SISTEMAS OPERATIVOS II

2026

TEMAS – UNIDAD 1

- **Introducción**. Historia de LINUX (desde UNICS hasta LINUX). Objetivos Interfaces. El Shell. Programas Utilitarios. Estructura del Kernel.

- **Procesos:** Conceptos Fundamentales. Llamadas al sistema para administrar procesos. Implementación de procesos e hilos. Planificación. Arranque.

- **Interbloqueo:** Caracterización. Detección y Recuperación. Prevención. Predicción.

HISTORIA DE LINUX

SO	CARACTERISTICAS
MULTICS (1964)	Todo comienza con un proyecto llamado **MULTICS** (Multiplexed Information and Computing Service), desarrollado por: MI, GeneralElectric, Bell Labs. Su objetivo era crear un sistema operativo: • Multiusuario • Multiprogramado • Seguro • Compartido por muchos usuarios simultáneamente Sin embargo, el proyecto era muy complejo y Bell Labs decidió abandonarlo
UNIX (1969)	Después de abandonar MULTICS, dos investigadores de Bell Labs: Ken Thompson Y Dennis Ritchie desarrollaron un sistema más simple que heredaba muchas ideas de MULTICS. Nació así UNIX en 1969 con características revolucionarias • Multiusuario • Multitarea • Sistema de archivos jerárquico • Uso intensivo de comandos pequeños combinables

HISTORIA DE LINUX

SO	CARACTERISTICAS
UNIX EN C (1964)	Esto fue una revolución porque permitió que UNIX: • Fuera portable • Pudiera ejecutarse en distintos tipos de hardware • Se difundiera rápidamente en universidades y empresas.
BSD y la expansión de UNIX (1977)	La Universidad de California en Berkeley recibió una licencia de UNIX y creó: BSD (Berkeley Software Distribution)BSD incorporó muchas mejoras: • Redes TCP/IP • Herramientas avanzadas • Mejoras en el kernel BSD se convirtió en una rama muy importante de la familia UNIX.
Proyecto GNU (1983)	Richard Stallman inició el proyecto GNU con un objetivo muy ambicioso: Crear un sistema operativo completamente libre compatible con UNIX. GNU desarrolló: • Compiladores • Shell • Editores • Bibliotecas • Utilidades Pero le faltaba una pieza fundamental: **el kernel**

HISTORIA DE LINUX

SO	CARACTERISTICAS
MINIX (1987)	Andrew Tanenbaum creó MINIX. MINIX era: • un sistema tipo UNIX • Pequeño • Educativo • Utilizado para enseñar Sistemas Operativos. Muchos estudiantes aprendieron UNIX gracias a MINIX.
Linux (1991)	En 1991 un estudiante finlandés de la Universidad de Helsinki : Linus Torvalds comenzó un proyecto personal. Su idea inicial era crear un kernel mejor que MINIX para su computadora 80386. El 25 de agosto de 1991 publicó un famoso mensaje en Internet anunciando su nuevo proyecto
GNU/LINUX	Aquí ocurre algo muy importante. GNU tenía: • Shell • Compiladores • Utilidades • Bibliotecas Pero no tenía kernel. Linux era precisamente el kernel que faltaba . Entonces la comunidad combinó: GNU + LINUX = GNU/Linux

GENERALIDADES SOBRE LINUX

OBJETIVOS

- Sistemas Interactivo diseñado para manejar varios procesos y usuarios
- Usuarios sofisticados e involucrados en proyectos de software
- Potencia y Flexibilidad

- Diseñado por Programadores y para programadores
- Sistemas simples, elegantes y consistentes
- Sin Redundancia inútil

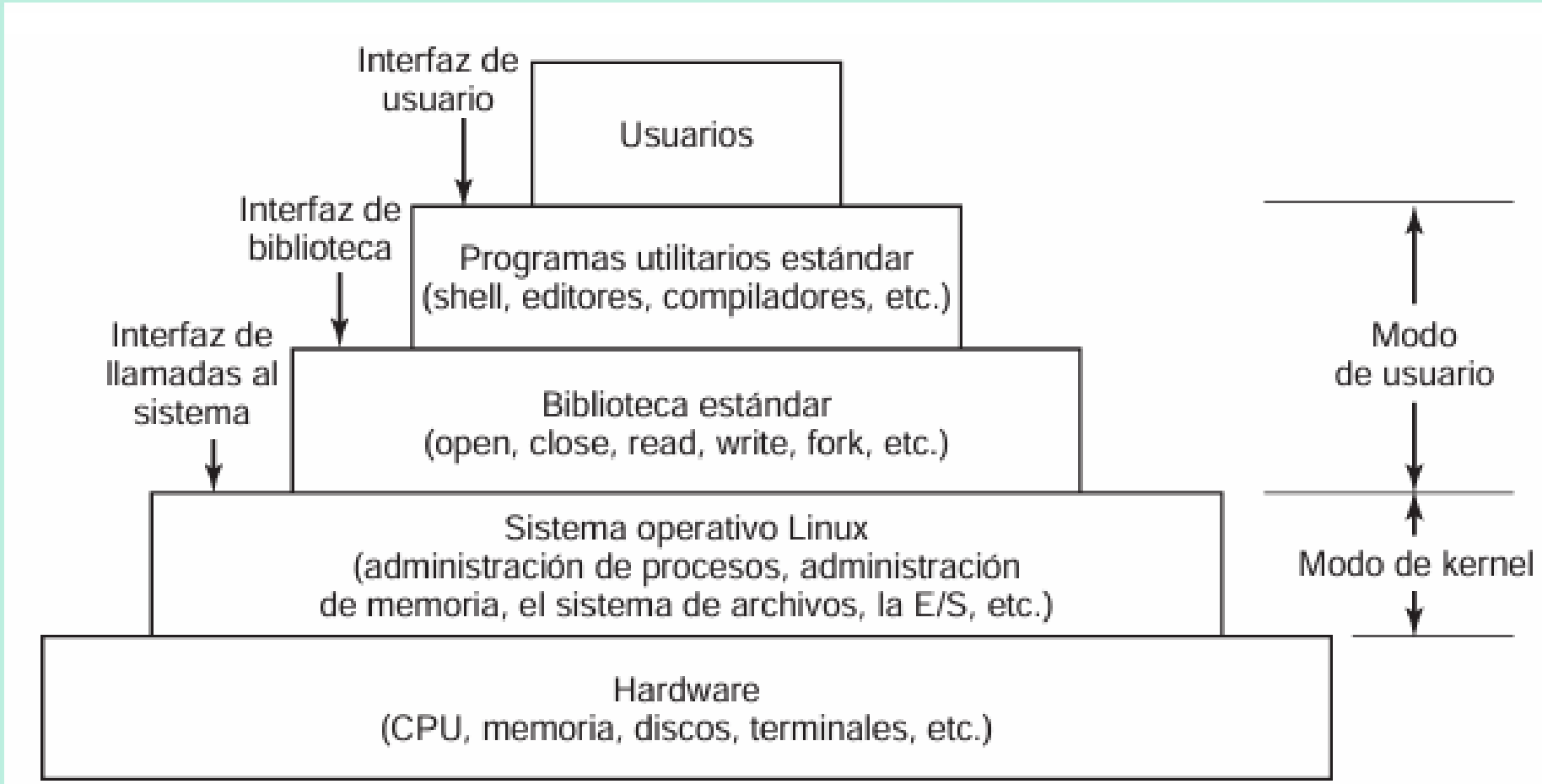
copy → cp

grep ard f

OBJETIVOS

- ✓ Identificar las capas de Linux.
- ✓ Comprender el funcionamiento del Shell.
- ✓ Explicar la creación de procesos.
- ✓ Diferenciar procesos e hilos.
- ✓ Analizar el planificador Linux.
- ✓ Comprender el proceso de arranque.
- ✓ Explicar como Linux trata los interbloqueos

INTERFACES PARA LINUX



"Linux puede verse como una pirámide. En la base está el hardware y encima se encuentra el kernel, responsable de administrar procesos, memoria y dispositivos. Los usuarios interactúan mediante programas y bibliotecas sin acceder directamente al hardware."

INTERFACES PARA LINUX

Linux está organizado en capas que permiten aislar al usuario del hardware.

Componentes

- Hardware.
- Kernel (núcleo del sistema operativo).
- Biblioteca de llamadas al sistema.
- Programas utilitarios.

Linux ofrece tres interfaces principales

1. Interfaz de llamadas al sistema.
2. Interfaz de biblioteca.
3. Interfaz de usuario.

Entornos de trabajo

- Línea de comandos (Shell).
- Interfaz gráfica (GUI).
- GNOME y KDE son entornos populares.

Sistema gráfico

- Basado en X Window System (X11).
- Utiliza servidor X y clientes gráficos.

EL SHELL

DEFINICIONES

El shell (intérprete de comandos) es el programa que permite al usuario comunicarse con el sistema operativo. Su función es:

- Recibir comandos del usuario.
- Interpretarlos.
- Solicitar al kernel que los ejecute.
- Mostrar el resultado en pantalla.

El shell es la interfaz entre el usuario y el kernel.

Los SO Linux tienen GUI, los programadores prefieren la **interfaz de líneas de comandos**

→ Mas rápida de utilizar, mas poderosa y se entiende con facilidad

→ ***Bash*** (Bourne Again Shell) es el más utilizado (predeterminado)

EL SHELL

El Shell se inicia y espera a que el usuario escriba *una línea de comandos*

El Shell extrae la primera palabra y asume que es el nombre de un *programa a ejecutar*

Comandos: Nombre + Argumentos

cp org dest

Modificadores o Banderas

head -20 archivo

Comodines

→ cualquier cadena.

? → cualquier carácter.

[abc] → cualquiera de esos caracteres.

Tuberías y redirecciones

sort ent < | head -30

Cada vez que el shel se inicia (o cualquier programa) tiene acceso a un archivo llamado **entrada estándar** (para leer), a un archivo llamado **salida estándar** (para escribir la salida) y un archivo llamado **error estándar** (para escribir mensajes de error)

EN EL SO LINUX TODO ES UN ARCHIVO

PROGRAMAS UTILITARIOS

La interfaz de línea de comandos consiste en un gran número de programas utilitarios:

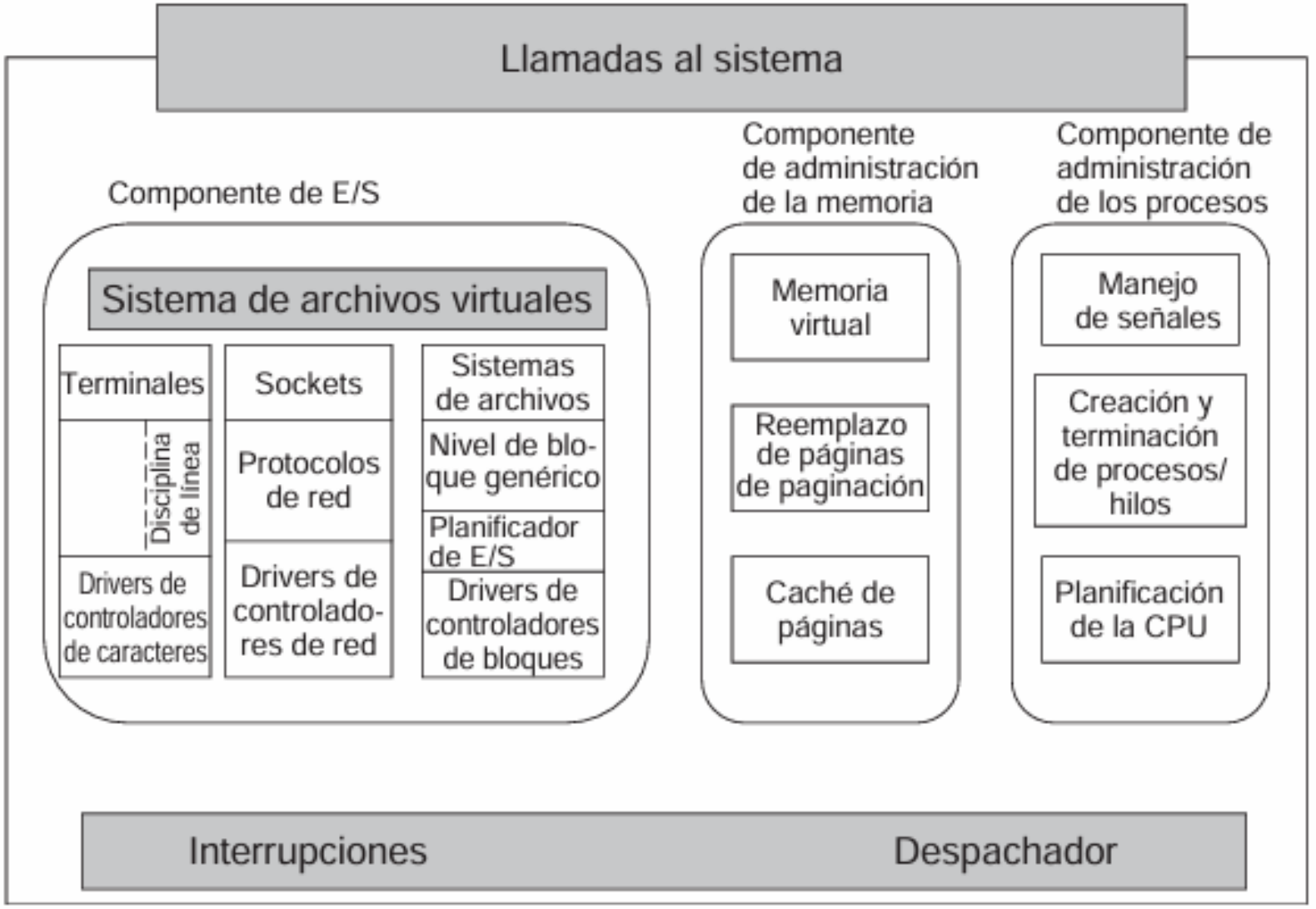
- Comandos de manipulación de archivos y directorios
- Filtros
- Herramientas de desarrollo de programas, como editores y compiladores
- Procesamiento de texto
- Administración del sistema
- Miscelaneos

Programa	Uso común
cat	Concatena varios archivos y los coloca en la salida estándar
chmod	Modifica el modo de protección del archivo
cp	Copia uno o más archivos
cut	Recorta columnas de texto de un archivo
grep	Busca cierto patrón en un archivo
head	Extrae las primeras líneas de un archivo
ls	Lista un directorio
make	Compila archivos para crear un binario
mkdir	Crea un directorio
od	Realiza un vaciado octal de un archivo
paste	Pega columnas de texto en un archivo
pr	Da formato a un archivo para imprimirlo
ps	Lista los proceso en ejecución
rm	Elimina uno o más archivos
rmdir	Elimina un directorio
sort	Ordena un archivo de líneas en forma alfabética
tail	Extrae las últimas líneas de un archivo
tr	Traduce entre conjuntos de caracteres

El estándar POSIX especifica la sintaxis y semántica para al menos 100 de estos programas, principalmente de las tres primeras categorías

La idea de estandarizarlos es que sea posible para cualquiera escribir secuencias de comandos de Shell que utilicen estos programas y funcionen en todos los sistemas Linux

ESTRUCTURA DEL KERNEL



El kernel de Linux es el núcleo del sistema operativo. Actúa como intermediario entre el hardware y las aplicaciones, gestionando procesos, memoria, dispositivos, sistema de archivos y seguridad. Linux utiliza un kernel monolítico modular que puede ampliarse mediante módulos cargables dinámicamente.

PROCESOS EN LINUX – CONCEPTOS FUNDAMENTALES

¿Qué es un proceso?

- Programa en ejecución.
- Tiene memoria propia.
- Posee uno o varios hilos.

Linux es multiprogramado

- Varios procesos ejecutándose simultáneamente.
- Puede haber cientos o miles de procesos activos.

Demonios (Daemons)

Procesos que funcionan en segundo plano.

Ejemplo:

- cron

PROCESOS EN LINUX – CREACIÓN DE PROCESOS: FORK()

Llamada principal

```
pid = fork();
```

Resultado

- Padre.
- Hijo.

El hijo es inicialmente una copia exacta del padre.

Diferencia importante

Fork() devuelve:

- PID del hijo al padre.
- 0 al hijo.

PROCESOS EN LINUX – EJECUCIÓN DE PROGRAMAS: EXEC()

Función

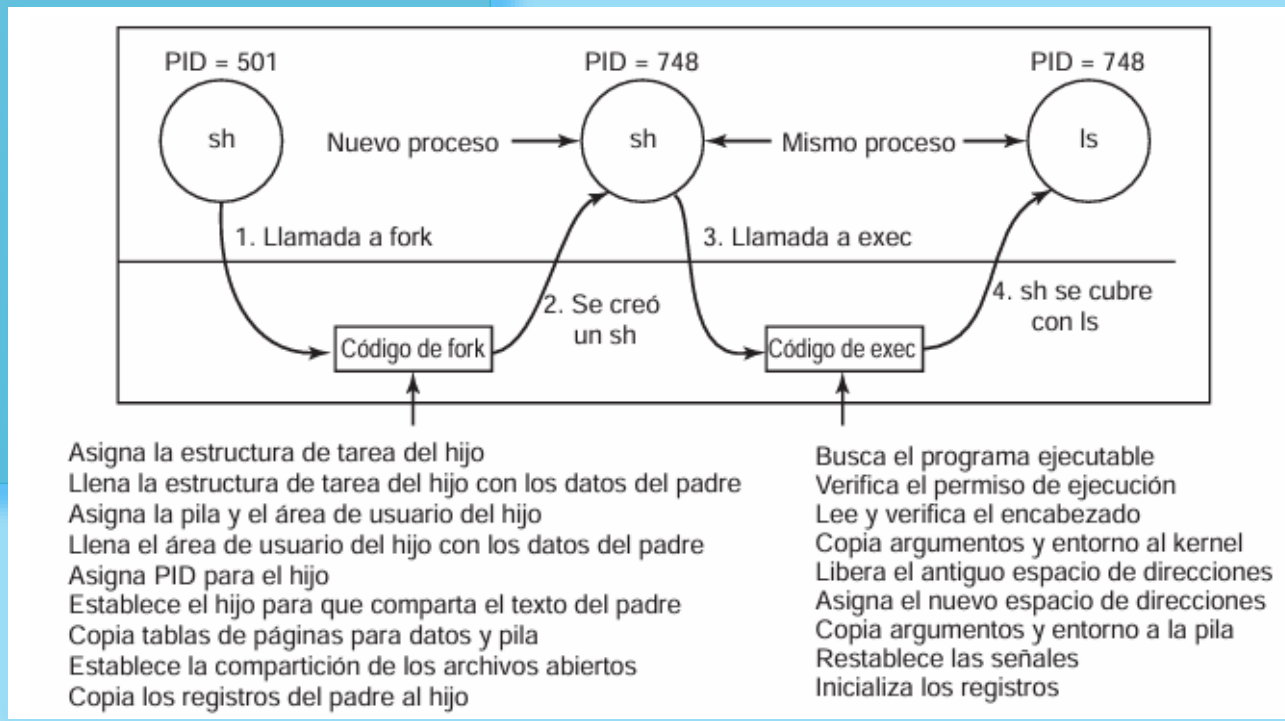
execve()

Objetivo

Reemplaza completamente la imagen del proceso por otro programa.

Secuencia típica

1. Shell ejecuta fork().
2. Se crea un hijo.
3. El hijo ejecuta exec().
4. Se carga el nuevo programa.



Ejemplo

Shell → fork → hijo → exec(ls)

PROCESOS EN LINUX – SEÑALES

¿Qué son?

Interrupciones de software que permiten comunicación entre procesos.

Señales importantes (requeridas por POSIX)

Señal	Uso
SIGTERM	Finalización normal
SIGKILL	Finalización forzada
SIGALRM	Temporizador
SIGFPE	Error aritmético
SIGSEGV	Acceso inválido a memoria

PROCESOS EN LINUX – PROCESOS E HILOS

Particularidad

- Linux representa procesos e hilos mediante estructuras de tarea (task_struct).
- Identifica a los procesos a través del PID

Llamada clone()

Permite decidir qué compartir:

- Memoria.
- Archivos.
- Señales.
- Directorios.

Ventaja

Gran flexibilidad para crear procesos o hilos.

Descriptores de Procesos

1. Parámetros de Planificación
2. Imagen de Memoria
3. Señales
4. Registros de la máquina
5. Estado de Llamadas al Sistema
6. Tabla de descriptores de archivos
7. Contabilidad
8. Pila del kernel
9. Misceláneos

PROCESOS EN LINUX – PLANIFICACIÓN DE LA CPU

Clases de planificación

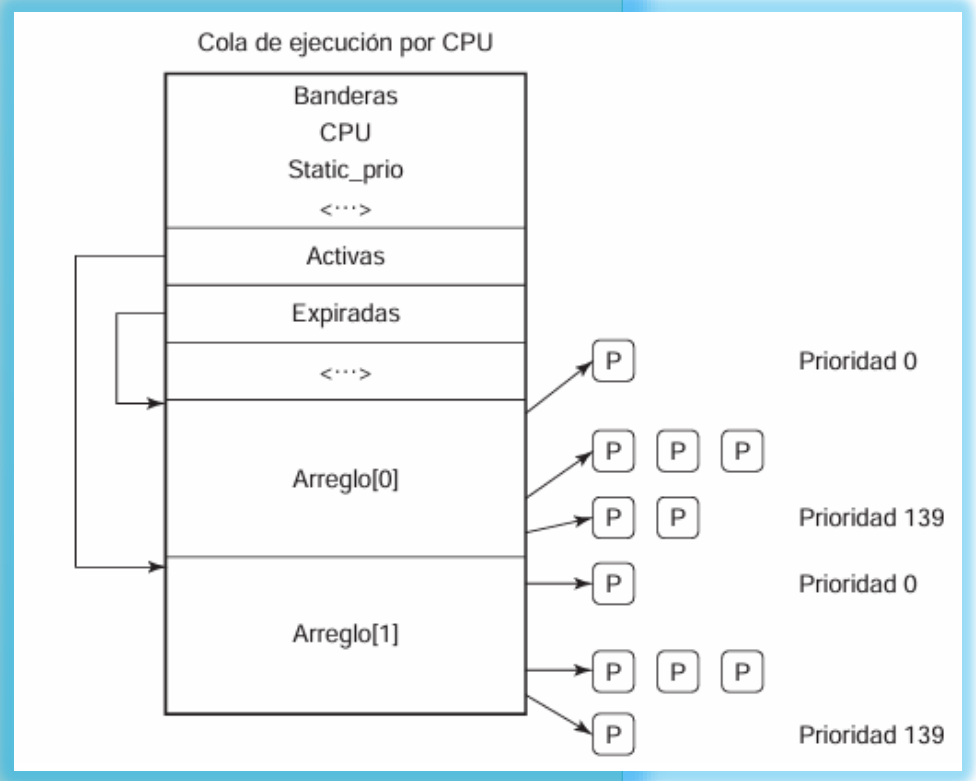
1. Planificación "Primero el llegar, primero es salir" FIFO en tiempo real.
2. Planificación Circular Round Robin en tiempo real.
3. Tiempo compartido.

Conceptos

- Prioridades.
- Quantum.
- Cola de ejecución (Runqueue).

Objetivo

Distribuir el tiempo de CPU de manera eficiente y justa.

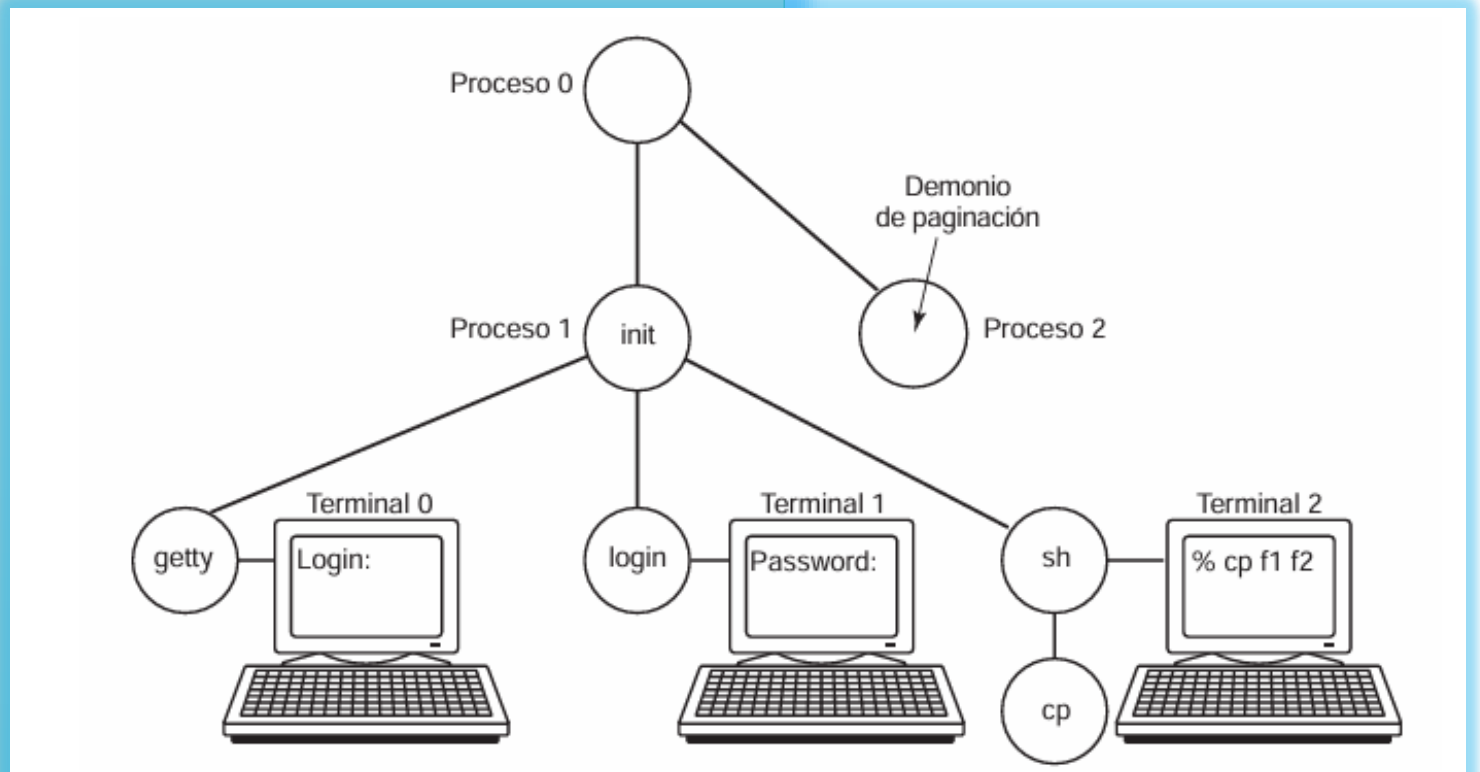


Linux favorece procesos interactivos y mantiene balance de carga

ARRANQUE DE LINUX

Secuencia de inicio

1. BIOS / POST.
2. MBR.
3. Bootloader (GRUB).
4. Kernel.
5. Proceso 0.
6. init.
7. login.
8. shell.



Resultado

Sistema listo para recibir comandos del usuario.

"El proceso de arranque comienza con el BIOS y culmina cuando el usuario obtiene un shell para interactuar con el sistema.

CONCLUSIONES

✓ Linux está organizado en capas.

✓ El Shell es la principal herramienta de interacción por comandos.

✓ Los procesos se crean mediante fork()

✓ La ejecución de nuevos programas se realiza con exec().

✓ Las señales permiten comunicación y control de procesos.

✓ Linux utiliza una planificación avanzada basada en prioridades.

✓ El kernel administra procesos, memoria y dispositivos de forma integrada

INTERBLOQUEOS

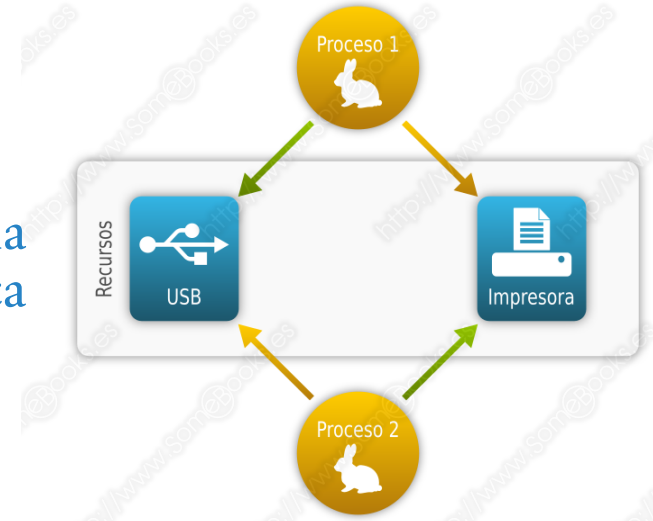
¿Qué pasa si dos procesos se esperan mutuamente para siempre?

Ejemplo cotidiano

Dos automóviles llegan simultáneamente a un puente angosto desde extremos opuestos.

- Ninguno puede avanzar.
- Ninguno quiere retroceder.
- Ambos esperan.

"Un interbloqueo es exactamente esa situación dentro de una computadora. Los procesos quedan esperando recursos que nunca podrán obtener."



INTERBLOQUEOS

Definición

Un conjunto de procesos está en interbloqueo cuando cada proceso del conjunto espera por un recurso retenido por otro proceso del mismo conjunto.

Ejemplo

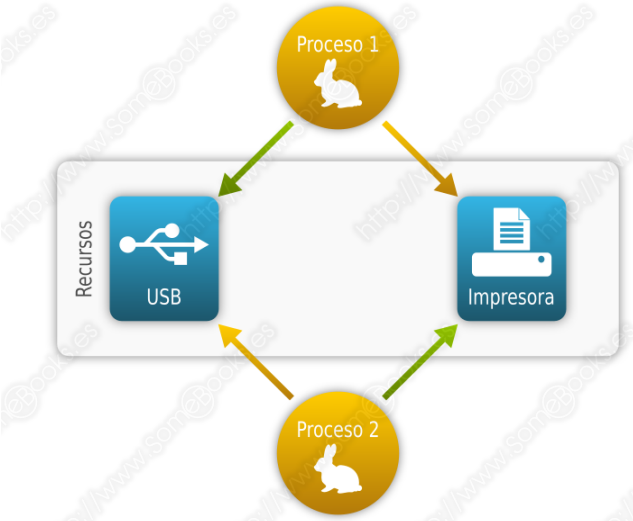
Proceso P1:
Tiene impresora.
Espera USB.
Proceso P2:
Tiene USB.
Espera impresora.

Resultado

Ninguno continúa.

Pregunta

¿Puede resolverse solo?



INTERBLOQUEOS

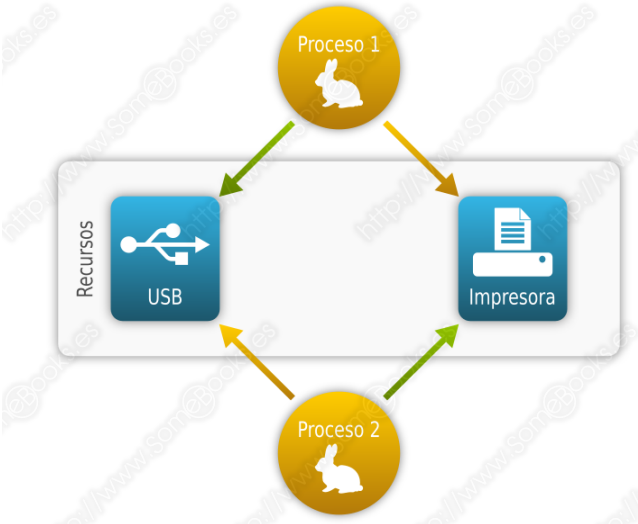
Recursos del sistema

Recursos reutilizables

- CPU
- Memoria
- Impresoras
- Archivos
- Semáforos

Recursos consumibles

- Mensajes
- Señales
- Eventos



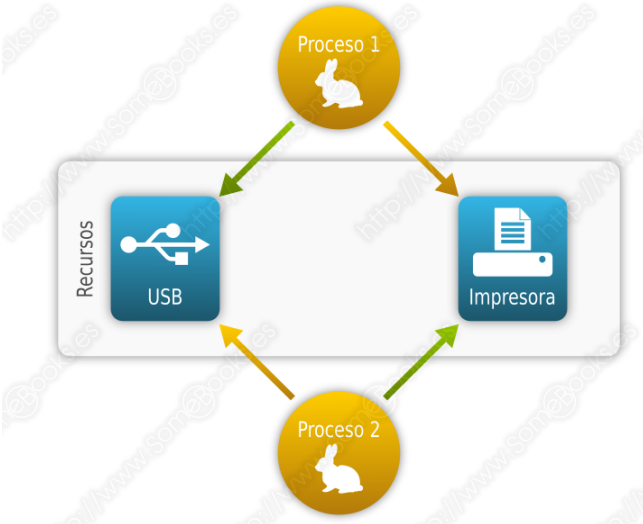
INTERBLOQUEOS

CONDICIONES DE COFFMAN

- Exclusión mutua.
- Retención y espera.
- No expropiación.
- Espera circular.

ESTRATEGIAS CONTRA DEADLOCK

- Evitación (Algoritmo del Banquero)
- Prevención (de algunas de las 4 condiciones de Coffman)
- Detección y Recuperación
- Ignorar el problema (Algoritmo del Avestruz)



INTERBLOQUEOS

LINUX Y LOS DEADLOCKS

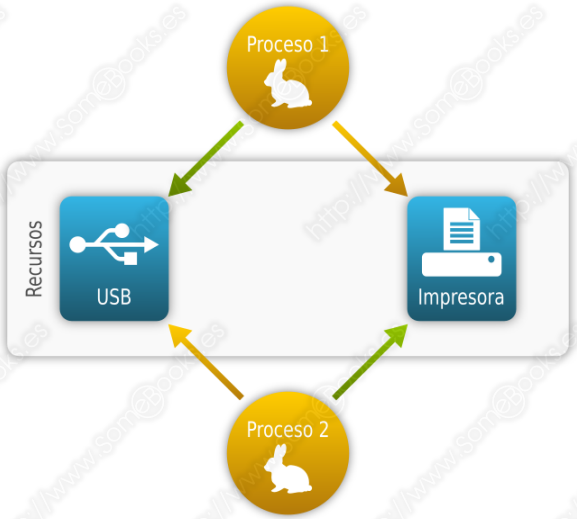
Linux utiliza:

- Mutex
- Spinlocks
- Semáforos
- RW Locks

Pero si se utilizan incorrectamente pueden producir:

- ✅ Espera circular
- ✅ Retención y espera
- ✅ Deadlock

Los mutex, spinlocks, semáforos y RW locks son herramientas de sincronización. Su objetivo es evitar condiciones de carrera. Sin embargo, si se adquieren en un orden incorrecto pueden transformarse en la causa de un interbloqueo."



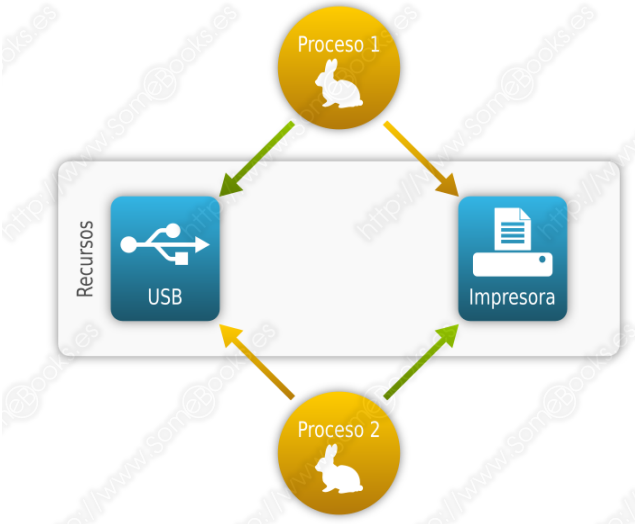
INTERBLOQUEOS

Linux no utiliza algoritmos generales de evitación como el **Algoritmo del Banquero** para todos los procesos. En cambio, se basa principalmente en **prevención y buenas prácticas de sincronización**, especialmente dentro del kernel.

BUENAS PRACTICAS DE LINUX

✓ Orden global de adquisición.
✓ Bloqueos de corta duración.
✓ Timeout.
✓ Herramientas de depuración.
✓ Diseño concurrente cuidadoso.

"Linux combate los interbloqueos principalmente eliminando la condición de espera circular mediante políticas de adquisición ordenada de locks. En lugar de detectar todos los deadlocks posibles, se concentra en evitar que puedan formarse."



MISCELÁNEOS LINUX – SOFTWARE LIBRE

Libertad 0: Ejecutar el programa como se desee y con cualquier propósito.

Libertad 1: Estudiar cómo funciona el programa y adaptarlo a las necesidades propias. Esto requiere acceso al código fuente.

Libertad 2: Redistribuir copias para ayudar a otros usuarios o a la comunidad.

Libertad 3: Distribuir versiones modificadas del programa original, permitiendo que toda la comunidad se beneficie de las mejoras

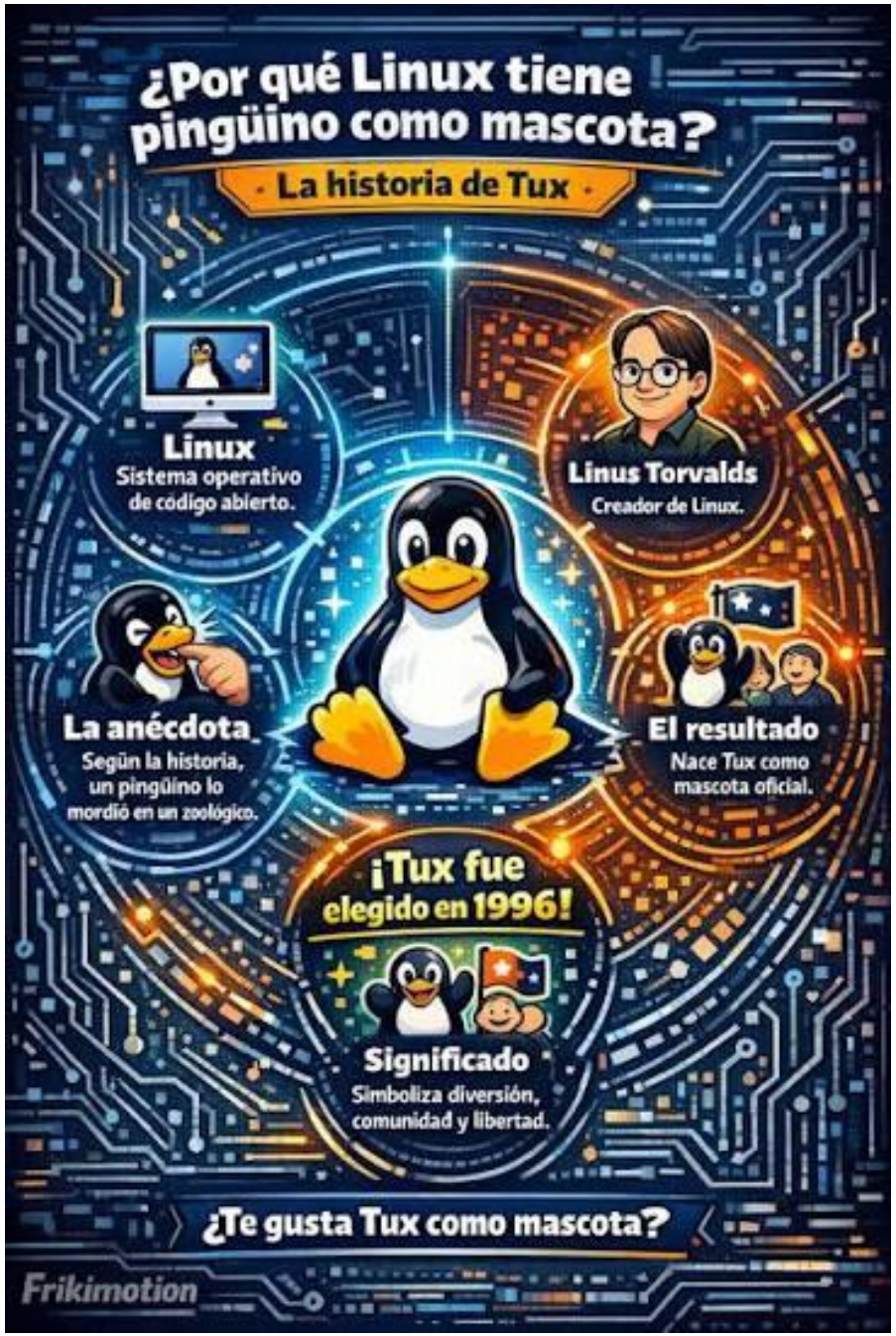
Acceso al código fuente: Las libertades 1 y 3 exigen que el código fuente esté disponible para el usuario.

No es solo cero costo: El software libre se enfoca en la libertad de uso, no en el precio. Se puede comercializar o vender, siempre y cuando se mantengan las libertades al entregarlo.

Definición oficial: Un programa solo se considera software libre si respeta íntegramente estas cuatro libertades a través de su licencia.

DISTRIBUCIONES DE LINUX

El ecosistema de **Linux** se compone de cientos de versiones llamadas **distribuciones** (**o distros**). Cada una está diseñada con una filosofía, interfaz y propósito técnico diferente para adaptarse a las necesidades de cada usuario



MUCHAS GRACIAS

Ing. María Fernanda Vazquez

fvazquez@fi.unju.edu.ar