el archivo secuencial indexado. Los archivos secuenciales indexados mantienen las características básicas de los archivos secuenciales. Los registros se organizan en una secuen-

TABLA 11.1 Grados de rendimiento para cinco métodos básicos de archivo [WIED87]

	Aprovechamiento del Espacio Libre		Actualización		Recuperación		
Método de Archivo	Tamaño de Registro Variable	Tamaño de Registro Fijo	Tamaño de Registro Igual	Tamaño de Registro Mayor	Registro Único	Subconjunto	Subconjunto Exhaustivo
Pila	A	B	A	E	E	D	B
Secuencial	F	A	D	F	F	D	A
Secuencial Indexado	F	B	B	D	B	D	B
Indexado	B	C	C	C	A	B	D
Dispersión	F	B	B	F	B	F	E

A = Excelente, se ajusta bien a este propósito
B = Bueno
C = Adecuado
D = Requiere un esfuerzo extra
E = Posible con un esfuerzo extremo
F = No razonable para este propósito

$\approx O(r)$
$\approx O(o \times r)$
$\approx O(r \log n)$
$\approx O(n)$
$\approx O(r \times n)$
$\approx O(n^2)$

donde
r = tamaño del resultado
o = número de registros que se desbordan
n = número de registros en el archivo

cia basada en un campo clave, pero se añaden dos características nuevas: un índice del archivo para soportar los accesos aleatorios y un archivo de desbordamiento *(overflow).* El índice proporciona una capacidad de búsqueda para llegar rápidamente a las proximidades de un registro deseado. El archivo de desbordamiento es similar al archivo de registro usado en un archivo secuencial, pero está integrado de forma que los registros del archivo de desbordamiento se ubican en la dirección de un puntero desde su registro precedente.

La estructura secuencial indexada más simple utiliza un único nivel de indexación. El índice en este caso es un archivo secuencial simple. Cada registro del archivo índice consta de dos campos: un campo clave, que es el mismo que el campo clave del archivo principal y un puntero al archivo principal. Para encontrar un campo específico, se busca en el índice hasta encontrar el valor mayor de la clave que es igual o precede al valor deseado de la clave. La búsqueda continúa en el archivo principal a partir de la posición indicada por el puntero.

Para comprobar la eficacia de este método, considérese un archivo secuencial con 1 millón de registros. La búsqueda de un valor particular de la clave necesitará, por término medio, 500.000 accesos a registros. Ahora supóngase que se construye un índice que contiene 1000 entradas, con las claves del índice más o menos uniformemente distribuidas a lo largo del archivo principal. Para encontrar el registro se tardará ahora, por término medio, 500 accesos al archivo de índice, seguidos de 500 accesos al archivo principal. El tamaño medio de la búsqueda se reduce de 500.000 a 1000.

Añadir datos al archivo se maneja de la forma siguiente. Cada registro del archivo principal contiene un campo adicional no visible para la aplicación, que es un puntero al archivo de desbordamiento. Cuando se inserta un nuevo registro al archivo, se añade al archivo de desbordamiento. El registro del archivo principal que precede inmediatamente al nuevo registro, según la secuencia lógica, se actualiza con un puntero al registro nuevo en el archivo de desbordamiento. Si el registro inmediatamente anterior está también en el archivo de desbordamiento, se actualizará el puntero en dicho registro. Al igual que en un archivo secuencia!, el archivo secuencial indexado se combina de vez en cuando con el archivo de desbordamiento en un tratamiento por lotes.

Los archivos secuenciales indexados reducen enormemente el tiempo necesario para acceder a un solo registro sin sacrificar la naturaleza secuencia! del archivo. Para procesar secuencialmente el archivo completo, los registros del archivo principal se procesarán en secuencia hasta encontrar un puntero al archivo de desbordamiento. El acceso continúa en el archivo de desbordamiento hasta que se encuentre un puntero nulo, momento en que se reanuda el acceso donde se abandonó en el archivo principal.

Para ofrecer una eficacia mayor aún en el acceso, se pueden usar múltiples niveles de indexación. De este modo, el nivel inferior del archivo de índice se trata como un archivo secuencial, creándose un archivo de índice de un nivel superior para el archivo. Considérese de nuevo un archivo con 1 millón de registros. Se construye un índice de nivel inferior con 10.000 entradas. Puede entonces construirse un índice de nivel superior de 100 entradas, definido sobre el de nivel inferior. La búsqueda comienza en el índice superior (longitud media = 50 accesos) para hallar un punto de entrada al índice inferior. Entonces se busca en este índice (longitud media = 50) para encontrar un punto de entrada al archivo principal, desde donde se sigue buscando (longitud media = 50). En total, la longitud media de la búsqueda se ha visto reducida de 500.000 a 1000 y después a 150.

Archivos Indexados

Los archivos secuenciales indexados conservan una de las limitaciones de los archivos secuenciales: La eficacia en el procesamiento se limita al basado en un único campo del archivo. Cuando es necesario buscar un registro basándose en algún otro atributo distinto del campo clave, ambas formas de archivo secuencia! no son adecuadas. En algunas aplicaciones, esta flexibilidad es deseable.

Para alcanzar esta flexibilidad, se necesita una estructura que utilice múltiples índices, uno para cada tipo de campo que pueda ser objeto de la búsqueda. En los archivos generales indexados, se abandonan los conceptos de secuencialidad y clave única. Los registros se acceden sólo a través de sus índices. El resultado es que no hay ahora restricción a la ubicación de los registros, en tanto que al menos un índice contiene un puntero a cada registro. Además, pueden emplearse registros de longitud variable.

Se suelen utilizar dos tipos de índices. Un índice exhaustivo contiene una entrada para cada registro del archivo principal. El índice se organiza en sí mismo como un archivo secuencial, para facilidad de la búsqueda. Otro índice parcial contendrá entradas a los registros donde esté el campo de interés. Con registros de longitud variable, algunos registros no contendrán todos los campos. Cuando se añade un registro al archivo principal, todos los archivos de índice deben actualizarse.

Los archivos indexados son muy usados en aplicaciones donde es crítica la oportunidad de la información y donde los datos son rara vez procesados de forma exhaustiva. Algunos ejemplos son los sistemas de reserva de líneas aéreas y los sistemas de control de inventario.

Archivos Directos o de Dispersión

Los archivos directos o de dispersión explotan la capacidad de los discos para acceder directamente a cualquier bloque de dirección conocida. Como en los archivos secuenciales y secuenciales indexados, se requiere un campo clave en cada registro. Sin embargo, aquí no hay concepto de ordenación secuencial.

El archivo directo hace uso de las técnicas de dispersión sobre el valor de la clave. Esta técnica se explicó en el Apéndice 7A. La figura 7.27b muestra el tipo de organización dispersa con el archivo de desbordamiento que se usa normalmente en un archivo de dispersión.

Los archivos directos son usados a menudo donde se necesita un acceso muy rápido, donde se usan registros de longitud fija y donde siempre se accede a los registros de una vez. Algunos ejemplos son las guías telefónicas, tablas de precios, horarios y listas de nombres.

11.3 DIRECTORIOS DE ARCHIVOS

Contenido

Asociado con cualquier sistema de gestión de archivos o cualquier colección de archivos suele haber un directorio de archivos. El directorio contiene información sobre los archivos, incluyendo atributos, ubicación y propietario. Gran parle de esta información, especialmente la relativa al almacenamiento, la gestiona el sistema operativo. El directorio es pro-

piamente un archivo, poseído por el sistema operativo y accesible a través de diversas rutinas de gestión de archivos. Aunque parte de la información de los directorios está disponible para los usuarios y aplicaciones, en general, la información se proporciona indirectamente, a través de rutinas del sistema. De este modo, los usuarios no pueden acceder directamente al directorio, incluso en modo de sólo lectura.

TABLA 11.2 Elementos de información de un directorio de archivos

	Información Básica
Nombre del archivo	Nombre elegido por el creador (usuario o programa). Debe ser único en un directorio específico.
Tipo de Archivo	Por ejemplo: texto, binario, módulo de carga, etc.
Organización del Archivo	Para sistemas que soportan varias organizaciones.
	Información de Direccionamiento
Volumen	Indica el dispositivo donde se almacena el archivo.
Dirección de Comienzo	Dirección física de inicio en memoria secundaria (cilindro, pista y número de bloque en disco).
Tamaño Usado	Tamaño actual del archivo en bytes, palabras o bloques.
Tamaño Asignado	Tamaño máximo del archivo.
	Información de Control de Acceso
Propietario	Usuario con control sobre el archivo. El propietario puede otorgar o denegar acceso a otros usuarios y cambiar estos privilegios.
Información de Acceso	Una versión simple de este elemento incluye el nombre del usuario y la contraseña para cada usuario autorizado.
Acciones Permitidas	Controla la lectura, escritura, ejecución y transmisión por una red.
	Información de Uso
Fecha de Creación	Cuándo se añadió el archivo al directorio
Identidad del Creador	Normalmente, pero no siempre, el propietario.
Fecha de Última Lectura	Fecha de la última vez que se leyó un registro.
Identidad del Último Lector	Usuario que hizo la lectura.
Fecha de Última Modificación	Fecha de la última actualización, inserción o borrado.
Identidad del Último Modificador	Usuario que hizo la modificación
Fecha de la Última Copia de Seguridad	Fecha de la última vez que el archivo fue copiado en otro medio de almacenamiento.
Utilización Actual	Información sobre la actividad actual sobre el archivo, tal como el (los) proceso(s) que tienen abierto el archivo, si está bloqueado por un proceso y si el archivo ha sido actualizado en memoria principal, pero aún no en disco.

En la tabla 11.2 se propone la información que se almacena normalmente en el directorio para cada archivo del sistema. Desde el punto de vista del usuario, el directorio ofrece una traducción entre los nombres de archivo conocidos para usuarios y aplicaciones y los archivos, propiamente dicho. Por tanto, cada entrada incluirá el nombre del archivo. Casi lodos los sistemas trabajan con clases diferentes de archivos y diferentes organizaciones de archivos, por lo que también se incluye esta información. Un tipo de información importante sobre cada archivo es aquella relativa a su almacenamiento, incluyendo su ubicación y tamaño. En los sistemas compartidos, también es importante ofrecer información para controlar el acceso al archivo. Normalmente, un usuario será el propietario del ar-

chivo y podrá otorgar ciertos privilegios de acceso a otros usuarios. Finalmente, se necesita información sobre su uso para gestionar la utilización actual del archivo y guardar un histórico.

Estructura

La manera en que se almacena la información de la tabla 11.2 difiere mucho en varios sistemas. Parle de la información puede guardarse en un registro de cabecera asociado al archivo: esto reduce la cantidad de espacio necesario para el directorio, haciendo mas fácil mantener todo el directorio o parte en memoria principal para mejorar la velocidad. Por supuesto, algunos elementos clave deben permanecer en el directorio; normalmente, estos incluyen el nombre, dirección, tamaño y organización.

La forma más simple de estructuración de un directorio es una lista de entradas, una para cada archivo. Esta estructura puede representarse con un simple archivo secuencial, con el nombre del archivo haciendo las veces de clave. En algunos sistemas antiguos monousuario se ha usado esta técnica. Sin embargo, no es adecuada cuando múltiples usuarios comparten el sistema e incluso para un solo usuario con muchos archivos.

Para comprender los requisitos de una estructura de archivo, merece la pena considerar los tipos de operaciones que pueden realizarse con un directorio:

- *Buscar:* Cuando un usuario o aplicación referencia a un archivo, debe buscarse en el directorio la entrada correspondiente al archivo.
- *Crear archivo:* Al crear un nuevo archivo, debe añadirse una entrada al directorio.
- *Borrar archivo:* Al borrar un archivo, debe eliminarse una entrada del directorio.
- *Listar directorio:* Puede solicitarse todo el directorio o una parte. Generalmente, esta petición la hace un usuario y el resultado es una lista de todos los archivos poseídos por dicho usuario, junto a algunos de los atributos de cada archivo (tipo, información de control de acceso, información de uso, etc.)

Una simple lista no se ajusta bien a estas operaciones. Considérense las necesidades de un solo usuario. El usuario puede tener muchos tipos de archivos, incluyendo documentos de texto, archivos gráficos, hojas de cálculo, etc. El usuario puede querer tenerlos organizados por proyecto, tipo o de otra manera conveniente. Si el directorio es una simple lista secuencial, no ofrecerá ayuda alguna en la organización de los archivos y obligará al usuario a tener cuidado de no usar el mismo nombre para dos tipos diferentes de archivo. El problema es mucho peor en un sistema compartido. Los nombres únicos se convierten en un problema serio. Además, es difícil ocultar a los usuarios determinadas partes del directorio global cuando no hay una estructura inherente en el mismo.

Un buen comienzo para resolver estos problemas podría ser acudir a un esquema de dos niveles donde hay un directorio para cada usuario y un directorio maestro. El directorio maestro dispone de una entrada para cada directorio de usuario, incluyendo una dirección e información de control de acceso. Cada directorio de usuario es una simple lista de los archivos del usuario. Esta disposición significa que los nombres deben ser únicos sólo dentro de la colección de archivos de cada usuario y que el sistema de archivos puede hacer cumplir fácilmente unas restricciones de acceso a los directorios. Sin embargo, todavía no ofrece a los usuarios ayuda alguna para estructurar sus colecciones de archivos.

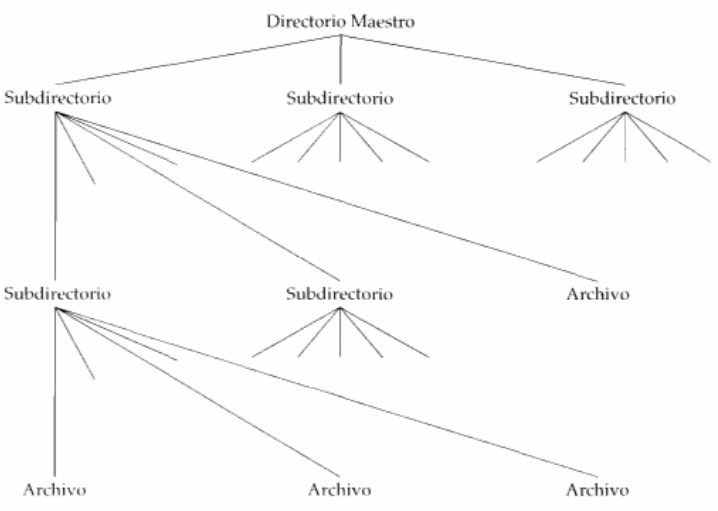


FIGURA 11.4 Directorio estructurado en árbol

Un método más potente y flexible, adoptado casi universalmente, es el directorio jerárquico o estructurado en árbol (figura 11.4). Como antes, existe un directorio maestro que contiene un número determinado de directorios de usuario. Cada uno de estos directorios puede tener a su vez subdirectorios y archivos como entradas. Esto se cumple en cualquier nivel. Es decir, en cualquier nivel, un directorio puede constar de entradas para subdirectorios y/o entradas para archivos.

Queda comentar cómo se organiza cada directorio y subdirectorio. El método más simple es, por supuesto, almacenar cada directorio como un archivo secuencia!. Cuando los directorios contengan un número muy grande de entradas, tal organización puede conducir a tiempos de búsqueda innecesariamente grandes. En tal caso, se prefiere una estructura de dispersión.

Designación

Los usuarios deben poder referirse a un archivo por medio de un nombre simbólico. Evidentemente, cada archivo del sistema debe tener un nombre único para que las referencias al archivo no sean ambiguas. Por otra parte, proporcionar nombres únicos es una carga inaceptable para los usuarios, especialmente en un sistema compartido.

El uso de directorios estructurados en árbol minimiza la dificultad de asignar nombres únicos. Cualquier archivo del sistema puede ser localizado siguiendo un camino desde el directorio raíz o maestro, descendiendo por varias ramas hasta que se alcance el archivo. La serie de nombres de directorios, terminados con el propio nombre del archivo, constituye el **nombre de camino** del archivo. Como ejemplo, el archivo de la esquina inferior izquierda

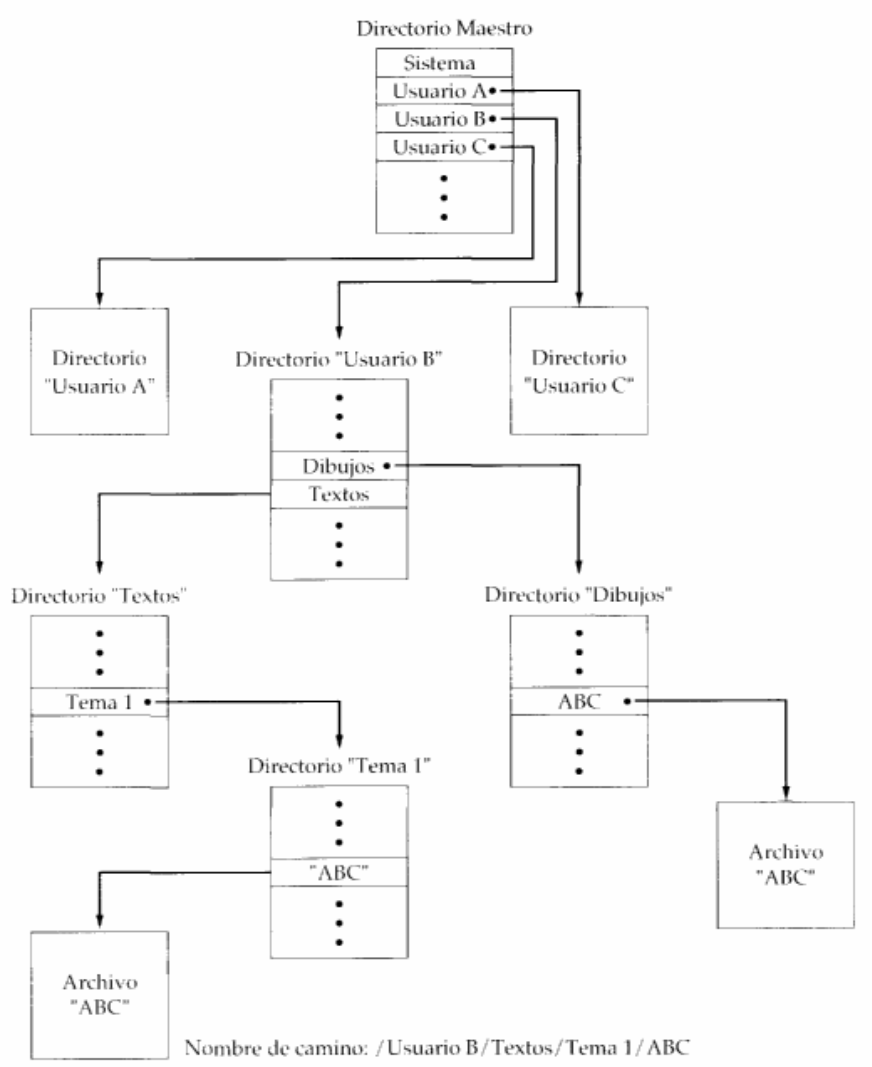


FIGURA 11.5 Ejemplo de directorio estructurado en árbol

de la figura 11.5 tiene el nombre de camino /UsuarioB/Textos/Temal/ABC. La barra oblicua se utiliza para delimitar los nombres en la secuencia. El nombre del directorio maestro queda implícito porque todos los caminos comienzan en tal directorio. Es perfectamente aceptable tener varios archivos con el mismo nombre de archivo mientras tengan nombres de camino únicos. De esta forma, puede haber otro archivo en el sistema con el nombre ABC, pero su nombre de camino es /UsuarioB/Dibujos/ABC.

Aunque el nombre de camino facilita la elección de los nombres de archivo, para un usuario sería incómodo tener que deletrear el nombre de camino entero cada vez que haga una referencia a un archivo. Normalmente, cada usuario interactivo o proceso tiene asociado un directorio actual, conocido a menudo como **directorio de trabajo.** Las referencias a los archivos son entonces relativas al directorio de trabajo. Por ejemplo, si el directorio de trabajo del usuario B es "Textos", entonces el nombre de camino Temal/ABC es suficiente para identificar al archivo de la esquina inferior izquierda de la figura 11.5. Cuando un usuario interactivo se conecte o cuando se cree un proceso, el valor por defecto para el directorio de trabajo será el directorio del usuario. Durante la ejecución, el usuario puede navegar por el árbol y así definir directorios de trabajo diferentes.

11.4 COMPARTICION DE ARCHIVOS

En un sistema multiusuario. casi siempre existe la necesidad de permitir a los usuarios compartir archivos. Emergen entonces deis cuestiones: los derechos de acceso y la gestión de los accesos simultáneos.

Derechos de Acceso

El sistema de archivos debe ofrecer una herramienta flexible para permitir la compartición general de archivos entre los usuarios, así como un conjunto de opciones de forma que se pueda controlar la manera en que se accede a cada archivo en particular. Normalmente, a los usuarios o grupos de usuarios le son concedidos ciertos derechos de acceso a cada archivo. Se ha venido usando un amplio rango de derechos de acceso. La lista siguiente es representativa [CAL1S2] de los derechos de acceso que pueden asignarse a un usuario particular para un archivo específico:

- *Ninguno:* El usuario no puede siquiera conocer la existencia del archivo, ni mucho menos acceder al mismo. Para aplicar esta restricción, no se permite al usuario leer el directorio de usuario que incluya al archivo.
- *Conocimiento:* El usuario puede determinar que el archivo existe y quién es su propietario. El usuario es capaz de solicitar derechos de acceso adicionales al propietario.
- *Ejecución:* El usuario puede cargar y ejecutar un programa pero no puede copiarlo. Los programas comerciales se hacen a menudo accesibles con esta restricción.
- *Lectura:* El usuario puede leer el archivo para cualquier propósito, incluyendo copia y ejecución. Algunos sistemas son capaces de hacer valer la distinción entre visualizar y copiar. En el primer caso. el contenido del archivo puede mostrarse al usuario, pero éste no tiene medios para hacer una copia.
- *Adición:* El usuario puede añadir datos al archivo, generalmente al final, pero no puede modificar o borrar el contenido del mismo. Este derecho es útil en la recopilación de datos a partir de un conjunto de fuentes.
- *Actualización:* El usuario puede modificar, borrar y añadir dalos al archivo. La actualización incluye generalmente la escritura del archivo al principio, la reescritura por completo o en parte y la eliminación de todos los datos o parte de ellos. Algunos sistemas distinguen varios grados de actualización.

- *Cambio de protección:* El usuario puede cambiar los derechos de acceso otorgados a otros usuarios. Normalmente, este derecho es detentado sólo por el propietario. En algunos sistemas, el propietario puede otorgar este derecho a otros usuarios. Para evitar el abuso de este mecanismo, el propietario del archivo es normalmente capaz de especificar qué derechos pueden ser cambiados por el poseedor de este derecho.
- *Borrado:* El usuario puede borrar el archivo del sistema de archivos.

Puede considerarse que estos derechos constituyen una jerarquía, con cada uno incluyendo a todos los que le preceden. De este modo, si un usuario particular adquiere el derecho de actualización para un archivo determinado, también habrá adquirido los derechos siguientes: conocimiento, ejecución, lectura y adición.

Un usuario es designado como propietario de un archivo dado. Normalmente será la persona que creó el archivo al principio. El propietario dispone de los derechos de acceso listados antes y puede otorgar derechos a los otros. Puede ofrecerse acceso a las siguientes clases de usuarios :

- *Usuario específico:* Usuarios individuales designados por su ID de usuario.
- *Grupos de usuarios:* Un conjunto de usuarios no definidos individualmente. El sistema deberá disponer de algún medio para guardar constancia de la militancia de los grupos de usuarios.
- *Todos:* Todos los usuarios que tengan acceso al sistema. Estos serán archivos públicos.

Accesos Simultáneos

Cuando se otorga acceso para añadir o actualizar un archivo a más de un usuario, el sistema operativo o el sistema de gestión de archivos debe hacer cumplir una disciplina. Un método de tuerza bruta consiste en permitir a los usuarios bloquear el archivo entero cuando lo vaya a actualizar. Un mejor control es bloquear los registros individuales durante la actualización. Básicamente, este es el problema de los lectores/escritores discutido en el capítulo 4. Al diseñar la posibilidad de accesos compartidos, deben abordarse aspectos de exclusión mutua e interbloqueo.

11.5 AGRUPACIÓN DE REGISTROS

Como se indica en la figura 11.2, los registros son la unidad lógica de acceso a los archivos, mientras que los bloques son la unidad de E/S para almacenamiento secundario. Para realizar E/S, los registros deben organizarse en bloques.

Hay varios aspectos a considerar. En primer lugar, ¿los bloques deberán ser de longitud fija o variable? En la mayoría de los sistemas, los bloques son de longitud fija. Esto simplifica la E/S, la asignación de memoria intermedia *(buffers)* en memoria principal y la organización de los bloques en memoria secundaria. En segundo lugar, ¿cuál deberá ser el tamaño relativo de un bloque, en comparación con el tamaño medio de registro? El compromiso es éste: Cuanto mayor sea el bloque, más registros se pasarán en una operación de E/S. Esto es una ventaja si el archivo se está procesando secuencialmente, porque el número de operaciones de E/S se reduce usando bloques mayores, acelerando así el procesamiento. Por otro lado, si se accede aleatoriamente a los registros y no se observa ninguna cercanía particular en las referencias, el uso de bloques mayores redunda en una transferencia innecesaria de re-

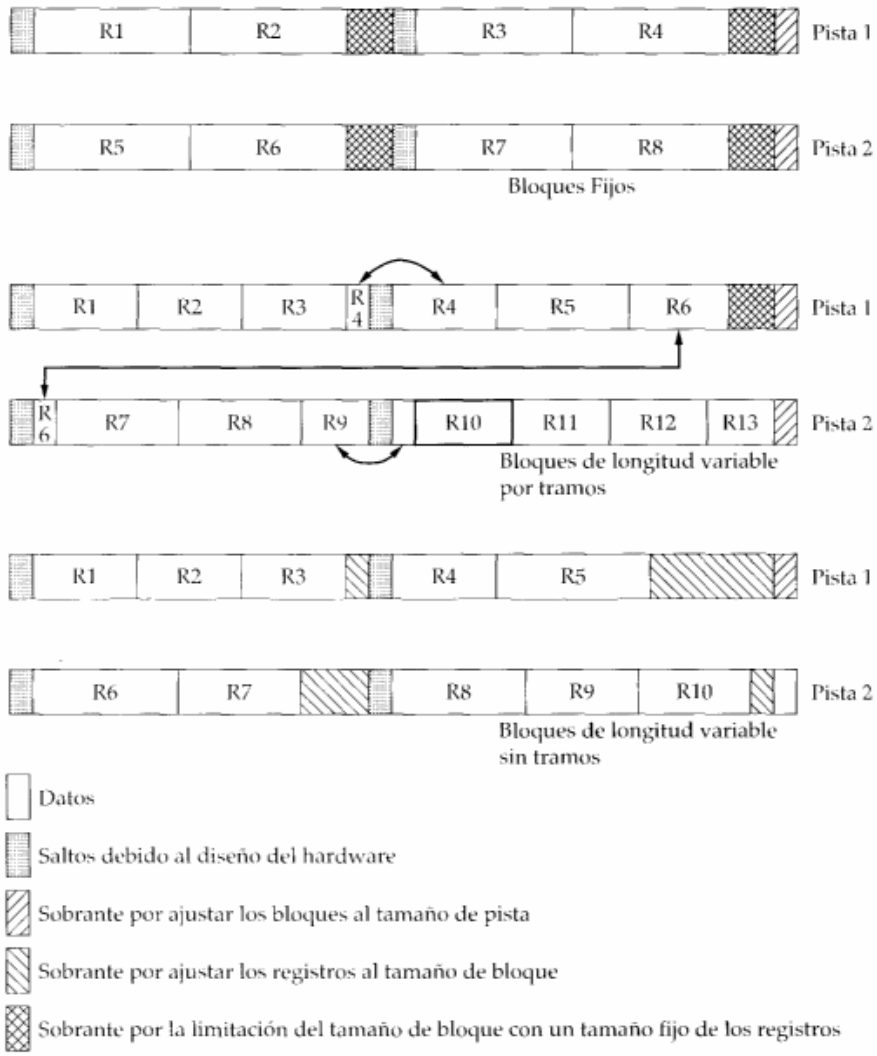


FIGURA 11.6 Métodos de agrupación de registros [WIED87]

gistros sin usar. Sin embargo, combinando la frecuencia de las operaciones secuenciales con la posibilidad de cercanía de referencias, se puede afirmar que el tiempo de transferencia de E/S se reduce usando bloques mayores. La pega es que los bloques grandes necesitan buffers de E/S mayores, haciendo más dificultosa la gestión de buffers.

Dado un tamaño de bloque, pueden seguirse los siguientes tres métodos de agrupación en bloques:

• *Bloques fijos:* Se usan registros de longitud fija, guardándose en cada bloque un número entero de registros. Puede existir espacio sin usar al final de cada bloque.

- *Bloques de longitud variable por tramos:* Se usan registros de longitud variable que se agrupan en bloques sin dejar espacio sin usar. De este modo, algunos registros deben abarcar dos bloques, indicando el tramo de continuación con un puntero al bloque siguiente.
- *Bloques de longitud variable sin tramos:* Se usan registros de longitud variable, pero no se dividen en tramos. En la mayoría de los bloques habrá un espacio desperdiciado, debido a la imposibilidad de aprovechar el resto del bloque si el registro siguiente es mayor que el espacio sin usar restante.

En la figura 11.6 se ilustran estos métodos, suponiendo que se guarda un archivo en bloques secuenciales de un disco. El efecto no cambiaría si se usara algún otro esquema de asignación (ver sección 11.6).

Los bloques de tamaño fijo son el modo mas común de archivos secuenciales con registros de longitud variable. Los bloques de longitud variable por tramos constituyen un almacenamiento eficaz y no ponen límites al tamaño de los registros. Sin embargo, esta técnica es difícil de implementar. Los registros que ocupan dos bloques necesitan dos operaciones de E/S y los archivos se hacen difíciles de actualizar, sin tener en cuenta la organización. Los bloques de longitud variable sin tramos producen un desperdicio de espacio y limitan el tamaño del registro al tamaño del bloque.

La técnica de agrupación de registros puede colaborar con el hardware de memoria virtual si procede. En un entorno de memoria virtual, es deseable que la unidad básica de transferencia sea la página. Generalmente, las páginas son bastante pequeñas, de forma que es poco práctico tratarlas como bloques sin tramos. De acuerdo a esto, algunos sistemas combinan múltiples páginas para crear un bloque mayor para la E/S de archivos. Este método es usado en los archivos VSAM de las máquinas **IBM.**

11.6 GESTIÓN DEL ALMACENAMIENTO SECUNDARIO

En memoria secundaria, un archivo consta de un conjunto de bloques. El sistema operativo o el sistema de gestión de archivos es responsable de la asignación de los bloques a archivos. Esto suscita dos cuestiones sobre la gestión. En primer lugar, debe asignarse el espacio de memoria secundaria a los archivos y, en segundo lugar, es necesario guardar constancia del espacio disponible para asignar. Se comprobará que estas dos tareas están relacionadas; es decir, el método tomado para asignar los archivos puede influir en el método de gestión del espacio libre. Además, se verá que existe una interacción entre la estructura de archivo y la política de asignación.

Esta sección va a comenzar observando las alternativas de asignación de archivos en un solo disco. La gestión del espacio libre se abordará más tarde. Por último, se examinarán las técnicas de almacenamiento de archivos sencillos en varios discos.

Asignación de Archivos

En la asignación de archivos surgen varias cuestiones:

1. Cuando se crea un nuevo archivo, ¿se asigna de una sola vez el máximo espacio que necesite?

2. El espacio se asigna a un archivo en forma de una o más unidades contiguas, que se llaman *secciones.* El tamaño de una sección puede variar desde un único bloque a un archivo entero. ¿.Qué tamaño de sección debería usarse para asignar archivos?
3. ¿Qué tipo de estructura de datos o tabla se usará para guardar constancia de las secciones asignadas a un archivo? Dicha tabla se conoce normalmente como **tabla de** asignación **de archivos** (FAT).

A continuación se examinan estas cuestiones.

Asignación Previa frente a Asignación Dinámica

Una política de asignación previa requeriría que el tamaño máximo de un archivo se declarase en el momento de crearlo. En algunos casos, como al compilar los programas, al crear archivos de datos de resumen o al transferir un archivo desde otro sistema por una red de comunicaciones, este valor puede estimarse. Sin embargo, para muchas aplicaciones es difícil, si no imposible, estimar de manera fiable el posible tamaño máximo del archivo. En esos casos, los usuarios y programadores de aplicaciones se inclinarían por sobrestimar el tamaño del archivo de forma que no se quedaran sin espacio. Evidentemente, esto es un derroche desde el punto de vista de la asignación de memoria secundaria. Por tanto, existen ventajas en el uso de la asignación dinámica, que asigna espacio a los archivos en secciones a medida que se necesitan.

Tamaño de Sección

La segunda cuestión de la lista anterior es la del tamaño de sección asignada a los archivos. En un extremo, se puede asignar una sección suficientemente grande para guardar el archivo entero. En el otro extremo, se asigna el espacio en disco de bloque en bloque. Al elegir el tamaño de sección, debe haber un compromiso relativo a la eficiencia desde el punto de vista de un solo archivo trente al del sistema global. [WEID87] considera los siguientes cuatro elementos en esta elección:

1. La contigüidad del espacio aumenta el rendimiento, especialmente para las operaciones de Recuperar_Siguiente y. sobremanera, para ejecutar las transacciones de un sistema orientado a transacciones.
2. Disponer de un gran número de secciones pequeñas aumenta el tamaño de las tablas necesarias para gestionar la asignación de información.
3. Disponer de secciones de tamaño fijo —por ejemplo, bloques— simplifica la reasignación del espacio.
4. Disponer de secciones de tamaño variable o secciones pequeñas de tamaño fijo minimiza la pérdida de espacio no usado provocada por la sobreasignación.

Por supuesto, estos elementos interactúan entre sí y deben considerarse en conjunto. Como resultado se tienen dos opciones principales:

- *Secciones contiguas variables y grandes:* Esta opción ofrecerá un rendimiento mejor. El tamaño variable evitará la pérdida y las tablas de asignación de archivos serán pequeñas. Sin embargo, el espacio es difícil de reutilizar.
- *Bloques:* Las secciones fijas y pequeñas ofrecen una flexibilidad mayor. Se pueden necesitar tablas grandes o estructuras complejas para su asignación. La contigüidad se abandona; los bloques se asignan a medida que se necesiten.

Cualquier opción es compatible con la asignación previa o con la asignación dinámica. En el primer caso, se asigna previamente a los archivos un grupo contiguo de bloques. Esto elimina la necesidad de una tabla de asignación de archivos; todo lo que se necesita es un puntero al primer bloque y el número de bloques asignados. En el segundo caso, todas las secciones necesarias son asignadas de una vez. Esto significa que la tabla de asignación del archivo permanecerá con tamaño fijo.

Con secciones de tamaño variable, hay que preocuparse de la fragmentación del espacio libre. Esta cuestión surgió cuando se estudiaba la memoria principal particionada en el capítulo 6. Las siguientes son algunas estrategias alternativas posibles:

- *Primer hueco (first fit:* Elegir el primer grupo de bloques contiguo sin usar de tamaño suficiente.
- *Mejor hueco (best fit):* Elegir el grupo más pequeño sin usar que tenga tamaño suficiente.
- *Hueco mas cercano (nearest fit):* Elegir el grupo sin usar de tamaño suficiente que este más cercano al asignado previamente al archivo para aumentar la cercanía.

No está claro qué estrategia es la mejor. La dificultad de modelar estrategias alternativas está en que intervienen muchos factores, incluyendo los tipos de archivo, la pauta de los accesos a archivo, el grado de multiprogramación, otros factores de rendimiento, la cache de disco, la planificación del disco, etc.

Métodos de Asignación de Archivos

Después de ver la discusión entre asignación previa y dinámica y el tamaño de sección, se está en posición de considerar métodos específicos de asignación de archivos. Son de uso común tres métodos: contiguo, encadenado e indexado. En la tabla 11.3 se resumen algunas de las características de cada método.

Con **asignación contigua,** cuando se crea un archivo se le asigna un único conjunto contiguo de bloques (figura 11.7). Por tanto, ésta es una estrategia de asignación previa que emplea secciones de tamaño variable. La tabla de asignación de archivos necesita sólo una entrada por cada archivo, que muestre el bloque de comienzo y la longitud del archivo. La asignación contigua es la mejor desde el punto de vista de un archivo secuencial individual. Se pueden traer múltiples bloques de una vez para mejorar el rendimiento en los tratamientos secuenciales. También es fácil recuperar un solo bloque. Por ejemplo, si un archivo empieza en el bloque *b* y se necesita el bloque i-ésimo del archivo, su ubicación en memoria secundaria es simplemente $b + i$. La asignación contigua presenta algunos problemas. Se producirá fragmentación externa, haciendo difícil encontrar bloques contiguos de espacio de tamaño suficiente. De cuando en cuando, será necesario ejecutar un algoritmo de compactación para liberar espacio adicional en el disco (figura 11.8). Además, en la asignación previa es necesario declarar el tamaño del archivo en el momento de su creación, lo que puede ocasionar los problemas antes mencionados.

En el extremo opuesto a la asignación contigua está la **asignación encadenada** (figura 11.9). Normalmente, la asignación se hace con bloques individuales. Cada bloque contendrá un puntero al siguiente bloque de la cadena. La tabla de asignación de archivos necesita de nuevo una sola entrada por cada archivo que muestre el bloque de comienzo y la longitud del archivo. Aunque es posible la asignación previa, es más común simplemente asignar bloques a medida que se necesiten. La elección de bloques es entonces una cuestión simple:

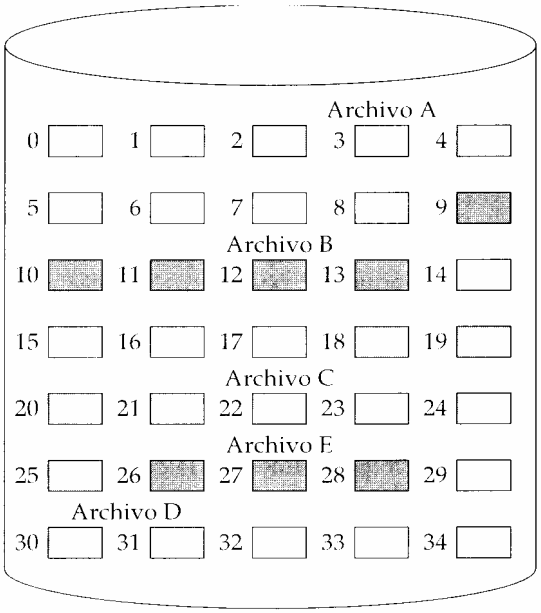


Tabla de Asignación de Archivos

Nombre de Archivo	Bloque de Inicio	Longitud
Archivo A	2	3
Archivo B	9	5
Archivo C	18	8
Archivo D	30	2
Archivo E	26	3

FIGURA 11.7 Asignación contigua de archivos

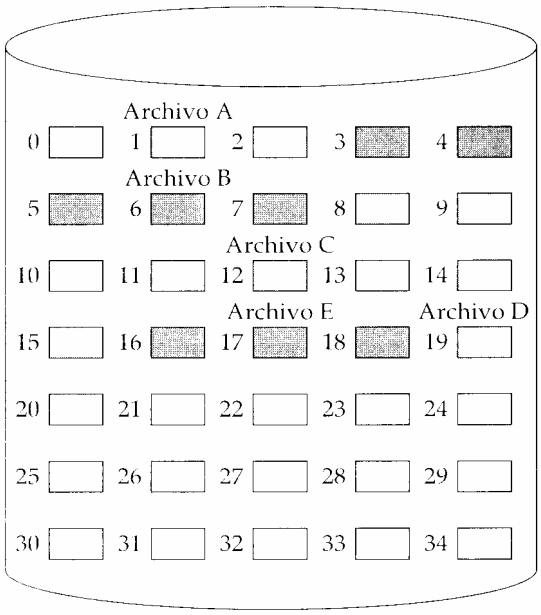


Tabla de Asignación de Archivos

Nombre de Archivo	Bloque de Inicio	Longitud
Archivo A	0	3
Archivo B	3	5
Archivo C	8	8
Archivo D	19	2
Archivo E	16	3

FIGURA 11.8 Asignación contigua de archivos (tras compactación)

cualquier bloque libre puede añadirse a la cadena. No hay que preocuparse por la fragmentación externa porque sólo se necesita un bloque cada vez. Este tipo de organización física se ajusta mejor a los archivos secuenciales que van a ser procesados secuencialmente. Para seleccionar un bloque individual de un archivo se requiere recorrer la cadena hasta el bloque deseado.

TABLA 11.3Métodos de asignación de archivos

Característica	Contigua	Encadenada	Indexada	Indexada
¿Secciones de tamaño fijo o variable?	Variable	Bloques fijos	Bloques fijos	Variable
¿Asignación previa?	Necesaria	Posible	Posible	Posible
Tamaño de sección	Grande	Pequeño	Pequeño	Medio
Frecuencia de asignación	Una vez	Medio-bajo	Alto	Bajo
Tiempo para asignar	Medio	Grande	Corto	Medio
Tamaño de la tabla de asignación de archivos	Una entrada	Una entrada	Grande	Medio

Una consecuencia del encadenamiento, como se ha descrito hasta ahora, es que no hay cabida para el principio de cercanía. De esta manera, si es necesario traer varios bloques de un archivo al mismo tiempo, como en el procesamiento secuencial, se necesita una serie de accesos a partes diferentes del disco. El efecto es quizá más significativo en un sistema monousuario, pero puede ser también de consideración en un sistema compartido. Para solventar este problema, algunos sistemas concentran los archivos periódicamente (figura I 1.10).

La **asignación indexada** trata muchos de los problemas de las asignaciones contigua y encadenada. En este caso, la tabla de asignación de archivos contiene un índice separado de un nivel para cada archivo; el índice posee una entrada para cada sección asignada al archivo. Normalmente, los índices no están almacenados físicamente como parte de la tabla de asignación de archivos. Más exactamente, el índice del archivo se guardará en un bloque aparte y la entrada del archivo en la tabla de asignación apuntará a dicho bloque. La asignación puede hacerse por bloques de tamaño fijo (figura 11.11) o en secciones de tamaño variable (figura 11.12). La asignación por bloques elimina la fragmentación externa, mientras que la asignación por secciones de tamaño variable mejora la cercanía. En cualquier caso, los archivos pueden concentrarse en zonas cercanas de cuando en cuando. La concentración reduce el tamaño del índice en el caso de secciones de tamaño variable, pero no en el caso de asignación por bloques. La asignación indexada soporta tanto el acceso secuencia! como el acceso directo a los archivos y por ello se ha convertido en la forma más popular de asignación de archivos.

Gestión del Espacio Libre

Al igual que el espacio asignado a los archivos, se debe gestionar el espacio que no queda asignado actualmente a ningún archivo. Para llevar a cabo cualquiera de las técnicas de asignación que se han descrito, es necesario saber qué bloques del disco están disponibles. Por tanto, hace falta una tabla de asignación de disco además de una tabla de asignación de archivos. Tres técnicas son de uso común: las tablas de bits, las secciones libres encadenadas y la indexación.

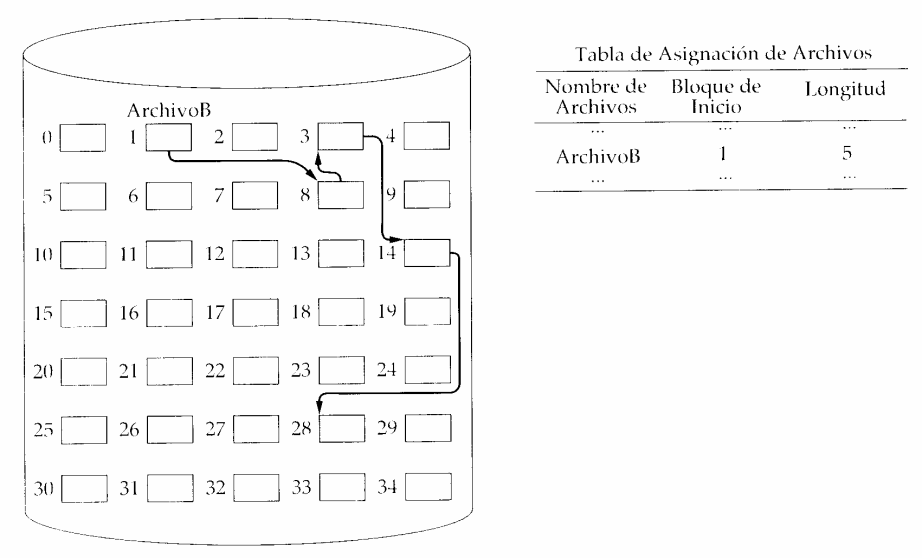


FIGURA 11.9 Asignación encadenada

Tablas de Bits

El método de las tablas de bits utiliza un vector que contiene un bit por cada bloque del disco. Cada entrada igual a O corresponde a un bloque libre y cada I corresponde a un bloque en uso. Por ejemplo, para la disposición del disco de la figura 11.7, se necesitaría un vector de longitud 35 que tendría el siguiente valor:

00111000011111000011111111111011000

Las labias de bits tienen la ventaja de que es relativamente fácil encontrar un bloqueo un grupo contiguo de bloques libres. Las tablas de bits trabajan bien con cualquiera de los métodos de asignación de archivos discutidos. Otra ventaja es que puede ser tan pequeña como sea posible y puede mantenerse en memoria principal. Esto evita la necesidad de traer la tabla de asignación de disco a memoria cada vez que se realice una asignación.

Secciones libres encadenadas

Las secciones libres pueden encadenarse juntas mediante un puntero y un valor de longitud en cada sección libre. Este método tiene un gasto insignificante porque no hay necesidad de tabla de asignación de disco, sino simplemente un puntero al comienzo de la cadena y la longitud de la primera sección. Este método sirve para todas las técnicas de asignación de archivos. Si la asignación se realiza por bloques, solamente hay que elegir el bloque libre del principio de la cadena y retocar el primer puntero o el valor de longitud. Si la asignación se hace por secciones de longitud variable, puede usarse el algoritmo del Primer Hueco: Se traen la cabeceras de la sección una a una para determinar la

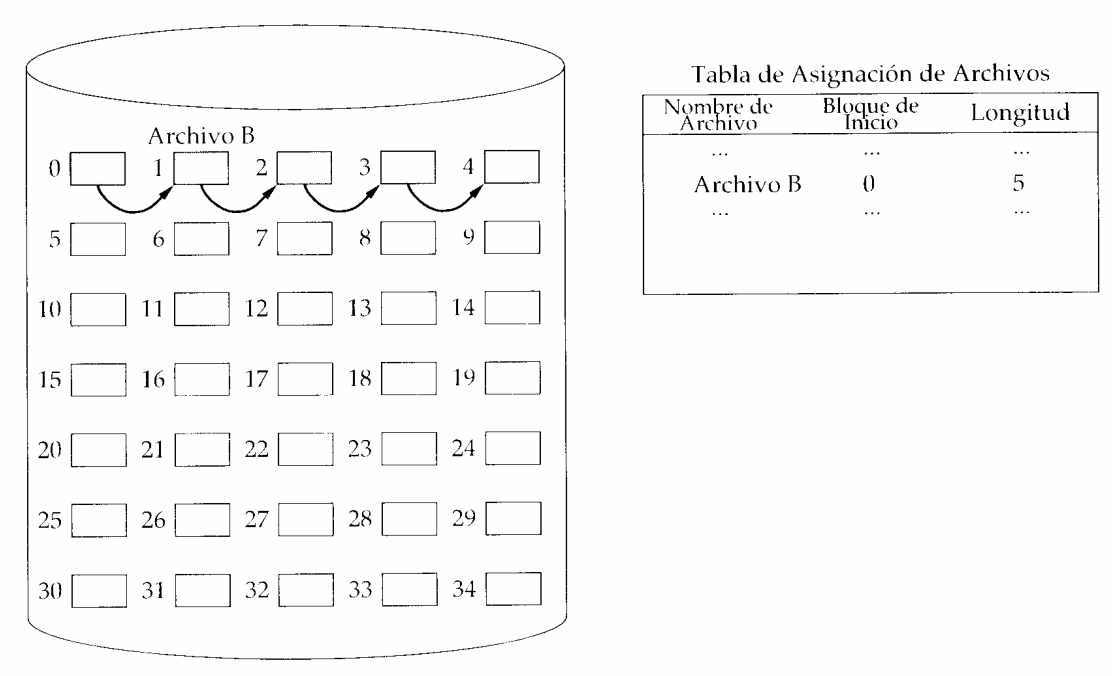


FIGURA 11.10 Asignación encadenada (tras concentración)

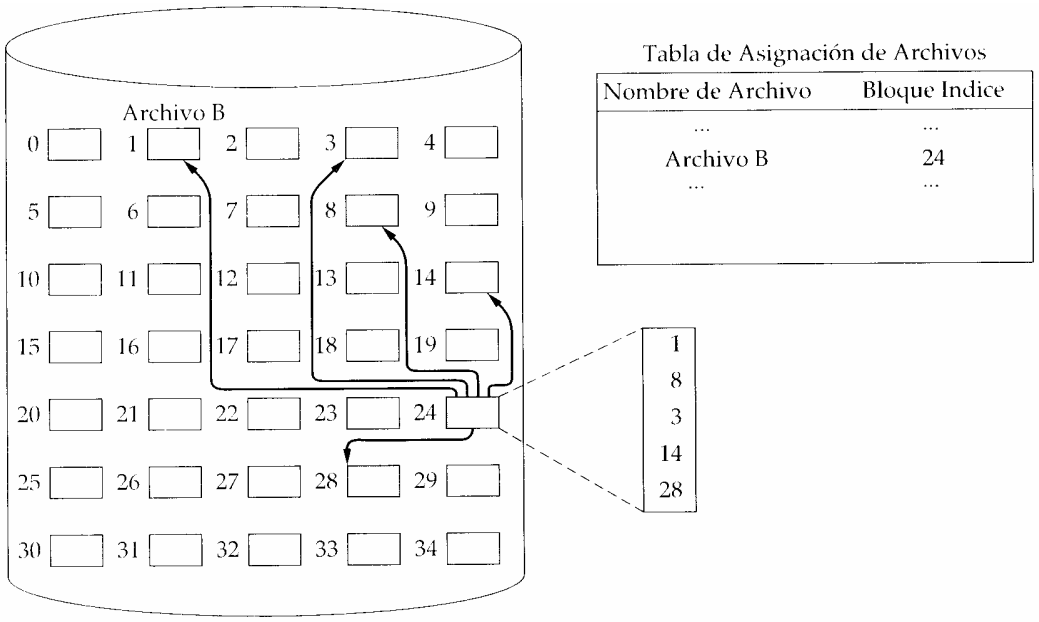


FIGURA 11.11 Asignación indexada por bloques

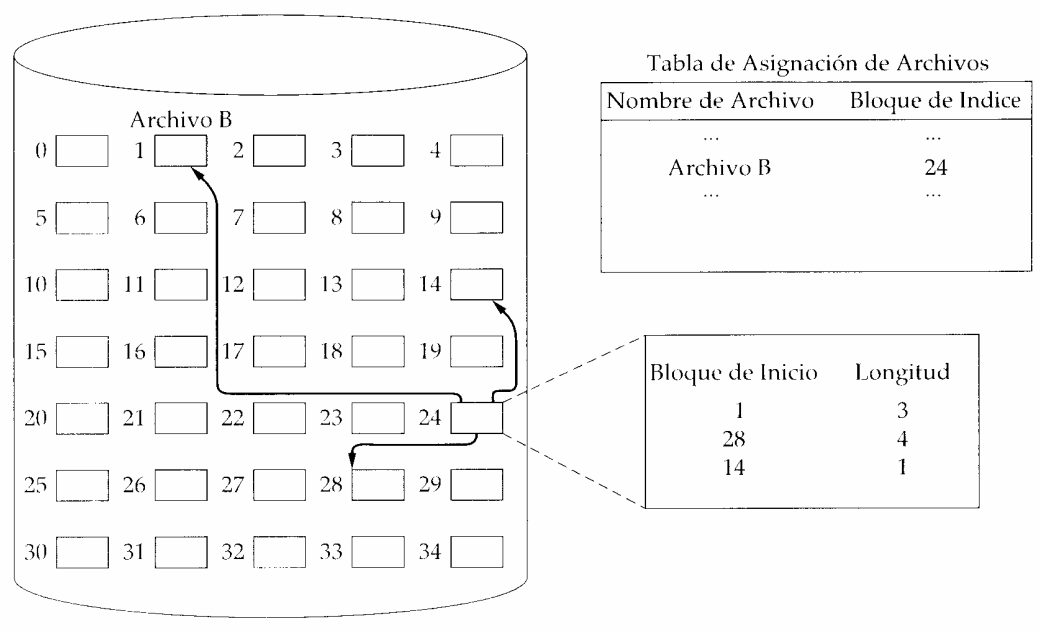


FIGURA 11.12 Asignación indexada por secciones de longitud variable

siguiente sección de la cadena que se ajuste. De nuevo, hay que retocar el puntero y las longitudes.

Indexación

El método de indexación trata el espacio libre como si fuera un archivo y utiliza una tabla índice como la descrita en el apartado "Métodos de Asignación de Archivos". Por rayones de eficiencia, el índice debe trabajar con secciones de tamaño variable mejor que con bloques. De este modo, habrá una entrada en la tabla para cada sección libre del disco. Este procedimiento ofrece un soporte eficaz para todos los métodos de asignación de archivos.

Fiabilidad

Considérese el escenario siguiente:

1. El usuario A solicita una asignación para añadir datos a un archivo existente.
2. La petición se atiende y se actualizan en memoria principal las tablas de asignación de disco y archivos, pero no aún en el disco.
3. El sistema se hunde y a continuación se reinicia.
4. El usuario B solicita una asignación y se le otorga un espacio en el disco que se solapa con la última asignación hecha al usuario A.
5. El usuario A accede a la sección solapada mediante una referencia que está almacenada en el archivo de A.

Esta dificultad surge debido a que, por eficiencia, el sistema mantiene copias de la tabla de asignación de disco y la tabla de asignación de archivos en memoria principal. Para evitar esta clase de errores, pueden darse los siguientes pasos cuando se solicite una asignación:

1. Bloquear en el disco la tabla de asignación de disco. Este impedirá a otro usuario alterar la tabla hasta que la asignación actual se complete.
2. Buscar espacio disponible en la tabla de asignación de disco. Se supone que se mantiene siempre en memoria principal una copia de la tabla de asignación de disco. Si no fuese así, primero debe leerse ésta del disco.
3. Asignar el espacio, actualizar la tabla de asignación de disco y actualizar el disco. Actualizar el disco implica volcar de nuevo al disco la tabla de asignación de disco. Con asignación encadenada, también se necesita actualizar algunos punteros del disco.
4. Actualizar la tabla de asignación de archivos y actualizar el disco.
5. Desbloquear la tabla de asignación de disco.

Esta técnica impedirá los errores. Sin embargo, cuando se asignan pequeñas secciones de manera frecuente, el impacto en el rendimiento será sustancial. Para reducir este gasto, puede usarse un esquema de asignación por lotes, en cuyo caso se obtienen lotes de secciones libres del disco para asignación. Las secciones correspondientes del disco se marcarán "en uso". La asignación usando lotes puede llevarse a cabo en memoria principal. Cuando se agota un lote, se actualiza en el disco la tabla de asignación de disco y se adquiere un lote nuevo. Si se produce un tallo del sistema, las secciones del disco marcadas "en uso" deben limpiarse de alguna forma antes de ser reasignadas. La técnica de limpieza dependerá de las características del sistema de archivos en particular.

Intercalado de Discos

En la sección 10.5 se discuten las tendencias en el rendimiento que han hecho de la E/S a disco un serio cuello de botella en muchos sistemas. Dos métodos para mejorar el rendimiento que se han presentado hasta ahora en el libro son la cache de disco y la planificación del disco. Sin embargo, ninguna de estas técnicas ofrece una solución satisfactoria en el futuro inmediato a los problemas de rendimiento en mantener archivos grandes o sistemas de bases de datos. La cache de disco es eficaz sólo si la pauta de acceso a los datos sigue el principio de cercanía y si la proporción de lecturas frente a escrituras es alta. El grado de cercanía depende del grado de multiprogramación, la estructura de archivo, el método de asignación de archivos y la naturaleza de la aplicación. Por tanto, aunque la cache de disco es útil, no ha resuelto por completo el problema del rendimiento. De forma similar, la planificación del disco puede mejorar el rendimiento, pero es improbable que solucione por sí misma la disparidad de rendimientos entre el disco y el computador.

En los últimos años, ha habido un gran interés en el uso de varios discos para almacenar un solo archivo, técnica conocida como *intercalado de discos (disk interleaving)* o *bandas de discos (disk sttripping)* [CHEN90, KATZ89, NG88, REDD89]. Un grupo de discos está intercalado si secciones sucesivas de un archivo se almacenan en discos diferentes de un vector *(array)* de discos. La granularidad de la distribución puede variar del nivel de bloque al nivel de byte. En el primer caso, se almacenan bloques sucesivos de un archivo en discos sucesivos; en el último caso, los bytes individuales de un archivo son distribuidos por los discos. En memoria, el archivo es almacenado de la forma usual, como un conjunto de bloques. Para leer o escribir un bloque, la asignación será:

B, se almacena en la unidad de disco $((B_i - 1) \bmod n) + 1$

donde

$\mathbf{B_i}$ = i-ésimo bloque (intercalado de bloques) o i-ésimo byte del bloque (intercalado de bytes) n = número de discos en el vector de discos

Con el intercalado de discos, las peticiones de E/S al archivo tienden a estar uniformemente distribuidas por todos los discos del vector, dando una productividad mayor y un mejor tiempo de respuesta. Este método de asignación aparta al administrador de las tareas de monitorizar el sistema y trasladar los archivos para obtener un equilibrio adecuado del sistema. Además, la carga de peticiones tenderá a mantenerse equilibrada aún con distintos tipos de peticiones de cada aplicación.

Se han estudiado tres procedimientos: unidades independientes, vectores sincronizados y vectores desincronizados.

Con las **unidades independientes** [OLS089], la granularidad del intercalado tiene lugar a nivel de bloque. Es decir, los bloques sucesivos de un archivo son ubicados en discos sucesivos. Debe mantenerse un índice global para cada archivo. El sistema operativo es responsable de manejar los discos en paralelo. Si las peticiones de E/S son para un solo bloque, serán encoladas en el disco que contenga tal bloque. Si llega una petición de varios bloques, se dividirá en un conjunto de peticiones de un solo bloque, encolándolas en los discos correspondientes, con lo que se logran accesos concurrentes a los bloques. Este método también permite dar servicio concurrentemente a las peticiones de múltiples bloques de diferentes archivos.

Este método puede ser eficaz cuando se genera un gran número de peticiones pequeñas independientes, como en los sistemas de proceso de transacciones. Sin embargo, el método no mejora necesariamente el rendimiento de una sola aplicación.

El método de los **vectores sincronizados** [KIM86] requiere que todos los discos giren, busquen y transfieran los datos en sincronía unos con otros. El intercalado es a nivel de byte. El resultado es que puede accederse a los datos en paralelo, reduciendo el tiempo de transferencia de datos en un factor de 1/*n*. donde *n* es el grado de intercalado. En realidad, todos los discos en conjunto funcionan como un único disco grande con *n* veces la velocidad de transferencia y *n* veces la capacidad de un solo disco. Como todos los discos están acoplados, cada uno recibe la misma proporción de peticiones y, por tanto, se obtiene un equilibrio de la carga en todos los discos.

Los vectores sincronizados no reducen el tiempo de búsqueda o la latencia de giro, pero ofrecen un gran ancho de banda al hacer la transferencia con todos los discos a la vez. Para una sola aplicación grande que lee o escribe de un archivo grande, este método puede aumentar la velocidad sustancialmente. La desventaja de los vectores sincronizados es que son complejos y caros de construir y no pueden ser construidos fácilmente a escala de decenas o cientos de discos.

El método de los **vectores desincronizados** [KIM87] simula el comportamiento de un vector sincronizado empleando discos usuales, sin modificar, en conjunción con una nueva arquitectura en el controlador. También aquí se usa el intercalado a nivel de bytes. Una petición de lectura se implementa con búsquedas independientes y transferencias de cada uno de los discos a un buffer

unidades de disco. Con este método se obtienen la mayor parte de los beneficios de rendimiento de los vectores sincronizados a un precio menor del hardware.

11.7 SISTEMA EJEMPLO — SISTEMA UNIX, VERSIÓN V

El núcleo *(kernel)* de UNIX contempla a todos los archivos como flujos de bytes. Cualquier estructura lógica interna será específica de la aplicación. Sin embargo, UNIX se ocupa de la estructura física de los archivos y distingue cuatro tipos de archivos de la manera siguiente:

- *Ordinarios:* Son archivos que contienen información introducida por un usuario, programa de aplicación o programa de utilidad del sistema.
- *Directorio:* Contiene una lista de nombres de archivo y punteros a nodos-i (nodos de información) asociados, que se describirán en breve. Los directorios están organizados jerárquicamente (figura 11.4). Los archivos de directorio son en realidad archivos ordinarios con unos privilegios especiales de protección de forma que sólo el sistema de archivos pueda escribir en ellos, mientras que los programas de usuarios disponen de acceso para lectura.
- *Especiales:* Usados para acceder a dispositivos periféricos, como terminales e impresoras. Cada dispositivo de E/S está asociado a un archivo especial, como se comenta en la sección 7.6.
- *Tubos con nombre:* son los estudiados en la sección 4.7.

Pista sección se va a ocupar del manejo de los archivos ordinarios, los cuales se corresponden con lo que la mayoría de los sistemas llaman simplemente archivos.

Nodos-i

Todos los tipos de archivos de UNIX son administrados por el sistema operativo por medio de nodos-i. Un nodo-i (nodo índice) es una estructura de control que contiene la información clave de un archivo necesaria para el sistema operativo. Pueden asociarse varios nombres de archivo a un mismo nodo-i. pero un nodo-i activo se puede asociar con un único archivo y cada archivo es controlado por un solo nodo-i.

Los atributos del archivo, así como sus permisos y otra información de control, se almacenan en el nodo-i. La tabla 11.4 muestra el contenido.

Asignación de Archivos

Los archivos se asignan en bloques. La asignación es dinámica, a medida que se necesita. No se emplea asignación previa. Por tanto, los bloques de un archivo no tienen por qué estar contiguos necesariamente. Se usa un método de indexación para seguir la pista de cada archivo, estando parte del índice almacenada en el nodo-i del archivo. El nodo-i incluye 39 bytes de información de direccionamiento, organizada como 13 direcciones o punteros de 3 bytes. Las primeras 10 direcciones apuntan a los primeros 10 bloques de datos del archivo. Si el archivo es mayor de 10 bloques, se usan uno o más niveles de indexación, de la forma siguiente:

La dirección undécima del nodo-i apunta a un bloque del disco que contiene la siguiente parte del índice. Este bloque se conoce como el *bloque de indexación simple.* Este bloque contiene los punteros a los siguientes bloques del archivo.

- Si el archivo contiene más bloques, la dirección duodécima del nodo-i apuntará a un bloque de indexación doble. Este bloque contiene una lista de direcciones de bloques de indexación simple adicionales. Cada uno de los bloques de indexación simple contiene a su vez punteros a los bloques del archivo.
- Si el archivo contiene aún más bloques, la dirección decimotercera del nodo-i apuntará a un bloque de indexación triple que consiste en un tercer nivel de indexación. Este bloque apunta a bloques de indexación doble adicionales.

Toda esta estructura se muestra en la figura 11.13. El número total de bloques de datos de un archivo dependerá de la capacidad de los bloques de tamaño fijo del sistema. En el UNIX Sistema V, la longitud de un bloque es de 1 Kb y cada bloque puede albergar un total de 256 direcciones de bloques. Por tanto, el tamaño máximo de un archivo usando este esquema se aproxima a los 16 Gb (tabla 11.5 de la página).

Este esquema tiene varias ventajas y son las siguientes:

1. Los nodos-i son de tamaño fijo y relativamente pequeños, por lo que pueden guardarse en memoria principal durante períodos largos.
2. Se puede acceder a los archivos más pequeños con poca indexación o ninguna, reduciendo así el procesamiento y el tiempo de acceso al disco.
3. El tamaño máximo teórico de un archivo es suficientemente grande como para satisfacer a casi todas las aplicaciones.

TABLA 11.4 Información de un nodo-i de UNIX

Modo de Archivo	Indicador *(flag)* de 16 bits que guarda los permisos de acceso y ejecución asociados con el archivo. 12-14 Tipo de archivo (regular, directorio, especial de caracteres o de bloques, tubo FIFO) 9-11 Indicadores de ejecución 8 Permiso de Lectura para el Propietario 7 Permiso de Escritura para el Propietario 6 Permiso de Ejecución para el Propietario 5 Permiso de Lectura para el Grupo 4 Permiso de Escritura para el Grupo 3 Permiso de Ejecución para el Grupo 2 Permiso de Lectura para el Resto 1 Permiso de Escritura para el Resto 0 Permiso de Ejecución para el Resto
Cuenta de Enlaces	Número de referencias al nodo-i en los directorios
ID del Propietario	Propietario individual del archivo
ID del Grupo	Grupo propietario asociado al archivo
Tamaño de Archivo	Número de bytes del archivo
Direcciones del Archivo	39 bytes de información de direccionamiento
Ultimo Acceso	Fecha del último acceso al archivo
Ultima Modificación	Fecha de la última modificación del archivo
Modificación del Nodo-i	Fecha de la última modificación del nodo-i

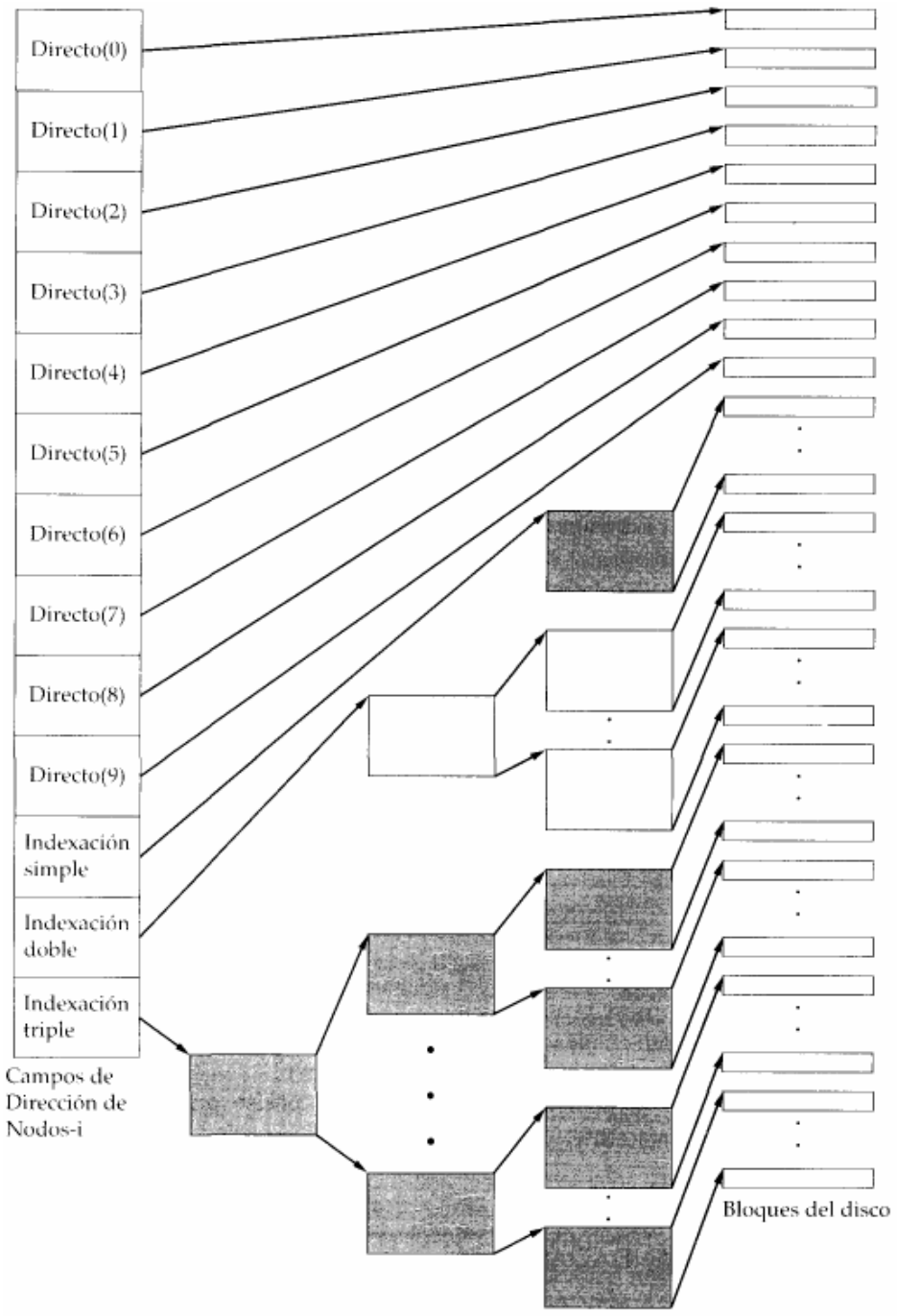


FIGURA 11.13 Esquema de direccionamiento de bloques en UNIX

11.8 RESUMEN

El sistema de gestión de archivos es el software del sistema que ofrece los servicios de manejo de archivos a usuarios y aplicaciones, incluyendo el acceso a archivos, conservación de directorios y control de acceso. Normalmente, el sistema de gestión de archivos se contempla como un servicio del sistema que se sirve a su ve/ del sistema operativo, mas que como una parte del propio sistema operativo. Sin embargo, en cualquier sistema, una parte de las funciones de gestión de archivos, por lo menos, las realiza el sistema operativo.

Un archivo es un conjunto de registros. La forma en que se accede a estos registros determina la organización lógica y. hasta cierto punto, la organización física en disco. Si un archivo va a ser básicamente procesado en su totalidad, la organización secuencia! es lamas simple y adecuada. Si el acceso secuencial es necesario pero también se desean el acceso aleatorio a archivos individuales, los archivos secuencial-indexados pueden dar el mejor rendimiento. Si el acceso a los archivos es principalmente aleatorio, los archivos indexados o los archivos de dispersión pueden ser los más apropiados.

Sea cual sea la estructura de archivo elegida, se necesita también un servicio de directorios para que los archivos puedan organizarse de una forma jerárquica. Esta organización es útil para que el usuario siga la pista de los archivos y para que el sistema de gestión de archivos proporcione a los usuarios un control de acceso junto a otros servicios.

Los registros de archivos, incluso los de tamaño fijo, no se ajustan generalmente al tamaño del bloque físico del disco. De esta forma, se necesita algún tipo de estrategia de agrupación. La estrategia de agrupación que se use quedará determinada por un compromiso entre la complejidad, el rendimiento y el aprovechamiento del espacio.

TABLA 11.5 Capacidad de un archivo de UNIX

Nivel	Número de Bloques	Número de Bytes
Directo	10	10 Kb
Indexación Simple	256	256 Kb
Indexación Doble	256 x 256 = 65K	65 Mb
Indexación Triple	256 x 65K = 16M	16 Gb

Una función clave de cualquier esquema de gestión de archivos es la gestión del espacio en disco. Una parte de esta función es la estrategia de asignación de bloques de disco a los archivos. Se han empleado una amplia variedad de métodos y de estructuras de datos para guardar constancia de la ubicación de cada archivo. Además, también debe gestionarse el espacio en disco sin asignar. Esta última función consiste principalmente en mantener una tabla de asignación de disco que indique los bloques que están libres.

11.9 LECTURAS RECOMENDADAS

Hay bastantes libros buenos sobre la gestión de archivos. Todos los siguientes se centran en la gestión de archivos pero también tratan cuestiones relacionadas con los sistemas operativos. El más útil es, quizás,

[WEID87], que toma una visión cuantitativa de la gestión de archivos y trata las cuestiones propuestas en la figura 11.2, desde la planificación del disco hasta la estructura de los archivos. [LIVA90] hace énfasis en las estructuras de archivo, ofreciendo un buen y amplio estudio con análisis comparativos de rendimiento. [GROS86] ofrece una visión equilibrada de los aspectos relacionados con la E/S a archivos y los métodos de acceso. También contiene descripciones generales de todas las estructuras de control necesarias en un sistema de archivos. Estas descripciones proporcionan un valioso catálogo a la hora de valorar el diseño de un sistema de archivos. [FOLK87] hace hincapié en el tratamiento de archivos, abordando cuestiones como el mantenimiento, la búsqueda, la ordenación y el coprocesamiento secuencial. En [MART77] se tratan cuestiones sobre bases de datos, pero cerca de la mitad del libro está dedicado a las estructuras de archivo, cubriendo gran parte del contenido de este libro.

Los estudios sobre el intercalado de discos, con énfasis en los modelos de rendimiento, pueden encontrarse en [CHEN90], [REDD89] y [NG88].

CHEN90 CHEN, P. y PATTERSON, D. "Maximizing Performance in a Stripped Disk Array". *Proceedings, 17th Annual International Symposium on Computer Architecture,* mayo de 1990.

FOEKS7 FOLK, M. y ZOEELICK. B. *File Structures: A Conceptual Toolkit.* Addison-Wesley, Reading, MA. 1987.

GROSS86 GROSSHANS, D. *File Systems: Design and Implementation.* Prentice-Hall, Englewood Cliffs, N.J, 1986.

LIVA90 LIVADAS, P. *File Structures: Theory und Practice.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

MART77 MARTIN, J. *Computer Data-Base Organization.* Prentice-Hall. Englewood Cliffs, NJ, 1977.

NG88 NG. S.: LANG. D. y SELINGER. R. "Trade-Offs Between Devices and Paths in Achieving Disk Interleaving". *Proceedings. 15th . International Symposium on Computer Architecture,* junio de 1988.

REDD89 REDDY. A. y BANERJEE. P. "An Evaluation of Multiple-Disk I/O Systems". *IEEE Transactions on Computers.* diciembre de 1989.

WEID87 WEIDERHOLD. G. *File Organization for Database Design.* McGraw-Hill, Nueva York, 1987.

11.10 PROBLEMAS

11.1 Dados:

B = tamaño del bloque

R = tamaño del registro

P = tamaño de un puntero a bloque

F = factor de agrupación: número esperado de registros en un bloque

Dar una fórmula para *F* para cada uno de los métodos de agrupamiento representados en la figura 11.6.

11.2 Un esquema para evitar el problema de la asignación previa frente al desperdicio o falta de contiguedad es asignar secciones de tamaño creciente a medida que el archivo crezca. Por ejemplo, se puede comenzar con un tamaño de la sección en cada asignación.

Considérese un archivo de *n* registros con un factor de agrupación de *F* y supóngase que se usa un índice simple de un nivel como tabla de asignación de archivos.

a) Dar un límite superior para el número de entradas de la tabla de asignación de archivos, en función de *F y n.*

h) ¿Cuál es la cantidad máxima de espacio sin usar asignado a un archivo en un momento dado?

11.3 Determine qué organización de archivos usaría para maximizar la eficiencia en términos de velocidad de acceso, aprovechamiento del espacio de almacenamiento y facilidad de actuali-

zación (añadir/borrar/modificar) cuando los datos son:

a) Actualizados poco frecuentemente y accedidos frecuentemente en orden aleatorio.
b) Actualizados frecuentemente y accedidos en su totalidad de manera relativamente frecuente.
c) Actualizados frecuentemente y accedidos frecuentemente y de forma aleatoria.

11.4 Los directorios pueden implementarse como "archivos especiales" a los que sólo se puede acceder de manera limitada o como archivos ordinarios de datos. ¿Cuáles son las ventajas y desventajas de cada procedimiento?

11.5 Algunos sistemas operativos poseen un sistema de archivos estructurado en árbol, pero limitan la profundidad del árbol a un pequeño número de niveles. ¿Qué efecto causa este límite en los usuarios? ¿Cómo simplifica esto el diseño del sistema de archivos (si lo hace)?

11.6 Discuta los pros y los contras de la protección y compartición de archivos basándose en la noción de grupos de usuarios frente a listas de acceso, donde los usuarios individuales reciben privilegios de acceso específicos a archivos. Tenga en cuenta en la respuesta la complejidad de la implementación, facilidad de uso. flexibilidad y seguridad.

11.7 Los tipos de organizaciones de archivos (pilas, secuenciales, secuenciales indexados. indexados y de dispersión) determinan la organización lógica. El método de asignación de almacenamiento (contiguo, encadenado, indexado) determina la organización física del archivo en el soporte. ¿Qué combinaciones de organizaciones lógica y física funcionan bien juntas y cuáles no?

11.8 Comparar los resultados de la asignación de espacio en un sistema de archivos con la asignación de memoria principal. ¿Cuáles son las similaridades y las diferencias?

11.9 ¿Cuáles son las ventajas y desventajas del intercalado de discos, independientemente del tipo de intercalado empleado?

11.10 Determinados sistemas ofrecen archivos compartidos. permitiendo a varios usuarios acceder simultáneamente a una única copia de un archivo. Otros sistemas proporcionan una copia del archivo compartido a cada usuario. Discutir las ventajas y desventajas de ambos métodos.

11.11 Un organismo oficial desea un sistema de información computarizado para ayudarles en la preparación de varios informes estadísticos. Los datos que quieren almacenar constan de información sobre los empleados de las compañías de los EE.UU. y los registros de los archivos contendrán los datos siguientes:

Tamaño de los Datos	Descripción de los datos
4 bytes	Número de identificación del patrón
4 bytes	Número de la seguridad social del empleado
4 bytes	Sueldo
4 bytes	Descripción del puesto (codificado)
1 byte	Sexo
1 byte	Edad
1 byte	Estado civil
1 byte	Origen étnico

El disco de su sistema de computador dispone de 10.000 bloques, cada uno de los cuales puede alojar 4.000 bytes. Cada registro contiene 20 bytes y todo el archivo contendrá cerca de un millón de registros.

Al archivo se accederá ocasionalmente para actualizarlo. Esto incluye añadir nuevos registros y borrar o modificar los existentes. El uso principal del archivo será para análisis estadísticos; estos accesos se ha previsto que sean casi 50 veces más frecuentes que las actualizaciones.

Las peticiones de estadísticas pueden hacer uso de todos o algunos de los campos antes mencionados, excepto del número de seguridad social; a continuación se dan varios ejemplos:

Encontrar el sueldo más alto y el más bajo de todas las mujeres caucasianas que trabajen para el patrón número 22452.

Encontrar la edad media de todos los empleados casados que ganen menos de 10.000 dólares.

Imprimir una tabla que muestre cuántos trabajadores que desempeñan el puesto número 4522 son varones o mujeres, cuántos están solteros o casados y cuántos están en varias categorías de edad.

Se pide diseñar un método de almacenamiento de este archivo en el disco de forma que las operaciones descritas arriba puedan ser realizadas lo más eficientemente posible. Se debe describir completamente la estructura del sistema de archivos, incluyendo la(s) estructura(s) del registro, corno seguir la pista de qué bloques del disco pertenecen al (los) archivo(s) y su orden, cómo llevar la cuenta de los bloques sin usar y qué métodos(s) se usarían para acceder al (los) archivo(s). No hay por qué limitarse sólo a estas cuestiones.

11.12 En la última década, la capacidad de almacenamiento de los discos se ha incrementado en casi un orden de magnitud, pero la velocidad de transferencia no ha crecido de manera similar. ¿Qué efecto supone que tendrá en las organizaciones de los archivos y en los esquemas de asignación de almacenamiento?

CAPÍTULO 12

Redes y proceso distribuido

Dada la creciente disponibilidad de computadores personales y minicomputadores económicas pero potentes, se ha producido una tendencia al alza del procesamiento de datos distribuidos (DDP). El uso de DDP permite dispersar los procesadores, datos y otros elementos del sistema dentro de una organización. La dispersión ofrece un sistema más sensible a las necesidades del usuario, capaz de ofrecer tiempos de respuesta mejores y de minimizar los costes de las comunicaciones, en comparación con los métodos centralizados. Un sistema DDP trae consigo una división de las funciones de computación y puede venir acompañado por una organización distribuida de las bases de datos, el control de los dispositivos y el control de las interacciones (en la red).

En muchas organizaciones existe una dependencia muy fuerte en computadores personales conectados a una gran instalación central de proceso de datos. Los computadores personales se vienen usando para respaldar toda una variedad de aplicaciones fáciles de utilizar, como proceso de textos, hojas de cálculo y gráficas de presentación. El computador central *(mainframe)* alojará las bases de datos corporativas junto a sistemas sofisticados de gestión de bases de datos y sistemas de información. Se necesitará enlazar los computadores personales entre si' y con el computador central. Varios métodos son de uso común para el enlace de los micros con el computador central, desde tratar al computador persona! como un simple terminal hasta conseguir un alto grado de integración entre las aplicaciones de los computadores personales y las bases de datos del computador central.

La tendencia en estas aplicaciones ha sido apoyada por la evolución de las capacidades distribuidas del sistema operativo y de las utilidades de soporte. Se ha explorado un amplio espectro de posibilidades, como se expone a continuación:

- *Arquitectura de Comunicaciones:* El software que soporta una red de computadores independientes. Ofrece respaldo a las aplicaciones distribuidas, como correo electrónico, transferencia de archivos y acceso remoto de terminales. Sin embargo, los computadores conservan una identidad distintiva para los usuarios y las aplicaciones, quienes deben comunicarse con otros computadores haciendo referencias explícitas. Cada computador posee su propio sistema operativo y será posible disponer de una combinación heterogénea de computadores y sistemas operativos, siempre que todas las máquinas utilicen la misma arquitectura de comunicaciones.

- *Sistema Operativo de Red:* Es la configuración en que existe una red de máquinas de aplicación, generalmente puestos de trabajo *(workstations)* monousuario y una o más máquinas "servidoras". Las máquinas servidoras ofrecen servicios de red o aplicaciones, como almacenamiento de archivos y gestión de impresoras. Cada computador posee su propio sistema operativo. El sistema operativo de red es simplemente un añadido al sistema operativo local que permite a las máquinas de aplicación interactuar con los servidores. El usuario siempre es consciente de que existen múltiples computadores independientes y opera con ellas explícitamente. Normalmente, se usa una arquitectura común de comunicaciones para dar soporte a estas aplicaciones de la red.
- *Sistemas operativos distribuidos:* Es un sistema operativo común compartido por una red de computadores. Aparece ante los usuarios como un sistema operativo centralizado ordinario pero que ofrece al usuario un acceso transparente a los recursos de un conjunto de máquinas. Un sistema operativo distribuido puede descansar sobre una arquitectura de comunicaciones para las funciones básicas de comunicación; es más frecuente que se extraiga un conjunto de las funciones de comunicación y se incorporen al sistema operativo para dar mayor eficacia.

La tecnología de las arquitecturas de comunicación está muy desarrollada y la soportan todos los fabricantes. La primera arquitectura de comunicaciones disponible comercialmente fue la SNA *(System Network Architecture)* de **IBM,** presentada en 1974. Los sistemas operativos de red constituyen un fenómeno mucho más reciente, pero existen bastantes productos comerciales. El área puntera de investigación y desarrollo en sistemas distribuidos es el de los sistemas operativos distribuidos. Aunque se han presentado algunos sistemas comerciales, los sistemas operativos distribuidos del todo funcionales están todavía en fase experimental.

En este capítulo y en el siguiente, se va a llevar a cabo un estudio de este abanico de posibilidades de proceso distribuido. Se comenzara con un examen de las arquitecturas de comunicaciones, considerando los principios básicos y el conocido modelo de interconexión de sistemas abiertos (OSI). Más tarde se ofrece una visión general de la arquitectura de comunicaciones más usada e independiente de los fabricantes, la serie de protocolos TCP/IP. Seguidamente, se examinan algunos conceptos clave del software servidor. Por último, se atiende a dos técnicas de comunicación entre procesos del sistema, conocidas como paso de mensajes y llamadas a procedimientos remotos. El capítulo 13 examinará una serie de aspectos clave en los sistemas operativos distribuidos.

12.1 ARQUITECTURAS DE COMUNICACIONES

Necesidad de una Arquitectura de Comunicaciones

Cuando los computadores, terminales u otros dispositivos de proceso de datos intercambian datos, el procedimiento puede ser muy complejo. Considérese, por ejemplo, la transferencia de un archivo entre dos computadores conectados a una red. Las siguientes son algunas de las larcas normales a desarrollar:

1. El sistema de origen debe informar a la red sobre la identidad del sistema de destino deseado.

2. El sistema de origen debe determinar que el sistema de destino está listo para recibir datos.
3. La aplicación de transferencia de archivos del sistema origen debe determinar que el programa de gestión de archivos del sistema destino está preparado para aceptar y guardar el archivo de un usuario particular.
4. Si los formatos de archivo que se usan en los dos sistemas son incompatibles, uno de los dos debe llevar a cabo una función de traducción de formato.

Está claro que debe existir un alto grado de cooperación entre los dos computadores. En vez de implementar la lógica de esta cooperación como un único módulo, la tarea se dividirá en subtareas, cada una de las cuales se implementa por separado. Como ejemplo, la figura 12.1 propone la forma en que se podría implementar un servicio de transferencia de archivos. En ella se usan tres módulos. Las tareas 3 y 4 de la lista anterior las podrían realizar un módulo de transferencia de archivos. Los dos módulos de ambos sistemas intercambian archivos y órdenes. Sin embargo, mejor que hacer que el módulo de transferencia de archivos maneje los detalles de la transmisión de datos y órdenes, cada módulo de transferencia de archivos se apoyará en otro módulo que da un servicio de comunicaciones. Este módulo es responsable de asegurar que las órdenes de transferencia sean intercambiadas de forma fiable entre ambos sistemas. Entre otras cosas, este módulo se encargaría de la tarea 2. Entonces, la esencia del intercambio entre los sistemas es independiente del tipo de red que los conecta. Por tanto, en vez de atender a los detalles de la interfaz de red en el módulo de servicio de comunicaciones, parece mejor construir un tercer módulo, el de acceso a la red, que lleve a cabo la tarea 1 mediante interacción con la red.

Se va a intentar resumir las razones de los tres módulos de la figura 12.1. El módulo de transferencia de archivos contiene toda la lógica especial de la aplicación de transferencia de archivos, como la transmisión de contraseñas, órdenes y registros de archivos. Se necesita transmitir estos archivos y órdenes de forma fiable. Sin embargo, los mismos requisitos de fiabilidad son importantes para una variedad de aplicaciones (por ejemplo, correo electrónico y transferencia de documentos). Por tanto, un módulo de servicio de comunicaciones aparte, que puedan usarlo muchas aplicaciones, cumplirá con estos requisitos. El módulo de servicio de comunicaciones se ocupará de asegurar que los dos sistemas están activos y pre-

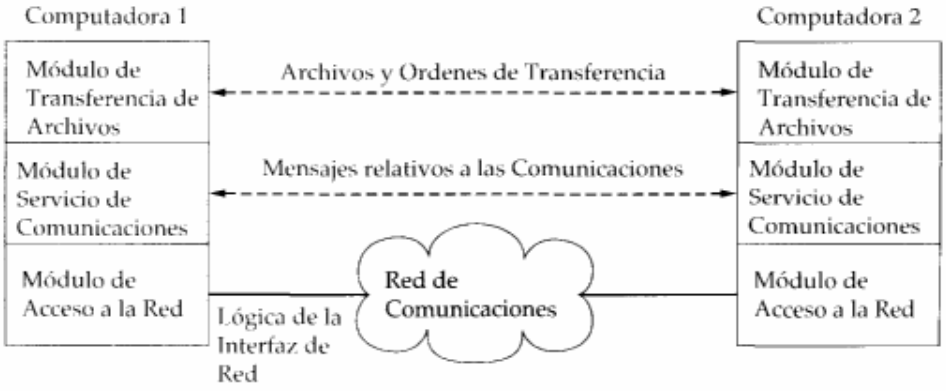


FIGURA 12.1 Una arquitectura simple para la transferencia de archivos

parados para la transferencia de datos y para guardar constancia de los datos que son intercambiados, asegurando así el envío. Esto es, la lógica de manejo real de la red queda aparte en un módulo separado de acceso a la red. De esta manera, si se cambia la red usada, sólo el módulo de acceso a la red se ve afectado.

De este modo, en vez de tener un único módulo que lleve a cabo las comunicaciones, se dispone de un conjunto estructural de módulos que implementa las funciones de la comunicación. Dicha estructura se conoce como **arquitectura de comunicaciones.** En el resto de esta sección se generaliza el ejemplo anterior para introducir una arquitectura de comunicaciones simplificada. A través de ella. se va a considerar un ejemplo real y mas complejo.

Una Arquitectura de Comunicaciones Simple

En términos generales, se dice que las comunicaciones implican a tres agentes: las aplicaciones. los ordenadores y las redes. Las aplicaciones que son objeto de preocupación son las aplicaciones distribuidas, en las que entra en juego el intercambio de datos entre dos sistemas: algunos ejemplos son el correo electrónico y la transferencia de archivos. Estas y otras aplicaciones se ejecutan en computadores multiprogramados, que soportan muchas aplicaciones simultáneas. Los computadores están conectadas a redes y los datos intercambiados se transfieren por la red de un computador a otra. Así, la transferencia de datos de una aplicación a otra implica obtener primero los datos del computador donde reside la aplicación y enviarlos a la aplicación deseada del otro computador.

Con estos conceptos en mente, parece natural organizar las comunicaciones en los siguientes tres niveles independientes:

- Nivel de acceso a la red
- Nivel de transporte
- Nivel de aplicación

El *nivel de acceso a la red* se ocupa del intercambio de dalos entre el computador y la red a la que está conectada. El computador que emite debe proporcionarle a la red la dirección del computador destino, de forma que la red pueda encaminar los datos al destino apropiado. El computador origen puede querer invocar ciertos servicios, como las prioridades, que podrían ser ofrecidos por la red. El software específico utilizado en este nivel dependerá del tipo de red usada; se han desarrollado diferentes estándares para la conmutación de circuitos, la conmutación de paquetes, redes de área local y otras más. Por ejemplo, X.25 es un estándar que especifica el acceso a una red de conmutación de paquetes. Así, parece razonable separar aquellas funciones que tengan que ver con el acceso a la red en un nivel separado. Haciendo esto, el resto del software de comunicaciones por encima del nivel de acceso a la red no necesita preocuparse de los detalles de la red empleada. El software del nivel superior funcionará bien sin que importe la red particular a la que está conectado el computador.

Sin importar la índole de las aplicaciones que intercambian datos, suele pedirse que los datos se intercambien de forma fiable. Es decir, estaría bien asegurarse de que lodos los datos llegan a su aplicación de destino y que lo hacen en el mismo orden en que fueron enviados. Los mecanismos para ofrecer fiabilidad son, en esencia, independientes de la naturaleza de las aplicaciones. De esta forma, parece razonable reunir estos mecanismos en un nivel común compartido por todas las aplicaciones; este nivel es conocido como *nivel de transporte.*

Finalmente, el nivel de aplicación contiene la lógica necesaria para soportar varias aplicaciones de usuario. Para cada clase diferente de aplicación, como la transferencia de archivos, se necesitará un módulo separado y particular para la misma.

En las figuras 12.2 y 12.3 se ilustra esta simple arquitectura. La figura 12.2 muestra tres computadores conectadas a una red. Cada computador dispone de software de los niveles de acceso a la red y transporte, así como software del nivel de aplicación para una o más aplicaciones. Para una comunicación con éxito, cada entidad del sistema global debe tener una dirección única. En realidad, se necesitan dos niveles de direccionamiento. Cada computador de la red debe tener una dirección única en la red para que ésta envíe los datos al computador correcto. Cada aplicación de un computador debe tener una dirección que sea única dentro de su computador, lo que permite que el nivel de transporte envíe datos a la aplicación correcta. Estas últimas direcciones se conocen como *puntos de acceso al servicio* (SAP), implicándose el hecho de que cada aplicación individual accede a los servicios del nivel de transporte.

La figura 12.3 indica la forma en que módulos del mismo nivel en computadores diferentes se comunican uno con otro: por medio de protocolos. Un *protocolo* es un conjunto de reglas o convenciones que gobiernan la forma en que dos entidades cooperan en el intercambio de datos. Una especificación de protocolo detallará las funciones de control que pueden realizarse, los formatos y códigos de control utilizados para comunicar dichas funciones y los procedimientos que ambas entidades deben seguir.

Puede recorrerse el camino de una operación simple. Supóngase que una aplicación asociada con el SAP 1 del computador A quiere enviar un mensaje a otra aplicación asociada con el SAP 2 del computador B. La aplicación de A entrega el mensaje a su nivel de transporte con instrucciones de envío al SAP 2 del computador B. El nivel de transporte entrega el mensaje al nivel de acceso a la red, que manda a la red enviar el mensaje al computador

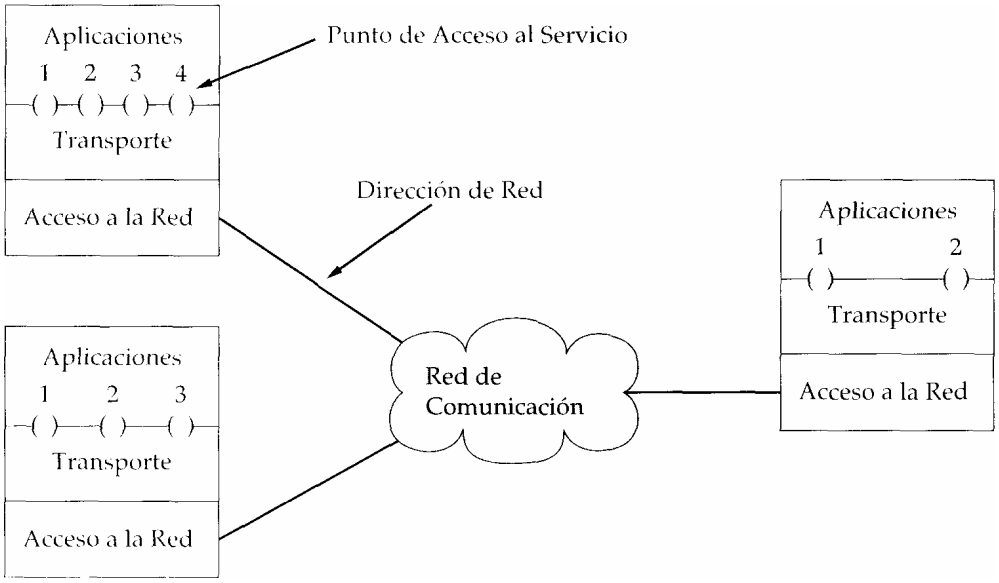


FIGURA 12.2 Redes y arquitecturas de comunicaciones

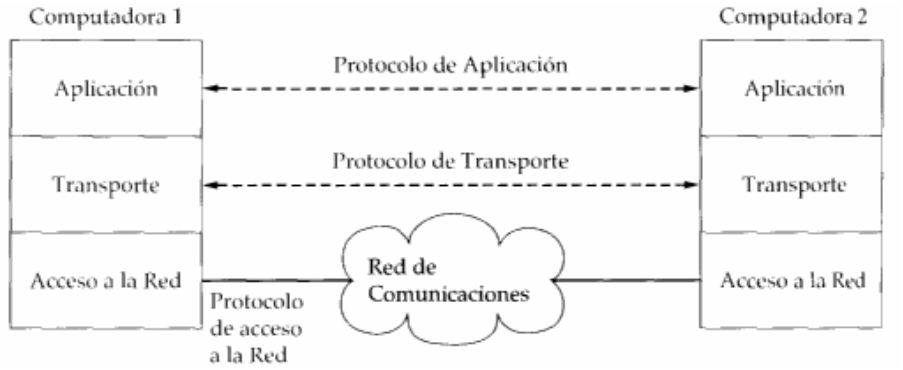


FIGURA 12.3 Protocolos de una arquitectura simplificada

B. No hace falta decirle a la red la identidad del punto de acceso al servicio de destino. Todo lo que necesita saber es que los datos son para el computador B.

Para controlar esta operación, debe transmitirse tanto información de control como datos de usuario, como se muestra en la figura 12.4. Puede decirse que la aplicación emisora genera un bloque de datos y lo pasa al nivel de transporte. El nivel de transporte puede dividir este bloque en partes más pequeñas para hacerlo más manejable. A cada una de estas partes el nivel de transporte le añade una cabecera de transporte, que contiene información de control del protocolo. La combinación de datos del nivel superior siguiente y la información de control se conoce como una unidad de datos del protocolo (PDU); en este caso se llama PDU de transporte. La cabecera de cada PDU de transporte contiene información de control a usar por el protocolo de transporte del computador B. Algunos ejemplos de elementos que pueden almacenarse en esta cabecera son los siguientes:

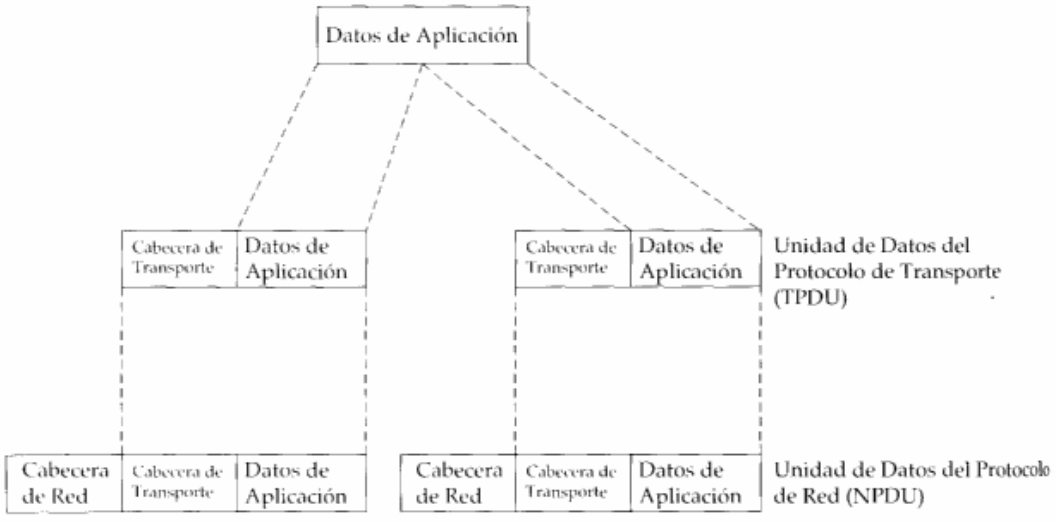


FIGURA 12.4 Unidades de datos de protocolo

- *SAP de destino:* Cuando el nivel de transporte de destino reciba la unidad de datos del protocolo de transporte, deberá saber a quien se van a enviar los datos.
- *Número de secuencia:* Como el protocolo de transporte envía una serie de unidades de datos de protocolo, las numerará secuencialmente de forma que, si llegan desordenadas, la entidad de transporte de destino pueda reordenarlos.
- *Código de detección de errores:* La entidad de transporte emisora puede calcular e insertar un código de detección de errores de forma que el receptor pueda determinar si se ha producido algún error, descartar la PDU y pedir su retransmisión.

La etapa siguiente consiste en que el nivel de transporte entregue cada PDU al nivel de red junto con instrucciones para transmitirlas al computador de destino. Para cumplir con este requisito, el protocolo de acceso a la red debe presentar los datos a la red mediante una solicitud de transmisión. Como antes, esta operación requiere el uso de información de control. En este caso, el protocolo de acceso a la red añadirá una cabecera a los datos que reciba del nivel de transporte, para crear una PDU de acceso a la red. Algunos ejemplos de elementos que pueden guardarse en esta cabecera son los siguientes:

- *Dirección del computador de destino:* La red debe saber a quién (qué computador en la red) van a enviarse los datos.
- *Solicitudes de servicio:* El protocolo de acceso a la red puede desear que la red haga uso de ciertos servicios, como las prioridades.

En la figura 12.5 se aglutinan todos estos conceptos, mostrándose la interacción entre los módulos para transferir un bloque de datos. Puede decirse que el módulo de transferencia de archivos del computador X transfiere un archivo registro a registro al computador Y. Cada registro es entregado al módulo del nivel de transporte. Esta acción se puede representar en forma de una orden o llamada al procedimiento A-ENVIAR (aplicación-enviar). Los argumentos de esta llamada incluyen la dirección del computador de destino, el punto de acceso al servicio de destino y el registro. El nivel de transporte añade al registro el punto de acceso al servicio de destino junto a otra información de control, para crear una PDU de transporte. Se entrega entonces al nivel de acceso a la red mediante una orden T-ENVIAR (transporte-enviar). En este caso, los argumentos de la orden son la dirección del computador de destino y la unidad de datos del protocolo de transporte. El nivel de acceso a la red utiliza esta información para construir una PDU de red. Supóngase que la red es de conmutación de paquetes X.25. En tal caso, la unidad de datos del protocolo de red es un paquete de datos de X.25. La PDU de transporte es el campo de datos del paquete y la cabecera del paquete incluirá el número del circuito virtual que conectará a X e Y.

La red aceptará el paquete de datos procedente de X y lo enviará a Y. El módulo de acceso a la red de Y recibirá el paquete, extraerá la cabecera y transferirá la unidad de datos adjunta del protocolo de transporte al módulo de nivel de transporte. El nivel de transporte examinará la cabecera de la unidad de datos del protocolo de transporte y, basándose en el campo SAP de destino, enviará el registro adjunto a la aplicación correspondiente, que en este caso será el módulo de transferencia de archivos de Y.

Este ejemplo merece ser estudiado más en profundidad. En el resto de esta sección, se va a examinar una arquitectura de comunicaciones más compleja. Sin embargo, dicha arquitectura se basa en los mismos principios y mecanismos del sencillo ejemplo anterior.

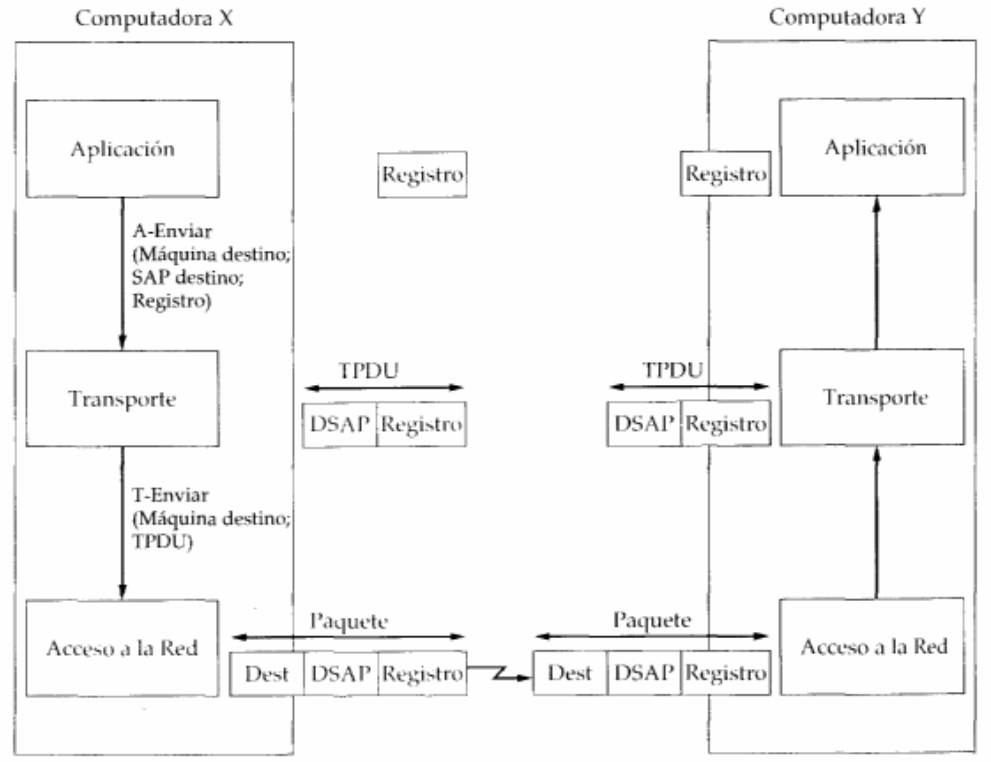


FIGURA 12.5 Operación en una arquitectura de comunicaciones

La Arquitectura OSI

Cuando se desea comunicar computadores de diferentes fabricantes, el esfuerzo en el desarrollo de software puede llegar a ser una pesadilla. Cada fabricante utiliza diferentes formatos y protocolos para el intercambio de datos. Incluso en la línea de productos de un mismo fabricante, diferentes modelos de computador pueden comunicarse de forma exclusiva.

A medida que proliferan las redes y el uso de comunicaciones en los computadores, un método de desarrollo particular del software de comunicaciones es demasiado costoso para ser aceptable. La única alternativa para los fabricantes es adoptar e implementar un conjunto común de convenios. Para que esto ocurra se necesitan estándares internacionales. De estos estándares se obtendrían dos beneficios:

- Se fomentaría que los fabricantes implementaran los estándares por la sospecha de que, debido al amplio uso de los mismos, sus productos tendrían un menor valor comercial sin ellos.
- Los clientes estarían en posición de requerir a cualquier fabricante que desee equiparlos que implemento los estándares.

Este tipo de razonamiento condujo a la Organización Internacional de Estándares (ISO) a desarrollar una arquitectura de comunicaciones conocida como el modelo de **interconexión**

de sistemas abiertos (OSI). Este modelo es un marco de definición de estándares de conexión de computadores heterogéneos. El término *abiertos* denota la capacidad de dos sistemas cualesquiera para ajustarse a la arquitectura y a los estándares asociados para comunicarse.

El Concepto de Sistema Abierto

La interconexión de sistemas abiertos se basa en el concepto de aplicaciones distribuidas co-operativas. Según el modelo OSI. un sistema consta de un computador, todo su software y cualquier dispositivo periférico conectado, terminales inclusive. Una aplicación distribuida es cualquier actividad en la que interviene el intercambio de información entre dos sistemas abiertos. Algunos ejemplos de dichas actividades son los siguientes:

- Un usuario en un terminal de un computador se conecta a una aplicación de proceso de transacciones de otro computador.
- Un programa de gestión de archivos de un computador transfiere archivos a un programa de gestión de archivos de otro computador.
- Un usuario envía un mensaje de correo electrónico a otro usuario de otro computador.
- Un programa de control de procesos envía una señal dé control a un robot.

El modelo OSI se preocupa del intercambio de información entre un par de sistemas abiertos y no del funcionamiento interno de cada sistema individual. Concretamente, se ocupa de la capacidad de los sistemas para cooperar en el intercambio de información y en el cumplimiento de tareas.

El objetivo del esfuerzo de OSI es definir un conjunto de estándares que habilitará a los sistemas abiertos ubicados en cualquier lugar del mundo para cooperar, interconectándolos mediante servicios de comunicaciones estándares y ejecutando protocolos OSI estándares.

Un sistema abierto puede implementarse de cualquier forma que esté de acuerdo con un conjunto mínimo de estándares que permitan conseguir la comunicación con otros sistemas abiertos. Un sistema abierto consta de un conjunto de aplicaciones, un sistema operativo y software del sistema, como un sistema gestor de bases de datos y un paquete de manejo de terminales. También se incluye el software de comunicaciones que convierte un sistema cerrado en un sistema abierto. Los diversos fabricantes implementarán los sistemas abiertos de formas distintas, para conseguir una identidad de producto que incremente su cuota de mercado o genere un nuevo mercado. Sin embargo, casi todos los fabricantes se comprometen a proporcionar un software de comunicaciones que se comporte de acuerdo a OSI, para ofrecer a sus clientes la capacidad de comunicarse con oíros sistemas abiertos.

El Modelo OSI

Una técnica de estructuración muy aceptada y elegida por la ISO es la división en niveles. Las funciones de comunicación se dividen en un conjunto de niveles jerárquicos. Cada nivel desempeña un subconjunto de las funciones necesarias para comunicarse con otro sistema. Cada nivel confía en que el nivel inferior adyacente realice unas funciones primitivas, oculte sus detalles y ofrezca servicios al nivel superior adyacente. En el mejor de los casos, los niveles deben definirse de forma que los cambios en uno no provoquen cambios en los otros niveles. De este modo, se ha descompuesto un problema en un conjunto de subproblemas más manejables.

La tarea de la ISO es definir una serie de niveles y servicios desempeñados por cada uno. La división debe agrupar lógicamente a las funciones y debe disponer suficientes niveles para hacer que el manejo de cada uno no sea muy complicado. Sin embargo, no deben existir tantos niveles que la sobrecarga de procesamiento impuesta por la serie de niveles sea muy onerosa. La arquitectura OSI resultante tiene siete niveles, que se encuentran definidos brevemente en la tabla 12.1.

La figura 12.6 sirve para ilustrar la arquitectura OSI. Cada computador tiene siete niveles. La comunicación se produce entre las aplicaciones de ambos computadores, etiquetadas en la figura como aplicación X y aplicación Y. Si la aplicación X quiere enviar un mensaje ala aplicación **Y,** llamará al nivel de aplicación (nivel 7). El nivel de aplicación establece una relación con el nivel 7 del computador objetivo, mediante un protocolo de nivel 7 (protocolo de aplicación). Este protocolo requiere los servicios del nivel 6, de forma que las dos entidades del nivel 6 utilicen su propio protocolo. Así se sigue hasta llegar al nivel físico, que se encarga de transmitir realmente los bits por el medio de transmisión.

TABLA 12.1 Los niveles OSI

Nivel	Definición
1 Físico	Se ocupa de la transmisión de un flujo no estructurado de bits por el enlace físico; incluye parámetros tales como el nivel de voltaje de la señal y la duración de los bits; trata con características mecánicas, eléctricas y procesales para establecer, mantener y desactivar el enlace físico.
2 Enlace de Datos	Ofrece una transferencia fiable de datos a través del enlace físico; envía bloques de datos (tramas) con la sincronización, el control de errores y de flujo necesarios.
3 Red	Ofrece a los niveles superiores independencia de las tecnologías de transmisión de datos y conmutación empleadas para conectar los sistemas; responsable de establecer, mantener y finalizar las conexiones.
4 Transporte	Ofrece transferencia fiable y transparente de datos entre los puntos finales; ofrece recuperación de errores y control de flujo.
5 Sesión	Proporciona la estructura de control para la comunicación entre las aplicaciones; establece, gestiona y finaliza las conexiones (sesiones) entre las aplicaciones cooperativas.
6 Presentación	Lleva a cabo transformaciones de datos para proporcionar una interfaz estándar de aplicaciones y ofrecer servicios de comunicación comunes; por ejemplo: cifrado, compresión de textos y reformateado.
7 Aplicación	Ofrece servicios a los usuarios del entorno OSI; por ejemplo: servidor de transacciones, servicio de transferencia de archivos, gestión de la red.

La figura 12.6 también ilustra la forma en que se llevan a cabo los protocolos de cada nivel. Cuando la aplicación X tiene un mensaje que enviar a la aplicación Y, transfiere dichos datos a un módulo de nivel de aplicación. Dicho módulo añade una cabecera de aplicación a los datos; la cabecera contiene la información de control necesaria para el mismo nivel del otro extremo. Los datos originales junto con la cabecera, conocidos como PDU de aplicación, se pasan como una unidad al nivel 6. El módulo de presentación interpreta la unidad completa como si fueran datos y les añade su propia cabecera. Este proceso continúa hasta el nivel 2, que normalmente añade una cabecera y un pie *(trailer).* Esta unidad de datos de

protocolo de nivel 2, generalmente llamada *trama,* es transmitida a través del nivel físico hasta el medio de transmisión. Cuando la trama es recibida en el computador objetivo, se produce el proceso inverso. A medida que se sube en los niveles, cada uno extrae la cabecera más externa, sigue las indicaciones de la información de protocolo contenida y pasa el resto al siguiente nivel.

Los Niveles OSI

El motivo principal del desarrollo del modelo OSI fue proporcionar un marco para la estandarización. Pueden desarrollarse uno o más protocolos estándar dentro de cada nivel del modelo. El modelo define en términos generales las funciones a realizar en cada nivel y facilita el proceso de construcción de estándares de dos maneras:

- Como las funciones de cada nivel quedan bien definidas, se pueden desarrollar estándares de forma independiente y simultánea para cada nivel. Esto acelera el proceso de construcción de estándares.
- Como las fronteras entre los niveles están bien definidas, los cambios en los estándares de un nivel no afectarán al software existente de otros niveles. Esta condición hace más fácil introducir nuevos estándares.

A continuación se hará una breve descripción de cada nivel y se discutirán algunos de los estándares que se han desarrollado para cada nivel.

Nivel Físico

El nivel físico se encarga de la interfaz física entre un dispositivo de transmisión de datos y el medio de transmisión y de las reglas con que los bits se pasan de uno a otro. El nivel tísico tiene cuatro elementos importantes:

- *Mecánicos:* Están relacionados con el punto de contacto *(demarcation).* Normalmente, es un conector con un número específico de patillas que soportan una serie de cables portadores de señal a través de la interfaz.
- *Eléctricos:* Tienen que ver con los niveles de voltaje y la temporización de los cambios en el voltaje con que se definen los bits. Estas características determinan la velocidad de los datos y las distancias que pueden alcanzarse.
- *Funcionales:* Especifican las funciones que se desempeñan mediante asignación de un significado a cada cable. Por ejemplo, uno o más circuitos pueden transportar datos, mientras que otros pueden llevar señales de control.
- *Procedimentos:* Especifican la secuencia de sucesos en la transmisión de datos. La secuencia se basa en las características funcionales. Por ejemplo, una señal de control de un extremo puede venir seguida de una señal de control del otro extremo.

Una de las interfaces físicas estándar más comunes es la RS-232-C y su siguiente versión, la EIA-232-D. Las interfaces físicas con una red de área local se especifican en un conjunto de estándares conocidos como ISO 8802.

Nivel de Enlace de Datos

El nivel físico sólo proporciona un mero flujo de bits. El nivel de enlace de datos intenta hacer el enlace físico fiable y proporciona los medios para activar, mantener y desactivar el enlace. Los principales servicios ofrecidos por el nivel de enlace de datos a los niveles supe-

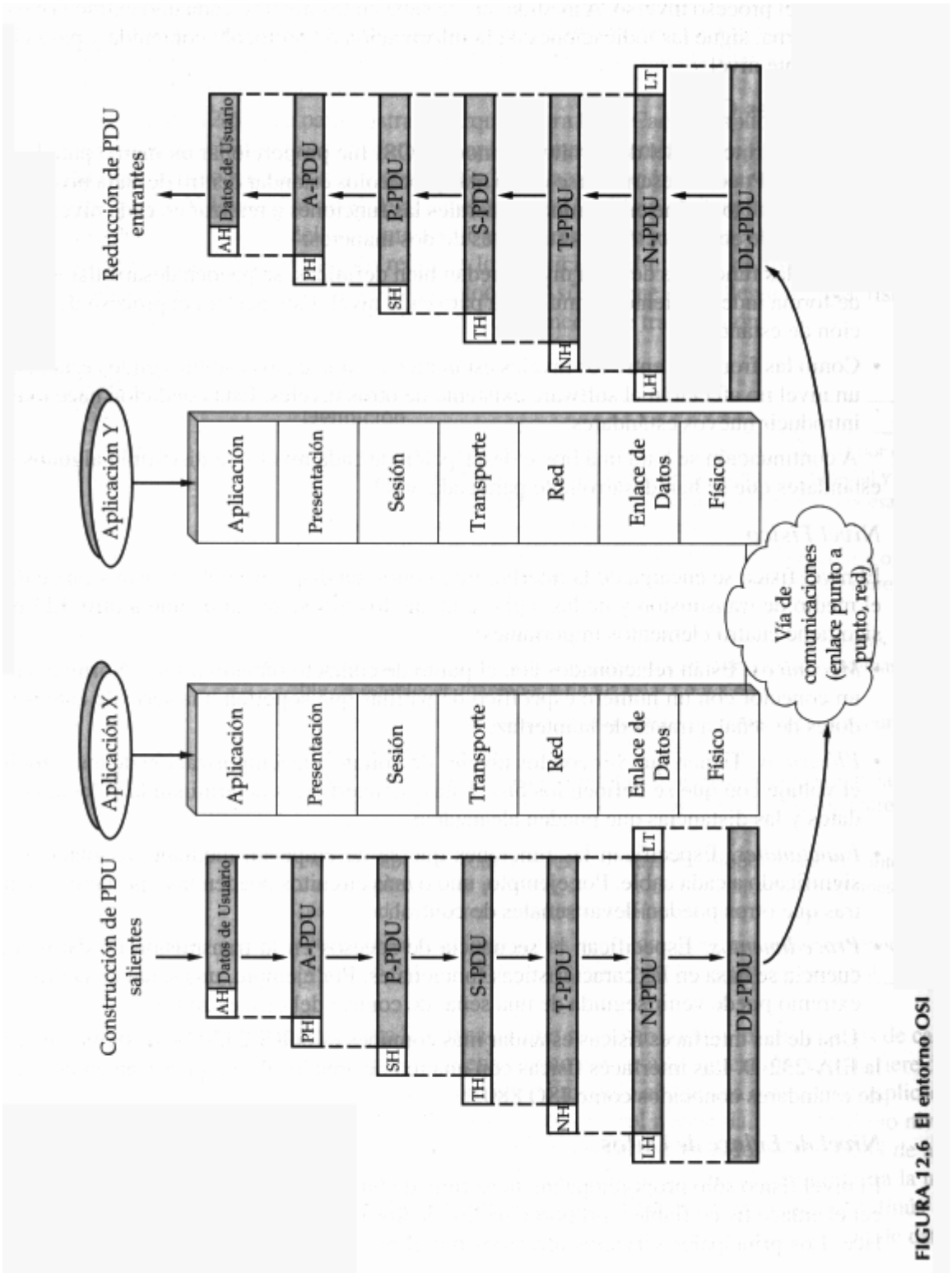


FIGURA 12.6 El entorno OSI

riores son la detección y el control de errores. De esta forma, con un protocolo del nivel de enlace de datos completamente funcional, el nivel superior adyacente puede dar por sentada una transmisión libre de errores por el enlace.

El procedimiento tomado para proporcionar fiabilidad es pasar información de control junto a los datos por el enlace. Los datos se transmiten en bloques llamados *tramas* Cada trama consta de una cabecera, un pie y un campo opcional de datos. La cabecera y el pie contienen información de control empleada en la gestión del enlace.

Como ejemplo de protocolo del nivel de enlace de datos, considérese el HDLC (control de alto nivel del enlace de datos). El HDLC es un estándar de ISO que es muy usado en los enlaces entre computadores y de computadores a terminales. La mayoría de los estándares de enlace patentados, corno el SDLC de IBM, son similares al HDLC. Además, otros estándares de control del enlace de datos, como los de las redes de área local, están basados en HDLC.

En HDLC se definen tres modos de operación, tal y como sigue:

Modo de respuesta normal: Empleado para enlaces punto a punto (enlaces que conectan solo dos puestos) y también para enlaces multipunto. Estos últimos están representados por una línea que conecta un conjunto de terminales a un computador. Hay un puesto principal y uno o más secundarios. El puesto principal es responsable de iniciar la actividad, controlar el flujo de datos con los puestos secundarios, recuperarse de los errores y desconectar lógicamente los puestos secundarios. Un puesto secundario puede transmitir solo como respuesta a un muestreo desde el principal. Este modo se adapta perfectamente para un computador central que dispone de una serie de terminales.

- *Modo de respuesta asíncrono:* Es similar al modo de respuesta normal, pero ahora los puestos secundarios pueden iniciar una transmisión sin ser muestreados por el puesto principal. Este modo es útil en cierto tipo de configuraciones cíclicas.
- *Mode de respuesta equilibrado:* Solamente empleado en enlaces punto a punto. Cada puesto asurne el papel tanto de principal como de secundarlo. Este modo es más eficaz para enlaces punto a punto porque no tiene lugar la sobrecarga del muestreo y ambos puestos pueden iniciar la transmisión.

Los datos se transmiten en tramas que constan de seis campos (figura 12.7a):

- *Indicador (flag):* Usado para sincronización. Este campo aparece al principio y al final de cada trama.
- *Dirección:* indica el puesto secundario de la transmisión. Se necesita en el caso de una línea multipunto: el principal puede emitir a alguno de los secundarios y uno de los secundarios puede emitir al principal.
- *Control:* Identifica la función de la trama. Se describirá posteriormente.
- *Información:* Este campo contiene los datos de usuario proporcionados por el nivel superior adyacentes.
- *CRC:* El código de redundancia cíclica es una función de los contenidos de los campos de dirección, control y datos. Es generado por el emisor y por el receptor. Si el resultado del receptor difiere del campo CRC recibido, se ha producido un error en la transmisión.
- *Indicador:* como se dijo en el primer elemento, este campo aparece al final de cada trama.

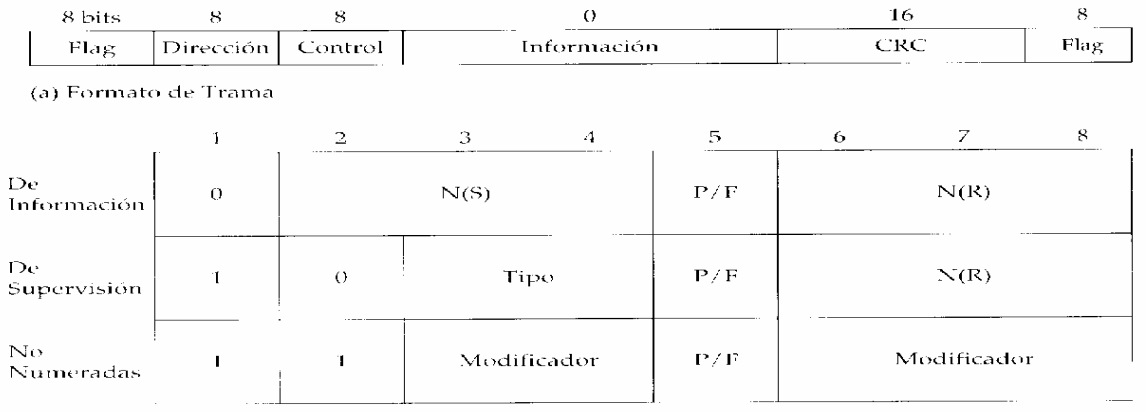


FIGURA 12.7 Estructura de la trama de HDLC

Se utilizan tres tipos de tramas, cada una con un formato diferente para el campo de control (figura 12.7b). Las tramas de información llevan datos de usuario. Las tramas de supervisión ofrecen funciones básicas de control del enlace y las tramas no numeradas ofrecen funciones adicionales de control de enlace.

El bit P/F se usa en el puesto primario para solicitar una respuesta (muestreo). Como respuesta se puede enviar más de una trama, con el bit P/F activado para marcar la última trama (final).

Los campos N(S) y N(R) de la trama de información proporcionan una técnica eficiente tanto para el control de flujo como de errores. Cada puesto numera secuencialmente y en módulo 8 las tramas de información que envía, mediante el campo N(S) (Números de secuencia de emisión). Cuando un puesto recibe una trama de información valida, podrá acusar el recibo de dicha trama la próxima vez que envíe su propia trama de información, rellenando el campo N(R) (Número de secuencia de recepción) con el número de la siguiente trama que espera recibir. Esto se conoce como acuse de recibo a cuestas (piggybacked acknowlegment) porque el reconocimiento vuelve cargado sobre cada trama de información.

Los acuses de recibo pueden ser también enviados en las tramas de supervisión. El campo de 2 bits "tipo' refleja una de las cuatro tramas de supervisión:

• *Listo para recepción (RR):* Utilizada para acusar el recibo correcto de las tramas de información hasta la N(R) exclusive.
• *No listo para recepción (RNR):* Utilizado para indicar la condición de ocupado temporalmente. N(R) se emplea para un posible acuse de recibo redundante.
• *Rechazo (REJ):*Utilizado para indicar un error en la trama de información N(R) y para pedir su retransmisión y la de todas las siguientes.
• *Rechazo selecto (SREJ):* Utilizado para solicitar la retransmisión de una única trama de información.

El uso de números de secuencia cumple tres funciones importantes:

- *Control de flujo:* Una vez que un puesto ha enviado siete tramas, no podrá enviar más hasta que se haya acusado recibo de la primera. Esto impide al emisor abrumar al receptor.
- *Control de errores:* Si se recibe una trama errónea, un puesto puede enviar un REJ o SREJ para especificar que la tramase recibió con errores.
- *Encauzamiento* (pipelining) En un instante puede haber más de una trama en tránsito. Esto permite el empleo eficiente de los enlaces con altos retardos de propagación, como los enlaces por satélite.

La técnica de N(S)/N(R) se conoce como *protocolo de ventana deslizante,* porque el puesto emisor mantiene una ventana de tramas a enviar que se reduce automáticamente con las transmisiones y se expande con los acuses de recibo. El proceso está representado en la figura 12.8.

Las tramas no numeradas no tienen número de secuencia, pero disponen de un campo modificador de 5 bits para indicar una variedad de funciones, como inicializar un puesto, ajustar el modo, desconectar un puesto y rechazar una orden.

Nivel de Red

El nivel de red proporciona los medios para la transferencia de información entre los computadores a través de algún tipo de red de comunicaciones. Este nivel libra a los superiores de la necesidad de conocer nada sobre la transmisión de datos subyacente y las tecnologías de conmutación empleadas en la conexión de los sistemas. El servicio de red es responsable de establecer, mantener y finalizar las conexiones con la red en cuestión. En este nivel, el sistema de computadores entra en un diálogo con la red para especificar la dirección de destino y para solicitar ciertos servicios de red, como las prioridades.

Hay un abanico de posibilidades para que el nivel de red gestione los servicios participantes en las comunicaciones. En un extremo, puede haber un enlace directo punto a punto entre los puestos. En tal caso, puede que no se necesite un nivel de red porque el nivel de enlace de datos puede realizar las funciones necesarias de gestión del enlace.

A continuación, puede que los sistemas se conecten a través de una única red, sea de conmutación de circuitos o de paquetes. Como ejemplo, el nivel de paquetes de X.25 es un estándar de nivel de red para esta situación. La figura 12.9 muestra cómo la presencia de una red afecta a la arquitectura OSI. Los ti-es niveles inferiores se preocupan de la conexión y de la comunicación con la red. Los paquetes (unidades de datos del protocolo de nivel 3) creados por el sistema de un extremo pasan a través de uno o varios nodos de la red que actúan como retransmisiones entre dos sistemas. En los nodos de la red se implementan los niveles del 1 al 3 de la arquitectura. En la figura, dos sistemas finales se conectan a través de un solo nodo de la red. El nivel 3 del nodo realiza las funciones de conmutación y encaminamiento *(routing)* En cada nodo hay dos niveles de enlace de dalos y dos niveles físicos, correspondientes a los enlaces para ambos sistemas. Cada nivel de enlace de datos (y físico) opera independientemente para ofrecer servicios al nivel de red sobre su enlace respectivo. Los cuatro niveles superiores son protocolos "de extremo a extremo" entre los computadores conectados.

En el otro extremo, dos puestos pueden querer comunicarse sin estar conectados a la misma red. Mejor dicho, estén conectados a redes que, directa o indirectamente, están co-

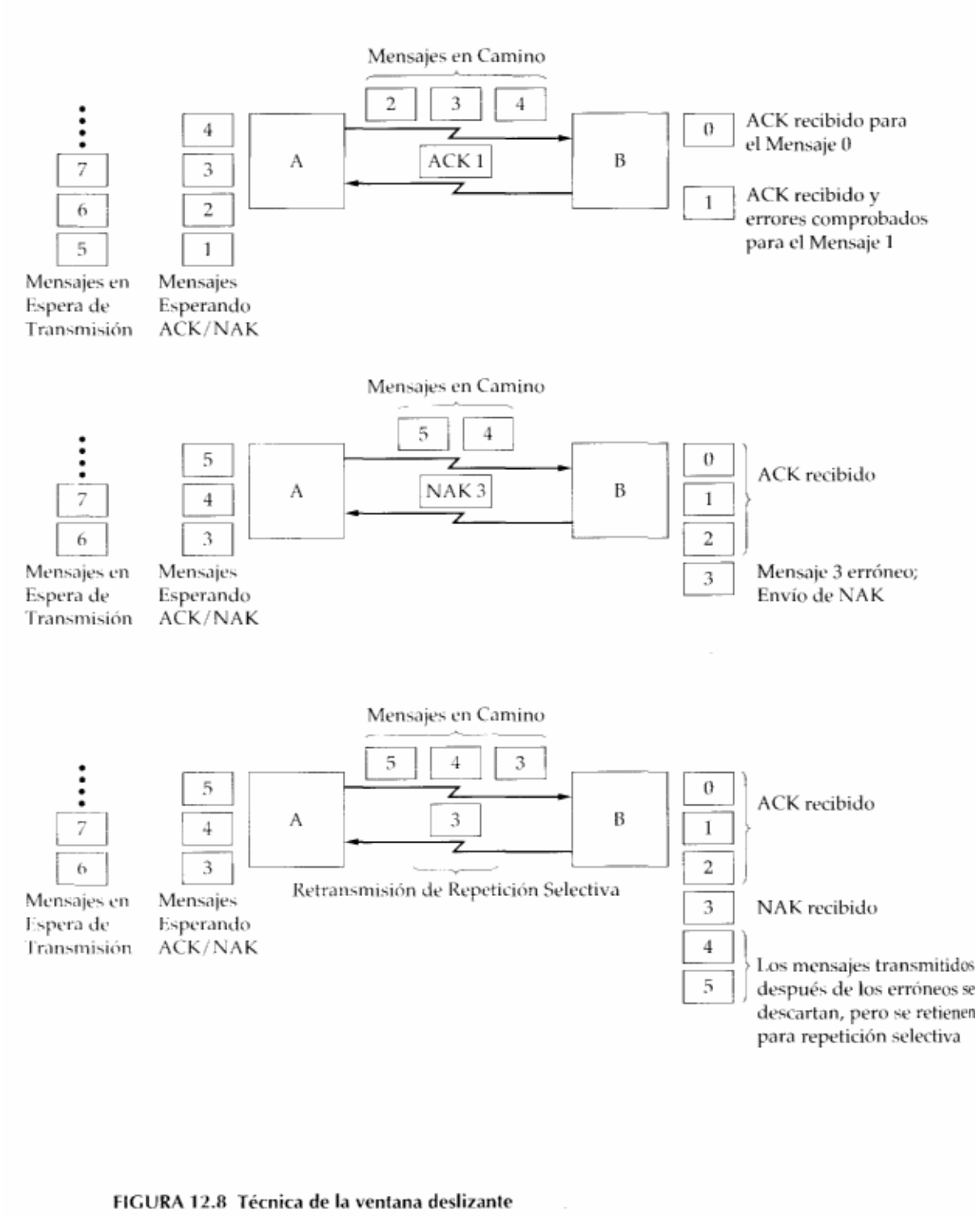


FIGURA 12.8 Técnica de la ventana deslizante

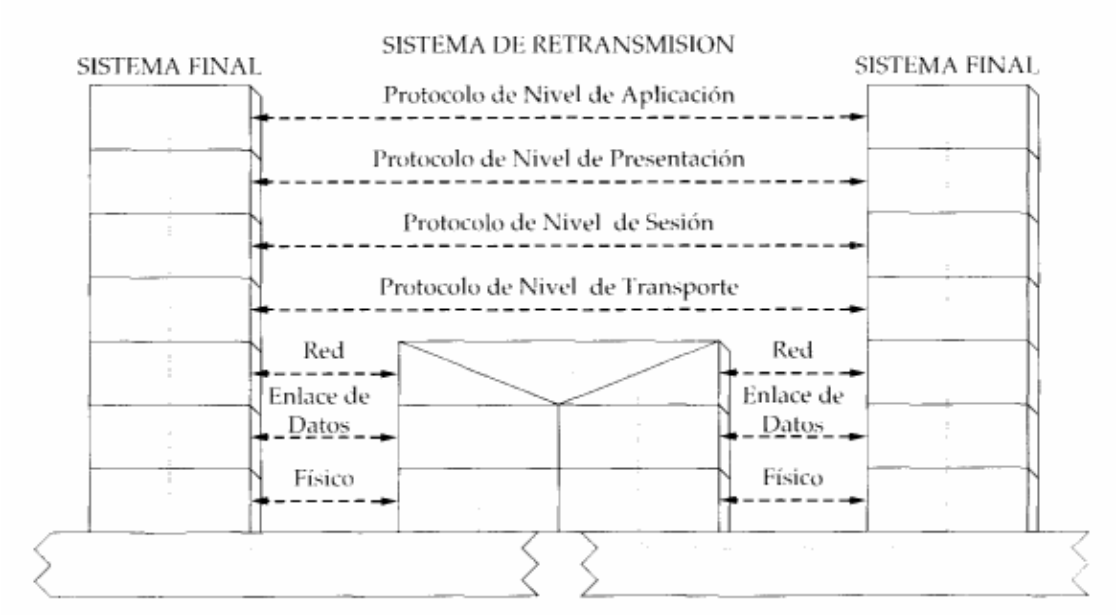


FIGURA 12.9 Utilización de un esquema de retransmisión

nectadas entre sí. Este caso requiere el uso de algún tipo de técnica de interconexión de redes. En esencia, se siguen utilizando paquetes, pero la lógica que entra en juego incluye funciones de encaminamiento entre ambas redes, así como de encaminamiento dentro de cada red.

Nivel de Transporte

El nivel de transporte ofrece un mecanismo fiable para el intercambio de datos entre computadores. Este nivel asegura que los datos son enviados sin errores, en secuencia y sin pérdidas ni duplicaciones. El nivel de transporte puede ocuparse también de optimizar el uso de los servicios de red y de satisfacer las demandas de calidad de servicio. Por ejemplo. el nivel de sesión puede especificar el retardo máximo, la prioridad y las características de seguridad.

Los mecanismos que emplea el protocolo de transporte para ofrecer fiabilidad son muy similares a los usados en los protocolos de control del enlace de datos, cono el HDLC: el empleo de números de secuencia, códigos de detección de errores y la retransmisión tras vencer los plazos de tiempo. La razón de esta duplicidad aparente del esfuerzo es que el nivel de enlace trata con un único enlace directo, mientras que el nivel de transporte se encarga de una cadena de enlaces y nodos de la red. Aunque cada enlace de la cadena es fiable por el uso de HDLC, un nodo de la cadena puede fallar en un instante crítico. Tal fallo afectará al envío de datos y será el protocolo de transporte el que se encargue del problema.

El tamaño y la complejidad de un protocolo de transporte depende de cómo de fiables sean la red subyacente y los servicios del nivel de red. De acuerdo con esto, la ISO ha desarrollado una familia de cinco estándares de protocolos de transporte, cada uno orientado a un servicio básico diferente.

Nivel de Sesión

El nivel de sesión ofrece los mecanismos para controlar el diálogo entre dos sistemas finales. En muchos casos, hay una necesidad escasa o nula de nivel de sesión, pero algunas aplicaciones utilizarán servicios tales. Los servicios clave ofrecidos por el nivel de sesión incluyen los siguientes:

• *Disciplina de diálogo:* Éste puede ser simultáneo en ambos sentidos *(full duplex)* o alterno en cada sentido (*half dulplex).*

• *Agrupación:* El flujo de datos puede ser marcado para definir grupos de datos. Por ejemplo, si un almacén detallista transmite datos de ventas a una oficina regional, los datos pueden etiquetarse para indicar el final de los datos de ventas de un departamento, indicando así al computador central que puede calcular los totales para dicho departamento y comenzar una nueva cuenta para el departamento siguiente.

• *Recuperación:* El nivel de sesión puede ofrecer un mecanismo de puntos de control, de forma que si se produce un fallo de algún tipo entre dos puntos de control, la entidad de sesión pueda retransmitir todos los datos desde el último control.

La ISO ha publicado un estándar para el nivel de sesión que incluye como opcionales unos servicios como los descritos.

Nivel de Presentación

El nivel de presentación define el formato de los datos a intercambiar entre las aplicaciones y ofrece a los programas de aplicación un conjunto de servicios de transformación de datos. Por ejemplo, la compresión de datos o el cifrado pueden realizarse en este nivel.

Nivel de Aplicación

El nivel de aplicación proporciona a los programas de aplicación un medio para acceder al entorno OSI. Este nivel contiene funciones de gestión y mecanismos útiles en general para respaldar a las aplicaciones distribuidas. Además, en este nivel se considera que residen las aplicaciones de propósito general, como la transferencia de archivos, el correo electrónico y el acceso por terminal a computadores remotas.

12.2 LA SERIE DE PROTOCOLOS TCP/IP

Gran parte de las labores de estudio y desarrollo realizadas en los protocolos de comunicación que aparecen en la literatura han utilizado la terminología y el marco de referencia del modelo OSI. Hasta hace poco, se ha prestado una menor atención a una arquitectura de comunicaciones anterior al modelo OSI y para la que existe mucha más experiencia práctica y de implementación. Esta arquitectura es el resultado de la investigación y el desarrollo de protocolos llevados a cabo en la red experimental de conmutación de paquetes ARPANET, financiada por la Agencia de Proyectos de Investigación Avanzados para la Defensa (DARPA) y conocida como la serie de protocolos **TCP/IP.**

Esta serie de protocolos consta de un gran conjunto de protocolos que han sido publicados cono los estándares de internet por el Consejo de Actividades de Internet (IAB). Los proto-

colos mejor conocidos y más usados de esta serie han sido publicados como un estándar militar por el Departamento de Defensa de los EE.UU. (tabla 12.2).

La arquitectura TCP/IP, al igual que el modelo OSI, está estructurada en niveles. En el caso del TCP/IP, hay implicados cuatro niveles: acceso a la red, internet, transporte y aplicación. En la figura 12.10 se compara esta arquitectura con la de OSI. En el nivel de acceso a la red se incluyen los protocolos que ofrecen acceso a la red de comunicaciones, como puede ser una red de área local (LAN), Los protocolos de este nivel se establecen entre un nodo de comunicaciones y un computador conectado. La serie TCP/IP no incluye un único protocolo en este nivel, sino que se usa el protocolo adecuado para cada red específica (por ejemplo, Ethernet, IEEE 802 o X.25).

El nivel de internet consta de los procedimientos necesarios para que los datos puedan atravesar múltiples redes entre computadores. De esta manera, debe ofrecerse una función de encaminamiento. El protocolo se implementa en computadores y en encaminadores (routers). Un encaminador es un procesador que conecta dos redes y cuya función principal es retransmitir los datos entre las redes. Usando un protocolo de interconexión de redes. El protocolo de este nivel se conoce como **protocolo de internet (IP).** Un uso típico de IP es conectar múltiples LAN dentro de un mismo edificio o conectar varias LAN de sitios distintos mediante una red de área extensa de conmutación de paquetes.

El nivel de transporte proporciona la lógica necesaria para asegurar que los datos intercambiados entre los computadores son enviados de forma fiable. También es responsable de la llegada directa de datos a la aplicación en cuestión. El protocolo de este nivel es el **protocolo de control de la transmisión (TCP).**

TABLA 12.2 La serie de protocolos TCP/IP

MIL-STD-1777 Protocolo de internet (IP)
Ofrece un servicio sin conexión para que los sistemas finales se comuniquen a través de una o varias redes. No supone que la red sea fiable.

MIL- STD-1778 Protocolo de Control de la Transmisión (TCP)
Servicio de transferencia fiable de un extremo a otro. Equivalente al protocolo de transporte de OSI.

MI L-STD-1 780 Protocolo de Transferencia de Archivos (FTP)
Una aplicación simple para la transferencia de archivos ASCII, EBCDIC y binarios.

MIL-STD-1781 Protocolo Simple Transferencia de Correo (SMTP)
Servicio simple de correo electrónico.

MIL-STD-1782 TELNET
Proporciona un servicio de conexión remota para terminales

Para finalizar, en el nivel de aplicación se incluyen protocolos para aplicaciones específicas del usuario. Para cada tipo diferente de aplicación, como la transferencia de archivos, se necesita un protocolo que respalde a la aplicación. Los tres protocolos de este tipo más usados de la serie TCP/IP son SMTP, FTP y TELNET, descritos a continuación.

Operaciones de TCP e IP

En la figura 12.11 se indica la forma en que están configurados estos protocolos para comunicarse. Para que quede claro que el servicio global de comunicaciones puede constar de múlti-

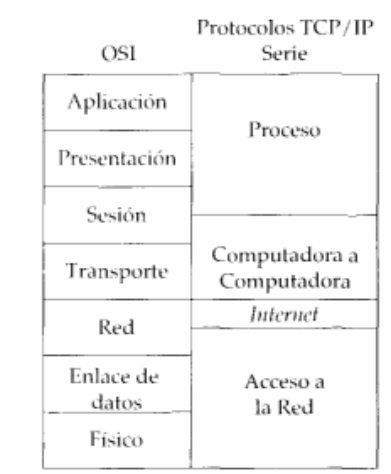


FIGURA 12.10 Comparación de las arquitecturas de comunicaciones OSI y TCP/IP

ples redes, las redes integrantes se conocen en general como subredes. Se utilizará algún tipo de protocolo de acceso a la red, como puede ser la lógica de Ethemet, para conectar un computador a una subred. Este protocolo sirve para que el computador envíe datos por la subred hacia otro computador o, en el caso de un computador de otra subred, hacia un encaminador. IP está implementado en todos los sistemas finales y encaminadores. Actúa como un relé que mueve un bloque de datos ele un computador, a través de uno o varios encaminadores, hacia otro computador. TCP está implementado solo en los sistemas finales; debe seguir la pista a los bloques de datos para asegurar que todos se envían de manera fiable a la aplicación pertinente.

Para una comunicación con éxito, cada entidad del sistema global debe tener una dirección única. En realidad, se necesitan dos niveles de direcciones. Cada computador de una subred debe tener una dirección global de internet que sea única; esto hace que los datos se envíen a la máquina adecuada. Cada proceso dentro ele un computador debe tener una dirección que sea única dentro del computador; esto permite al protocolo entre máquinas (TCP) enviar lo datos al proceso correcto. Estas últimas direcciones se conocen como puertos.

Puede realizarse un seguimiento de una operación simple. Supóngase que un proceso, ase ciado con el puerto 1 del computador A, quiere enviar un mensaje a otro proceso, asociado con el puerto 2 del computador B. El proceso de A entregará el mensaje al nivel TCP con instrucciones para enviarlo al puerto 2 de la máquina B. TCP entrega el mensaje a IP con instrucciones para enviarlo al computador B. Nótese que IP no necesita conocer la identidad del puerto de destino. Todo lo que necesita saber es que los datos son para la máquina *B.* Seguidamente, IP entrega el mensaje al nivel de acceso a la red (por ejemplo, Ethernet) con instrucciones para enviarlo al encaminador X (el primer salto en el camino hacia *B).*

Para controlar esta operación, se debe transmitir información de control junto con los datos ele usuario, como se propone en la figura 12.12. Puede decirse que el proceso emisor genera un bloque de datos y lo pasa a TCP. TCP puede dividir este bloque en trozos más pequeños para hacerlo más manejable. TCP añade una información de controla cada uno de

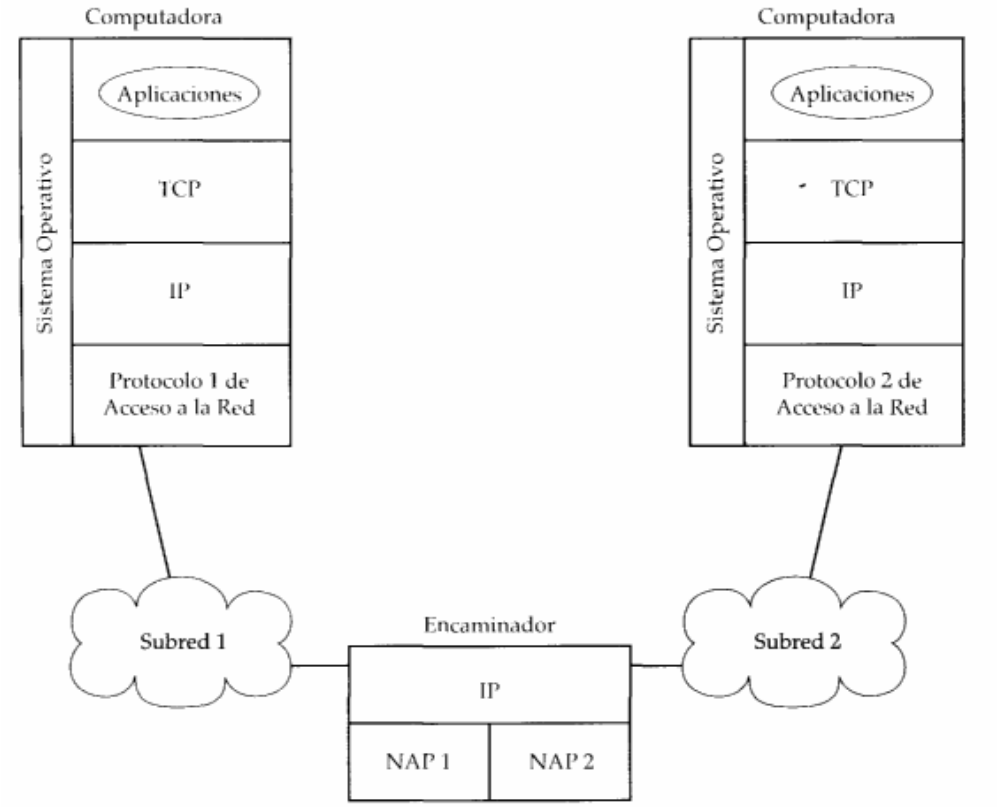


FIGURA 12.11 Comunicaciones en la arquitectura de los protocolos TCP/IP

estos trozos, conocida como la cabecera TCP, para formar un *segmento TCP*. La entidad pareja del protocolo TCP en el computador *B* utilizará dicha información de control. Algunos ejemplos de elementos que se incluyen en esta cabecera son:

Puerto de destino: Cuando la entidad TCP de *B* recibe el segmento, debe saber a quién se van a enviar los datos.

- *Número de secuencia: TCP* numera de forma secuencial los segmentos que envía a un puerto de destino particular, de forma que, si llegan desordenados, la entidad TCP de *B* pueda reordenarlos.
- *Código de control (checksum):* El TCP emisor incluirá un código que es función del contenido de lo que queda del segmento. El TCP receptor realiza el mismo cálculo y compara el resultado con el código recibido. Si ha habido algún error en la transmisión, se producirá una discrepancia.

A continuación, TCP entrega cada segmento a IP, con instrucciones para transmitirlo a B. Estos segmentos deben transmitirse a través de una o más subredes y retransmitirse mediante uno o más encaminadores intermedios. Esta operación también requiere el uso de información de control. De esta forma, IP añade una cabecera de información de control a

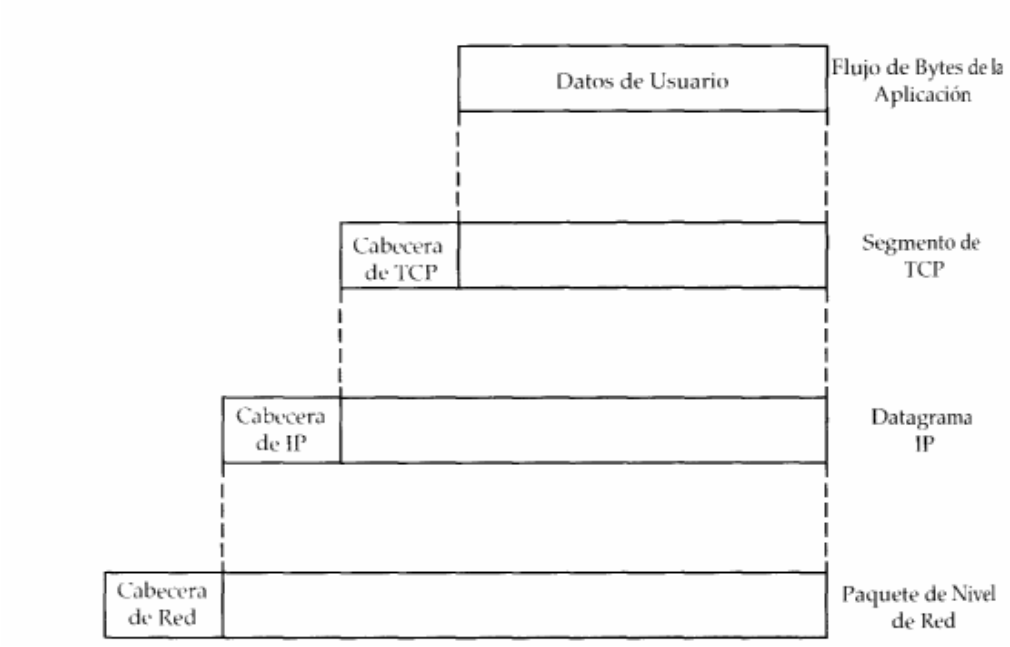


FIGURA 12.12 Unidades de datos de protocolo en la arquitectura TCP/IP

cada segmento para formar un *datagrama* lP. Un ejemplo de elemento guardado en la cabecera IP es la dirección del computador de destino (en nuestro ejemplo, B).

Para finalizar, se introduce cada datagrama IP en el nivel de acceso a la red para su transmisión por la primera subred, en su viaje hacia el destino. El nivel de acceso a la red añade su propia cabecera, construyendo un paquete o trama. El paquete es transmitido a través de la subred hacia el encaminador X. La cabecera del paquete contiene la información que necesita la subred para transmitir los datos por ella. Algunos ejemplos de elementos que pueden incluirse en esta cabecera son:

- *Dirección de la subred de destino:* La subred debe saber a qué dispositivo conectado se va a enviar el paquete.
- *Solicitudes de Servicios:* El protocolo de acceso a la red podría solicitar el uso de ciertos servicios de la subred, como las prioridades.

En el encaminador X, se extraerá la cabecera del paquete y se examinará la cabecera IP. A partir de la información de dirección de destino en la cabecera IP, el módulo IP del encaminador dirigirá el datagrama, a través de la subred 2, hacia *B.* Para hacer esto, se añade al datagrama una cabecera de acceso a la red.

Cuando se reciben los datos en *B,* se produce el proceso inverso. En cada nivel, se elimina la cabecera correspondiente y el resto es pasado al nivel superior adyacente, hasta que los datos originales del usuario se hayan enviado al proceso de destino.

Aplicaciones TCP/IP

El protocolo simple de transferencia de correo (SMTP) ofrece un servicio básico de correo electrónico. Proporciona un mecanismo de transferencia de mensajes entre computado

res separados. Las características de SMTP incluyen listas de envíos, acuses de recibo y el envío progresivo *(forwarding).* El protocolo SMTP no especifica la forma en que tienen que crearse los mensajes; se necesita algún editor local o un servicio sencillo de correo electrónico. Una vez que se ha creado el mensaje, SMTP lo acepta y hace uso de TCP para enviarlo al módulo SMTP de otro computador. El módulo SMTP de destino empleará un paquete local de correo electrónico para guardar el mensaje llegado en un buzón de usuario.

El protocolo de transferencia de archivos (FTP) se utiliza para enviar archivos (le un sistema a otro, mediante órdenes del usuario. Hay cabida tanto para archivos de texto como binarios y el protocolo ofrece características para controlar el acceso de los usuarios. Cuando un usuario desea transferir un archivo, FTP establece una conexión TCP con el sistema objetivo para el intercambio de mensajes de control. Estos sirven para transmitir identificadores de usuario y contraseñas y permiten al usuario especificar el archivo y las acciones deseadas. Una vez que se ha establecido el contacto, se establece una segunda conexión para la transferencia de datos. El archivo se transfiere por la conexión de datos, sin la sobrecarga de las cabeceras o de la información de control del nivel de aplicación. Cuando se complete la transferencia, la conexión de control se utilizará para indicar la finalización y aceptar nuevas órdenes de transferencia.

TELNET ofrece una capacidad de conexión remota, que permite a los usuarios de un terminal o un computador personal conectarse a un computador remota y funcionar como si estuviesen conectados directamente a la misma. El protocolo se diseñó para trabajar con terminales simples de desplazamiento *(scroll-mode).* TELNET está implementado en realidad en dos módulos: El TELNET de Usuario interactúa con el módulo de E/S del terminal para comunicarse con un terminal local. Convierte las características específicas de cada terminal real a un estándar de red y viceversa. El TELNET Servidor interactúa con una aplicación, funcionando como un manejador suplente de terminales de forma que los terminales remotos parezcan locales a la aplicación. El tráfico del terminal entre el TELNET de Usuario y el Servidor se transporta en una conexión TCP.

12.3 PROCESO CLIENTE/SERVIDOR

Posiblemente, la tendencia más significativa en los sistemas de información de los últimos años ha sido la subida del proceso cliente/servidor. Este modelo de procesamiento está reemplazando a una gran velocidad a los métodos de procesamiento dominados por computadores centrales, el proceso centralizado y otras formas alternativas de proceso distribuido de datos.

Esta sección comienza con una descripción de la esencia general del proceso cliente/servidor. Se sigue con una discusión sobre las formas alternativas de organizar las funciones cliente/servidor. Después se examina el tema de la consistencia de la cache de archivos, de relieve debido al uso de servidores de archivos. Para finalizar, se introduce el concepto de *middleware.*

¿Qué es el Proceso Cliente/Servidor?

El proceso cliente/servidor ha llegado de una forma definitiva. Aunque la "revolución" varía dependiendo del fabricante, con todo existe un ligero acuerdo sobre en qué consiste el

proceso cliente/servidor. En la tabla 12.3 se citan algunas definiciones propuestas para el concepto de cliente/servidor. Merece la pena leer estas definiciones cuidadosamente. De todas ellas emergen algunas ideas generales.

En la figura 12.13 se intenta capturar lo esencial de estas ideas. Como el término sugiere, un entorno cliente/servidor estará poblado de clientes y servidores. Las máquinas cliente serán, en general, PC monousuario o puestos de trabajo que ofrezcan una interfaz muy fácil de usar para el usuario final. Los puestos cliente presentan, en general, un tipo de interfaz gráfica que sea cómoda para los usuarios, incluyendo el uso de ventanas y un ratón. Algunos ejemplos comunes de tal tipo de interfaces son las ofrecidas por Microsoft Windows y Macintosh. Las aplicaciones cliente están confeccionadas para su facilidad de uso e incluyen herramientas tan familiares como puedan ser las hojas de cálculo.

TABLA 12.3 Definiciones del Proceso Cliente/Servidor

Proceso cliente/servidor: Significa dividir una aplicación en tareas y poner cada tarea en la plataforma donde pueda ser manejada más eficazmente. Esto suele significar que se sitúe en la máquina del usuario el proceso necesario para la presentación y, en el servidor, la gestión y el almacenamiento de los datos. Dependiendo de la aplicación y del software empleado, todo el tratamiento de los datos puede tener lugar en el cliente o repartirse entre cliente y servidor. El servidor se conecta a sus clientes a través de una red. El software servidor acepta peticiones de datos del software cliente y devuelve los resultados al cliente. El cliente manipula los datos y presenta los resultados al usuario [DEW193].
Cliente/servidor: Un entorno de trabajo en red donde el control de los datos está radicado en un nodo servidor y se halla disponible para que los otros nodos accedan, pero no para su actualización [INMO91].
Proceso cliente/servidor: Implica la existencia de un procesamiento cooperativo de solicitudes emitidas por un cliente o demandante a un servidor que las procesa y devuelve los resultados al cliente. En este modelo, el proceso de las aplicaciones se divide (aunque no es obligatorio) entre cliente y servidor. El proceso se inicia en realidad en el cliente y es controlado parcialmente por éste, pero no de una forma maestro-esclavo. Por el contrario, tanto cliente como servidor cooperan para ejecutar con éxito una aplicación [BERS92].
Cliente/servidor: Un modelo de interacción entre procesos de software que ejecutan concurrentemente. Los procesos cliente envían solicitudes a un proceso servidor que les responde con los resultados respectivos. Como se deduce del nombre, los procesos servidores ofrecen servicios a sus clientes, generalmente por medio de un proceso específico que sólo ellos pueden realizar. El proceso cliente, libre de la complejidad y la sobrecarga de, procesar la transacción, puede realizar otras tareas útiles. La interacción entre el proceso cliente y el servidor es un intercambio cooperativo y transaccional en el que el cliente es activo y el servidor es receptivo [RENA93].
Proceso cliente/servidor: Cualquier aplicación en la que el demandante de una acción está en un sistema y el proveedor puede estar situado en otro. Además, la mayor parte de las soluciones cliente/servidor tienen un diseño "de varios a uno", es decir, mas (le un cliente hacen peticiones normalmente al servidor [MOSK93].

En un entorno cliente/servidor. cada servidor ofrece una serie de servicios compartidos a los usuarios. Actualmente, el tipo más común de servidor es el servidor de bases de datos, cine controla generalmente una base de datos relacional. El servidor permite a los clientes compartir el acceso a la misma base de datos y habilita un sistema de computación de alto rendimiento para gestionar la base de datos. Además de los clientes y servidores, el tercer ingrediente básico del entorno cliente/servidor es la red. El proceso cliente/servidor es un proceso distribuido. Los usuarios, aplicacio-

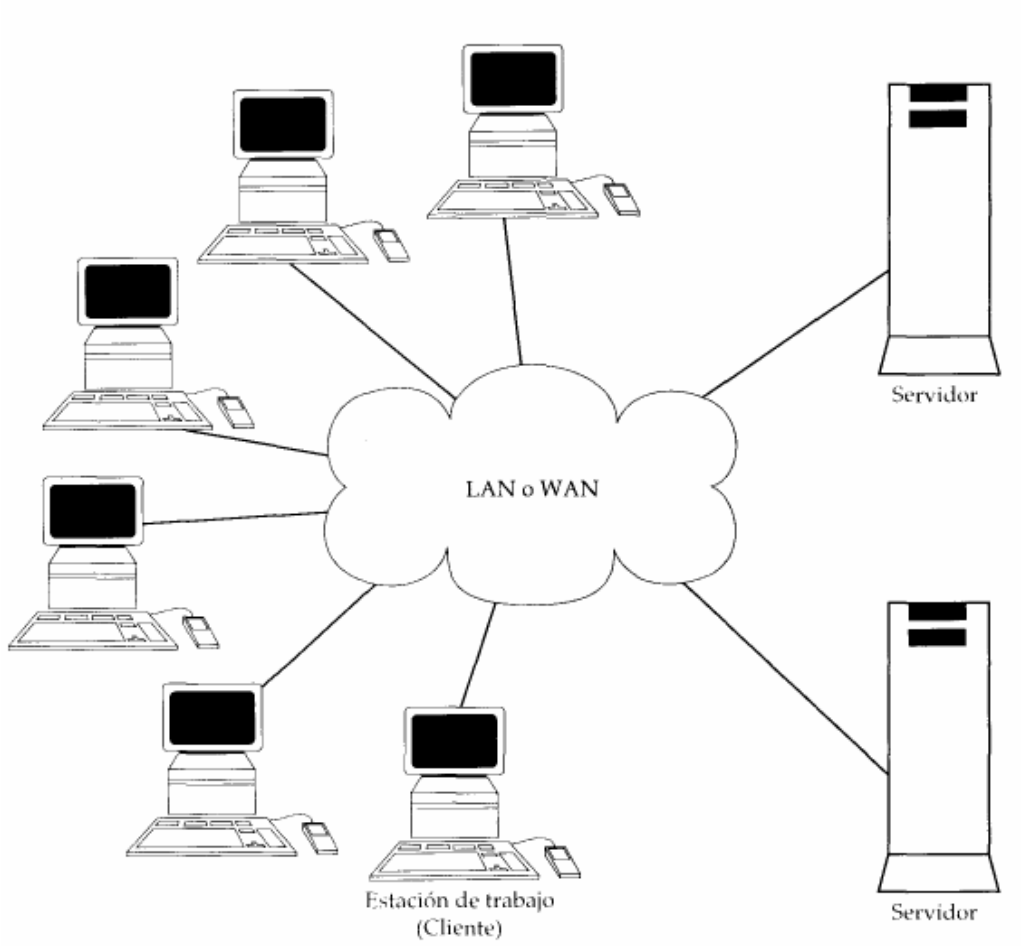


FIGURA 12.13 Entorno genérico cliente/servidor

nes y recursos se hallan distribuidos en respuesta a los requisitos del negocio y quedan enlazados por una sola LAN o WAN o por una serie de subredes.

¿En qué se diferencia una configuración cliente/servidor de cualquier otra solución al proceso distribuido? Existe una serie de características que llaman la atención y que, juntas, hacen distinto al proceso cliente/servidor del distribuido normal y corriente:

- Hay una gran confianza en depositar aplicaciones que sean fáciles de usar para los usuarios en sus propios sistemas. Esto da a los usuarios un alto grado de control sobre la medida del tiempo y el estilo de utilización del computador y ofrece a los directores de departamentos la posibilidad de responder a sus necesidades locales.
- Al mismo tiempo que las aplicaciones se dispersan, se produce un énfasis en la centralización de las bases de datos corporativas y de muchas funciones de utilidad y de gestión de la red. Esto habilita una gestión corporativa para mantener un control global de la inversión total en sistemas de información e informática y, además, permite una gestión corpo-

rativa que ofrezca interoperatividad, de manera que los sistemas queden vinculados. Al mismo tiempo, alivia a los departamentos individuales y divisiones de gran parte de la carga de mantener servicios de computación sofisticados, permitiéndoles elegir cualquier tipo de máquina e interfaz que necesiten para acceder a los datos y la información.

- Existe un compromiso, tanto por parte de las organizaciones de usuarios como de los fabricantes, hacia los sistemas abiertos y modulares. Esto significa que los usuarios disponen de ofertas mejores en la elección de productos y en la combinación de equipos de varios fabricantes.
- El trabajo en red es fundamental para la operación. De este modo, la gestión y seguridad de la red tienen una prioridad alta en la organización y operación de los sistemas de información.

Aplicaciones Cliente/Servidor

La característica central de la arquitectura cliente/servidor es la ubicación de las tareas del nivel de aplicación entre clientes y servidores. La figura 12.14 ilustra el caso general. Tanto en el cliente como el servidor, por supuesto, el software básico es un sistema operativo ejecutando en la plataforma de hardware. Las plataformas y los sistemas operativos del cliente y el servidor pueden ser diferentes. De hecho, puede existir un número de clases distintas de plataformas y sistemas operativos clientes y otro de clases distintas de plataformas y sistemas operativos servidores en un mismo entorno. En tanto que un cliente particular y un servidor compartan los mismos protocolos de comunicación y soporten las mismas aplicaciones, estas diferencias de niveles inferiores no son relevantes.

El software de comunicaciones es el que permite interoperar a cliente y servidor. Algunos ejemplos de dicho software incluyen TCP/IP, OSI y varias arquitecturas de propietario, como SNA. Por supuesto, el objeto de todo este software de soporte (comunicaciones y sistema operativo) es proporcionar una base para las aplicaciones distribuidas. En el mejor de los casos, las funciones reales de lis aplicación pueden repartirse entre cliente y servidor de forma que se optimicen los recursos de la red y de la plataforma, así corno la posibilidad de los usuarios para realizar varias tareas y cooperar uno con otro en el uso de los recursos compartidos. En algunos casos, estos requisitos dictan que el grueso del software de la aplicación se ejecute en el servidor, mientras que, en otros casos, la mayor parte de la lógica de la aplicación se ubica en el cliente.

Para finalizar, un factor esencial para el éxito de un entorno cliente/servidor es la manera en que el usuario interactúa con el sistema en su globalidad. De esta forma, el diseño de la interfaz de usuario es vital para la máquina cliente. En la mayoría de los sistemas cliente/servidor, se hace un gran hincapié en ofrecer un interfaz de usuario gráfico (GUI) que sea fácil de utilizar, fácil de aprender, pero potente y flexible. Así pues, se puede pensar en un módulo de servicios de presentación' en el puesto de trabajo cliente, responsable de ofrecer una interfaz fácil de usar a las aplicaciones distribuidas disponibles en el entorno.

Aplicaciones de Bases de Datos

Como un ejemplo que ilustra el concepto de división de la lógica de una aplicación entre cliente y servidor, considérese la familia más común de aplicaciones cliente/servidor: aquellas que utilizan bases de datos relacionales. En este entorno, el servidor es, básicamente, un

PI módulo de servicios de presentación no debe confundirse con el nivel de presentación del modelo OSI. El nivel de presentación se ocupa del formateo de los datos, de forma que las dos máquinas que se comunican puedan interpretarlos correctamente t n módulo de servicios de presentación se ocupará de la forma en que el usuario interactúa con una aplicación y con la disposición y funcionalidad de lo que se presenta al usuario en la pantalla.

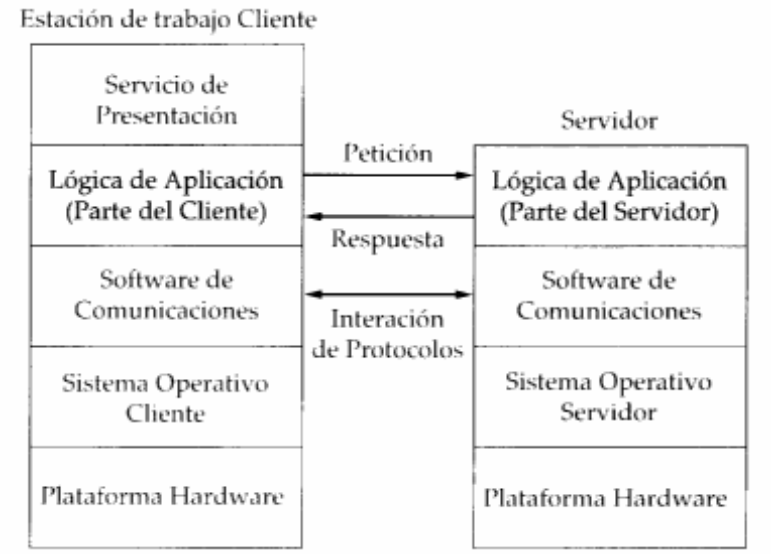


FIGURA 12.14 Arquitectura genérica cliente/servidor

servidor de base de datos. La interacción entre el cliente y el servidor se hace en forma de transacciones donde el cliente realiza una petición a la base de datos y recibe una respuesta de aquella.

La figura 12.15 ilustra, en términos generales, la arquitectura de este tipo de sistemas. El servidor es responsable de mantener la base de datos, para cuyo objeto se necesitan complejos sistemas gestores de bases de datos. En las máquinas cliente se pueden guardar una variedad de aplicaciones diferentes que hagan uso de la base de datos. El "pegamento" que enlaza al cliente con el servidor es el software que permite al cliente realizar peticiones de acceso a la base de datos del servidor. Un ejemplo popular de dicha lógica es el lenguaje estructurado de consulta SQL.

En la figura 12.15 se propone que toda la lógica de la aplicación -el software de tratamiento numérico u otras clases de análisis de datos- resida en el cliente, mientras que el servidor sólo se debe ocupar de la gestión de la base de datos. El hecho de que esta configuración sea adecuada dependerá del estilo y el propósito de la aplicación. Por ejemplo, supóngase que la finalidad principal es ofrecer un acceso directo para buscar registros. La figura 12.16a propone un posible funcionamiento. Supóngase que el servidor mantiene una base de datos de 1 millón de registros (denominados filas en el campo de las bases de datos relacionales) y que el usuario quiere realizar una búsqueda que culmine, todo lo más, en unos pocos registros o ninguno. El usuario podría buscar estos registros empleando un conjunto de criterios de búsqueda (por ejemplo, registros anteriores a 1992; registros referentes a individuos de Ohio; registros referentes a un suceso o característica específica, etc.). Una pregunta inicial del cliente podría provocar una respuesta del servidor de que existen 100.000 registros que satisfacen los criterios de búsqueda. El usuario añade entonces calificadores adicionales y emite una nueva pregunta. Esta vez, se devuelve una respuesta que indica que hay 1.000 registros posibles. Finalmente, el cliente emite una tercera petición con más calificadores añadidos. Los criterios de búsqueda resultantes deparan una única equiparación y se responde al cliente con el registro.

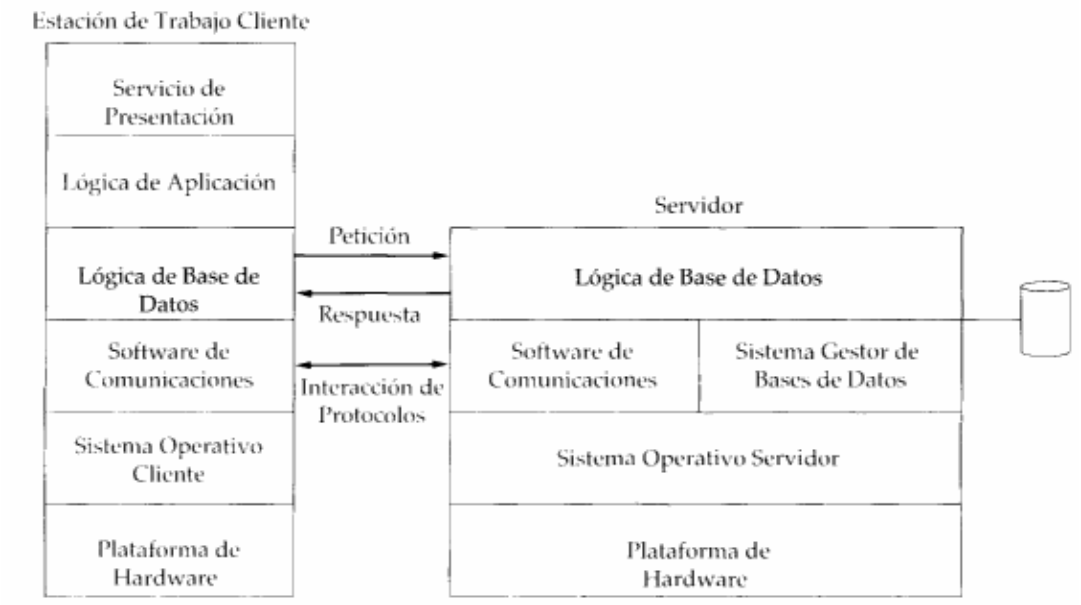
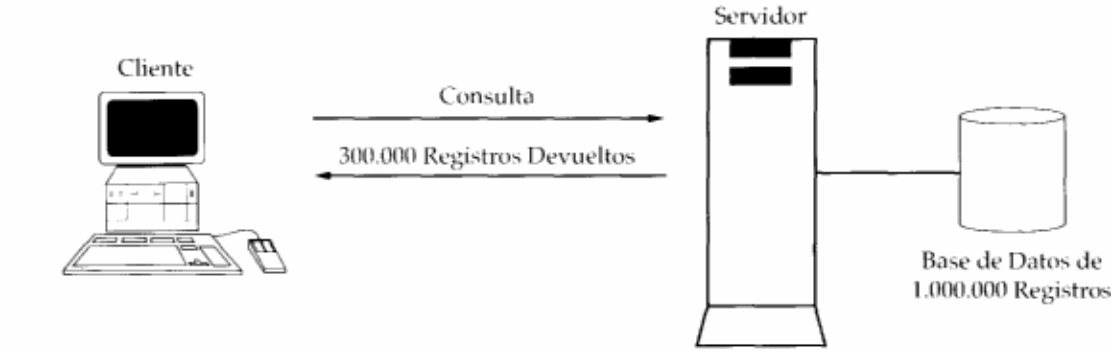


FIGURA 12.15 Arquitectura cliente/servidor para aplicaciones de base de datos

La aplicación anterior se adapta bien a la arquitectura cliente/servidor por dos razones:
1 Existe una labor masiva de ordenación y búsqueda en la base de datos. Esto requiere un disco grande o una serie de discos, una CPU y una arquitectura de E/S de alta velocidad. En un PC o puesto de trabajo monousuario, no hacen falta tales capacidades y potencian además son demasiado caras.
2 Mover el archivo completo de 1 millón de registros al cliente para realizar la búsqueda introduciría una carea de tráfico demasiado grande en la red. Por tanto, no es suficiente que el servidor sea sólo capaz de recuperar los registros en nombre del cliente; el servidor tiene que disponer de la lógica de base de datos que permita realizar búsquedas de parte del cliente.
Ahora se va a considerar la escena de la figura 12.16b, donde se tiene la misma base de datos de 1 millón de registros. En este caso, de una sola petición se obtiene la transmisión de 300.000 registros por la red. Esto podría ocurrir si, por ejemplo, el usuario quiere calcularla suma total o el valor medio de algún campo de varios registros o incluso de la base de datos entera.
Evidentemente, esta última escena es inaceptable. Una solución al problema, que conserva la arquitectura cliente-servidor con todos sus beneficios, es mover parte de la lógica de la aplicación al servidor. Es decir, el servidor puede equiparse con la lógica de la aplicación necesaria para realizar análisis de datos, así como recuperación y búsqueda de datos.

Clases de Aplicaciones Cliente/Servidor
Dentro del entorno general cliente/servidor, se dispone de una gama de posibles implementaciones que dividen el trabajo entre el cliente y servidor de manera diferente. La figura 12.17 ilustra, en líneas generales, alunas de las opciones principales para las aplicaciones

(a) Cliente/servidor bien empleado



(b) Cliente/servidor mal empleado

FIGURA 12.16 Utilización de bases de datos cliente/servidor [MORS93]

de bases de datos. Para otras clases de aplicaciones, son posibles otras divisiones y las opciones pueden caracterizarse de forma diferente. En cualquier caso, resulta útil examinar la figura para hacerse una idea del tipo de decisiones posibles.

La figura representa cuatro clases:

- *Proceso basado en el host:* El proceso basado en una máquina central *(host)* no es realmente un proceso cliente/servidor, de acuerdo con el uso que se hace del término. El proceso basado en host se refiere más bien al entorno tradicional de grandes sistemas en el que todo o casi todo el tratamiento se realiza en un computador central. La interfaz de usuario consiste a menudo en un terminal tonto. Incluso si el usuario emplea un microcomputador, el puesto del usuario se limita en general al papel de un emulador de terminales.
- *Proceso basado en el servidor:* El tipo más básico de configuración cliente/servidor es aquél en que el servidor es básicamente responsable de ofrecer una interfaz de usuario Gráfica y casi todo el tratamiento se hace en el servidor. Esta configuración es típica de los

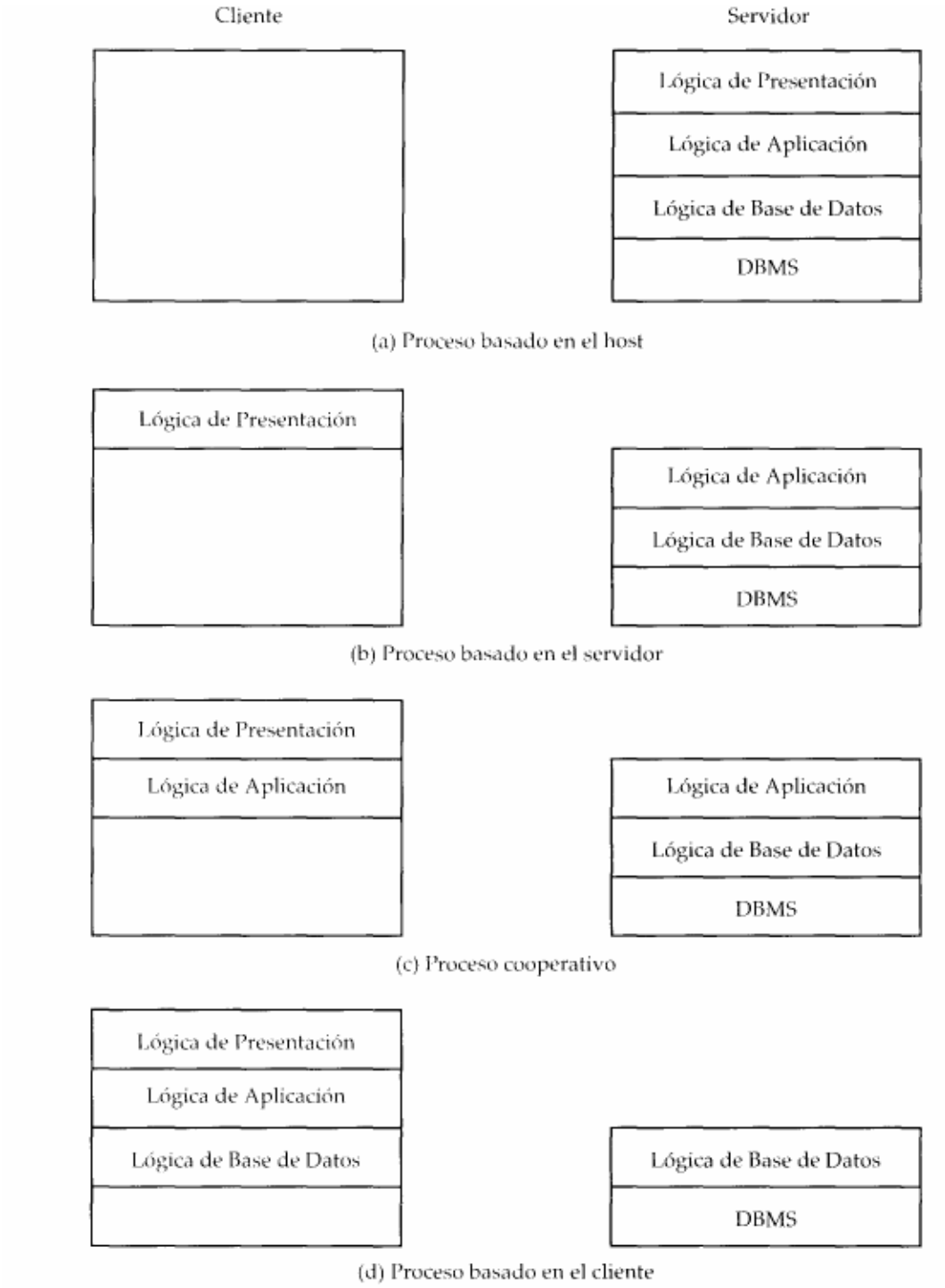


FIGURA 12.17 Clases de aplicaciones cliente/servidor

primeros intentos en cliente/servidor, especialmente de los sistemas departamentales. La razón fundamental que subyace en dichas configuraciones es que los puestos de trabajo de los usuarios se adaptan mejor a una interfaz fácil de usar y que las bases de datos y las aplicaciones pueden mantenerse fácilmente en sistemas centrales. Aunque el usuario gana la ventaja de una interfaz mejor, este tipo de configuración no se presta en generala ganancias significativas de productividad ni a cambios fundamentales en las funciones de negocio reales que el sistema ofrece.

- Proceso basado en el cliente: En el otro extremo, casi todo el proceso de la aplicación puede hacerse en el cliente, con la excepción de las rutinas de validación de datos y otras funciones lógicas de la base de datos que se realizan mejor en el servidor. En general, varias de las más sofisticadas funciones de la base de datos residen en el cliente. Esta arquitectura es posiblemente el método cliente/servidor más común de empleo actual. Permite al usuario utilizar aplicaciones a la medida de sus necesidades locales.
- Proceso cooperativo: En una configuración de proceso cooperativo, el proceso de la aplicación se lleva a cabo de forma optimizada, aprovechando la potencia de las máquinas cliente y servidora y la distribución de los datos. Esta configuración es más compleja de instalar y mantener pero, a largo plazo, este tipo de configuración puede ofrecer una mayor ganancia de productividad del usuario y una mayor eficacia de la red que otros métodos cliente/servidor.

Por supuesto, la distribución exacta del tratamiento de datos y aplicaciones dependerá de la naturaleza de la información de la base de datos, de los tipos de aplicaciones soportados, la disponibilidad de equipos compatibles y de las pautas de utilización en cada organización.

Consistencia de la caché de Archivos

Cuando se utiliza un servidor de archivos, el rendimiento de la E/S referente a los accesos locales a archivos puede degradarse sensiblemente por causa del retardo introducido por la red. Para reducir esta cara, los sistemas individuales pueden usar caches de archivos para almacenar los registros a los que se ha accedido hace poco. Debido al principio de cercanía, el empleo de una caché local de archivos debe reducir el número de accesos a servidores remotos a realizar. En la figura 12.18 se ilustra un mecanismo distribuido típico de caché de archivos en una serie de puestos de trabajo en red. Cuando un proceso realiza un acceso a archivo, la petición es cursada, en primer lugar, a la caché del puesto de trabajo del proceso ("tráfico de archivo"). Si la petición no se satisface, se pasa al disco local ("tráfico de disco") o al servidor donde se *almacene el* archivo ("tráfico de servidor"). Una vez en el servidor, se examina primero su caché y, si *se* produce una falla, se accederá al disco del servidor. Se suele utilizar un procedimiento de doble caché para reducir el tráfico de comunicaciones (caché cliente) y la E/S a disco (caché servidora).

Cuando las caches contienen siempre copias exactas de los datos remotos, se dice que las caches son **consistentes.** Puede ser posible que las caches lleguen a ser inconsistentes cuando se cambian los datos remotos y no se desechen las copias obsoletas correspondientes de las caches locales. Esto puede ocurrir si un cliente modifica un archivo almacenado en la caché de otro cliente. Esta dificultad se produce en realidad a dos niveles. Si un cliente adopta la política de notificar inmediatamente al servidor cualquier cambio en un archivo, cualquier otro cliente que tenga copia en caché de la parte relevante del archivo dispondrá

de datos obsoletos. El problema es aún peor si el cliente se retrasa en notificar los cambios al servidor. En tal caso, el mismo servidor tendrá una versión obsoleta del archivo y las nuevas peticiones de lectura al servidor pueden llevar datos también obsoletos. El problema de mantener actualizadas las copias de las caches locales se conoce como problema de **consistencia de caches.** El método más simple para la consistencia ele caches consiste en emplear técnicas ele bloqueo de archivos para prevenir el acceso simultáneo a un archivo por parte de más de un cliente. Esto garantiza la consistencia a costa de rendimiento y flexibilidad. Un método más potente es el ofrecido en Sprite [NELS88, OUST88]. Cualquier proceso remoto puede abrir un archivo para lectura y crear su propia cache cliente. Pero cuando la solicitud al servidor es de abrir un archivo para escritura y otros procesos tienen el mismo archivo abierto para lectura, el servidor tomará dos acciones. En primer lugar, notifica al proceso escritor que, aunque puede mantener una cache, debe reescribir todos los bloques alterados imnediatamente después ele actualizarlos. Puede haber como mucho un cliente de este tipo. En segundo lugar; el servidor notifica a todos los procesos lectores que tengan abierto el archivo que sus copias en cache ya no son válidas.

Middleware

El desarrollo y la distribución de productos cliente/servidor ha superado con mucho los esfuerzos de estandarizar todos los aspectos del proceso distribuido, desde el nivel físico hasta el nivel de aplicación. Esta carencia de estándares hace difícil implementar una configuración cliente/servidor empresarial integrada, válida para múltiples fabricantes. Congo gran parte del beneficio de la filosofía cliente/servidor viene dado por su modularidad y por la capacidad de combinar plataformas y aplicaciones para ofrecer soluciones comerciales, debe resolverse este problema ele interoperación. Para alcanzar los beneficios reales de la filosofía cliente/servidor, los productores deben disponer de un conjunto de herramientas que proporcionen una manera uniforme de acceder a los recursos del sistema en todas las plataformas. Esto servirá para que los programadores construyan aplicaciones que no sólo parezcan las mismas en PC y puestos de trabajo diferentes, sino que también utilicen el mismo método de acceso a los datos, sin importar la ubicación de los mismos.

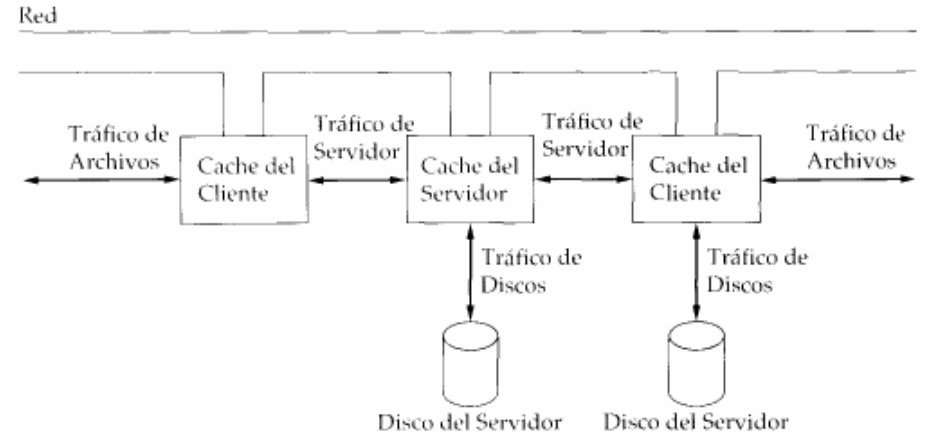


FIGURA 12.18 Caché de archivos distribuida empleada en Sprite

La forma más común de cumplir con este requisito es utilizar interfaces estándares de programación y protocolos que se sitúen entre la aplicación y el software de comunicaciones y sistema operativo. Dichos interfaces y protocolos estándares han venido a llamarse middleware2 interfaces estándares de programación, es fácil implementar una misma aplicación en una variedad de tipos de servidores y de puestos de trabajo. Esto tiene un beneficio obvio para los clientes, pero los fabricantes se verán motivados a ofrecer dichas interfaces. La razón es que los clientes compran aplicaciones, no servidores; los clientes sólo elegirán entre aquellos servidores donde se ejecuten las aplicaciones que ellos deseen. Se necesitarán protocolos estándares para enlazar las distintas interfaces de servidor con los clientes que necesiten acceder a ellos.

Existe una gran variedad de paquetes de middleware, desde los muy simples a los muy complejos. 'lodos ellos tienen en común la capacidad de ocultar las complejidades y diferencias de los diferentes protocolos de red y sistemas operativos. En general, los fabricantes de clientes y servidores ofrecen opcionalmente un conjunto de paquetes populares de middIeware. De este modo, un usuario puede decidirse por una estrategia particular de middleware y reunir equipos de varios fabricantes que soporten dicha estrategia.

Arquitectura de Middleware

En la figura 12.19 se propone el papel del middleware en una arquitectura cliente/servidor. El papel exacto del componente middleware dependerá del estilo del proceso distribuido que se utilice. En una nueva referencia a la figura 12.17, se recuerda que hay una serie de métodos diferentes de cliente/servidor, dependiendo de la forma en que se dividen las funciones de la aplicación. En cualquier caso, la figura 12.19 da una idea general bastante buena de la arquitectura considerada.

Nótese que en el middleware existen componentes cliente y servidor. La finalidad básica del middleware es hacer que una aplicación o usuario del cliente acceda a una serie de servicios del servidor sin preocuparse de las diferencias entre servidores. Si se considera un área específica de aplicación, se supone que el lenguaje estructurado de consulta (SQL) proporciona una forma estándar de acceder a una base de datos relacional tanto a usuarios o aplicaciones locales como remotos. Sin embargo, muchos fabricantes de bases de datos relacionales, aunque también soportan SQL, le han añadido sus propias ampliaciones. Así se consigue que los fabricantes diferencien sus productos, pero también se crean posibles incompatibilidades.

Como ejemplo, considérese un sistema distribuido usado para gestionar, entre otras cosas, un departamento de personal. Los datos básicos de un empleado, como el nombre, la dirección, etc., pueden guardarse en una base de datos Gupta, mientras que la información de salarios puede estar en una base de datos Oracle. Cuando un usuario del departamento de personal necesita acceder a determinados registros, no deberá preocuparse del fabricante de la base de datos que contiene los registros que necesita. El middleware proporciona una capa de software que permite un acceso uniforme a estos sistemas diferentes.

Es interesante considerar el papel del middleware desde un punto de vista lógico, más que desde la implementación. Este punto de vista se ilustra en la figura 12.20. El middleware

El término middleware c, muy u,;ido en la literatura cubre cIiente/servidor. Sin embargo, cuando se fue a publicar esta obra, TechGnosis Inc., un fabricante de software, anunció que la Oficina de Patentes y Marcas Registradas de los EE.UU. le había otorgado la marca registrada ,obre dicho término, en reconocimiento del "temprano uso" de la palabra por parte de la compañía. Queda por ver si este término seguirá usándose de forma genérica.

permite cumplir la promesa del proceso distribuido. El sistema distribuido entero puede verse como un conjunto de aplicaciones y recursos disponibles para los usuarios. Estos no necesitan preocuparse de la ubicación de los datos ni de la ubicación de las aplicaciones. Todas las aplicaciones operan sobre una interfaz uniforme de programación de aplicaciones (API). El middleware, que atraviesa todas las plataformas clientes y servidoras, es el responsable de encaminar las peticiones de los clientes al servidor apropiado.

Un ejemplo de cómo se aplica el Middleware para integrar productos dispares es la instalación representada en la figura 12.21. En este caso, el middleware se usa para soslayarlas incompatibilidades de redes y sistemas operativos. Una red esqueleto conecta redes DECnet, Novel] y **TCP/IP.** El middleware que se ejecuta en cada componente de la red, asegura que todos los usuarios de la red disponen de un acceso transparente a las aplicaciones y los recursos de cualquiera de las tres redes.

Aunque hay una amplia variedad de productos de middleware estos se basan normalmente en uno de estos dos mecanismos básicos: el paso de mensajes o las llamadas a procedimientos remotos. En las siguientes secciones se examinan estos dos métodos.

12.4 PROCESO DISTRIBUIDO MEDIANTE ENVÍO DE MENSAJES

En los sistemas de proceso distribuido reales se suele dar el caso de que los computadores no comparten una memoria principal; cada una se convierte en un sistema de computadores aislado. Por tanto, no pueden emplearse técnicas de comunicación entre procesadores que se basen en memoria compartida, como son los semáforos y el uso de un área de me-

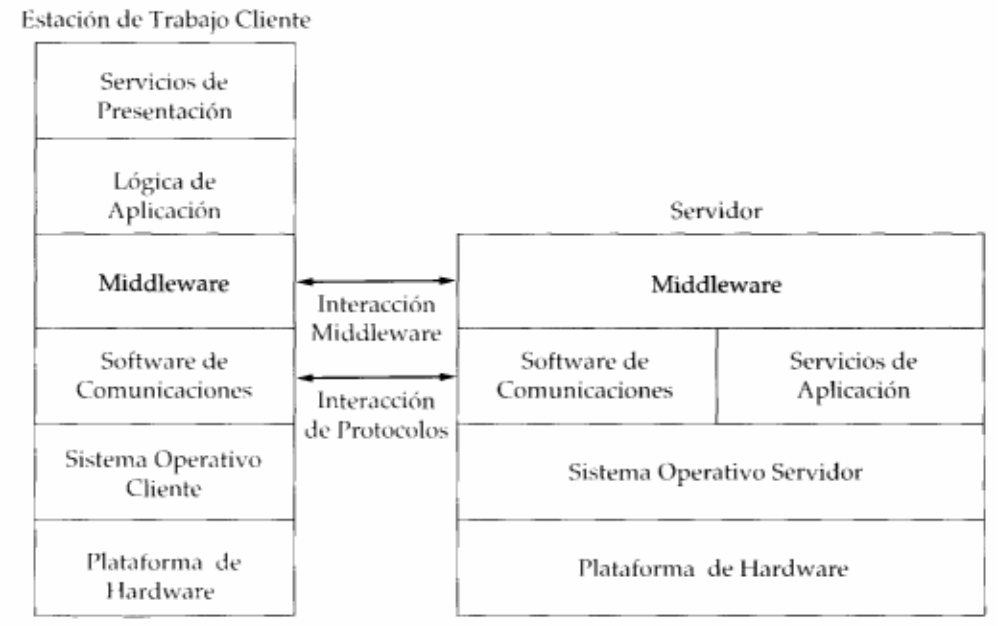


FIGURA 12.19 El papel del middleware en la arquitectura cliente/servidor

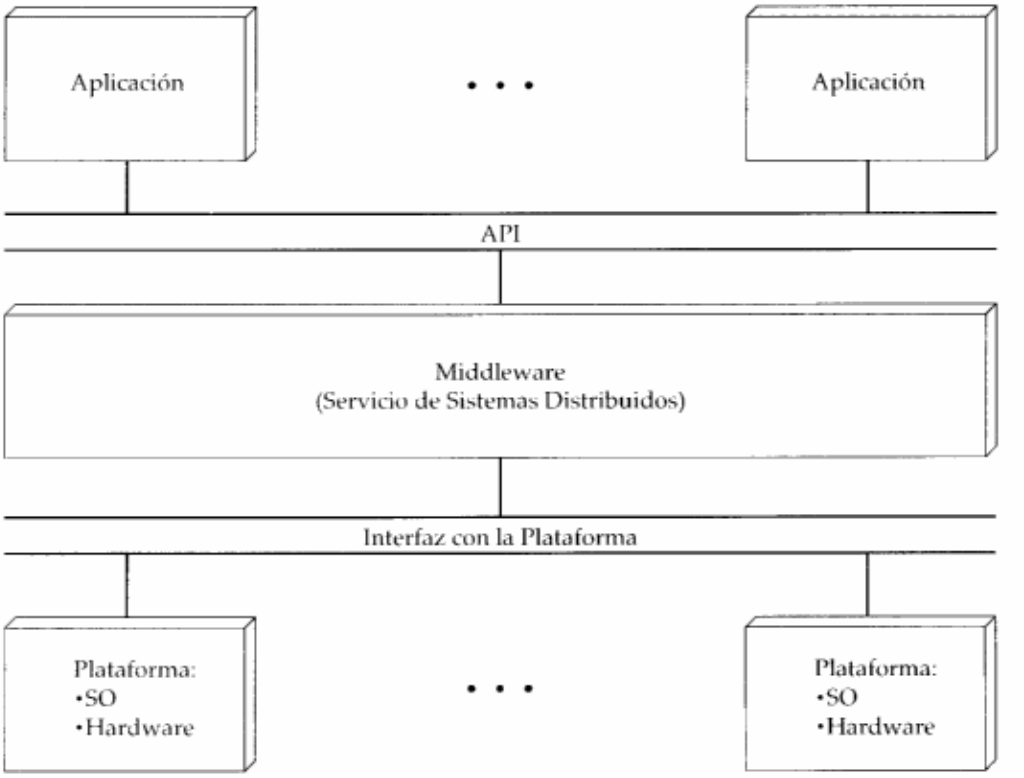


FIGURA 12.20 Visión lógica del middleware [BERN93]

moria común. En su lugar, se usan técnicas basadas en el paso de mensajes. En esta sección y en la siguiente se estudian los dos procedimientos más usuales. El primero es la aplicación simple de los mensajes, tal y como se hace en un mismo sistema. El segundo es una técnica distinta que se basa en el paso de mensajes como función básica: la llamada a procedimientos remotos.

En la figura 12.22 se muestra el modelo más usual de paso distribuido de mensajes, conocido como *modelo cliente-servidor.* Un proceso cliente solicita un servicio (por ejemplo, leer un archivo o imprimir) y envía un mensaje que contiene una petición de servicio a un proceso servidor. El proceso servidor cumple con la petición y envía una respuesta. En su forma más simple, sólo se necesitan dos funciones: Enviar y Recibir. La función Enviar debe especificar un destino e incluir el contenido del mensaje. La función Recibir dice de quién se desea recibir mensajes (incluyendo a "todos") y proporciona un almacenamiento intermedio (buffer) donde se guardará el mensaje.

En la figura 12.23 se propone un método de implementación del paso de mensajes. Los procesos hacen uso de los servicios de un módulo de paso de mensajes, que forma parte del sistema operativo. Las solicitudes de servicio pueden expresarse en forma de primitivas y parámetros. La primitiva especifica la función a realizar y los parámetros se usan para pasar datos e información de control. El formato real de las primitivas depende del sistema operativo. Pueden ser llamadas a procedimientos o en forma de mensajes a un proceso que sea parte del sistema operativo.

La primitiva Enviar la utiliza el proceso que quiere enviar el mensaje. Sus parámetros son el identificador del proceso de destino y el contenido del mensaje. El módulo de paso de mensajes construirá una unidad de datos que incluya estos dos elementos (compárese con la figura 12.5). Esta unidad de datos es enviada a la máquina que alberga al proceso de destino mediante algún tipo de servicio de comunicaciones, como la arquitectura de comunicaciones OSI. Cuando se recibe la unidad de datos en el sistema de destino, se encaminará, mediante el servicio de comunicaciones, hacia el módulo de paso de mensajes. Este módulo examina el campo identificador del proceso y almacenará el mensaje en el buffer de dicho proceso.

En esta escena, el proceso receptor debe anunciar su intención de recibir mensajes, designando un área de almacenamiento intermedio e informando al módulo de paso de mensajes por medio de una primitiva Recibir. Una medida alternativa consiste en no requerir dicho anuncio. En su lugar, cuando el módulo de paso de mensajes reciba un mensaje, avisará al proceso de destino con algún tipo de señal de recepción y después hará que el mensaje recibido esté disponible en un buffer compartido.

En el resto del apartado se tratarán varias cuestiones de diseño relacionadas con el paso distribuido de mensajes.

Fiables frente a No Fiables

Un servicio de paso de mensajes fiable es aquél que garantiza el envío si es posible. Dicho servicio debería hacer uso de un protocolo de transporte fiable o de alguna lógica similar y llevaría a cabo chequeos de errores, acuses de recibo, retransmisiones y reordenación de mensajes desordenados. Como el envío está garantizado, no es necesario hacer que el proceso emisor sepa que el mensaje fue enviado. Sin embargo, puede ser útil proporcionar un acuse de recibo al proceso emisor de manera que se entere de que tuvo lugar el envío. En cualquier caso, si el servicio falla al completar el envío (por ejemplo, por un fallo persistente de la red o una caída del sistema de destino), se notificará de este fallo al proceso emisor.

En el otro extremo, el servicio de paso de mensajes puede enviar simplemente el mensaje a la red de comunicaciones sin informar de su éxito ni de su fracaso. Esta alternativa reduce enormemente la complejidad y la sobrecarga de proceso y de comunicaciones en el servicio de paso de mensajes. Las aplicaciones que necesiten confirmación de que se ha enviado un mensaje pueden usar mensajes de Petición y Respuesta para cumplir con tal requisito.

Bloqueantes frente a No Bloqueantes

Con primitivas no bloqueantes, un proceso no es suspendido como resultado de hacer un Enviar o un Recibir. De esta forma, cuando un proceso emita una primitiva Enviar, el sistema operativo le devolverá el control tan pronto como el mensaje se haya puesto en cola para su transmisión o se haya hecho una copia. Si no se hace copia, cualquier cambio en el mensaje por parte del proceso emisor, antes de la transmisión o durante la misma, se hará bajo la responsabilidad del proceso.

Cuando el mensaje se haya transmitido o se haya copiado a un lugar seguro para su posterior transmisión, e1 proceso emisor se verá interrumpido e informado de que el buffer del mensaje puede reciclarse. De forma similar, un Recibir no bloqueante lo emite un proceso para después seguir ejecutando. Cuando llegue un

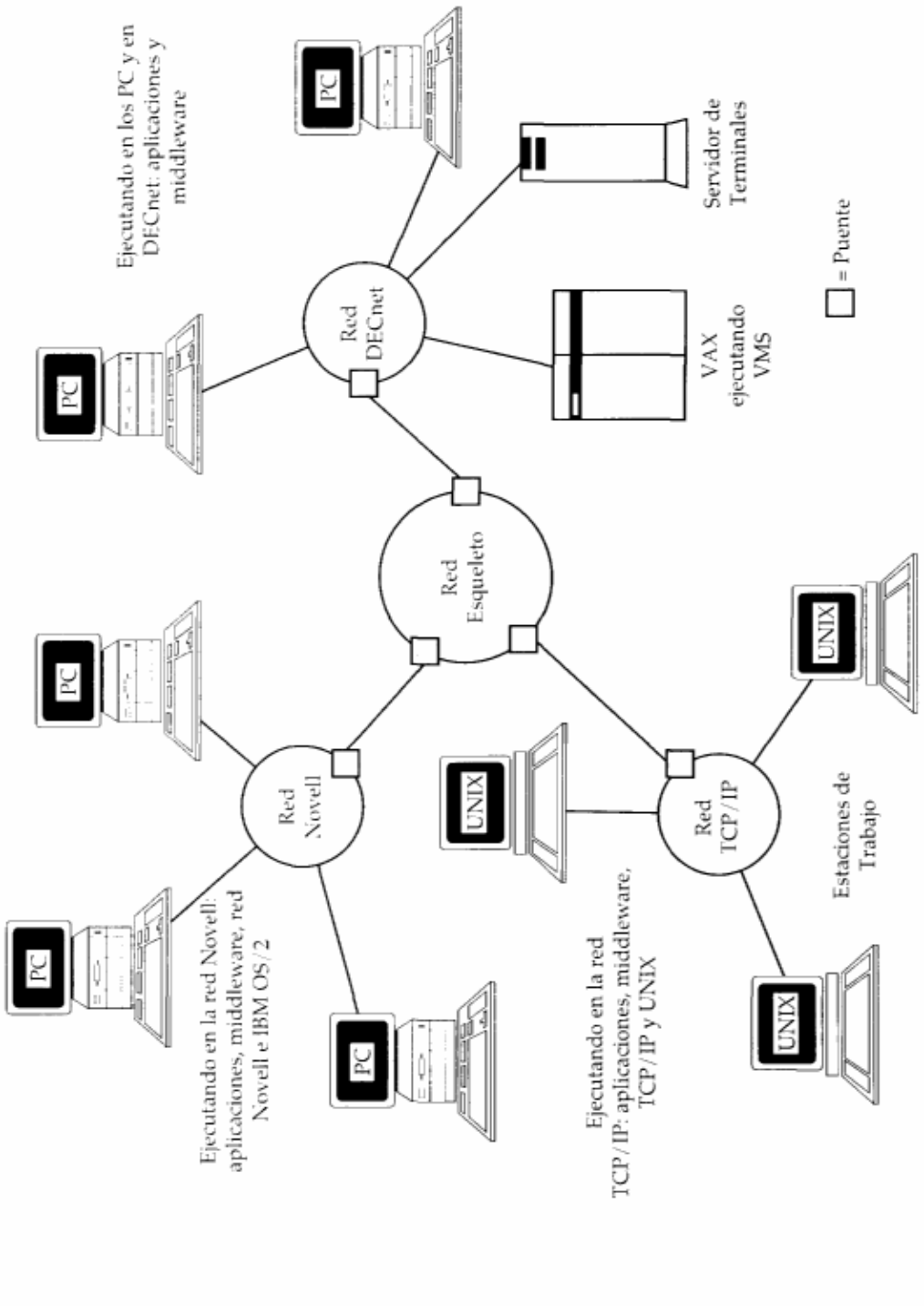


FIGURA 12.21 Ejemplo de la funcionalidad del middleware [KORZ93]

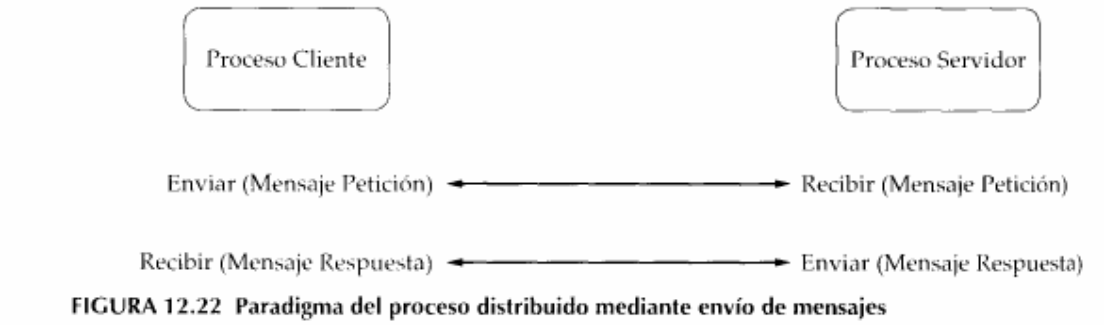


FIGURA 12.22 Paradigma del proceso distribuido mediante envío de mensajes

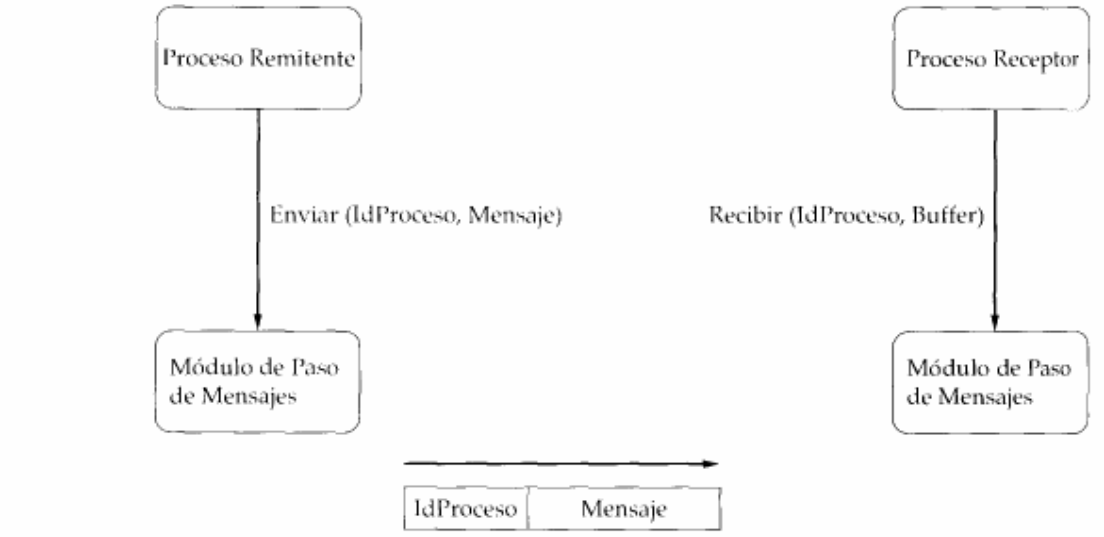


FIGURA 12.23 Primitivas básicas de paso de mensajes

mensaje, el proceso es informado mediante interrupción o bien puede muestrear su estado periódicamente.

Las primitivas no bloqueantes ofrecen un empleo eficiente y flexible del servicio de paso de mensajes para los procesos. La desventaja de este enfoque es que los programas que emplean estas primitivas son difíciles de probar y depurar. Corno se ha visto al discutir sobre concurrencia, las secuencias irreproducibles dependientes del tiempo pueden originar problemas sutiles y complicados.

La otra alternativa es emplear primitivas bloqueantes. Un Enviar bloqueante no devuelve el control al proceso emisor hasta que el mensaje se haya transmitido (servicio no fiable) o hasta que el mensaje se haya enviado y obtenido un acuse de recibo (servicio fiable). Un Recibir bloqueante no devuelve el control hasta que el mensaje se haya ubicado en el buffer asignado.

12.5

LLAMADAS A PROCEDIMIENTOS REMOTOS

Una variante del modelo básico de paso de mensajes es la llamada a procedimientos remotos, que es un método común muy aceptado actualmente para encapsular la comunicación en un sistema distribuido. Lo fundamental de la técnica es permitir que programas de máquinas diferentes interactúen mediante la simple semántica de los procedimientos de llamada/retorno, como si los dos programas estuvieran en la misma máquina. Es decir, se va a usar la llamada a procedimiento para acceder a servicios remotos. La popularidad de este enfoque se debe a las siguientes ventajas:

1. La llamada a procedimiento es una abstracción muy usada, aceptada y bien comprendida.
2. El empleo de llamadas a procedimientos remotos permite que las interfaces remotas se especifiquen como un conjunto de operaciones con nombre y tipo determinado. De este modo, la interfaz puede documentarse de forma clara y los programas distribuidos pueden chequearse para detectar errores de tipo.
3. Como la interfaz es estándar y está definida de forma precisa, el código de comunicaciones de una aplicación puede generarse automáticamente.
4. Como la interfaz es estándar y está definida de forma precisa, los productores de software pueden escribir módulos clientes y servidores que pueden trasladarse entre computadores y sistema operativos con pocas modificaciones.

El mecanismo de las llamadas a procedimientos remotos puede considerarse como un refinamiento del paso de mensajes fiable y bloqueante. En la figura 12.24 se ilustra la arquitectura genérica, mientras que la figura 12.25 muestra la lógica interna de programa. El programa llamador realiza una llamada normal a un procedimiento con los parámetros situados en su máquina. Por ejemplo,

CALL P (X, Y)

donde

P = nombre de procedimiento
X = argumentos pasados
Y = valores devueltos

El hecho de que se intente invocar a un procedimiento remoto de otra máquina puede resultar o no trasparente al usuario. En el espacio de direcciones del llamador debe incluirse un procedimiento P sustituto *(stub),* o bien debe enlazarse dinámicamente en el momento de la llamada. Este procedimiento crea un mensaje para identificar al procedimiento llamado e incorpora los parámetros. Después de enviar este mensaje, mediante una primitiva Enviar, queda a la espera de una respuesta. Cuando se recibe la respuesta, el procedimiento sustituto retorna al programa llamador, proporcionando los valores devueltos.

En la máquina remota, se asocia otro procedimiento sustituto con el procedimiento invocado. Cuando llega un mensaje, se examina y se genera una llamada local CALL P (X, Y). Este procedimiento remoto es invocado de forma local y, de esta manera, los supuestos sobre dónde encontrar los parámetros, el estado de la pila y otros similares son idénticos al caso de una llamada a un procedimiento local.

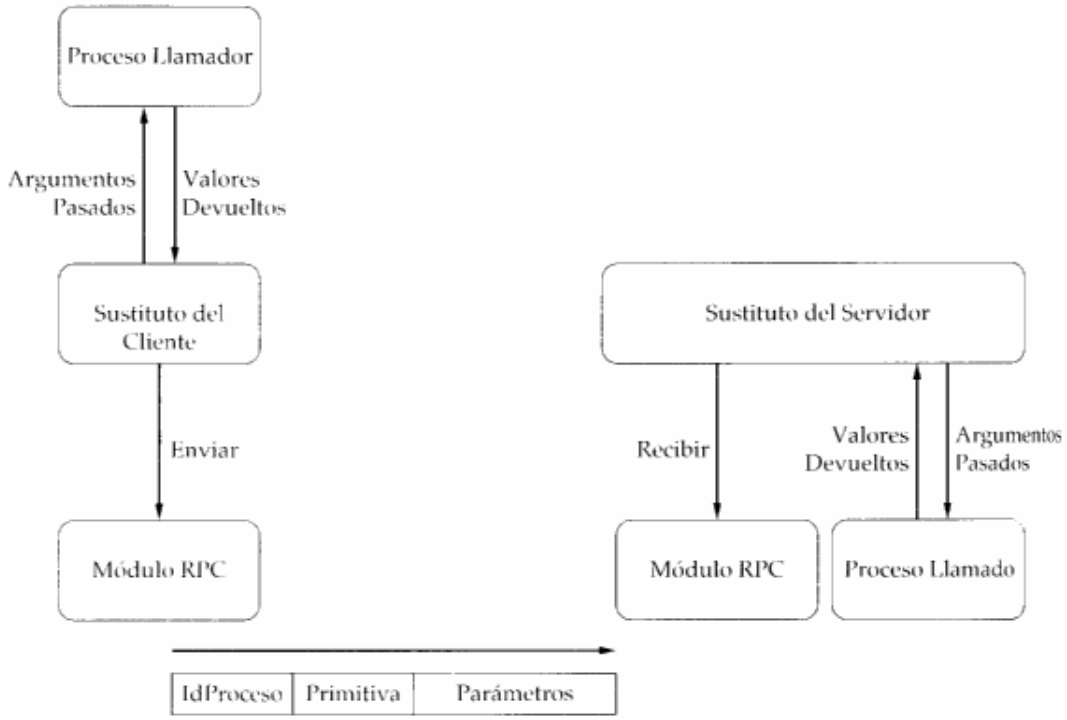


FIGURA 12.24 Mecanismo de llamada a procedimientos remotos

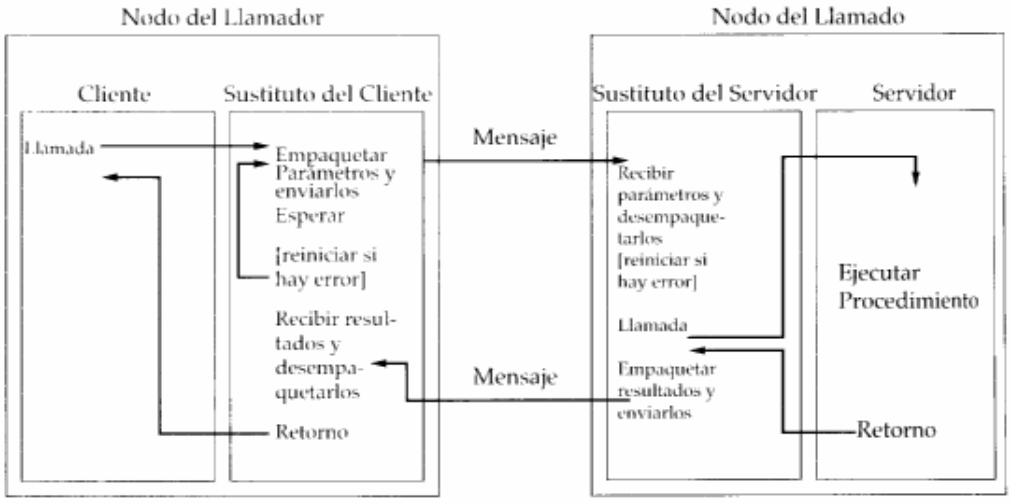


FIGURA 12.25 Lógica de llamada a procedimientos remotos [KRAK88]

En el resto del apartado se tratarán varias cuestiones de diseño relacionadas con las llamadas a procedimientos remotos.

Paso de Parámetros

La mayoría de los lenguajes de programación permiten pasar parámetros por valor (llamada por valor) o corno punteros a ubicaciones cine contienen el valor (llamada por referencia). El paso de parámetros por valor es sencillo para las llamadas a procedimientos remotos. Los parámetros se copian simplemente en el mensaje y se envían al sistema remoto. Las llamadas por referencia son más difíciles de implementar. Hace falta un único puntero para cada objeto, válido en todo el sistema. El coste de este servicio puede no merecer la pena.

Representación de Parámetros

Otra cuestión es cómo representar los parámetros y los resultados en los mensajes. Si el programa llamador y el invocado están construidos en los mismos lenguajes de programación, sobre el mismo tipo de máquinas y con el mismo sistema operativo, los requisitos de representación no son un problema. Si existen diferencias en estos aspectos, probablemente habrá diferencias en la manera en que se representan los datos numéricos e incluso los textos. Si se emplea una arquitectura de comunicaciones en condiciones, esta cuestión la manejará el nivel de presentación. Sin embargo, el coste que supone tal arquitectura ha llevado hacia el diseño de servicios de llamadas a procedimientos remotos que evitan la mayor parte de la arquitectura de comunicaciones y ofrecen unos servicios básicos propios de comunicaciones. En tal caso, la responsabilidad de la conversión recae en el servicio de llamadas a procedimientos remotos (como ejemplo, véase **[GIBB87]**).

El mejor enfoque para este problema es ofrecer un formato estándar para los objetos comunes, como los enteros, números en coma flotante, caracteres y cadenas de caracteres. De esta forma, los parámetros propios de cualquier máquina pueden convertirse a la representación estándar.

Enlace Cliente-Servidor

El enlace especifica la forma en que se establecerá la relación entre un procedimiento remoto y el programa llamador. Un enlace se forma cuando dos aplicaciones han establecido una conexión lógica y se encuentran preparadas para intercambiar órdenes y datos.

Los **enlaces no persistentes** suponen que la conexión lógica se establece entre dos procesos en el momento de la llamada remota y que la conexión se pierde tan pronto como se devuelvan los valores. Como una conexión requiere el mantenimiento de información de estado en ambos extremos. se consumirán recursos. El estilo no persistente se utiliza para reservar dichos recursos. Por otro lacio, el coste de establecer las conexiones hace que los enlaces no persistentes no sean muy apropiados para procedimientos remotos que son invocados con frecuencia por un mismo llamador.

Con **enlaces persistentes,** una conexión establecida para tina llamada a un procedimiento remoto se mantiene después de terminar el procedimiento. La conexión puede utilizarse para llamadas futuras. Si transcurre un periodo de tiempo específico sin actividad en la conexión, se finaliza la misma. Para aplicaciones que realicen llamadas repetidas a procedimientos remotos, el enlace persistente mantiene la conexión lógica y permite que una secuencia de llamadas y retornos utilicen la misma conexión.

Síncrono frente a Asíncrono

El concepto de llamadas a procedimientos remotos síncronos y asíncronos es análogo al concepto de mensajes bloqueantes y no bloqueantes. La llamada tradicional a procedimiento remoto es síncrona, lo que requiere que el proceso llamador espere hasta que el proceso llamado devuelva un valor. Así, la **RPC síncrona** se comporta de manera muy parecida a una llamada a subrutina.

La RPC síncrona es fácil de comprender y de programar puesto que su comportamiento es predecible. Sin embargo, no es capaz de explotar por completo el paralelismo inherente a las aplicaciones distribuidas. Esto limita el tipo de interacción que las aplicaciones distribuidas pueden realizar, obteniéndose un rendimiento menor.

Con objeto de ofrecer una mayor flexibilidad, se han implementado varios servicios de **RPC asíncrona** que consiguen un grado mayor de paralelismo, a la vez que conservan la simplicidad y familiaridad de la RPC [ANAN92]. Las RPC asíncronas no bloquean al llamador; las respuestas pueden recibirse cómo y cuándo se necesiten, permitiendo que la ejecución de los clientes continúe localmente y en paralelo con la invocación al servidor.

Una aplicación típica de las RPC asíncronas es hacer que un cliente invoque repetidamente a un servidor, generando una serie de peticiones de una vez, cada una con su propio conjunto de datos. La sincronización del cliente y el servidor puede conseguirse de dos maneras:

1. Una aplicación de nivel superior en el cliente y en el servidor puede iniciar el intercambio y comprobar al final que se han llevado a cabo todas las acciones solicitadas.
2. Un cliente puede emitir una cadena de RPC asíncronas, seguidas de una RPC síncrona final. El servidor responderá a la RPC síncrona sólo después de culminar todo el trabajo solicitado en las RPC asíncronas precedentes.

En algunos enfoques, las RPC asíncronas no exigen una respuesta del servidor y éste no puede enviar mensajes de respuesta. Otros esquemas requieren o permiten una respuesta, pero el llamador no quedará a la espera de la misma.

12.6 RESUMEN

Los computadores operan cada vez más como parte de una red de computadores y terminales y no de forma aislada. Para respaldar estos sistemas distribuidos, se ha desarrollado un abanico de posibilidades, entre las que se incluyen arquitecturas de comunicaciones, sistemas operativos de red y sistemas operativos distribuidos.

Una **arquitectura de comunicaciones** es un conjunto estructurado de hardware y software que soporta el intercambio de datos entre varios sistemas, así como aplicaciones distribuidas cono el correo electrónico y la transferencia de archivos. El modelo de interconexión de sistemas abiertos (OSI) es una arquitectura estándar de comunicaciones diseñada para hacer que cooperen computadores heterogéneos. El modelo OSI consta de siete niveles funcionales. Se han desarrollado estándares para implementar las funciones de cada nivel. La arquitectura de comunicaciones más usada e independiente del fabricante es la serie de protocolos TCP/IP.

Un sistema operativo de red no es en realidad un sistema operativo sino un conjunto distribuido de software de sistemas que respalda el empleo de servidores en una red. Las máquinas servidoras ofrecen servicios o aplicaciones a toda la red, como el almacenamiento de archivos y la gestión de impresoras. Cada computador dispone de su propio sistema operativo. El sistema operativo de red es, simplemente, un anexo al sistema operativo local que permite que las máquinas con aplicaciones interactúen con las servidoras. Normalmente, para respaldar estas aplicaciones de red se emplea una arquitectura común de comunicaciones.

Un **sistema operativo distribuido** es un sistema operativo común compartido por una red de computadores. Ante los usuarios aparece corno un sistema operativo normal centralizado, pero les ofrece un acceso transparente a los recursos de una serie de máquinas. Para realizar las funciones básicas de comunicación, un sistema operativo distribuido puede apoyarse en una arquitectura de comunicaciones; generalmente, se desgajan algunas funciones de comunicación para incorporarlas dentro del sistema operativo y así obtener una mayor eficiencia.

El proceso cliente/servidor es la clave para poner en marcha la capacidad de los sistemas de información y las redes hacia una mejora significativa de la productividad en las organizaciones. En el proceso cliente/servidor, las aplicaciones se distribuyen a los usuarios situándolas sobre puestos de trabajo y computadores personales monousuario. Al mismo tiempo, los recursos que pueden y deben compartirse se mantienen en sistemas servidores que están disponibles para todos los clientes. De esta forma, la arquitectura cliente/servidor es una mezcla de procesos centralizados y descentralizados.

Normalmente, el sistema cliente proporciona una interfaz gráfica de usuario (GUI) que permite que el usuario utilice varias aplicaciones con una formación mínima y relativa facilidad. Los servidores soportan utilidades compartidas, como sistemas gestores de bases de datos. Las aplicaciones reales se dividen entre cliente y servidor de forma que se optimen la facilidad de uso y el rendimiento.

El mecanismo clave necesario en cualquier sistema distribuido es la comunicación entre procesos. Dos técnicas son las más usuales. El uso de mensajes en sistemas sencillos se puede generalizar con servicios de paso de mensajes. En estos se aplican el mismo tipo de convenios y reglas de sincronización. Otro enfoque es el empleo de llamadas a procedimientos remotos. Esta es una técnica por la que dos programas de máquinas diferentes interactúan mediante la sintaxis y la semántica de los procedimientos de llamada/retorno. Tanto el programa llamador corno el invocado se comportan cono si la otra parte estuviera ejecutando en la misma máquina.

12.7 LECTURAS RECOMENDADAS

[STAL94a] expone las arquitecturas de comunicación haciendo énfasis en los aspectos tecnológicos y de diseño. [STAL93b] es una descripción detallada del modelo OSI y de los estándares de cada nivel del modelo.

[BERS92] y [DEWI93] ofrecen ambos una interesante discusión sobre los aspectos de diseño involucrados en la asignación de aplicaciones al cliente y al servidor y en los procedimientos de middleware. También se discute en ambos libros sobre los productos y los esfuerzos de estandarización. [INMO91] es un estudio menos técnico que se centra en las clases de aplicaciones que pueden verse

mejor respaldadas utilizando proceso cliente/servidor. IRENA93] se orienta hacia los aspectos de gestión de la instalación de sistemas cliente/servidor y de la selección de aplicaciones para dicho entorno.

[TANE85] es un estudio de los sistemas operativos distribuidos que abarca tanto la comunicación como la gestión distribuida de procesos. [CHAN90b] ofrece una visión general de los sistemas operativos con paso distribuido de mensajes. [TAY90] es un estudio del enfoque adoptado en varios sistemas operativos para implementar llamadas a procedimientos remotos, mientras que IMALA901 examina la aplicación de las llamadas a procedimientos remotos.

BERS92 BERSON, A. *Client/Server Architecture.* McGraw-Hill, Nueva York, 1992.

CHAN90b CHANDRAS, R. "Distributed Message Passing Operating Systems". *Operating Systems Review* enero de 1990.

DEWI93 DEWIRE, D. *Client/Server Computing.* McGraw-Hill, Nueva York, 1993.

INMO91 Inmon, W. *Developing Client/Server applications* in *an Architected Enviroment QED,* Boston, MA, 1991.

MALA90 MALAMUD, C. "Sharing the Wealth: RPCs Help Programs Go Places". *Data Communications 21* de junio (le 1990.

RENA93 RENAUD, P. *Introduction Client/Server Systems* Wiley, Nueva York, 1993.

STAL93b STALLINGS, W. *Networking - Standards A Guide to OSI, ISDN, LAN, and MAN Standards.* Addison-Wesley, Reading. MA, 1993.

STAL94a S TALLINGS W. *Data and Computer Communications, 4`* ed. Macmillan Nueva York, 1991.

TANE85 TANENBAUM, A. y RENESSE, R. "Distributed Operating Systems". *Computing Surveys* diciembre de 1985.

TAY90 TAY, B. y ANANDA, A.- A Survey of Remote Procedure Calls". *Operating Systems Review* julio de 1990.

12.8 PROBLEMAS

12.1Escribir un programa que implemente la técnica de ventana deslizante parí (a) REJ y (b) SREJ.

12.2 Considérese un enluce para la transmisión entre los puestos A y B con una probabilidad p de error en Una trama

a) Supóngase un protocolo SREJ. que A envía datos (tramas de información), B envía solo acuses de recibo (RR, SREJ) y que B acusa el recibo individual de cada trama Supóngase que los acuses de recibo no se pierden nunca. ¿Cuál es el número medio de retransmisiones necesarias por trama?

b) Supóngase ahora un protocolo REJ y que el enlace es tal que A va a retransmitir tres tramas adicionales antes de recibir un

RR o REJ para una trama en particular. Supóngase también que los acuses de recibo nunca se pierden. ¿Cuál es el número medio de retransmisiones necesarias por trama?

12.3Enumere las desventajas principales de enfocar los protocolos por niveles.

12..1Entre los principios empleados por ISO para definir los niveles OSI estaban:

- El número de niveles debe ser suficientemente pequeño como para evitar que el diseño y la implementación sean difíciles de abordar, pero suficientemente grande como para que los niveles separados manejen funciones diferentes en proceso o en tecnología.

Las fronteras entre los niveles deben elegirse para minimizar el número y el tamaño de las interacciones entre las mismas.

Basándose en estos principios, diseñar una arquitectura de ocho niveles y proponer un ejemplo de ella. Diseñar otra de seis niveles y proponer otro ejemplo.

12.5 Dos ejércitos azules están apostados en dos colinas opuestas preparándose para atacar a un solo ejército rojo situado en el valle. El ejército rojo puede derrotar a cualquiera de los dos azules por separado, pero no logrará derrotar a ambos si atacan simultáneamente. Los ejércitos azules se comunican mediante un sistema de comunicaciones no fiable (un soldado de infantería). El comandante de uno de los ejércitos azules desea atacar al mediodía. Su problema es el siguiente: Si envía un mensaje ordenando el ataque, no puede estar seguro de que el mensaje llegará. Puede pedir un acuse de recibo, pero éste también podría no llegar. ¿Hay algún protocolo que puedan usar los dos ejércitos azules para evitar la derrota?

12.6 En la figura 12.6, una sola unidad (le datos (le protocolo (PDU) del nivel *N* queda encapsulada dentro (le una PDU de nivel *(N - 1). Es* posible dividir (segmentación) una PDU de nivel *N* en varias PDU (le nivel *(N- I*) o agrupar (agrupación) varias PDU de nivel *N* en una sola PDU de nivel (,N- I).

a) En el caso de la segmentación, ¿es necesario que cada segmento (le nivel *(N - I) con*tenga una copia de la cabecera de nivel *N?*

b) En el caso de la agrupación, ¿es necesario que cada PDU de nivel *N* conserve su propia cabecera o pueden los datos concentrarse en una sola PDUJ de nivel *N* con una única cabecera de nivel *N?*

CAPÍTULO 13

Gestión distribuida de procesos

En este capítulo se examinan los mecanismos clave utilizados en los sistemas operativos distribuidos. la migración de procesos es la posibilidad de mover un proceso activo de una máquina a otra: este tema se ha convertido en un punto cada vez más crítico en los sistemas operativos distribuidos.

A continuación, se atenderá a la cuestión de cómo pueden coordinar sus actividades procesos de sistemas diferentes, si cada uno está gobernado por un reloj local y se produce un retardo en el intercambio de información.

Para finalizar, se abordarán dos aspectos clave de la gestión distribuida de procesos: la exclusión mutua y el interbloqueo.

13.1 MIGRACIÓN DE PROCESOS

La migración de procesos es la transferencia de una parte suficiente del estado de un proceso desde una máquina a otra para que el proceso se pueda ejecutarse en la máquina de destino. El interés en este concepto surgió de la investigación sobre formas de equilibrar la carga entre varios sistemas en red, aunque la aplicación del concepto se extiende actualmente más allá de este campo.

Anteriormente, sólo algunas de las numerosas publicaciones sobre la distribución de la carga se basaban en implementaciones reales de migración de procesos, lo que incluye la capacidad de expulsar un proceso de una máquina y reactivarlo más tarde en otra máquina. La experiencia demostró que era posible la migración apropiativa de procesos, no sin producir una complejidad y costes mayores que los previstos [ARTS89a]. Este coste condujo a algunos observadores a concluir que la migración de procesos no era práctica. Por ejemplo, considérese la siguiente afirmación [TANE85]:

La migración real de procesos en ejecución es trivial en teoría, pero cerca de lo imposible en la práctica.

Se ha comprobado que tales aseveraciones son demasiado pesimistas. Las nuevas implementaciones, incluyendo las de productos comerciales, han alimentado un continuo interés y nuevos desarrollos en este campo. Esta sección ofrece una visión general.

Motivación

La migración de procesos es deseable en los sistemas distribuidos por una serie de razones, incluyendo las siguientes [JUL88, SMIT88]:

- *Compartición d*e carga. Trasladando los procesos desde sistemas muy sobrecargados hacia otros menos cargados, la carga puede equilibrarse y así mejorar el rendimiento global. Los datos empíricos sugieren que es posible una mejora significativa de rendimiento [CABR86, LELA86]. Sin embargo, hay que tener cuidado en el diseño de los algoritmos de equilibrado de cara. [EAGES6] señala que el rendimiento empeora con el aumento de la comunicación necesaria para que el sistema distribuido realice el equilibrado. Puede encontrarse una discusión al respecto, con referencias a otros estudios, en [ESKI90].

- *Rendimiento de las comunicaciones:* Los procesos que interactúan de forma intensiva pueden moverse a un mismo nodo para reducir el coste de las comunicaciones durante su interacción. Además, cuando un proceso realiza un análisis de datos sobre algún archivo o conjunto de archivos mayores que el tamaño del proceso, puede resultar más ventajoso llevar al proceso hasta los datos que a la inversa.

- *Fiabilidad:* Los procesos que duran mucho tiempo pueden necesitar moverse para sobrevivir frente a los fallos de los que se pueda obtener previo aviso o en previsión del plazo planificado. Si el sistema operativo proporciona tal aviso, un proceso que desea continuar puede emigrar a otro sistema o bien asegurarse de que se reanudará más tarde en el sistema actual.

- *Utilización de características especiales*: Un proceso puede trasladarse para sacar partido de algunas características de hardware o software disponibles en un nodo determinado.

Mecanismos de Migración de Procesos

Hace falta considerar una serie de cuestiones a la hora de diseñar un servicio de migración de procesos. Entre ellas se encuentran las siguientes:

- ¿Quién da inicio a la migración?
- ¿Qué "parte" del proceso emigra?
- ¿Qué les ocurre a los mensajes y señales pendientes`?

Inicio de la Migración

Quién inicia la migración dependerá del objetivo del servicio de migración. Si el objetivo es equilibrar la carga, algún módulo supervisor de la carga del sistema operativo es normalmente el responsable de decidir cuándo tendrá lugar la migración. Este módulo es responsable de expulsar o indicar a un proceso que va a emigrar. Para determinar dónde va a emigrarse, el módulo necesita estar en comunicación con módulos similares de otros sistemas, de forma que se pueda supervisar la composición de la carga de otros sistemas. Si el objetivo es llegar hasta unos recursos determinados, el proceso puede emigrar por sí mismo cuando surja la necesidad. En este último caso, el proceso debe ser consciente de la existencia de un sistema distribuido. En el primer caso, la función de migración por completo puede ser transparente al proceso, así como la existencia de varios sistemas.

¿Qué emigra?

Cuando un proceso emigra, hace falta destruirlo en el sistema de origen y crearlo en el sistema de destino. Esto es un movimiento de procesos y no una duplicación. Por tanto, debe moverse la imagen del proceso, que consta de, por lo menos, el bloque de control del proceso. Además, debe actualizarse cualquier enlace entre éste y otros procesos, como los de paso de mensajes y señales. En la figura 13.1 se ilustran estas ideas. El proceso 3 ha emigrado fuera de la máquina S para convertirse en el proceso 4 de la máquina D. Todos los identificadores de enlace adquiridos por los procesos (denotados en minúscula) permanecen como antes. Es responsabilidad del sistema operativo mover el bloque de control del proceso y actualizar las correspondencias de los enlaces. La transferencia del proceso de una máquina a otra es invisible al proceso que emigra y a sus asociados en la comunicación. El movimiento del bloque de control del proceso es sencillo. Desde el punto de vista del rendimiento, la dificultad estriba en el espacio de direcciones del proceso y en los archivos abiertos que tenga asignados. Considérese primero el espacio de direcciones y supóngase que se está utilizando un esquema de memoria virtual (segmentación y/o paginación). Pueden surgir dos estrategias:

1. Transferir todo el espacio de direcciones en el momento de la migración. Este es el método más elegante. No hace falta dejar rastro del proceso en el sistema anterior. Sin embargo, si el espacio de direcciones es muy grande y si es probable que el proceso no necesite la mayor parte. este procedimiento puede ser costoso sin necesidad.
2. Transferir sólo aquella parte del espacio de direcciones que reside en memoria principal. Cualquier bloque adicional del espacio de direcciones virtuales será transferido sólo bajo demanda. Esto minimiza la cantidad de datos que se transfieren. Sin embargo, se necesita que la máquina de origen siga involucrada en la vida del proceso, manteniendo entradas en la tabla de páginas y/o segmentos.

Si es probable que el proceso no utilice eran parte de su espacio de direcciones no residente (por ejemplo, el proceso sólo esta accediendo temporalmente a otra máquina para trabajar con un archivo y retornará pronto), la segunda estrategia tiene más sentido. Por otra parte, si finalmente se accede a una gran parte del espacio de direcciones no residente, la transferencia a trozos de bloques del espacio de direcciones puede ser menos eficaz que transferir simplemente todo el espacio de direcciones en el momento de la migración.

En muchos casos. puede que no sea posible conocer por adelantado si hará falta gran parte del espacio de direcciones no residente. Sin embargo, si los procesos se estructuran en hilos y si la unidad elemental de la migración es el hilo en vez del proceso, la segunda estrategia mencionada parece ser la mejor. De hecho, la segunda estrategia es casi de obligado cumplimiento puesto que los hilos restantes del proceso que no se llevan consigo y también necesitan acceder al espacio de direcciones del proceso. La migración de hilos está implementada en el sistema operativo Emerald [JUL88, JUL89].

Pueden aplicarse ideas similares al traslado de los archivos abiertos. Si el archivo está inicialmente en el mismo sistema que el proceso que va a emigrar y si el archivo está bloqueado para acceso exclusivo por parte de dicho proceso, puede tener sentido transferir el archivo junto con el proceso. El peligro aquí radica en que el proceso puede desaparecer temporalmente y no hacer falta el archivo hasta su vuelta. Por tanto, puede tener sentido transferir el archivo completo sólo después de que el proceso emigrado haga una solicitud de acceso.

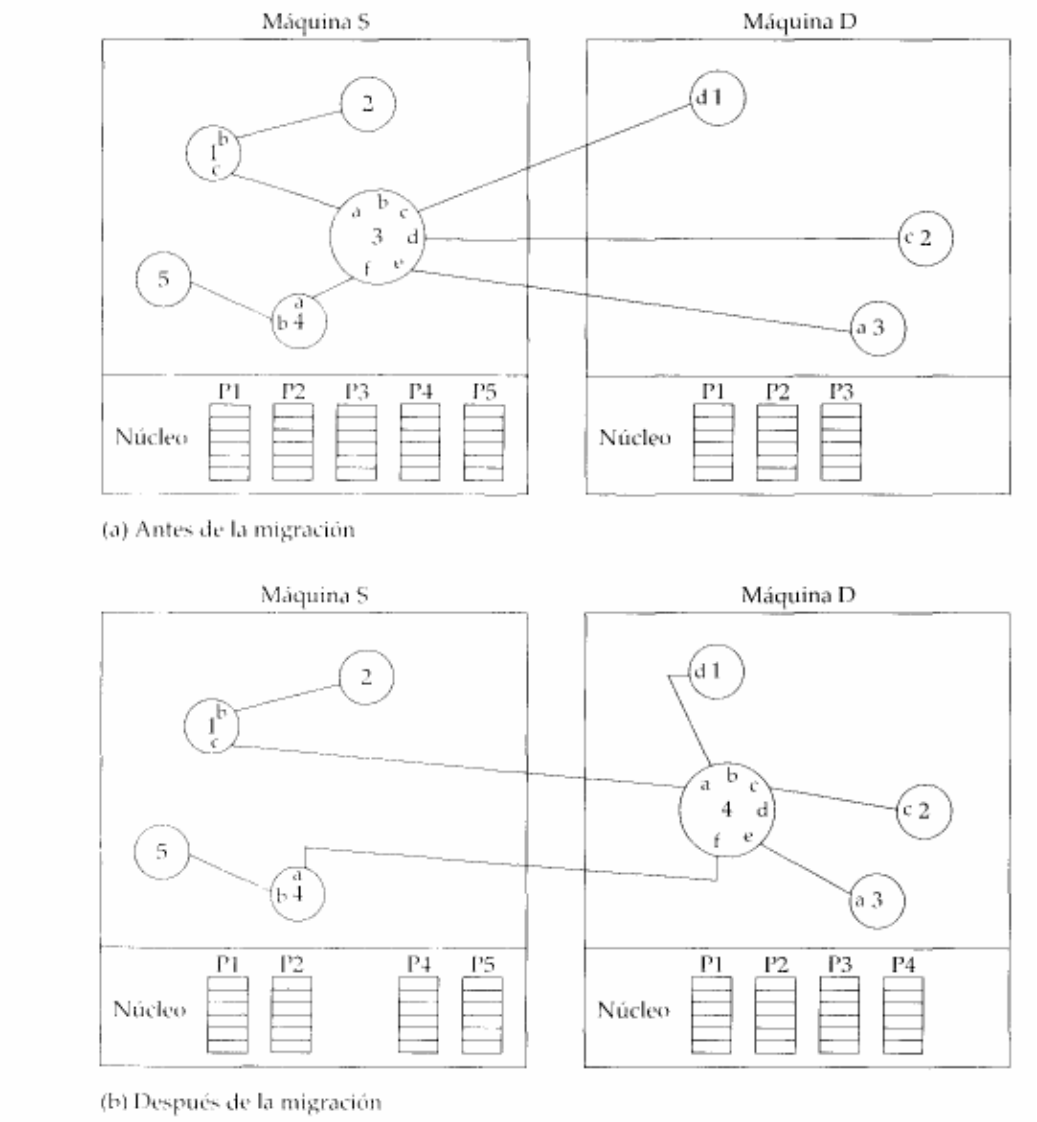


FIGURA 13.1 Ejemplo de migración de procesos

Si se permite disponer de cache. como en el sistema Sprite (figura 12.18), se presenta una complicación más. Por ejemplo, si un proceso tiene un archivo abierto para escritura, hace un *fork*, y emigra el proceso hijo. el archivo estaría entonces abierto para escritura en dos máquinas diferentes. El algoritmo de consistencia de caches de Sprite dispone que no se mantenga al archivo en la cache de las máquinas donde están ejecutando ambos procesos [DOUG89].

Mensajes y *Señales*

La cuestión final antes enumerada, sobre el destino de los mensajes y las señales pendientes, se puede tratar mediante un mecanismo de almacenamiento temporal, durante la migración,

de los mensajes y señales pendientes para, posteriormente, dirigirlos a su nuevo destino. Puede hacer falta mantener una información de desvío en el emplazamiento inicial durante algún tiempo, para asegurar que llegan todos los mensajes y las señales pendientes.

Un Escenario de Migración

Como ejemplo representativo de la automigración, se puede considerar el servicio disponible en el sistema operativo AIX de IBM [WALK89], que es un sistema operativo UNIX distribuido. Un servicio similar está disponible en el sistema operativo LOCUS [POPE85] y, en realidad, el sistema AIX está basado en el desarrollo de LOCUS.

Se produce 1a siguiente secuencia de sucesos:

1. Cuando un proceso decide por sí mismo emigrar, selecciona una máquina de destino y envía un mensaje de tarea remota. El mensaje lleva una parte de la imagen del proceso y de información de archivos abiertos.
2. En el emplazamiento receptor, un proceso servidor del núcleo crea un hijo y le cede esta información.
3. El nuevo proceso extrae los datos, los argumentos, la información del entorno y de la pila que necesita hasta completar su operación. El código del programa se copia si se ha alterado y si no, se pagina por demanda en el sistema de archivos global.
4. Se indica con una señal al proceso originario que la migración ha terminado. Este proceso envía un mensaje final de terminación al nuevo proceso y se destruye.

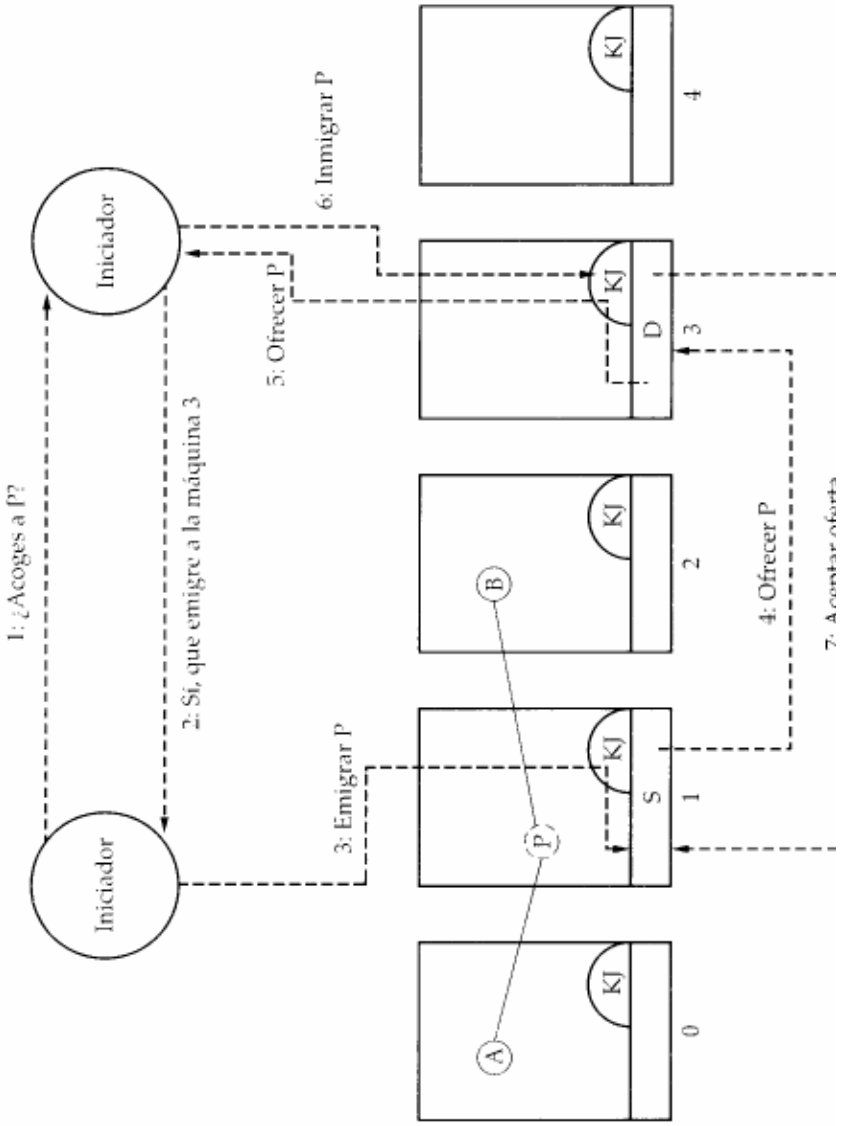
Una secuencia similar se produce cuando otro proceso inicia la migración. La diferencia principal es que el proceso que va a emigrar debe suspenderse de manera que puede hacerlo en un estado de no ejecución. Este procedimiento es el seguido, por ejemplo, en Sprite [DOUG89].

En el escenario anterior, la migración es una actividad dinámica en la que intervienen una serie de pasos para trasladar la imagen de un proceso. Cuando la migración la inicia otro proceso, en vez de ser automigración, un posible enfoque consiste en copiar la imagen del proceso y todo su espacio de direcciones a un archivo, destruir el proceso, copiar el archivo a la otra máquina mediante un servicio de transferencia de archivos y volver a crear el proceso a partir del archivo de la máquina de destino. [SMIT89] describe este enfoque.

Negociación de la Migración

Otro aspecto de la migración de procesos está relacionado con la decisión de emigrar. En algunos casos, la decisión la torna una única entidad. Por ejemplo, si se pretende equilibrar la carga, un módulo supervisará la carga relativa de varias máquinas y llevará a cabo la migración si es necesaria, para mantener la carga equilibrada. Si se usa automigración para hacer que los procesos accedan a servicios especiales o grandes archivos remotos, el mismo proceso puede tomar la decisión. Sin embargo, algunos sistemas permiten que el destino designado participe en la decisión, entre otras cosas para conservar el tiempo de respuesta a los usuarios. Un usuario de un puesto de trabajo, por ejemplo, puede sufrir una degradación notable del tiempo de respuesta si hay procesos que emigran al sistema del usuario, incluso si dicha migración sirve para conseguir un mejor equilibrado global.

Un ejemplo de mecanismo de migración se puede encontrar en Charlotte [ARTS89b, FINK89]. La política de migración (cuándo emigrar, qué proceso y a qué destino) es responsabilidad de la utilidad Iniciadora (Starter), que es un proceso responsable además de la


Iniciador
Iniciador
1: ¿Acoges a P?
2: Sí, que emigre a la máquina 3
3: Emigrar P
4: Ofrecer P
5: Ofrecer P
6: Inmigrar P
KJ
S
D
A
B
P
0
1
2
3
4

planificación a largo plazo y de la asignación de memoria. El Iniciador puede, por tanto, coordinar las políticas de estos tres campos. Cada proceso Iniciador puede controlar un grupo de máquinas. El Iniciador recibe oportunamente del núcleo de cada máquina unas estadísticas de carga bastante elaboradas.

Los dos procesos Iniciadores deben tomar en común la decisión de emigrar, como se ilustra en la figura 1 3.2. Para ello se suceden las siguientes etapas:

1. El Iniciador que controla el sistema de origen (S) decide que un proceso (P) debe emigrar a un sistema de destino determinado (D). Entonces envía un mensaje al Iniciador de D solicitando la transferencia.
2. Si el Iniciador de D está preparado para recibir al proceso, devuelve un acuse de recibo afirmativo
3. El Iniciador de S le comunica su decisión al núcleo de S, a través de la llamada a un servicio (si el iniciador se está ejecutando en S) o mediante un mensaje a la tarea del núcleo *o KernJob (KJ)* de la máquina S, que es un proceso que sirve para convertir los mensajes procedentes de procesos remotos en llamadas a servicios.
4. El núcleo de S se ofrece entonces para enviar el proceso a D. En la oferta se incluyen estadísticas sobre P. como pueden ser su estado y la carga de procesador y de comunicaciones que conlleva.
5. Si D anda escaso de recursos, puede rechazar la oferta. En otro caso, el núcleo de D propone la olerla a su Iniciador. En la propuesta se incluye la misma información recibida de S.
6. La decisión según la política del Iniciador es comunicada a D por medio de una llamada *MigrateIn* (Inmigrar)
7. D reserva los recursos necesarios para evitar problemas de interbloqueo y control de flujo y, posteriormente, envía a S una aprobación.

En la figura 13.2 también aparecen otros dos procesos, A y B, que tienen enlaces abiertos con P. Si se siguen los pasos anteriores, la máquina 1 donde reside S debe enviar mensajes de actualización de enlaces a las máquinas 0 y 2 para conservarlos. Los mensajes de actualización de enlaces comunican la nueva dirección de cada enlace reservado por P y son confirmados por el núcleo a efectos de sincronización. A partir de entonces, un mensaje enviado a P por cualquiera de sus enlaces irá directamente a D. Estos mensajes pueden intercambiarse de forma concurrente con los pasos antes descritos.

Finalmente, tras el paso 7 y después de que todos los enlaces se hayan actualizado, S recogerá el contexto de P en un solo mensaje y lo envía a D.

La máquina 4 también está ejecutando Charlotte, pero no está involucrada en la migración y, por tanto, esta vez no tiene comunicación con los otros sistemas.

Desalojo

El proceso de negociación permite que un sistema de destino rechace la inmigración de un proceso. Además, puede ser útil hacer que un sistema desaloje un proceso que ha emigrado hacia él. Por ejemplo, si un puesto de trabajo se activa, puede hacer falta desalojar los procesos inmigrados para ofrecer un tiempo de respuesta apropiado.

Un ejemplo de la posibilidad ele desalojo se tiene en Sprite [DOUG89]. En Sprite, que es un sistema operativo de puestos de trabajo, cada proceso parece ejecutarse en una única má-

quina *(host)* durante toda su vida. Este *host* se conoce como el *nodo de origen* del proceso. Si un proceso emigra, se convierte en un *proceso extranjero* en la máquina de destino. En cualquier momento, la máquina de destino puede desalojar al proceso extranjero, que es obligado a volver a su nodo de origen

Los elementos del mecanismo de desalojo de Sprite son los siguientes:

1. Un proceso supervisor en cada nodo lleva la cuenta de la carga actual para determinar cuándo se pueden aceptar nuevos procesos extranjeros. Si el supervisor detecta actividad en 1a consola del puesto de trabajo, inicia un procedimiento de desalojo para cada proceso extranjero.
2. Si se desaloja un proceso, éste volverá a su nodo de origen. El proceso puede emigrar de nuevo si hay algún otro nodo disponible.
3. Aunque desalojar todos los procesos puede tardar algún tiempo, los designados para expulsión son suspendidos de inmediato. Permitir que un proceso expulsado se ejecute mientras espera su desalojo reduciría el tiempo que el proceso queda congelado, pero también reduciría la potencia de procesamiento disponible en el *host* mientras se espera el desalojo.
4. El espacio de direcciones por completo de un proceso desalojado es transferido al nodo (1e origen El tiempo de migración de un proceso puede verse reducido considerablemente se recupera del *host* previo la imagen de memoria del proceso emigrado. Sin embargo, esto exige que el *host* anterior dedique recursos y satisfaga las peticiones de servicio del proceso desalojado por más tiempo que el necesario.

Transferencias Apropiativas y No Apropiativas

La discusión en esta sección se centra en la migración apropiativa de procesos, que consiste en la transferencia de un proceso parcialmente ejecutado o, al menos, cuya creación se haya completado. Una función más simple es la transferencia no apropiativa, que involucra solamente a procesos que no lean comenzado su ejecución y, por tanto, no se necesita transferir el estado del proceso. En ambos casos, debe transferirse información al nodo remoto sobre el entorno de ejecución del proceso. Esta puede incluir el directorio de trabajo actual del usuario y los privilegios heredados por el proceso.

La migración no apropiativa de procesos puede ser útil en el equilibrado de la carga (véase como ejemplo [SHIV921]). Tiene la ventaja de evitar el coste de una migración completa de los procesos. La desventaja es que dicho esquema no reacciona bien ante cambios repentinos en la distribución de la carga.

13.2 ESTADOS GLOBALES DISTRIBUIDOS

Estados Globales e Instantáneas Distribuidas

Todos los problemas de concurrencia que se plantea un sistema fuertemente acoplado, como la exclusión mutua, el interbloqueo y la inanición, también aparecen en un sistema distribuido. Las estrategias de diseño en este campo se complican con el hecho de que no existe

un estado global del sistema. Es decir, no es posible que el sistema operativo o algún proceso conozcan el estado actual de todos los procesos del sistema distribuido. Un proceso puede conocer solamente el estado actual de lodos los procesos del sistema local, accediendo a los bloques de control de proceso en la memoria. De los procesos remotos, un proceso sólo puede conocer la información de estado que recibe mediante mensajes, pero estos representan el estado del proceso remoto en algún instante pasado. Esta situación es similar a la de la astronomía: El conocimiento de una estrella o galaxia lejana consiste en una serie de ondas luminosas y electromagnéticas que proceden de un objeto lejano y estas ondas ofrecen una imagen del objeto correspondiente a algún momento del pasado. Por ejemplo, el conocimiento que se tiene de un objeto situado a una distancia de 5 años-luz tiene 5 años de antigüedad en el momento en que se recibe.

Los retardos temporales impuestos por la naturaleza de los sistemas distribuidos complican todas las cuestiones relativas a la concurrencia. Para ilustrar esto, se presenta un ejemplo tomado de [ANDR90]. Para representar el problema, se utilizarán grafos de procesos/eventos (figuras 13.3 y 13.4). En estos grafos se emplea una línea horizontal para cada proceso que representa el eje del tiempo. Un punto sobre la línea corresponde a un suceso (por ejemplo, un suceso interno de un proceso, un envío o una recepción de un mensaje). Un recuadro alrededor de un punto representa una instantánea del estado local del proceso tornada en dicho punto. Una flecha representa un mensaje entre dos procesos.

En el ejemplo, un individuo dispone de una cuenta bancaria distribuida en dos sucursales de un banco. Para determinar la cantidad total disponible en la cuenta del cliente, el banco debe averiguar la cantidad guardada en cada sucursal. Supóngase que la resolución se produce exactamente a las 3:00 **p.m.** En la figura 13.3 se ofrece un caso en el que el balance de la cuenta combinada es de 100,00 dólares. Pero la situación de la figura 13.3h también es posible. Esta vez, el balance en la sucursal A está de camino hacia la sucursal B en el instante de observación; el resultado es una falsa lectura de 0,00 dólares. Este problema particular puede resolverse examinando todos los mensajes en camino en el momento de observación. La sucursal A mantendrá un registro de todas las transferencias desde la cuenta, junto con la identidad del destinatario de la transferencia. Por tanto, en el "estado" de la cuenta de cada sucursal se incluirá el balance actual y un registro de transferencias. Cuando se examinen las dos cuentas, el observador hallará que una transferencia ha partido de la sucursal A con destino a la cuenta del cliente en la sucursal B. Como la cantidad no ha llegado todavía a la sucursal B, se añadirá al balance total. Cualquier cantidad que se haya transferido y recibido se contabiliza una sola vez, como parte del balance de la cuenta de destino.

Esta estrategia no es infalible, como se demuestra en la figura 13.3c. En este ejemplo, los relojes de ambas sucursales no están perfectamente sincronizados. El estado de la cuenta del cliente en la sucursal A alas 3:00 p.m. muestra un balance de 100,00 dólares. Sin embargo, esta cantidad será transferida posteriormente a la sucursal B, a las 3:01 según el reloj de A, pero llegará a B a las 2:59 según el reloj de B. Por tanto, la cantidad se contabilizará dos veces si se observa a las 3:00 p.m. de B.

Para comprender la dificultad afrontada y poder formular una solución, se van a definir los siguientes términos:

- Canal: Existe un canal entre dos procesos siempre que intercambien mensajes. Se puede pensar en un canal como un camino o un medio por el que se transmiten los mensajes. Por conveniencia, los canales se contemplarán como si fueran de un solo sentido. De este

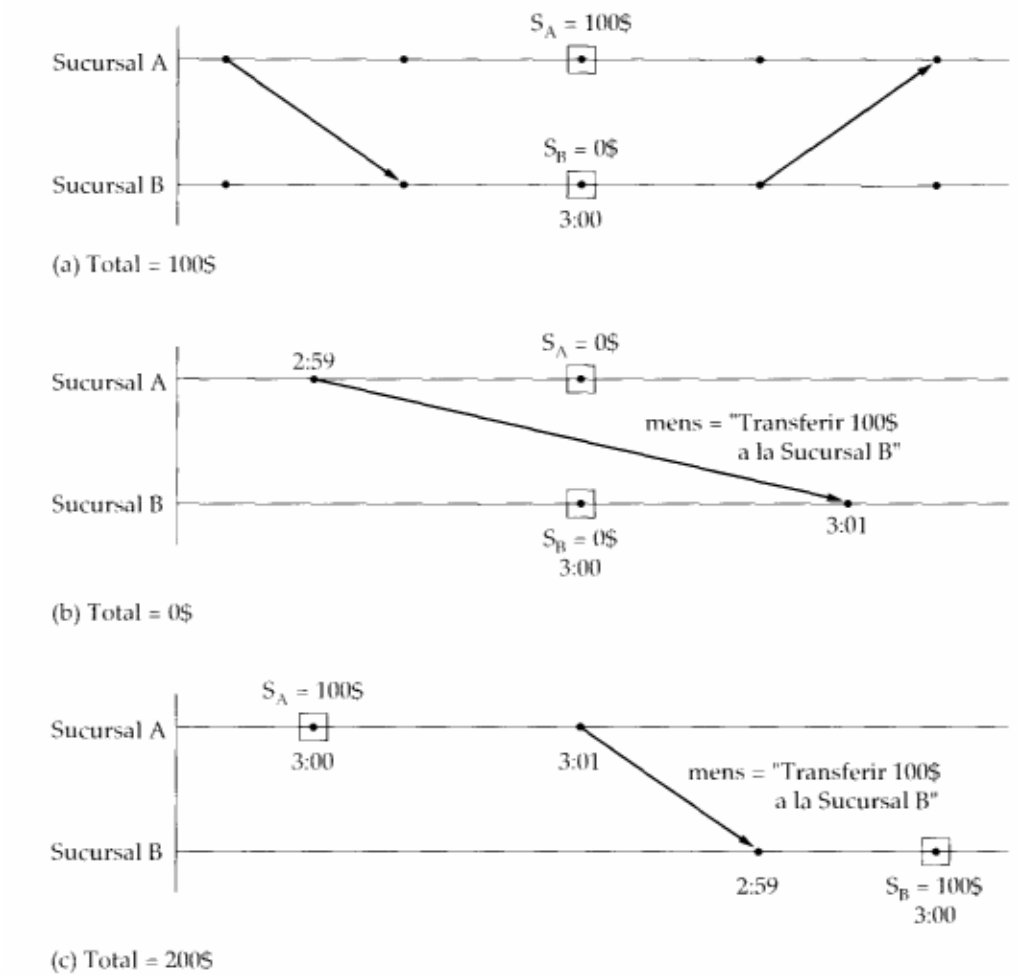


FIGURA 13.3 Ejemplo de resolución del estado global

modo, si dos procesos intercambian mensajes, se necesitarán dos canales, uno para cada dirección de la transferencia.

- *Estado:* El estado de un proceso es la secuencia de mensajes enviados y recibidos por los canales del proceso.
- *Instantánea:* Una instantánea registra el estado de un proceso. Cada instantánea incluye un registro de todos los mensajes enviados y recibidos por todos los canales desde la instantánea anterior.
- *Estado global:* El estado combinado de todos los procesos.
- *Instantánea distribuida:* Un conjunto de instantáneas, una para cada proceso.

El problema que se afronta es no poder determinar un estado global cine sea cierto, debido al lapso de tiempo asociado con cada transferencia de un mensaje. Se puede intentar definir un estado global reuniendo instantáneas de todos los procesos. Por ejemplo, el estado global

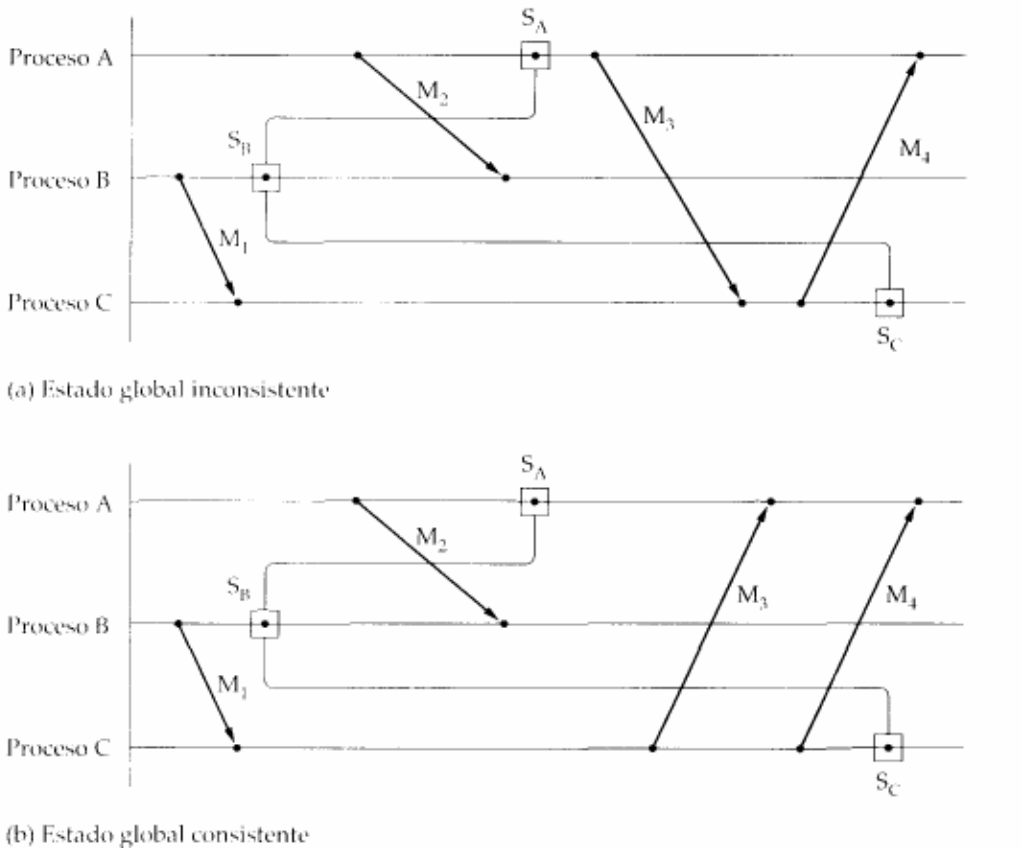


FIGURA 13.4 Estados globales consistentes e inconsistentes

de la figura 13.4a en el momento de tomar las instantáneas muestra un mensaje de camino por el canal <A, B>; uno de camino por el <A, C> y otro de camino por el canal <C, A>. Los mensajes 2 y 4 están bien representados, pero no lo está el mensaje 3. La instantánea distribuida indica que este menaje se ha recibido, pero ¡aún no se ha enviado!

Se quiere que en la instantánea distribuida se registre un estado global consistente. Un estado global es consistente si para cada estado de un proceso que registra la recepción de un mensaje, el envío de dicho mensaje ya se ha registrado en el estado del proceso que lo envió. En la figura 13.4b se ofrece un ejemplo. Un estado global inconsistente surge cuando un proceso ha registrado la recepción (le un mensaje, pero el proceso emisor correspondiente no ha registrado que el mensaje se envió (figura-13.4a).

Algoritmo de Instantáneas Distribuidas

En [CHAN85] se describe un algoritmo de instantáneas distribuidas en el que se registra un estado global consistente. El algoritmo supone que los mensajes se entregan en el mismo orden en que son enviados y que no se producen pérdidas. Un protocolo de transporte fiable (como TCP cumpliría con estos requisitos. El algoritmo hace uso de un mensaje de control

Algunos procesos dan comienzo al algoritmo registrando su estado y enviando un marcador por todos los canales salientes antes de enviar ningún mensaje. Después, cada proceso procede de la manera siguiente. Al recibir el primer marcador (supóngase que desde el proceso *q),* el proceso receptor da los pasos siguientes:

1 . p registra su estado local Sp.
2. p registra un estado vacío para el canal que va desde *q* a p.
3. p propaga el marcador por todos los canales salientes hacia todos sus vecinos.

Estos pasos deben darse de forma atómica, es decir, p no puede enviar o recibir mensajes hasta que se hayan completado los tres pasos.

En cualquier instante tras registrar su estado, cuando p reciba un marcador de algún otro canal entrante (supóngase que desde el proceso r), realiza lo siguiente:

1. p registra el estado del canal de r a p como la secuencia de mensajes que p ha recibido de r desde que p registró su estado local Sp hasta que recibió el marcador de r.

El algoritmo termina en un proceso una vez que se haya recibido el marcador por todos los canales entrantes.

[ANDR90] hace las siguientes observaciones sobre el algoritmo:

1. Cualquier proceso puede iniciar el algoritmo enviando un marcador. De hecho, varios nodos podrían decidir registrar el estado de forma independiente y aún así el algoritmo terminaría bien.
2. El algoritmo terminará en un tiempo finito si cada mensaje (incluyendo los marcadores) se entrega en un tiempo finito.
3. Este algoritmo es distribuido: Cada proceso es responsable de registrar su propio estado y el de todos los canales entrantes.
4. Una vez que todos los estados se hayan registrado (el algoritmo ha terminado en todos los procesos), el estado global consistente que obtiene el algoritmo puede reunirse en cada proceso haciendo que cada uno envíe por todos los canales salientes los datos de estado que haya registrado y que también retransmitan por dichos canales los datos de estado que reciben. Otra solución sería que el proceso iniciador muestree a todos los procesos para obtener el estado global.
5. El algoritmo no afecta ni se ve afectado por cualquier otro algoritmo distribuido en el que participen los procesos.

Como ejemplo del empleo del algoritmo (tomado de [BEN90]), considérese el conjunto de procesos que se ilustra en la figura 13.5. Cada proceso está representado por un nodo y cada canal unidireccional se representa con una línea entre dos nodos, con la dirección indicada por una flecha. Supóngase que se ejecuta el algoritmo de instantáneas, enviándose nueve mensajes por cada uno de los canales salientes de cada proceso. El proceso 1 decide registrar el estado global después de enviar tres mensajes. Al terminar, se recogen las instantáneas de cada proceso; los resultados se muestran en la figura 13.6. El proceso 2 envió cuatro mensajes porcada uno de los dos canales salientes hacia los procesos 3 y 4 antes de registrar su estado. Recibió cuatro mensajes del proceso 1 antes de registrar su estado, dejando que los mensajes 5 y 6 se asocien con el canal. Si se comprueba la consistencia de la instantánea, cada mensaje enviado fue recibido en el proceso de destino o bien registrado como en tránsito por el canal.

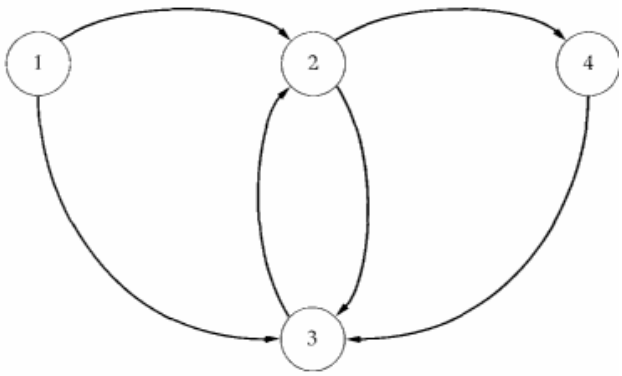


FIGURA 13.5 Grafo de procesos y canales

El algoritmo de instantáneas distribuidas es una herramienta potente y flexible. Puede utilizarse para adaptar cualquier algoritmo centralizado a un entorno distribuido, ya que lo fundamental de cualquier algoritmo centralizado es el conocimiento del estado global. Algunos ejemplos específicos incluyen una detección de interbloqueo y una detección de la terminación de los procesos (como ejemplo, puede consultarse [BEN90]). También se puede usar para ofrecer un punto de control de algoritmos distribuidos para la recuperación y vuelta atrás de un fallo que se detecte.

Proceso 1
Canales salientes
2 envió 1,2,3,4,5,6
3 envió 1,2,3,4,5,6
Canales entrantes
Proceso 2
Canales salientes
3 envió 1,2,3,4
4 envió 1,2,3,4
Canales entrantes
1 recibió 1,23,4 y guardó 5,6
3 recibió 1,2,3,4,5,6,7,8

Proceso 3
Canales salientes
2 envió 1,2,3,4,5,6,7,8
Canales entrantes
1 recibió 1,2,3 y guardó 4,5,6
2 recibió 1,2,3 y guardó 4
4 recibió 1,2,3
Proceso 4
Canales salientes
3 envió 1,2,3
Canales entrantes
2 recibió 1,2 y guardó 3,4

FIGURA 13.6 Un ejemplo de instantánea

13.3

GESTIÓN DISTRIBUIDA DE PROCESOS - EXCLUSIÓN MUTUA

Hay que recordar que en los capítulos 4 y 5 se trataron los problemas relativos a la ejecución ele procesos concurrentes. Las dos cuestiones clave que surgieron fueron la exclusión mutua y el interbloqueo. El capítulo se centraba en las soluciones a este problema en el contexto de un sistema sencillo que disponía de uno o más procesadores y una memoria principal común. Al tratar con un sistema distribuido y un conjunto de procesadores que no comparten la memoria principal, surgen nuevas dificultades y se demandan nuevas soluciones. Los algoritmos de exclusión mutua e interbloqueo deben depender del intercambio de mensajes y no pueden hacerlo del acceso a una memoria común. En esta sección y en la siguiente, se examinará la exclusión mutua y el interbloqueo en el contexto de un sistema operativo distribuido.

Exclusión Mutua Distribuida

Cuando dos o más procesos compiten por el uso de los recursos del sistema, hay necesidad de un mecanismo que haga cumplir la exclusión mutua. Supóngase que dos o más procesos requieren el acceso a un único recurso no compartible, como una impresora. Durante el curso de la ejecución, cada proceso enviará órdenes al dispositivo de E/S, recibirá información de estado, enviará y/o recibirá datos. Dicho recurso será conocido como recurso crítico y la parte del programa que lo usa será una sección crítica del programa. Es importante que se deje entrar sólo a un programa en su sección crítica en un instante. No se puede confiar simplemente en el sistema operativo para comprender y hacer cumplir esta restricción, porque el requisito detallado puede no ser obvio. En el caso de la impresora, por ejemplo, se desea que cualquier proceso individual tenga control de la misma mientras imprime un archivo completo. En otro caso, se entremezclarán las líneas de los procesos que tornen parte.

Para un uso afortunado de la concurrencia entre procesos se necesita la capacidad para definir secciones críticas y hacerlas cumplir. Esta capacidad es fundamental para cualquier esquema de proceso concurrente. Cualquier servicio o capacidad que proporcione soporte para la exclusión mutua debe cumplir los siguientes requisitos:

1 .La exclusión mutua debe hacerse cumplir: En un instante dado, sólo se deja entrar a un proceso en la sección crítica del servicio, de entre todos los procesos que tengan secciones críticas para el mismo recurso u objeto compartido.
2.Un proceso que se detiene en su sección no crítica debe hacerlo sin interferir en los otros procesos.
3. No debe ser posible que un proceso que solicita acceso a la sección crítica sea postergado indefinidamente: ausencia de interbloqueo o inanición.
4. Cuando no haya ningún proceso en la sección crítica, se debe permitir entrar sin dilación a cualquiera que solicite la entrada en su sección crítica.

5. No se hacen suposiciones sobre las velocidades relativas o el número de procesadores.

6. Un proceso permanece dentro de su sección crítica sólo durante un tiempo finito.

En la figura 13.7 se puede apreciar un modelo que puede usarse para examinar los enfoques de la exclusión mutua en un contexto distribuido. Se supone que hay un conjunto de sistemas interconectados por algún tipo de servicio de red. Se supone que alguna función o algún proceso del sistema operativo es responsable de la asignación de recursos. Cada proceso controla una serie de recursos y sirve a un conjunto de procesos de usuario. La tarea es idear un algoritmo por el que estos procesos puedan cooperar en el cumplimiento de la exclusión mutua.

Los algoritmos de exclusión mutua pueden ser centralizados o distribuidos. En un algoritmo totalmente **centralizado,** se designa a un nodo como el nodo de control del acceso a todos los objetos compartidos. Cuando un proceso requiera el acceso a un recurso crítico, emitirá una Petición a su proceso local de control de recursos. Este proceso, a su vez, envía un mensaje de Petición al nodo de control, quien devuelve un mensaje de Respuesta (permiso) cuando el objeto compartido esté disponible. Cuando un proceso ha terminado con un recurso, se envía un mensaje de Liberación al nodo de control. Este algoritmo centralizado tiene dos propiedades básicas:

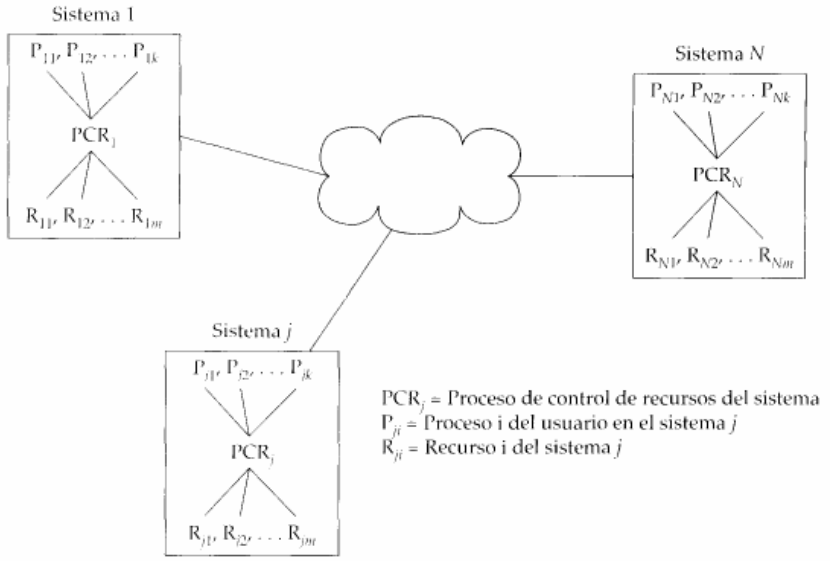


FIGURA 13.7 Modelo para el problema de la exclusión mutua en la gestión distribuida de procesos

1. Sólo el nodo de control torna decisiones de asignación de recursos.
2. Toda la información necesaria se concentra en el nodo de control, incluyendo la identidad y ubicación de todos los recursos, así como el estado de asignación de cada recurso.

El enfoque centralizado es sencillo y es fácil ver cómo se cumple la exclusión mutua: El nodo de control no aprobará una petición de un recurso hasta que éste haya sido liberado. Sin embargo dicho esquema sufre de varios inconvenientes. Si el nodo de control falla, el mecanismo de exclusión mutua dejará de funcionar, por lo menos temporalmente. Además, cada asignación y deasignación de recurso requiere el intercambio de mensajes con el nodo de control. De esta forma, el nodo de control puede convertirse en un cuello de botella.

Debido a los problemas de los algoritmos centralizados, se ha producido un mayor interés en el desarrollo de **algoritmos distribuidos.** Un algoritmo completamente distribuido se caracteriza por las siguientes propiedades [MAEK87]:

1. Todos los nodos disponen de una cantidad igual de información, por término medio.
2. Cada nodo dispone sólo de una representación parcial del sistema total y debe tomar decisiones basándose en esta información.
3. Todos los nodos tienen igual responsabilidad en la decisión final.
4. Todos los nodos dedican igual esfuerzo, por término medio, en llevar a cabo una decisión final.
5. El fallo de un nodo no provocará, en general, el colapso total del sistema.
6. No existe un reloj común a todo el sistema con el que regular el ritmo de los sucesos.

Los puntos 2 y 6 pueden requerir aleo más de detalle. Con respecto al punto 2, algunos algoritmos distribuidos requieren que toda la información conocida por un nodo sea comunicada a los otros nodos. Incluso en este caso, en un momento dado, parte de esa información estará de camino y no habrá llegado por completo a los otros nodos. Por tanto, debido a los retardos de la comunicación de mensajes, la información de un nodo no suele estar completamente actualizada y, en ese sentido, es sólo una información parcial.

Con respecto al punto 6, debido al retardo de la comunicación entre sistemas, es imposible mantener un reloj de todo el sistema que esté disponible instantáneamente para todos. Además, también es difícil mantener un reloj central y a todos los relojes locales sincronizados de forma precisa con el reloj central; con el tiempo, se producirá una deriva en los relojes locales que originará la pérdida de la sincronía.

El retardo de las comunicaciones, unido a la falta de un reloj común, hace mucho más difícil idear mecanismos de exclusión mutua para sistemas distribuidos, en comparación con los centralizados. Antes de ver algunos algoritmos de exclusión mutua distribuida, se examinará un método común para superar la dificultad del reloj.

Ordenación de Sucesos en un Sistema Distribuido

La ordenación temporal de sucesos es fundamental para la operación de la mayoría de los algoritmos distribuidos de exclusión mutua e interbloqueo. La carencia de un reloj común o de un medio de sincronizar los relojes locales es, por tanto, la restricción principal. El problema puede expresarse de la manera siguiente. Se desea poder decir si el suceso a del sistema *i* ocurrió antes (o después) del suceso b en el sistema *j* y ser capaz de llegar de forma consistente a esta conclusión en todos los sistemas de la red. Desgraciadamente, esta afir-

mación no es precisa por dos razones. En primer lugar, puede producirse un retardo entre el acontecimiento real de un suceso y el momento en que éste es observado en algún otro sistema. En segundo lugar, la falta de sincronización lleva a un desacuerdo en las lecturas de los relojes de sistemas diferentes.

Para superar estas dificultades, se ha propuesto un método conocido como registros ele *tiempo o marcas de tiempo* por parte de Lamport [LAMP78] que sirve para ordenar los sucesos en un sistema distribuido sin utilizar relojes físicos. Esta técnica es tan eficiente y efectiva que se usa en la gran mayoría de los algoritmos de exclusión mutua e interbloqueo.

Para comenzar, hace falta optar por una definición del término suceso. En el fondo, hay que preocuparse de las acciones que se producen en un sistema local, como la entrada o salida de procesos de su sección crítica. Sin embargo, en un sistema distribuido, la forma en que los procesos interaccionan es con mensajes. Por tanto, parece lógico asociar los sucesos con los mensajes. Un suceso local puede limitarse a un simple mensaje; por ejemplo, un proceso puede enviar un mensaje cuando desee entrar en su sección crítica o cuando salga de ella. Para evitar la ambigüedad, se asociarán los sucesos sólo con el envío de mensajes, no con la recepción. Así, cada vez cine un proceso transmita un mensaje, se definirá un suceso correspondiente al momento en que el mensaje sale del proceso.

El esquema de enarcas de tiempo tiene la finalidad de ordenar los sucesos consistentes en la transmisión de los mensajes. Cada sistema *i* de la red mantiene un contador local Ci, que funciona como un reloj. Cada vez que un sistema transmita un mensaje, incrementará su reloj en 1 . El mensaje se envía de la forma:

(m. T_i , i)

donde

m = contenido de mensaje

T = marca de tiempo del mensaje, puesta a C,

i = identificador numérico del nodo

Cuando se recibe un mensaje, el sistema receptor.¡ pone su reloj a uno más que el máximo de su valor actual y la marca de tiempo entrante:

C_i: = 1 + max[C_j, T_i]

En cada nodo, la ordenación de sucesos queda determinada por las reglas siguientes. Para los mensajes x del nodo i e y del nodo j, se dice que x precede a y si se cumple una de las siguientes condiciones:

Si Ti < Tj ó

Si Ti=Tj e i<j

El tiempo asociado con el suceso de cada mensaje es la marca de tiempo que acompaña al mensaje, mientras que la ordenación de estos tiempos se determina por las dos reglas anteriores. En ellas, dos sucesos de mensajes con las mismas marcas de tiempo se ordenan por sus nodos. Como la aplicación de estas reglas es independiente del nodo, este método evita cualquier problema de diferencia entre los diversos relojes de los procesos comunicantes.

Un ejemplo del funcionamiento de este algoritmo se ofrece en la figura 13.8, donde aparecen tres nodos, cada uno de los cuales está representado por un proceso que controla el algoritmo de marcas de tiempo. El proceso P, comienza con un valor del reloj de 0. Para trans-

mitir el mensaje *a*. incrementa su reloj y transmite (*a*, 1 , 1), donde el primer valor numérico es la marca de tiempo y el segundo es la identidad del nodo. Este mensaje es recibido por los procesos de los nodos 2 y 3. En ambos casos, los relojes locales tienen un valor 0 y se ponen a 2 = 1 + max[1, 0]. P_2 emite el siguiente mensaje, incrementando previamente su reloj hasta 3. Al recibir este mensaje, P_1 y P_3 deben incrementar sus relojes hasta 4. Entonces, P_1 emite el mensaje b y P_3 emite el mensaje *j* aproximadamente al mismo tiempo y con la misma marca de tiempo. Por el principio de ordenación, esto no causa confusión. Después de que todos estos sucesos hayan tenido lugar, la ordenación de los mensajes permanece igual en todos los nodos, es decir, (a, ,x, b, *j).*

El algoritmo funciona a pesar de las diferencias en tiempos de transmisión entre cada pareja de sistemas, como se ilustra en la figura 13.9, donde P_1 y P_4 aparecen emitiendo mensajes con la misma marca de tiempo. El mensaje de P_1 llega antes que el de P_4 al nodo 2, pero más tarde que el de P_4 al nodo 3. Sin embargo, después de que se reciban todos los mensajes en todos los nodos, la ordenación de los sucesos de mensaje es la misma en todos los nodos, es decir, {a, q}.

Nótese que la ordenación impuesta por este esquema no tiene por qué coincidir con la secuencia real en el tiempo. Para los algoritmos basados en este esquema de marcas de tiempo, no es importante qué suceso ocurre primero en realidad. Sólo importa que todos los procesos que implementen el algoritmo estén de acuerdo en la ordenación que se impone sobre los sucesos.

En los dos ejemplos recién estudiados. cada mensaje se envía desde un proceso a todos los demás. Si algunos mensajes no se envían de esta forma, algunos nodos no recibirán todos los mensajes del sistema y, por consecuentemente, es imposible que todos los nodos dispon-

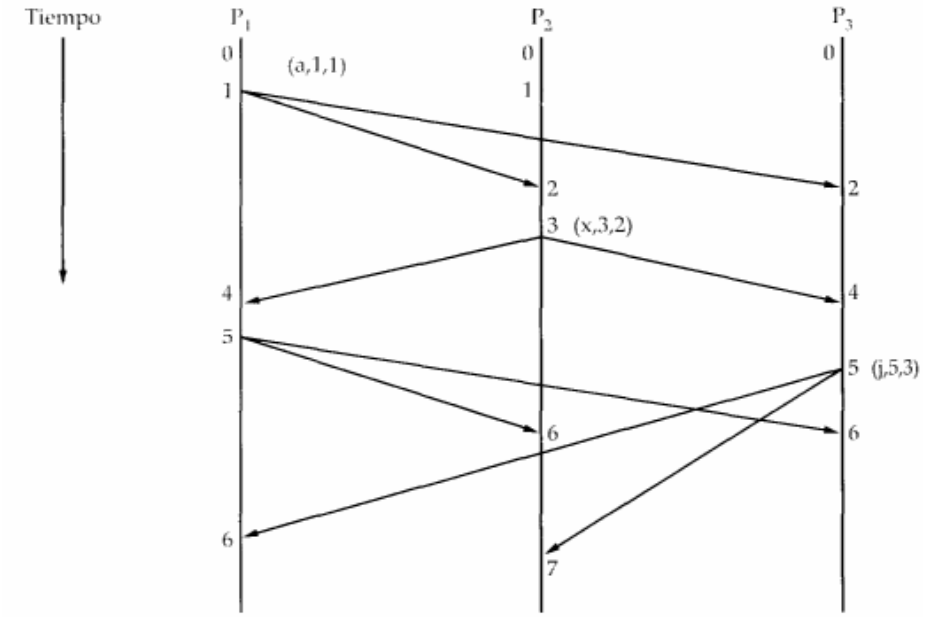


FIGURA 13.8 Ejemplo de funcionamiento del algoritmo de marcas de tiempo

gan de la misma ordenación de mensajes. En tal caso, existe un conjunto con ordenación parcial. Sin embargo, la preocupación principal es el uso de marcas de tiempo en los algoritmos distribuidos para la exclusión mutua y la detección de interbloqueo. En tales algoritmos, un proceso suele mandar un mensaje (con su marca de tiempo) a todos los demás procesos, empleándose las marcas de tiempo para determinar cómo se procesan los mensajes.

Cola Distribuida

Primera Versión

Uno de los primeros enfoques propuestos para mantener la exclusión mutua distribuida está basado en el concepto de cola distribuida [LAMP78]. El algoritmo se basa en los siguientes supuestos:

1. Un sistema distribuido consta de *N* nodos, numerados de forma única desde 1 hasta *N*. Cada nodo alberga un proceso que hace peticiones de acceso mutuamente exclusivo a recursos de parte de otros procesos; este proceso también sirve como un árbitro para resolver las peticiones entrantes que se solapan en el tiempo.
2. Los mensajes enviados desde un proceso a otro se reciben en el mismo orden en que fueron enviados.
3. Cada mensaje se entrega correctamente a su destino dentro de un tiempo finito.
4. La red está totalmente conectada, lo que significa que cada proceso puede enviar directamente mensajes a cualquier otro proceso sin necesidad de que un proceso intermedio pase el mensaje.

Los supuestos 2 y 3 pueden llevarse a cabo mediante el uso de un protocolo de transporte fiable, tal y como se discutió en la sección 12.1.

Por simplicidad, se describirá el algoritmo para el caso en que cada nodo controla solamente un recurso. La generalización para múltiples recursos es trivial.

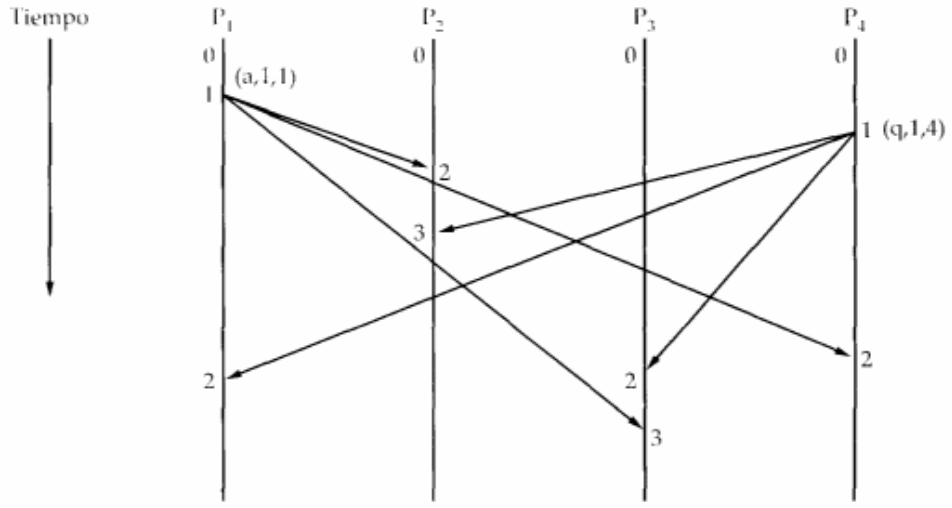


FIGURA 13.9 Otro ejemplo de funcionamiento del algoritmo de marcas de tiempo

El algoritmo es un intento de generalizar uno que funcionaría de forma sencilla en un sistema centralizado. Si un único proceso central gestionara el recurso, podría encolar las peticiones entrantes y servirlas de la forma "primera en entrar, primera en salir". Para llevar a cabo el mismo algoritmo en un sistema distribuido, todos los nodos deben tener una copia de la misma cola. Se pueden usar las marcas de tiempo para asegurar que todos los nodos se ponen de acuerdo en el orden en que se van a servir las peticiones. Surge una complicación: Como se tarda una cantidad finita de tiempo en que los mensajes atraviesen la red, existe el peligro de que dos nodos diferentes no estén de acuerdo en qué proceso está en cabeza en la cola. Considérese la figura 13.9. Existe un punto en que el mensaje *a* ha llegado a P_2 y el mensaje g ha llegado a P_3, pero el otro mensaje está de camino. Por tanto, hay un periodo en que P_1 y P_2 creen que el mensaje *a* es la cabeza de la cola, mientras P_3 y P_4 creen que es el mensaje *q* el que está en la cabeza de la cola. Esto podría guiar a una violación del requisito de exclusión mutua. Para impedir la violación, se impone la siguiente regla: Para que un proceso tome una decisión de asignación basada en su propia cola, tiene que haber recibido un mensaje de todos los demás nodos que garantice que no hay ningún mensaje de camino anterior al mensaje en cabeza de su propia cola.

En cada nodo se conserva una estructura de datos que guarda un registro de los mensajes recibidos más recientes de cada nodo (incluyendo el mensaje más reciente generado en este nodo). Lamport se refiere a esta estructura como una cola; en realidad, es un vector con una entrada para cada nodo [LAMP78]. En cualquier momento, la entrada *q[j]* del vector local contiene un mensaje de P_j. El vector se inicializa como sigue:

q[J] = (Liberación, 0,.j) $j = 1, ..., N$

En este algoritmo se usan tres clases de mensajes:

- (Petición, T_i , i): Petición de acceso a un recurso realizada por P_i.
- (Respuesta, T_j , j): P_j concede el recurso bajo su control.
- (Liberación, T_k , k): P_k libera un recurso que le fue asignado previamente.

El algoritmo es como sigue:

1. Cuando P_i requiere acceder a un recurso, emite una respuesta (Petición, T_i , i), con la marca de tiempo del valor actual del reloj local. Añade el mensaje a su propio vector en q[i] y lo envía a todos los demás procesos.
2. Cuando P_j recibe una (Petición, T_i, i), añade el mensaje a su propio vector en q[i] y transmite una (Respuesta, T_j, *j)* a todos los demás procesos. Esta acción es la que hace efectiva la regla antes descrita, que supone que no existen mensajes de Petición anteriores en camino en el momento de la decisión.
3. P, podrá acceder al recurso (entrar en su sección crítica) cuando se cumplan las dos condiciones siguientes:
   a) El mensaje de Petición del propio P, es el mensaje de Petición más antiguo en el vector q; como los mensajes están ordenados de forma consistente en todos los nodos, esta regla hace que un y sólo un proceso acceda al recurso en cualquier instante.
   b) Todos los demás mensajes del vector local son posteriores al mensaje de q[i]; esta regla garantiza que P, se ha enterado de todas las peticiones que preceden a su petición actual.

4. P_i libera un recurso emitiendo un mensaje (Liberación, T_i, i), que coloca en su propio vector y transmite a todos los demás procesos.
5. Cuando P_i recibe una (Liberación, T_j, *j),* reemplaza el contenido actual de *q[j]* con este mensaje.
6. Cuando P_i recibe una (Respuesta, T_j . j), reemplaza el contenido actual de *q[j]* con este mensaje.

Se demuestra fácilmente que este algoritmo hace cumplir la exclusión mutua, es equitativo, impide el interbloqueo y evita la inanición:

- *Exclusión mutua:* Las peticiones de entrada a la sección crítica se manejan de acuerdo al orden de los mensajes impuesto por el mecanismo de marcas de tiempo. Una vez que P_i decide entrar en su sección crítica, no pueden haber otros mensajes de Petición en el sistema que se transmitieran antes que el suyo, pues P_i habrá tenido que recibir entonces un mensaje de todos los demás nodos y estos mensajes datan de antes que su propio mensaje. Esto puede asegurarse por el mecanismo del mensaje de Respuesta; recuérdese que los mensajes entre dos nodos no pueden llegar desordenados.
- *Equidad:* Las peticiones se conceden con base estricta en la ordenación por marcas de tiempo. Por tanto, todos los procesos tienen igualdad de oportunidades.
- *Ausencia ele interbloqueo:* Como en todos los nodos se mantiene la ordenación por marcas de tiempo de manera consistente, no puede ocurrir un interbloqueo.
- *Ausencia de inanición:* Una vez que P_i haya terminado con su sección crítica, transmitirá un mensaje de Liberación. Este produce el efecto de eliminar los mensajes de Petición de P_i en todos los demás nodos, permitiendo que algún otro proceso entre en su sección crítica.

Como medida de la eficacia de este algoritmo, nótese que, para garantizar la exclusión mutua, se necesitan 3 *x (N - 1)* mensajes: *(N - 1)* mensajes de Petición, (N - 1) mensajes de Respuesta y (N - 1*)* mensajes de Liberación.

Segunda Versión

En [RICA81] se propone una modificación del algoritmo de Lamport. Se pretende optimizar el algoritmo original, eliminando los mensajes de Liberación. Siguen vigentes los mismos supuestos que antes, excepto que no es necesario que los mensajes enviados de un proceso a otro sean recibidos en el mismo orden en que se enviaron.

Como antes, cada nodo incluye un proceso que controla la asignación de recursos. Este proceso mantiene un vector *q y obra* de acuerdo a las reglas siguientes:

1. Cuando P_i requiere acceder a un recurso, emite una petición (Petición, T_i, i), marcada con el tiempo del valor actual de su reloj local. Añade este mensaje a su propio vector en *q[i]* y lo envía a todos los demás procesos.
2. Cuando P_j recibe una (Petición, T_i, , i), obra de acuerdo a las reglas siguientes:

a) Si P_j está actualmente en su sección crítica, posterga el envío de un mensaje de Respuesta (ver la regla 4 a continuación).

b) Si P_j no está esperando para entrar en su sección crítica (no ha emitido una Petición que esté aún pendiente), transmite una (Respuesta, T_j, *j)* a todos los demás procesos.

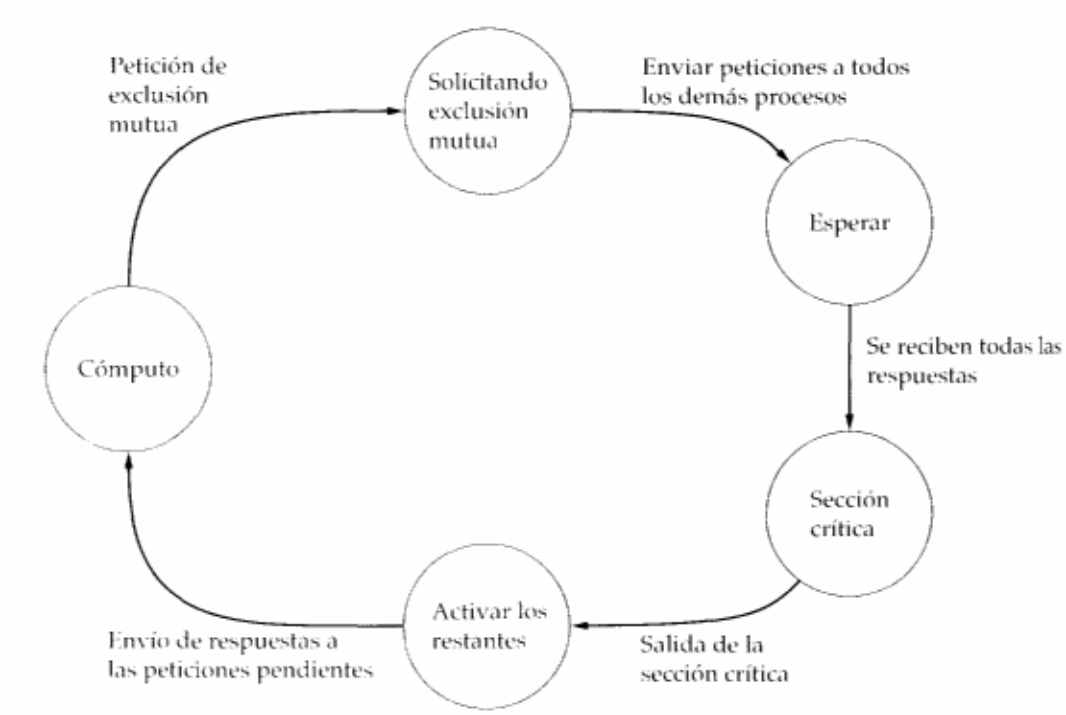


FIGURA 13.10 Diagrama de estados para el algoritmo de [RICA81]

c) Si P_j está esperando a entrar en su sección crítica y el mensaje entrante es posterior a la petición de P_j, añade este mensaje a su propio vector en q[i] y aplaza el envío de un mensaje de Respuesta.

d) Si P_j está esperando entrar en su sección crítica y el mensaje entrante es anterior a la petición de P_j añade este mensaje a su propio vector en q[i] y transmite una (Respuesta, T_j, j) a P_j.

3. P_i puede acceder a un recurso (entrar en su sección crítica) cuando haya recibido un mensaje de Respuesta de todos los demás procesos.
4. Cuando P_i abandona su sección crítica, libera el recurso enviando un mensaje de Respuesta para cada Petición pendiente.

En la figura 13. 10 se muestra el diagrama de transición de estados de cada proceso. En resumen, cuando un proceso desea entrar en su sección crítica, envía un mensaje de Petición con marca de tiempo a todos los demás procesos. Cuando reciba una Respuesta de todos los demás, podrá entrar en su sección crítica. Cuando un proceso reciba una Petición de otro proceso, deberá enviar en su momento la Respuesta correspondiente. Si un proceso no desea entrar en su sección crítica. envía una Respuesta de inmediato. Si quiere entrar en su sección crítica, compara la marca de tiempo de su Petición con la de la última Petición recibida y, si esta última es más reciente, aplazara, su Respuesta; en cualquier otro caso, se envía una Respuesta de inmediato.

Con este método hacen falta 2 (N - l) mensajes: *(N - 1)* mensajes de Petición para señalar la intención de P_i de entrar en su sección crítica y (N - 1*)* mensajes de Respuesta para conceder el acceso solicitado.

El uso de las marcas de tiempo en este algoritmo hace cumplir la exclusión mutua. También evita el interbloqueo. Para probar esto último, supóngase lo contrario: que es posible que, cuando no hay más mensajes en camino, se tenga una situación en la que cada proceso ha transmitido una Petición y no ha recibido la Respuesta necesaria. Esta situación no puede surgir porque la decisión de aplazar la Respuesta está basada en la relación de ordenación de las Peticiones. Habrá, por tanto, una Petición que disponga de la marca de tiempo más antigua y que recibirá todas las Respuestas necesarias.

También se impide la inanición, ya que las Peticiones están ordenadas. Como las Peticiones se sirven en ese orden, cada Petición se convertirá en algún momento en la más antigua y será entonces servida.

Método del Paso de Testigo

Un conjunto de investigadores han propuesto un enfoque bastante diferente para la exclusión mutua que consiste en pasar un testigo (token) entre los procesos participantes. El testigo es una entidad que, en un momento dado, es retenida por un proceso. El proceso que posee el testigo puede entrar a su sección crítica sin pedir permiso. Cuando el proceso abandona su sección crítica, pasa el testigo a otro proceso.

En este apartado, se examinará uno de los esquemas más eficaces. Fue propuesto primero en [SUZU82]; también apareció una propuesta equivalente en [RICA83]. En este algoritmo, se necesitan dos estructuras de datos. El testigo, que se pasa de proceso en proceso, es en realidad un vector testigo, cuyo k-ésimo elemento registra la marca de tiempo de la última vez que el testigo visitó al proceso P_k. Además, cada proceso mantiene un vector petición, cuyo j-ésimo elemento registra la marca de tiempo de la última Petición recibida de P_j.

El procedimiento es el siguiente. Al principio, se asigna el testigo de forma arbitraria a uno de los procesos poniendo testigo_presente como cierto para dicho proceso. Cuando un proceso desee usar su sección crítica, podrá hacerlo si en ese momento está en posesión del testigo; en otro caso, difunde un mensaje de petición con marca de tiempo a todos los demás procesos y espera hasta recibir el testigo. Cuando el proceso P_i abandone su sección crítica, debe transmitir el testigo a algún otro proceso. El siguiente proceso en recibir el testigo lo elige examinando el vector petición según el orden i + 1, i + 2, . . ., l , 2, . . ., i (1, en busca de la primera entrada petición[j] tal que la marca de tiempo de la última petición del testigo por parte de P_j sea mayor que el valor registrado en el testigo para la última posesión del mismo por parte de P_j: petición[j] > testigo[j].

En la figura 13.11 se puede apreciar el algoritmo, que está dividido en dos partes. La primera parte se refiere al uso de la sección crítica y consta de una sección de entrada, seguida por la sección crítica y, después, una sección de salida. La segunda parte se ocupa de las acciones a ejecutar cuando se recibe una petición. La variable reloj es el contador local empleado para las marcas de tiempo. La operación Esperar(acceso, testigo) origina que el proceso espere hasta recibir un mensaje de tipo "acceso". El mensaje contiene un valor del testigo, que se colocará después en el vector variable testigo.

El algoritmo exige una de estas dos cosas:

- *N* mensajes *(N - 1* para difundir la petición y 1 para transferir el testigo) cuando el proceso demandante no posee el testigo.
- Ningún mensaje si el proceso ya posee el testigo.

13.4 INTERBLOQUEO DISTRIBUIDO

En el capítulo 5 se definió el interbloqueo como el bloqueo permanente de un conjunto de procesos que compiten por los recursos del sistema o se comunican uno con otro. Esta definición es válida tanto en un sistema sencillo como para un sistema distribuido. Al igual que la exclusión mutua, el interbloqueo introduce problemas más complejos en un sistema distribuido, en comparación con un sistema de memoria compartida. El manejo del interbloqueo es complicado en un sistema distribuido porque no hay un nodo con el conocimiento exacto del estado actual del sistema global y porque cada transferencia de mensajes entre los procesos conlleva un retardo impredecible.

En la literatura se ha prestado atención a dos tipos de interbloqueo distribuido: los que surgen en la asignación de recursos y los que aparecen con la comunicación de mensajes. En los interbloqueos de recursos, los procesos intentan acceder a los recursos, como pueden ser los objetos de datos de una base de datos o los recursos de E/S de un servidor; el interbloqueo se produce si cada proceso de un conjunto de procesos solicita un recurso empleado por otro proceso del conjunto. En los interbloqueos de comunicaciones, los mensajes son los recursos por los que esperan los procesos; el interbloqueo se produce si cada proceso de un conjunto está esperando un mensaje de otro proceso del conjunto y ningún proceso del conjunto enviará nunca el mensaje.

Interbloqueo en la Asignación de Recursos

Recuérdese que en el capítulo 5 se dice que existe interbloqueo en la asignación de un recurso sólo si se cumplen todas las condiciones siguientes:

I . Exclusión mutua: Sólo un proceso puede usar un recurso en un instante.
2. *Retención y* espera: Un proceso puede utilizar unos recursos asignados mientras que espera la asignación de otros.
3. *No expulsión: No* puede quitarse a la fuerza ningún recurso a un proceso que lo está utilizando.
4. *Círculo vicioso de espera:* Existe una cadena cerrada de procesos tal que cada proceso utiliza, por lo menos, un recurso necesario para el proceso siguiente de la cadena.

La intención de un algoritmo que trate el interbloqueo es prevenir la formación de un círculo vicioso de espera o detectar su aparición real o potencial. En un sistema distribuido, los recursos están distribuidos por varios nodos y su acceso está regulado por procesos de control que no disponen de un conocimiento completo y actualizado del estado global del sistema, así que deben tomar decisiones basándose en la información local. Por tanto, se necesitan nuevos algoritmos de interbloqueo.

Un ejemplo de la dificultad afrontada en la gestión distribuida del interbloqueo es el fenómeno del *interbloqueo* fantasma En la figura 13.12 se ilustra un ejemplo de interbloqueo fantasma La notación $P_1 \rightarrow P_2 \rightarrow P_3$ quiere decir que P_1 está detenido esperando un recurso empleado por P_2 y P_2 está detenido esperando un recurso empleado por P_3. Supóngase que, al comienzo del ejemplo, P_3 posee el recurso R_a *y* P_1 posee el recurso R_b. Supóngase entonces que P_3 emite primero un mensaje liberando R_a y después un mensaje solicitando R_b. Si el primer mensaje llega 11 1111 proceso detector de ciclos antes que el segundo, se producirá la

```
if not testigo-presente then begin
    reloj := reloj + l;                                  [Sección de entrada]
    difundir (petición, reloj, i);
    esperar (acceso, testigo):
    testigo_presente := Cierto
                              end;
endif;
posesión_testigo := Cierto;
    <sección crítica>
testigo (i) := reloj:                                    [Sección de salida]
posesión testigo := Falso;
for .j := i + 1 to n, 1 to i - I do
    if (petición (j) > testigo (j)) ^ testigo_presente
        then begin
        testigo_presente := Falso;
        enviar (j, acceso, testigo) end
    endif;
```

(a) Primera Parte

```
when recibido (petición, t, j) do
    petición (j) := max (petición (j), t);
    if testigo_presente ^ not posesión_testigo then
        <texto de la sección de salida>
    endif
enddo;
```

(b) Segunda Parte

Notación:

enviar (j. acceso, testigo)	enviar un mensaje de tipo acceso, junto con el testigo, al proceso j
difundir (petición, reloj, i)	enviar un mensaje de tipo petición desde el proceso i, con tina marca de tiempo reloj, a todos los demás procesos
recibido (petición, l. ,j)	recibir un mensaje del Proceso J, de tipo petición, con marca de tiempo t

FIGURA 13.11 Algoritmo de paso de testigo (para el proceso P_i)

secuencia de la figura 13 .12a, que refleja debidamente las exigencias de recursos. Sin embargo segundo mensaje llega antes que el primero, se registra un interbloqueo (figura 13.12b). Esta es una falsa detección y no un interbloqueo real, originado por la falta de un estado global, como podría existir el un sistema centralizado.

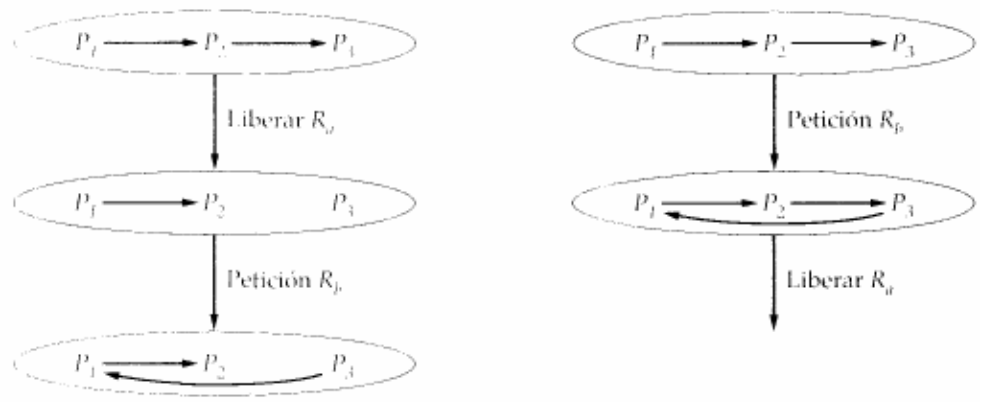
Prevención del Interbloqueo

Dos de las técnicas de prevención del interbloqueo estudiadas en el capítulo 5 pueden emplearse en un sistema distribuido.

1. La condición del círculo vicioso de espera puede prevenirse definiendo una ordenación lineal de los tipos de recursos. Si a un proceso le han sido concedidos recursos de tipo R, puede solicitar a continuación sólo aquellos recursos de los tipos que siguen a R en la ordenación. La principal desventaja de este método es que puede que los recursos no se pidan en el orden en que son usados; por tanto, puede que los recursos sean retenidos más tiempo del necesario.
2. La condición de retención y espera puede prevenirse pidiendo que los procesos soliciten todos los recursos necesarios de una vez, bloqueando el proceso hasta que todas las peticiones puedan concederse simultáneamente. Este método no es eficiente por dos razones. En primer lugar, un proceso puede verse interrumpido durante mucho tiempo esperando a que todas sus peticiones de recursos se concedan, cuando, de hecho, podría haber continuado con sólo algunos recursos. En segundo lugar, los recursos asignados a un proceso pueden permanecer sin usar por un periodo considerable, tiempo durante el cual sor determinados a otros procesos.

Ambos Métodos requieren que los procesos determinen por adelantado sus necesidades de recursos. Esto no es siempre posible; un ejemplo es una aplicación de base de datos en que se pueden añadir dinámicamente nuevos elementos. Como ejemplo de un enfoque que no requiera este conocimiento previo, considérense los dos algoritmos propuestos en [ROSE78]. Estos algoritmos fueron desarrollados en el campo de trabajo de las bases de datos, así que se hablará de transacciones en vez de procesos.

Los Métodos propuestos hacen uso de las enarcas de tiempo. Cada transacción lleva durante su vida la marca de tiempo de su creación, lo que constituye una ordenación estricta de las transacciones. Si un recurso R que está siendo utilizado por la transacción T_1 es solicitado por otra transacción T_2, el conflicto se resuelve por comparación de sus marcas de tiempo. Esta comparación se usa para prevenir la formación de círculos viciosos. Rosenthal



a) Llegada de la liberación antes que la petición b) Llegada de la petición antes que la liberación

FIGURA 13.12 Interbloqueo fantasma

y otros propusieron dos variaciones de este método básico, conocidas como el método "esperar-morir" (wait-die) y el método "herir-esperar" (wound-wait) [ROSE78].

Supóngase que T_1 está utilizando *R* y *que* T_2 emite una petición. Para el **método esperar-morir,** la figura 13.13a ¡nuestra el algoritmo empleado por el asignado¡ de recursos en el nodo de *R.* Las marcas de tiempo de las dos transacciones se denotan por $e(T_1)$ y $e(T_2)$. Si T_2, es más antigua, se queda bloqueada hasta que T_1 libere R, bien emitiendo de forma activa una liberación o bien "matándose" cuando solicite otro recurso. Si T_2 es más joven, se reinicia T_2, pero con la misma marca de tiempo que antes.

Consecuentemente, en caso de conflicto, la transacción más antigua lleva la prioridad. Una transacción matada revivirá con su marca de tiempo original, de forma que envejecerá y, por tanto, obtendrá un incremento de prioridad. Ningún nodo necesita saber el estado de asignación de todos los recursos. Todo lo que se necesita son las marcas de tiempo de las transacciones que solicitan sus recursos.

El **método herir-esperar** concederá de inmediato las peticiones de transacciones más antiguas, matando a la transacción más joven que está utilizando el recurso requerido, como se ¡nuestra en la figura 1 3.13b. A diferencia del método esperar-morir, una transacción nunca tiene que esperar por un recurso que está siendo utilizado por una transacción más joven.

```
if e(T2) < e(T1 )then detener T2 ('esperar')
        else eliminar T2 ('morir' )
endif
```

(a) Método esperar-morir

```
if e(T2) < e(T I )  then eliminar T1 ('herir' )
        else detener T2 ('esperar' )
endif
```

(b) Método herir-esperar

FIGURA 13.13 Métodos de prevención del interbloqueo

Predicción del Interbloqueo

La predicción del interbloqueo es una técnica en la que se toma la decisión dinámicamente sobre si una petición dada de asignación de un recurso podría conducir, si se concediese, a un interbloqueo. [SING94] indica que la predicción distribuida del interbloqueo es poco práctica por las siguientes razones:

1. Cada nodo guarda constancia del estado global del sistema; esto requiere un almacenamiento considerable y una **s**obrecarga de comunicaciones.
2. El proceso de inspección de un estado global seguro debe hacerse con exclusión mutua. En otro caso, dos nodos podrían tener en cuenta peticiones de un recurso por parte de procesos diferentes y llegar concurrentemente a la conclusión de que es seguro conceder la petición, cuando en realidad, si se conceden ambas peticiones, se producirá interbloqueo.
3. Inspeccionar los estados seguros implica un proceso extra considerable en un sistema distribuido con un gran número de procesos y recursos.

Detección del Interbloqueo

Con detección del interbloqueo, se permite a los procesos obtener los recursos libres cuando deseen y después se determina la existencia de un interbloqueo Si se detecta un interbloqueo, se selecciona uno de los procesos integrantes y se le pide que libere los recursos necesarios para deshacer el interbloqueo.

La dificultad de la detección distribuida de interbloqueos es que cada nodo sólo conoce seis propios recursos, mientras que el interbloqueo puede involucrar a recursos distribuidos. Son posibles varios enfoques, dependiendo de si el control del sistema es centralizado, jerárquico o distribuido (tabla 13. 1).

Con un **control centralizado,** un nodo es el responsable de la detección del interbloqueo. Todos los mensajes de Petición y Liberación se envían al proceso central así como al proceso que controla el recurso específico. Como el proceso central tiene una representación completa está en posición de detectar interbloqueos Este método necesita muchos mensajes y es vulnerable a fallos en el nodo central. Además. pueden detectarse interbloqueos fantasmas.

Con un **control Jerárquico,** los iodos están organizados en una estructura en árbol, con un nodo actuando cono la raíz del árbol. En cada nodo que no sea una hoja, se reúne información sobre la asignación de recursos de todos los nodos dependientes. Esto permite detectar interbloqueos en niveles inferiores al nodo raíz. Explícitamente, un interbloqueo que compromete a un conjunto de recursos será detectado por el nodo que sea el ascendiente común de todos nodos cuyos recursos están entre los objetos en conflicto.

Con un **control distribuido,** todos los procesos cooperan en la detección de interbloqueos En general, esto significa que deberá intercambiarse una información considerable con Marcas de tiempo Por tanto, el coste es significativo. [RAYN88] cita una serie de métodos todos basados en el control distribuido y [DATT90] ofrece un examen detallado de un método.

Ahora se dará un ejemplo de algoritmo de detección distribuida del interbloqueo, expuesto en [JOHN91]. El algoritmo trata Con un sistema de base de dalos distribuida en el que cada nodo guarda una parte de la base de datos y se pueden iniciar transacciones desde cualquier nodo**.** Una transacción puede tener como Máximo una petición pendiente de recursos. Si una transacción necesita mas de un objeto de datos, el segundo objeto puede pedirse sólo después de que el primero haya sido concedido.

Asociado a cala objeto de datos i de un nodo hay dos parámetros: un identificador único D_i y la variable Bloqueada_por (D_i). Esta variable tiene un valor *nil* si el objeto de datos no ha sido bloqueado por ninguna transacción: en otro caso, su valor será el identificador de la transacción que lo bloqueó.

Asociado a cada transacción j de un nodo hay cuatro parámetros:

- Un identificador único T*j*.
- La Variable Retenida_por(T_j), que se pone a *nil* si 1a transacción T_j está ejecutándose o en estado de preparada para ejecución. En otro caso, su valor será el de la transacción que esta empleando el objeto de datos requerido por la transacción T_j.
- La variable Esperando(T_j), que tiene un valor *nil* si la transacción T_j no está esperando a otra transacción. En otro caso, su valor será el identificador de la transacción que está en cabeza de la lista ordenada de transacciones bloqueadas.

TABLA 13.1 Estrategias de detección distribuida del interbloqueo

Algoritmos Centralizados		Algoritmos Distribuidos		Algoritmos Jerárquicos	
Ventajas	Inconvenientes	Ventajas	Inconvenientes	Ventajas	Inconvenientes
• Los algoritmos son conceptualmente simples y fáciles de implementar. • El nodo central dispone de información completa y puede resolver óptimamente los interbloqueos.	• Sobrecarga considerable de comunicaciones; cada nodo debe enviar información de estado al nodo central. • Vulnerable a fallos del nodo central.	• No vulnerable a fallos en un solo punto • Ningún nodo es abrumado con la actividad de detección de interbloqueo	• La resolución del interbloqueo es engorrosa porque varios nodos pueden detectar el mismo interbloqueo y pueden no darse cuenta de otros nodos involucrados en el interbloqueo. • Los algoritmos son difíciles de diseñar debido a las consideraciones de tiempos.	• No vulnerable a fallos en un solo punto. • La actividad de resolución del interbloqueo se ve limitada si la mayor parte de los posibles interbloqueos están relativamente localizados.	• Puede ser difícil configurar el sistema de forma que la mayoría de los posibles interbloqueos estén localizados; en otro caso, puede haber realmente más sobrecarga que en el enfoque distribuido.

- Una cola C_Peticiones(T_j), que contiene todas las peticiones pendientes de los objetos de datos que están siendo utilizados por T_j. Cada elemento de la cola es de la forma (T_k, D_k), donde T_k es la petición demandante y D_k, es el objeto de datos utilizado por T_j.

Por ejemplo, supóngase que la transacción T_j está esperando al objeto de datos empleado por T_i, que, a su vez, está esperando al objeto de datos empleado por T_j. Los parámetros pertinentes tendrán los valores siguientes:

Transacción	Esperando_para	Retenida_por	C- Peticiones
T0	nil	nil	T1
T1	T0	T0	T2
T2	T0	T1	nil

Este ejemplo realza las diferencias entre Esperando_para(T_i) y Retenida _por(T_i). Ningún proceso puede seguir hasta que T,, libere el objeto de datos reclamado por T,, momento en que podrá ejecutar v liberar el objeto de datos requerido por T,

La figura 13.14 muestra el algoritmo empleado para detección de¡ interbloqueo. Cuando una transacción hace una petición de bloqueo de un objeto de datos, un proceso servidor asociado a dicho objeto concede o deniega la petición. Si no se concede la petición el proceso servidor devuelve la identidad de la transacción que está utilizando el objeto de datos.

Cuando la transacción demandante reciba una respuesta de concesión, bloqueará el objeto. En otro caso, la transacción demandante actualiza su variable Retenida por con la identidad de la transacción que está utilizando el objeto de datos. Añadirá su identidad a la C_Peticiones de la transacción que está utilizando el objeto. Actualizará su variable Esperando con la identidad de la transacción que está utilizando el objeto (si la transacción no está en espera) o con el valor de la variable Esperando de esta transacción. De esta forma, la variable Esperando se rellena con el valor de la transacción que en definitiva está bloqueando la ejecución. Para finalizar, la transacción demandante emite un mensaje de actualización a todas las transacciones de su C_Peticiones para modificar todas las variables Esperando que se vean afectadas por este cambio.

Cuando una transacción reciba un mensaje de actualización, pondrá al día si la variable Esperando para reflejar el hecho de que la transacción por la que ha estado esperando últimamente está ahora bloqueada por otra transacción más. Después, realiza la labor real de detección del interbloqueo, para ver si ahora está esperando a alguno de los procesos que están esperándole a su vez. Si es así, la transacción envía un mensaje de desatasco a la transacción que esté reteniendo al objeto solicitado y asigna cada objeto que ésta posee al primer demandante de su C Peticiones, encolando el resto de demandantes en la nueva transacción.

Un Templo de funcionamiento del algoritmo se muestra en la figura 13.15. Cuando T_i hace una petición de un Objeto de datos empleado por T,, se origina un ciclo. T,, emite un mensaje de actualización que se propaga desde T_i a T_j, pasando por T_k. En este punto, T_i; descubre que la intersección de sus variables Esperando y C_Peticiones no es vacía. T; envía nil mensaje de desatasco a la T_i de manera que se liquide a T; de C_Peticiones(T_i) y libera el objeto que retiene, activando T_i y T_j.

```
{Objeto de datos Dj que recibe una petición_bloqueo(Ti)}
begin
    if Bloqueada_por(Dj) = nil then enviar concesión
    else
        begin
            enviar no concesión a Ti;
            enviar Bloqueada_por(Dj) a Ti
        end
end.

{La transacción Ti hace una petición de bloqueo del objeto
de datos Dj}
begin
    enviar petición_bloqueo(Ti) a Dj;
    esperar concesión/no concesión;
    if concesión then
        begin
            Bloqueada_por(Dj) := Ti;
            Retenida_por(Ti) := ϕ;
        end
    else {se supone que Dj está siendo utilizado por la
          transacción Tj}
        begin
            Retenida_por(Tj) := Tj;
            Poner_en_cola(Ti, C_Peticiones(Tj));
            if Esperando_para(Tj) = nil then Esperando_para(Ti) := Tj
            else Esperando_para(Ti) := Esperando_para(Tj);
            actualizar(Esperando_para(Ti), C_Peticiones(Tj))
        end
end.
{Transacción Tj que recibe un mensaje de actualización}
begin
    if Esperando_para(Tj) ≠ Esperando_para(Ti)
        then Esperando_para(Tj) := Esperando_para(Ti);
    if Esperando(Ti) ∩ C_Peticiones(Tj) = nil
        then actualizar(Esperando_para(Ti), C_Peticiones(Tj))
    else
        begin
            DECLARAR INTERBLOQUEO;
            {inicio de la resolución del interbloqueo como sigue}
            {Tj es elegida como la transacción a abandonar}
            {Tj libera todos los objetos de datos que retiene}
            enviar desatasco(Tj, Retenida_por(Tj));
            asignar cada objeto de datos Dj retenido por Tj al
              primer demandante Tk de la C_Peticiones(Tj);
            for cada transacción Tn de C_Peticiones(Tj)
              solicitando un objeto de datos Dj retenido por Tj do
                Poner_en_cola(Tn, C_Peticiones(Tk));
        end
end.

{Transacción Tk que recibe un mensaje de limpiar(Tj, Tk)}
begin
    eliminar de C_Peticiones(Tk) a la tupla que tiene
a Tj como la transacción demandante
end.
```

FIGURA 13.14 Un algoritmo de detección distribuida del interbloqueo

Interbloqueo en la Comunicación de Mensajes

Espera Mutua

Se produce interbloqueo en la comunicación de mensajes cuando cada uno de los procesos de un grupo de procesos está esperando un mensaje de otro miembro del grupo y no hay mensajes en camino.

Para analizar esta situación con más detalle, se definirá el conjunto de dependencia (DS) de un proceso. Para un proceso P, que está detenido esperando un mensaje, DS(P_i) consta de todos los procesos de quienes P_i espera un mensaje. Normalmente, P_i puede continuar si llega cualquier mensaje de los esperados. Una formulación alternativa es que P_i puede continuar sólo después de que lleguen todos los mensajes esperados. La primera situación es más común y es la que aquí se considera.

Con la definición anterior, un interbloqueo en un conjunto S de procesos puede definirse como sigue:

1 . Todos los procesos de S están detenidos en espera de mensajes.
2. S contiene al conjunto de dependencia de todos los procesos de S.
3. Ningún mensaje está de camino entre los miembros de S.

Cualquier proceso de S está interbloqueado porque no puede recibir nunca un mensaje que lo libere.

En términos gráficos, hay una diferencia entre el interbloqueo de mensajes y el interbloqueo de recursos. En el interbloqueo de recursos, se produce interbloqueo cuando hay un bucle cerrado o ciclo en el grafo que representa las dependencias de los procesos. En el caso de los recursos, un proceso depende de otro si el último emplea un recurso que el primero necesita. En el interbloqueo de mensajes, 1a condición de interbloqueo es que todos los sucesores de cualquier miembro de S estén asimismo en S; es decir, el grafo de S es un nudo.

La figura 13.16 ilustra este apartado. En la figura 31.16a, P_1 está esperando un mensaje de P_2 o de P_5; P_5 no está esperando ningún mensaje así que puede enviar un mensaje a P_1 que, por tanto, es liberado. Como resultado, los enlaces (P_1, P_5) y (P_1, P_2) se eliminan. En la figura 13.16b se añade una dependencia: P_5 está esperando un mensaje de P_2. El grafo es ahora un nudo y existe interbloqueo

Como con los recursos, el interbloqueo de mensajes puede atacarse mediante prevención o detección. [RAY88] ofrece varios ejemplos.

No Disponibilidad de Buffers de Mensajes

Otra tornea en que puede producirse interbloqueo en un sistema de paso de mensajes es por la asignación de buffers para el almacenamiento de los mensajes en camino. Este tipo de interbloqueo es muy conocido en las redes de datos de conmutación de paquetes I [STAL94a]. Se examinará primero este problema en el contexto de las redes de datos, para posteriormente observarlo desde el punto de vista de un sistema operativo distribuido.

La forma más sencilla de interbloqueo en una red de datos es el interbloqueo directo por almacenamiento y reenvío (store-and-forward) y puede suceder si un nodo de conmutación ele paquetes utiliza una reserva común de buffers de donde son asignados por solicitud a los paquetes. La figura 13.17a muestra una situación en que todo el espacio para bufferes del

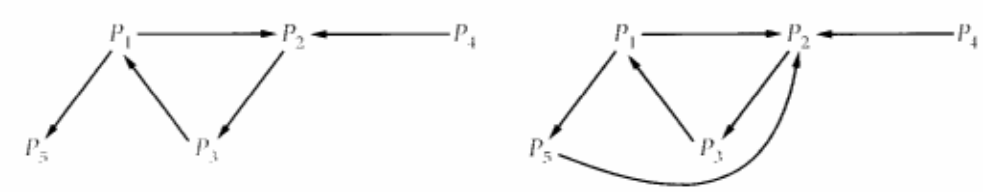
Transación	Esperando	Retenida_por	C_Peticiones
T_0	nil	nil	T_1
T_1	T_0	T_0	T_2
T_2	T_0	T_1	T_3
T_3	T_0	T_2	T_4, T_6
T_4	T_0	T_3	T_5
T_5	T_0	T_4	nil
T_6	T_0	T_3	nil

(a) Estado del sistema antes de la petición

Transación	Esperando	Retenida_por	C_Peticiones
T_0	T_0	T_3	T_1
T_1	T_0	T_0	T_2
T_2	T_0	T_1	T_3
T_3	T_0	T_2	T_4, T_6, T_0
T_4	T_0	T_3	T_5
T_5	T_0	T_4	nil
T_6	T_0	T_3	nil

(b) Estado del sistema después de que T_0 haga una petición a T_3

FIGURA 13.15 Ejemplo del algoritmo de detección distribuida del interbloqueo de la figura 13.14



(a) Sin interbloqueo **(b) interbloqueo**

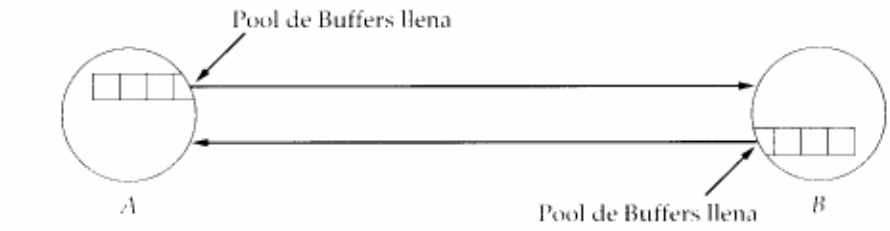
FIGURA 13.16 Interbioqueo en la comunicación de mensajes

nodo A está ocupado por paquetes destinados a B. Lo contrario es cierto en B. Ningún nodo puede aceptar más paquetes porque sus buffers están llenos. Por tanto, ningún nodo puede transmitir ni recibir por ningún enlace. El interbloqueo directo por almacenamiento y reenvío puede impedirse si no se permite que todos los buffers acaben dedicados a un único enlace. Empleando buffers separados de tamaño fijo, uno por cada enlace, se logrará este impedimento. Incluso si se usa una reserva común de buffers se puede evitar el interbloqueo si no se permite que enlaces únicos consigan todo el espacio de buffers.

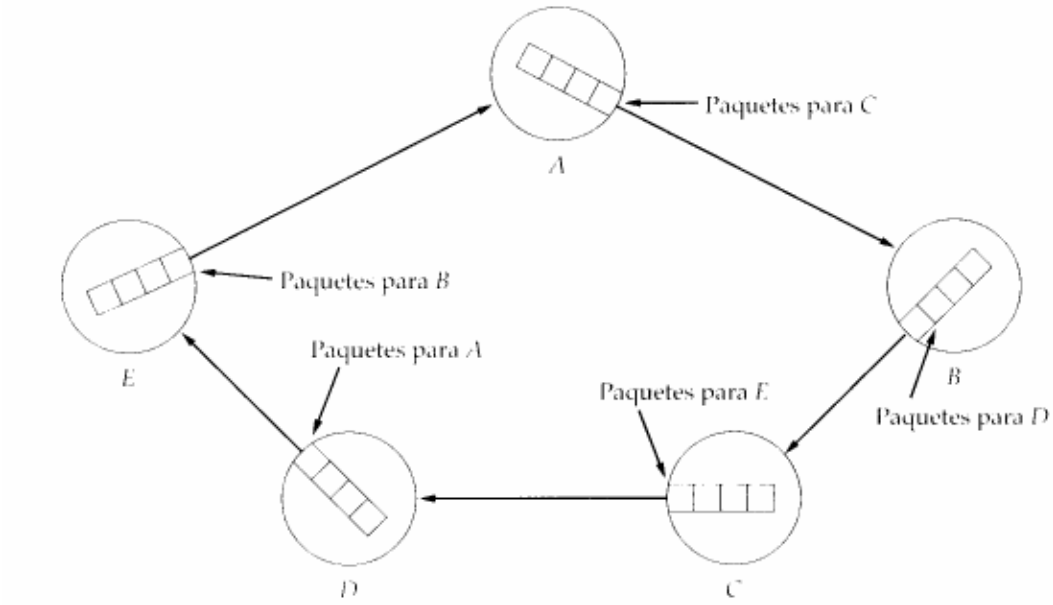
Una forma más sutil de interbloqueo el *interbloqueo indirecto por almacenamiento almacenamiento y reenvio,*, se ilustra en la figura 13.17b. Para cada nodo, la cola dirigida al nodo adyacente en una dirección está llena de paquetes destinados al nodo más allá del siguiente. Una forma simple de impedir este tipo de interbloqueo es emplear una reserva estructurada de buffers (figura 13.18). LoS buffers están organizados jerárquicamente. La reserva de memoria del nivel 0 esta libre; cualquier paquete entrante puede almacenarse allí. Desde el nivel 1 al nivel *N* (donde *N* es el número máximo de saltos por cualquier- camino de la red), los buffers se reservan de la forma siguiente: Los buffers del nivel k se reservan para los paquetes que han viajado al menos *k* saltos hasta aquí. De este modo, en condiciones de mucha carga, los buffers se rellenan progresivamente desde el nivel 0 hasta el nivel N. Si todos los buffers hasta el nivel *k* están llenos. los paquetes que lleguen habiendo cubierto k o menos saltos son descartados. Se puede demostrar [GOPA85] que esta estrategia elimina el interbloqueo por almacenamiento y reenvío. tanto directo como indirecto.

El problema de interbloqueo recién descrito se aplicaría en el contexto de una arquitectura de comunicaciones normalmente en el nivel 3 de OSI (nivel de red). El mismo tipo de problema puede surgir en un sistema distribuido que utilice paso de mensajes para la comunicación entre procesos. Explícitamente, si la operación *enviar* es no bloqueante, se necesita un buffer para albergar los mensajes salientes. Se puede pensar que el buffer empleado para guardar los mensajes a enviar desde el proceso X al proceso Y sea un canal de comunicaciones entre X e Y. Si este canal tiene una capacidad limitada (tamaño limitado del buffer), es posible que la operación *enviar* motive la suspensión del proceso. Es decir, si el buffer es de tamaño *n y* actualmente hay n mensajes en camino (no recibidos aún por el proceso de destino) la ejecución de un *enviar* adicional bloqueará al proceso emisor hasta que un *recibir* haya hecho espacio en el buffer.

La figura 13.19 ilustra cómo el empleo de canales limitados puede conducir a interbloqueo. En la figura se muestran dos canales cada uno con una capacidad de cuatro mensajes uno desde el proceso X al proceso Y y otro desde Y hasta X. Si hay cuatro mensajes



(a) Interbloqueo directo de almacenamiento y reenvío



(b) Interbloqueo indirecto por almacenamiento y reenvío

FIGURA 13.17 Interbloqueo por almacenamiento y reenvío

exactamente en camino en cada uno de los canales y si tanto X cono Y intentan una transmisión más antes de ejecutar un Recibir, ambos quedan suspendidos y aparece un interbloqueo.

Ahora es posible establecer un límite superior para el número de mensajes que estarán de camino en un instante entre cada par de procesos del sistema, por lo que la estrategia obvia de prevención sería asignar a estos canales tantos huecos de buffers como hagan falta. Esta estrategia puede ser sumamente dispendiosa y, por supuesto, necesita un conocimiento previo. Si no se pueden saber los requisitos por adelantado o si la asignación basada en un límite superior se considera demasiado dispendiosa, hace falta alguna técnica de estimación para optimizar la asignación. Puede demostrarse que este problema es irresoluble generalmente; algunas estrategias heurísticas para enfrentarse con esta situación se sugieren en [BARB90].

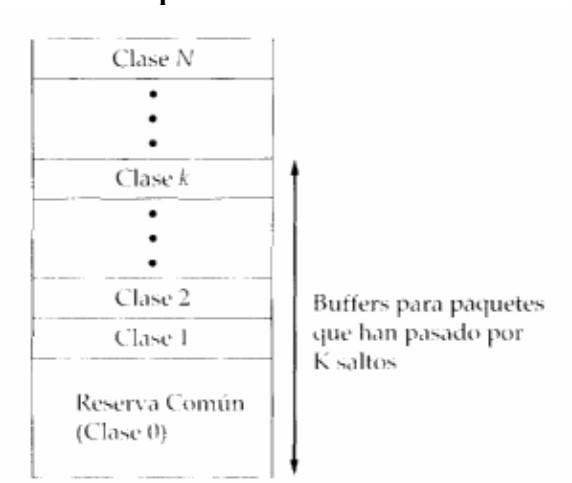


FIGURA 13.18 Pool estructurado de buffers para la prevención del interbloqueo

13.5 RESUMEN

Un sistema operativo distribuido puede ofrecer migración de procesos, es decir, la transferencia de una parte del estado de un proceso de una máquina a otra para que el proceso se ejecute en la máquina de destino. La migración de procesos puede utilizarse para equilibrar la carga, mejorar el rendimiento al disminuir la actividad de comunicación, aumentar la disponibilidad o permitir a los procesos acceder a servicios especializados remotos.

En un sistema distribuido, suele ser importante construir una información del estado global, para resolver la contienda por los recursos y coordinar los procesos. Debido al retardo tan variable e impredecible de la transmisión de mensajes, se debe tener cuidado de asegurar que los diferentes procesos se ponen de acuerdo en el orden en que se producen los sucesos.

La gestión de procesos de un sistema distribuido incluye servicios para hacer cumplir la exclusión mutua y para tomar acciones de tratamiento del interbloqueo. En ambos casos, los problemas son más complejos que en un sistema sencillo.

13.6 LECTURAS RECOMENDADAS

Recientemente se han publicado varias recopilaciones útiles. [CASA94] contiene 31 artículos que cubren un amplio rango de cuestiones teóricas y prácticas sobre los sistemas operativos distribuidos, sistemas ele archivos y bases de datos. [BRAZ94] incluye ejemplos ele sistemas operativos distribuidos. así corno artículos que versan sobre redes. [SRIM92] contiene segundas ediciones de algunos de los artículos más importantes sobre exclusión mutua distribuida.

[ESKI90] y [SMIT 88] constituyen útiles estudios sobre los mecanismos de migración de procesos. [ARTS89a] es una edición especial dedicada a los mecanismos y políticas de migración de procesos.

Se pueden encontrar algoritmos de gestión distribuida de procesos (exclusión mutua e interbloqueo) en [SING94a] [RAYN88] y [MAEK87] Un tratamiento más formal se puede hallar en [RAYN90]

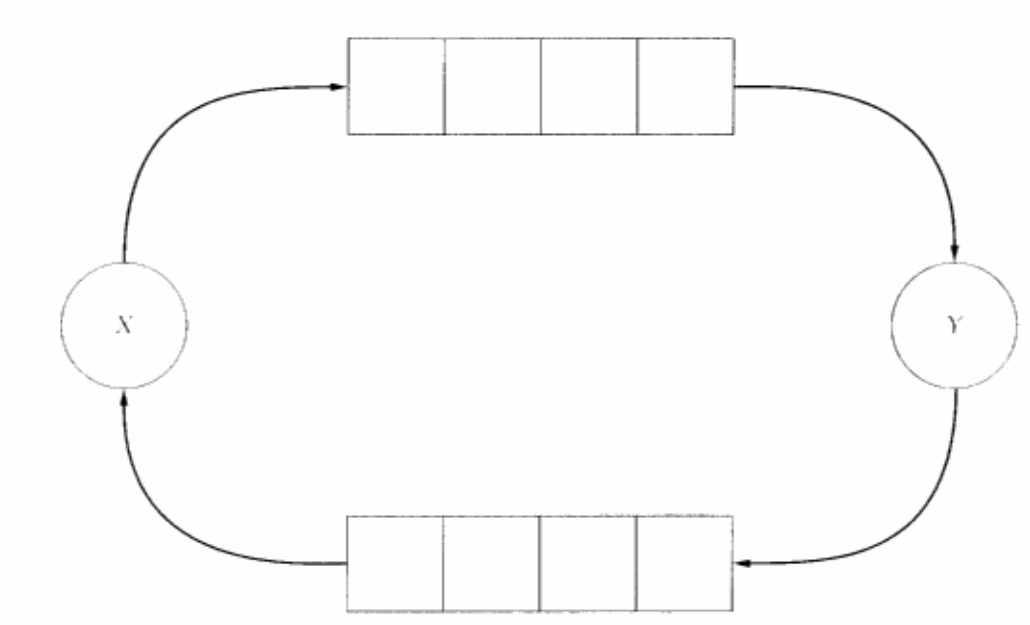


FIGURA 13.19 Interbloqueo de comunicaciones en un sistema distribuido

ARTS89a ARTSY, Y., ed. "Special Issue on Process Migration". Newsletter of the IDEE Computer iety tec hnical committee can Oper ating .S1 stems invierno de 1989.

BRAZ94 BRAZIER, F. y JOHANSEN, D. Distributed Open Systems IEEE Computer Society Press, Los Alamitos, CA, 1994.

CASA94 CASAVANT, T. y SINGHAL, M. Distributed Conputing Systems IEEE Computer Sociely Press, Los Alamitos, CA, 1994.

ESK190 ESKICIOGLU, M. "Design Issues of Process Migration Facilities in Distributed Systems New sletter of the IEEE Computer Soc iety Tec hnic al Committee can Oper ating Systems carel Applic ations Environments velarlo de 1990.

MAFK87 MAEKAWA, M., OLDEHOEFT A. y OLDEHOEFT, R. Operating Systems Advanced Conc epts. Benjamín Cummings, Menlo Park, CA, 1987. ,

RAYN88 RAYNAL, M. Distributed Algor ithms and protoc ols Wiley, Nueva York, 1988.

RAYN90 RAYNAL, M.. y HELARY, .I. Sync hronization and Control n/ Distributed Systems and Programs Wiley, Nueva York, 1991.

SING94a SINGHAL, M. y SHIVARATRI, N. Advanced Concepts in Operating Systems McGraw-Hill. Nueva York, 1994.

SMIT88 SMITH, .l. "A Survey of Process Migration Mechanisms". Operating Systems Review, julio de 1988.

SRIM92 SRIMANI, P. y DAS, S. Distributed Mutual Exclusion Algorithms IDEE Computer Society Press, Los Alamitos, CA, 1994.

13.7

PROBLEMAS

13.1 Comprobar la corrección del algoritmo de instantáneas distribuidas suponiendo lo siguiente

a) El algoritmo no registra un estado global inconsistente. Indicación: Demostración por reducción al absurdo.

b) El algoritmo registra correctamente el estado de cada canal. Es decir, para cada canal que va desde un proceso r hasta un proceso p, el estado del canal <r,p> es, una secuencia de mensajes enviados por hasta Si , pero no recibidos por p hasta Si, . Por ejemplo, el estado del canal <C, A> de la figura 13.4b es *(M;, M4}* Indicación: La demostración se basa en el supuesto de que los mensajes se reciben en el orden en que se envían.

13.2 En la figura 1 3.1), considerar que hay un punto en el tiempo donde el nodo 2 ha recibido el mensaje n. pero no el mensaje c/. mientras que el nodo 3 ha recibido el mensaje c/. pero no el mensaje a. De esta forma. durante un tiempo, habrá una visión inconsistente entre los dos nodos. Se ha visto cómo el algoritmo de Lamport resuelve este problema. ¿Produce esto alguna dificultad a alguno de los restantes algoritmos de exclusión mutua discutidos en la sección 13.3?

13.3 En el algoritmo de Lamport, ¿hay alguna circunstancia en que P, pueda ahorrarse la transmisión del mensaje de Respuesta"

13.4 En el algoritmo de exclusión mutua de [RICA81].

a) Probar que se cumple la exclusión mutua.

b) Si los mensajes no llegan en el orden en Clase se enviaron, el algoritmo no garantiza que los secciones críticas se ejecuten por orden de petición. ¿Es posible la inanición'?

13.5 Un el algoritmo de exclusión mutua con paso de testigo, alas marcas de tiempo se usan para borrar los relojes y corregir las derivas, como en el algoritmo de colas distribuidas? Si no es así, ¿cuál es la función de las marcas de

13.6 En el algoritmo de exclusión mutua con paso de testigo, probar que:

a) Garantiza la exclusión mutua

b) Impide el interbloqueo

c) Es equitativo

13.7 Supóngase un sistema distribuido con relojes lógicos. Si el valor del reloj lógico del proceso P, es 12 en el momento en que envía un mensaje al proceso P2 y el valor del reloj lógico del proceso P= es 8 en el momento en que recibe el mensaje de Y,> ¿cuál será el valor del reloj lógico de P2 en el siguiente evento local:' Si, en cambio, el valor del reloj lógico de P, es 15 en el momento de la recepción, ;,cuál será el valor del reloj lógico de P2 en el siguiente suceso local'.'

13.8 En la sección 13.3, uno (le los seis requisitos enumerados paro la exclusión mutua distribuida es que un proceso que se detiene en su sección no crítica debe hacerlo sin interferir con los otros procesos. Supóngase que algún otro proceso ejecuta un algoritmo que envía un mensaje al proceso detenido y después espera una respuesta. ¿Cómo puede evitarse este interbloqueo?

13.9 En la parte (b) de la figura 13.11, explicar por qué la segunda línea no puede mostrar simplemente "petición (j) := I".

La marca *Healhkit* comercializa un reloj que recibe las señales de tiempo difundidas desde la W W V, a partir de un reloj de cesio guardado por el Instituto Nacional de Estándares y Tecnología (NIST), sincronizando su muestra en conformidad. Se tiene en cuenta incluso el retardo de propagación de la señal de radio. Este hardware puede utilizarse para suministrar un reloj distribuido sincronizado. ¿Cómo afectaría esto a los diversos algoritmos propuestos en este capítulo?

13.10 Para el algoritmo de interbloqueo distribuido definido en la figura 13.14, demostrar que sólo son detectados los interbloqueos reales y que lo son en un tiempo limitado. Indicación: Supóngase que una transacción T, hace una petición que origina un ciclo.

CAPÍTULO 1 4

Seguridad

Las exigencias de seguridad de la información en una organización han experimentado dos transformaciones principales en las últimas décadas. Antes del uso extendido de equipos de proceso de datos, la seguridad de la información que se creía valiosa en una organización se aseguraba principalmente por medios físicos y administrativos. Un ejemplo de los primeros es el uso de archivadores robustos con una cerradura de combinación para almacenar documentos delicados. Un ejemplo de los segundos son los procedimientos de vigilancia del personal empleados durante el proceso de contratación.

Con la introducción del computador, se hizo evidente la necesidad de herramientas automatizadas para proteger los archivos y otra información guardada en el computador. Este es, en especial, el caso de un sistema compartido, como los sistemas de tiempo compartido, siendo más acusada la necesidad en los sistemas a los que se puede acceder a través de una red pública telefónica o de datos. El nombre genérico del conjunto de herramientas diseñadas para proteger los datos y frustrar a los piratas informáticos *(hackers)* es el de seguridad de computadores.

La segunda transformación que afectó a la seguridad fue 1a introducción de sistemas distribuidos y el ejemplo de redes y servicios de comunicación para transportar datos entre usuarios finales y computadores, así como entre computadores. Hacen falta medidas de seguridad de redes para proteger los datos durante la transmisión.

El campo de la seguridad de computadores y redes es amplio y abarca controles físicos y administrativos, así como controles automatizados. Este capítulo se limita a considerar las herramientas automatizadas de seguridad. La figura 14.1 sugiere el ámbito de responsabilidad de dichas herramientas. Se comenzará por examinar los tipos de amenazas afrontadas por el compuesto computador-comunicaciones. Más tarde, el grueso del capítulo trata de herramientas específicas que pueden usarse para aumentar la seguridad. La sección 14.2 versa sobre métodos tradicionales de seguridad de computadores, que se basan en la protección de los diversos recursos, incluyendo la memoria y los datos. Después se examinarán las amenazas planteadas por individuos que intenten vencer estos mecanismos de protección. La sección siguiente examina un reciente e inquietante tipo de amenaza: la planteada por virus y mecanismos similares. Seguidamente, se examina un enfoque relativamente nuevo de seguridad, los sistemas de confianza *(trusted syatems)* Esto es seguido por una discusión sobre la seguridad en redes. Finalmente, en un apéndice del capítulo se presenta el cifrado, que es una herramienta básica empleada en muchas aplicaciones de seguridad.

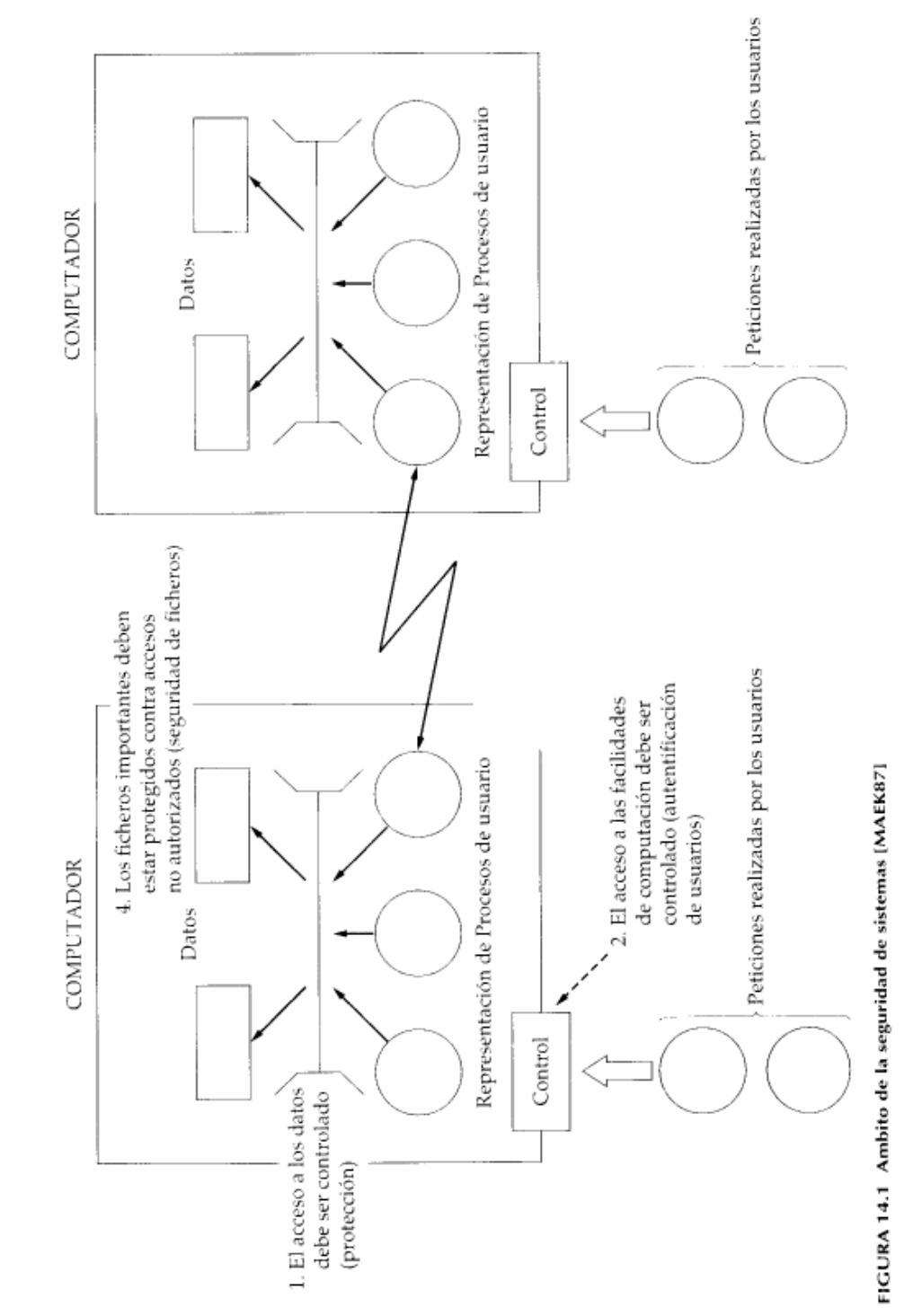


FIGURA 14.1 Ambito de la seguridad de sistemas [MAEK87]

14.1
AMENAZAS A LA SEGURIDAD

Para comprender los diversos tipos de amenazas a la seguridad, hace falta disponer de una definición de los requisitos de seguridad. En la seguridad de computadores y redes se abordan los siguientes requisitos:

- *Confidencialidad:* Exige que la información de un sistema de computadores sea accesible para lectura solamente por grupos autorizados. Este tipo de acceso incluye impresión, visualización y otras formas de revelación, incluyendo el simple revelado de la existencia de un objeto.
- *Integridad:* Exige que los elementos de un sistema de computadores puedan ser modificados sólo por grupos autorizados. La modificación incluye escritura, cambio, cambio de estado, borrado y creación.
- *Disponibilidad* Exige que los elementos de un sistema de computadores estén disponibles a grupos autorizados.

Tipos de Amenazas

Los tipos de amenazas a la seguridad de un sistema de computadores o una red se caracterizan mejor contemplando la función del sistema como suministrador de información. En general, se produce un flujo de información desde un origen, como un archivo o una región de memoria principal, hacia un destino, como otro archivo o un usuario. Este flujo normal está representado en la figura 14.2a. El resto de la figura muestra cuatro categorías generales de amenazas:

- *Interrupción:* Se destruye un elemento del sistema o se hace inasequible o inútil. Esta es una amenaza a la **disponibilidad.** Como ejemplos se incluyen la destrucción de una pieza de hardware, como un disco duro, el corte de una línea de comunicaciones o la inutilización del sistema de gestión de archivos.
- *lntercepción:* Una parte no autorizada consigue acceder a un elemento. Esta es una amenaza a la confidencialidad. La parte no autorizada puede ser una persona, un programa o un computador. Como ejemplos se incluyen la interceptación de las conexiones telefónicas para capturar datos de una red y la copia ilícita de archivos o programas.
- *Alteración:* Una parte no autorizada no sólo consigue acceder, sino que falsifica un elemento. Esta es una amenaza a la **integridad.** Como ejemplos se incluyen el cambio de valores en un archivo de datos, la alteración de un programa para que se comporte de manera diferente y la modificación del contenido de los mensajes transmitidos en una red.
- *Invención:* Una parte no autorizada inserta objetos falsos en el sistema. Esta es también una amenaza a la **integridad.** Como ejemplos se incluyen la inserción de mensajes falsos en una red o la adición de registros a un archivo.

Elementos de un Sistema informático

Los elementos de un sistema informático pueden clasificarse en hardware, software, datos, líneas de comunicación y redes. La tabla 14.1 señala la naturaleza de las amenazas con que se enfrenta cada clase de elemento. Se van a considerar sucesivamente cada una de ellas.

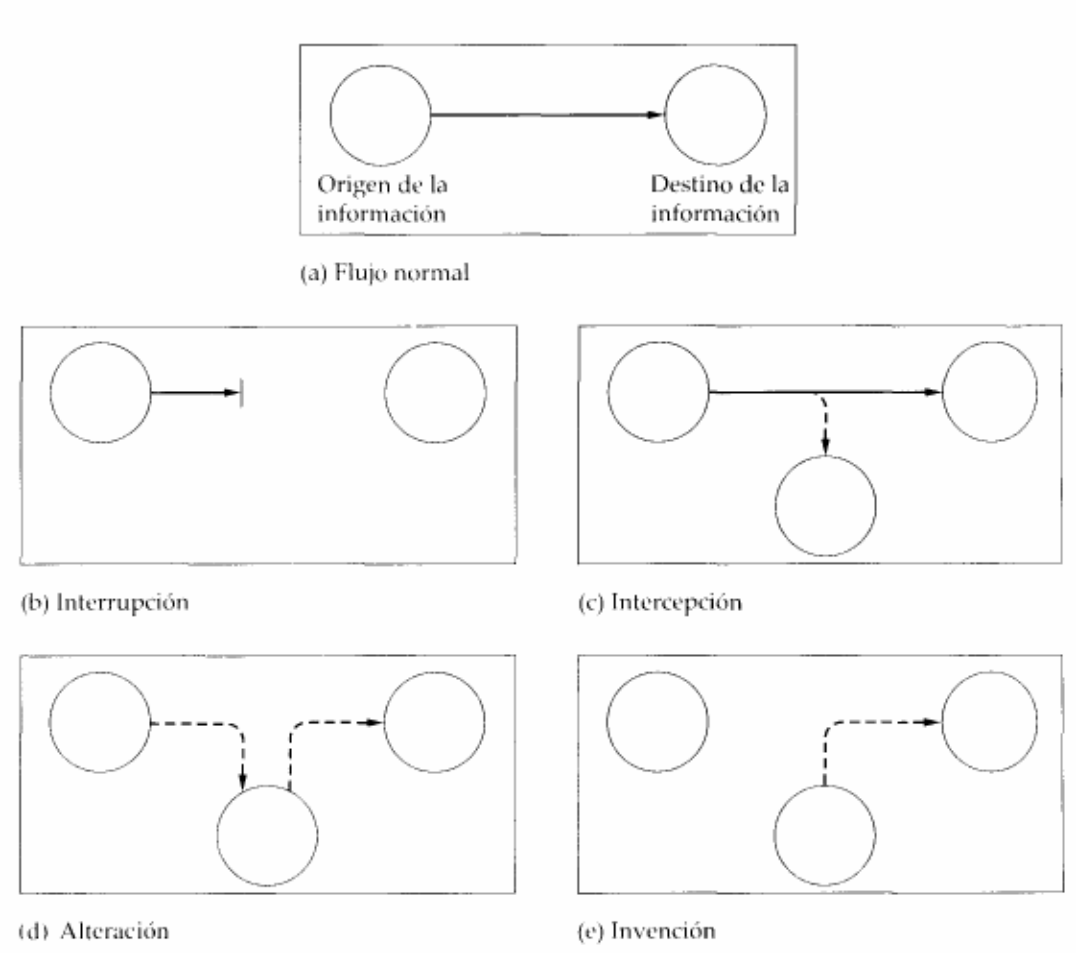


FIGURA 14.2 Amenazas a la seguridad

Hardware

La amenaza principal al hardware de un sistema informático se produce en el campo de la disponibilidad. El hardware es el más vulnerable a los ataques y menos flexible a los controles automatizados. Las amenazas comprenden daños accidentales y deliberados a los equipos, así como el hurto. La proliferación de computadores personales y puestos de trabajo y el uso creciente de redes de área local incrementa la posibilidad de pérdidas de este tipo. Hacen falta medidas de Seguridad físicas y administrativas para hacer frente a estas amenazas. *Software.*

El sistema operativo. las utilidades y los programas de aplicación (el software) son los que hacen que el hardware del sistema sea útil para negocios e individuos. Hay que considerar varias amenazas distintas.

La amenaza principal al software es la disponibilidad. El software, en especial el de aplicaciones, es, asombrosamente fácil de eliminar. El software puede ser alterado o dañado para inutilizarlo. Una gestión cuidadosa de la configuración del software, que incluye la realización de copias de reserva de las versiones más recientes, puede conservar una alta disponibilidad.

TABLA 14.1 Amenazas a la seguridad y Elementos

Elemento	Disponibilidad	Confidencialidad	Integridad
Hardware	Robo o inutilización de equipos, eliminando e I servicio		
Software	Eliminación de programas, denegando el acceso a los usuarios.	Realización de copias no autoriza-das del software.	Alteración de un programa en funcionamiento haciéndolo fallar du rante la ejecución o haciendo que realice alguna tarea no pretendida.
Datos	Eliminación (le archivos, denegando el acceso a los usuarios.	Lecturas de datos no autorizadas. Un analisis de datos estadísticos revela datos ocultos.	Modificación de archivos existentes o invención de nuevos archivos
Líneas de Comunicación	Destrucción o eliminación de mensajes. Las líneas de comunicación o redes se hacen no disponibles.	Lectura de mensajes. Observación de la muestra de tráfico de mensajes.	Mensajes modificados, retardados, reordenados o duplicados. Invención de mensajes falsos.

Un problema más difícil de afrontar es la modificación del software que provoca que un programa siga funcionando pero se comporte de forma diferente que antes. Los virus informáticos y ataques afines caen dentro de esta categoría y se tratarán posteriormente en este capítulo. Un problema final es la confidencialidad del software. Aunque se facilitan ciertas contramedidas, por lo general el problema de la copia no autorizada de software no se ha resuelto.

Datos

La seguridad de hardware y software es normalmente preocupación de los profesionales de centros de proceso de datos o preocupación individual de los usuarios de computadores personales. Un problema mucho más amplio es la seguridad de los datos, donde entran en juego los archivos y otros tipos de datos controlados por individuos, grupos y organizaciones de negocios. El interés de la seguridad con respecto a los datos es amplio, abarcando la disponibilidad, la confidencialidad y la integridad. En el caso de la disponibilidad, la preocupación es la destrucción de los archivos de datos, lo (fue puede ocurrir accidentalmente o como consecuencia de una mala intención.

La preocupación obvia de la confidencialidad es, por supuesto, la lectura no autorizada de archivos o bases de datos. Este campo ha sido objeto de casi más investigación y esfuerzo que cualquier otro aspecto de la seguridad de computadores. Una amenaza menos obvia para la confidencialidad involucra al análisis de datos y se hace evidente en el uso de las llamadas bases de datos estadísticas, que ofrecen información global o resumida. Presumiblemente, la existencia de información global no amenaza la privacidad de los individuos involucrados. Sin em-

bargo, a medida que aumenta la utilización de bases de datos estadísticas, hay una posibilidad creciente de revelación de información personal. Esencialmente, mediante un análisis cuidadoso se pueden identificar las características de los individuos integrantes. Tomando un ejemplo simple, si una tabla registra el total de los ingresos de los sujetos A, *B,* C y D y otra registra el total de los ingresos de A, *B,* C, D y E, la diferencia entre los dos totales sería el ingreso de E. Este problema se acentúa por la creciente intención de combinar conjuntos de datos. En muchos casos, comparar varios conjuntos de datos por coherencia a los niveles de agregación adecuados al problema exige la extracción de las unidades elementales del proceso de construcción de los totales precisos. De este modo, las unidades elementales, que constituyen un objeto de preocupación sobre la privacidad, están disponibles en varios niveles del procesamiento de los conjuntos de datos. Una discusión seria y detallada de estos problemas se ofrece en [DUNN74].

Finalmente, la integridad de los datos es una preocupación fundamental de la mayor parte de instalaciones. Las modificaciones de archivos de datos pueden tener consecuencias poco trascendentes o desastrosas.

Redes y Líneas de Comunicaciones

Los sistemas de comunicaciones se utilizan para transmitir datos. Por tanto, las preocupaciones de disponibilidad, seguridad e integridad que eran importantes para la seguridad de los datos también se aplican a la seguridad de las redes. En este contexto, las amenazas se clasifican de forma conveniente en pasivas y activas (figura 14.3).

Las amenazas pasivas son del género de las escuchas a escondidas o supervisión de las transmisiones de una organización El objetivo del agresor es obtener la información que se esté transmitiendo. Entran en juego aquí dos tipos de amenazas: la divulgación del contenido de los mensajes y el análisis del tráfico.

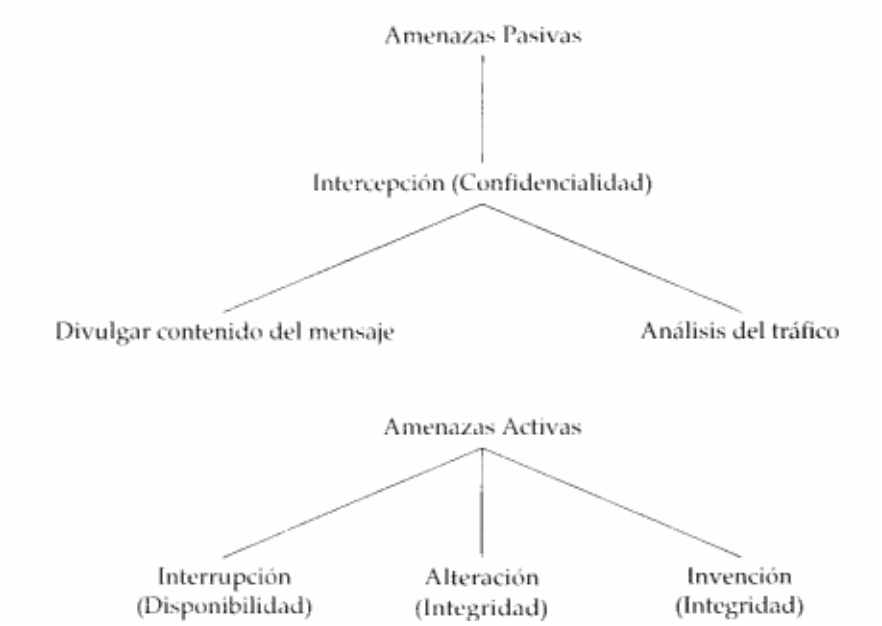


FIGURA 14.3 Amenazas activas y pasivas a la seguridad de las redes

La amenaza de *revelación del contenido de mensajes* es claramente comprensible por la mayoría de los observadores. Una conversación telefónica, un mensaje por correo electrónico o un archivo transferido pueden contener información delicada o confidencial. Se desea evitar cine los agresores conozcan el contenido de estas transmisiones.

La segunda amenaza pasiva, el *análisis del tráfico,* es más sutil y suele ser menos aplicable. Supóngase que se dispone de una forma de enmascarar el contenido de un mensaje u otra información de tráfico de forma que un agresor, aun capturando el mensaje, fuera incapaz de extraer su información. La técnica común de enmascarado es el *cifrado* que será discutido en profundidad más adelante. Si se dispusiera de una protección tal, el agresor aún podría observar el tipo de estos mensajes. El agresor podría determinar la ubicación y la identidad de los computadores que se comunican y también podría observar la frecuencia y longitud de los mensajes intercambiados. Esta información podría ser útil para adivinar la índole de la comunicación que tuviera lugar.

Las amenazas pasivas son muy difíciles de detectar porque no acarrean alteración alguna de los datos. Sin embargo, es factible impedir que estos ataques tengan éxito. Por tanto, la importancia de hacer frente a las amenazas pasivas está en la prevención y no en la detección.

En la segunda clase principal de amenazas se **encuentran las amenazas activas,** que suponen alteraciones del flujo de datos o la creación de un flujo falso. Se pueden subdividir estas amenazas en tres categorías: alteración del flujo de mensajes, privación del servicio de mensajería y suplantación.

La *alteración del flujo de mensajes* simplemente supone que se modifica una porción de un mensaje legítimo o que los mensajes se retrasan, se repiten o se reordenan para conseguir un efecto no autorizado. Por ejemplo, un mensaje que diga "Permitir a John Smith leer el archivo confidencial *cuentas"* se modifica para que diga "Permitir a Fred Brown leer el archivo confidencial *cuentas"*

La *privación de servicio* impide o inhibe el uso normal o la gestión de servicios de comunicaciones. Esta agresión puede tener un objetivo específico; por ejemplo, una entidad puede suprimir todos los mensajes dirigidos a un destino particular (como el servicio de auditoría de seguridad). Otra forma de privación de servicio es la interrupción de toda una red, incapacitándola o sobrecargándola con mensajes que degraden el rendimiento. Una *suplantación* tiene lugar cuando una entidad finge ser una entidad diferente. El ataque por suplantación generalmente incluye alguna de las otras dos formas de ataques activos. Dichos ataques pueden tener lugar, Por ejemplo, capturando y repitiendo una secuencia de autentificación.

Las amenazas activas presentan las características opuestas a las pasivas. Aunque los ataques pasivos son difíciles de detectar, hay medidas disponibles que impiden su triunfo. Por otra parle, es bastante difícil prevenir de forma absoluta los ataques activos porque la prevención requeriría una protección física de todos los servicios y rutas de comunicaciones en cualquier momento. En su lugar, el objetivo con respecto a los ataques activos es detectarlos y recuperarse de cualquier interrupción o retardo causado. La detección tiene un efecto disuasorio que también puede contribuir a la prevención.

Principios de Diseño

[SALT75] identifica una serie de principios en el diseño de medidas de seguridad para las diversas amenazas a los sistemas informáticos. Entre estos principios se incluyen los siguientes:

- *Mínimo privilegio:* Todos los programas y usuarios del sistema deben operar utilizando el menor conjunto de privilegios necesarios para completar la labor. Los derechos de acceso deben adquirirse sólo por permiso explícito; por omisión deberían ser "sin acceso".
- *Ahorro de mecanismos:* Los mecanismos de seguridad deben ser tan pequeños y simples como sea posible, ayudando en su verificación. Esta exigencia suele suponer que deben ser una parte integral del diseño, más que mecanismos añadidos a diseños existentes.
- *Aceptación:* Los mecanismos de seguridad no deben interferir excesivamente en el trabajo de los usuarios, mientras cumplen al mismo tiempo las necesidades de aquellos que autoricen el acceso. Si los mecanismos no son fáciles de usar, probablemente no van a ser usados o lo serán de forma incorrecta.
- *Mediación total:* Cada acceso debe ser cotejado con la información de control de acceso, incluyendo aquellos accesos que suceden fuera de la operación normal, como la recuperación y el mantenimiento.
- *Diseño abierto:* La seguridad del sistema no debe depender de guardar en secreto el diseño de sus mecanismos. De esta forma, los mecanismos podrán ser revisados por muchos expertos y los usuarios podrán, por tanto, depositar una alta confianza en ellos.

14.2 PROTECCIÓN

La introducción de la multiprogramación originó la posibilidad de compartir recursos entre los usuarios. La compartición compromete no sólo al procesador, sino también a lo siguiente

- Memoria
- Dispositivos de E/S, como discos e impresoras
- Programas
- Datos

La capacidad ele compartir recursos introdujo la necesidad de protección. [PFLE89] señala que un sistema operativo puede ofrecer protección en el siguiente abanico:

- *Ninguna protección:* Apropiada cuando se ejecutan procedimientos delicados en momentos distintos.
- *Aislamiento:* Este enfoque implica que cada proceso opera separadamente de los demás, sin compartición ni comunicación. Cada proceso tiene su propio espacio de direcciones, archivos y otros objetos.
- *Compartir todo o fiada:* El propietario de un objeto (por ejemplo, un archivo o un segmento de memoria) los declara como público o privado. En el primer caso, cualquier proceso puede acceder al objeto; en el último caso, sólo los procesos del propietario pueden acceder al objeto.
- *Compartir por limitación del acceso:* El sistema operativo comprueba la licencia de cada acceso de un usuario específico a un objeto determinado. El sistema operativo actúa, por lo tanto, como un guarda o portero entre usuarios y objetos, asegurando que sólo ocurren accesos autorizados.

- *Compartir por capacidades dinámicas:* Este tipo de protección amplía el concepto de control de acceso, incorporando la creación dinámica de derechos de compartición para los objetos.
- *Uso limitado de un objeto:* Esta forma de protección limita no sólo el acceso a un objeto, sino también la utilidad a que se puede dedicar dicho objeto. Por ejemplo, se puede permitir a un usuario ver un documento delicado, pero no imprimirlo. Otro ejemplo: Se puede permitir a un usuario acceder a una base de datos para sacar resúmenes estadísticos, pero no determinar valores de datos específicos.

Los elementos anteriores están enumerados más o menos en orden creciente de dificultad de implementar, pero también en orden creciente de la bondad de la protección que ofrecen. Un sistema operativo determinado puede ofrecer grados diferentes de protección para distintos objetos, usuarios o aplicaciones. Hace falta que el sistema operativo equilibre la necesidad de compartir, lo que aumenta la utilidad del sistema informático, con la necesidad de proteger los recursos de los usuarios individuales. En esta sección se consideran algunos de los mecanismos, mediante los cuales, los sistemas operativos han hecho respetar la protección de estos objetos.

Protección de Memoria

En un entorno de multiprogramación, la protección de la memoria principal es fundamental. El interés no es sólo la seguridad, sino también el funcionamiento correcto de los diversos procesos que estén activos. Si un proceso puede escribir inadvertidamente en el espacio de memoria de otro proceso, este último puede que no ejecute correctamente. La separación del espacio de memoria de los diversos procesos se lleva a cabo fácilmente con un esquema de memoria virtual. La segmentación, paginación o la combinación de ambas proporciona un medio eficaz de gestión de la memoria principal. Si se persigue un aislamiento total, el sistema operativo simplemente debe asegurar que cada segmento o cada página es accesible sólo para el proceso al que está asignada. Esto se lleva a cabo fácilmente exigiendo que no haya entradas duplicadas en las tablas de páginas o segmentos. Si se va a permitir la compartición, el mismo segmento o página puede ser referenciado en más de una tabla. Este tipo de compartición se consigue mejor en un sistema que soporta segmentación o una combinación de segmentación y paginación.. En tal caso, la estructura del segmento es visible a la aplicación y la aplicación puede declarar segmentos individuales como compartibles o no compartibles. En un entorno de paginación pura, se hace más difícil discriminar entre los (los tipos (le memoria debido a que la estructura de memoria es transparente a la aplicación.

Un ejemplo del soporte de hardware que puede ofrecerse para la protección de memoria es el de la familia de máquinas IBM Sistema/370, donde se ejecuta MVS. Asociado con cada marco de página en memoria principal hay una clave de control de almacenamiento de 7 bits, que puede ser ajustada por cl sistema operativo. Dos de los bits indican si la página que ocupa un marco ha sido referenciada y ha cambiado; estos bits son usados por el algoritmo de reemplazo de páginas. Los bits restantes son usados por el mecanismo de protección: una clave de control de acceso de 4 bits y un hit de protección de ciclo *(fetch)* Las referencias del procesador a memoria y las referencias a memoria de la E/S por DMA deben emplear una clave correcta para obtener permiso para acceder a la página. El bit de protección de ciclo indica si la clave de control de acceso se aplica a las Escrituras, o tanto a Lecturas como Escrituras En el procesador existe una palabra de estado del programa (PSW)

que contiene información de control relativa al proceso que se está ejecutando en un momento dado. Incluida en esta palabra hay una clave PSW de 4 bits. Cuando un proceso intenta acceder a una página o iniciar una operación DMA sobre una página, la clave de la PSW actual es comparada con el código de acceso. Una operación de Escritura es permitida si los códigos coinciden. Si el bit de lectura está activo, entonces la clave de la PSW debe verificar el código de acceso para operaciones de Lectura.

Control de acceso orientado al usuario

Las medidas tomadas para controlar el acceso en los sistemas de proceso de datos pueden encuadrarse en dos categorías: las asociadas con el usuario y las asociadas con los datos. Al control de acceso al usuario se le conoce a veces, de forma desafortunada, por autentificación Como este término se usa mucho en el sentido de autentificación de los mensajes, se evitará su aplicación aquí. Se avisa, sin embargo, de que se puede encontrar este uso en la bibliografía.

La técnica más habitual de control de acceso al usuario, en un sistema de tiempo compartido o en un servidor, es en la conexión del usuario, que requiere un identificador de usuario (ID) y una contraseña. El sistema permitirá a un usuario conectarse sólo si el ID es conocido por el sistema y si el usuario sabe la contraseña asociada por el sistema a dicho ID. Este esquema ID/contraseña es un método notablemente poco fiable de control de acceso al usuario. Los usuarios pueden olvidar sus contraseñas y pueden revelarlas accidental o deliberadamente. Los piratas informáticos (hackers) son muy habilidosos en adivinar los ID de usuarios específicos, como el personal de controlo de gestión del sistema. Por último, el esquema ID/contraseña está sujeto a los intentos de penetración. En la sección 14.3 se discutirán las contramedidas.

El problema del control de acceso a los usuarios se complica en las redes de comunicaciones. El diálogo de conexión debe tener lugar a través del medio de comunicación y las escuchas se convierten en una amenaza potencial: En tal caso, deben aplicarse los enfoques de seguridad en redes discutidos en este capítulo.

El control de acceso al usuario en entornos distribuidos puede ser centralizado o descentralizado. Con un enfoque centralizado, la red proporciona un servicio de conexión para determinar a quién se le permite usar la red y a qué se le permite conectarse. El control de acceso descentralizado considera la red cono un enlace transparente de comunicaciones y el procedimiento usual de conexión lo lleva a cabo el servidor de destino. Desde luego, debe seguir considerándose la seguridad concerniente a la transmisión de contraseñas por la red.

En muchas redes, pueden emplearse dos niveles de control de acceso. Los servidores individuales pueden estar provistos de un servicio de conexión que proteja las aplicaciones y recursos específicos del servidor. Además, la red en conjunto puede ofrecer una protección para restringir el acceso a la red a los usuarios no autorizados. Este servicio a dos niveles es conveniente en el caso habitual de que la red conecte servidores dispares y proporcione simplemente un medio oportuno de acceso de los terminales al servidor. En una red más uniforme de servidores, podría aplicarse alguna política centralizada de acceso en un centro de control de la red.

Control de acceso orientado a los datos

Después de una conexión con éxito, al usuario se le habrá concedido el acceso a uno o más servidores y aplicaciones. Esto no suele ser suficiente en un sistema que incluya datos sensibles en su base de datos. Mediante el procedimiento de control de acceso al usua-

rio, un usuario puede identificarse ante el sistema Asociado con cada usuario, puede haber un perfil de usuario que especifique las operaciones y los accesos a archivos permisibles. El sistema operativo puede hacer valer unas reglas en función del perfil del usuario. El sistema gestor de la base de datos, sin embargo, debe controlar el acceso a registros específicos o incluso partes de un registro. Por ejemplo, puede permitirse que cualquier administrador obtenga un listado del personal de una compañía, pero solamente unos individuos elegidos pueden tener acceso a la información de salarios. La cuestión tiene más de un nivel de detalle.

Mientras que el sistema operativo puede otorgar a un usuario permiso para acceder a un archivo o utilizar una aplicación, tras lo cual no se producen más controles de seguridad, el sistema gestor de la base de datos debe tomar decisiones sobre cada intento de acceso individual. Dicha decisión dependerá no sólo de la identidad del usuario, sino también de las partes específicas de datos a las que se accede e, incluso, de la información ya divulgada al usuario.

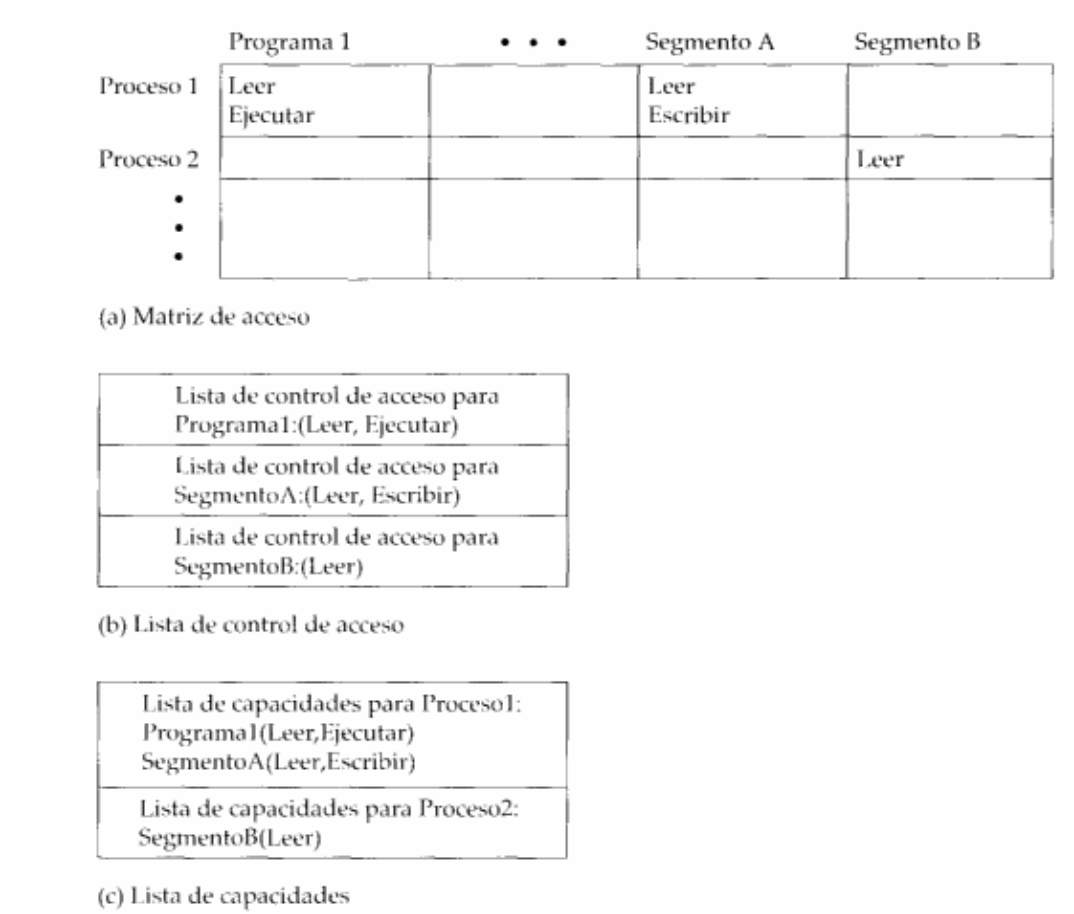
Un modelo general de control de acceso ejercido por un sistema gestor de archivos o bases de datos es el de una **matriz de acceso** (figura 14.4a). Los elementos básicos del modelo son los siguientes:

- *Sujeto:* Una entidad capaz de acceder a los objetos. En general, el concepto de sujeto es equiparable con el de proceso. Cualquier usuario o aplicación consigue acceder en realidad a un objeto por medio de un proceso que representa al usuario o la aplicación.
- *Objeto:* Cualquier cosa cuyo acceso debe controlarse. Como ejemplos se incluyen los archivos, partes de archivos, programas y segmentos de memoria.
- *Derecho de acceso* La manera en que un sujeto accede a un objeto. Como ejemplos están Leer, Escribir y Ejecutar.

Una dimensión de la matriz consta de los sujetos identificados que pueden intentar acceder a los datos. Normalmente, esta lista consta de usuarios individuales o de grupos de usuarios, aunque se puede controlar el acceso para terminales, servidores o aplicaciones, en su lugar o en conjunto. La otra dimensión enumera los objetos a los que se puede acceder. En el mayor nivel de detalle, los objetos pueden ser campos de datos individuales. También pueden ser objetos de la matriz agrupaciones más globales, como registros, archivos o incluso la base de datos entera. Cada entrada de la matriz indica los derechos de acceso del sujeto al objeto.

En la práctica, las matrices de acceso suelen estar dispersas y se implementan por descomposiciones en una o dos de las dimensiones. La matriz se puede descomponer en columnas, obteniéndose **listas de control de acceso** (figura 14.4b). Así pues, para cada objeto, una lista de control de acceso (ACL, *Access Control List)* expresa los usuarios y sus derechos de acceso permitidos. La lista de control de acceso puede contener una entrada por omisión o pública. Se permite que los usuarios a los que no se les haya concedido explícitamente unos derechos especiales dispongan de un conjunto de derechos por omisión. Los elementos (le la lista por omisión pueden incluir a usuarios individuales, así como a grupos de usuarios.

Con la descomposición por filas se obtienen **etiquetas de capacidades** (capabilities) (figura 14.4c). Una etiqueta de capacidades especifica los objetos y las operaciones autorizadas para un usuario. Cada usuario tiene un número de etiquetas y puede estar autorizado para prestarlas o concederlas a los otros. Como las etiquetas pueden estar dispersas por el



	Programa 1	• • •	Segmento A	Segmento B
Proceso 1	Leer Ejecutar		Leer Escribir	
Proceso 2				Leer
•				
•				
•				

(a) Matriz de acceso

Lista de control de acceso para Programa1:(Leer, Ejecutar)
Lista de control de acceso para SegmentoA:(Leer, Escribir)
Lista de control de acceso para SegmentoB:(Leer)

(b) Lista de control de acceso

Lista de capacidades para Proceso1: Programa1(Leer,Ejecutar) SegmentoA(Leer,Escribir)
Lista de capacidades para Proceso2: SegmentoB(Leer)

(c) Lista de capacidades

FIGURA 14.4 Estructuras de control de acceso

sistema, presentan un problema de seguridad mayor que el de las listas de control de acceso. En concreto, las etiquetas no pueden ser falsificables. Una manera de conseguirlo es que el sistema operativo guarde todas las etiquetas en vez de los usuarios. Las etiquetas deben guardarse en una zona de memoria inaccesible para los usuarios.

Las consideraciones sobre las redes en el control de acceso orientado a los datos son análogas a las del control de acceso orientado a los usuarios. Si sólo se permite a ciertos usuarios acceder a ciertos elementos de datos, puede hacer falla el cifrado para proteger estos elementos durante su transmisión a los usuarios autorizados. Normalmente, el control de acceso orientado a los datos es descentralizado, es decir, controlado mediante sistemas gestores de bases de datos radicados en los servidores. Si hay un servidor de bases de datos en una red, el control de acceso a los datos se convierte en una función de la red.

Windows NT

Un buen ejemplo de los conceptos de control de acceso que se han expuesto es el de Windows NT, que aprovecha los conceptos de orientación a objetos para ofrecer una capacidad potente y flexible de control de acceso.

Windows NT ofrece un servicio uniforme de control de acceso que se aplica a los procesos, hilos, archivos, semáforos, ventanas y otros objetos. El control de acceso está gobernado por dos entidades: una señal de acceso asociada con cada proceso y un descriptor de seguridad asociado con cada objeto para el que es posible el acceso entre procesos.

Esquema de control de accesos

Cuando un usuario se conecta a un sistema NT, éste utiliza un esquema de nombre/contraseña para autentificar al usuario. Si se acepta la conexión, se crea un proceso para el usuario y se le asocia una señal de acceso al objeto que representa a dicho proceso. La señal de acceso, cuyos detalles se describen más tarde, incluye un ID de seguridad (SID, Security ID), que es el identificador por medio del cual el usuario es conocido ante el sistema con fines de seguridad. Cuando el proceso inicial del usuario genera más procesos, los objetos de estos nuevos procesos heredan la misma señal de acceso.

Las señales de acceso sirven para dos fines:

1. Reunen toda la información de seguridad necesaria para acelerar la validación de los accesos. Cuando un proceso asociado a un usuario intenta acceder, el subsistema de seguridad puede hacer uso de la señal asociada a dicho proceso para determinar los privilegios de acceso del usuario.
2 Permite a los procesos modificar sus características de seguridad de forma limitada, sin afectar a otros procesos lanzados por el usuario.

El significado principal del segundo punto tiene que ver con los privilegios que un usuario puede tener asociados. La señal de acceso indica los privilegios que un usuario puede tener. En general, la señal se inicializa con estos privilegios en un estado inhabilitado. A continuación, si alguno de los procesos del usuario necesita llevar a cabo una operación privilegiada, el proceso puede habilitar el privilegio apropiado e intentar el acceso. No sería conveniente mantener toda la información de seguridad de un usuario en un lugar para todo el sistema, porque, en tal caso. habilitar un privilegio para un proceso significaría habilitarlo para todos.

Hay un descriptor de seguridad asociado con cada objeto para el que es posible el acceso entre procesos. El componente principal del descriptor de seguridad es una lista de control de acceso que especifica los derechos de acceso de varios usuarios y grupos de usuarios para el objeto. Cuando un proceso intenta acceder al objeto, se compara el SID del proceso con la lista de control de acceso del objeto, para determinar si se permite el acceso.

Cuando una aplicación abre una referencia a un objeto asegurable, NT verifica que el descriptor de seguridad del objeto concede el acceso al usuario de la aplicación. Si la comprobación tiene éxito, NT guarda en una cache los derechos de acceso concedidos que resultan.

Un aspecto importante de la seguridad de NT es el concepto de imitación, que simplifica la seguridad en un entorno cliente/servidor. Si el cliente y el servidor hablan a través de una conexión por RPC, el servidor puede asumir temporalmente la identidad del cliente, de forma que pueda evaluar una solicitud de acceso relativa a los derechos de dicho cliente. Tras el acceso, el servidor vuelve a tomar su propia identidad.

Señales de acceso

La figura 14.5a muestra la estructura general de una señal de acceso, que incluye los siguientes parámetros:

- ID de seguridad: Identifica unívocamente a cada usuario de todas las máquinas de la red. Este corresponde en general al nombre de conexión del usuario.
- SID de grupos: Una lista de los grupos a los que pertenece el usuario. Un grupo es simplemente un conjunto de ID de usuarios que se identifican como grupo con fines de control de acceso. Cada grupo tiene un único SID de grupo. El acceso a un objeto puede definirse en función de SID de grupos, SID individuales o una combinación de ambos.
- Privilegios: Una lista de servicios del sistema sensibles a la seguridad que el usuario puede pedir. Un ejemplo es crear señal Otro ejemplo es activar privilegio de copla de reserva los usuarios con dicho privilegio pueden usar una herramienta de copia para hacer copias de reserva de los archivos que generalmente no serían capaces de leer. La mayoría de los usuarios no tendrán privilegios.
- Propietario por omisión: Si un proceso crea otro objeto, este campo especifica quién es el propietario del nuevo objeto. En general, el propietario de un proceso nuevo es el mismo que el del proceso que lo crea. No obstante, un usuario puede especificar que el propietario por omisión de todos los procesos lanzados por un proceso sea el SID de un grupo al que pertenezca el usuario.
- ACL por omisión: Es una lista inicial de protecciones que se aplican a los objetos que crea el usuario. El usuario puede modificar a continuación la ACL para cualquier objeto que posea o que posea uno de sus grupos.

Descriptores de seguridad

La figura 14.5b muestra la estructura general de un descriptor de seguridad, que incluye los siguientes parámetros:

- Indicadores:: Definen el tipo y el contenido de un descriptor de seguridad. Los indicadores muestra» si están presentes o no la SACL y la DACL, si están situadas o no en el objeto con un mecanismo por omisión y si los punteros del descriptor utilizan direccionamiento absoluto o relativo. Los descriptores relativos son necesarios para los objetos transmitidos por la red, tales como la información transmitida en una RPC.
- Propietario: El propietario del objeto puede realizar en general cualquier acción sobre el descriptor de seguridad. El propietario puede ser un SID individual o de grupo. El propietario dispone de autoridad para cambiar el contenido de la DACL.
- Lista de Control de Acceso del Sistema (SACL, System Access Control List): Especifica los tipos de operación sobre el objeto que deben generar mensajes de auditoría. Una aplicación debe disponer del privilegio pertinente en su señal de acceso para leer o escribir en la SACL de cualquier objeto. Esto sirve para impedir a las aplicaciones no autorizadas que lean las SAC (aprendiendo, por lo tanto, qué no deben hacer y así evitar generar sucesos de auditoría) o cine escriban en ellas (y así generar tantos sucesos de auditoría que hagan que una operación ilícita pase desapercibida).
- Lista de Control de Acceso Discrecional (DACL, Discretionary Access Control List): Determina qué usuarios y grupos pueden acceder a un objeto y para qué operaciones. Consta de una lista de entradas de control de acceso (ACE, Access Control Entry).

Cuando se crea un objeto, el proceso creador, corno propietario que es, puede asignarle una sedal de acceso con su propio SID o el SID de algún grupo. El proceso creador no puede

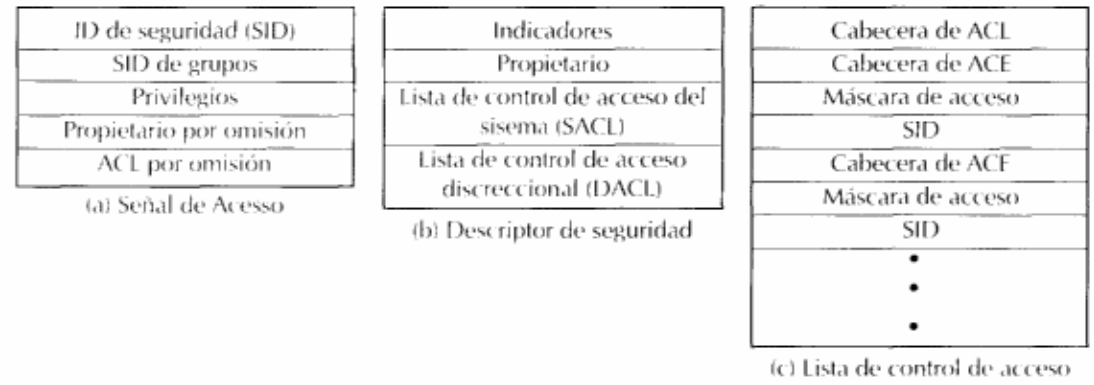


FIGURA 14.5 Estructuras de seguridad de Windows NT

asignarle un propietario que no esté en la señal de acceso actual. Por consiguiente, cualquier proceso que disponga del derecho de cambiar el propietario de un objeto podrá hacerlo, pero también con la misma restricción. La razón de dicha limitación es impedir a un usuario borrar sus huellas tras intentar una acción no autorizada.

Va a observarse con mayor detalle la estructura de las listas de control de acceso, puesto que constituyen el alma del servicio de control de acceso de NT (figura 14.5c). Cada lista consta de una cabecera global y, de un número variable de ACE. Cada entrada especifica a un SID individual o de grupo y una máscara de acceso que define los derechos otorgados a dicho SID. Cuando un proceso intenta acceder a un objeto, el administrador de objetos del ejecutor de NT lee el SID y el SID de `Tropo de la señal de acceso y recorre la DACL del objeto. Si se encuentra una correspondencia, es decir, si se encuentra una ACE con un SID igual a uno de los SID de la señal de acceso, el proceso tendrá los derechos especificados en la máscara de acceso de dicha ACE.

La figura 14.6 muestra el contenido de la máscara de acceso. Los 16 bits menos significativos especifican los derechos de acceso que se aplican a un tipo particular de objeto. Por ejemplo, el bit 0 de un objeto archivo significa derecho para Leer Datos_Archivo y el bit 0 de un objeto suceso significa derecho para Consultar Estado Suceso.

Los 16 bits más significativos de la máscara se aplican a todos los tipos de objetos. Cinco de estos bits se conocen como tipos de acceso estándar:

- *Sincronizar:* Concede permiso para sincronizar la ejecución con algún suceso asociado al objeto. En particular, el objeto puede usarse en una función *wait.*
- *Escribir_propietario:* Permite que un programa modifique el' propietario del objeto. Es útil porque el propietario de un objeto siempre puede cambiar la protección del objeto (al propietario no se le puede denegar el acceso de Escribir en la DACL).
- *Escribir_DAC:* Permite a la aplicación modificar la DACL y, por tanto, la protección del objeto.
- *Leer -control:* Permite a la aplicación consultar los campos propietario y DACL del descriptor de seguridad del objeto.
- *Eliminar:* Permite a la aplicación eliminar el objeto.

La mitad más significativa de la ACE también contiene los cuatro tipos de acceso genéricos Estos bits proporcionan una forma cómoda de activar tipos de acceso específicos

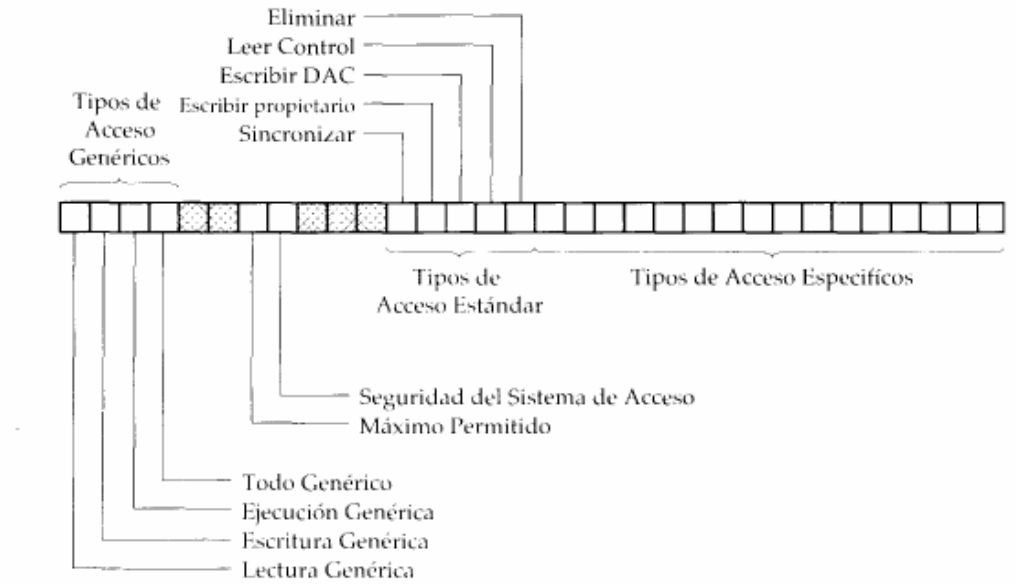


FIGURA 14.6 Máscara de acceso

en una serie de tipos diferentes de objetos. Por ejemplo, supóngase que una aplicación desea crear varios tipos de objetos y asegurarse que los usuarios tienen acceso para *leer* los objetos, incluso aunque leer tenga un significado algo diferente para cada tipo de objeto. Para proteger los objetos de cada tipo sin los bits de acceso genérico, la aplicación tendría que construir una ACE distinta para cada tipo de objeto y tener cuidado de pasar la ACE correcta cuando crea cada objeto. ES más conveniente crear una única ACE que exprese el concepto genérico permitir leer; aplicando simplemente esta ACE a cada objeto que se cree se tendrá la situación deseada. Esta es la finalidad de los bits de acceso genéricos, que son:

- Todo_genérico: permitir todos los accesos
- Ejecución genérica: permitir la ejecución si es ejecutable
- Escrituro genérica: permitir acceso de escritura
- Lee tura ge iré ric a: permitir acceso sólo de lectura

Los bits genéricos también influyere en los tipos de acceso estándar. Por ejemplo, para un objeto archivo, el bit Lectura_genérica se corresponde con los bits estándares Leer Control y Sincronizar y con los bits específicos de objeto Leer-Datos-Archivo, Leer-Atributos _Archivo y Leer -EA Archivo. Si se sitúa una ACE en un objeto archivo que concede Lectura_genérica a algún SID, se conceden los cinco derechos de acceso al igual que si se hubieran especificado individualmente en la máscara de acceso.

Los dos bits restantes de la ACE tienen un significado especial. El bit de Seguridad Sistema Acceso permite modificar el control de alarma y auditoría para el objeto. Sin embargo este bit no sólo debe estar activado en la ACE para un SID, sino que la señal de acceso del proceso con dicho SID *debe tener* el privilegio correspondiente habilitado.

Por último, el bit Máximo-Permitido no es en realidad un bit de acceso, sino un bit que modifica el algoritmo que NT usa para recorrer la DACL en busca de un SID. Generalmente, NT recorrerá la DACL hasta que encuentre una ACE que conceda específicamente (bit activado) o deniegue (bit no activado) el acceso solicitado por el proceso demandante o hasta que encuentre el final de la DACL, en cuyo caso se niega el acceso. El bit Máximo_Permitido permite al propietario de un objeto definir un conjunto de derechos de acceso máximo que se le puede dar a un usuario. Teniendo esto en cuenta, supóngase que una aplicación no conoce todas las operaciones que va a pedir que se realicen sobre un objeto durante la sesión. Hay tres opciones para solicitar acceso.

1. Intentar abrir el objeto con todos los posibles accesos. La desventaja de este método es que puede denegarse el acceso incluso si la aplicación puede disponer de todos los derechos necesarios para la sesión.
2. Abrir solamente el objeto cuando se solicite un acceso específico y abrir un descriptor nuevo del objeto para cada tipo diferente de petición. Éste es el método más preferido, porque no denegará el acceso innecesariamente ni permitirá más accesos que los necesarios. Sin embargo, impone una sobrecarga adicional.
3. Intentar abrir el objeto para tantos accesos como permita para el SID. La ventaja es que no se privará artificialmente del acceso al usuario, pero la aplicación puede disponer de más accesos de los que necesita. Esta última situación puede enmascarar errores en la aplicación.

Una característica importante de la seguridad de NT es que las aplicaciones pueden aplicar el entorno de seguridad de NT a objetos definidos por el usuario. Por ejemplo, un servidor de pases de datos podría crear sus propios descriptores de seguridad y engancharlos a determinadas partes de una base de datos. Además de las típicas restricciones de lectura/escritura, el servidor podría hacer seguras determinadas operaciones de la base de datos, como navegar por un conjunto de resultados o llevar a cabo una unión natural. Sería responsabilidad del servidor definir el significado de los derechos específicos y realizar comprobaciones de acceso. Lo interesante es que las comprobaciones tendrían lugar en un contexto estándar, se utilizarían cuentas de usuarios y grupos del sistema e informes de auditoría del sistema. Aquellos que implementen sistemas de archivos ajenos pueden comprobar la utilidad del modelo ampliable de seguridad.

14.3 INTRUSOS

Una de las dos amenazas más conocidas a la seguridad (la otra es la de los virus) es el intruso, conocido en general como pirata *(hacker o cracker).* Los ataques de intrusos varían desde benignos hasta serios. En el extremo benigno de la escala, hay mucha gente que simplemente desea explorar las interredes y ver que hay ahí fuera. En el extremo serio están los individuos que intentan leer datos privilegiados, realizar modificaciones no autorizadas a los datos o trastornar el sistema

La amenaza de los intrusos ha sido muy publicada, siendo particularmente famoso el incidente del "Pirata Astuto` de 1986-1987, documentado por Cliff Stoll [STOL88, 89].

En inglés, Wily Hacker

En 1990 se tomaron medidas enérgicas en todos los EE.UU. contra los piratas informáticos ilegales, con arrestos, imputaciones criminales, juicios, y confiscación de cantidades masivas de datos y equipos de computadores [STER92]. Mucha gente creyó que el problema ya se tenía bajo control.

De hecho, el problema no se había sometido a control alguno. Para citar un ejemplo reciente, un grupo de los Laboratorios Bell [BELL92, BELL931 informó sobre ataques frecuentes y persistentes a su complejo de computadores a través de Internet durante un amplio periodo de tiempo y desde una gran variedad de orígenes. En el momento de dichos informes, estaban sufriendo lo siguiente:

- Intentos de copiar el archivo de contraseñas (discutido más tarde), a razón de más de una vez al día
- Solicitudes sospechosas de llamadas a procedimientos remotos (RPC), a razón de más de una vez por semana.
- Intentos de conexión a maquinas "cebo" inexistentes, al menos cada dos semanas.

Los intrusos benignos podían tolerarse, aunque consumían recursos y podían ralentizar el rendimiento de los usuarios legítimos. Sin embargo, no hay forma de conocer por adelantado si un intruso será benigno o maligno. En consecuencia, incluso para los sistemas sin recursos susceptibles, hay un motivo para controlar este problema. Más aún, los ataques serios de intrusos son un problema real y creciente. [FREE93] enumera las siguientes razones como básicas para esta tendencia:

- *Globalización:* Lis presión de la competencia internacional ha provocado una serie de casos recientes de espionaje industrial. También hay evidencia de que una serie de clubs de piratas" están comenzando a ofrecer sus servicios con tal fin.
- *El cambio /lacia la arquitectura cliente/servidor:* Las compañías han guardado la mayoría de sus datos tradicionalmente en computadores centrales, que pueden ser protegidas con un software sofisticado de seguridad o en PC solitarios, que normalmente no han estado accesibles de forma remota. Pero a medida que la arquitectura cliente/servidor se hace cada vez más popular, ambas barreras se están eliminando. La mayoría de los servidores ejecutan UNIX, que es notorio por la falta de seguridad del estilo de los computadores centrales y que es favorito en concreto para los piratas.
- *La curva de aprendizaje tan inclinada de los piratas:* A los piratas les encanta compartir información. Se utilizan tablones de anuncios clandestinos para intercambiar números de teléfono de puertos de conexión, contraseñas comprometidas, agujeros en la seguridad de los sistemas y técnicas de intrusión [HAFN91, STER92]. Debido a la reticencia innata del personal de seguridad y sistemas para compartir información relativa a la seguridad, especialmente los puntos vulnerables preocupantes, los intrusos son más capaces que sus adversarios de estar al corriente de los últimos trucos de vulnerabilidad en comercios y compañías. Es más, cuando el personal de seguridad intercambia información sobre los puntos vulnerables, los atacantes suelen poder escuchar y aprovecharse de estos puntos vulnerables antes de que los agujeros se rellenen en todos los sistemas afectados.

Un ejemplo cine ilustra drásticamente este último punto sucedió en la Universidad A&M de Texas [SAFF93]. En agosto de 1992, el centro (le cálculo fue avisado de que una de sus máquinas se estaba utilizando para atacar computadores de otros lugares de Internet. Supervisando las actividades, el personal del centro de cálculo aprendió que había varios intrusos

externos involucrados que estaban ejecutando rutinas para averiguar contraseñas de varias computadores (el lugar constaba de un total de 12.000 máquinas interconectadas). El centro desconectó las máquinas afectadas, rellenó los agujeros conocidos en la seguridad y reanudó el Funcionamiento normal. Unos días más tarde, uno de los administradores de sistemas locales detectó que el ataque de los intrusos se había reanudado. Resultó que el ataque era mucho más sofisticado de lo que se había creído originalmente. Se encontraron archivos que contenían cientos de contraseñas capturadas, incluyendo algunas de los servidores principales y, supuestamente, más seguros. Además, una máquina local había sido preparada como un tablón de anuncios pirata, que los piratas utilizaban para contactar unos con otros y discutir las técnicas y su progreso.

Un análisis de estos ataques reveló que había en realidad dos niveles de piratas. Los atacantes de alto nivel eran usuarios sofisticados con un conocimiento completo de la tecnología; los atacantes de nivel interior eran los "soldados de a pie", que simplemente utilizaban los programas suministrados con poco entendimiento de cómo funcionaban. Este equipo de trabajo combinó las dos armas más serias del armamento de los intrusos: conocimiento sofisticado de como entrometerse y voluntad de pasar innumerables horas intentando "abrir puertas" para encontrar debilidades.

Uno de los resultados del conocimiento creciente del problema de los intrusos ha sido el establecimiento de una serie de equipos de reacción ante emergencias en computadores (CERT, *Computer Emergenc Response Team).* Estas empresas cooperativas reúnen información sobre los puntos vulnerables de los sistemas y los divulgan a los administradores de sistemas. Por desgracia, los piratas también pueden tener acceso a los informes de los CERT. En el incidente de Texas A&M, un análisis posterior demostró que los piratas habían desarrollado programas para indagar las máquinas atacadas en casi todos los puntos Flacos que los CERT habían anunciado. Si una máquina hubiese fallado en reaccionar rápidamente a una asesoría del CERT, estaba abierta a tales ataques.

Además de ejecutar programas para averiguar contraseñas, los intrusos intentaron modificar el software de conexión para permitirles capturar contraseñas de los usuarios que se conectasen a los sistemas. Esto les capacitó para construir una colección impresionante de contraseñas comprometidas, que estaban disponibles en el tablón de anuncios preparado en una de las propias víctimas.

Se comenzará esta sección echando una ojeada a las técnicas empleadas para la intrusión. Después se examinan formas de prevenir la intrusión. A falta de prevención, la detección de intrusiones es una segunda línea de defensa y se discute en la sección final.

Técnicas de intrusión

El objetivo de los intrusos es obtener acceso a un sistema o aumentar el conjunto de privilegios accesibles en un sistema. En general, esto requiere que el intruso obtenga información que debería estar protegida. En la mayoría de los casos, esta información está en forma de una contraseña de usuario. Si se conocen las contraseñas de algunos usuarios, un intruso podrá conectarse a un sistema y hacer ejercicio de los privilegios convenidos con el usuario legítimo.

Normalmente, un sistema debe mantener un archivo que asocia una contraseña a cada usuario autorizado. Si dicho archivo se almacena sin protección, es un asunto fácil obtener acceso al mismo y conseguir las contraseñas. El archivo de contraseñas puede protegerse de dos maneras:

• *Cifrado unidireccional:* El sistema almacena las contraseñas de los usuarios de forma cifrada solamente. Cuando un usuario presenta una contraseña, el sistema cifra dicha contraseña y la compara con el valor almacenado. En la práctica, el sistema suele realizar una transformación unidireccional (no reversible) en la que la contraseña se emplea en generar una clave para la función de cifrado, produciendo una salida de longitud fija.

• *Control de acceso:* El acceso al archivo de contraseñas está limitado a una o muy pocas cuentas.

Si se toma aluna de estas contramedidas, se requiere cierto esfuerzo para que los posibles intrusos aprendan las contraseñas. A partir de un estudio de la bibliografía y de entrevistas con una serie de averiguadores de contraseñas, [ALVA90] informa de las técnicas siguientes de obtención de contraseñas:

I . Probar las contraseñas por omisión empleadas en las cuentas estándares que se suministran junto al sistema. Muchos administradores no se molestan en cambiar estos valores.

2. Probar exhaustivamente todas las contraseñas cortas (aquellas de uno a tres caracteres).

3. Probar palabras del diccionario del sistema o de una lista de contraseñas probables.

4. Reunir información sobre los usuarios, como sus nombres completos, los nombres de sus esposas e hijos, cuadros de la oficina y libros relativos a pasatiempos y aficiones.

5. Probar los números de teléfono de los usuarios, sus números de DNI y números de habitación.

6. Probar todos los números legales de matrículas del país o del estado.

7. Emplear un caballo de Troya (descrito en la sección 14.4) para saltarse las restricciones de acceso.

8. Intervenir la línea situada entre un usuario remoto y el sistema anfitrión.

Los primeros seis métodos constituyen varias maneras de adivinar una contraseña. Ahora bien, si un intruso tiene que verificar sus conjeturas intentando conectarse, esto constituye un medio de ataque tedioso y fácil de contrarrestar. Por ejemplo, el sistema puede rechazar simplemente cualquier conexión tras intentarse tres contraseñas, requiriendo que el intruso se conecte de nuevo al servidor para intentarlo de nuevo. En estas circunstancias, no es práctico probar más que un puñado de contraseñas. No obstante, no es probable que el intruso emplee métodos tan ordinarios. Por ejemplo, si un intruso puede tener acceso a un archivo de contraseñas cifrado con un nivel bajo de privilegios, la estrategia sería capturar el archivo y utilizar el mecanismo de cifrado del sistema en particular a placer hasta que se descubra una contraseña válida que proporcione privilegios mayores.

Los ataques por adivinación (le contraseñas son factibles y, de hecho, muy efectivos, cuando se puede hacer un gran número de intentos automáticamente y se puede verificar cada uno, sin que el proceso de adivinación sea detectable. Posteriormente, en esta misma sección, habrá mucho que comentar sobre la frustración de los ataques por adivinación.

El séptimo método de ataque de la lista anterior, el caballo de Troya, puede resultar particularmente difícil de contrarrestar. En [ALVA90] se cita un ejemplo de programa que se salta los controles de acceso. Un usuario poco privilegiado generaba un programa de juegos e invitaba al operador del sistema a usarlo en su tiempo libre. El programa era, de hecho, un juego, pero en el fondo también contenía un código para copiar el archivo de contraseñas, que estaba descifrado pero protegido contra determinados accesos, a un archivo del usuario.

Como el juego se ejecutaba en **el modo privilegiado del operador, era capaz de** conseguir acceder al archivo de contraseñas.

El octavo ataque enumerado, la intervención de líneas, es una cuestión de seguridad física. Puede contrarrestarse con técnicas de cifrado, discutidas en la sección 14.6.

Se retoma ahora la discusión sobre las dos contramedidas principales: la prevención y la detección. La prevención ha constituido desde siempre un objetivo de seguridad desafiante y una lucha ardua. La dificultad proviene del hecho de que el defensor debe intentar frustrar todos los ataques posibles, mientras que el atacante tiene la libertad de intentar encontrar la línea más débil de la cadena defensiva y atacar por dicho punto. La detección tiene que ver con el conocimiento de un ataque, tanto antes como después de su triunfo.

Protección de contraseñas

La línea de vanguardia de la defensa ante los intrusos es el sistema de contraseñas. Casi todos los servidores y sistemas multiusuario requieren que el usuario suministre no sólo un nombre o identificador (ID), sino también una contraseña. La contraseña sirve para autentificar el ID del individuo que se conecta al sistema. A su vez, el ID introduce seguridad en dos sentidos:

- El ID determina si el usuario está autorizado para obtener acceso al sistema. En algunos sistemas, sólo se permite acceder a aquellos que ya tienen un ID registrado en el sistema.
- El ID determina los privilegios convenidos con el usuario. Unos pocos usuarios pueden tener un status de supervisor o "superusuario" que les habilita para leer archivos y llevar a cabo funciones protegidas especialmente por el sistema operativo. Algunos sistemas disponen de cuentas anónimas y los usuarios de estas cuentas tienen privilegios más restringidos que los demás.
- El ID se emplea en lo que se conoce como *control de acceso discrecional.* Por ejemplo, enumerando los ID de los demás usuarios, un usuario les puede conceder permisos para leer archivos poseídos por él.

La vulnerabilidad de las contraseñas

Para comprender la naturaleza de los ataques, se va a examinar el esquema tan utilizado de los sistemas UNIX, donde las contraseñas nunca se almacenan en claro. Más bien, se emplea el siguiente procedimiento (figura 14.7x). Cada usuario elige una contraseña de hasta H caracteres de longitud. Ésta es convertida a un valor de 56 bits (empleando ASCII de 7 bits) que sirve como clave de una rutina de cifrado. La rutina de cifrado, conocida como crypt(3). está basarla en el algoritmo Estándar de Cifrado de Datos (DES, Data Encryption Standard: véase el Apéndice 14A). El algoritmo DES se modifica mediante un valor "base" de 12 bits. Normalmente, este valor está relacionado con el momento en que se asigna la contraseña al usuario. El algoritmo DES modificado se ejecuta con una entrada de datos consistente en un bloque de ceros de 64 bits. La salida del algoritmo sirve como entrada para un segundo cifrado. Este proceso se repite por un total de 25 cifrados. La salida de 64 bits resultante se traduce entonces a una secuencia de 11 caracteres. En el archivo de contraseñas se guarda la contraseña cifrada junto a una copia en claro de la base para el ID de usuario correspondiente.

La base sirve para tres fines:

- Impide que las contraseñas duplicadas sean visibles en el archivo de contraseñas. Aun cuando dos usuarios elijan la misma contraseña, dichas contraseñas serán asignadas en momentos diferentes. Entonces, las contraseñas "ampliadas" de los dos usuarios serán diferentes.
- Aumenta de forma efectiva la longitud de las contraseñas sin que haga falta que el usuario recuerde dos caracteres adicionales. Por tanto, el número de contraseñas posibles se incrementa en un factor de 40%, aumentando a su vez la dificultad de adivinar una contraseña.
- Previene el uso de una implementación del DES en hardware, que facilitaría la dificultad de un ataque de adivinación por fuerza bruta.

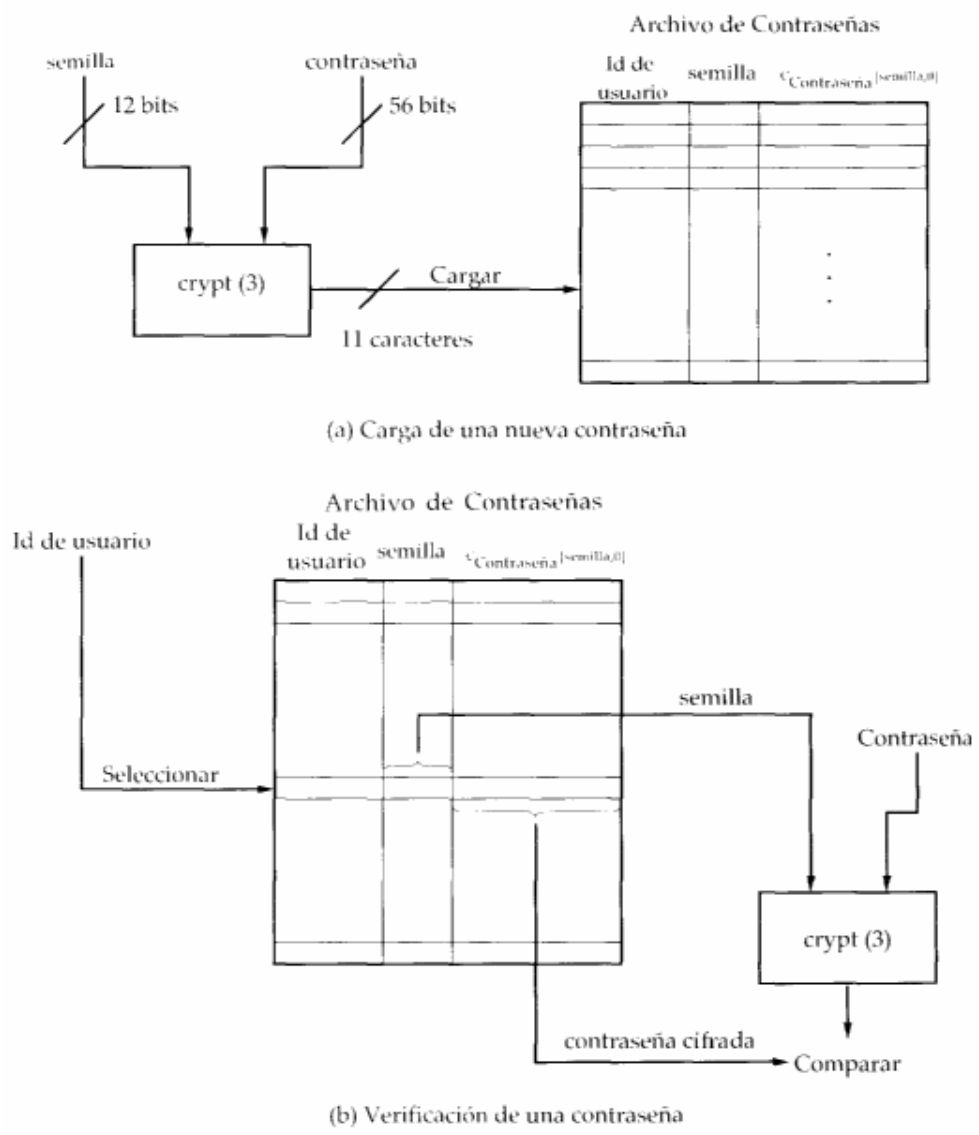


FIGURA 14.7 Esquema de contraseñas de UNIX

Cuando un usuario intenta conectarse a un sistema UNIX, debe proporcionar un ID y una contraseña. El sistema operativo utiliza el ID como índice en el archivo de contraseñas y recupera la base en claro y la contraseña cifrada, que se usa como entrada a la rutina de cifrado. Si el resultado es igual al valor almacenado, la contraseña es aceptada.

La rutina de cifrado está diseñada para rechazar los ataques por adivinación. Las implementaciones por software del DES son lentas en comparación con las versiones en hardware y el empleo de 25 iteraciones multiplica el tiempo necesario por 25. No obstante, a partir del diseño original del algoritmo se han producido dos cambios. En primer lugar, las nuevas implementaciones del algoritmo han dado como resultado una aceleración del mismo. Y, en segundo lugar, el rendimiento del hardware sigue aumentando, de modo que cualquier algoritmo de software se ejecuta más rápidamente.

Así pues, hay dos amenazas al esquema de contraseñas de UNIX. Primero, un usuario puede obtener acceso a una máquina mediante una cuenta de invitado o por otros me

dios y, después, ejecutar un programa de adivinación de contraseñas, denominado averiguador *de contraseñas (password cracker)* en dicha máquina. E1 atacante debe ser capaz de comprobar cientos y, quizá, miles de contraseñas posibles con poco consumo de recursos. Además. si un adversario es capaz de obtener una copia del archivo de contraseñas, se podrá ejecutar un programa averiguador en otra máquina a placer. Esto habilita al adversario para probar muchos centenares de contraseñas posibles en un tiempo razonable.

Como ejemplo, el averiguador de contraseñas más rápido conocido se halló en Internet el] agosto de 1993 [MADS93]. Mediante un computador paralela de la *Thinking Machines Corporation,* se lograba un rendimiento de 1560 cifrados por segundo por unidad vectorial. Cuatro unidades vectoriales por nodo de procesamiento (una configuración estándar), daban como resultado 500.000 cifrados por segundo en una máquina de 128 nodos (que es un tamaño modesto) y 14,4 millones de cifrados por segundo en una de 1024 nodos.

Incluso estas tasas de adivinaciones tan asombrosas aún no hacen que sea factible para un atacante emplear una técnica simple de fuerza bruta que pruebe todas las combinaciones posibles de caracteres para descubrir una contraseña. En su lugar, los averiguadores de contraseñas dependen del hecho de que algunas personas eligen contraseñas fácilmente adivinables.

Por desgracia la naturaleza humana ofrece al atacante una alternativa practica. Algunos usuarios, cuando pueden elegir su propia contraseña, eligen una ridículamente corta. En la tabla 14.2 se muestran los resultados de un estudio de la Universidad de Purdue. El estudio observó las elecciones en el cambio de contraseñas de 54 máquinas, que representaban aproximadamente a 7000 cuentas de usuario. Casi el 3% de las contraseñas eran de tres caracteres de largo o menores. Un atacante podría comenzar su asedio comprobando exhaustivamente todas las contraseñas posibles de longitud menor o igual que 3. Un remedio sencillo es que el sistema rechace cualquier elección de contraseña de menos de, por ejemplo, 6 caracteres o, incluso, exigir que todas las contraseñas tengan exactamente 8 caracteres. La mayoría de los usuarios no se quejarían de tal limitación.

Desgraciadamente, la longitud de las contraseñas sólo es una parte del problema. Mucha dente, cuando se le permite elegir su propia contraseña, toma una adivinable, como su propio nombre, el nombre de su calle, una palabra común del diccionario y otras así. Esto hace que la turca de averiguar contraseñas sea sencilla. El averiguador simplemente tiene que comprobar

el archivo de contraseña con una lista de contraseñas probables. Como muchas personas utilizan contraseñas adivinables, dicha estrategia debería tener éxito en casi todos los sistemas.

TABLA 14.2 Longitud Observada de las Contraseñas [SPAF92]

Longitud	Número	Porcentaje del Total
1	55	0,004
2	87	0,006
3	212	0,02
4	449	0,03
5	1200	0,09
6	3035	0,22
7	2917	0,21
8	5772	0,42
Total	13787	1 ,000

En [KLEI90] se informa de una demostración de la efectividad de la adivinación. A partir de fuentes distintas. Klein reunió archivos de contraseñas de UNIX que contenían casi 14.000 contraseñas cifradas. El resultado, que Klein califica de aterrador con razón, se muestra en la tabla 14.3. En total, casi una cuarta parte de las contraseñas fueron adivinadas. La estrategia que se utilizó fue la siguiente:

1.Probar el nombre del usuario, sus iniciales, el nombre de la cuenta y otra información personal destacable. En total, se probaron 130 permutaciones diferentes para cada usuario.

2. Probar palabras de varios diccionarios. Klein recopiló un diccionario de cerca de 60.000 términos, incluyendo el diccionario en línea del propio sistema y otras listas diversas.

3. Probar varias permutaciones de las palabras del paso 2. Esto incluye transformar la primera letra a mayúsculas o a un carácter de control, transformar la palabra entera a mayúsculas invertir las letras, cambiar la letra "o" por el dígito "0" y cosas así. Estas permutaciones añadieron otro millón de palabras a la lista.

4. Probar varias permutaciones de mayúsculas y minúsculas con las palabras del paso 2 que no se consideraron en el paso3. Esto añadió casi 2 millones de palabras adicionales a la lista.

Así pues, en la comprobación entraban en juego alrededor de 3 millones de palabras. Utilizando la rápida implementación de Thinking Machines comentada anteriormente, el tiempo para cifrar todas estas palabras para todos los posibles valores de la base está por debajo de una hora. Téngase en cuenta que una búsqueda tan completa podía producir una tasa de aciertos cercana al 251/0. mientras que un solo bit puede ser suficiente para obtener un amplio rango de privilegios en un sistema.

Control ríe acceso

Una manera de frustrar un ataque a las contraseñas es denegar al adversario el acceso al archivo de contraseña. Si la parte del archivo con las contraseñas cifradas es accesible sólo para un usuario privilegiado, el adversario no podrá leerlas sin conocer la contraseña del usuario privilegiado. [SPAF92] señala varios defectos en esta estrategia:

• Muchos sistemas, incluyendo a la mayoría de los sistemas UNIX, son susceptibles de robos no previstos. Una vez que un atacante ha conseguido acceder por algún medio, puede querer obtener una colección de contraseñas para utilizar cuentas diferentes en sesiones de co-

TABLA 14.3 Contraseñas Averiguadas de una Muestra de 13.797 Cuentas [KLEI90]

Tipo de contraseña	Tamaño de Búsqueda	Número de Aciertos	Contraseñas Adivinadas (%)	Razón [a] Coste/Beneficio
Nombre de usuario/cuenta	130	368	2,7	2,830
Secuencias de caracteres	866	22	0,2	0,025
Números	427	9	0,1	0,021
Chinos	392	56	0,4	0,143
Nombres de lugares	628	82	0,6	0,131
Nombres habituales	2239	548	4,0	0,245
Nombres de mujer	4280	161	1,2	0,038
Nombres de varón	2866	140	1,0	0,049
Nombres no habituales	4955	130	0,9	0,026
Mitos y leyendas	1246	66	0,5	0,053
Shakespeare	473	11	0,1	0,023
Términos deportivos	238	32	0,2	0,134
Ciencia ficción	691	59	0,4	0,085
Películas y actores	99	12	0,1	0,121
Dibujos animados	92	9	0,1	0,098
Gente famosa	290	55	0,4	0,190
Frases y refranes	933	253	1,8	0,271
Apellidos	33	9	0,1	0,273
Biología	58	1	0,0	0,017
Diccionario del sistema	19683	1027	7,4	0,052
Nombres de máquinas	9018	132	1,0	0,015
Mnemónicos	14	2	0,0	0,143
Biblia del Rey Jaime	7525	83	0,6	0,011
Palabras variadas	3212	54	0,4	0,017
Palabras en judío	56	0	0,0	0,000
Asteroides	2407	19	0,1	0,007
TOTAL	62727	3340	24,2	0,053

[a] Calculado como el número de aciertos dividido por el tamaño de la búsqueda. Cuanto más palabras se necesite comprobar para encontrar un acierto, menor será la razón coste/beneficio.

nexión diferentes y así disminuir el riesgo de detección. También, un usuario con una cuenta puede rogar al usuario de otra que acceda a datos privilegiados o sabotee el sistema.

- Un percance en la protección podría reverter en que el archivo de contraseñas fuese legible, comprometiendo así todas las cuentas.
- Algunos usuarios disponen de cuentas en otras máquinas de otros dominios de protección donde utilizan la misma contraseña.. Por tanto, si las contraseñas pudiesen ser leídas, comprometerían las máquinas de otras ubicaciones.

Así pues, una estrategia más eficaz sería obligar a los usuarios a elegir contraseñas difíciles de adivinar.

Estrategias de elección de contraseñas

La lección a aprender de los dos experimentos descritos en las tablas 14.2 y 14.3 es que, si se deja a los usuarios que se las arreglen por sí solos, muchos elegirían una contraseña demasiado corta o demasiado fácil de adivinar. En el otro extremo, si se les asignan contraseñas a los usuarios que consten de R caracteres imprimibles seleccionados aleatoriamente, la averiguación de las contraseñas es prácticamente imposible. Pero también sería casi imposible que la mayoría de los usuarios recordasen sus contraseñas. Afortunadamente, aunque limitemos el universo de contraseñas a cadenas de caracteres cine se puedan recordar fácilmente, el tamaño del universo sigue siendo demasiado grande como para permitir la averiguación práctica de contraseñas. El objetivo es, entonces, eliminar las contraseñas adivinables a la vez que se permite a los usuarios elegir una contraseña recordable. Para ello se pueden utilizar cuatro técnicas básicas:

- Formación del usuario
- Contraseñas generadas por el computador
- Inspección reactiva de contraseñas
- Inspección proactiva de contraseñas

Se puede comentar a los usuarios la importancia de emplear contraseñas difíciles de adivinar y se les puede dar directrices para elegir contraseñas seguras. La tabla 14.4 ofrece algunas propuestas. Esta estrategia de formación del usuario no es probable que sea fructífera en la mayoría de las instalaciones, en particular cuando hay una gran población ele usuarios o un gran volumen de transacciones. Muchos usuarios harán caso omiso de las directrices. Puede que otros no sepan juzgar bien lo que es una contraseña segura. Por ejemplo, muchos usuarios creen (equivocadamente) que invertir una palabra o poner en mayúsculas la última letra hace que una contraseña no sea adivinable.

Las contraseñas generadas por computador también presentan problemas. Si las contraseñas son de naturaleza bastante aleatoria, los usuarios pueden no ser capaces de recordarlas. Aun cuando la contraseña sea pronunciable, el usuario puede encontrar dificultades a la hora de recordarlas y así caer en la tentación de escribirlas. En general, los esquemas de contraseñas generadas por el computador tienen unos antecedentes de escasa aceptación por parte de los usuarios.

Una estrategia de **inspección reactiva de contraseñas** es aquella en la que el sistema ejecuta periódicamente su propio averiguador de contraseñas para encontrar contraseñas adivinables. El sistema cancela todas las contraseñas que se adivinen y se lo notifica al usuario. Esta táctica presenta una serie de inconvenientes. En primer lugar, si el trabajo se hace correctamente, se consumen muchos recursos. Como un determinado adversario que sea capaz de robar un archivo de contraseñas puede dedicar el tiempo total de CPU a dicha tarea, durante horas o incluso días, un inspector reactivo que sea eficaz está en una situación de desventaja distinta. Es más, cualquier contraseña existente será vulnerable hasta que el inspector reactivo de contraseñas la encuentre.

El enfoque más prometedor para mejorar la seguridad de las contraseñas es una **inspección proactiva de contraseñas.** En este esquema, al usuario se le permite elegir su propia contraseña. Sin embargo, en el momento de la selección, el sistema comprueba si la contraseña es permisible y, si no lo es, la rechaza. Dichos inspectores se basan en la filosofía de chic, con las suficientes directrices del sistema, los usuarios pueden elegir contraseñas recordables de un espacio bastante grande de contraseñas que no es probable que sean adivinadas en un ataque con diccionario.

TABLA 14.4 Estrategias Efectivas para la Selección de Contraseñas Fáciles de Recordar por parte del Usuario [ALVA90]

Verso	Contraseña
Con cien cañones por banda	100cbanda
viento en popa a toda vela	vpvela
no corta el mar sino vuela	ncmvuela
un velero bergantín	1vbergantin

(a) Líneas de unos versos escogidos

Ciudad	Expresión intermedia	Contraseña
Sevilla	Sevilla tiene un color especial	STUCE
París	París bien vale una misa	PBVUM
Granada	Granada, tierra soñada por mí	GTSPM
Valencia	Valencia es la tierra de las flores	VELTDLF

(b) Expresiones inspiradas por nombres de ciudades

Comida	Contraseña
cacahuetes con chocolate	caconchoco
Pepsi-cola con galletas	pepcolgal
caramelos de piña y coco	carpicoco
berenjenas fritas	berenfrit

(c) Comidas que no gustaban en la infancia

Transformación	Expresión ilustrativa	Contraseña
Transliteración	gibraltar	jivraltar
	verongena	berenjena
Entretejer caracteres	canciller, hierro	canchierllerro
de palabras sucesivas	tienda, barra	tienbardara
Traducción	extranjeros	etraniere
Sustitución de letras por dígitos decimales (índice de la letra en orden alfabético mod 10)	repollo	9576226
Sustitución de números decimales por letras (de la posición correspondiente en orden alfabético)	12/10/1492	abaadrb
Desplazamiento desde una posición de partida en un teclado	calabacín	vsñsnsvom
Sustitución por sinónimos	quemarropa	ardevestido
Sustitución por antónimos	malnacido	bienmuerto
Activación de las "mayúsculas" del teclado	6-6-1944	& _ & _ !) $ $
Sustitución por abreviaturas	humedad relativa	humrel
Sustitución por acrónimos	Madres Contra la Droga	mcdonm
	Organización Nacional de Mujeres	
Repeticiones	pan	panpan
Manipulación de imágenes de letras (rotar las letras 180 grados)	cambembo	cawpewpo

(d) Técnicas de transformación

El truco del inspector proactivo de contraseñas es establecer un equilibrio entre la aceptación del usuario y la fortaleza. Si el sistema rechaza demasiadas contraseñas, los usuarios se quejarán de que es muy difícil elegir una contraseña. Si el sistema utiliza un algoritmo sencillo para definir lo que se considera aceptable, se proporciona una guía a los averiguadores de contraseñas para refinar sus técnicas de adivinación. En el resto de este apartado, se examinan los posibles enfoques de inspección proactiva de contraseñas.

El primer método es un sistema simple de aplicación de reglas. Por ejemplo, se pueden hacer valer las siguientes reglas:

- Todas las contraseñas tienen que ser como mínimo de R caracteres.
- Los primeros R caracteres deben incluir, por lo menos, una mayúscula, una minúscula, dígitos numéricos y símbolos de puntuación.

Estas reglas pueden combinarse con avisos al usuario. Aunque este método es mejor que instruir simplemente a los usuarios, puede no ser suficiente para frustrar a los averiguadores de contraseñas. Este esquema alerta a los averiguadores sobre las contraseñas que no deben probar y puede Seguir siendo posible averiguar las contraseñas.

Otro procedimiento posible consiste en recopilar un diccionario grande de posibles contraseñas "malas". Cuando un usuario elige una contraseña, el sistema comprueba que no está en la lista desaprobada. El inconveniente de este método e1 el tiempo y el espacio extra necesarios. Para un estudio de las técnicas de búsqueda eficiente en diccionarios, puede acudirse a [STAL94b].

Detección de intrusos

Inevitablemente hasta los mejores sistemas de prevención de intrusos pueden fallar. Una segunda línea de defensa es la detección de los intrusos, que ha sido objeto de mucha investigación en los últimos años. Este interés viene motivado por una serie de consideraciones, entre las que se incluyen las siguientes:

1. Si se detecta un intruso suficientemente pronto, éste podrá ser identificado y expulsado del sistema antes de que haga algún daño o comprometa determinados datos. Aun cuando la detección no sea suficientemente oportuna en expulsar al intruso, cuanto más pronto se detecte la intrusión, menor cantidad de daños se tendrá y más rápidamente se podrá lograr la recuperación.
2. Un sistema eficaz de detección de intrusos puede servir como medida disuasoria, obrando también como prevención de intrusiones.
3. La detección de intrusos facilita el conjunto de información sobre las técnicas de intrusión que pueden reforzar al servicio de prevención de intrusos.

La detección de intrusos se basa en el supuesto de que el comportamiento del intruso se diferencia del comportamiento del usuario legítimo en formas que pueden cuantificarse. Obviamente no se puede esperar que huya una distinción radical y exacta entre los ataques de los intrusos y el uso normal de los recursos que hace un usuario autorizado. Más bien, se puede esperar que haya cierta coincidencia.

En la figura 14.8 se sugiere, de forma muy abstracta, la naturaleza de la tarea a la que se enfrenta el diseñador de un sistema de detección de intrusos. Aunque el comportamiento típico de un intruso difiere del comportamiento típico de un usuario autorizado, hay cierta coincidencia en dichos comportamientos. Así pues, una interpretación vaga del comporta-

miento de un intruso, que provocará la captura de un mayor número, también llevará consigo una serie de "positivos falsos", es decir, usuarios autorizados que son identificados como intrusos. Por otro lado, el intento de reducir los positivos falsos por una interpretación estricta del comportamiento de los intrusos, llevará a un incremento de los negativos falsos, o intrusos que no son identificados como tales. Así pues, hay una parle de compromiso y de darle en la práctica de la detección de intrusiones.

[PORR92] identifica los siguientes enfoques para la detección de intrusiones:

1. *Detección de anomalías estadísticas:* Este método supone la recolección de datos del comportamiento de los usuarios legítimos durante un periodo de tiempo. Después se aplican pruebas estadísticas al comportamiento observado para determinar, con un alto grado de confianza, si los comportamientos no son de usuarios legítimos.
   a) *Detección de nivel crítico:* Este método supone la definición de umbrales, independientes del usuario, para la frecuencia de aparición de diversos sucesos.
   b) *Detección basada en perfiles:* Se construye un perfil de la actividad de cada usuario y se usa éste para detectar cambios en el comportamiento de cuentas individuales.
2. *Detección basada en reglas:* Este método supone el intento de definir un conjunto de reglas que pueden emplearse para decidir si un comportamiento dado es el de un intruso.
   a) *Detección de anomalías:* Se construyen reglas para detectar desviaciones con respecto a pautas de uso anteriores.
   b) *Identificación de penetraciones:* Un método de un sistema experto que persigue comportamientos sospechosos.

En resumidas cuentas, los métodos estadísticos intentan definir un comportamiento normal o esperado, mientras que los métodos basados en reglas intentan definir un comportamiento correcto.

La detección de anomalías estadísticas es eficaz contra los atacantes exteriores, quienes no es probable que emiten las pautas de comportamiento de las cuentas de las que se adueñan. Por otro lado, dichas técnicas pueden no ser capaces de hacer frente a los usuarios legítimos que intenten obtener acceso a unos recursos que requieran mayores privilegios que los que tienen. Para tales ataques, los métodos basados en reglas pueden ser capaces de reconocer sucesos y secuencias que, en el contexto, revelen una penetración. En la práctica, un sistema puede exponer una combinación de ambos métodos para ser eficaz contra un amplio abanico de ataques.

Una herramienta fundamental para la detección de intrusiones es un registro de auditoría. Debe conservarse algún registro de actividades en curso por parte de los usuarios como entrada al sistema de detección de intrusiones. Básicamente, se utilizan dos planes:

- *Registros de auditoría nativos:* Casi todos los sistemas operativos multiusuario ya incorporan un software de contabilidad que reúne información sobre la actividad de los usuarios. La ventaja de emplear esta información es que no se necesita ningún software adicional. La desventaja es que puede que los registros nativos de auditoría no contengan la información necesaria o que no dispongan de ella en la forma conveniente.
- *Registros de auditoría específicos para la detección:* Se puede implantar un servicio de recopilación que genere registros de auditoría que contengan sólo aquella información que sea necesaria para el sistema de detección de intrusiones. Una ventaja de dicho procedimiento es que

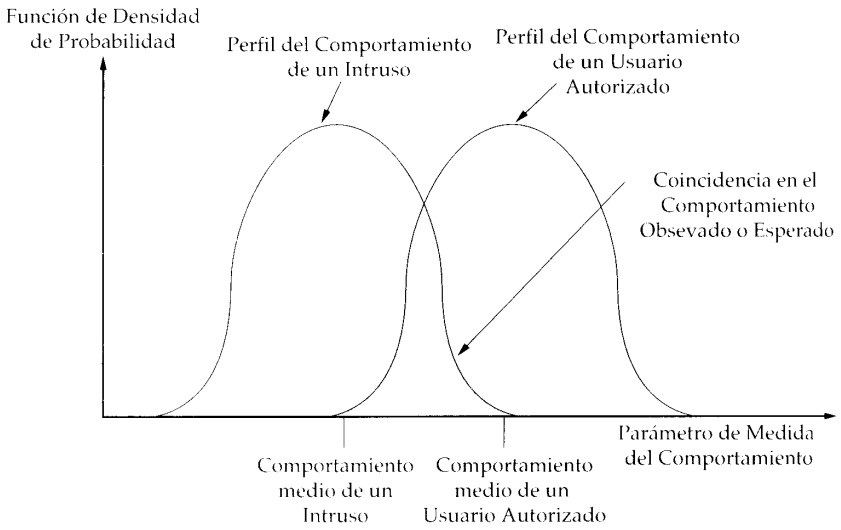


FIGURA 14.8 Perfiles de comportamiento de intrusos y usuarios autorizado

podría ser independiente del fabricante y llevado a diversos sistemas. La desventaja es el coste extra ocasionado por tener, de hecho, dos paquetes de contabilidad ejecutando en una máquina.

Un buen ejemplo de registro de auditoría específico para la detección es el desarrollado por Dorothy Denning [DENN87]. Cada registro de auditoría contiene los campos siguientes (figura 14.9):

• *Sujeto:* Iniciadores de las acciones. Un sujeto es normalmente el usuario de un terminal, pero también podría serlo un proceso que actúa en nombre de un usuario o grupo de usuarios. Toda la actividad surge de órdenes emitidas por los sujetos. Los sujetos pueden agruparse en diferentes clases de acceso, que pueden solaparse.

• *Acción:* Operación realizada por el sujeto con un objeto o sobre un objeto; por ejemplo, conexión, lectura, escritura. E/S. ejecución.

• *Objeto:* Receptores de las acciones. Como ejemplos se tienen los archivos, programas, mensajes, registros, terminales, impresoras y estructuras creadas por usuarios o programas. Cuando un sujeto es el beneficiario de una acción, como puede ser en el correo electrónico, dicho sujeto se considera un objeto. Los objetos pueden agruparse por tipos. La granularidad de los objetos puede variar en función del tipo de objeto y del entorno. Por ejemplo, las acciones de una base de datos pueden auditarse para la base de datos en su totalidad o a nivel de los registros.

• *Condición de excepción:* Indica que condición de excepción, si se produce alguna, hay que elevar a la vuelta.

• *Utilización de recurso:* Una lista de elementos cuantitativos en la que cada elemento dice la cantidad empleada de un recurso (por ejemplo, número de líneas impresas o visualizadas, número de registros leídos o escritos, tiempo del procesador, unidades de E/S empleadas o tiempo transcurrido de la sesión).

• *Muren de tiempo:* Marca única de fecha y hora que identifica cuándo tuvo lugar la acción.

Sujeto	Acción	Objecto	Condición de Excepción	Utilización de Recursos	Marca de Tiempo

FIGURA 14.9 Formato de registro de auditoría específico para la detección

La mayoría de las operaciones de los usuarios se componen de una serie de acciones elementales. Por ejemplo, copiar un archivo implica la ejecución de la orden del usuario, que incluye la validación de acceso y la preparación de la copia, así como la lectura de un archivo y la escritura en otro. Considérese la orden:

COPY JUEGO.EXE TO <Biblioteca>JUEGO.EXE

emitida por López para copiar el archivo ejecutable JUEGO del directorio actual al directorio <Biblioteca>. Se podrían generar los siguientes registros de auditoría:

López	ejecutar	<Biblioteca>COPY.EXE	0	CPU=00002	11058721678
López	leer	<López>JUEGO.EXE	0	REGISTROS=0	11058721679
López	ejecutar	<Biblioteca>COPY.EXE	viol-escr	REGISTROS=0	11058721680

En tal caso, la copia se interrumpe porque López no tiene permiso de Escritura sobre <Biblioteca>.

La descomposición de una operación del usuario en acciones elementales tiene tres ventajas:

1. Como los objetos son las entidades protegibles del sistema, el uso de acciones elementales facilita una auditoría de todos los comportamientos que afecten a un objeto. Así pues, el sistema puede detectar intentos de trastorno de los controles de acceso (apuntando las anormalidades en el número de condiciones de excepción devueltas) y puede detectar trastornos con éxito apuntando las anormalidades en el conjunto de objetos accesibles para el sujeto.
2. Los registros de auditoría de un solo objeto y una sola acción simplifican el modelo y la implementación.
3. Debido a la estructura tan simple y uniforme de los registros de auditoría específicos para la detección, puede resultar relativamente sencillo obtener esta información o, al menos, parte de ella, mediante una simple transformación de los registros nativos de auditoría a registros específicos para la detección.

14.4 VIRUS Y AMENAZAS AFINES

Quizá, los tipos de ataque más sofisticados a los sistemas informáticos son los presentados por programas que se aprovechan de los puntos vulnerables de los mismos. En este contexto. la preocupación la constituyen tanto los programas de aplicación como los de utilidad, como editores y compiladores.

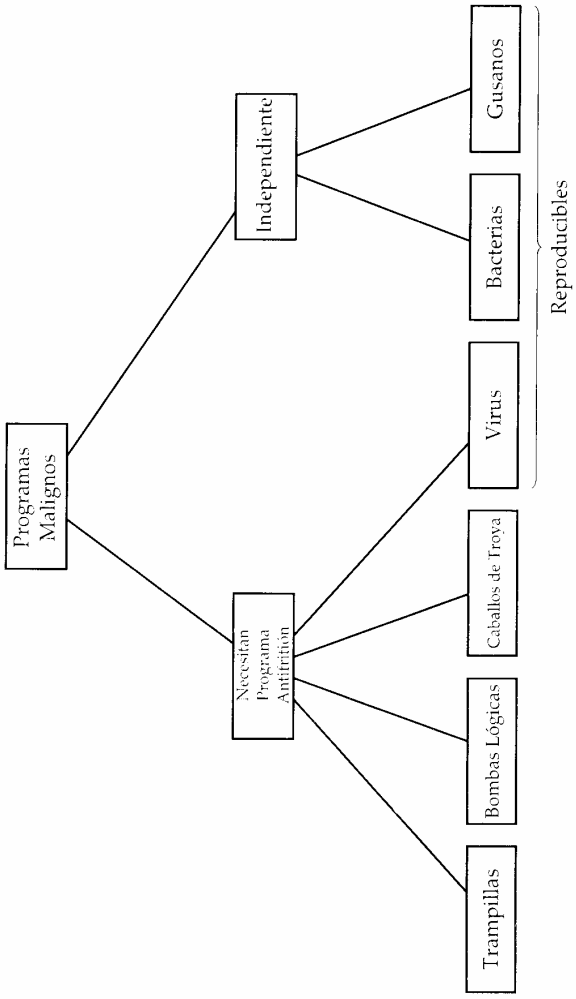


FIGURA 14.10 Taxonomía de programas malignos [BOWL92]

TABLA 14.5 Amenazas a Programas

Bacterias

Programa que consume recursos del sistema reproduciéndose a sí mismo.

Bomba Lógica

Lógica incrustada en un programa de computador que se comprueba cuando se presenta un cierto conjunto de condiciones en el sistema. Cuando se cumplan dichas condiciones, ejecutara alguna función que provee a acciones no autorizadas.

Trampilla

Punto de entrada secreto y no documentado a un programa, empleado para otorgar el acceso tuera de los métodos usuales de autentificación.

Caballo de Troya

Rutina secreta y no documentada incrustada en un programa útil del usuario. La ejecución del programa origina la ejecución de dicha rutina.

Virus

Código incrustado en un programa que hace que se inserte una copia de sí mismo en uno o más programas. Además de la propagación, el virus suele realizar alguna función no deseada.

Gusano

Programa que puede reproducirse y enviar copias de un computador a otra a través de las conexiones de red. A su llegada, el gusano puede activarse y reproducirse para propagarse de nuevo. Además de la propagación, el gusano suele realizar alguna acciones no deseada.

Esta sección comienza con una visión general del posible abanico de tales amenazas por software. El resto de la sección está dedicado a los virus, examinando primero su naturaleza y, después, las medidas para contrarrestarlos.

Programas malignos

La figura 14.10 [BOWL92] proporciona una taxonomía general de las amenazas por software o programas malignos, mientras que la tabla 14.5 ofrece unas definiciones breves. Estas amenazas pueden dividirse en dos categorías: aquellas que necesitan un programa anfitrión y las que son independientes. Las primeras son, básicamente, fragmentos de programas que no tienen existencia independiente de algún programa de aplicación, de utilidad o del sistema. Las segundas son programas autocontenidos que pueden ser planificadas y ejecutadas por el sistema operativo.

Se puede distinguir entre las amenazas por software que no se reproducen y las que lo hacen. Las primeras son fragmentos de programas que se activan cuando se invoca al programa anfitrión para realizar una función determinada. Las segundas consisten en un fragmento de programa (virus) o un programa independiente (gusanos y bacterias) que, cuando se ejecutan, pueden hacer que se active más tarde una o más copias de sí misinos, en el mismo sistema o en algún otro.

Aunque la taxonomía de la figura 14.10 vale para organizar la información que se está discutiendo, no constituye la escena completa. En concreto, las bombas lógicas o los caballos de Troya pueden formar parte de un virus o de un gusano.

Trampillas

Una trampilla es un punto de entrada secreto a un programa que permite a alguien que la conoce conseguir el acceso sin pasar por los procedimientos usuales de seguridad

de acceso. Las trampillas las han usado los programadores de una forma legítima durante muchos años para depurar y probar los programas. La depuración y las pruebas se suelen hacer cuando el programador está desarrollando una aplicación que dispone de un procedimiento de autentificación o una preparación muy larga, que requiere del usuario introducir muchos valores diferentes para ejecutar la aplicación. Para depurar el programa, el desarrollador puede querer disponer de privilegios especiales o evitar toda la preparación y autentificación necesarias. El programador también puede querer asegurarse que hay un método para activar el programa en el caso de que algo vaya mal en el procedimiento de autentificación que se está construyendo en la aplicación. La trampilla es un código que reconoce alguna secuencia de entrada especial o que es lanzado al ser ejecutado por un cierto ID de usuario o mediante una secuencia improbable de sucesos.

Las trampillas se convierten en amena/.as cuando son empleadas por programadores desaprensivos para conseguir el acceso no autorizado. La trampilla era la idea básica de la vulnerabilidad representada en la película "Juegos de Guerra "[COOP89]. En un caso real, los auditores descubrieron una trampilla en un producto de software comercial |GOLD85| en el que el nombre de su autor servía como contraseña de paso. Otro ejemplo: Durante el desarrollo de Multics, las pruebas de penetración estaban dirigidas por un "equipo tigre" de las Fuerzas Aéreas (adversarios simulados). Una táctica empleada fue enviar una versión falsa del sistema operativo a un nodo que ejecutaba Multics. La versión contenía un caballo de Troya (descrito mas tarde) que podía activarse por una trampilla y que permitía que el equipo tigre lograra el acceso. La amenaza estaba tan bien implementada que los desarrolladores de Multics no pudieron encontrarla, incluso tras haber sido informados de su presencia [ENGE80].

Es difícil implementar controles para trampillas en el sistema operativo. Las medidas de seguridad deben centrarse en el desarrollo de programas y en las actualizaciones del software.

Bomba Lógica

Lino de los tipos de amenaza más antiguos, anterior a los virus y gusanos, es la bomba lógica. La bomba lógica es un código incrustado en algún programa legítimo que "explota" cuando se cumplen ciertas condiciones. Ejemplos de condiciones que pueden emplearse como disparadores de una bomba lógica son la presencia o ausencia de ciertos archivos, un día concreto de la semana, o una fecha, o un usuario particular que ejecute la aplicación. En un caso famoso |SPAFS9|, una bomba lógica inspeccionaba el número de ID de un cierto empleado (el del autor de la bomba) y entonces se disparaba si el **ID** no aparecía en dos cálculos consecutivos de la nómina. Lina vez disparada, la bomba podía modificar o borrar datos o archivos enteros, hacer que la máquina se detuviese o causar algún otro daño. Un ejemplo sorprendente de cómo se pueden utilizar las bombas lógicas fue el caso del sistema de la biblioteca del Condado de Montgomery, en Maryland [TIME90]. El contratista que había desarrollado el sistema de préstamos computerizado insertó una bomba lógica que inhabilitaba el sistema en una cierta fecha, a menos que se le hubiera pagado. Cuando la biblioteca negó el pago final porque el sistema tenía un tiempo de respuesta malo, el contratista reveló la existencia de la bomba y amenazó con permitir que estallase a menos que el pago estuviese disponible.

Caballos de Troya

Un caballo de Troya en un programa o procedimiento útil o aparentemente útil que contiene un código oculto que, cuando se invoca, lleva a cabo alguna función dañina o no deseada.

Los programas con caballos de Troya se pueden usar para efectuar funciones indirectamente que un usuario no autorizado no podría efectuar directamente. Por ejemplo, para obtener acceso a los archivos de otro usuario en un sistema compartido, un usuario podría crear un programa con un caballo de Troya que, cuando se ejecutase, cambiara los permisos de los archivos del usuario que lo llamase de forma que pudieran ser leídos por cualquier usuario. El autor del programa podría entonces invitar a ¡os usuarios a ejecutar el programa situándolo en un directorio común y dándole un nombre de forma que pareciese una utilidad provechosa. Un ejemplo es un programa que produce ostensiblemente un listado de los archivos del usuario en un formato deseado. Después de que otro usuario haya ejecutado el programa, su autor podría acceder a la información de los archivos del usuario. Un ejemplo de programa con caballo de Troya que sería difícil de detectar es un compilador que haya sido modificado para que inserte un código adicional en ciertos programas cuando son compilados, como los programas de conexión a los sistemas [THOM84]. El código crea una trampilla en el programa de conexión que le permite al autor conectarse al sistema mediante una contraseña especial. Este caballo de Troya nunca se descubrirá leyendo el código fuente del programa de conexión.

Otra intención habitual de los caballos de Troya es la destrucción de datos. El programa parece estar realizando alguna función de utilidad (por ejemplo, un programa de calculadora), pero también puede estar eliminando silenciosamente los archivos del usuario. Por ejemplo, un ejecutivo de la CBA fue víctima de un caballo de Troya que destruyó toda la información contenida en la memoria de su computador [TIME90]. El caballo de Troya fue implantado en una rutina gráfica ofertada en un sistema de tablón de anuncios electrónico.

Virus

Un virus es un programa que puede "infectar" a otros programas modificándolos; la modificación incluye una copia del programa de virus, que puede entonces seguir infectando a otros programas.

Los virus biológicos son fragmentos minúsculos de código genético (ADN o ARN) que pueden tomar el control de la organización de una célula viva y trucarla para hacer miles de réplicas impecables del virus original. Como sus equivalentes biológicos, un virus informático porta en su código de instrucciones la receta para hacer copias perfectas de sí mismo. Una vez alojado en un computador anfitrión, el típico virus toma el control temporalmente del sistema operativo situado en el disco del computador. Entonces, cuando el computador infectado entre en contacto con un elemento de software no infectado, se pasa una nueva copia del virus al programa. Así pues, la infección puede extenderse de un computador a otro a través de usuarios no sospechosos que intercambian sus discos o bien se envían programas de uno a otro a través de la red. En un entorno en red, la capacidad de acceder a las aplicaciones y los servicios del sistema de otros computadores ofrece un cultivo perfecto para la propagación de un virus.

Los virus se examinan en mayor detalle en esta misma sección.

Gusanos

Los programas de gusanos de la red emplean las conexiones de la red para extenderse de un sistema a otro. Una vez activos en un sistema, un gusano de red puede comportarse como un virus informático o una bacteria (discutidos antes) o bien puede implantar caballos de Troya o realizar cualquier número de acciones trastornadoras o destructoras.

Para reproducirse, un gusano de red utiliza algún tipo de vehículo en la red. Como ejemplos se tienen los siguientes:

- *Servicio de correo electrónico:* El gusano divulga una copia de sí mismo a través del correo a otros sistemas.
- *Posibilidad de ejecución remota:* El gusano ejecuta una copia de sí mismo en otro sistema.
- *Posibilidad de conexión remota:* El gusano se conecta a un sistema remoto como un usuario y después emplea órdenes para copiarse a sí mismo de un sistema a otro.

La nueva copia del programa del gusano se ejecuta después en el sistema remoto donde, además de las funciones que realiza, continúa propagándose de la misma manera.

Un gusano de red muestra las mismas características que un virus informático: una fase latente, una fase de propagación, una fase de activación y una fase de ejecución. La fase de propagación lleva a cabo generalmente las siguientes funciones:

1. Busca otros sistemas a infectar, examinando las tablas de servidores o similares de las direcciones de sistemas remotos.
2. Establece una conexión con el sistema remoto.
3. Se copia a sí mismo al sistema remoto y hace que la copia se ejecute.

El gusano de red también puede intentar determinar si el sistema ha sido infectado previamente, antes de copiarse a sí mismo. En un sistema de multiprogramación, también puede disfrazar su presencia tomando el nombre de un proceso del sistema o usando algún otro nombre que no sea notado por el operador del sistema.

Como con los virus, los gusanos de red son difíciles de contrarrestar. No obstante, tanto las medidas de seguridad en redes como en sistemas sencillos, si se diseñan e implementan correctamente, minimizan la amenaza de los gusanos.

Bacterias

Las bacterias son programas que no dañan explícitamente los archivos. Su único propósito es reproducirse. Un programa bacteria típico puede que no haga más que ejecutar dos copias suyas simultáneamente en un sistema de multiprogramación o, quizás, crear dos archivos nuevos, cada uno de los cuales es una copia del archivo fuente original de la bacteria. Ambos programas pueden entonces copiarse dos veces más y así sucesivamente. Las bacterias se reproducen exponencialmente, agotando finalmente toda la capacidad del procesador, memoria o espacio en disco, privando a los usuarios del acceso a dichos recursos.

La naturaleza de los virus

Un virus puede hacer cualquier cosa que hagan otros programas. La única diferencia es que se engancha a otro programa y se ejecuta de forma oculta cada vez que se ejecuta el programa anfitrión. La tabla 14.6 muestra un ejemplo sencillo de cómo se puede implementar

TABLA 14.6 Rastro de los Virus

Un virus muy sencillo en lenguaje ensamblador que no haga más que infectar programas podría hacer algo como lo siguiente:

- Encontrar la primera instrucción de programa
- Sustituirla por un salto a la posición de memoria siguiente a la última instrucción del programa.
- Insertar una (opia del código del virus en elidía posición.
- Hacer que el virus simule la instrucción sustituida por el salto.
- Saltar de vuelta a la secunda instrucción del programa anfitrión.
- Terminar la ejecución del programa anfitrión.

Cada vez que el programa anfitrión se ejecute, el virus infecta otro programa y después ejecuta el programa anfitrión. Excepto una corta demora, el usuario no nota nada sospechoso.

un virus de modo que se propague. Este ejemplo sólo indica el mecanismo por el cual el virus permanece oculto y mediante el que se propaga. Si esto fuera todo lo que tienen los virus, no serían motivo de preocupación. Por desgracia, una vez, que un virus se ejecuta, puede efectuar cualquier función, como borrar archivos y programas. Esta es la amenaza de los virus. La tabla 14.7 enumera algunos de los virus más conocidos e indica los daños que pueden hacer.

Durante su vida, un virus típico pasa por las siguientes cuatro etapas:

1. Una fase latente, en la que el virus está inactivo. El virus será finalmente activado por algún suceso, como una fecha, la presencia de otro programa o archivo o que la capacidad del disco exceda de cierto límite. No todos los virus pasan por esta etapa.
2. La fase de propagación, durante la cual el virus sitúa una copia idéntica suya en otros programas o en ciertas zonas del sistema en el disco. Cada programa infectado contendrá ahora un clónico del virus, que entrará a su vez en la fase de propagación.
3. La fase de activación, en la que el virus se activa para llevar a cabo la función para la que está propuesto. Como en la fase latente, la fase de activación puede ser causada por una variedad de sucesos del sistema, incluyendo la cuenta del número de veces que esta copia del virus ha hecho copias de sí mismo.
4. La fase de ejecución, en la que se lleva a cabo la función. La función puede ser no dañina, como dar un mensaje por la pantalla, o dañina, como la destrucción de archivos de programas y datos.

La mayoría de los virus llevan a cabo su trabajo de manera específica para un sistema operativo concreto y, en algunos casos, específicamente para una plataforma de hardware en particular. Así pues, están diseñados para sacar partido de los detalles y las debilidades de los sistemas concretos.

Estructura de los virus

Un virus puede añadirse por delante o por detrás de un programa ejecutable o bien puede incrustarse de algún modo. La clave para su funcionamiento es que el programa infectado, cuando se le invoque, ejecute primero el código del virus y, después, el código original del programa.

TABLA 14.7 Algunos Virus Comunes

Virus de los IBM PC

Cerebro pakistaní

Uno de los virus más predominarnos, llamado así porque se originó en Pakistán. Infecta el sector de arranque de los discos con PC -DOS y se reproduce, infectando todos los discos flexibles que se inserten en el sistema. El virus toma el control de la interfaz controladora del disco. Si detecta una operación de Lectura, sitúa a un lado la operación de Lectura original e intenta leer la pista de arranque. Si determina que el arranque no está infectado, lo modifica para que contenga el virus. En algunas versiones, el virus empieza a marcar zonas del disco como defectuosas aunque estén en buenas condiciones. Al final, el disco no contendrá otra cosa que sectores defectuosos.

Virus de Jerusalén

Este virus infecta programas ejecutables .COM o .EXE. Reside en la memoria e infecta todos los programas que se ejecuten. Destruye las tablas de asignación de archivos, lo que hace imposible acceder a los archivos del disco v revuelve los datos del disco. Este virus se propaga por los discos flexibles pero también ataca a los discos duros.

Virus de LeHigh

Este virus infecta al sistema operativo introduciéndose en el procesador de órdenes. Cada vez que se hace un acceso al disco, comprueba si el procesador de comandos de tal disco está infectado. Si no lo está, se introduce el virus. Si está infectado, se incrementa un contador controlado por el virus, cuando el contador llega a cuatro (en versiones más recientes, diez), el virus destruye lodos los datos del disco duro.

Virus Alameda

Este virus infecta el sector de arranque del sistema. Después infecta cualquier disco flexible insertado durante el nuevo arranque y destruye la última pista del disco.

Virus de Macintosh

Virus de cuenta

Este virus está diseñado para reproducirse durante un número específico de días, seguido por varios días de latencia. Tras ello, cuando el usuario intenta guardar información en un archivo, el virus se lo impedirá y hundirá el sistema.

nVIR

Este virus se presenta en varias formas, de las cuales se han detectado, por lo menos, una docena. La técnica de propagación es especialmente virulenta. Invade el archivo del sistema; una vez que este recurso crucial está infectado, todas las aplicaciones que se lanzan a continuación son contaminadas.

Una representación muy general de la estructura de un virus es la de la figura 14.11 (basada en [COHE90]). En este caso, el código V del virus se añade por delante de los programas infectados y se supone que el punto de entrada al programa, cuando éste es invocado, es la primera línea del programa. El programa infectado comienza con el código del virus y funciona como sigue. La primera línea de código es un salto al programa principal del virus. La segunda línea es un marcador especial que es empleado por el virus para determinar si un posible programa víctima está ya infectado con el virus. Cuando se invoca al programa, el control se transfiere inmediatamente al programa principal del virus. El programa del virus busca primero ejecutables no infectados y los infecta. A continuación, el virus puede realizar alguna acción, normalmente en detrimento del sistema. Esta acción podría ser efectuada cada vez que se invocara al programa o

podría ser una bomba lógica que se disparara sólo bajo ciertas condiciones. Por último, el virus transfiere el control al programa original. Si la tase cíe infección del programa es razonablemente rápida, no es probable que un usuario detecte diferencia alguna entre la ejecución de un programa infectado y uno que no lo está.

Un virus como el descrito es fácil de detectar porque la versión infectada del programa es mayor que la correspondiente no infectada. Una forma de frustrar un medio tan sencillo de detectar el virus es comprimir el archivo ejecutable de forma que tanto la versión infectada como la no infectada sean de longitud idéntica. La figura 14.12 [COHE90] muestra en líneas generales la lógica necesaria. Las líneas clave de este virus están numeradas y la figura 14.13 [COHE90] ilustra el funcionamiento. Se supone que el programa P_1 está infectado con el virus CV. Cuando este programa sea invocado, el control pasa a su virus, que efectúa los pasos siguientes:

1. Para cada archivo no infectado P_2 que se encuentre, el virus comprime primero el archivo para generar P_2', reduciendo el programa original en el tamaño del virus.
2. Se añade una copia del virus por delante del programa comprimido.
3. Se descomprime la versión comprimida del programa original infectado, P_1'.
4. Se ejecuta el programa original descomprimido

En este ejemplo, el virus no hace otra cosa que propagarse. Como en el ejemplo anterior,el virus puede incorporar una bomba lógica.

Infección inicial

Una vez que un virus ha tenido acceso a un sistema por la infección de un solo programa, está en posición de infectar algunos o todos los archivos ejecutables del sistema, cuando se ejecute el programa infectado. Así pues, la infección vírica puede ser prevenida por completo impidiendo que el virus entre por primera vez. Por desgracia, la prevención es extremadamente difícil porque un virus puede formar parte de cualquier programa exterior a un sistema. Así pues, a menos que uno se conforme con tomar un pedazo de acero en bruto y escribir su propio sistema y todos los programas de aplicación, se es vulnerable.

Sólo una pequeña parte de las infecciones tienen comienzo a través de conexiones de red. La mayoría de ellas se obtienen de un sistema de tablón de anuncios electrónico. Normalmente, un empleado traerá por la red un juego o una utilidad aparentemente útil para descubrir más tarde que tenía un virus.

Tipos de virus

Desde que los virus aparecieron por vez primera, se ha producido una carrera de armamento entre los escritores de virus y los escritores de software antivirus. A medida que se han desarrollado contramedidas eficaces para los tipos de virus existentes, se han desarrollado nuevos tipos. ISTEP93] propone las siguientes categorías de entre los tipos de virus más significativos:

• *Virus parásitos:* La forma más tradicional y, todavía, más común de virus. Un virus parásito se engancha a archivos ejecutables y se reproduce, al ejecutar el programa infectado. buscando oíros archivos ejecutables que infectar.

• *Virus residentes en memoria:* Se alojan en la memoria principal como parte de un programa del sistema residente. Desde ese momento, el virus infecta todos los programas que se ejecutan.

• *Virus del sector de arranque:* Infecta el sector principal de arranque o el sector de arranque y se propaga cuando el sistema se arranca desde el disco que contiene el virus.

```
        program V :=
{goto main;
        1234567;
        subroutine infect-executable :=
          {loop:
          file := get-random-executable-file;
          if (first-line-of-file = 12345467)
            then goto loop
            else prepend V to file; }
        subroutine do-damage :=
          {whatever damage is to be done}
        subroutine trigger-pulled :=
            {return true if some condition holds}
main: main-program :=
         {infect-executable;
         if trigger-pulled then do-damage;
         goto next;}
next:
}
```

FIGURA 14.11 Un virus simple (basado en [COHE90])[2]

• *Virus clandestino:* Una forma de virus diseñado explícitamente para esconderse de la detección por software antivirus.
• *Virus polimorfo:* Un virus que muta con cada infección, haciendo que la detección por la "firma" del virus sea imposible.

```
        program CV :=
{goto main;
        01234567;
        subroutine infect-executable :=
                {loop:
                        file := get-random-executable-file;
                if (first-line-of-file = 012345467) then goto loop;
              (1) compress file;
              (2) prepend CV to file;
              }
main: main-program :=
                 {if ask-permission then infect-executable;
            (3) uncompress rest-of-file;
            (4) run uncompressed file;}
            }
```

FIGURA 14.12 Lógica de un virus con compresión [COHE90][2]

• Se conservan estas dos rutinas en ingles, por respetar el espíritu del autor. al ser citas bibliográficas. Véase referencia bibliográfica COHFE90 (N. del T.)

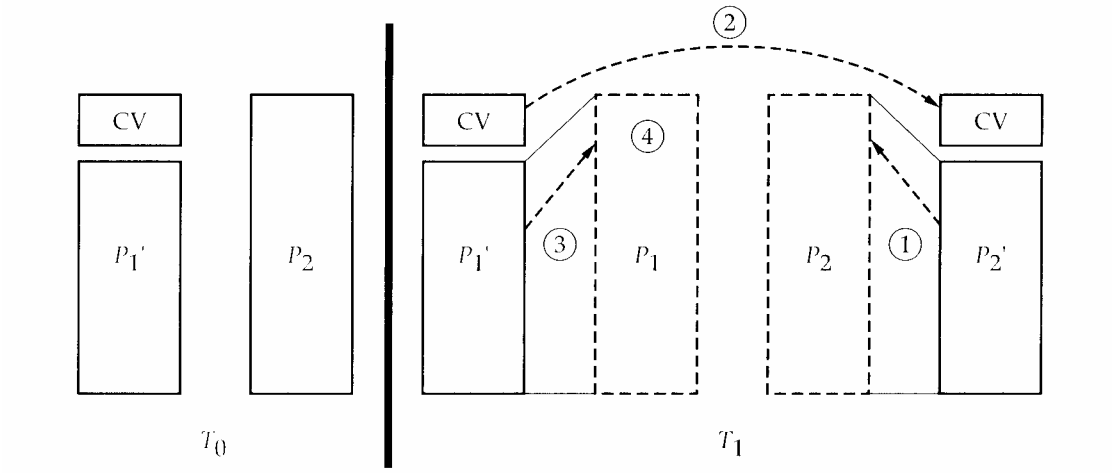


FIGURA 14.13 Un virus con compresión [COHE90]

Un ejemplo de **virus clandestino** es el discutido antes: un virus que utiliza compresión para que el programa infectado tenga exactamente la misma longitud que una versión no infectada. Son posibles técnicas mucho más sofisticadas. Por ejemplo, un virus puede poner alguna lógica de intercepción en las rutinas de E/S con el disco, de modo que cuando haya un intento de leer partes sospechosas del disco con estas rutinas, el virus presente el programa original no infectado. Así pues, el termino clandestino no se aplica a los virus como tales sino, mas bien, es una técnica empleada por los virus para evitar su detección.

Un **virus polimorfo** crea copias durante la reproducción que son funcionalmente equivalentes pero que tienen diferentes patrones de bits. Como con los virus clandestinos, la finalidad es vencer a los programas que buscan virus. En tal caso, la "firma" del virus varía con cada copia. Para lograr esta variación, el virus puede insertar aleatoriamente instrucciones supérfluas o intercambiar el orden de las instrucciones independientes. Un método más eficaz, es usar técnicas de cifrado. Una parte del virus, generalmente llamada *motor de mutación,* crea una clave de cifrado aleatoria para cifrar el resto del virus. Dicha clave es almacenada junto con el virus y el motor de mutación es modificado. Cuando se invoca a un programa infectado, el virus utiliza la clave aleatoria almacenada para descifrar el virus. Cuando el virus se reproduce, se escoge una clave aleatoria diferente.

Otra arma del armamento de los escritores de virus es un juego de utilidades para la creación de virus. Dicho juego permite que un novato cree rápidamente una serie de virus diferentes. Aunque los virus creados con estas utilidades tienden a ser menos sofisticados que los virus diseñados desde cero, el número absoluto de nuevos virus que pueden generarse crea un problema para los procedimientos antivirus.

Otra herramienta más del escritor de virus es el tablón de anuncios para intercambio de virus. Una serie de tablones de este estilo han surgido [ADAM92] en los Estados Unidos y otros países. Estos tablones ofrecen copias de virus que pueden traerse por la red, así como consejos para la creación de virus.

Métodos antivirus

La solución ideal para la amenaza de los virus es la prevención: En primer lugar, no permitir que los virus entren en el sistema. Esta meta es, en general, imposible de alcanzar, aunque la prevención puede reducir el número de ataques víricos fructuosos. El siguiente mejor método es ser capaz de hacer lo siguiente:

• *Detección:* Una vez que se ha producido la infección, determinar que ha tenido lugar y localizar el virus.

• *Identificación:* Una vez que se ha logrado la detección, identificar el virus específico que ha infectado un programa. Eliminar el virus de todos los sistemas infectados, de forma que la plaga no pueda extenderse más.

• *Eliminación:* Una vez que se ha identificado el virus específico, eliminar todo rastro del virus del programa infectado y reponerlo a su estado original.

Si la detección tiene éxito, pero la identificación o la eliminación no son posibles, la alternativa es descartar el programa infectado y cargar una copia limpia de reserva.

Los avances en la tecnología de virus y antivirus van de la mano. Los primeros virus eran trozos de código relativamente simple y podían identificarse y liquidarse con paquetes antivirus relativamente sencillos. A medida que la carrera de armamentos de los virus ha avanzado, tanto los virus como, necesariamente, los antivirus han crecido en complejidad y sastisfacción.

[STEP93] identifica cuatro generaciones de software antivirus:

• *Primera generación:* rastreadores simples

• *Secunda generación:* rastreadores heurísticos

• *Tercera generación:* trampas de actividad

• *Cuarta generación:* protección completa

Un rastreador de **primera generación** requiere una firma del virus para identificarlo. El virus puede contener "comodines", pero tiene básicamente la misma estructura y patrón de bits en todas las copias. Dichos rastreadores de firmas específicas están limitados a la detección de virus conocidos. Otro tipo de rastreadores de primera generación mantienen un registro de la longitud de los programas y buscan cambios de longitud.

Un rastreador de **segunda generación** no depende de una firma específica. Más bien, el rastreador emplea reglas heurísticas para buscar infecciones probables por virus. Un tipo de tales rastreadores buscan trozos de código que suelen estar asociados con virus. Por ejemplo, un rastreador puede buscar el comienzo de un bucle de cifrado empleado en un virus polimorfo y descubrir la clave de cifrado. Una vez. que se ha descubierto la clave, el rastreador puede descifrar el virus para identificarlo, eliminar entonces la infección y volver a poner el programa en servicio.

Otro método de segunda generación es la prueba de integridad. Se puede añadir un código de prueba *(cliecksum)* a cada programa. Si un virus infecta el programa sin cambiar el código, una prueba de integridad detectará el cambio. Para contrarrestar un virus suficientemente sofisticado, como para cambiar el código de prueba cuando infecta un programa, se puede emplear una función de dispersión *(hash}* cifrada. La clave de cifrado se almacena por separado del programa, de forma que el virus no pueda generar un nuevo código de dispersión y cifrarlo. Utilizando una función de dispersión en vez de un sencillo código de prueba, se impide que el virus prepare el programa para producir el mismo código de dispersión que antes.

Los programas de **tercera generación** son programas residentes en memoria que identifican un virus por sus acciones más que por la estructura de un programa infectado. Dichos programas tienen la ventaja de que no hace talla construir firmas y heurísticas para una amplia muestra de virus. Más bien, sólo es necesario identificar el pequeño conjunto de acciones que indican que se está intentando una infección y, en tal caso, intervenir.

Los productos de **cuarta generación** son paquetes que constan de una variedad de técnicas antivirus utilizadas en conjunto. Entre estos se incluyen componentes de rastreo y trampas de actividad. Además, dichos paquetes incluyen posibilidades de control de acceso, que limitan la capacidad de los virus para penetrar en un sistema y, por tanto, limitan la capacidad de los virus para actualizar los archivos y contagiar la infección.

La carrera de armamentos continúa. En los paquetes de cuarta generación se emplea una estrategia de defensa más completa, ampliando el alcance de la defensa a más medidas generales de seguridad en los computadores.

14.5 SISTEMAS DE CONFIANZA

Gran parte de lo que se ha discutido hasta ahora tenía que ver con la protección de un mensaje o un elemento dado de un ataque pasivo o activo por parte de un usuario dado. Un requisito algo diferente pero muy aplicable es proteger los datos o los recursos en función de niveles de seguridad. Es habitual encontrar esto en todo lo militar, donde la información se clasifica en: sin clasificar (N), confidencial (C), secreto (S), alto secreto (AS) o aún más. Este concepto es igualmente aplicable a otros campos, donde puede organizarse la información en categorías brutas, y los usuarios puede recibir autorización para acceder a ciertas categorías de datos. Por ejemplo, el nivel mayor de seguridad podría ser para dalos y documentos de planificación corporativa estratégica, accesible sólo para los oficiales corporativos y sus equipos; el siguiente podría ser para datos delicados, financieros y de personal, accesibles sólo para el personal de administración, los oficiales corporativos, etc.

Cuando se definen varias categorías o niveles de datos, el requisito se conoce como seguridad multinivel. El enunciado general del requisito de seguridad multinivel es que un sujeto de un nivel superior no puede transmitir información a un sujeto de un nivel inferior o no comparable, a menos que el flujo de información refleje exactamente el deseo de un usuario autorizado. Con fines de implementación, este requisito se divide en dos y se enuncia de forma sencilla. Un sistema seguro multinivel debe cumplir lo siguiente:

• *No leer arriba:* Un sujeto puede leer un objeto sólo de un nivel de seguridad menor o igual. Esto se conoce en la bibliografía como **propiedad de seguridad simple.**

• *No escribir abajo:* Un sujeto puede escribir un objeto sólo a un nivel de seguridad mayor o igual. Esto se conoce en la literatura como la **propiedad-*** (se pronuncia *propiedad estrella)*[3].

[3] El asterisco no significa nada en particular. Nadie podía pensar en un nombre apropiado para la propiedad durante la escritura del primer informe del modelo. El asterisco fue un carácter artificioso tecleado en el borrador de forma que un editor de texto pudiera encontrar v reemplazar rápidamente Iodos los casos de uso una vez que se diera nombre a la propiedad. No se ideo ningún nombre, así que el informe fue publicado con el asterisco hilado.

Estas dos reglas, si se aplican correctamente, proporcionan una seguridad multinivel. Para un sistema de proceso de datos, el método que se ha adoptado y que ha sido objeto de mucha investigación y desarrollo se basa en el concepto de *monitor de referencia.* Este método se representa en la figura 14.14. El monitor de referencia es un elemento de control en el hardware y el sistema operativo de un computador: regula el acceso de los sujetos a los objetos en función de unos parámetros de seguridad del sujeto y del objeto. El monitor de referencia tiene acceso a un archivo conocido como *base de datos de seguridad del núcleo,* que enumera los privilegios de acceso (credenciales de seguridad) de cada sujeto y los atributos de protección (nivel de clasificación) de cada objeto. El monitor de referencia hace valer las reglas de seguridad (no leer arriba, no escribir abajo) y tiene las propiedades siguientes:

• *Mediación completa:* Las reglas de seguridad se aplican en todos los accesos, no solamente, por ejemplo, cuando se abre un archivo.
• *Aislamiento:* El monitor de referencia y la base de datos están protegidos de modificaciones no autorizadas.
• *Verificabilidad:* La corrección del monitor de referencia debe ser comprobable. Es decir, tiene que ser posible demostrar matemáticamente que el monitor de referencia aplica las reglas de seguridad y proporciona aislamiento y mediación completa.

Estos requisitos son estrictos. El requisito de mediación completa quiere decir que todos los accesos a los datos en memoria principal y en disco o cinta deben ser mediados. Las implementaciones puras con software imponen una penalización al rendimiento demasiado grande para que sean prácticas: la solución debe estar al menos parcialmente en el hardware. El requisito de aislamiento quiere decir que no debe ser posible que un atacante, sin importar lo astuto que sea, cambie la lógica del monitor de referencia de seguridad o el contenido de la base de datos de seguridad del núcleo. Por último, el requisito de prueba matemática es formidable para algo tan complejo como un computador de propósito general. Un sistema que pueda ofrecer tal verificación se conoce como **sistema** de **confianza.**

Un elemento final ilustrado en la figura 14.14 es el archivo de auditoría. Los sucesos importantes de seguridad, como las violaciones de seguridad que se detecten y los cambios autorizados a la base de datos de seguridad del núcleo, se guardan en el archivo de auditoría.

En un esfuerzo para satisfacer sus propias necesidades y como servicio para el público, el Departamento de Defensa de los EE.UU. fundaron en 1981 el Centro de Seguridad de Computadores dentro de la Agencia de Seguridad Nacional (NSA, *Nationat Scuirity Agency),* con el objetivo de promover una amplia disponibilidad de los sistemas informáticos de confianza. Este objetivo se lleva a cabo mediante el Programa de Evaluación de Productos Comerciales del centro. Básicamente, el centro intenta evaluar si los productos disponibles comercialmente cumplen los requisitos de seguridad recién descritos. El centro clasifica los productos evaluados de acuerdo con un rango de características de seguridad que proporcionan. Estas evaluaciones son necesarias para el aprovisionamiento del Departamento de Defensa, pero están publicadas y disponibles de forma gratuita. Por tanto, pueden servir como una directriz a los clientes comerciales para la compra de equipos disponibles comeré i al mente.

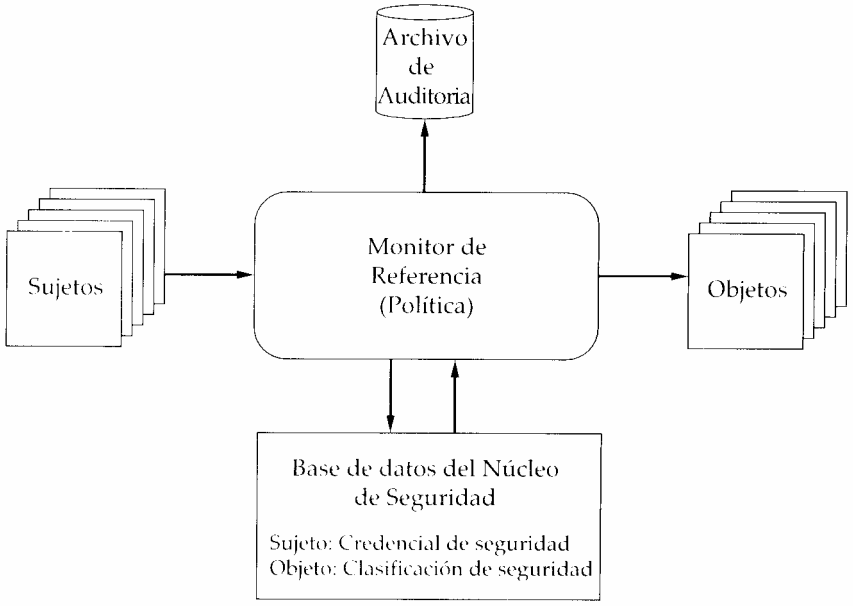


FIGURA 14.14 Concepto de monitor de referencia

Defensa contra caballos de Troya

Una forma de asegurarse contra los ataques de los caballos de Troya es el empleo de un sistema operativo seguro y de confianza. La figura 14.15 ilustra un ejemplo tomado de [BOEB85]. En este caso, se utiliza un caballo de Troya para saltarse el mecanismo estándar de seguridad utilizado por la mayoría de los sistemas operativos y de gestión de archivos: la lista de control de acceso. En este ejemplo, un usuario llamado David interactúa mediante un programa con un archivo de datos que contiene la cadena de caracteres críticamente susceptible "CPE1704TKS". El usuario David ha creado el archivo con permisos de lectura/escritura, ofreciéndoselo sólo a los programas que ejecutan de su parte, es decir, sólo los procesos poseídos por David pueden acceder al archivo.

El ataque del caballo de Troya comienza con un usuario hostil, llamado Drake, que consigue un acceso legítimo al sistema e instala un programa con caballo de Troya y un archivo privado para usarlo en el ataque como "bolsillo trasero". Drake se concede a sí mismo permiso de lectura/escritura para este archivo y le da a David permiso sólo de lectura (figura 14.15a). Drake induce ahora a David a invocar el programa con el caballo de Troya, quizá publicándolo como una utilidad provechosa. Cuando el programa detecta que David lo está ejecutando, copia la cadena de caracteres susceptible del archivo de David al archivo del bolsillo trasero de Drake (figura 14.15b). Las operaciones de Lectura y Escritura satisfacen las restricciones impuestas por las listas de control de acceso. Drake tiene entonces que acceder a su archivo en un momento posterior para conocer el valor de la cadena.

Considérese ahora el uso de un sistema operativo seguro en esta situación (figura 14.15c). Se asignan niveles de seguridad a los sujetos en la conexión, en función de criterios como el terminal desde el que se accede al computador y el usuario involucrado, identificado por su contraseña/lD. En este ejemplo, hay dos niveles de seguridad, específico y público, ordena-

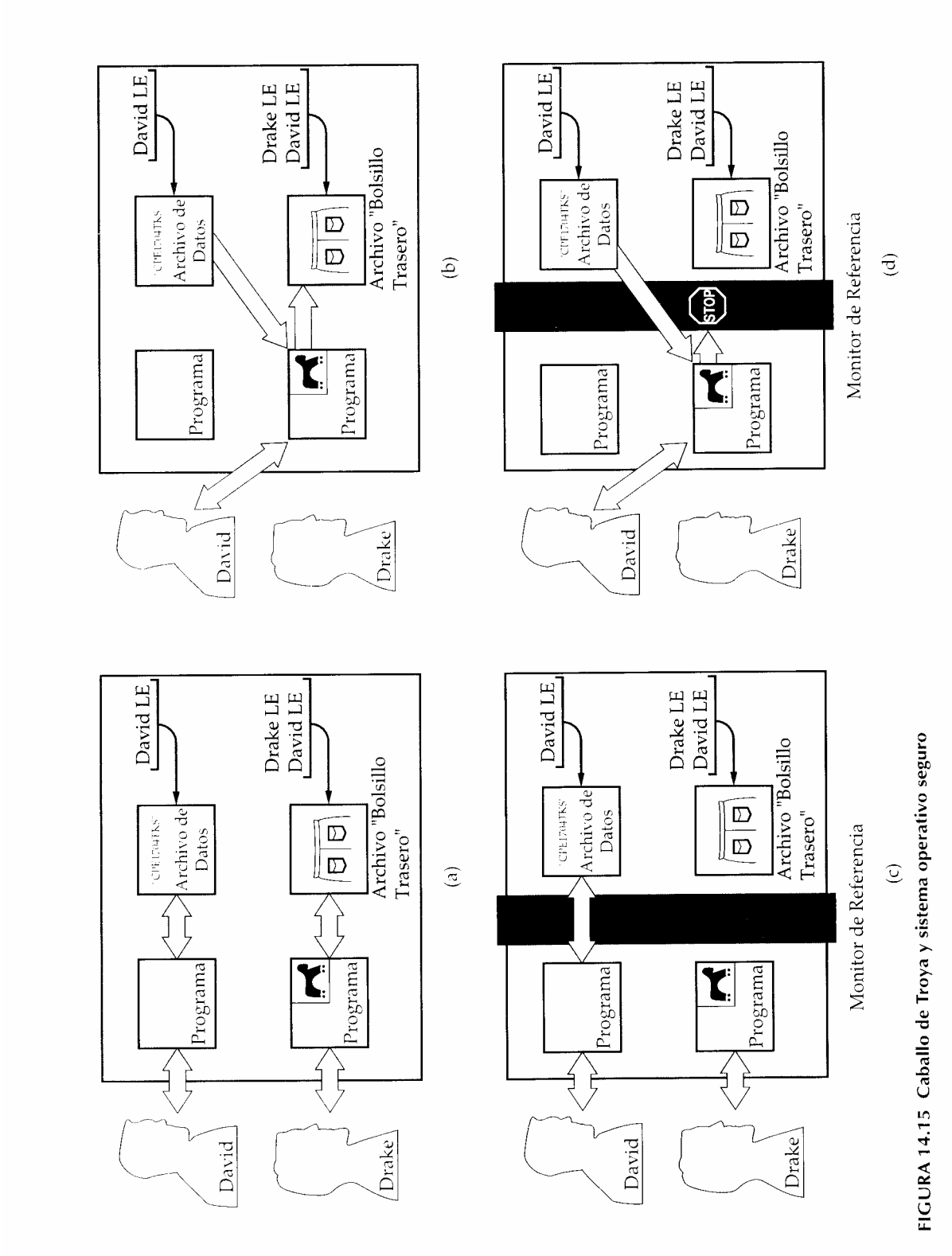


FIGURA 14.15 Caballo de Troya y sistema operativo seguro

dos de forma que específico es mayor que público. Los procesos poseídos por David y los archivos de datos de David tienen asignado el nivel de seguridad "específico". El archivo de Drake y sus procesos están restringidos a "público". Ahora, si David invoca al programa caballo de Troya (figura 14.15d), dicho programa adquiere el nivel de seguridad de David. Por tanto, es capa/, según la Propiedad de Seguridad Simple, de observar la cadena de caracteres específica. Cuando el programa intenta almacenar la cadena en un archivo público (el bolsillo trasero), se viola la propiedad-* y el monitor de referencia no permite el intento. Así pues, el intento de escribir en el archivo del bolsillo trasero es denegado aun cuando la lista de control de acceso lo permita: La política de seguridad tiene prioridad sobre el mecanismo de la lista de control de acceso.

14.6 SEGURIDAD EN REDES

La seguridad en las redes presenta una multitud de problemas nuevos, no encontrados en las implementaciones de sistemas sencillos. En esta sección, se examina la esencia de los problemas y algunos de los métodos disponibles. Estos métodos pueden implementarse como parte de un sistema operativo distribuido o como software de utilidad adicional apoyado en el sistema operativo distribuido.

Posibles posiciones de ataque a la seguridad

La figura 14.16 ilustra la variedad de lugares en que puede ocurrir un ataque, dando un ejemplo de la ruta de las comunicaciones que podría seguirse entre un terminal y un host. La información introducida en el terminal por el usuario debe pasar por un enlace de comunicación hasta un controlador de grupos. Desde allí entra a una red de conmutación de paquetes mediante un enlace de comunicación desde el controlador hasta uno de los nodos de la red. Dentro de la red, la información pasa por una serie de nodos y enlaces hasta que llega al nodo al que está conectado el host de servicio de destino. En este caso, el host no está directamente conectado a la red. En su lugar, la información pasa primero por un procesador frontal *(front-end processor}* y, después, a través de un enlace, hasta el host.

El ataque puede tener lugar en cualquiera de los enlaces de comunicación. Para un ataque activo, el atacante tiene que disponer de control físico sobre una parte del enlace y ser capaz de insertar y capturar las transmisiones. Para un ataque pasivo, el atacante sólo tiene que observar las transmisiones interviniendo la línea de comunicaciones.

Además de la vulnerabilidad potencial de los diferentes enlaces de comunicación, los procesadores de la ruta también están sujetos al ataque. Un ataque puede tomar la forma de intentos de modificar el hardware o el software, obtener acceso a la memoria del procesador o supervisar las ondas electromagnéticas. Estos ataques son menos probables que aquellos que comprometen a los enlaces de comunicación, pero son, a pesar de todo, una fuente de riesgo.

Así pues, hay un gran número de posiciones en las que puede ocurrir un ataque. Además, en comunicaciones de área extensa, muchas de estas ubicaciones no están bajo el control físico del usuario final. Aun en el caso de las redes de área local, en las que son posibles medidas de seguridad física, siempre existe la amenaza del empleado disgustado.

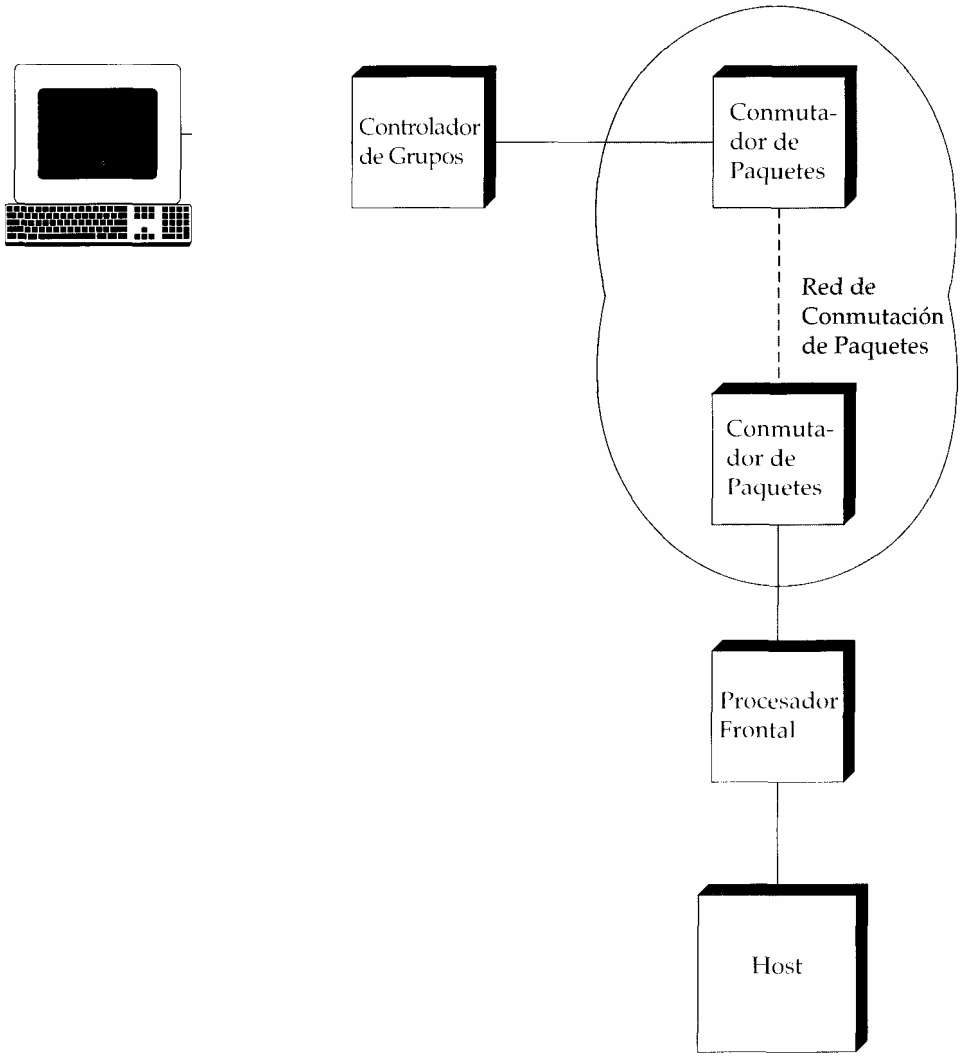
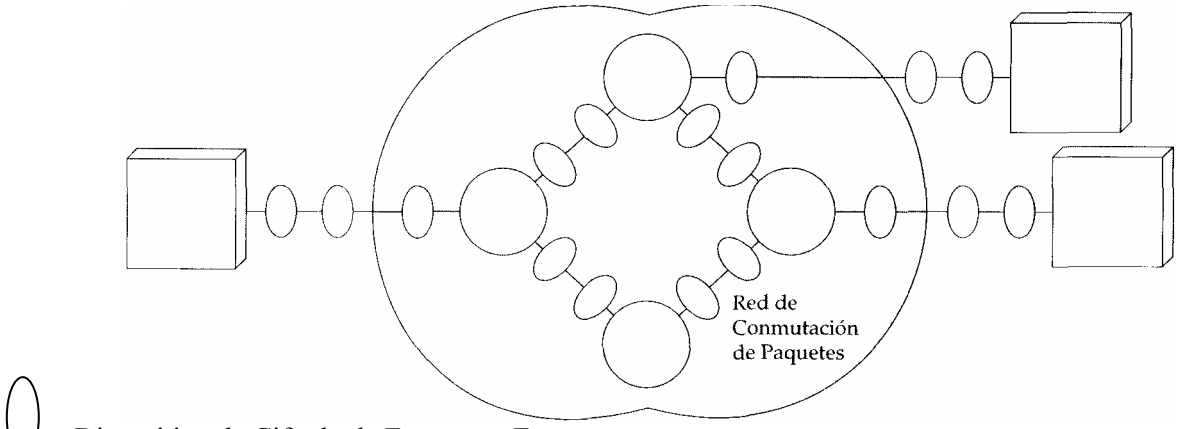


FIGURA 14.16 Ruta física usual

Ubicación de dispositivos de cifrado

El método más potente y habitual cíe contrarrestar las amenazas, realzadas en la discusión de la figura 14.16, es el cifrado. Si se va a usar cifrado para contrarrestar estas amena/as, hace falta decidir qué cifrar y dónde debe ubicarse el mecanismo de cifrado. Como señala la figura 14.17, hay dos alternativas fundamentales: cifrado de enlaces y cifrado de extremo a extremo.

Con el cifrado de enlaces, cada enlace de comunicaciones vulnerable se equipa en ambos extremos con dispositivos de cifrado. Así pues, se asegura todo el tráfico de todos los enlaces de comunicaciones. Aunque esto requiere muchos dispositivos de cifrado en una red grande, el valor de este método es evidente. Lina desventaja del procedimiento es que los



= Dispositivo de Cifrado de Extremo a Extremo
= Dispositivo de Cifrado de Enlaces

FIGURA 14.17 Cifrado en una red de conmutación de paquetes

mensajes deben ser descifrados cada vez que entran en un conmutador de paquetes: el descifrado hace falta porque el conmutador debe leer la dirección (número de circuito virtual) de la cabecera del paquete para encaminarlo. Así pues, el mensaje es vulnerable en cada conmutación. Si es una red pública de conmutación de paquetes, el usuario no tiene control sobre la seguridad de los nodos.

En el cifrado de extremo a extremo, el proceso de cifrado se efectúa en los dos sistemas finales. El host de origen o el terminal cifra los datos. Los datos cifrados son entonces transmitidos sin modificación por la red hacia el terminal o el host de destino. El destino comparte una clave con el origen y así es capa/ de descifrar los datos. Este método parece asegurar la transmisión contra los ataques en los enlaces o nodos de la red. Sin embargo, aún hay un punto débil.

Considérese la siguiente situación. Un host se conecta a una red de conmutación de paquetes X.25, prepara un circuito virtual con otro host y se dispone a transferir datos al otro host mediante cifrado de extremo a extremo. Los datos se transmiten por una red en forma de paquetes que constan de una cabecera y algunos datos del usuario. ¿.Qué parte de cada paquete cifrara el host? Supóngase que el host cifra lodo el paquete, incluyendo la cabecera. Esta táctica no funcionará porque, recuérdese, sólo el otro host puede realizare! descifrado. El nodo de conmutación de paquetes recibirá un paquete cifrado y será incapaz de leer la cabecera. Por tanto, no será capaz de encaminar el paquete. Se deduce que el host puede cifrar sólo aquella parte del paquete que contenga los datos del usuario y debe dejar la cabecera en claro para que pueda ser leída por la red.

Así pues, con el cifrado de extremo a extremo, los datos de usuario están seguros. Sin embargo, la composición del tráfico no lo es porque las cabeceras de los paquetes se transmiten en claro. Para lograr una seguridad mayor, se necesitan tanto el cifrado de enlaces como el cifrado de extremo a extremo, como se muestra en la figura 14.17.

La figura 14.18 ilustra los efectos en conjunto y por separado de las dos formas de cifrado. Cuando se emplean ambas formas, el host cifra los datos de usuario mediante una clave de cifrado de extremo a extremo. El paquete entero es cifrado mediante una clave de cifrado de enlace. A medida que el paquete recorre la red, cada conmutador descifra el paquete con la clave de cifrado de enlace, para leer la cabecera y, después, cifra de nuevo el paquete entero para enviarlo al siguiente enlace. Ahora, el paquete está seguro, excepto durante el tiempo que está en la memoria de un conmutador de paquetes, momento en que la cabecera del paquete está en claro.

Distribución de la clave

Para que el cifrado convencional funcione, las dos partes de un intercambio deben disponer de la misma clave y ésta debe estar protegida del acceso de otras partes. Más aún, los cambios frecuentes de la clave son convenientes para limitar la cantidad de dalos comprometidos si un atacante aprende la clave. Por tanto, la solidez de cualquier sistema de cifrado descansa en la *técnica de distribución de la clave,* un término que se refiere al medio de entrega de la clave a las dos partes que desean intercambiar datos, sin permitir que otros vean la clave. La distribución de la clave puede lograrse de una serie de formas. Para dos partes A y B:

1. A puede elegir una clave y entregarla físicamente a B.
2. Una tercera parte puede elegir la clave y entregarla físicamente a A y B.
3. Si A y B han empleado hace poco una clave, una parte puede transmitirle a la otra la nueva clave, que puede cifrarse usando la clave antigua.
4. Si tanto A como B tienen una conexión cifrada con una tercera parte **C, C** puede entregar una clave por los enlaces cifrados con A y B.

Las opciones 1 y 2 exigen la entrega manual de la clave. Para el cifrado de enlaces, este requisito es razonable porque cada dispositivo de cifrado del enlace va a intercambiar datos con su propia pareja del otro extremo del enlace. Sin embargo, para el cifrado de extremo a extremo, la entrega manual es delicada. En un sistema distribuido, cualquier host o terminal dado puede tener que dedicarse al intercambio con muchos otros hosts y terminales a lo largo del tiempo. Así pues, cada dispositivo necesita una serie de claves, suministradas dinámicamente. El problema es especialmente difícil en un sistema distribuido de área extensa.

La opción 3 es una posibilidad tanto para el cifrado de enlaces como para el cifrado de extremo a extremo, pero si un atacante tiene éxito alguna ve/ y consigue acceder a una de las claves, todas las claves siguientes serán reveladas. Aun cuando se hagan cambios frecuentes en las claves de cifrado del enlace, los cambios deben hacerse manualmente. Para obtener claves de cifrado de extremo a extremo, es preferible la opción 4.

La figura 14.19 ilustra una implementación que satisface la opción 4 para un cifrado de extremo a extremo. En la figura, se ignora el cifrado de enlaces, pero se puede añadir o no según convenga. Para este esquema, se identifican dos tipos de clave:

- *Clave de sesión:* Cuando dos sistemas finales (hosts, terminales, etc.) quieren comunicarse, establecen una conexión lógica (por ejemplo, un circuito virtual). Por la duración de dicha conexión lógica, todos los datos del usuario se cifran con una clave de sesión de un sólo uso. Al concluir la sesión o la conexión, la clave de sesión es destruida.

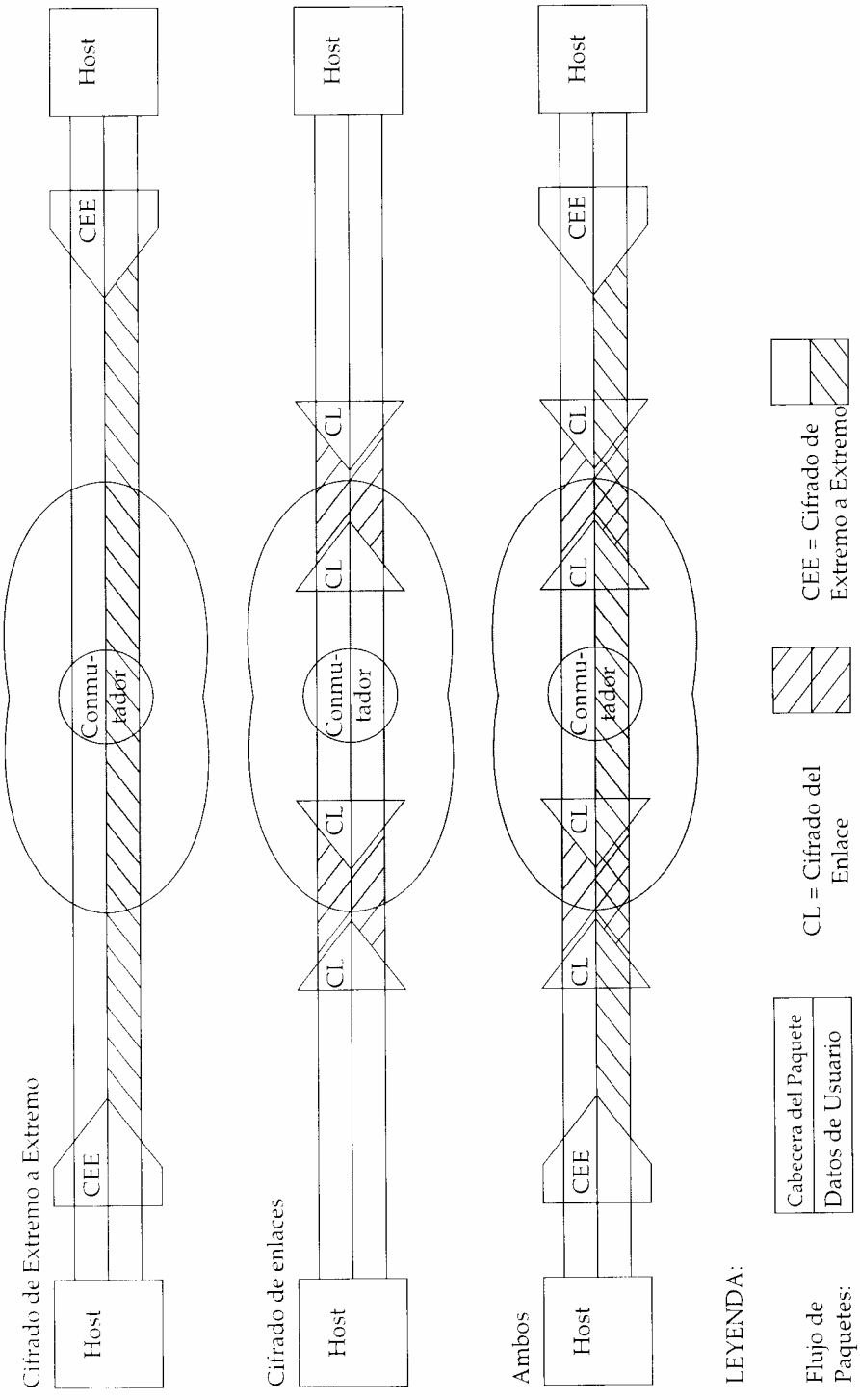


FIGURA 14.18 Cifrado de extremo a extremo y de enlaces

• *Clave permanente:* Una clave permanente es una clave empleada entre dos entidades para la distribución de las claves de sesión.

La configuración consta de los siguientes elementos:
• *Centro de control de acceso:* El centro de control de acceso determina a qué sistemas se les permite comunicar con los otros.
• *Centro de distribución de claves:* Cuando el centro de control de acceso otorga permiso a dos sistemas para establecer una conexión, el centro de distribución de claves proporciona una clave de sesión de un sólo uso para esa conexión.
• *Procesador frontal:* El procesador frontal (PF) lleva a cabo el cifrado de extremo a extremo y obtiene las claves de sesión de parte de un host o un terminal.

Los pasos efectuados en el establecimiento de una conexión se muestran en la figura 14.19. (1) Cuando un host quiere preparar una conexión con otro host, transmite un paquete de solicitud de conexión. (2) El PF guarda el paquete y solicita permiso al centro de control de accesos para establecer la conexión. (3) La comunicación entre el procesador frontal y el centro de control de acceso es cifrada con la clave permanente compartida solamente por el centro de control de acceso y el procesador frontal. El centro de control de acceso tiene una clave única para cada procesador frontal y para el centro de distribución de claves. Si el centro de control de acceso aprueba la solicitud de conexión, envía un mensaje al centro de distribución de claves y le pide que se genere una clave de sesión. (4) El centro de distribución

1. El host envía un paquete solicitando conexión.
2. El frontal almacena el paquete, pidiendo al CCA una clave de sesión.
3. El CCA aprueba la solicitud y se la ordena al CDC.
4. El CDC distribuye una clave de sesión a ambos frontales.
5. Se transmiten los paquetes almacenados.

PF = Procesador Frontal
CCA = Centro de Control de Acceso
CDC = Centro de Distribución de Claves

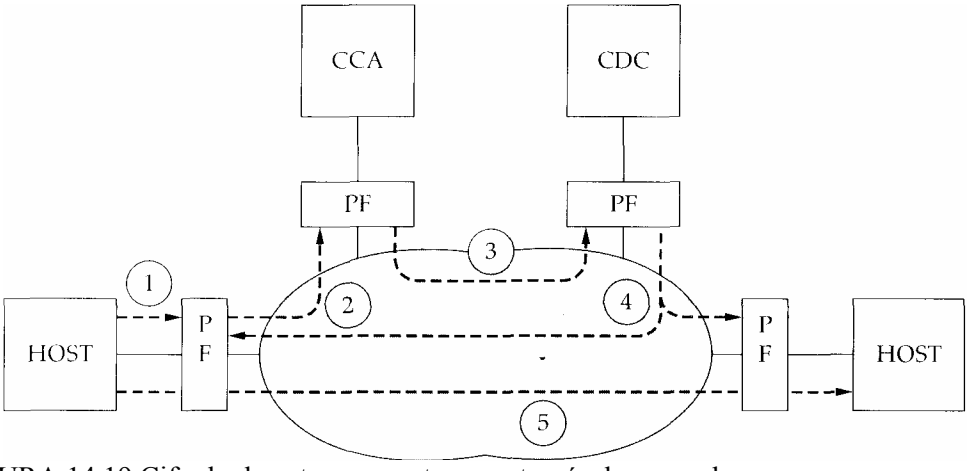


FIGURA 14.19 Cifrado de extremo a extremo a través de una red

de claves genera la clave de sesión y la entrega a los procesadores frontales apropiados, empleando una única clave permanente para cada frontal. (5) El procesador frontal demandante ya puede lanzar el paquete de solicitud de conexión y la conexión se establece entre los dos sistemas finales. Todos los datos de usuario intercambiados entre los dos sistemas finales están cifrados por sus respectivos procesadores frontales mediante la clave de sesión de un sólo uso.

Son posibles diversas variaciones del esquema anterior. Las funciones de control de acceso y distribución de la clave pueden combinarse en un solo sistema. La separación esclarece las dos funciones y proporciona un nivel de seguridad levemente mejorado. Si se desea dejar que dos dispositivos se comuniquen a voluntad, la función de control de acceso no es necesaria: Cuando dos dispositivos desean establecer una conexión, uno de ellos solicita al centro de distribución de claves una clave de sesión. Por último, las funciones realizadas por el procesador frontal no tienen por qué alojarse en un dispositivo separado, sino que pueden incorporarse en el host. La ventaja del procesador frontal es que minimiza el impacto sobre la red. Desde el punto de vista del host, el PF aparece como un nodo de conmutación de paquetes y la interfaz del host con la red no cambia. Desde el punto de vista de la red, el PF aparece como un host más y la interfaz de conmutación de paquetes con el host no cambia.

El método de distribución automática de claves ofrece la flexibilidad y las características dinámicas necesarias para que una serie de usuarios de terminales puedan acceder a una serie de hosts e intercambien datos unos con otros.

Por supuesto, otro método de distribución de claves es el cifrado de clave pública (véase el Apéndice de este capítulo). Una desventaja fundamental del cifrado de clave pública, en comparación con el cifrado convencional, es que los algoritmos para el primero son mucho más complejos. Así pues, en tamaño y coste comparables del hardware, el esquema de clave pública proporciona mucha menor productividad. Una aplicación posible del cifrado de clave pública es emplearlo para la parte de la clave permanente de la figura 14.19, empleando claves convencionales como claves de sesión. Como hay pocos mensajes de control relativos a la cantidad de tráfico de datos de usuario, la reducción de productividad no debería ser un handicap.

Tráfico de relleno

Antes se comentó que, en algunos casos, los usuarios estaban interesados en la seguridad contra el análisis de tráfico. Con el uso del cifrado de enlaces, las cabeceras de los paquetes van cifradas, reduciendo la oportunidad de un análisis de tráfico. Sin embargo, sigue siendo posible en estas circunstancias que un atacante evalúe la cantidad de tráfico en la red y observe la cantidad de tráfico entrante y saliente de cada sistema final. Una contramedida efectiva para este ataque es el tráfico de relleno, ilustrado en la figura 14.20.

El tráfico de relleno es una función que produce una salida cifrada continuamente, incluso en la ausencia de texto en claro. Se genera un flujo continuo de dalos aleatorios. Cuando hay texto en claro disponible, se cifra y se transmite. Cuando la entrada de texto en claro no está presente, los dalos aleatorios se cifran y se transmiten. Esto hace imposible que un atacante distinga entre flujos de datos verdaderos y ruido, y es, por tanto, imposible deducir la cantidad de tráfico.

14.7

RESUMEN

La dependencia creciente de los negocios en el uso de sistemas de proceso de datos y el uso creciente de redes y servicios de comunicación para construir sistemas distribuidos ha originado una tuerte exigencia de seguridad en computadores y redes. La seguridad en las computadores tiene que ver con los mecanismos que están dentro de y relacionados con un sistema informático. El objetivo principal es proteger los recursos de datos del sistema. La seguridad en redes se encarga de la protección de datos y mensajes que se comunican.

Los requisitos de seguridad se evalúan mejor examinando las diversas amenazas a la seguridad a las que se enfrenta una organización. La interrupción del servicio es una amenaza a la disponibilidad. La intercepción de información es una amenaza a la confidencialidad. *Por* último, la modificación de información legítima y la invención no autorizada de información son amenazas a la integridad.

Un campo clave de la seguridad en los computadores es la protección de memoria. Esta es esencial en cualquier sistema en el que hayan varios procesos activos al mismo tiempo. Los esquemas de memoria virtual suelen estar equipados con los mecanismos apropiados para esta tarea.

Otra técnica importante de seguridad es el control de acceso. La finalidad del control de acceso es asegurarse que sólo los usuarios autorizados tienen acceso a un sistema particular y a sus recursos, así como asegurar que el acceso y la modificación de partes concretas de los datos están restringidas a los individuos y programas autorizados. Estrictamente hablando, el control de acceso es un problema de seguridad de computadores más que de seguridad en redes. Es decir, en la mayoría de los casos, los mecanismos de control de acceso se implementan en computadores sencillos para controlar el acceso a dichos computadores. No obstante, puesto que muchos de los accesos a un computador son por medio de un servicio de comunicaciones o de una red, los mecanismos de control de acceso deben diseñarse para funcionar correctamente en un entorno de red distribuido.

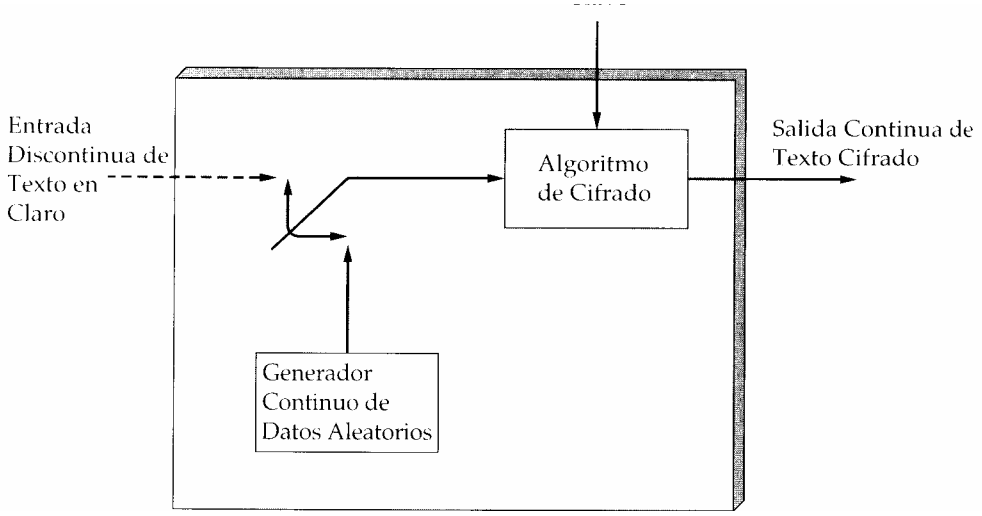


FIGURA 14.20 Dispositivo de cifrado con tráfico de relleno

Una tecnología relativamente nueva, de aplicación creciente en entornos militares y comerciales es la de los sistemas de confianza. El sistema de confianza proporciona un medio de regular el acceso a los datos en función de quien está autorizado a acceder a qué. El punto clave es que se diseño e implemento el sistema de forma que los usuarios puedan tener completa confianza de que el sistema aplicará una política de seguridad determinada. La Agencia de Seguridad Nacional de los EE.UU. está inmersa en un programa de evaluación de productos disponibles comercialmente para certificar aquellos que sean de confianza con un rango particular de capacidades. Esta evaluación puede servir para los clientes no militares.

La seguridad en redes introduce nuevas cuestiones en la protección de información. En este caso, el foco está en la protección de las transmisiones de datos. El *cifrado* es, con mucho, la herramienta automática más importante para la seguridad en redes y comunicaciones. El cifrado es un proceso que oculta los significados cambiando mensajes inteligibles por mensajes ininteligibles. La mayoría de equipos de cifrado disponibles comercialmente emplean cifrado convencional, donde las dos partes comparten una única clave de cifrado y descifrado. El desafío principal del cifrado convencional es la distribución y protección de las claves. La alternativa es un esquema de cifrado de clave pública en el que entran en juego dos claves, una para el cifrado y otra pareja para el descifrado. Una de las claves es privada de la parte que genera el par de claves y la otra se hace pública.

Una amenaza preocupante que es reciente y cada vez mayor es la planteada por los virus y mecanismos de software similares. Esta amenazas se aprovechan de los puntos vulnerables del software del sistema para obtener acceso no autorizado a la información o para degradar el servicio del sistema.

14.8 LECTURAS RECOMENDADAS

[STAL95] ofrece un tratamiento detallado de los temas del capítulo. Otros libros dignos de mención son [DAV189] y [PFLE89]. Una buena discusión de las cuestiones de los sistemas operativos se puede encontrar en [MAEK87]. [HOFF90] y [DENN90] contienen reimpresiones de muchos artículos clave que tratan de intrusos y virus. [GASS88] ofrece un estudio completo de los sistemas informáticos de confianza.

DAVI88 DAVIES, D. y PRICE, W. *Security for Computer Networks*. Wiley, Nueva York, 1989.

DENN90 DENNING, P. *Computers Under Attack: Intruders, Worms, and Viruses*. Addison-Wesley, Reading, MA, 1990.

GASS88 GASSER, M. *Building a Secure Computer System*. Van Nostrand Reinhold, Nueva York, 1988.

HOFF90 HOFFMAN, L., ed. *Rogue Programs: Viruses, Worms, and Trojan Horses*. Van Nostrand Reinhold, Nueva York, 1990.

MAEK87 MAEKAWA, M., OLDEHOEFT, A. y OLDEHOEFT, R. *Operating Systems: Advanced Concepts*. Benjamin Cummings, Menlo Park, CA, 1987.

PFLE89 PFLEEGER, C. *Security in Computing*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

STAL94b STALLINGS, W. *Network and Internetwork Security: Principles and Practice*. Macmillan, Nueva York, 1994.

4.9

PROBLEMAS

14.1 Supóngase que las contraseñas se eligen como combinaciones de 4 caracteres entre 26 caracteres alfabéticos. Supóngase que un adversario es capaz, de intentar contraseñas a razón de una por segundo.

a) Suponiendo que no hay realimenlación al adversario hasta que se completa cada intento, ¿cuál es el tiempo que se estima necesario para descubrir la contraseña correcta?

b) Suponiendo que hay realimentación al adversario que es notificado con cada carácter incorrecto que introduzca. ¿cuál es el tiempo que se estima necesario para descubrir la contraseña correcta?

14.2 Supóngase que unos elementos de origen de longitud k se traducen de alguna forma uniforme a elementos de destino de longitud p. Si cada dígito puede tomar uno de entre r valores, el numero de elementos de origen es r^k y el número de elementos de destino es el número menor r^p. Un elemento de origen concreto x_i, se traduce a un elemento de destino concreto y_j.

a) ¿Cuál es la probabilidad de que el adversario seleccione el elemento correcto de origen en un intento?

b) ¿Cuál es la probabilidad de que el adversario genere un elemento de origen x_K diferente ($x_I \neq x_K$) que resulte en el mismo elemento de destino y_i?

c) ¿Cuál es la probabilidad de que el adversario genere el elemento de destino correcto en un intento?

14.3 Un generador fonético de contraseñas coge dos segmentos aleatorios de cada contraseña de seis letras. La forma de cada segmento es CVC (consonante, vocal, consonante), donde V = <a. e. i. o, u> y C = V

a) ¿Cuál es la población total de contraseñas?

b) ¿Cuál es la posibilidad de que un adversario adivine una contraseña correctamente?

14.4 En la a figura 14.15, un enlace de la cadena de "copiar v observar después" del caballo de Troya está roto. Hay otras dos perspectivas posibles de ataque por parte de Drake: la conexión de Drake y el intento de leer la cadena directamente: y la asignación por parte de Drake de un nivel de seguridad susceptible al archivo del bolsillo trasero. ¿Impide estos ataques el monitor de referencia?

14.5 La necesidad de la regla de "no leer arriba" para un sistema de seguridad multinivel es bastante obvia. ¿Cuál es la importancia de la regla de "no escribir abajo"?

14.6 Dar algunos ejemplos donde el análisis de tráfico podría comprometer la seguridad. Describir situaciones donde el cifrado de extremo a extremo combinado con el cifrado de enlaces seguirían permitiendo que el análisis de tráfico fuera peligroso.

14.7 La necesidad de seguridad y el deseo de compartir los datos han estado siempre reñidos uno con el otro. Cuanta más gente accede a las redes de computadores de miles de nodos, ¿cuáles son las implicaciones de los gusanos y virus en esta interacción creciente? Considérese no sólo las ramificaciones técnicas, sino también el impacto potencial en la comunicación humana.

14.8 Los esquemas de distribución de claves que emplean un centro de control de acceso y/o un centro de distribución de claves tienen puntos centrales vulnerables a los ataques. Discutir las implicaciones en la seguridad de dicha centralización.

14.9 La protección de los usuarios de una red contra gusanos y amenazas similares, ¿debería ser responsabilidad de la propia red o de las máquinas que la utilizan? Razónese la respuesta.

14.10 Debido a los riesgos conocidos del sistema de contraseñas de UN1X, la documentación del SunOS 4.0 recomienda que el archivo de contraseñas sea eliminado y sustituido con un archivo públicamente leíble llamado /etc/publickey. Una entrada en el archivo para el usuario A consta de un identificador del usuario, ID_A, la clave pública del usuario, KU_A y la clave privada correspondiente

KR_A. Esta clave privada se cifra mediante el DES con una clave obtenida de la contraseña de conexión del usuario, P_A. Cuando A se conecte al sistema, descifra E_{PA} [KU_A] para obtener KU_A.

a) El sistema verifica entonces que P_A fue suministrada correctamente. ¿Cómo lo hace?

b) ¿Cómo puede atacar el sistema un adversario?

14.11 Se dijo que la inclusión de la base en el esquema de contraseñas de UNIX aumenta la dificultad de adivinar una en un factor de 4.096. Pero la base está almacenada en claro en la misma entrada que la contraseña cifrada correspondiente. Por tanto, dichos caracteres son conocidos para el atacante y no tienen que ser adivinados. Entonces, ¿por qué se afirma que la base aumenta la seguridad?

14.13 Suponiendo que se ha respondido correctamente al problema anterior y se ha comprendido la importancia de la base. aquí va otra cuestión. ¿No sería posible frustrar por completo a todos los averiguadores de contraseñas incrementando drásticamente el tamaño de la base hasta, por ejemplo, 24 ó 48 bits?

14.14 Supóngase que las contraseñas están limitadas a emplear 64 caracteres (A-Z. a-z, 0-9. "." y "/") y que todas las contraseñas son de ocho caracteres de longitud. Supóngase un averiguador de contraseñas con una tasa de cifrado de 14,4 millones de cifrados por segundo.

a) ¿Cuánto tardará en probar exhaustivamente todas las contraseñas posibles de un sistema UNIX?

b) Si un archivo de contraseñas de UNIX contiene *N* contraseñas cifradas, ¿cuál es el número esperado de adivinaciones hasta encontrar, al menos, una contraseña?

c) ¿Cuánto tardará en llevar a cabo el número esperado de adivinaciones?

14.15 Hay un defecto en el programa de virus de la figura 14.11.

a) ¿Cuál es?

b) Arreglarlo

APÉNDICE 14A

Cifrado

Una de las herramientas automáticas más importantes en la seguridad de los computadores es el cifrado. El cifrado es un proceso que oculta los significados cambiando mensajes inteligibles por mensajes ininteligibles. El cifrado se puede lograr por medio de códigos o cifras. Los sistema de códigos emplean una tabla predefinida o diccionario para sustituir las palabras o frases sin sentido de cada mensaje o parte de un mensaje. El código más simple sustituye una letra por otra del alfabeto. Con cifras se emplea un algoritmo computable que traduce cualquier flujo de bits de un mensaje en un criptograma ininteligible. Como las técnicas de cifra se prestan más fácilmente a su automatización, son éstas las técnicas que se emplean en los servicios de seguridad actuales en redes y computadores. Este apéndice discute solamente este segundo tipo de cifrado.

Se comenzará examinando los enfoques tradicionales de cifrado, conocidos en la actualidad como cifrado convencional. Después de verá una técnica nueva y bastante útil, conocida como cifrado de clave pública.

14A.1 *Cifrado convencional*

La figura 14.21a ilustra el proceso de cifrado convencional. El mensaje original inteligible, conocido como texto en claro, se convierte a otro aparentemente aleatorio y sin sentido, conocido como texto cifrado. El proceso de cifrado consta de un algoritmo y una clave. La clave es una cadena de hits relativamente corta que controla el algoritmo. El algoritmo pro

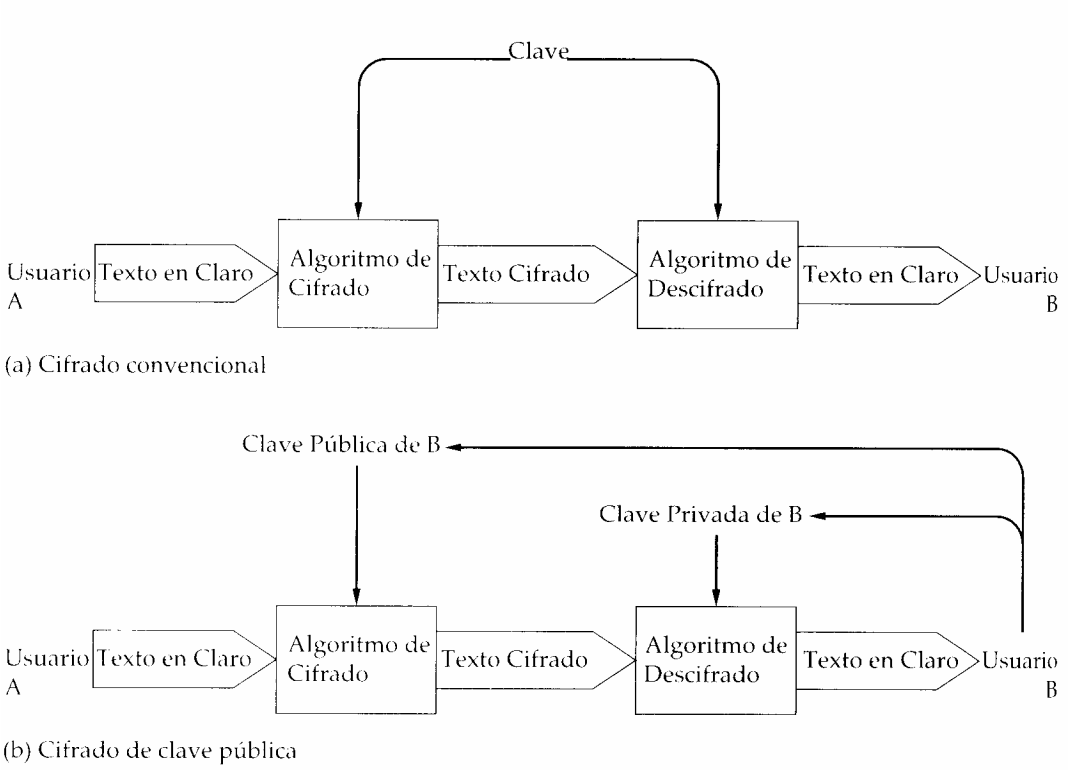


FIGURA 14.21 Cifrado

duce una salida diferente, dependiendo de la clave específica que se use en cada momento. Un cambio en la clave cambia radicalmente la salida del algoritmo.

Una vez que se produce el texto cifrado, éste es transmitido. A la recepción, el texto cifrado puede transformarse al texto en claro original, mediante un algoritmo de descifrado y la misma clave que se utilizó para el cifrado.

La seguridad del cifrado convencional depende de varios factores. En primer lugar, el algoritmo de cifrado debe ser suficientemente potente como para que sea impracticable descifrar un mensaje a partir del texto cifrado solamente. Más que eso, la seguridad del cifrado convencional depende del secreto de la clave, no del secreto del algoritmo. Es decir, se supone que es impracticable descifrar un mensaje a partir del texto cifrado junto al conocimiento del algoritmo de cifrado o descifrado. Con otras palabras, no se necesita mantener el secreto del algoritmo: sólo se necesita mantener el secreto de la clave.

Esta característica de los algoritmos convencionales es el que hace factible su uso tan extendido. El hecho de que el algoritmo necesario no tiene que mantenerse en secreto significa que los fabricantes pueden desarrollar implementaciones de bajo coste en chips de algoritmos de cifrado de dalos. Estos están ampliamente disponibles y están incorporados en una serie de productos. Con el uso del cifrado convencional, el principal problema de la seguridad es mantener el secreto de la clave.

El Estándar de Cifrado de Datos

El esquema más usado de cifrado se basa en el estándar de cifrado de datos (DES, *Data Encryption Standard),* adoptado en 1977 por la Oficina Nacional de Estándares (NBS, *Natio-*

Acrobarnal Bareau of Standards}. Para el DES, los datos se cifran en bloques de 64 bits, mediante una clave de 56 bits. Con la clave, la entrada de 64 bits es transformada en una serie de pasos en una salida de 64 bits. Los mismos pasos, con la misma clave, se usan para deshacer el cifrado.

El DES ha disfrutado de un amplio y creciente uso. Por desgracia, también ha sido objeto de mucha controversia sobre cómo es de seguro. La preocupación principal es la longitud de la clave, que algunos consideran demasiado corta. Para apreciar la naturaleza de la controversia, se va a revisar rápidamente la historia del DES.

El DES es el resultado de una solicitud de propuesta de un estándar nacional de cifrado lanzado por la NBS en 1973. Por aquel tiempo, IBM estaba en las etapas finales de un proyecto llamado Lucifer, para desarrollar su propio servicio de cifrado. IBM propuso el esquema de Lucifer, que era, de largo, el mejor sistema propuesto. Era tan bueno, de hecho. que desconcertó considerablemente a algunas personas de la Agencia de Seguridad Nacional (NSA), quienes lo han considerado hasta la actualidad muy por delante del resto del mundo en el todavía misterioso arte de la criptografía. El DES, adoptado finalmente, fue esencialmente el mismo que Lucifer, con una diferencia crucial: El tamaño original de la clave de Lucifer era de l28 bits, mientras que el estándar final empleaba una de 56 bits. ¿Cuál es la importancia de los 72 bits abandonados?

Hay básicamente dos maneras de romper un cifrado. Una forma es aprovechar las propiedades de cualesquiera funciones matemáticas que formen la base de los algoritmos de cifrado para llevar a cabo un ataque criptoanalítico. Se supone en general que el DES es inmune a tales ataques, aunque el papel de la NSA en el perfilado del estándar final del DES deja algunas dudas. La otra forma es un ataque de fuerza bruta en el que se prueben todas las claves posibles en una búsqueda exhaustiva. Es decir, se intenta descifrar el texto cifrado con todas las posibles combinaciones de claves de 56 bits hasta que aparece algo inteligible. Con sólo 56 bits en la clave del DES, hay 2^{56} claves diferentes (un número que es incómodamente pequeño y se hace cada vez, más pequeño a medida que los computadores se hacen más rápidas).

De acuerdo con David Kahn, autor de *Codebreakers* (Nueva York: Macmillan, 1967) y un experto notable en criptografía. Lucifer hizo estallar un debate en el seno de la NSA. "El lado de los rompecódigos quería asegurarse de que el cifrado era suficientemente débil como para que la NSA lo solucionara al ser usado por naciones y compañías extranjeras. El lado de los fabricacódigos quería que el cifrado fuera tal que certificara que su uso por los americanos fuese realmente bueno. El compromiso burocrático resultante fue que el cifrado fuese suficientemente débil de leer pero fuerte para proteger el tráfico contra el observador casual".

Sean cuales sean los méritos del caso, el DES ha prosperado en los últimos años y es muy usado, especialmente en aplicaciones financieras. Excepto en campos de susceptibilidad extrema, el uso del DES en aplicaciones comerciales no debería ser causa de preocupación de los administradores responsables.

14A.2 Cifrado de clave pública

Como se ha visto, una de las mayores dificultades de los esquemas de cifrado convencional es la necesidad de distribuir las claves de forma segura. Una forma astuta para dar un rodeo a esta exigencia es un esquema de cifrado que, sorprendentemente, no requiere la distribución de la clave. Este esquema, conocido como cifrado de clave pública y propuesto por primera vez en 1976, se ilustra en la figura l4.21b.

Para los esquemas de cifrado convencionales, las claves empleadas para el cifrado y el descifrado eran las mismas. Esta no es una condición necesaria. En su lugar, es posible desarrollar un algoritmo que utilice una clave para el cifrado y otra pareja pero diferente para el descifrado. Mas aún, es posible desarrollar algoritmos tales que el conocimiento del algoritmo de cifrado junto con la clave de cifrado no sea suficiente para determinar la clave de descifrado. Así pues, la siguiente técnica puede funcionar:

1. Cada sistema final de la red genera un par de claves para usar en el cifrado y descifrado de los mensajes que reciba.
2. Cada sistema publica su clave de cifrado situándola en un registro público o en un archivo. Esta es la clave pública. La clave pareja se mantiene privada.
3. Si A desea enviar un mensaje a B, cifra el mensaje con la clave pública de B.
4. Cuando B recibe el mensaje, lo descifra con la clave privada de B. Ningún otro receptor puede descifrar el mensaje porque sólo B conoce la clave privada de B.

Como se puede observar, el cifrado de clave pública resuelve el problema de la distribución de la clave porque no hay claves que distribuir. Todos los participantes tienen acceso a las claves públicas, mientras que las privadas se generan localmente por cada participante y, por tanto, no necesitan ser distribuidas. Siempre que el sistema controle su clave privada, las comunicaciones que lleguen serán seguras. En cualquier momento, un sistema podrá cambiar su clave privada y publicar la clave pública pareja que reemplace a la antigua.

Una gran desventaja del cifrado de clave pública comparado con el cifrado convencional es que los algoritmos de los primeros son mucho más complejos. Así pues, por tamaño y coste del hardware comparables, el esquema de clave pública ofrecerá una productividad mucho menor.

La tabla 14.8 resume algunos de los aspectos importantes del cifrado convencional y de clave pública.

TABLA 14.8 Cifrado Convencional y de Clave Pública

Cifrado convencional	Cifrado de clave pública
Necesario para funcionar:	*Necesario para funcionar:*
1. El mismo algoritmo con la misma clave puede usarse para el cifrado y el descifrado	1. Un mismo algoritmo se emplea para el cifrado y el descifrado con un par de claves, una para el cifrado y otra para el descifrado
2. El emisor y el receptor deben compartir el algoritmo y la clave.	2. El emisor y el receptor deben tener una de las claves de cada par.
Necesario para la seguridad:	*Necesario para la seguridad:*
1. La clave debe mantenerse secreta.	1. Una de las dos claves debe mantenerse en secreto.
2. Debe resultar imposible o, al menos, impracticable, descifrar un mensaje si no hay ninguna otra información disponible.	2. Debe resultar imposible o, al menos, impracticable, descifrar un mensaje si no hay ninguna otra información disponible.
3. El conocimiento del algoritmo junto a muestras de texto cifrado debe ser insuficiente para determinar la clave.	3. El conocimiento del algoritmo y de una de las claves junto a muestras de texto cifrado debe ser insuficiente para determinar la otra clave

APÉNDICE A

Análisis de colas

En varios puntos de este libro, especialmente en el capítulo 6, se hace referencia al análisis de colas y se presentan resultados en función de dicho análisis. En realidad, el análisis de colas es una de las herramientas más importantes para las personas relacionadas con el análisis de computadores y de redes. Puede usarse para proporcionar respuestas aproximadas a una multitud de preguntas, tales como las siguientes:

- ¿Qué ocurre con el tiempo de recuperación de un archivo cuando aumenta la utilización de la E/S con el disco?
- ¿.Cambia el tiempo de respuesta si se dobla la velocidad del procesador y el número de usuarios del sistema?
- ¿Cuántas líneas podría soportar un sistema de tiempo compartido en un disco de conexión?
- ¿Cuántos terminales se necesitan en un centro de consultas en línea y cuánto tiempo estará desocupado el operador?

El número de preguntas que se pueden resolver con el análisis de colas es indefinido y éstas tocan casi todos los temas tratados en este libro. La posibilidad de hacer tal análisis es una herramienta esencial para cualquiera que esté implicado en este campo.

Si bien la teoría de colas es matemáticamente compleja, la aplicación de la teoría de colas al análisis del rendimiento es, en la mayoría de los casos, extraordinariamente sencilla. Todo lo que se necesita es el conocimiento de unos conceptos elementales de estadística (medias y desviaciones típicas) y una comprensión básica de la aplicabilidad de la teoría de colas. Armado de este conocimiento, el analista puede realizar a menudo un análisis de colas en el reverso de un sobre empleando unas tablas que se pueden conseguir fácilmente o usando programas sencillos de computador que ocupan sólo unas pocas líneas de código.

El propósito de este apéndice es ofrecer una guía práctica del análisis de colas. Se hace un tratamiento importante de un subconjunto del tema. En la sección final, se dan referencias a lecturas adicionales. Un anexo de este apéndice sirve para revisar algunos conceptos elementales de probabilidades y estadística.

A.1 ¿POR QUE EL ANÁLISIS DE COLAS?

Hay muchos casos en los que es importante ser capaz de prever el efecto de los cambios en un diseño, tanto cuando se espera un incremento de la carga del sistema como cuando se estima un cambio de diseño. Por ejemplo, supóngase una organización que dé soporte a un cierto número de terminales, varios computadores personales y estaciones de trabajo con una red de área local (LAN, *Local Area Netwok*) de 4 Mbps (megabits por segundo). Se va a añadir a la misma red otro departamento situado en el edificio. ¿Podrá gestionar la LAN existente el incremento de la carga de trabajo o sería mejor crear una segunda LAN con un puente entre ambas? Hay otros casos en los que no existe una instalación previa, sino que se necesita diseñar el sistema en función de la demanda esperada. Por ejemplo, un departamento intenta equipar a todo su personal con computadores personales y configurar todas estas computadores en una red con un servidor de archivos. En función de la experiencia de cualquier otra sección de la compañía, se puede estimar la carga generada por cada *PC.*

La preocupación es el rendimiento del sistema. En una aplicación interactiva o de tiempo real, el tiempo de respuesta suele ser el parámetro de interés. En otros casos, la cuestión principal es la productividad. En cualquier caso, las estimaciones del rendimiento se van a hacer en función de la información de carga existente o en función de la carga estimada para un nuevo entorno. Son posibles varias soluciones:

1. Hacer un análisis a posterior en función de valores reales.
2. Hacer una simple estimación a escala de la experiencia previa adaptada al futuro entorno.
3. Desarrollar un modelo analítico basado en la teoría de colas.
4. Construir y ejecutar un modelo de simulación.

La opción I no es una opción válida: Esperar y ver qué pasa. Esto genera usuarios insatisfechos y elecciones poco aconsejables.

La opción 2 parece más prometedora. El analista puede tomar la decisión de que es imposible estimar las demandas futuras con cierto grado de certeza. Por tanto, no tiene sentido acometer un procedimiento de modelado exacto. En su lugar, una perspectiva tosca pero eficaz proporcionará un cálculo aproximado. El problema de este método es que el comportamiento de la mayoría de los sistemas sometidos a una carga cambiante no es el que uno podría esperar de manera intuitiva. Si en el entorno existe algún servicio compartido (por ejemplo, una red, una línea de transmisión o un sistema de tiempo compartido), el rendimiento del sistema suele reaccionar de forma exponencial a los incrementos de la demanda.

La figura A. I es un ejemplo habitual. La línea superior muestra lo que ocurre con el tiempo de respuesta del usuario en un servicio compartido a medida que se incrementa la carga de dicho servicio. La carga se expresa como una fracción de la capacidad. De este modo, si se trabaja con una entrada de disco que es capaz de transferir 1000 bloques por segundo, una carga de 0,5 representa una transferencia de 500 bloques por segundo y el tiempo de respuesta es la cantidad de tiempo que se tarda en transmitir cualquier bloque. La

línea inferior es una estimación simple, basada en el conocimiento del comportamiento del sistema con una carga de hasta 0,5.[1] Nótese que, aunque las cosas parecen de color de rosa cuando se realiza la estimación simple, el rendimiento del sistema se desploma de hecho con una carga superior a 0,8 ó 0,9.

Por tanto, hace falta una herramienta de predicción más exacta. La opción 3 propone un modelo analítico, el cual puede expresarse como un conjunto de ecuaciones que pueden resolverse para obtener los parámetros deseados (tiempo de respuesta, productividad, etc.). En los problemas con computadores, sistemas operativos y redes y, de hecho, para los muchos problemas prácticos reales, los modelos analíticos basados en la teoría de colas presentan una aproximación razonablemente buena de la realidad. La desventaja de la teoría de colas es que deben hacerse una serie de supuestos para obtener las ecuaciones de los parámetros de interés.

El último método es el de los modelos de simulación. Con un lenguaje de simulación potente y flexible, el analista puede modelar la realidad con gran detalle y evitar los numerosos supuestos requeridos por la teoría de colas. Sin embargo, en la mayoría de los casos, un modelo de simulación no es necesario o, al menos, no es aconsejable, como primer paso del análisis. De una parte, tanto las medidas existentes como las estimaciones de la carga futura llevan consigo un cierto margen de error. Así pues, sin importar lo bueno que sea el modelo de simulación, la valía de los resultados está limitada por la calidad de la entrada. Por otra

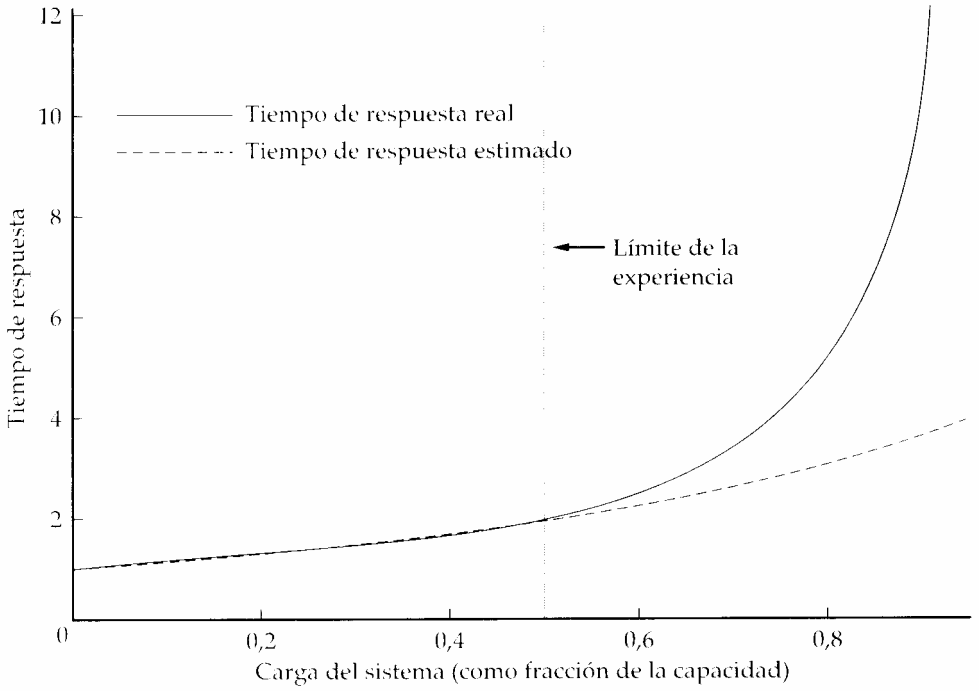


FIGURA A.1 Tiempo de respuesta real frente al estimado

[1] De hecho, la línea inferior está basada en el ajuste de un polinomio de tercer orden para los datos disponibles con una carga de hasta 0,5.

parte, pese a los numerosos supuestos exigidos por la teoría de colas, los resultados generados suelen ser bastante más aproximados que los que se obtendrían con un análisis de simulación más cuidadoso. Es más, un análisis de colas para un problema bien definido puede hacerse literalmente en cuestión de minutos, mientras que el ejercicio de la simulación puede llevar días, semanas o más para su programación y puesta en marcha.

En consecuencia, es labor del analista conocer a fondo las bases de la teoría de colas.

A.2 MODELOS DE COLAS

Colas de un solo servidor

El sistema de cola más simple existente es el de la figura A.2. El elemento central del sistema es un servidor, que proporciona cierto servicio a los individuos. Al sistema llegan individuos de una cierta población para ser servidos. Si el servidor está libre, se sirve inmediatamente al individuo. En otro caso, el individuo se añade a una fila de espera.[2] Cuando el servidor ha terminado de servir a un individuo, éste sale del servicio. Si hay individuos esperando en la cola, se expide uno inmediatamente hacia el servidor.

La figura A.2 también ilustra los parámetros asociados con un modelo de colas. Los individuos llegan al servicio con una cierta tasa media de llegada (individuos que llegan por segundo). En un instante dado, en la cola habrá un cierto número de individuos (cero o más); el número medio de individuos esperando es w y el tiempo medio que un individuo debe esperar es t_w. Nótese que t_w es la media de todos los individuos que llegan, incluyendo aquellos que no tienen que esperar nada. El servidor gestiona los individuos entrantes con un tiempo medio de servicio x; éste es el intervalo de tiempo entre que se expide un individuo hacia el servidor y la salida de dicho individuo del servidor. La utilización es la tracción de tiempo que el servidor está ocupado, medida a lo largo de un cierto intervalo de tiempo. Por último, hay dos parámetros que se aplican al sistema en conjunto. El parámetro q es el número medio de individuos en el sistema, incluyendo al

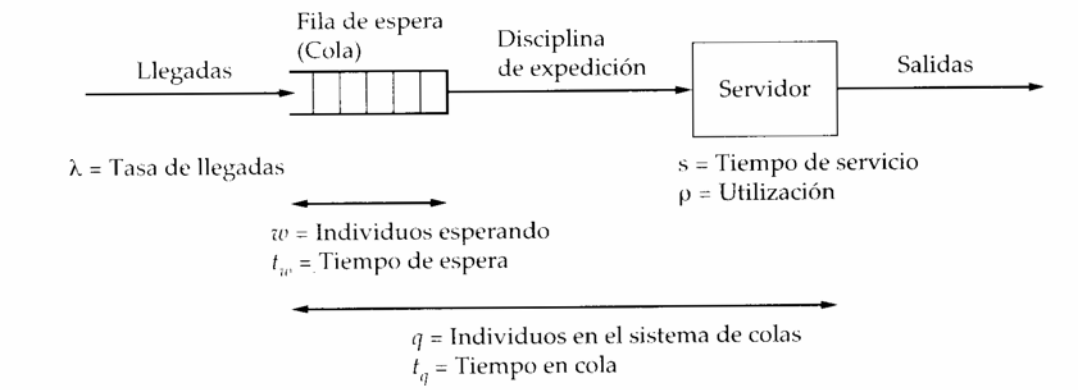


FIGURA A.2 Estructura y parámetros de un sistema de colas de un solo servidor

[2] La fila de espera se llama cola en algunos textos; también es habitual referirse a todo el sistema como una cola.

individuo que está en servicio (si lo hay) y a los individuos que están esperando (si los hay); y t_q, es el tiempo medio que un individuo pasa en el sistema, tanto esperando como en servicio.

Si se supone que la capacidad de la fila de espera es infinita, los individuos no se rechazan nunca del sistema; únicamente son retrasados hasta que se les pueda servir. Bajo estas circunstancias, la tasa de salida es igual a la tasa de llegadas. A medida que se incrementa la tasa de llegada, que es el porcentaje de tranco que pasa por el sistema, se incrementa la utilización y, con ella, la congestión. La fila de espera se hace más larga, aumentando el tiempo de espera. Con p = 1, el servidor se satura, estando funcionando el 100% del tiempo.

Así pues, la tasa máxima teórica de entradas que puede manejar el sistema es:

$$\lambda_{max} = \frac{1}{s}$$

Sin embargo, las filas de espera se hacen muy largas cuando se está próximo a la saturación del sistema, creciendo sin límites cuando p = 1. Consideraciones prácticas, tales como exigencias en tiempos de respuesta o en el tamaño de los buffers, limitan normalmente la tasa de entrada para un servidor único del 70 al 90% del máximo teórico.

Antes de continuar, hay que hacer algunos supuestos sobre este modelo:

• *Población de individuos:* Normalmente, se supone una población infinita lo que significa que la tasa de llegada no se ve alterada por la pérdida de población. Si la población es finita, la población disponible para llegadas se reduce en el número de individuos que estén en el sistema: esto suele reducir la tasa de llegadas de forma proporcional.

• *Tamaño de la cola:* Normalmente, se supone un tamaño de cola infinito. Así pues, la fila de espera puede crecer sin límites. Con una cola finita, es posible perder individuos del sistema. En la práctica, cualquier cola es finita. En muchos casos, esto no introduce diferencias notables en el análisis. Más adelante, se abordará esta cuestión.

• *Método de expedición:* Cuando el servidor queda libre y hay más de un individuo esperando, se debe tomar una decisión sobre el individuo que se va a expedir a continuación. El método más simple es el de primero en llegar, primero en salir; ésta es la disciplina que se emplea normalmente cuando se usa el término de cola. Otra posibilidad es la de último en llegar, primero en salir. En la práctica, se pueden encontrar disciplinas basadas en el tiempo de servicio. Por ejemplo, un nodo de conmutación de paquetes puede elegir para expedir primero los paquetes más cortos (para generar el mayor número posible de paquetes salientes) o primero los más largos (para minimizar el tiempo de procesamiento relativo al de transmisión). Por desgracia, una disciplina basada en el tiempo de servicio es muy difícil de modelar analíticamente.

La tabla A. 1 resume la notación empleada en la figura A.2 e introduce algunos otros parámetros de utilidad. En particular, suele interesar la variabilidad de ciertos parámetros, lo cual queda representado fielmente por la desviación típica.

Colas multiservidor

La figura A.3 muestra una generalización del modelo simple que se ha tratado. En este caso. hay múltiples servidores, todos los cuales comparten una tila de espera. Si un individuo llega y hay, por lo menos, un servidor disponible, el individuo se expide inmediatamente a ese servidor. Se supone que todos los servidores son iguales; así pues, hay disponible más de un servidor, es indiferente cuál se elija. Si todos los servidores están ocupados, se em-

TABLA A.1 Notación empleada en este apéndice

λ	= número medio de llegadas por segundo
s	= tiempo medio de servicio para cada llegada
σ_s	= desviación típica del tiempo de servicio
ρ	= utilización; fracción de tiempo que está ocupado el servicio
q	= número medio de individuos en el sistema (esperando y en servicio)
t_q	= tiempo medio que pasa un individuo en el sistema
σ_q	= desviación típica de q
σ_{tq}	= desviación típica de t_q
w	= número medio de individuos esperando servicio
t_w	= tiempo medio de espera del servicio para un individuo
t_d	= tiempo medio de espera para individuos que tienen que esperar (no incluye los individuos con tiempo de espera = 0)
σ_w	= desviación típica de w
M	= número de servidores
$m_x(r)$	= percentil r-ésimo; el valor de r por debajo del cual x aparece el r % del tiempo

pieza a formar una tila de espera. Tan pronto como quede libre un servidor, se expide un individuo de la fila de espera, usando la disciplina de servicio en vigor.

Con excepción de la utilización, todos los parámetros de la figura A.2 se pueden trasladar al caso multiservidor con la misma interpretación. Si se tienen M servidores idénticos, p es la utilización de cada servidor y se puede considerar M, como la utilización de todo el sistema. Así pues, la utilización máxima teórica es M x 100% y la lasa de entrada máxima teórica es:

$$\lambda_{max} = \frac{M}{s}$$

Relaciones básicas para las colas

Para llegar más lejos, se van a tener que hacer algunos supuestos de simplificación. Estos supuestos corren el riesgo de hacer que los modelos sea menos valederos para diversas situaciones del mundo real. Afortunadamente, en la mayoría de los casos, los resultados serán lo suficientemente exactos para el diseño y la planificación.

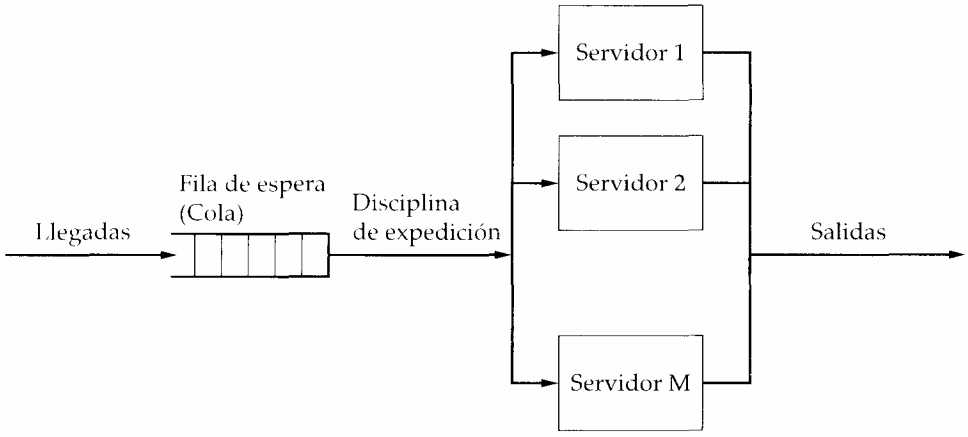


FIGURA A.3 Estructura de un sistemas de colas multiservidor

Sin embargo, hay algunas relaciones que son ciertas en el caso general y que se ilustran en la tabla A.2. Por sí mismas, estas relaciones no son particularmente útiles.

Supuestos

La labor fundamental del análisis de colas es la siguiente: Dada la siguiente información como entrada:

- Tasa de llegadas
- Tiempo de servicio

proporcionar una información de salida sobre:

- Individuos esperando
- Tiempo de espera
- Individuos en la cola
- Tiempo en la cola

¿Qué se desearía conocer especialmente de estas salidas? Sin duda, se desean conocer los valores medios (w, t_w, q, t_q). Además, sería útil saber algo sobre sus variabilidades. Así pues, serían útiles las desviaciones típicas (σ_w, σ_{tw}, σ_q, σ_{tq}). También pueden ser útiles otros valores. Por ejemplo, para diseñar el buffer asociado a un multiplexor, podría ser útil conocer cuál es el tamaño del buffer para que la probabilidad de desbordamiento sea menor que 0,001?. Es decir, ¿cuál es el valor de N, tal que $P_r[q<N] = 0{,}999$?

Responder a estas cuestiones exige, en general, un conocimiento completo de la distribución de probabilidad de la tasa de llegadas y del tiempo de servicio. Es más, incluso con dicho conocimiento, las fórmulas resultantes son extremadamente complejas. Así pues, para hacer tratable el problema, hay que realizar algunos supuestos que lo simplifiquen.

El más importante de estos supuestos es que la tasa de llegadas sigue una distribución de Poisson, lo que equivale a decir que el tiempo entre llegadas es exponencial. Este supuesto se hace casi siempre. Sin él, el análisis de colas es poco práctico. Con este supuesto, resulta que se pueden obtener muchos resultados útiles si sólo se conocen la media y la desviación típica de la tasa de llegadas y del tiempo de servicio.

Las cosas pueden simplificarse aún más y se pueden obtener resultados más detallados si se supone que el tiempo de servicio es exponencial o constante.

Se ha construido una notación útil para resumir los supuestos principales que se realizan en el desarrollo de un modelo de colas. La notación es *X/Y/N*, donde *X* se refiere a la distri-

TABLA A.2 Algunas relaciones básicas para las colas

$\rho = \lambda s$	para servidor único
$\rho = \frac{\lambda s}{M}$	para varios servidores
$q = \lambda t_q$	Fórmula de Little
$w = \lambda t_w$	
$t_q = t_w + s$	
$q = w + \rho$	para servidor único
$q = w + M_\rho$	para varios servidores

bución de los tiempos entre llegadas, *Y* se refiere a la distribución de los tiempos de servicio y *N* se refiere al número de servidores. Las distribuciones más comunes se denotan como:

G = llegada o tiempo de servicio general e independiente

M = distribución exponencial negativa

D = llegadas deterministas o servicios de duración fija.

De este modo, M/M/1 se refiere a un modelo de colas de un solo servidor con llegadas según Poisson y tiempos de servicio exponenciales.

A. 3 COLAS DE UN SOLO SERVIDOR

La tabla A.3a ofrece algunas fórmulas aplicables a las colas de servidor único que siguen el modelo M/G/1, es decir, la tasa de llegadas es una Poisson. Haciendo uso de un factor de escala. A, es sencillo obtener las ecuaciones de algunas de las variables clave de salida. Nótese que el factor clave en el parámetro de escala es la razón entre la desviación típica del tiempo de servicio y la media. No es necesaria ninguna otra información sobre el tiempo de servicio. Dos casos son de especial interés. Cuando la desviación típica es igual a la media, la distribución del tiempo de servicio es exponencial. Este es el caso más simple y el de resultados más sencillos de calcular. La tabla A.3b muestra las versiones simplificadas de las ecuaciones de (σ_w, σ_{tw}, σ_q, σ_{tq}), además de algunos otros parámetros de interés. El otro caso interesante es cuando la desviación típica del tiempo de servicio es igual a cero, es decir, el tiempo de servicio es constante. Las fórmulas correspondientes se muestran en la tabla A.3c.

Las figuras A.4 y A.5 representan los valores del tamaño medio de la cola y del tiempo en cola frente a la utilización, para tres valores de σ_s/s. Nótese que el rendimiento peores el del tiempo de servicio exponencial y el mejor es el del tiempo de servicio constante. Normalmente, se puede considerar que el tiempo de servicio exponencial es el caso peor. Un análisis basado en esta suposición ofrecerá resultados conservadores. Esto es bueno porque existen tablas disponibles para el caso M/M/1 donde pueden buscarse los valores rápidamente.

¿Qué valores de σ_s/s tienen más probabilidad de aparecer? Se pueden considerar cuatro regiones:

- *Cero:* Es el caso poco habitual de un tiempo de servicio constante. Por ejemplo, si lodos los mensajes transmitidos fuesen de la misma longitud, se estaría dentro de esta categoría.
- *Cociente menor que I:* Puesto que este cociente es mejor que el del caso exponencial, las tablas M/M/1 darán tamaños de cola y tiempos algo mayores de lo que deberían. El modelo M/M/1 dará respuestas con mayor seguridad. Un ejemplo de esta categoría podría ser una aplicación de introducción de datos desde un formulario concreto.
- *Cociente cercano a I*: Este es el caso más común y corresponde a un tiempo de servicio exponencial. Es decir, los tiempos de servicio son, en genial, aleatorios. Considérese la longitud de mensajes de un terminal de computador: Una pantalla completa puede tener 1920 caracteres y el tamaño de mensajes varía en todo ese rango. Como ejemplos de sistemas que se encuadran a menudo en esta categoría se tienen las reservas de líneas aéreas, consultas y búsquedas en archivos, redes locales compartidas y redes de conmutación de paquetes.

TABLA A.3 Fórmulas para Colas de un Solo Servidor

Supuestos:
1. Tasa de llegadas según Poisson.
2. La política de expedición no da preferencia a los individuos en función de los tiempos de servicio.
3. Las fórmulas para la desviación típica suponen una expedición del tipo primero en llegar, primero en servirse.
4. Los individuos no abandonan la cola (se retrasan las llamadas perdidas).

(a) Tiempos de Servicio Generales (M/G/1)

$$A = \frac{1}{2}\left[1 + \left(\frac{\sigma_s}{s}\right)^2\right] \qquad \text{parámetro útil}$$

$$q = \rho + \frac{\rho^2 A}{1-\rho}$$

$$w = \frac{\rho^2 A}{1-\rho}$$

$$t_q = s + \frac{\rho s A}{1-\rho}$$

$$t_w = \frac{\rho s A}{1-\rho}$$

(b) Tiempos de Servicio Exponenciales (M/M/1)

$$q = \frac{\rho}{1-\rho}$$

$$w = \frac{\rho^2}{1-\rho}$$

$$t_q = \frac{s}{1-\rho}$$

$$t_w = \frac{\rho s}{1-\rho}$$

$$\sigma_q = \frac{\sqrt{\rho}}{1-\rho}$$

$$\sigma_{tq} = \frac{s}{1-\rho}$$

$$\Pr[q = N] = (1-\rho)\rho^N$$

$$\Pr[q \le N] = \sum_{i=0}^{N} (1-\rho)\rho^i$$

$$\Pr[t_q \le t] = 1 - e^{-t(1-\rho)/s}$$

$$m_{tq}(r) = t_q \times \log_e\left(\frac{100}{100-r}\right)$$

$$m_{tw}(r) = \frac{t_w}{\rho} \times \log_e\left(\frac{100\rho}{100-r}\right)$$

(c) Tiempos de Servicio Constantes (M/D/1)

$$q = \frac{\rho^2}{2(1-\rho)} + \rho$$

$$w = \frac{\rho^2}{2(1-\rho)}$$

$$t_q = \frac{s(2-\rho)}{2(1-\rho)}$$

$$t_w = \frac{\rho s}{2(1-\rho)}$$

$$\sigma_q = \frac{1}{(1-\rho)} R$$

$$R = \sqrt{\rho - \frac{3\rho^2}{2} - \frac{5\rho^3}{6} - \frac{\rho^4}{12}}$$

$$\sigma_{tq} = \frac{s}{1-\rho}\sqrt{\frac{\rho}{3} - \frac{\rho^2}{12}}$$

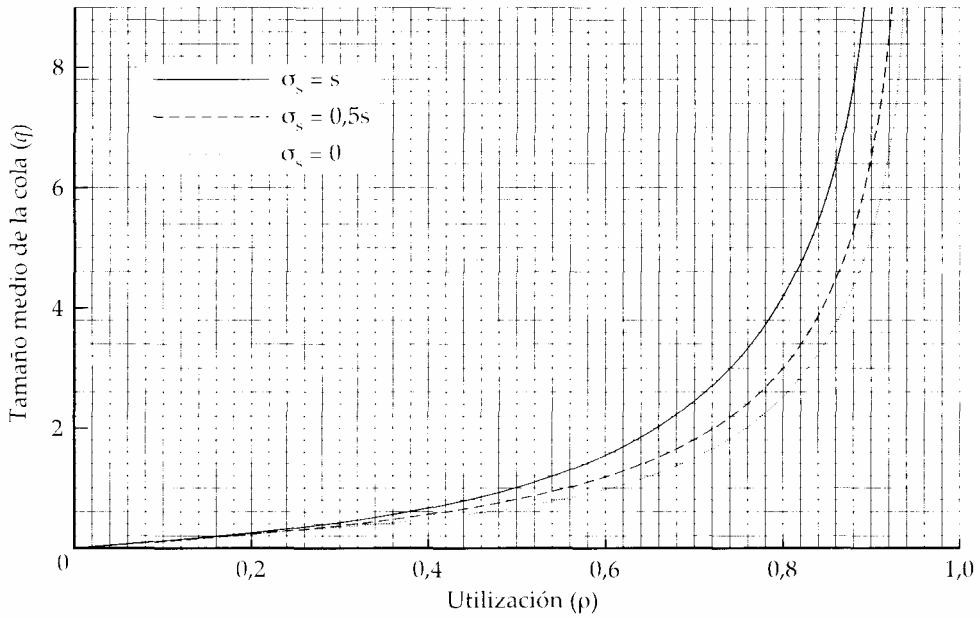


FIGURA A.4 Tamaño medio de la cola para el modelo M/G/1

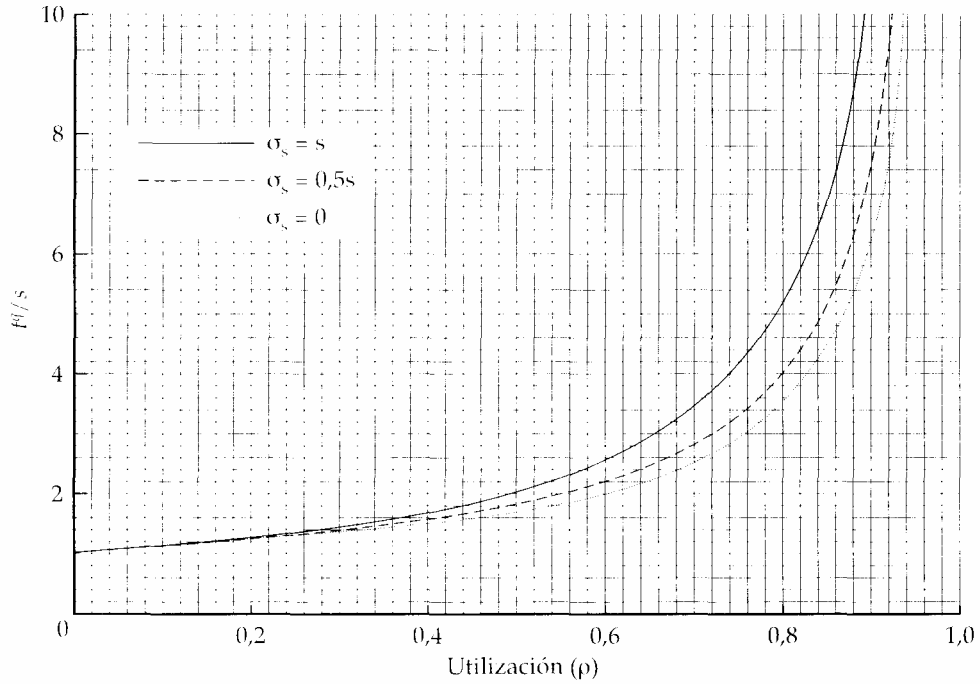


FIGURA A.5 Tiempo medio en cola para el modelo M/G/1

• *Cociente mayor que 1:* Si se da este caso, hay que usar el modelo M/G/1 y no el M/M/1. La situación más común es la de una distribución bimodal, con picos muy dispersos. Un ejemplo es el de un sistema que reciba muchos procesos cortos, muchos procesos largos y pocos entre medias.

De pasada, se aplica la misma consideración a la tasa de llegadas. Para una tasa de llegadas de Poisson, los tiempos entre llegadas son exponenciales y el cociente entre la desviación típica y la media es de 1. Si el cociente observado es mucho menor que I, las llegadas tienden a estar uniformemente espaciadas (sin demasiada variabilidad) y la suposición de Poisson sobrestimará los tamaños de cola y los retardos. Por otro lado, si el cociente es mayor que 1, las llegadas tienden a agruparse y se agudizará la congestión.

A.4

COLAS MULTISERVIDOR

La tabla A.4 enuncia las fórmulas de algunos parámetros clave en el caso multiservidor. Nótese lo restrictivo de las suposiciones. Sólo se han obtenido estadísticas de congestión útiles para este modelo en el caso M/M//V, donde los tiempos de servicio exponenciales son iguales para los *N* servidores.

A.5

REDES **DE COLAS**

En un entorno distribuido, las colas aisladas no son, por desgracia, el único problema que se le presenta al analista. A menudo, el problema a analizar está formado por varias colas interconectadas. La figura A.6 representa esta situación, empleando nodos para representar las colas y líneas de interconexión para representar el fluir del tráfico.

Los dos elementos que complican los métodos en estas redes son:

• La división y mezcla del tráfico, como se muestra en los nodos 1 y 5 de la figura. respectivamente.

• La existencia de colas en tándem o en serie, como se muestra en los nodos 3 y 4.

No se ha desarrollado ningún método exacto para analizar problemas generales de colas que dispongan de los dos elementos anteriores. Sin embargo, si el fluir del tráfico sigue una Poisson y los tiempos de servicio son exponenciales, existe una solución exacta y simple. En esta sección, se examinarán en primer lugar los dos elementos enunciados, para presentar después un enfoque de análisis de colas.

División y mezcla de flujos de tráfico

Supóngase que llega tráfico a la cola con una tasa media de llegadas de Á. y que hay dos caminos, A y B, por los cuales puede salir un individuo (figura A.7a). Cuando se da servicio a un individuo y éste sale de la cola, lo hace por el camino A con una probabilidad P y por B con una probabilidad (I - P). En general, la distribución de tráfico de A y B será distinta

TABLA A.4 Fórmulas para Colas Multiservidor (M / M / N)

Supuestos:
1. Tasa de llegadas según Poisson.
2. Tiempos de servicio exponenciales.
3. Todos los servidores tienen una carga pareja.
4. Todos los servidores tienen el mismo tiempo medio de servicio.
5. Expedición primero en entrar/primero en salir.
6. Ningún individuo abandona la cola.

$$K = \frac{\sum_{N=0}^{M-1} \frac{(M\rho)^N}{N!}}{\sum_{N=0}^{M} \frac{(M\rho)^N}{N!}}$$ parámetro útil

Probabilidad de que todos los servidores estén ocupados $= B = \frac{1 - K}{1 - \rho K}$

$$q = B \frac{\rho}{1 - \rho} + M\rho$$

$$w = B \frac{\rho}{1 - \rho}$$

$$t_q = \frac{B}{M} \frac{s}{1 - \rho} + s$$

$$t_w = \frac{B}{M} \frac{s}{1 - \rho}$$

$$\sigma_{tq} = \frac{s}{M(1 - \rho)} \sqrt{B(2 - B) + M^2(1 - \rho)^2}$$

$$\sigma_w = \frac{1}{1 - \rho} \sqrt{B\rho(1 + \rho - B\rho)}$$

$$PR[t_w > t] = Be^{-M(1-\rho)t/s}$$

$$t_d = \frac{s}{M(1 - \rho)}$$

de la distribución de las llegadas. Sin embargo, si la distribución de las llegadas es una Poisson, entonces los dos flujos de tráfico de salida siguen también una Poisson, con una tasa media de $P\lambda$ y $(1 - P)\lambda$.

Se da una situación similar en la mezcla del tráfico. Si se mezclan dos flujos que siguen una Poisson con tasas medias de λ_1 y λ_2, el flujo resultante sigue una Poisson con una tasa media de de $\lambda_1 + \lambda_2$.

Ambos resultados se pueden generalizar a más de dos flujos de salida para el caso de la división y a más de dos flujos de entrada para el caso de la mezcla.

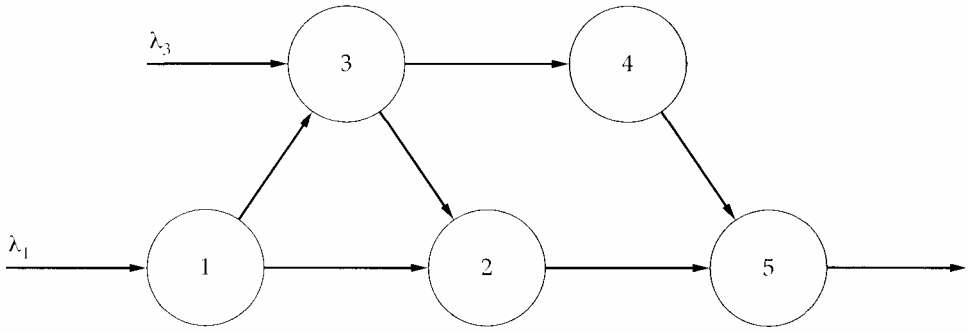


FIGURA A.6 Ejemplo de red de colas

Colas en tándem

La figura A.7c es un ejemplo de un conjunto de colas monoservidor en tándem: La entrada de cada cola, excepto la de la primera, es la salida de la anterior. Supóngase que la entrada de la primera cola es una Poisson. Entonces, si el tiempo de servicio de cada cola es exponencial y las tilas de espera son infinitas, la salida de cada cola es un flujo Poisson estadísticamente idéntico a la entrada. Cuando este flujo alimenta a la cola siguiente, los retardos de la segunda cola son los mismos que si el tráfico original hubiera evitado la primera cola y alimentase directamente a la segunda. Así pues, las colas son independientes y se pueden analizar de una en una. Por tanto, el retardo medio total del sistema en tándem es igual a la suma de los retardos medios de cada etapa.

Este resultado puede extenderse al caso en el que uno o más nodos del tándem sean colas multiservidor.

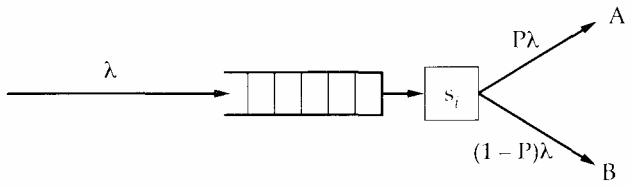
Teorema de Jackson

El teorema de Jackson sirve para analizar una red de colas. El teorema se basa en tres supuestos:

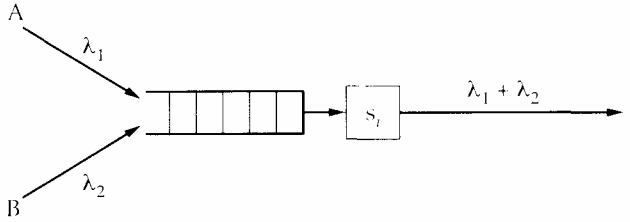
1. La red de colas consta de *m* nodos, cada uno de los cuales ofrece un servicio exponencial independiente.
2. Los individuos llegan del exterior del sistema a cualquiera de los nodos con una tasa de Poisson.
3. Una vez servido en un nodo, un individuo se dirige (inmediatamente) a uno de los nodos restantes con una probabilidad tija o bien abandona el sistema.

El teorema de Jackson afirma que. en una red de colas como la descrita, cada nodo es un sistema de colas independiente, con una entrada de Poisson determinada por los principios de división, mezcla y colas en tándem. Así pues, cada nodo puede analizarse independientemente de los demás mediante un modelo M/M/1 ó *M/M/N y* los resultados pueden combinarse mediante métodos estadísticos convencionales. Los retardos medios de cada nodo pueden sumarse para deducir el retardo del sistema, pero no puede decirse nada acerca de los instantes de mayor retardo (por ejemplo, de la desviación típica).

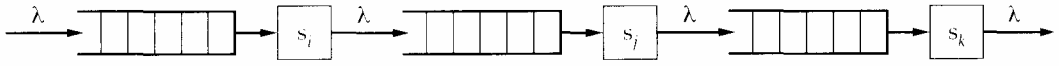
La aplicación del teorema de Jackson resulta atractiva en redes de conmutación de paquetes. Se puede modelar la red de conmutación de paquetes como una red de colas. Cada paquete representa un elemento individual. Se supone que cada paquete se transmite por sepa-



(a) División del tráfico



(b) Mezcla del tráfico



(c) Cola en tándem simple

FIGURA A.7 Elementos de las redes de colas

rárlo y que, en cada nodo de conmutación del camino desde el origen al destino, el paquete se pone en cola para su transmisión en el tramo siguiente. El servicio en una cola es la transmisión real del paquete y es proporcional a la longitud del mismo.

La pega de esta solución es que viola una condición del teorema, a saber, no es el caso de distribuciones de servicio independientes. Como la longitud de un paquete es la misma en cada enlace de transmisión, el proceso de llegadas a cada cola está correlacionado con el proceso de servicio. Sin embargo. Kleinrock [KLEI76] ha demostrado que, debido al efecto de promedio en las mezclas y las divisiones, suponer unos tiempos de servicio independientes proporciona una buena aproximación.

Aplicación a redes de conmutación de paquetes

Considérese una red de conmutación de paquetes, formada por nodos interconectados por enlaces de transmisión, donde cada nodo actúa como la interfaz con cero o más sistemas conectados, cada uno de los cuales funciona como origen y destino del tráfico[1] La carga de trabajo externa que se le presenta a la red puede caracterizarse como:

$$\gamma = \sum_{j=1}^{N} \sum_{k=1}^{N} \gamma_{jk}$$

[1] Este estudio se basa en el desarrollo de [MOLL89].

donde
λ = carga de trabajo total en paquetes por segundo
λ_{jk} = carga de trabajo entre el origen j y el destino *k*

Puesto que un paquete puede recorrer más de un enlace entre el origen y el destino, la carga de trabajo interna total puede ser mayor que la carga propuesta:

$$\lambda = \sum_{i=1}^{I} \lambda_i$$

donde
λ = carga total en todos los enlaces de la red
λ_i = carga del enlace *i*

La carga interna dependerá del camino real que tomen los paquetes a través de la red. Supóngase que se dispone de un algoritmo de encaminamiento tal que la carga λ_i en los enlaces individuales puede determinarse a partir de la carga propuesta, λ_{jk} Para una ruta escogida en particular, se puede determinar el número medio de enlaces que atravesará un paquete a partir de dichos parámetros de carga. Un cierto razonamiento demuestra que la longitud media de todos los caminos viene dada por:

$$E \text{ [número de enlaces en un camino]} = \frac{\lambda}{\gamma}$$

Ahora, el objetivo es determinar el retardo medio *T* que sufre un paquete en la red. Con tal propósito, resulta útil aplicar la fórmula de Little (tabla A.2). Para cada enlace de la red, el número medio de individuos en la cola de dicho enlace viene dado por:

$$q_i = \lambda_i t_i$$

donde t_i es el retardo que queda aún por determinar de cada cola. Supóngase que se suman estas cantidades. Se obtendrá el número medio total de paquetes que esperan en todas las colas de la red. Ahora bien, resulta que la formula de Little también funciona con la suma.[4] Así pues, el número de paquetes que esperan en la red puede expresarse como γT. Combinando los dos resultados, se tiene:

$$T = \frac{1}{\gamma} \sum_{i=1}^{I} \lambda_i t_i$$

Para determinar el valor de *T,* hace falta determinar los valores de los retardos individuales t_i Puesto que se supone que cada cola puede tratarse como un modelo M/M/1 independiente. esto es fácil de calcular:

$$t_i = \frac{S_i}{1 - \rho_i} = \frac{S_i}{1 - \lambda_i S_i}$$

El tiempo de servicio s_i del enlace *i* es justo el producto de la velocidad de los datos en dicho enlace (en bits por segundo) y la longitud media del paquete (en bits). Para ser consistentes con la notación usada habitualmente, se denotarán estos valores como C_i y I/μ, respectivamente. En tal caso:

$$t_i = \frac{\frac{1}{\mu C_i}}{1 - \frac{\lambda_i}{\mu C_i}} = \frac{1}{\mu C_i - \lambda_i}$$

Reuniendo todos estos elementos, se puede calcular el retardo medio de los paquetes en-lados por la red:

$$T = \frac{1}{\gamma} \sum_{i=1}^{I} \frac{\lambda_i}{\mu C_i - \lambda_i}$$

A6

EJEMPLOS

A continuación, se mostrarán algunos ejemplos para hacerse una idea del empleo de esas ecuaciones.

Servidor de base de datos

Considérese una LAN con 100 computadores personales y un servidor que mantiene una base de datos común para una aplicación de consultas. El tiempo medio para que el servidor responda a una consulta es de 0,6 sg. y la desviación típica se estima igual que la media. En los momentos críticos. la tasa de consultas en la LAN alcanza las 20 por minuto. Se quieren hallar los siguientes valores:

- El tiempo medio de respuesta ignorando el coste extra de la línea.
- Si se considera que el máximo tiempo de respuesta aceptable es de 1,5 sg., ¿qué porcentaje de crecimiento puede producirse en la carga de mensajes antes de alcanzar el máximo?
- Si se experimenta una utilización mayor en un 20%, ¿el tiempo de respuesta se incrementará más o menos del 20%?

Se supone un modelo M/M/1, con el servidor de base de datos como servidor del modelo. Se ignora el efecto de la EAN, suponiendo que su contribución al retardo es despreciable. En utilización del servicio se calcula como:

$p = \lambda_s$ = (20 llegadas por minuto)(0,6 sg por transmisión)/(60 sg/min)
=0,2

El primer valor, el tiempo medio de respuesta, se calcula fácilmente:

$t_q = s/(1 - \rho)$
$= 0,6/(1 - 0,2) = 0,75$ sg.

El segundo valor es más difícil de obtener. En realidad, como se ha expresado, no hay respuesta porque existe una probabilidad distinta de cero de que algunos casos del tiempo de respuesta excedan de 1,5 sg. para cualquier valor de la utilización. En su lugar, se puede decir que se desearía que el 90% de todas las respuestas esté por debajo del 1,5 sg. Entonces. se puede usar la ecuación de la tabla A.3b:

$$m_{tq}(r) = t_q \times \log_e (100/(100 - r))$$
$$m_{tq}(90) = t_q(\log_e(10)) = \frac{s}{1 - \rho} \times 2,3 = 1,5 \text{ sg.}$$

Se obtiene .v = 0,6. Resolviendo se obtiene **ρ** = 0,08. De hecho, se tendría que disminuir del 20% al 8%[1] para situar el valor de 1,5 sg. en el percentil 90-ésimo.

La tercera parte de la cuestión es encontrar la relación entre el incremento en la carga y el tiempo de respuesta. Puesto que la utilización del servicio de 0,2 está en la parte plana de la curva, el tiempo de respuesta se incrementará más lentamente que la utilización. En este caso, si la utilización del servicio aumenta del 20% al 40%, lo que supone un incremento del 100%.. el valor decrece de 0,75 sg a 1,0 sg, lo que supone un incremento sólo del 33%,.

Multiprocesador fuertemente acoplado

En la sección 6.3 se estudió el uso de multiprocesadores fuertemente acoplados en un único sistema informático. Una de las decisiones de diseño tenía que ver con el hecho de si los procesos estaban dedicados a los procesadores. Si un proceso se asigna permanentemente a un procesador, desde su activación hasta su finalización, se tiene una cola a corto plazo separada para cada procesador. En este caso, un procesador puede estar libre, con la cola vacía, mientras que otro procesador tiene trabajo atrasado. Para impedir esta situación, se puede usar una cola común. Todos los procesos entran en una cola y se planifican en cualquier procesador disponible. Así pues, a lo largo de la vida de un proceso, este puede ejecutarse en diferentes procesadores en diferentes instantes.

Uno se puede hacer una idea de la aceleración del rendimiento que se logra por medio de una cola común. Considérese un sistema con cinco procesadores. El tiempo medio de procesador que se ofrece a un proceso mientras está en el estado de Ejecución es de 0,1 sg. Supóngase que la desviación típica observada del tiempo de servicio es de 0,094 sg. Como la desviación típica es cercana a la media, se supone un tiempo de servicio exponencial. Supóngase también que los procesos llegan al estado de Listos a razón de 40 por sg.

Enfoque monoservidor

Si los procesos se distribuyen uniformemente entre los procesadores, la carga de cada procesador es $40/5 = \lambda_s$ paquetes por segundo. Así pues:

$\rho = \lambda s$
$= 8 \times 0,1 = 0,8$

El tiempo en cola se calcula fácilmente:

$$tq = \frac{s}{1 - r} = \frac{0,1}{0,2} = 0,5 \text{ sg.}$$

Enfoque muItiservidor

Supóngase ahora que se mantiene una única cola de Listos para todos los procesadores. Se tiene una tasa de llegadas total de 40 procesos por segundo. Sin embargo, la utilización del servicio es aún de 0,8 (λ_s /M). Para calcular el tiempo en cola a partir de la fórmula de la tabla A.4, hace falta calcular primero B. Si no se tiene programado el parámetro, se puede buscar en una tabla bajo una utilización de servicio de 0,8 y 5 servidores para obtener B = 0,544. Sustituyendo:

$$t_q = (0,1) + \frac{(0,544)(0,1)}{5(1 - 0,8)} = 0,1544$$

Así pues, el empleo de una cola multiservidor ha reducido el tiempo medio en cola de 0.5 sg a 0,1544 sg, lo que constituye un factor de más que 3. Si se considera sólo el tiempo de espera, en el caso multiservidor es de 0,0544 sg, frente a los 0,4 sg, lo que constituye un factor de 7.

Aunque vd. no sea un experto en teoría de colas, ahora sabe lo bastante para irritarse cuando tenga que esperar en un servicio de varias colas de un servidor.

Cálculo de percentiles

Considérese una configuración en la cual los mensajes se envían desde unos computadores de una red de área local (LAN) a sistemas de otras redes. Todos los mensajes deben pasar por un computador que conecta la LAN a una red de área extensa y, por tanto, al exterior. Este sistema de enlace se denomina normalmente encaminador *(router)* o pasarela (*gateway).* Considérese el tráfico de la LAN a través del encaminador. Los mensajes llegan a una tasa media de 5 por segundo. La longitud media de los mensajes es de 144 octetos y se supone que la longitud de los mensajes está distribuida exponencialmente. La velocidad de la línea que va desde el encaminador hasta la red de área extensa es de 9600 bps. Se plantean las siguientes preguntas:

1. ¿Cuál es el tiempo medio en cola en el encaminador?
2. ¿Cuántos mensajes están, en promedio, en el centro encaminador, incluyendo aquellos que esperan transmisión y el que esté transmitiéndose actualmente (si lo hay)?
3. La misma pregunta de (2), para el percentil 90-ésimo.
4. La misma pregunta de (2), para el percentil 95-ésimo.

λ = 5 mcns/sg

s = (144 octetos x 8 bits/octeto) / 9600 bps =0,12 sg.

$\rho = \lambda_s$ =5 x 0,12= 0,6

Tiempo medio en cola:

$t_q = s/(1 - \rho) = 0,3$ sg

Longitud media de la cola:

$q = \rho /(I - \rho) = 1.5$ paquetes

Para obtener los percentiles, se emplea la ecuación de la tabla A.3b:

$$\Pr[q = N] = (1 - \rho)\rho^N$$

Para calcular el percentil r-ésimo del tamaño de cola, se escribe la ecuación anterior en forma acumulativa:

$$\frac{r}{100} = \sum_{k=0}^{m(r)} (1 - \rho)\rho^k = 1 - \rho^{1+m(r)}$$

Aquí, *m(r)* representa el número máximo de mensajes en la cola que se espera estén dentro del *r* % del tiempo. En la forma dada, se puede determinar el percentil para cualquier tamaño de cola. Pero se desea hacer lo contrario: dado *r*, encontrar *m(r),* por lo que, tomando logaritmos en base 10 a ambos lados de la igualdad:

$$m(r) = \frac{\log\left(1 - \frac{r}{100}\right)}{\log \rho} - 1$$

Si es fraccionario, se toma el siguiente entero mayor; si es negativo, se pone a 0. En el ejemplo, p = 0,6 y se desea encontrar *m*(90) y *m*(95):

$$m(90) = \frac{\log (1 - 0{,}90)}{\log (0{,}6)} - 1 = 3{,}5$$

$$m(95) = \frac{\log (1 - 0{,}95)}{\log (0{,}6)} - 1 = 4{,}8$$

Así pues, el 90% del tiempo hay menos de cuatro mensajes en la cola y el 95% del tiempo hay menos de cinco. Si se diseña con un criterio del percentil 95-ésimo, se debe ofrecer un buffer para almacenar al menos cinco mensajes.

A.7

OTROS MODELOS DE COLAS

Este apéndice se ha centrado en una clase de modelo de colas. En realidad, hay varios modelos, en función de dos factores clave:

- El modo en el que se gestionan los individuos bloqueados
- El número de orígenes del tráfico

Cuando un individuo llega a un servidor y lo encuentra ocupado o cuando llega a un sistema multiservidor y encuentra todos los servidores ocupados, se dice que el individuo está bloqueado. Los individuos bloqueados se pueden gestionar de varias formas. En primer lugar, el individuo puede ponerse en una cola esperando un servidor libre. Este método se conoce en la bibliografía de tráfico telefónico como *retrasar las llamadas perdidas,* aunque la llamada realmente no se

pierde. Por otra parte, se puede no ofrecer una fila de espera. Esto conduce, a su vez, a dos suposiciones sobre la acción del individuo. El individuo puede esperar una cantidad de tiempo aleatoria y después intentarlo de nuevo; esto se conoce como *eliminar las llamadas perdidas.* Si el individuo intenta la obtención del servicio de forma repetida y sin pausa, se conoce como *retener las llamadas perdidas.* El modelo de retraso de llamadas perdidas es el más adecuado para la mayoría de los problemas de computadores y comunicaciones. La eliminación de llamadas perdidas es, normalmente, la más apropiada en un entorno de conmutación telefónica.

El segundo elemento clave de un modelo de tráfico es si el número de orígenes se supone finito o infinito. Para un modelo de orígenes infinitos, se supone que hay una tasa de llegadas fija. Para el caso de fuentes finitas, la tasa de llegadas dependerá del número de fuentes ya empleadas. Así pues, si cada una de las L fuentes genera llegadas con una tasa λ/L, cuando el servicio de colas esté desocupado, la tasa de llegadas será de λ. Sin embargo, si hay K fuentes en el servicio de colas en un momento determinado, la tasa de llegadas instantánea en ese momento es de $\lambda(L- K)/L$. Los modelos de infinitos orígenes son sencillos de resolver. La suposición de orígenes infinitos es razonable cuando el número de fuentes es, al menos, de 5 a 10 veces la capacidad del sistema.

A.8 LECTURAS RECOMENDADAS

Quizá la referencia más útil que se puede adquirir es [MART72]. Pese a la antigüedad de este libro, es una fuente práctica y valiosa. El libro ofrece varios gráficos y tablas que se pueden usar para realizar un rápido análisis de colas. También ofrece una guía detallada para aplicar el análisis de colas, además de una serie de ejemplos resueltos.

Además del libro de Martin, hay varias publicaciones poco conocidas, pero disponibles, que son de gran ayuda práctica. [IBM71] es un tratado conciso y excelente del análisis de colas aplicado a problemas de comunicaciones y computadores, con muchos ejemplos, ¡unto a gráficas y tablas. [FRAN76] es una buena colección de tablas para varios modelos de colas. Una guía excelente para la aplicación práctica de la Estadística es [NBS63]; el libro contiene tablas, fórmulas y ejemplos que ayudan a determinar el proceso correcto para estimar los valores a partir de muestras y evaluar los resultados.

Las referencias anteriores son suficientes para aquellos que sólo deseen aplicar el análisis de colas. Para los que deseen ahondar más en el tema, hay disponible una multitud de libros. Algunos de los que más merecen la pena son los siguientes. Algunos textos que ofrecen un tratamiento de la teoría de colas y su aplicación a los computadores y las comunicaciones son [MOLL89] y [KOBA78]. [STUC85] es un tratado excelente que se centra en las comunicaciones y las redes. [COOP81] aborda el tema haciendo énfasis en los conceptos del análisis del tráfico telefónico. El estudio clásico de teoría de colas con aplicaciones a los computadores, con una discusión detallada sobre las redes de computadores, se encuentra en [KLEI75] y [KLEI76]. [CONW89] ofrece un buen resumen de los resultados de las colas, incluyendo los más recientes; el tratamiento matemático, sin embargo, resulta más bien duro de leer.

CONW89 CONWAY, A. y GEORGANAS, N. *Queueing Networks - Exact Computational Algorithms: A Unified Theory Based on Decomposition and Aggregation.* MIT Press, Cambridge, MA, 1989.

COOP81 COOPER, R. *Introduction to Queuing Theory*, 2ª ed. North Holland, Nueva York, 1981.

FRAN76 FRANKEL, T. *Tables for Traffic Management and Design*. abc Teletraining, Geneva, IL, 1976. (P.O. Box 537, Geneva, IL 60134).

IBM71 IBM CORP. *Analysis of Some Queuing Models in Real-Time Systems*. IBM Document GF20-0007, 1971. Disponible en centros de distribución de documentación de IBM.

KLEI75 KLEINROCK, L. *Queuing Systems, Volume I: Theory*. Wiley, Nueva York, 1975.

KLEI76 KLEINROCK, L. *Queuing Systems, Volume II: Computer Applications*. Wiley, Nueva York, 1976.

KOBA78 KOBAYASHI, H. *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*. Addison-Wesley, Reading, MA, 1978.

MART72 MARTIN, J. *Systems Analysis for Data Transmission*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

MOLL89 MOLLOY, M. *Fundamentals of Performance Modeling*. Macmillan, Nueva York, 1989.

NBS63 NATIONAL BUREAU OF STANDARDS. *Experimental Statistics*. NBS Handbook 91, 1963. (Disponible en las Oficinas de Publicaciones del Gobierno de los EE.UU., GPO Stock Nº 003-003-00135-0).

STUC85 STUCK, B. y ARTHURS, E. *A Computer and Communications Network Performance Analysis Primer*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

ANEXO A

CONCEPTOS BÁSICOS DE PROBABILIDADES Y ESTADÍSTICA

Medidas de probabilidad

Una variable aleatoria continua X puede describirse por su función de distribución o por su función de densidad:

$F(x) = \Pr[X \le x]$ función de distribución

$f(x) = \frac{d}{dx} F(x)$ función de densidad

Para una variable aleatoria discreta, su distribución de probabilidad viene caracterizada por

$P_X(k) = \Pr[X = k]$

A menudo se buscan algunas características de las variables aleatorias en vez de la distribución completa. Por ejemplo, el valor de la media:

$E[X] = \int_{-\infty}^{\infty} x f(x)\, dx$ caso continuo

$E[X] = \sum_{\text{todo k}} k\, Pr[x = k]$ caso discreto

Otras medidas útiles:

Momento de orden dos: $E[X^2] = E[X^2] = \int_{\infty}^{\infty} x^2 f(x)\, dx$ caso continuo

Momento de orden dos: $E[X^2] = E[X^2] = \sum_{\text{todo k}} k^2\, Pr[x = k]$ caso continuo

Varianza: $\text{Var}[X] = E[(X - E[X])^2] = E[X^2] - E^2[X]$

Desviación típica: $\sigma_X = \sqrt{\text{Var}[X]}$

La varianza y la desviación típica son medidas de dispersión de los valores alrededor de la media.

distribución exponencial (figuras A.8a y A.8b) viene dada por:

$$F(x) = 1 - e^{\mu x} \quad \text{distribución}$$
$$f(x) = \mu e^{-\mu x} \quad \text{densidad}$$

La distribución exponencial tiene la propiedad interesante de que su media es igual a su desviación típica:

$$E[X] = \sigma_X = \frac{1}{\mu}$$

Cuando se usa para referirse a un intervalo de tiempo, tal como un tiempo de servicio, esta distribución se conoce en ocasiones como *distribución aleatoria.* La razón es que, para un intervalo de tiempo que ya ha empezado, los instantes en que puede terminar el intervalo son igualmente probables.

Esta distribución es importante en la teoría de colas debido a que, con frecuencia, se puede suponer que el tiempo de servicio de un servidor en un sistema de colas es exponencial. En el caso de tráfico telefónico, el tiempo de servicio es el tiempo durante el que un abonado contacta con el equipo que le interesa. En una red de conmutación de paquetes, el tiempo de servicio es el tiempo de transmisión y, por tanto, es proporcional a la longitud del paquete. Es difícil dar una razón teórica firme por la que los tiempos de servicio deben ser exponenciales, pero el hecho es que en la mayoría de los casos son muy cercanos a exponenciales. Esto es bueno porque simplifica enormemente el análisis de colas.

Otra distribución importante es la de Poisson:

$$Pr[X = k] = \frac{\lambda^k}{k!} e^{-\lambda}$$

$$E[X] = \sigma_X = \lambda$$

La distribución de Poisson también es importante en el análisis de colas debido a que se puede suponer una pauta de llegadas de Poisson para poder desarrollar las ecuaciones de colas. Afortunadamente, la suposición de llegadas según Poisson suele ser válida.

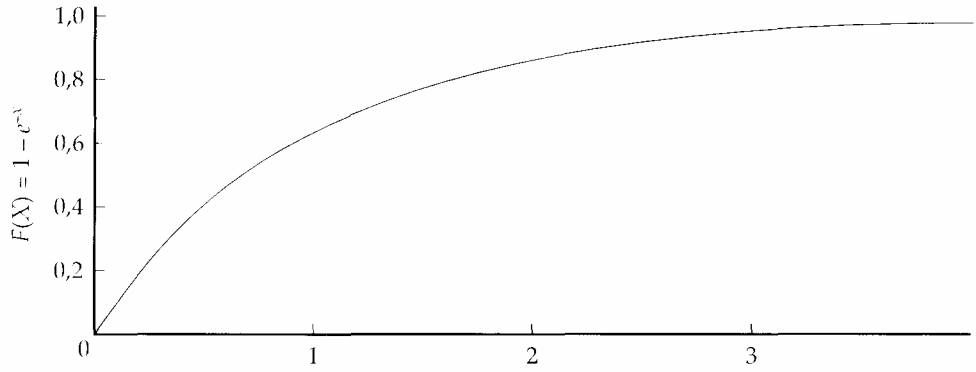
La forma en la que puede aplicarse la distribución Poisson a la tasa de llegadas es la siguiente. Si los individuos llegan a una cola de acuerdo a un proceso de Poisson, tal situación puede expresarse como:

$Pr[k$ individuos lleguen en el intervalo $T] = (\lambda T)^k/k!\ e^{-\lambda T}$

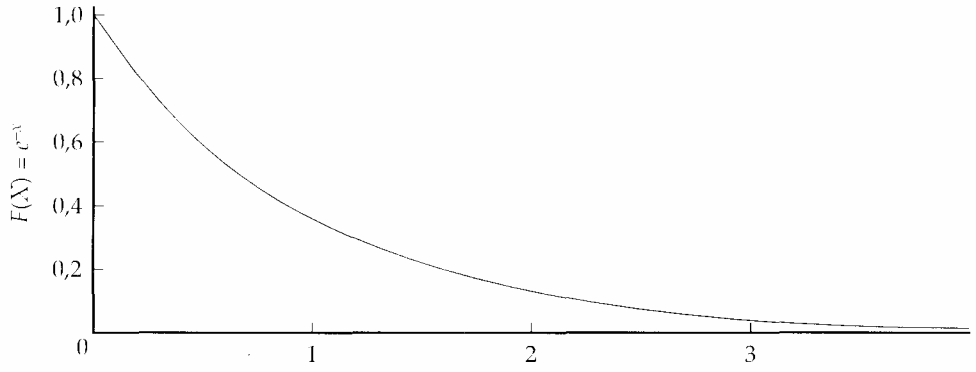
Número esperado de llegadas de individuos en el intervalo $T = \lambda T$

Tasa media de llegadas, en individuos por segundo $= \lambda$

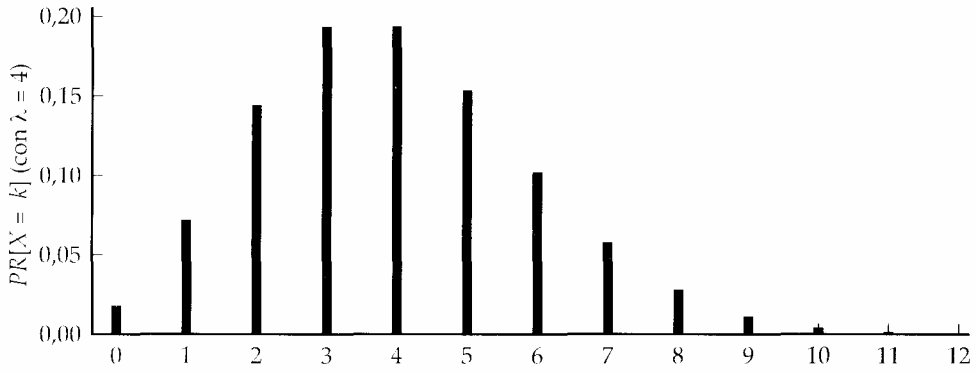
Las llegadas que se producen según un proceso de Poisson se suelen denominar *llegadas aleatorias* porque la probabilidad de la llegada de un individuo en un intervalo pequeño es proporcional a la longitud del intervalo y es independiente del tiempo transcurrido desde la llegada del último individuo. Es decir, cuando los individuos llegan según un proceso de Poisson, un individuo tiene la misma probabilidad de llegar en un instante que en otro, sin importar los instantes en que llegaron los otros individuos.



(a) Función de distribución de probabilidad exponencial



(b) Función de densidad de probabilidad exponencial



(c) Distribución de probabilidad de Poisson

FIGURA A.8 Algunas funciones de probabilidad

Otra propiedad interesante del proceso de Poisson es su relación con la distribución exponencial. Si se consideran los instantes entre llegadas de individuos *Ta* (llamados instantes entre llegadas), se descubre que esta cantidad obedece a una distribución exponencial:

$$Pr[Ta \le t] = 1 - e^{-\lambda t}$$

$$E[Ta] = \frac{1}{\lambda}$$

Así pues, el tiempo medio entre llegadas es el inverso de la tasa de llegadas, como se podía esperar.

Muestreo

Para llevar a cabo un análisis de colas, es necesario estimar los valores de los parámetros de la entrada, especialmente la media y las desviaciones típicas de la tasa de llegadas y del tiempo de servicio. Si se considera un sistema nuevo, estas estimaciones pueden hacerse en función de un juicio y una valoración de los equipos y las pautas de trabajo que probablemente prevalecerán. Sin embargo, es frecuente el caso en el que el sistema está disponible para su examen. Por ejemplo, un conjunto de terminales, computadores personales y host están interconectados en un edificio por conexiones directas y multiplexores, y se desean reemplazar las interconexiones con una red de área local. Para poder dimensionar la red, es posible medir la carga actual generada por cada dispositivo.

Las medidas se toman en forma de muestras. Un parámetro particular (por ejemplo, la tasa de paquetes generados por un terminal o el tamaño de los paquetes) se estima observando el número de paquetes generados durante un periodo.

Para estimar una cantidad, tal como la longitud de un paquete, se pueden usar las siguientes ecuaciones:

media muestreada: $$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

varianza muestreada: $$S^2 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}$$

$$= \frac{\sum_{i=1}^{n} X_i^2 - \left(\sum_{i=1}^{n} X_i\right)^2}{n(n-1)}$$

desviación típica muestreada: $$S = \sqrt{S^2}$$

donde

n = tamaño de la muestra

X_i = muestra i-ésima

Para estimar la tasa de llegadas de una muestra, se puede hacer la operación siguiente:

$$\bar{\lambda} = \frac{n}{T}$$

donde *n* es el número de individuos observados en un periodo de duración *T*. Otra solución consiste en considerar cada instante de llegada corno una muestra y calcular la media muestreada y la desviación típica muestreada, como se hizo anteriormente.

Cuando se estiman valores tales como la media y la desviación típica en función de una muestra, se abandona la estera de las Probabilidades y se entra en la de la Estadística. Este es un tema complejo que no se explorará aquí, excepto para ofrecer algunos comentarios.

Es importante darse cuenta de que la media muestreada y la desviación típica muestreada son variables aleatorias en sí mismas. Por ejemplo, si se toma una muestra de una cierta población, se calcula la media muestreada, y si se hace esto varias veces, los valores calculados diferirán. Así pues, se puede hablar de la media y de la desviación típica de la media muestreada o incluso hablar de la distribución de probabilidad de la media muestreada.

Se deduce que la naturaleza probabilística de los valores estimados es una fuente de error, conocido como *error de muestreo.* En general, cuanto mayor sea el tamaño de la muestra, menor será la desviación típica de la media muestreada y, por tanto, se está más cerca de que la estimación sea la media real. Mediante ciertos supuestos razonables sobre la naturaleza de la variable aleatoria que se comprueba y sobre la aleatoriedad del procedimiento de muestreo, se puede determinar la probabilidad de que una media muestreada o una desviación típica muestreada esté dentro de un cierto entorno de la media o la desviación típica reales. Este concepto se presenta a menudo junto con los resultados de un muestreo. Por ejemplo, es común incluir, junto con el resultado de una encuesta de opinión, un comentario del tipo: "El resultado está dentro de un 5% de valor de certeza, con un nivel de confianza [probabilidad] del 99%".

Hay, sin embargo, otra fuente de error, mucho menos apreciada entre los no estadísticos, que es el sesgo. Por ejemplo, si una encuesta de opinión está dirigida y sólo son entrevistados los miembros de un determinado grupo socioeconómico, los resultados no son necesariamente representativos de la población al completo. En el contexto de las comunicaciones, los muestreos realizados durante un momento del día pueden no reflejar la actividad de otros momentos. Si se quiere diseñar un sistema que gestione los picos de carga que es probable que se sufran, se debe observar el tráfico durante el momento del día en el que es más probable que se genere la carga mayor.

APÉNDICE B

Diseño orientado a objetos

Windows NT y algunos otros sistemas operativos están basados, en gran medida, en los principios del diseño orientado a objetos. Este apéndice ofrece una breve introducción a los conceptos principales del diseño orientado a objetos.

B.1 MOTIVACIÓN

Los conceptos de orientación a objetos se han convertido en algo bastante común en el campo de la programación de computadores, con la promesa de crear componentes de software intercambiables, reutilizables, fácilmente actualizables e interconectables. Más recientemente, los diseñadores de bases de datos han comenzado a apreciar las ventajas de la orientación a objetos, con el resultado del comienzo de la aparición de los sistemas gestores de bases de datos orientadas a objetos (OODBMS, *Object Oriented Data Base Management System).* Los diseñadores de sistemas operativos también han reconocido las ventajas del enfoque orientado a objetos.

La programación orientada a objetos y los sistemas gestores de bases de datos orientadas a objetos son, en realidad, cosas diferentes, pero comparten un concepto clave: que el software o los datos pueden ser "contenerizado". Todo está dentro de una caja y puede haber cajas dentro de otras cajas. En el programa convencional más sencillo, un paso de programa equivale a una instrucción; en un lenguaje orientado a objetos, cada paso podría ser un lote completo de instrucciones. Análogamente, en una base de datos orientada a objetos, una variable, en vez de equivaler a un solo elemento de datos, puede ser equivalente a un lote completo de datos.

La tabla B.1 introduce alguno de los términos clave empleados en el diseño orientado a objetos.

TABLA B.1 Términos Clave de la Orientación a Objetos

Término	Definición
Objeto	Una abstración de una entidad del mundo real.
Clase de Objeto	Un conjunto de objetos que comparten los mismos nombres, conjuntos de atributos y servicios.
Instancia de Objeto	Un miembro específico de una clase de objeto, con unos valores asignados a los atributos.
Atributo	Variables de datos incluidas dentro de un objeto.
Servicio	Una función que realiza una operación sobre un objeto.
Encapsulación	El aislamiento de los atributos y servicios de una instancia de objeto con respecto al entorno exterior. Sólo se puede llamar a los servicios por su nombre y sólo se puede acceder a los atributos por medio de los servicios.
Herencia	Una relación entre dos clases de objeto en la que una clase hija adquiere los atributos y servicios de una clase madre.
Contención	Una relación entre dos instancias de objetos en la que el objeto contenedor incluye un apuntador al objeto contenido.
Polimorfismo	Se refiere a la existencia de múltiples objetos que usan los mismos nombres para los servicios y presentan el mismo interfaz con el mundo exterior, pero que representan a distintos tipos de entidades.

B.2 CONCEPTOS DE ORIENTACIÓN A OBJETOS

El concepto fundamental del diseño orientado a objetos es el **objeto.** Un objeto es una unidad de software independiente que contiene un conjunto de datos y procedimientos relacionados. Generalmente, estos datos y procedimientos no son directamente visibles desde fuera del objeto. Más bien, existen unas interfaces bien definidas que permiten a otro software tener acceso a los datos y los procedimientos.

Estructura de un objeto

Los datos y los procedimientos contenidos en un objeto se denominan, generalmente, variables y métodos, respectivamente. Todo lo que un objeto "conoce" puede expresarse con sus variables y todo lo que puede hacer se expresa con sus métodos.

Las **variables** de un objeto son, normalmente, simples escalares o tablas. Cada variable tiene un tipo, posiblemente un conjunto de valores disponibles y pueden ser tanto constantes como variables (por convenio, se usa el término *variable* aún para las constantes). También se pueden imponer unas limitaciones de acceso sobre las variables a determinados usuarios, clases de usuarios o situaciones.

Los **métodos** de un objeto son procedimientos que pueden activarse desde fuera para llevar a cabo ciertas funciones. El método puede cambiar el estado del objeto, actualizar algunas de sus variables o actuar sobre un recurso exterior al que el objeto tiene acceso.

Los objetos interactúan por medio de **mensajes.** Un mensaje incluye el nombre del objeto emisor, el nombre del objeto receptor, el nombre de un método del objeto receptor y cualquier parámetro necesario para cualificar la ejecución del método. Nótese que el mensaje sólo puede usarse para hacer una llamada a un método de un objeto. La única forma de acceder a

los datos de dentro de un objeto es por medio de sus métodos. Así pues, un método puede provocar la ejecución de una acción, el acceso a las variables del objeto o ambas cosas.

La propiedad que tiene un objeto de que su única interfaz con el mundo exterior es por medio de mensajes se conoce como **encapsulación.** Las variables y los métodos de un objeto están encapsulados y disponibles sólo a través de la comunicación con mensajes. Esta propiedad ofrece dos ventajas:

1. Protege las variables de un objeto de la corrupción por parte de otros objetos. Esta pro lección puede serlo contra accesos no autorizados y para los tipos de problemas que surgen en el acceso concurrente, tales como el interbloqueo y los valores inconsistentes.
2. Oculta la estructura interna del objeto para que la interacción con él sea relativamente simple y estándar. Es más, si la estructura interna o los procedimientos de un objeto se modifican sin cambiar su funcionalidad externa, los otros objetos no se ven afectados.

Clases de objetos

Un objeto representa algo, bien sea una entidad física, un concepto, un módulo de software o una entidad dinámica tal como un circuito virtual. Los valores de las variables del objeto expresan la información que se conoce acerca de lo que el objeto representa. Los métodos incluyen procedimientos cuya ejecución influye en los valores del objeto y, posiblemente, también influyen en lo que éste representa.

En la práctica, habrá normalmente una serie de objetos que representan al mismo tipo de cosas. Por ejemplo, si un proceso está representado por un objeto, habrá un objeto por cada proceso presente en el sistema. Desde luego, cada uno de los objetos necesita su propio conjunto de variables. Sin embargo, si los métodos del objeto son procedimientos reentrantes, todos los objetos similares podrían compartir los mismos métodos. Es más, sería ineficiente volver a definir las variables y los métodos para cada objeto nuevo pero similar.

La solución a estas dificultades está en hacer una distinción entre una clase de objeto y una instancia de objeto. Una **clase de objeto** es una plantilla que define las variables y los métodos a incluir en un tipo de objeto en particular. Una **instancia de objeto** es un objeto real que incluye las características de la clase que lo define. La instancia contiene valores para las variables definidas en la clase de objeto.

Herencia

El concepto de clase de objeto es potente porque permite la creación de numerosas instancias de objetos con un mínimo de esfuerzo. Este concepto se hace aún más potente con el uso del mecanismo de herencia.

La herencia permite definir una nueva clase de objeto en términos de una clase existente. La nueva clase, llamada **subclase** incluye automáticamente las definiciones de variables y métodos de la clase original, llamada **superclase.** Una subclase puede diferir de su super-clase en varios sentidos:

1. La subclase puede incluir variables y métodos adicionales que no se encuentren en la superclase.
2. La subclase puede anular la definición de alguna variable o método de la superclase usando el mismo nombre con una nueva definición. Esto es una forma simple y eficiente de gestionar los casos especiales.

3. La subclase puede restringir de algún modo una variable o método heredado de su superclase.

El mecanismo de herencia es recursivo, permitiendo que una subclase se convierta en superclase de sus propias subclases. De esta manera, se puede construir **una jerarquía de herencia,** como se muestra en la figura B.1. Cualquier clase de objetos hereda todas las características de su superclase, incluyendo aquellas que la superclase heredó de más arriba en la jerarquía. Por ejemplo, la clase de objetos C-A2 incluye todos los métodos y variables de C-A que no están anulados en la definición de C-A2, mientras que la clase de objetos C-A2b incluye todos los métodos y variables definidos en C-A que no están anulados en C-A2 o en C-A2b, además de todas las variables definidas en C-A2 que no están anuladas en C-A2b.

Conceptualmente, se puede considerar la jerarquía de herencia como la definición de una técnica de búsqueda de variables y métodos. Cuando un objeto recibe un mensaje para ejecutar un método que no está definido en su clase, busca automáticamente hacia arriba en la jerarquía hasta encontrarlo. Análogamente, si la ejecución de un método genera una referencia a una variable no definida en una clase, el objeto busca ascendentemente en la jerarquía el nombre de la variable.

Polimorfismo

El polimorfismo es una característica potente y curiosa que hace posible ocultar distintas implementaciones bajo una interfaz común. Dos objetos que son polimórficos uno con otro utilizan los mismo nombres para los métodos y presentan la misma interfaz con los otros objetos. Por ejemplo, pueden haber varios objetos para imprimir en distintos dispositivos de salida, tales como imprimirMatricial, imprimirEaser, imprimirPantalla y así sucesivamente o para distintos tipos de documentos, tales como imprimirTexto, imprimirDibujo o impri-mirMixto. Si cada uno de los objetos incluye un método llamado *imprimir,* entonces cualquier documento podría escribirse enviando un mensaje *imprimir* al objeto apropiado, sin importar qué método se lleva a cabo realmente.

Es instructivo comparar el polimorfismo con las técnicas de programación modular convencionales. Un objetivo del diseño modular descendente es diseñar unos módulos de bajo nivel de uso general con una interfaz fija con los módulos de nivel superior. Esto permite que el modulo de nivel inferior sea invocado por muchos módulos diferentes de nivel superior. Si se cambia el interior del módulo de nivel inferior sin cambiar su interfaz, ninguno de los módulos de nivel superior que lo usan se verán afectados. En cambio, con el polimorfismo, el interés está en la capacidad del objeto de nivel superior para invocar a muchos objetos diferentes de nivel inferior, usando el mismo formato de mensaje para realizar funciones similares. Con el polimorfismo, se pueden añadir nuevos objetos de nivel inferior con cambios mínimos en los objetos existentes.

Contención

Las instancias de objeto que contienen otros objetos se denominan **objetos compuestos.** Se puede lograr la contención incorporando como valor en un objeto un apuntador al otro. La ventaja de los objetos compuestos es que permiten la representación de estructuras complejas. Por ejemplo, un objeto contenido dentro de un objeto compuesto puede ser a su vez un objeto compuesto.

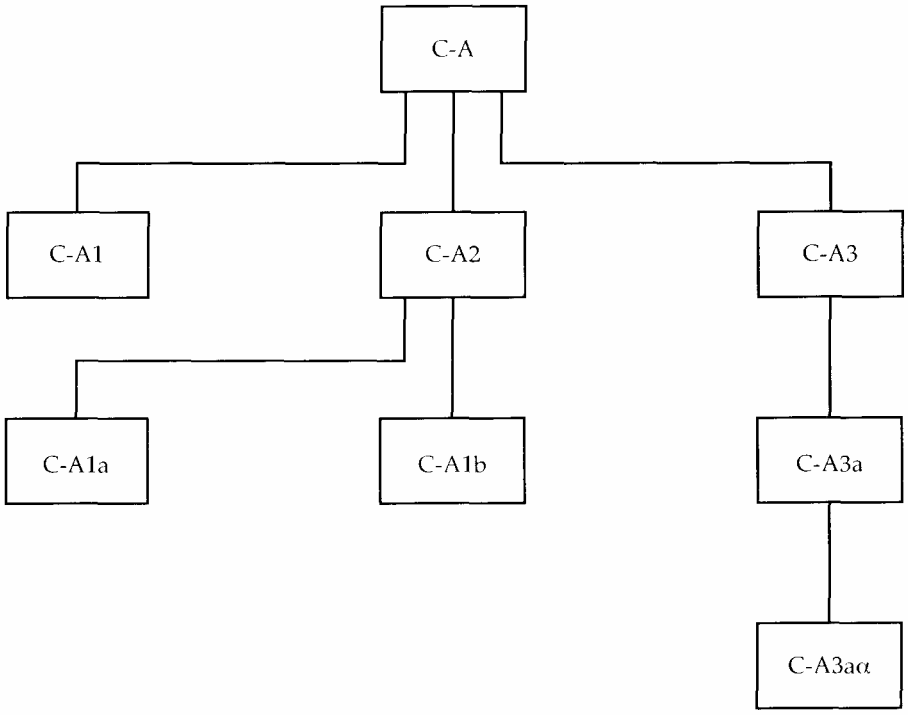


FIGURA B.1 Una jerarquía de herencia de clases de objetos

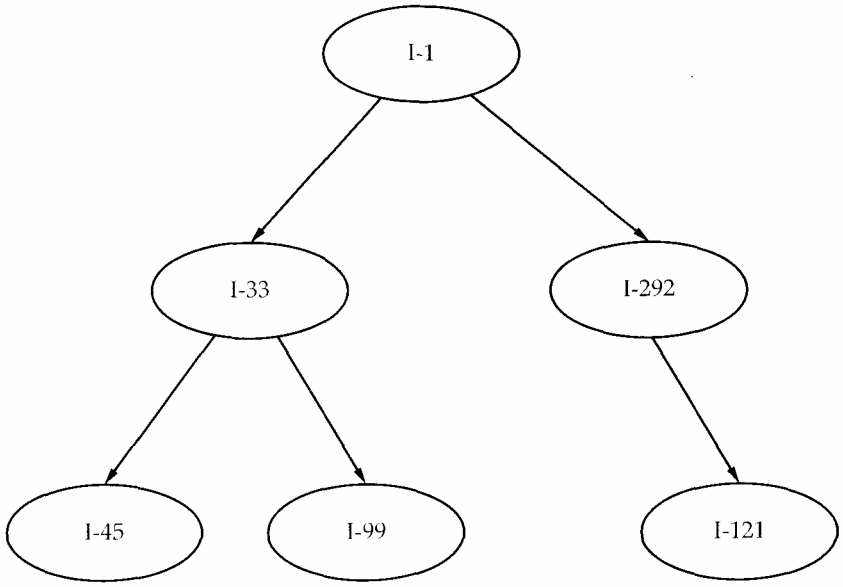


FIGURA B.2 Una jerarquía de contención de clases de objetos

Normalmente, las estructuras montadas sobre objetos compuestos están limitadas a una topología de árbol, es decir, no se permiten referencias circulares y cada instancia de objeto "hijo" puede tener sólo una instancia del objeto "padre". La figura B.2 ilustra el tipo de estructura jerárquica que se obtiene.

Es importante dejar clara la distinción entre jerarquía de herencia de clases de objetos y jerarquía de contención de instancias de objetos. Ambas no están relacionadas. El uso de la herencia simplemente permite definir varios tipos de objetos con un esfuerzo mínimo. El uso de la contención permite la construcción de estructuras de datos complejas.

B.3 VENTAJAS DEL DISEÑO ORIENTADO A OBJETOS

[CAST92] enuncia las siguientes ventajas del diseño orientado a objetos:

• *Una mejor organización de la complejidad inherente:* Gracias al uso de la herencia, pueden definirse de una forma más eficiente los conceptos afines, los recursos y otros objetos. Por medio del uso de la contención, pueden construirse estructuras de datos arbitrarias, que reflejan la tarea básica disponible. Los lenguajes de programación orientados a objetos y las estructuras de datos permiten a los diseñadores describir funciones y recursos del sistema operativo de forma que reflejen la visión que tiene el diseñador de esas funciones y recursos.

• *Reducción del esfuerzo de desarrollo mediante la reutilización:* La reutilización de clases de objetos que ya están escritas, probadas y mantenidas por otros reduce el tiempo de desarrollo, prueba y mantenimiento.

• *Sistemas mas ampliables y mantenibles:* El mantenimiento, incluidas las reparaciones y mejoras del producto, han consumido tradicionalmente cerca del 65% del coste de cualquier ciclo de vida de un producto. El diseño orientado a objetos reduce este porcentaje. El uso de ayudas de software basadas en objetos limita el número potencial de interacciones de las diferentes partes del software, garantizando que los cambios en la imple-mentación de una clase pueden hacerse con poco impacto en el resto del sistema.

Estas ventajas están encaminando el diseño de los sistemas operativos en la dirección de los sistemas orientados a objetos. Los objetos permiten a los programadores configurar a medida un sistema operativo para cumplir nuevos requisitos, sin alterar la integridad del sistema. Los objetos también permiten allanar el camino al proceso distribuido. Puesto que los objetos se comunican con mensajes, no importa si dos objetos que se comuniquen están en el mismo sistema o en sistemas diferentes de una red. Los datos, funciones e hilos pueden asignarse dinámicamente a estaciones de trabajo y servidores según sea necesario. Por consiguiente, la mayoría de los nuevos sistemas operativos para PC y estaciones de trabajo están adaptando un método de diseño orientado a objetos [WAYN94].

Glosario

Algunos de los términos de este glosario están tomados del *American Standard Dictionary for Information Systems* [ANSI90]. Estos términos se indican en el glosario con un asterisco.

Acceso Directo a Memoria Lina forma de E/S en la que un módulo especial, llamado módulo de DMA. controla el intercambio de datos entre la memoria principal y un dispositivo de E/S. El procesador envía una solicitud de transferencia de un bloque de datos al módulo de DMA y se ve interrumpido sólo después de que se haya transferido el bloque entero.

Acceso Directo* I,a capacidad de introducir u obtener datos de un dispositivo de almacenamiento en una secuencia independiente de su posición relativa, por medio de direcciones que señalan la ubicación física de los datos.

Acceso Indexado* Relativo a la organización y el acceso de los registros de una estructura de almacenamiento mediante un índice separado de las posiciones de los registros guardados.

Acceso Secuencial Indexado* Relativo a la organización y el acceso de los registros de una estructura de almacenamiento mediante un índice de las claves que se almacenan en unos archivos secuenciales divididos arbitrariamente.

Acceso Secuencial* I,a posibilidad de introducir datos en un medio o dispositivo de almacenamiento en la misma secuencia en que están ordenados los datos, o bien la de obtener datos en el mismo orden en que se introdujeron.

Almacenamiento Virtual* El espacio de almacenamiento que se puede considerar como almacenamiento principal direccionable por el usuario de un sistema informático en el que las direcciones virtuales se traducen a direcciones reales. El tamaño del almacenamiento virtual está limitado por el esquema de direccionamiento del sistema informático y por la cantidad de almacenamiento auxiliar disponible y no por el número de posiciones disponibles de almacenamiento principal.

Archivo de Dispersión Un archivo en el que se accede a los registros de acuerdo con los valores de un campo clave. Se emplea la dispersión para ubicar un registro en función del valor de su clave.

Archivo Indexado Un archivo en el que se accede a los registros de acuerdo con el valor de un campo clave. Hace falta un índice que indique la posición de cada registro a partir del valor de la clave.

Archivo Secuencial Indexado Un archivo en el que los registros se ordenan de acuerdo con los valores de un campo clave. El archivo principal se complementa con un archivo de índice que contiene una lista parcial de los valores de la clave: el índice da la posibilidad de búsqueda para llegar rápidamente a la vecindad de un registro deseado.

Archivo Secuencial* Un archivo en el que los registros están ordenados de acuerdo con los valores de uno o más campos clave y son procesados en la misma secuencia desde el comienzo del archivo.

Archivo* Un conjunto tic registros relacionados que se tratan como una unidad.

Arquitectura de Comunicaciones La estructura de hardware y software que implementa las funciones de comunicación.

Bacteria Programa que consume los recursos del sistema replicándose a sí misino.

Base de Datos* Un conjunto de datos interrelacionados. a menudo con una redundancia controlada, organizada de acuerdo con un esquema para dar servicio a una o mas aplicaciones: los datos se almacenan de forma que puedan ser utilizados por programas diferentes sin conocimiento de la organización o la estructura de los datos. Se emplea un método común para añadir dalos nuevos y para modificar y recuperar datos existentes.

Bloque de Control de Proceso La manifestación de un proceso en el sistema operativo. Es una estructura de datos que contiene información sobre las características y el estado del proceso.

Bloque* (1) Una colección de registros contiguos que se graban como una unidad, estando las unidades separadas por huecos. (2) Un grupo de bits que se transmiten como una unidad.

Buffer de Traducción Adelantada* Una cache de alta velocidad usada para guardar las entradas de la tabla de paginas a las que se haya hecho referencia recientemente, como parte de un esquema de memoria virtual. La TLB reduce la frecuencia de los accesos a memoria principal para recuperar entradas de la tabla de páginas.

Buzón Una estructura de dalos compartida entre una serie de procesos que se usa como un cola de mensajes. Los mensajes se envían al buzón y se recuperan del buzon en vez de pasar directamente del remitente al destinatario.

Caballo de Troya Rutina secreta no documentada, empotrada en un programa útil. 1.a ejecución del programa provoca la ejecución de la rutina secreta.

Cache de Disco Un almacenamiento intermedio, generalmente en memoria principal, que funciona como una cache de bloques de disco entre la memoria del disco y el resto de la memoria principal.

Cambio de Contexto Una operación del hardware que sucede cuando el programa que está ejecutando se ve interrumpido. Se salvaguardan el contador de programa. la palabra de estado del procesador, así como otros registros.

Campo* (1) Datos lógicos definidos que forman parte de un registro. (2) La unidad elemental de un registro que puede albergar un elemento de datos, una agregación de dalos, un puntero o un enlace.

Cepo* Un salto incondicional no programado a una dirección específica que es activada automáticamente por el hardware; se registra la posición desde la que se hace el salto.

Cifrado La conversión de texto o dalos en claro a una forma ininteligible por medio de cálculos matemáticos reversibles.

Compactación Una técnica empleada cuando la memoria está dividida en particiones de tamaño variable. *De* cuando en cuando, el sistema operativo desplaza las particiones para que queden contiguas y así toda la memoria libre este reunida en un solo bloque. Véase *Fragmentación Externa.*

Concurrente* Relativo a los procesos que tienen lugar en un intervalo común de tiempo durante el cual pueden tener que compartir recursos alternativamente.

Conjunto de Trabajo El conjunto de trabajo *W(t D)* con parámetro D de un proceso en el instante virtual *t*, es el conjunto de páginas de dicho proceso a las que se ha hecho referencia en las últimas *11* unidades de tiempo. Compárese con *Conjunto Residente.*

Conjunto Residente La parte de un proceso que está en memoria principal en un momento dado. Compárese con *Conjunto de Trabajo.*

Detección del Interbloqueo Una técnica en la que los recursos solicitados se conceden siempre que estén disponibles. Periódicamente, el sistema operativo comprueba si hay interbloqueo.

Dirección de Base* Una dirección empleada como origen en el cálculo de direcciones durante la ejecución de un programa de computador.

Dirección Física La posición absoluta de una unidad de datos en la memoria (por ejemplo, una palabra o un byte en memoria principal o un bloque en memoria secundaria).

Dirección Lógica Una referencia a una posición de memoria independiente de la asignación actual de dalos déla memoria. Se debe hacer una traducción a una dirección física antes de realizar el acceso a memoria.

Dirección Relativa* Una dirección calculada como un desplazamiento a partir de una dirección de base.

Dirección Virtual* La dirección de una posición del al macenamicnto virtual.

Dispersión La selección de una posición de almacenamiento de un elemento de datos mediante el cálculo de la dirección en función del contenido de los datos. Esta técnica complica la función de asignación del almacenamiento. pero da como resultado una rápida recuperación aleatoria.

Espacio de Direcciones* El rango de direcciones disponibles para un programa de computador.

Espera Activa Ejecución repetida de un bucle de código mientras se espera a que se produzca un suceso.

Expedir* Asignar tiempo del procesador a las tareas o trabajos que estén listos para su ejecución.

Fallo de Página Se produce cuando la página que contiene una palabra referenciada no está en memoria principal. Esto provoca una interrupción y exige que se traiga a la memoria la página adecuada.

Fraccionamiento del Tiempo* Un modo de operación en el que se asignan cuantos de tiempo del mismo procesador a dos o más procesos.

Fragmentación Externa Se produce cuando la memoria se divide en particiones de tamaño variable correspondientes a los bloques de dalos asignados de la memoria (por ejemplo, los segmentos de la memoria principal). A medida que se trasladan los segmentos dentro y lucra de la memoria principal, se producirán huecos entre las partes ocupadas de la memoria.

Fragmentación Interna Se produce cuando la memoria se divide en particiones de tamaño fijo (por ejemplo, marcos de página en la memoria principal o bloques físicos en el disco). Si un bloque de datos es asignado a una o más particiones, puede haber un espacio desaprovechado en la última partición. Esto se producirá si la última porción de los datos es más pequeña que la última partición.

Generación de Procesos La creación de nuevos procesos por parte de otros procesos.

Gestor de Interrupciones Una rutina, que generalmente forma parte del sistema operativo. Cuando se produce una interrupción, se transfiere el control al gestor de interrupciones correspondiente, quien lleva a cabo alguna acción como respuesta a la condición que originó la interrupción.

Gusano Un programa que puede viajar de computador en computador a través de las conexiones de red. Puede contener un virus o una bacteria.

Hilo La unidad de expedición. En la mayoría de los sistemas operativos, hay una correspondencia de uno a uno entre los procesos y los hilos. En algunos sistemas operativos. el proceso es la unidad de propiedad de recursos y el hilo representa la ruta de ejecución a través de uno o más programas.

Hiperpaginación Un fenómeno de los esquemas de memoria virtual en el que el procesador pasa la mayor parte del tiempo intercambiando tro/os de memoria en vez de ejecutando instrucciones.

Imagen de Proceso Todos los componentes de un proceso, incluyendo el programa, los datos, la pila y el bloque de control del proceso.

Inanición Una condición en la que un proceso se retarda indefinidamente porque otros procesos siempre tienen la preferencia.

Instrucción Privilegiada* Una instrucción que puede ejecutarse solamente en un modo específico, generalmente por un programa supervisor.

Interbloqueo* (1) Un punto muerto que se produce cuando varios procesos están esperando a que esté libre un recurso que no llegará a estar disponible porque está retenido por otro proceso que está en un estado de Espera análogo. (2) Un punto muerto que se produce cuando varios procesos están esperando una acción o una respuesta de otro proceso que está en un estado de Espera análogo.

Intercambio* Un proceso que intercambia el contenido de una zona del almacenamiento principal con el contenido de una zona del almacenamiento auxiliar.

Interfaz de Programas de Aplicación Una biblioteca estándar de herramientas de programación empleada por los desarrolladores de software para escribir aplicaciones que sean compatibles con un sistema operativo específico o con una interfaz gráfica de usuario.

Interrupción* Una suspensión de un proceso, tal como la ejecución de un programa de computador, originada por un suceso externo a dicho proceso y llevada a cabo de forma que el proceso pueda reanudarse.

Interrupciones Habilitadas Una condición, generalmente originada por el sistema operativo, en que el procesador responde a las señales de solicitud de interrupción de un tipo específico.

Interrupciones Inhabilitadas Una condición, generalmente originada por el sistema operativo, en la que el procesador ignorará las señales de solicitud de interrupción de un tipo específico.

Lenguaje de Control de Trabajos* Un lenguaje orientado al problema, diseñado para expresar sentencias de un trabajo que se emplean para identificar el trabajo o para describir sus requisitos ante el sistema operativo.

Lista Encadenada* Una lista en la que los elementos de datos pueden estar dispersos, pero donde cada elemento contiene un identificador para localizar el elemento siguiente.

Llamada a Procedimiento Remoto Una técnica por la que dos programas de máquinas diferentes pueden interactuar utilizando la sintaxis y la semántica de llamadas y retornos de los procedimientos. Tanto el programa llamado como el llamador se comportan como si el programa asociado estuviera ejecutando en la misma máquina.

Marco de Página* Una zona del almacenamiento principal empleada para guardar una página.

Memoria Cache memoria más pequeña y más rápida que la memoria principal y que se sitúa entre el procesador y la memoria principal. La cache actúa como un almacén intermedio de las posiciones de memoria usadas recientemente.

Memoria Principal Memoria interna del sistema informático. con direcciones accesibles por los programas y que puede cargarse en los registros para su posterior ejecución o procesamiento.

Memoria Secundaria La memoria ubicada fuera del sistema informático, incluyendo discos y cintas.

Mensaje Un bloque de información que puede intercambiarse cutre los procesos como medio de comunicación.

Método de Acceso* El método que se emplea para encontrar un archivo, un registro o un conjunto de registros.

Migración de Procesos 1.a transferencia de una cantidad suficiente del estado de un proceso desde una máquina hasta otra, para que el proceso ejecute en la máquina de destino.

Modelo de Referencia de Interconexión de Sistemas Abiertos Un modelo de comunicaciones entre dispositivos que cooperan. Define una arquitectura de siete niveles de funciones de comunicación.

Multiprocesador de Acceso a Memoria No Uniforme

Un multiprocesador con memoria compartida en el que el tiempo de acceso desde un procesador dado a una palabra de memoria varía según la posición de la palabra de memoria.

Multiprocesador* Un computador que tiene dos o más procesadores que disponen de acceso común a un almacenamiento principal.

Multiproceso Simétrico* Un método de multiproceso que permite que el sistema operativo se ejecute en cualquier procesador disponible de entre varios procesadores que estén disponibles simultáneamente.

Multiproceso* Modo de operación que incorpora el procesamiento en paralelo por parte de dos o más procesadores de un multiprocesador.

Multiprogramación* Un modo de operación que permite la ejecución intercalada de dos o más programas de computador en un solo procesador

Multitarea* Un modo de operación que permite la ejecución concurrente o intercalada de dos o más tareas en una computador.

Nivel de Multiprogramación El número de procesos que residen parcial o totalmente en la memoria principal.

Núcleo Una parle del sistema operativo que incorpora el software que más se usa. En general, el núcleo se mantiene permanentemente en memoria principal.

Operación Asincrona* Una operación que se realiza sin relación temporal predecible con un suceso específico. como por ejemplo, la llamada a una rutina de diagnóstico de errores que pueda recibir el control en cualquier momento durante la ejecución de un programa de computador.

Operación Síncrona* Una operación que se produce regularmente o de manera predecible con respecto al acontecimiento de un suceso determinado de otro proceso. Como, por ejemplo, la llamada a una rutina de entrada/salida que recibe el control en una posición codificada previamente de un programa.

Organización de Archivos* Ordenación física de los registros de un archivo, determinada por el método de acceso empleado para guardarlos y recuperarlos.

Página* En el almacenamiento virtual, es un bloque de longitud lija que dispone de una dirección virtual y que se transfiere entre el almacenamiento real y el auxiliar.

Paginación por Demanda* Transferencia de una página del almacenamiento auxiliar hacia el almacenamiento real en el momento en que se necesite. Compárese con *paginación por demanda.*

Paginación Previa La recuperación de páginas distintas de la solicitada por un fallo de página. La esperanza en que se necesiten páginas adicionales en el futuro cercano, sin aumentar la L/S con el disco. Compárese con *paginación por Demanda.*

Paginación* La transferencia de páginas entre el almacenamiento real y el auxiliar.

Partición de Memoria* La subdivisión del almacenamiento en secciones independientes.

Pila* Una lista que se construye y se mantiene de forma que el siguiente elemento de datos a recuperar sea el almacenado hace menos tiempo. Este método está caracterizado por un "último en entrar, primero en salir".

Planificación por Grupos La planificación de una serie de hilos afines para que ejecuten en un conjunto de procesadores al mismo tiempo, en una relación de uno a uno.

Planificar* Seleccionar trabajos o tareas que se vayan a expedir. En algunos sistemas operativos también se pueden planificar otras unidades de trabajo, como las operaciones de entrada/salida.

Predicción del Interbloqueo Una técnica dinámica que comprueba si cada petición de recurso provoca interbloqueo. Si la nueva petición puede conducir a un interbloqueo, se deniega la petición.

Prevención del Interbioqueo Una técnica que garantiza que no se producirá interbloqueo. La prevención se consigue asegurando que no se cumpla una de las condiciones necesarias para el interbloqueo.

Primero en Entrar, Primero en Salir* Una técnica de colas en la que el elemento siguiente que se recupera es el elemento que ha permanecido en la cola durante más tiempo.

Procedimiento Reentrante* Una rutina a la que se puede entrar antes de terminar una ejecución anterior de la misma rutina y que ejecuta correctamente.

Proceso por Lotes* Relativo a la técnica de ejecución de un conjunto de programas de computador de forma que se completa cada uno antes de empezar el siguiente.

Proceso Un programa en ejecución. Un proceso es controlado y planificado por el sistema operativo, es lo mismo que una *tarea.*

Recurso Consumible Un recurso que se puede crear (producir) y destruir (consumir). Cuando un recurso es adquirido por un proceso, el recurso deja de existir. Ejemplos de recursos consumibles son las interrupciones. las señales, los mensajes y la información de los *buffers".* de E/S.

Recurso Reutilizable Un recurso que puede ser usado deforma segura por un solo proceso en un instante y que no se agola por dicho uso. Los procesos adquieren unidades de recursos reutilizables que después liberan para que las reutilicen otros procesos. Algunos ejemplos de recursos reutilizables son los procesadores, los canales de E./S. la memoria principal y secundaria, los dispositivos y las estructuras de datos tales como los archivos, las liases de dalos y los semáforos.

Registro Lógico* Un registro independiente del entorno físico: varias parles de un registro lógico pueden ubicarse en registros tísicos diferentes o también pueden ubicarse varios registros lógicos o partes de un registro lógico en un mismo registro físico.

Registro* Una agrupación de elementos de dalos que se tratan como una unidad.

Rendezvous En el paso de mensajes, la situación en la que lanío el emisor como el receptor de un mensaje se quedan bloqueados hasta que se entregue el mensaje.

Reubicación Dinámica* Proceso de asignación de nuevas direcciones absolutas a los programas durante su ejecución, de forma que éstos puedan ejecutarse desde zonas diferentes del almacenamiento principal.

Sección Crítica* En un procedimiento asincrono de un programa de computador, es una parte que no puede ejecutarse simultáneamente con la sección crítica asociada a otro procedimiento asincrono. Véase *Exclusión Mutua.*

Segmentación La división de un programa o aplicación en segmentos como parte del esquema de memoria.

Segmento En la memoria virtual, un bloque que tiene una dirección virtual. Los bloques de un programa puede ser de longitud desigual y pueden ser incluso de longitud variable dinámicamente.

Seguridad Multinivel **La capacidad de aplicar un control de acceso a través de varios niveles de clasificación de los datos.**

Semáforo Binario **Un semáforo que sólo puede tomar los valores O ó I.**

Semáforo Un valor entero usado para la señalización entre procesos. Sólo se pueden realizar tres operaciones sobre un semáforo, ledas las cuales son atómicas: inicializar, decrementar e incrementar. Dependiendo de la definición precisa del semáforo, la operación de decremento puede originar el desbloqueo de un proceso.

Servidor* En una red, una estación de datos que proporciona servicios a otras estaciones: por ejemplo, un servidor de archivos, un servidor de impresión o un servidor de correo.

Sesión Un conjunto de uno o más procesos que representan a una sola aplicación interactiva de usuario o a una función del sistema operativo. Toda la entrada del teclado y del ratón es dirigida a una sesión interactiva y toda la salida de la sesión interactiva es dirigida a la pantalla de visualización.

Shell La parte del sistema operativo que interpreta las órdenes interactivas del usuario y las órdenes del lenguaje de control de trabajos. Funciona como una interfaz entre el usuario y el sistema operativo.

Sistema de Confianza Un computador y un sistema operativo que se pueden verificar para implementar una política de seguridad dada.

Sistema Gestor de Archivos **Un conjunto de software del sistema que da servicio a los usuarios v las aplicaciones para el uso de archivos, incluyendo el acceso a los archivos, el mantenimiento de directorios y el control de acceso.**

Sistema Operativo de Red El software complementario del sistema operativo que ofrece soporte liara el uso de sistemas servidores comunes en una red de computadores.

Sistema Operativo de Tiempo Real Un sistema operativo que debe planificar y gestionar tareas de tiempo real.

Sistema Operativo Distribuido Un sistema operativo común, compartido por una red de computadores. El sistema operativo distribuido ofrece soporte para la comunicación entre procesos, la migración de procesos, la exclusión mutua y la prevención o detección del interbloqueo.

Sistema Operativo* Software *que* controla la ejecución de programas y ofrece servicios tales como la asignación tic recursos, la planificación, el control de la entrada/salida y la gestión de los dalos.

Spooling* El empleo de un almacenamiento auxiliar como *buffer* para reducir el retardo del procesamiento cuando se transfieren dalos entre los equipos periféricos y los procesadores de un computador.

Tabla de Asignación de Archivos **Lina tabla que indica la posición física en el almacenamiento secundario del espacio asignado a un archivo. Hay una tabla de asignación de archivos para cada archivo.**

Tabla de Asignación del Disco Una tabla que indica los bloques del almacenamiento secundario que están libres y disponibles para su asignación a los archivos.

Tarea de Tiempo Real Una tarea que se ejecuta en conexión con algún proceso, función o conjunto de sucesos externos al sistema informático y que deben satisfacer uno o más plazos para que interactúen de forma eficaz y correcta con el entorno exterior.

Tarea **Lo mismo que *Proceso.***

Tiempo Compartido* El uso concurrente de un dispositivo por una serie de usuarios.

Traductor de Direcciones* Una unidad funcional que transforma las direcciones virtuales en direcciones reales.

Trampilla Punto de entrada secreto y no documentado en un programa, empleado para conseguir el acceso sin los métodos normales de autentificación.

Tubo Un *buffer* circular que permite que dos procesos se comuniquen según el modelo del productor/consumidor. Es. por tanto, una cola "primero en entrar, primero en salir", donde escribe un proceso y el otro lee. En algunos sistemas, el tubo se generaliza para permitir que se pueda escoger cualquier elemento de la cola para consumirlo.

Ultimo en Entrar, Primero en Salir* Una técnica de colas en la que el elemento siguiente que se recupera es el elemento situado hace menos tiempo en la cola.

Unidad de Datos de Protocolo Información que se entrega como una unidad entre entidades parejas de una red y que pueden contener información de control, de direccionamiento o datos.

Virus Rutina secreta no documentada, empotrada en un programa útil. La ejecución del programa provoca la ejecución de la rutina secreta.

Referencias

ADAM92 ADAM, J. "Virus Threats and Countermeasures." *IEEE Spectrum*, agosto de 1992.

AGAR89a AGARWAL, A. *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Kluwer Academic Publishers, Boston, 1989.

AGAR89b AGARWAL, A., HOROWITZ, M., y HENNESSY, J. "An Analytical Cache Model." *ACM Transactions on Computer Systems*, mayo de 1988.

ALMA89 ALMASI, G., y GOTTLIEB, A. *Highly Parallel Computing*. Benjamin/Cummings, Redwood City, CA, 1989.

ALVA90 ALVARE, A. "How Crackers Crack Passwords or What Passwords to Avoid." *Proceedings, UNIX Security Workshop II*, agosto de 1990.

ANAN92 ANANDA, A., TAY, B., y KOH, E. "A Survey of Asynchronous Remote Procedure Calls." *Operating Systems Review*, abril de 1992.

ANDE89 ANDERSON, T., LAXOWSKA, E., y LEVY, H. "The Performance Implications of Thread Management Alternatives for Shared-Memory Multiprocessors." *IEEE Transactions on Computers*, diciembre de 1989.

ANDL90 ANDLEIGH, P. *UNIX System Architecture*. Prentice Hall, Englewood Cliffs, NJ, 1990.

ANDR90 ANDRIANOFF, S. "A Module on Distributed Systems for the Operating System Course." *Proceedings, Twenty-First SIGCSE Technical Symposium on Computer Science Education, SIGSCE Bulletin*, febrero de 1990.

ANSI90 AMERICAN NATIONAL STANDARDS INSTITUTE. *American National Standard Dictionary for Information Systems*. X3-N-1990.

ARDE80 ARDEN, B., ed. *What Can Be Automated?* MIT Press, Cambridge, MA, 1980.

ARTS89a ARTSY, Y., ed. Special Issue on Process Migration. *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems* , invieno de 1989.

ARTS89b ARTSY, Y. "Designing a Process Migration Facility: The Charlotte Experience." *Computer*, septiembre de 1989.

ATLA89 ATLAS, A., y BLUNDON, B. "Time to Reach for It All." *UNIX Review*, enero de 1989.

ATT87a AT&T *UNIX System Readings and Examples*, Volume I. Prentice Hall, Englewood Cliffs, NJ, 1987.

ATT87b AT&T *UNIX System Readings and Examples*, Volume II. Prentice Hall, Englewood Cliffs, NJ, 1987.

AXFO88 AXFORD, T. *Concurrent Programming: Fundamental Techniques for Real-Time and Parallel Software Design*. Wiley, Nueva York, 1988.

BACH86 BACH, M. *The Design of the UNIX Operating System*. Prentice Hall, Englewood Cliffs, NJ, 1986.

BACO93 BACON, J. *Concurrent Systems*. Addison-Wesley, Reading, MA, 1993.

BAER80 BAER, J. *Computer Systems Architecture*. Computer Science Press, Rockville, MD, 1980.

BARB90 BARBOSA, V. "Strategies for the Prevention of Communication Deadlocks in Distributed Parallel Programs." *IEEE Transactions on Software Engineering*, noviembre de 1990.

BAYS77 BAYS, C. "A Comparison of Next-Fit, First-Fit, and Best-Fit." *Communications of the ACM*, marzo de 1977.

BECK90 BECK, L. *System Software*. Addison-Wesley, Reading, MA, 1990.

BELA66 BELADY, L. "A Study of Replacement Algorithms for a Virtual Storage Computer," *IBM Systems Journal*, No. 2, 1966.

BELL71 BELL, C., y NEWELL, A., editors. *Computer Structures: Readings and Examples*. McGraw-Hill, Nueva York, 1971.

BELL92 BELLOVIN, S. "There Be Dragons." *Proceedings, UNIX Security Symposium III*, septiembre de 1992.

BELL93 BELLOVIN, S. "Packets Found on an Internet." *Computer Communications Review*, julio de 1993.

BEN82 BEN-ARI, M. *Principles of Concurrent Programming*. Prentice Hall, Englewood Cliffs, NJ, 1982.

BEN90 BEN-ARI, M. *Principles of Concurrent and Distributed Programming*. Prentice Hall, Englewood Cliffs, NJ, 1990.

BERN89 BERNABEU, J. et al. "The Architecture of Ra: A Kernel for Clouds." *Proceedings, 22nd Annual Hawaii International Conference on System Science*, enero de 1989.

BERN90 BERNSTEIN, P. "Transaction Processing Monitors." *Communications of the ACM*, noviembre de 1990.

BERN93 BERNSTEIN, P. "Middleware: An Architecture for Distributed System Services." *Report CRL 93/6*. Digital Equipment Corporation Cambridge Research Lab, 2 de marzo de 1993.

BERS92 BERSON, A. *Client/Server Architecture*. McGraw-Hill, Nueva York, 1992.

BIC88 BIC, L., y SHAW, A. *The Logical Design of Operating Systems, 2nd ed.* Prentice Hall, Englewood Cliffs, NJ, 1988.

BLAC90 BLACK, D. "Scheduling Support for Concurrency and Parallelism in the Mach Operating System." *Computer*, mayo de 1990.

BOEB85 BOEBERT, W., KAIN, R., y YOUNg, W. "Secure Computing: The Secure Ada Target Approach." *Scientific Honeyweller*, julio de 1985. Reprinted in [ABRA87].

BOLO89 BOLOSKY, W., FITZGERALD, R., y SCOTT, M. "Simple But Effective Techniques for NUMA Memory Management." *Proceedings, Twelfth ACM Symposium on Operating Systems Principles*, diciembre de 1989.

BORG90 BORG, A., KESSLER, R., y WALL, D. "Generation and Analysis of Very Long Addres Traces." *Proceedings of the 17th Annual International Symposium on Computer Architecture*, mayo de 1990.

BOWL92 BOWLES, J., y PELGEZ, C. "Bad Code." *IEEE Spectrum*, agosto de 1992.

BRAN78 BRANSTAD, D., ed. *Computer Security and the Data Encryption Standard*. National Bureau of Standards, Special Publication No. 500-27, febrero de 1978.

BRAZ94 BRAZIER, F., y JOHANSEN, D. *Distributed Open Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1994.

BREN89 BRENT, R. "Efficient Implementation of the First-Fit Strategy for Dynamic Storage Allocation." *ACM Transactions on Programming Languages and Systems*, julio de 1989.

BRIN73 BRINCH-HANSEN, P. *Operating System Principles*. Prentice Hall, Englewood Cliffs, NJ, 1973.

BROW84 BROWN, R., Denning, P., y Tichy, W. "Advanced Operating Systems." *Computer*, octubre de 1984.

CABR86 CABREAR, L. "The Influence of Workload on Load Balancing Strategies." *USENIX Conference Proceedings*, verano de 1986.

CALI82 CALINGAERT, P. *Operating System Elements: A User Perspective*. Prentice Hall, Englewood Cliffs, NJ, 1982.

CARD89 CARDELLI, et al. *Modula-3 Report (rev.)*. Systems Research Center Report SRC-52, Digital Equipment Corp, Palo Alto, CA, noviembre de 1989.

CARR81 CARR, R., y HENNESSEY, J. "WSClock—A Simple and Efficient Algorithm for Virtual Memory Management." *Proceedings of the Eighth Symposium on Operating System Principles*, diciembre de 1981.

CARR84 CARR, R. *Virtual Memory Management*. UMI Research Press, Ann Arbor, MI, 1984.

CARR89 CARRIERO, N., y GELERNTER, D. "How to Write Parallel Programs: A Guide for the Perplexed." *ACM Computing Surveys*, septiembre de 1989.

CASA94 CASAVANT, T., y SINGHAL, M. *Distributed Computing Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1994.

CAST92 CASTILLO, C., FLANAGAN, E., y WILKINSON, N. "Object-Oriented Design and Programming." *AT&T Technical Journal*, noviembre/diciembre de 1992.

CHAN85 CHANDY, K., y LAMPORT, L. "Distributed Snapshots: Determining Global States of Distributed Systems." *ACM Transactions on Computer Systems*, febrero de 1985.

CHAN88 CHANG, A., y MERGEN, M. "801 Storage: Architecture and Programming." *ACM Transactions on Computer Systems*, febrero de 1988.

CHAN90a CHANG, A. et al. "Evolution of Storage Facilities in AIX Version 3 for RISC System/6000 Processors." *IBM Journal of Research and Development*, enero de 1990.

CHAN90b CHANDRAS, R. "Distributed Message Passing Operating Systems." *Operating Systems Review*, enero de 1990.

CHEN90 CHEN, P. y PATTERSON, D. "Maximizing Performance in a Striped Disk Array." *Proceedings, 17th Annual International Symposium on Computer Architecture*, mayo de 1990.

CHEN92 CHEN, J., BORG, A., y JOUPPI, N. "A Simulation Based Study of TLB Performance." *Proceedings of the 19th Annual International Symposium on Computer Architecture*, mayo de 1992.

CHU72 CHU, W., y OPDERBECK, H. "The Page Fault Frequency Replacement Algorithm." *Proceedings, Fall Joint Computer Conference*, 1972.

CHU76 CHU, W., y OPDERBECK, H. "Program Behavior and the Page-Fault-Frequency Replacement Algorithm." *Computer*, noviembre de 1976.

CLAR85 CLARK, D., y EMER, J. "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement." *ACM Transactions on Computer Systems*, febrero de 1985.

COFF71 COFFMAN, E., ELPHICK, M., y SHOSHANI, A. "System Deadlocks." *Computing Surveys*, junio de 1971.

COHE90 COHEN, F. *A Short Course on Computer Viruses*. ASP Press, Pittsburgh, PA, 1990.

CONW63 CONWAY, M. "Design of a Separable Transition-Diagram Compiler." *Communications of the ACM*, julio de 1963.

CONW67 CONWAY, R., MAXWELL, W., y MILLER, L. *Theory of Scheduling*. Addison-Wesley, Reading, MA, 1967.

CONW89 CONWAY, A., y GEORGANAS, N. *Queuing Networks—Exact Computational Algorithms: A Unified Theory Based on Decomposition and Aggregation*. MIT Press, Cambridge, MA, 1989.

COOP81 COOPER, R. *Introduction to Queuing Theory, Second Edition*. North Holland, Nueva York, 1981.

COOP89 COOPER, J. *Computer and Communications Security: Strategies for the 1990s*. McGraw-Hill, Nueva York, 1990.

COOP90 COOPER, E., STEENKISTE, P., SANSOM, R., y ZILL, B. "Protocol Implementation on the Nectar Communication Processor." *Proceedings, SIGCOMM '90 Symposium*, septiembre de 1990.

CORA88 CORAZA, J., editor. *IBM Application System/400 Technology*. IBM SA21-9540, 1988.

CORB62 CORBATO, F., Merwin-Daggett, M., y Dealey, R. "An Experimental Time-Sharing System." *Proceedings of the 1962 Spring Joint Computer Conference*, 1962.

CORB63 CORBATO, F., et al. *The Compatible Time-Sharing System: A Programmer's Guide*. M.I.T. Press, Cambridge, MA, 1963.

CORB68 CORBATO, F. "A Paging Experiment with the Multics System," *MIT Project MAC Report MAC-M-384*, mayo de 1968.

COX89 COX, A., y FOWLER, R. "The Implementation of a Coherent Memory Abstraction on a NUMA Multiprocessor: Experiences with PLATINUM." *Proceedings, Twelfth ACM Symposium on Operating Systems Principles*, diciembre de 1989.

CUST93 CUSTER, H. *Inside Windows NT.* Microsoft Press, Redmond, WA, 1993.

DASG88 DASGUPTA, P., LEBLANC, R., y APPELBE, W. "The Clouds Distributed Operating System: Functional Description, Implementation Details and Related Work." *Proceedings, 8th International Conference on Distributed Computing Systems*, junio de 1988.

DATT90 DATTA, A., y GHOSH, S. "Deadlock Detection in Distributed Systems." *Proceedings, Phoenix Conference on Computers and Communications*, marzo de 1990.

DATT92 DATTA, A., JAVAGAL, R., y GHOSH, S. "An Algorithm for Resource Deadlock Detection in Distributed Systems." *Computer Systems Science and Engineering*, octubre de 1992.

DAVI89 DAVIES, D., y PRICE, W. *Security for Computer Networks.* Wiley, Nueva York, 1989.

DAVI87 DAVIS, W. *Operating Systems: A Systematic View, Third Edition.* Addison-Wesley, Reading, MA, 1987.

DEIT90 DEITEL, H. *An Introduction to Operating Systems, Second Edition.* Addison-Wesley, Reading, MA, 1990.

DENN68 DENNING, P. "The Working Set Model for Program Behavior." *Communications of the ACM*, mayo de 1968.

DENN70 DENNING, P. "Virtual Memory." *Computing Surveys*, septiembre de 1970.

DENN71 DENNING, P. "Third Generation Computer Systems." *Computing Surveys*, diciembre de 1971.

DENN80a DENNING, P., Buzen, J., Dennis, J., Gaines, R., Hansen, P., Lynch, W., and Organick, E. "Operating Systems" in [ARDE80].

DENN80b DENNING, P. "Working Sets Past and Present." *IEEE Transactions on Software Engineering*, enero de 1980.

DENN84 DENNING, P., y BROWN, R. "Operating Systems." *Scientific American*, septiembre de 1984.

DENN87 DENNING, D. "An Intrusion-Detection Model." *IEEE Transactions on Software Engineering*, febrero de 1987.

DENN90 DENNING, P. *Computers Under Attack: Intruders, Worms, and Viruses.* Addison-Wesley, Reading, MA, 1990.

DEWA90 DEWAR, R., y SMOSNA, M. *Microprocessors: A Programmer's View.* McGraw-Hill, Nueva York, 1990.

DEWI93 DEWIRE, D. *Client/Server Computing.* McGraw-Hill, Nueva York, 1993.

DIJK65 DIJKSTRA, E. *Cooperating Sequential Processes.* Technological University, Eindhoven, The Netherlands, 1965. (Reprinted in *Programming Languages*, F. Genuys, ed., Academic Press, Nueva York, NY, 1968).

DOUG89 DOUGLAS, F., y OUSTERHOUT, J. "Progress Migration in Sprite: A Status Report." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems*, invierno de 1989.

DUNN74 DUNN, E., *Social Information Processing and Statistical Systems—Change and Reform.* Wiley, Nueva York, 1974.

EAGE86 EAGER, D., LAZOWSKA, E., y ZAHNORJAN, J. "Adaptive Load Sharing in Homogeneous Distributed Systems." *IEEE Transactions on Software Engineering*, mayo de 1986.

ENGE80 ENGER, N., y HOWERTON, P. *Computer Security.* Amacom, Nueva York, 1980.

ESKI90 ESKICIOGLU, M. "Design Issues of Process Migration Facilities in Distributed Systems." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems and Application Environments*, verano de 1990.

FALK88 FALK, H. "Deadline Scheduling for Timely Response." *Computer Design*, 1 de abril de 1988.

FEIT90a FEITELSON, D., y RUDOLPH, L. "Distributed Hierarchical Control for Parallel Processing." *Computer*, mayo de 1990.

FEIT90b FEITELSON, D., y RUDOLPH, L. "Mapping and Scheduling in a Shared Parallel Environment Using Distributed Heirarchical Control." *Proceedings, 1990 International Conference on Parallel Processing*, agosto de 1990.

FELD93 FELDMAN, L. *Windows NT: The Next Generation.* Sams, Carmel, IN, 1993.

FERR83 FERRARI, D., y YIH, Y. "VSWS: The Variable-Interval Sampled Working Set Policy." *IEEE Transactions on Software Engineering*, mayo de 1983.

FINK88 FINKEL, R. *An Operating Systems Vade Mecum*. Prentice Hall, Englewood Cliffs, NJ, 1988.

FINK89 FINKEL, R. "The Process Migration Mechanism of Charlotte." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems*, invierno de 1989.

FOLK87 FOLK, M., y ZOELLICK, B. *File Structures: A Conceptual Toolkit*. Addison-Wesley, Reading, MA, 1987.

FOST91 FOSTER, I. "Automatic Generation of Self-Scheduling Programs." *IEEE Transactions on Parallel and Distributed Systems*, enero de 1991.

FRAN76 FRANKEL, T. *Tables for Traffic Management and Design*. abc Teletraining (P.O. Box 537, Geneva, IL 60134), 1976.

FREE93 FREEDMAN, D. "The Goods on Hacker Hoods." *Forbes ASAP*, 13 de septiembre de 1993.

GASS88 GASSER, M. *Building a Secure Computer System*. Van Nostrand Reinhold, Nueva York, 1988.

GEHR87 GEHRINGER, E., SIEWIOREK, D., y SEGALL, Z. *Parallel Processing: The Cm* Experience*. Digital Press, Bedford, MA: 1987.

GIBB87 GIBBONS, P. "A StubGenerator for Multilanguage RPC in Heterogeneous Environments." *IEEE Transactions on Software Engineering*, enero de 1987.

GING90 GINGRAS, A. "Dining Philosophers Revisited." *ACM SIGCSE Bulletin*, septiembre de 1990.

GOLD85 GOLDSTONE, B. "A Backdoor Password Problem." *EDPACS*, julio de 1985.

GOLD89 GOLDMAN, P. "Mac VM Revealed." *Byte*, septiembre de 1989.

GOPA85 GOPAL, I. "Prevention of Store-and-Forward Deadlock in Computer Networks," *IEEE Transactions on Communications*, diciembre de 1985.

GROS86 GROSSHANS, D. *File Systems: Design and Implementation*. Prentice Hall, Englewood Cliffs, NJ, 1986.

GUPT78 GUPTA, R., y FRANKLIN, M. "Working Set and Page Fault Frequency Replacement Algorithms: A Performance Comparison." *IEEE Transactions on Computers*, agosto de 1978.

GUYN88 GUYNES, J. "Impact of System Response Time on State Anxiety." *Communications of the ACM*, marzo de 1988.

HAFN91 HAFNER, K., y MARKOFF, J. *Cyberpunk: Outlaws and Hackers on the Computer Frontier*. Simon & Schuster, Nueva York, 1991.

HALD91 HALDAR, S., y SUBRAMANIAN, D. "Fairness in Processor Scheduling in Time Sharing Systems." *Operating Systems Review*, enero de 1991.

HARB90 HARBISON, S. "Modula-3." *Byte*, noviembre de 1990.

HATF72 HATFIELD, D. "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance." *IBM Journal of Research and Development*, enero de 1972.

HAYE88 HAYES, J. *Computer Architecture and Organization, Second Edition*. McGraw-Hill, Nueva York, 1988.

HENN90 HENNESSY, J., y PATTERSON, D. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, 1990.

HENR84 HENRY, G. "The Fair Share Scheduler." *AT&T Bell Laboratories Technical Journal*, octubre de 1984.

HOAR74 HOARE, C. "Monitors: An Operating System Structuring Concept." *Communications of the ACM*, octubre de 1974.

HOAR85 HOARE, C. *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, NJ, 1985.

HOFF90 HOFFMAN, L., ed. *Rogue Programs: Viruses, Worms, and Trojan Horses*. Van Nostrand Reinhold, Nueva York, 1990.

HOFR90 HOFRI, M. "Proof of a Mutual Exclusion Algorithm." *Operating Systems Review*, enero de 1990.

HOLT72 HOLT, R. "Some Deadlock Properties of Computer Systems." *Computing Surveys*, septiembre de, 1972.

HONG89 HONG, J., TAN, X., y TOWSLEY, D. "A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System." *IEEE Transactions on Computers*, diciembre de 1989.

HORN89 HORNER, D. *Operating Systems: Concepts and Applications*. Glenview, IL: Scott, Foresman, 1989.

HUCK83 HUCK, T. *Comparative Analysis of Computer Architectures*. Stanford University Technical Report Number 83-243, mayo de 1983.

HUCK93 HUCK, J., y HAYS, J. "Architectural Support for Translation Table Management in Large Address Space Machines." *Proceedings of the 20th Annual International Symposium on Computer Architecture*, mayo de 1993.

HYMA66 HYMAN, H. "Comments on a Problem in Concurrent Programming Control." *Communications of the ACM*, enero de 1966.

IBM71 IBM CORP. *Analysis of Some Queuing Models in Real-Time Systems*. IBM Document GF20-0007, 1971. Available from IBM document distribution centers.

IBM86 IBM NATIONAL TECHNICAL SUPPORT. Large Systems. *Multiple Virtual Storage (MVS) Virtual Storage Tuning Cookbook*. Dallas, TX: Dallas Systems Center Technical Bulletin G320-0597, junio de 1986.

INMO91 INMON, W. *Developing Client/Server Applications in an Architected Environment*. QED, Boston, MA, 1991.

ISLO80 ISLOOR, S., y MARSLAND, T. "The Deadlock Problem: An Overview." *Computer*, septiembre de 1980.

JOHN89 JOHNSON, R. *MVS Concepts and Facilities*. McGraw-Hill, New York, 1989.

JOHN91 JOHNSTON, B., JAVAGAL, R., DATTA, A., y GHOSH, S. "A Distributed Algorithm for Resource Deadlock Detection." *Proceedings, Tenth Annual Phoenix Conference on Computers and Communications*, marzo de 1991.

JONE80 JONES, S., y SCHWARZ, P. "Experience Using Multiprocessor Systems—A Status Report." *Computing Surveys*, junio de 1980.

JUL88 JUL, E., LEVY, H., HUTCHINSON, N., y BLACK, A. "Fine-Grained Mobility in the Emerald System." *ACM Transactions on Computer Systems*, febrero de 1988.

JUL89 JUL, E. "Migration of Light-Weight Processes in Emerald." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems*, invierno de 1989.

KATZ84 KATZAN, H., y THARAYIL, D. *Invitation to MVS: Logic and Debugging*. Petrocelli, Nueva York, 1984.

KATZ89 KATZ, R., et al. "A Project on High Performance I/O Subsystems." *Computer Architecture News*, septiembre de 1989.

KAY88 KAY, J., y LAUDER, P. "A Fair Share Scheduler." *Communications of the ACM*, enero de 1988.

KENA88 KENAH, L., GOLDENBERG, R., y BATE, S. *VAX/VMS Internals and Data Structures*. Digital Press, Bedford, MA, 1988.

KESS92 KESSLER, R., y HILL, M. "Page Placement Algorithms for Large Real-Indexed Caches." *ACM Transactions on Computer Systems*, noviembre de 1992.

KHAL93 KHALIDI, Y., TALLURI, M., WILLIAMS, D., y NELSON, M. "Virtual Memory Support for Multiple Page Sizes." *Proceedings, Fourth Workshop on Workstation Operating Systems*, octubre de 1993.

KILB62 KILBURN, T., EDWARDS, D., LANIGAN, M., y SUMNER, F. "One-Level Storage System." *IRE Transactions*, abril de 1962; reprinted in [BELL71].

KIM86 KIM, M. "Synchronized Disk Interleaving." *IEEE Transactions on Computers*, noviembre de 1986.

KIM87 KIM, M., y TANTAWI, A. *Asynchronous Disk Interleaving*. IBM T. J. Watson Research Center, Report RC 12497, Yorktown Heights, NY, febrero de 1987.

KLEI75 KLEINROCK, L. *Queueing Systems, Volume I: Theory*. Wiley, Nueva York, 1975.

KLEI76 KLEINROCK, L. *Queueing Systems, Volume II: Computer Applications*. Wiley, Nueva York, 1976.

KLEI90 KLEIN, D. "Foiling the Cracker: A Survey of, and Improvements to, Password Security." *Proceedings, UNIX Security Workshop II*, agosto de 1990.

KNUT71 KNUTH, D. "An Experimental Study of FORTRAN Programs." *Software Practice and Experience*, Vol. 1, 1971.

KNUT73 KNUTH, D. *The Art of Computer Programming, Volume 1: Fundamental Algorithms, Second Edition*. Addison-Wesley, Reading, MA, 1973.

KOBA78 KOBAYASHI, H. *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*. Addison-Wesley, Reading, MA, 1978.

KORZ93 KORZENIOWSKI, P. "Make Way for Data." *Byte*, junio de 1993.

KRAK88 KRAKOWIAK, S., y BEESON, D. *Principles of Operating Systems*. MIT Press, Cambridge, MA, 1988.

KRIS94 KRISHNA, C., y LEE, Y., eds. "Special Issue on Real-Time Systems." *Proceedings of the IEEE*, enero de 1994.

KRON90 KRON, P. "A Software Developer Looks at OS/2." *Byte*, agosto de 1990.

KUCK78 KUCK, D. *The Structure of Computers and Computations*. Wiley, New York, 1978.

KURZ84 KURZBAN, S., HEINES, T., y KURZBAN, S. *Operating Systems Principles*, 2nd ed. Van Nostrand Reinhold, Nueva York, 1984.

LAMP74 LAMPORT, L. "A New Solution of Dijkstra's Concurrent Programming Problem." *Communications of the ACM*, agosto de 1974.

LAMP78 LAMPORT, L. "Time, Clocks, and the Ordering of Events in a Distributed System." *Communications of the ACM*, julio de 1978.

LAMP80 LAMPSON, B., y REDELL, D. "Experience with Processes and Monitors in Mesa." *Communications of the ACM*, febrero de 1980.

LAMP86 LAMPORT, L. "The Mutual Exclusion Problem." *Journal of the ACM*, abril de 1986.

LAMP91 LAMPORT, L. "The Mutual Exclusion Problem Has Been Solved." *Communications of the ACM*, enero de 1991.

LARM75 LARMOUTH, J. "Scheduling for a Share of the Machine." *Software Practices and Experience*, enero de 1975.

LARO92 LAROWE, R., HOLLIDAY, M., y ELLIS, C. "An Analysis of Dynamic Page Placement and a NUMA Multiprocessor." *Proceedings, 1992 ACM SIGMETRICS and PERFORMANCE '92*, junio de 1992.

LEBA84a LEBAN, J., y ARNOLD, J. *IBM I/O Architecture and Virtual Storage Concepts: System/370-Mode and 370-XA-Mode Processors*. Wiley, Nueva York, 1984.

LEBA84b LEBAN, J., y ARNOLD, J. *IBM CPU and Storage Architecture: System/370-Mode and 370-XA-Mode Processors*. Wiley, Nueva York, 1984.

LEBL87 LEBLANC, T., y MELLOR-CRUMMEY, J. "Debugging Parallel Programs with Instant Replay." *IEEE Transactions on Computers*, abril de 1987.

LEE93 LEE, Y., y KRISHNA, C., eds. *Readings in Real-Time Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1993.

LEFF88 LEFFLER, S., MCKUSICK, M., KARELS, M., QUARTERMAIN, J., y STETTNER, A. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, Reading, MA, 1988.

LEIB89 LEIBFRIED, T. "A Deadlock Detection and Recovery Algorithm Using the Formalism of a Directed Graph Matrix." *Operating Systems Review*, abril de 1989.

LELA86 LELAND, W., y OTT, T. "Load-Balancing Heuristics and Process Behavior." *Proceedings, ACM SigMetrics Performance 1986 Conference*, 1986.

LERO76 LEROUDIER, J., y POTIER, D. "Principles of Optimality for Multiprogramming." *Proceedings, International Symposium on Computer Performance Modeling, Measurement, and Evaluation*, marzo de 1976.

LETW88 LETWIN, G. *Inside OS/2*. Microsoft Press, Redmond, WA, 1988.

LEUT90 LEUTENEGGER, S., y VERNON, M. "The Performance of Multiprogrammed Multiprocessor Scheduling Policies." *Proceedings, Conference on Measurement and Modeling of Computer Systems*, mayo de 1990.

LIST88 LISTER, A., y EAGER, R. *Fundamentals of Operating Systems*, 4th ed. Macmillan Education Ltd, Londres, 1988.

LIU73 LIU, C., y LAYLAND, J. "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment." *Journal of the ACM*, febrero de 1973.

LIVA90 LIVADAS, P. *File Structures: Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1990.

MADS93 Madsen, J. "World Record in Password Checking." *Usenet, comp.security.misc newsgroup*, 18 de agosto de 1993.

MAEK87 MAEKAWA, M., OLDEHOEFT, A., y OLDEHOEFT, R. *Operating Systems: Advanced Concepts*. Benjamin Cummings, Menlo Park, CA, 1987.

MAJU88 MAJUMDAR, S., EAGER, D., y BUNT, R. "Scheduling in Multiprogrammed Parallel Systems." *Proceedings, Conference on Measurement and Modeling of Computer Systems*, mayo de 1988.

MANO88 MANO, M. *Computer Engineering Hardware Design*. Prentice Hall, Englewood Cliffs, NJ, 1988.

MART72 MARTIN, J. *Systems Analysis for Data Transmission*. Prentice Hall, Englewood Cliffs, NJ, 1972.

MART77 MARTIN, J. *Computer Data-Base Organization*. Prentice Hall, Englewood Cliffs, NJ, 1977.

MART88 MARTIN, J. *Principles of Data Communication*. Prentice Hall, Englewood Cliffs, NJ, 1988.

MILE92 MILENKOVIC, M. *Operating Systems: Concepts and Design*. McGraw-Hill, Nueva York, 1992.

MILL68 MILLER, R. "Response Time in Man-computer Conversational Transactions." *Proceedings, Spring Joint Computer Conference*, 1968.

MOLL89 MOLLOY, M. *Fundamentals of Performance Modeling*. Macmillan, New York, 1989.

MORG92 MORGAN, K. "The RTOS Difference." *Byte*, agosto de 1992.

MORS87 MORSE, S., ISAACSON, E., y ALBERT, D. *The 80386/387 Architecture*. Wiley, Nueva York, 1987.

MORS93 MORSE, S. "Client/Server Is Not Always the Best Solution." *Network Computing*, Special Issue on Client/Server Computing, primavera de 1993.

MOSK93 MOSKOWITZ, R. "What Are Clients and Servers Anyway?" *Network Computing*, Special Issue on Client/Server Computing, primavera de 1993.

NBS63 NATIONAL BUREAU OF STANDARDS. *Experimental Statistics*. NBS Handbook 91 (Available from Government Printing Office, GOP Stock No. 003-003-00135-0), 1963.

NEHM75 NEHMER, J. "Dispatcher Primitives for the Construction of Operating System Kernels." *Acta Informatica*, Vol. 5, 1975.

NELS88 NELSON, M., WELCH, B., y OUSTERHOUT, J. "Caching in the Sprite Network File System." *ACM Transactions on Computer Systems*, febrero de 1988.

NELS91 NELSON, G. *Systems Programming with Modula-3*. Prentice Hall, Englewood Cliffs, NJ, 1991.

NG88 NG, S., LANG, D., y SELINGER, R. "Trade-Offs Between Devices and Paths in Achieving Disk Interleaving." *Proceedings, 15th International Symposium on Computer Architecture*, junio de 1988.

NUTT92 NUTT, G. *Centralized and Distributed Operating Systems*. Prentice Hall, Englewood Cliffs, NJ, 1992.

OLSO89 OLSON, T. "Disk Array Performance in a Random I/O Environment." *Computer Architecture News*, septiembre de 1989.

OUST85 OUSTERHOUT, J., et al. "A Trace-Drive Analysis of the UNIX 4.2 BSD File System." *Proceedings, Tenth ACM Symposium on Operating System Principles*, 1985.

OUST88 OUSTERHOUT, J., et al. "The Sprite Network Operating System." *Computer*, febrero de 1988.

PAAN86 PAANS, R. *A Close Look at MVS Systems: Mechanisms, Performance, and Security*. Elsevier, Nueva York, 1986.

PANW88 PANWAR, S., TOWSLEY, D., y WOLF, J. "Optimal Scheduling Policies for a Class of Queues with Customer Deadlines in the Beginning of Service." *Journal of the ACM*, octubre de 1988.

PATT82 PATTERSON, D., y SEQUIN, C. " A VLSI RISC." *Computer*, septiembre de 1982.

PATT85 PATTERSON, D. "Reduced Instruction Set Computers." *Communications of the ACM*, enero de 1985.

PETE81 PETERSON, G. "Myths About the Mutual Exclusion Problem." *Information Processing Letters*, junio de 1981.

PFLE89 PFLEEGER, C. *Security in Computing*. Prentice Hall, Englewood Cliffs, NJ, 1989.

PINK89 PINKERT, J., y WEAR, L. *Operating Systems: Concepts, Policies, and Mechanisms*. Prentice Hall, Englewood Cliffs, NJ, 1989.

PIZZ89 PIZZARELLO, A. "Memory Management for a Large Operating System." *Proceedings, International Conference on Measurement and Modeling of Computer Systems*, mayo de 1989.

PLAM89 PLAMBECK, K. "Concepts of Enterprise Systems Architecture." *IBM Systems Journal*, No. 1, 1989.

POPE85 POPEK, G., y WALKER, B. *The LOCUS Distributed System Architecture*, MIT Press, Cambridge, MA, 1985.

PORR92 PORRAS, P. *STAT: A State Transition Analysis Tool for Intrusion Detection*. Master's Thesis, University of California at Santa Barbara, julio de 1992.

POWE93 POWELL, J. *Multitask Windows NT*. Corte Waite Group Press, Madera, CA, 1993.

PRAS81 PRASAD, N. *Architecture and Implementation of Large Scale IBM Computer Systems*. Q.E.D., Wellesley, MA, 1981.

PRAS89 PRASAD, N. *IBM Mainframes: Architecture and Design*. McGraw-Hill, New York, 1989.

PRZY88 PRZYBYLSKI, S., HOROWITZ, M., y HENNESSY, J. "Performance Tradeoffs in Cache Design." *Proceedings, Fifteenth Annual International Symposium on Computer Architecture*, junio de 1988.

RAMA94 RAMAMRITHAM, K., y STANKOVIC, J. "Scheduling Algorithms and Operating Systems Support for Real-Time Systems." *Proceedings of the IEEE*, enero de 1994.

RASH88 RASHID, R., et al. "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures." *IEEE Transactions on Computers*, agosto de 1988.

RASH89 RASHID, R., et al. "Mach: A System Software Kernel." *Proceedings, COMPCON Sprint '89*, marzo de 1989.

RAYN86 RAYNAL, M. *Algorithms for Mutual Exclusion*. MIT Press, Cambridge, MA, 1986.

RAYN88 RAYNAL, M. *Distributed Algorithms and Protocols*. Wiley, Nueva York, 1988.

RAYN90 RAYNAL, M., y HELARY, J. *Synchronization and Control of Distributed Systems and Programs*. Wiley, Nueva York, 1990.

REDD89 REDDY, A., y BANERJEE, P. "An Evaluation of Multiple-Disk I/O Systems." *IEEE Transactions on Computers*, diciembre de 1989.

RENA93 RENAUD, P. *Introduction to Client/Server Systems*. Wiley, Nueva York, 1993.

RICA81 RICART, G., y AGRAWALA, A. "An Optimal Algorithm for Mutual Exclusion in Computer Networks." *Communications of the ACM*, enero de 1981 (Corrigendum in *Communications of the ACM*, septiembre de 1981).

RICA83 RICART, G., y AGRAWALA, A. "Author's Response to 'On Mutual Exclusion in Computer Networks' by Carvalho and Roucairol." *Communications of the ACM*, febrero de 1983.

RITC74 RITCHIE, D., y THOMPSON, K. "The UNIX Time-Sharing System." *Communications of the ACM*, julio de 1974.

RITC78a RITCHIE, D., y THOMPSON, K. "The UNIX Time-Sharing System." *The Bell System Technical Journal*, julio de–agosto de 1978. This is a revision of RITC74 above.

RITC78b RITCHIE, D. "UNIX Time-Sharing System: A Retrospective." *The Bell System Technical Journal*, julio–agosto de 1978.

RITC84 RITCHIE, D. "The Evolution of the UNIX Time-Sharing System." *AT&T Bell Laboratories Technical Journal*, octubre de 1984.

ROBI90 ROBINSON, J., y DEVARAKONDA, M. "Data Cache Management Using Frequency-Based Replacement," *Proceedings, Conference on Measurement and Modeling of Computer Systems*, mayo de 1990.

ROSE78 ROSENKRANTZ, D., STEARNS, R., y LEWIS, P. "System Level Concurrency Control in Distributed Database Systems." *ACM Transactions on Database Systems*, junio de 1978.

RUDO90 RUDOLPH, B. "Self-Assessment Procedure XXI: Concurrency." *Communications of the ACM*, mayo de 1990.

RUSC77 RUSCHITZKA, M., y FABRY, R. "A Unifying Approach to Scheduling." *Communications of the ACM*, julio de 1977.

SAFF93 SAFFORD, D., SCHALES, D., y HESS, D. "The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment." *Proceedings, UNIX Security Symposium IV*, octubre de 1993.

SALT75 SALTZER, J., y SCHROEDER, M. "The Protection of Information in Computer Systems." *Proceedings of the IEEE*, septiembre de 1975.

SAMS90 SAMSON, S. *MVS Performance Management*. McGraw-Hill, Nueva York, 1990.

SATY81 SATYANARAYANAN, M., y BHANDARKAR, D. "Design Trade-Offs in VAX-11 Translation Buffer Organization." *Computer*, diciembre de 1981.

SAUE81 SAUER, C., y CHANDY, K. *Computer Systems Performance Modeling*. Prentice Hall, Englewood Cliffs, NJ, 1981.

SCHA62 SCHAY, G., y SPRUTH, W. "Analysis of a File Addressing Method." *Communications of the ACM*, agosto de 1962.

SCHL89 SCHLEICHER, D., y TAYLOR, L. "System Overview of the Application System/400." *IBM Systems Journal*, No. 3, 1989.

SCHN85 SCHNEIDER, G. *The Principles of Computer Organization*. Wiley, Nueva York, 1985.

SHA91 SHA, L., KLEIN, M., y GOODENOUGH, J. "Rate Monotonic Analysis for Real-Time Systems." In [TILB91].

SHAW87 SHAE, M., y SHAW, S. *UNIX Internals: A Systems Operations Handbook*. Tab Books, Blue Ridge Summit, PA, 1987.

SHIV92 SHIVARATRI, N., KRUEGER, P., y SINGHAL, M. "Load Distributing for Locally Distributed Systems." *Computer*, diciembre de 1992.

SHNE84 SHNEIDERMAN, B. "Response Time and Display Rate in Human Performance with Computers." *ACM Computing Surveys*, septiembre de 1984.

SHOR75 SHORE, J. "On the External Storage Fragmentation Produced by First-Fit and Best-Fit Allocation Strategies." Communications of the ACM, agosto de 1975.

SHUB90 SHUB, C. "ACM Forum: Comment of a Self-Assessment Procedure on Operating Systems." *Communications of the ACM*, septiembre de 1990.

SIEB83 SIEBER, J. *TRIX: A Communications-Oriented Operating System*. M.S. Thesis, M.I.T., septiembre de 1983.

SILB94 SILBERSCHATZ, A., y GALVIN, P. *Operating System Concepts*. Addison-Wesley, Reading, MA, 1994.

SING94a SINGHAL, M., y SHIVARATRI, N. *Advanced Concepts in Operating Systems*. McGraw-Hill, Nueva York, 1994.

SING94b SINGHAL, M. "Deadlock Detection in Distributed Systems." In [CASA94].

SMIT82 SMITH, A. "Cache Memories." *ACM Computing Surveys*, septiembre de 1982.

SMIT83 SMITH, D. "Faster Is Better: A Business Case for Subsecond Response Time." *Computerworld*, 18 de abril de 1983.

SMIT85 SMITH, A. "Disk Cache—Miss Ratio Analysis and Design Considerations." *ACM Transactions on Computer Systems*, agosto de 1985.

SMIT88 SMITH, J. "A Survey of Process Migration Mechanisms." *Operating Systems Review*, julio de 1988.

SMIT89 SMITH, J. "Implementing Remote *fork()* with Checkpoint/restart." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems*, invierno de 1989.

SPAF89 SPAFFORD, E., HEAPHY, K., y FERBRACHE, D. *Computer Viruses*. ADAPSO, Arlington, VA, 1989.

SPAF92 SPAFFORD, E. "Observing Reusable Password Choices." *Proceedings, UNIX Security Symposium III*, septiembre de 1992.

SRIM92 SRIMANI, P., y DAS, S. *Distributed Mutual Exclusion Algorithms*. IEEE Computer Society Press, Los Alamitos, CA, 1994.

STAL93a STALLINGS, W. *Computer Organization and Architecture*, 3rd ed. Prentice Hall, Englewood Cliffs, NJ, 1993.

STAL93b Stallings, W. *Networking Standards: A Guide to OSI, ISDN, LAN, and MAN Standards*. Addison-Wesley, Reading, MA, 1993.

STAL94 STALLINGS, W. *Data and Computer Communications*, 4th ed. Prentice Hall, Englewood Cliffs, NJ, 1994.

STAL95 STALLINGS, W. *Network and Internetwork Security: Principles and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1995.

STAN88 STANKOVIC, J., y RAMAMRITHAN, K. *Hard Real-Time Systems*. IEEE Computer Society Press, Washington, DC, 1988.

STAN93 STANKOVIC, J., y RAMAMRITHAM, K., eds. *Advances in Real-Time Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1993.

STEP93 STEPHENSON, P. "Preventive Medicine." *LAN Magazine*, noviembre de 1993.

STER92 STERLING, B. *The Hacker Crackdown: Law and Disorder on the Electronic Frontier.* Bantam, Nueva York, 1992.

STOL88 STOLL, C. "Stalking the Wily Hacker." *Communications of the ACM*, mayo de 1988. Reprinted in [DENN90].

STOL89 STOLL, C. *The Cuckoo's Egg.* Doubleday, Nueva York, 1989.

STON90 STONE, H. *High-Performance Computer Architecture*, 2nd ed. Addison-Wesley, Reading, MA, 1990.

STRE83 STRECKER, W. "Transient Behavior of Cache Memories." *ACM Transactions on Computer Systems*, noviembre de 1983.

STUC85 STUCK, B., y ARTHURS, E. *A Computer and Communications Network Performance Analysis Primer.* Prentice Hall, Englewood Cliffs, NJ, 1985.

SUZU82 SUZUKI, I., y KASAMI, T. "An Optimality Theory for Mutual Exclusion Algorithms in Computer Networks." *Proceedings of the Third International Conference of Distributed Computing Systems*, octubre de 1982.

TALL92 TALLURI, M., KONG, S., HILL, M., y PATTERSON, D. "Tradeoffs in Supporting Two Page Sizes." *Proceedings of the 19th Annual International Symposium on Computer Architecture*, mayo de 1992.

TAMI83 TAMIR, Y., y SEQUIN, C. "Strategies for Managing the Register File in RISC." *IEEE Transactions on Computers*, noviembre de 1983.

TANE78 TANENBAUM, A. "Implications of Structured Programming for Machine Architecture." *Communications of the ACM*, marzo de 1978.

TANE85 TANENBAUM, A., y RENESSE, R. "Distributed Operating Systems." *Computing Surveys*, diciembre de 1985.

TANE87 TANENBAUM, A. *Operating System Design and Implementation.* Prentice Hall, Englewood Cliffs, NJ, 1987.

TANE88 TANENBAUM, A. *Computer Networks.* Prentice Hall, Englewood Cliffs, NJ, 1988.

TANE90 TANENBAUM, A. *Structured Computer Organization.* Prentice Hall, Englewood Cliffs, NJ, 1990.

TANE92 TANENBAUM, A. *Modern Operating Systems.* Prentice Hall, Englewood Cliffs, NJ, 1992.

TAY90 TAY, B., y ANANDA, A. "A Survey of Remote Procedure Calls." *Operating Systems Review*, julio de 1990.

TEVA87 TEVANIAN, A., et al. "Mach Threads and the Unix Kernel: The Battle for Control." *Proceedings, Summer 1987 USENIX Conference*, junio de 1987.

TEVA89 TEVANIAN, A., y SMITH, B. "Mach: The Model for Future Unix." *Byte*, noviembre de 1989.

THAD81 THADHANI, A. "Interactive User Productivity." *IBM Systems Journal*, No. 1, 1981.

THEA83 THEAKER, C., y BROOKES, G. *A Practical Course on Operating Systems.* Springer-Verlag, Nueva York, 1983.

THOM84 THOMPSON, K. "Reflections on Trusting Trust (Deliberate Software Bugs)." *Communications of the ACM*, agosto de 1984.

TILB91 TILBORG, A., y KOOB, G., eds. *Foundations of Real-Time Computing: Scheduling and Resource Management.* Kluwer Academic Publishers, Boston, 1991.

TIME90 TIME, INC. *Computer Security, Understanding Computers Series.* Time-Life Books, Alexandria, VA, 1990.

TUCK89 TUCKER, A., y GUPTA, A. "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors." *Proceedings, Twelfth ACM Symposium on Operating Systems Principles*, diciembre de 1989.

TURN86 TURNER, R. *Operating Systems: Design and Implementations.* Macmillan, Nueva York, 1986.

VARH94 VARHOL, P. "Small Kernels Hit It Big." *Byte*, enero de 1994.

WALK89 WALKER, B., y MATHEWS, R. "Process Migration in AIX's Transparent Computing Facility." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems*, invierno de 1989.

WARD80 WARD, S. "TRIX: A Network-Oriented Operating System." Proceedings, COMPCON '80, 1980.

WARR91 WARREN, C. "Rate Monotonic Scheduling." *IEEE Micro*, junio de 1991.

WATE86 WATERS, F., editor. *RT Personal Computer Technology*. IBM SA23-1057, 1986.

WATS70 WATSON, R. *Timesharing System Design Concepts*. McGraw-Hill, Nueva York, 1970.

WAYN94 WAYNER, P. "Objects on the March." *Byte*, enero de 1994.

WEIZ81 WEIZER, N. "A History of Operating Systems." *Datamation*, enero de 1981.

WIED87 WIEDERHOLD, G. *File Organization for Database Design*. McGraw-Hill, Nueva York, 1987.

WOOD86 WOODSIDE, C. "Controllability of Computer Performance Tradeoffs Obtained Using Controlled-Share Queue Schedulers." *IEEE Transactions on Software Engineering*, octubre de 1986.

WOOD89 WOODBURY, P., et al. "Shared Memory Multiprocessors: The Right Approach to Parallel Processing." *Proceedings, COMPCON Spring '89*, marzo de 1989.

ZAHO90 ZAHORJAN, J., y MCCANN, C. "Processor Scheduling in Shared Memory Multiprocessors." *Proceedings, Conference on Measurement and Modeling of Computer Systems*, mayo de 1990.

Lista de acrónimos

ACRÓNIMO	TÉRMINO EN INGLÉS	TÉRMINO EN CASTELLANO
API	Application Programming Interface	Interfaz de Programas de Aplicación
CPU	Central Processing Unit	Unidad Central de Proceso
CTSS	Compatible Time-Sharing System	Sistema Compatible de Tiempo Compartido
DES	Data Encryption Standard	Estándar de Cifrado de Datos
DMA	Direct Memory Access	Acceso Directo a Memoria
FAT	File Allocation Table	Tabla de Asignación de Archivos
FCFS	First-Come-First-Served	Primero en Llegar, Primero en Servirse
FIFO	First-In-First-Out	Primero en Entrar, Primero en Salir
GUI	Graphical User Interface	Interfaz Gráfica de Usuario
IBM	International Business Machines	
JCL	Job Control Language	Lenguaje de Control de Trabajos
LAN	Local Area Network	Red de Área Local
LIFO	Last-In-First-Out	Último en Entrar, Primero en Salir
LRU	Least Recently Used	Usado hace más tiempo
MVS	Multiple Virtual Storage	Almacenamiento Virtual Múltiple
NUMA	Non-Uniform Memory Access	Acceso a Memoria No Uniforme
OSI	Open Systems Interconnection	Interconexión de Sistemas Abiertos
PC	Program Counter	Contador de Programa
PSW	Program Status Word	Palabra de Estado del Programa
PCB	Process Control Block	Bloque de Control de Proceso
RISC	Reduced Instruction Set Computer	Computador de Juego Reducido de Instrucciones
RPC	Remote Procedure Call	Llamada a Procedimiento Remoto
SMP	Symmetric Multiprocessing	Multiproceso Simétrico
SPOOL	Simultaneous Peripheral Operation On Line	Operación Simultánea de Periféricos en Línea
TLB	Translation Lookaside Buffer	Buffer de Traducción Adelantada

Índice Analítico

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Agradecimientos

Figura 2.3: de A. Silberschatz, J. Peterson. y P. Galvin, *Operating, Systems Concepts,* © 1991, por Addison-Wesley Publishing Company. Reproducido con permiso del editor. Figura 2.7 y 2.8: Reproducido con permiso de Macmillan Publishing Company de *Operating Systems: Design and Implementation* por Raymond W. Turner. Copyright © 1986 por Macmillan Publishing Company. una División de Macmillan, Inc. Figura 2.16: de Maurice J. Bach, *The Design of the UNIX® Operating System,* © 1986. pp. 20. Reproducido con permiso de Prentice Hall, Englewood Cliffs. New Jersey. Figura 2.17: Reproducido con permiso de *Architecture and Implementation of Large Scale IBM Computer Systems.* Segunda Edición. 1981, QED Information Sciences, Inc., Wellesley. MA. Tabla 3.2: de James R. Pinkert y Larry L. Wear. *Operating Svstems: Concepts, Policies and Mechanisms,* © 1989. p. 69. Reproducido con permiso de Prentice Hall, Englewood Cliffs, New Jersey. Tabla 3.11: de S. Krakowiak y D. Beeson, *Principies of Operating Systems.* Copyright © Massachusetts Institute of Technology 1988. Publicado por MIT Press. Figura 3.16: de Maurice J. Bach. *The Design of the UNIX® Operating System* © 1986, p. 148. Figura 3.17: de H. Custer, I*nside Windows NT,* © 1993. pp. 96. Reproducido con permiso de Microsoft Press. Reproducido con permiso de Prentice Hall, Englewood Cliffs, New Jersey. Tabla 5.1: de S. Isloor y T. Marsiand, "The Deadlock Problem: An Overview." *Computer.* September 1980. © IEEE. Figuras 4.3, 4.4, 4.5, y 4.12: de M. Ben-Ari. *Principies of Concurrent Programming;,* © 1982, pp. 29, 32, 39, 52. Reproducido con permiso de Prentice Hall, Englewood Cliffs, New Jersey. Figura 4.25: de Lubomir Bic y Alan C. Shaw. *The Logical Design of Operating Systems,* 2e., © 1988, p. 78. Reproducido con permiso de Prentice Hall, Englewood Cliffs, New Jersey. Figura 4.28: de M. Milenkovic. *Operating Systems: Concepts and Design* © 1987. Reproducido con permiso de Mc-Graw-Hill. Inc. Figura 7.14: de J. Hayes. *Computer Architecture and Organization,* Segunda Edición. © 1988. Reproducido con permiso de McGraw-Hill, Inc. Figura 7.18: de Maurice J. Bach, *The Design of the UN IX® Operating, System.* © 1986, p. 287. Reproducido con permiso de Prentice Hall. Englewood Cliffs, New Jersey. Figura 7.19: de M. Maekawa, A. Oidehoeft, y R. Oldehoeft, *Operating, Systems: Advanced Concepts.* Copyright © 1987. Reproducido con permiso de Benjamin/Cummings Publishing Co. Tabla 8.6: de James Martin, *Principies of Data Comuinnication.* © 1988. p. 290. Reproducido con permiso de Prentice Hall, Englewood Cliffs, New Jersey. Figuras 8.15 y 8.16:de Raphael A. Finkel, *An Operating, Systems Vade Mecum,* 2e, © 1988. pp. 35 y 36. Adaptado con permiso de Prentice Hall. Englewood Cliffs, New Jersey. Figura 8.17: From Maurice J. Bach, *The Design ofthe UNIX®' Operating System,* © 1986, p. 257. Reproducido con permiso de Prentice Hall. Englewood Cliffs. New Jersey. Figura 9.1: From Charles H. Sauer y K. Mani Chandy,*Computer Svstems Performance Modeling,* © 1981, p. 25. Reproducido con permiso de Prentice Hall, Englewood Cliffs, New Jersey. Figura 9.9:de Maurice J. Bach, *The Design of the UNIX® Operating System.* © 1986. p. 253. Reproducido con permiso de Prentice Hall, Englewood Clifts, New Jersey. Tablas 10.4 y 11.1 y Figura 11.6: de G. Wiederhold, *File Organization for Database Design.* © 1987, pp. 75. 225, y 420. Reproducido con permiso de McGraw-Hill. Inc. Figura 11.1: de Daniel Grosshans, *File Svstems: Design & implementation* © 1986, p. 395. Adaptado con permiso de Prentice Hall. Englewood Cliffs, New Jersey. Figura 12.25: de S. Krakowiak y D. Beeson. *Principies of Operating; Systems.* Copyright © Massachusetts Institute of Technology 1988. Publicado por MIT Press.