

Tema N° 6

Memoria Virtual

Nombre | Nombre del curso | Fecha

Memoria Virtual

La Memoria Virtual es una técnica que permite tener procesos más grandes que la memoria física disponible.

Hasta a ahora todas las técnicas que vimos eran que el proceso tenía que estar completo en memoria, sino teníamos que usar overling o swaping.

**Paginación por Demanda**

Es decir que el proceso no está en memoria, sino que va a ir subiendo las páginas a medida que las necesite. De esta manera vamos a lograr que:

1. Utilizamos menos memoria, porque solamente vamos a tener en memoria lo que necesitamos.
2. Se va a utilizar mucho la técnica de intercambio, ya que como podemos contener más procesos en memoria, los procesos pueden ser más grandes y va a ir subiendo las páginas a medida que se la necesiten, aquellas páginas que no sean necesarias para el proceso van a tener que estar en disco.

Por eso es que nos permite que todo el proceso no esté en memoria, por ello el proceso puede ser más grande que la memoria disponible, porque solamente se va a subir a la memoria principal aquellas páginas que se necesiten porque la dirección de memoria a la que el microprocesador está haciendo referencia se encuentran en dicha página y que debe estar en memoria, el Bit de **validez/invalidez** ahora es muyimportante porque el Bit de validez nos va a decir que la página está en memoria y si esta inválido no dirá que la página no está en memoria.

* **El Intercambio de páginas** ya no se llama intercambio porque es un intercambiador PEREZOSO que solo trabaja si necesita una página, entonces toma el nombre de **Paginador**.

La CPU genera direcciones lógicas que están compuestas por No de Página y desplazamiento, con esta dirección se dirige a la Tabla de Páginas que contiene No de Página, No de Marco y el bit de validez/invalidez. Cuando entra a la tabla de páginas con el No de Página encuentra el No de Página al que hacemos referencia, pero el **bit es de invalidez** entonces ocurre lo que se conoce como **Fallo de Página** (que es un interrupción no enmascarable), entonces llamamos al S.O, específicamente la **PAGINADOR** que será el que va a ir a buscar el No de Página en la **zona de intercambio.**

Puede ocurrir que no se tenga **zona de intercambio**, entonces el intercambio será más lento, porque está disperso en el disco y va a tener que ser manejado por un gestor de disco. Entonces el paginador la buscará y traerá a memoria principal a un marco libre y regresará a la tabla de páginas y cambiará el **Bit a válido**, porque ya está en memoria.

Cuando se llama al S.O ocurre un cambio de contexto, porque el proceso va a pasar a bloqueado hasta que se cargue en memoria, una vez que esto suceda el proceso se va a Listo y se ejecutará cuando le toque su turno. Esto quiere decir que:

* **Cada Fallo de página** obliga al proceso a liberar el microprocesador, por eso lo óptimo es tratar de generar la menos cantidad de **fallos de páginas** posible.
* **Cuando arranca por primera vez** un proceso se dice que el proceso hace Paginación por Demanda Pura, es decir que va a generar varios fallos de páginas hasta que cargue las páginas necesarias, de ahí en adelante el proceso se va a estar ejecutando con las paginas ya cargadas en memoria, por eso es importante la modularidad de los programas, porque cuando más modular sea menos fallos de páginas vamos a tener mientras trabaje siempre con las mismas páginas, por eso es importante que no se generen demasiada cantidad de variables, ni tampoco que se usen demasiadas Variables Globales porque pueden estar dispersos en varias páginas, ni tampoco estructuras dinámicas que van a necesitar memoria.



* **¿Qué ocurre si encontramos que toda la Memoria está ocupada?**

En el caso que ocurra tendremos que elegir a alguna página para ser sacrificada. En ese caso se dice que la página que se va a elegir es la **Página Victima**.



Cuando se elige la página Victima, si la sobrescribimos perdemos, si es un área de Datos, si es un área de Códigos no hay problema porque esto ya está en la **Zona de Intercambio** del disco y no sufrió modificación mientras se ejecutaba el proceso.

Pero si justo la página contiene datos, estos podrían haber sido modificados, y si fueron modificados si tendremos que grabar esta página en disco (zona de intercambio) y recién de la zona de intercambio vamos a poder cargar la nueva página.

Esto implica que vamos a tener dos accesos al disco, uno para escritura y el otro para lectura, por lo tanto el intercambio es el doble de lento. Por esta razón a la tabla de páginas se le va a agregar un **Bit más llamado Bit de Modificado o Bit Sucio**.

* **Bit Modificado o Bit Sucio**: va a decir si la página sufrió alguna alteración, es decir si fue modificado o no.

Entonces de todas las víctimas o posibles víctimas me conviene elegir las páginas que tengan el **Bit de Modificado** en cero, y de esta forma anulo uno de los accesos a disco.

Además es importante saber que de todas las posibles víctimas, nos convendría elegir una que no vaya a ser referenciada inmediatamente después, porque supongamos que un proceso viene y genera un fallo de página, viene el paginador y tiene que elegir a que página sacar, pero mientras tanto entra otro proceso que justo necesita esa página víctima, entonces este va a generar otro fallo de página. Por esta razón también se agrega aparte del bit de modificado, un bit más que se llama **Bit de Referencia**.

* **Bit de Referencia:** cuando su **valor es igual a 1** significa que recientemente fue referenciado por ese proceso, si su **valor es igual 0** a quiere decir que no ha sido referenciado.

Por lo que nos convendría elegir una página que tenga **Bit de modificación** en 0 y **Bit de referencia** en 0.

**Algoritmo de Reemplazo de Página**

Para ver los algoritmos de reemplazo de página necesitamos una serie de referencias, que son una serie de llamadas a páginas. ¿Cómo obtenemos esa serie de referencias?

Existen dos formas de obtenerlas:

* Una forma es ejecutando un programa y guardando en un registro todas las direcciones de memoria que aparecen, una tras otra. Pero ¿qué pasa con esto? Se guardara mucha basura, porque obtendremos un número donde una parte es el desplazamiento y la otra parte el Nº de Página, a nosotros solo nos interesa el Nº de Página. Y se sabe que mientras las referencias sean a una página, lo único que cambia es el desplazamiento, y lo que nos interesa es el cambio de página.
* Otra forma es generando números al azar y diciendo que esos números son Nº de páginas.

🡪Por tanto, el Nº de referencia es referencia a una página.

🡪También necesitamos saber cuántos marcos hay en memoria.

* **FIFO:** Primero en entrar, primero en salir.



El FIFO tiene un problema porque no le da importancia a las referencias, ni nada, o sea que es un algoritmo que genera demasiados fallos de páginas, es más fácil de hacer pero tiene el problema de que puede estar retirando una página que va a ser necesitada próximamente.

A parte de tener la mayor cantidad de fallos de páginas, tiene otro problema, sufre de Anomalía de Belady, que dice que si a FIFO se agregan más marcos a veces genera igual o más fallos de páginas, porque no depende de la cantidad de marcos sino depende del orden en el que se suceden las llamadas.

* **Optimo u Opt – Min** (Algoritmo base)

El algoritmo óptimo dice que hay que ver las referencias futuras para ver si es posible o no cambiarlas, o sea es mirar hacia el futuro que páginas voy a necesitar más adelante para no sacar esas páginas, pero electrónicamente no se puede hacer, por lo tanto es imposible realizarlo pero es el que genera menor cantidad de fallos de páginas y lo óptimo para cualquier algoritmo es acercarse más a este.



* LRU: El último recientemente Usado (Menos recientemente usado). Vemos las referencias pasadas para predecir el futuro, o sea vemos que paso como se fueron referenciando las páginas y según eso vamos a elegir la víctima.

No es el óptimo pero es el que más se aproxima, por eso el LRU es el algoritmo usado en las Bases de Datos, pero así como esta no se tiene un bit de referencia, lo que se necesita es un tiempo (el menos recientemente usado), es decir un reloj que nos diga hace cuánto tiempo no referenciamos, por eso el **LRU Puro** necesita muchísima ayuda del Hardware, tenemos que tener un contador o un reloj y dentro de la memoria tenemos que tener la capacidad de guardad ese tiempo.

Las computadoras que no tienen ayuda de Hard, la única forma de hacer esto es a través de estructuras de datos.

* **Pila o Lifo**: el último en entrar, el primero en salir.



* **Estructura Dinámica**: Lista Doblemente Enlazada

Esto es costoso en cuanto a memoria utilizada, es decir que el área de acumulo crece por cada proceso y aparte es lento si no tenemos los punteros necesarios.

Por esta razón las computadoras pequeñas usan lo que se conoce como:



* **Aproximación al LRU** 
  + Bit de Referencia: Es agregar un bit de referencia. Algunos para hacerlo más completo en vez de elegir un bit, usan un byte.

**Bit Histórico**

1. Cuando una página se referencia, por ejemplo cuando hacemos el primer fallo de pág., se le agrega **1** en el de mayor peso, o sea a la izquierda y se desplaza hacia la derecha.
2. En la segunda referencia (a página 2), y en la página uno corremos el **1** hacia la derecha.

Si tenemos que elegir una víctima ¿A quién vamos a elegir de estos dos? Se elegirá el que tenga menor valor, entonces elegimos la página **1.**

**🡪**En el **Caso de tener un Bit** existen algoritmos, y el algoritmo que usa ese bit se llama:

**Algoritmo del Ascensor o del Reloj o de la 2da Oportunidad**



Como se tiene el bit de referencia, tenemos un puntero que verifica si tiene un **1** la página, se la coloca en **0** y pasa al próximo, si la próxima página tiene un **0** es la víctima (2º). Si en una 3º referencia en la que se tiene cero, esta será la víctima y se cambiará el **0** por **1,** y el puntero queda listo para leer la siguiente página en la próxima referencia.

**Algoritmo de la 2da Oportunidad Mejorado**

Este algoritmo utiliza tanto el Bit REFERENCIADO como el Bit MODIFICADO, trabaja igual que el algoritmo del Reloj con la diferencia que usa un par ordenado.



Pregunta primero si el **Bit de Referencia es cero**, si es cero verifica el Bit Modificado si es cero, es la **Mejor Víctima.**

Si el **Bit de Referencia es uno**, la página fue modificado, entonces se debe **Guardar en Disco.** Ya no es tan buena víctima porque tiene un acceso a disco.

Si el **Bit de Referencia es uno**, y el Bit de Modificado es cero, puede generar un **Fallo de página**, porque puede ocurrir de que necesitemos la página en el próximo pedido porque recién fue referenciada.

Si el **Bit de Referencia es uno**, y el **Bit de Modificado es uno**, es la **Peor Víctima**, porque puede generar fallo y además genera un acceso al Disco. Si es posible no utilizarlo nunca.

**Algoritmos de Conteo (hablan de Frecuencia, porque no cuentan)**

Consiste en contar cuantas veces fue referenciada una página.

**LFU:** Menos frecuentemente usado (-). Cuando el Contador es más pequeño ese va a ser la víctima. Ej: Rutinas de Error.

**MFU**: Más frecuentemente usado (+). Cuando el contador es más grande esa es la víctima. Ej. cuando se realizan instalaciones.

Hay que tener cuidado con los algoritmos de conteo porque los Contadores tienen un límite y si hay un **Desbordamiento**, ya no hay lugar para contar, y vuelve a cero.

Se usa uno o el otro dependiendo del tipo de programa que se usa en esa computadora.

**Algoritmo de Colocación de Página en Buffer**

En realidad no es un algoritmo de reemplazo de página, sino es un método para acelerar el intercambio. Lo que hace en realidad la técnica es lo siguiente:

Como el S.O tiene la capacidad de crecer, es decir puede tomar la memoria del usuario y guardarlo para él, cuando se elige la víctima en vez de hacerla desaparecer o guardarla en disco, en realidad la guarda en una página dentro sus páginas de su memoria. Entonces si esa página vuelve a ser referenciada, en vez de ir a la zona de intercambio del disco para volverla a cargar primero se fija en su Buffer, si está en su Buffer lo activa como parte de memoria del usuario y no hace ningún cambio.

**Asignación de Marcos**

¿Por qué necesitamos saber cuántos marcos tiene cada proceso?

Porque todo proceso para que puede ejecutarse bien, tiene un mínimo de cantidad de marcos. Cualquier instrucción por más sencilla que sea mínimamente va a necesitar dos páginas o dos marcos, porque uno es la instrucción y el otro la variable sobre la cual actúa esa instrucción.

🡪Los procesos siempre va a necesitar más de un marco de memoria para poder ejecutarse, porque supongamos que tenemos el área de código cargada en memoria, y que ocurre cuando queremos ir al área de datos y no está en memoria, va haber un fallo de página, si un proceso tiene pocos marcos asignados o menos marcos que los que necesita va a empezar a generar fallos de páginas. Entonces ese proceso no se va a poder ejecutar porque va a estar continuamente en BLOQUEADO en espera de que le suban las páginas.

🡪Entonces, por eso el S.O tiene que ir controlando el nivel de multiprogramación, de manera que los procesos que están contenidos en memoria se puedan ejecutar y no generen demasiados fallos de páginas. Por eso el S.O lo que hace es tener un Contador de da cantidad de marcos que debe tener cada proceso en memoria.

🡪Existe un registro que se llama REGISTRO DE INDIRECCION donde se contiene la cantidad de marcos que debe tener ese proceso en memoria.

🡪Cada vez que se genera un fallo de página, va a ir descontando ese contador, porque ya se generó un fallo para ocupar un marco que necesitaba. Cuando ese contador llegue a cero, si el proceso vuelve a generar un fallo de página dará un error, o sea se va a producir una interrupción no enmascarable, que muestra un mensaje que dice error de indirección.

🡪**Error de Indirección**: El proceso tiene asignados menos marcos que lo que él necesita. Entonces ahí puede intervenir el PMP, y decir que como este proceso está generando muchos errores de indirección, lo podemos poner BLOQUEADO SUSPENDIDO, hasta que haya más marcos y le podamos asignar marcos de los que ya tenía.

Es decir, si un proceso está generando demasiados errores de indirección, o sea está generando muchos fallos de páginas y no se puede ejecutar, entonces quiere decir que tenemos un alto grado de MULTIPROGRAMACION por lo tanto debemos bajarlo para que los procesos se puedan ejecutar y tengan más espacio, y es aquí donde trabaja el PMP.

**Algoritmos de Asignación de Marcos de PN**

Estos algoritmos son los que determinan el **valor del contador de indirecciones**.

* **Asignación equitativa**: Va a repartir los marcos en forma igual entre todos los procesos que hay.

**Cantidad de Marcos** = Cantidad de Marcos del PN

Cantidad de Procesos

Si tenemos: 10 procesos

100 marcos

Cantidad de marcos del PN = 100/10 = 10 marcos//.

Entonces, en una Asignación Equitativa para todos los procesos el Contador de Indirecciòn va a ser igual.

* Problema. Hay procesos que son muy pequeños, y no van a necesitar tantos marcos y hay procesos que son muy grandes y necesitan más marcos de los que tienen asignados, entonces se dice que puede haber un desperdicio marcos que no son usados y a la vez que hayan otros procesos que nos estén generando muchos fallos de páginas.
* **Asignación Proporcional**: Se le asigna la cantidad de marcos según el tamaño del proceso.

Cantidad Marcos de Proceso PN= **Tamaño PN / ∑ Tamaño de los procesos \* Total de Marcos.**

**Ejemplo:** 2 procesos, 62 marcos

**P1 10 P1 🡪 10/137\*62 = 4,53 ˜= 4**

**P2 127 P2 🡪127/137\*62 = 57,47 ˜= 57**

**Políticas de Asignación de Marcos**

* **Asignación Local:** Que el proceso solo elegirá la victima dentro de los marcos que él tiene asignado, no puede robarle marcos a otro proceso.

**Ventajas**

* La asignación equitativa o proporcional tiene mayor control y los procesos que están dormidos, cuando vuelven a ejecutarse sabrán que le faltan marcos, es decir, le asegura a cada proceso que los marcos que él está utilizando aunque no se esté ejecutando no se los van a quitar, o sea cuando inicie ese proceso no va a iniciar con un fallo de página.
* Va más acorde con la programación modular.

La asignación local tiene esta ventaja si se programa bien. Ahora si los programadores hacen todo a prueba y error, lo más factible es que necesiten muchos marcos para que el proceso se ejecute, en consecuencia desperdician la cantidad de marcos disponibles porque van a ser programas grandes, así y todo no le van a alcanzar la cantidad de marcos, entonces esos programas no se van a ejecutar nunca, en ese caso conviene las asignaciones globales.

* **Asignación Global**: Todo marco que haya en memoria puede sufrir y ser víctima. El Proceso puede robar marcos a otro procesos. En este caso deja de tener sentido el contador de indirecciones, porque puede llegar a tener más marcos que los que necesita inicialmente.

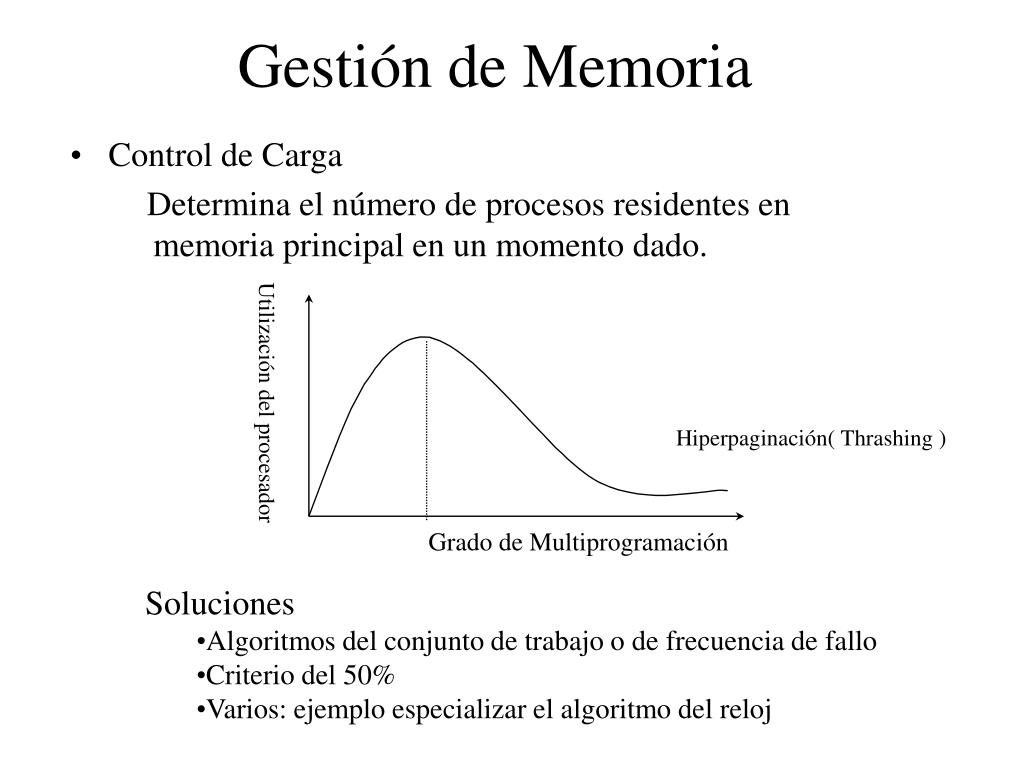
Tenemos esta memoria y hay un proceso que usa dos marcos, el proceso se fue dejando los marcos con el bit de referencia en 1 vuelve y genera una indirección, a los marcos que tienen 1 le dan una segunda oportunidad, y elige otro, entonces ahora el proceso tiene 3 marcos e hizo un solo fallo.

**Hiperpaginación**

La Hiperpaginación se define como el alto grado de fallos de páginas. Dicho de otra forma, cuando un proceso pasa más tiempo paginando que ejecutando.

Esto quiere decir que el proceso está más tiempo en bloqueado, por fallos de páginas, que en el estado ejecutando.

Cuando hay un alto grado de Hiperpaginación tiene que entrar a ejecutarse el PMP.



* **Hiperpaginación:** La CPU se usa continuamente, está siendo usada por el paginador, o sea el S.O, para subir páginas constantemente, es decir se está haciendo intercambio todo el tiempo.

Para solucionar o controlar esto y poder bajar el nivel de multiprogramación, bajando los procesos a Bloqueado Suspendido, liberar marcos de memoria, para que los procesos se ejecuten y bajen, y podamos cargar de nuevo.

Para poder bajar el nivel de multiprogramación existen dos métodos:

1. **El Modelo de Conjunto de trabajo**: Utiliza lo que se conoce como Localidad de un Programa. **La Localidad de un Programa** son las páginas que se están usando en un instante de tiempo; son la cantidad de instrucciones secuenciales que se usan dentro de una o dos páginas, por ese instante de tiempo solo se van a ejecutar esas páginas.

Cuando se cambia de Localidad, quiere decir que hubo un salto a otra área de memoria, y esa área de memoria (esas páginas) también se van a ejecutar por una X cantidad de tiempo, hasta que volvamos a otra localidad y así sucesivamente.

**El Modelo de Conjunto de trabajo**, dice bueno ya que se usa constantemente una cantidad de páginas por un instante de tiempo definamos un conjunto donde vamos a contener todas las páginas que se usan en esa localidad.



* **Problema:** El problema del modelo de conjunto de trabajo es determinar el ΔT, porque si es muy grande vamos a estar abarcando más de una localidad, es decir que vamos a estar reservando marcos de memoria a ese proceso que no los necesita.

1. **Frecuencia de Fallos de Página:**

****

**Consideraciones**:

* Es muy importante determinar cuál es el tamaño de la página.
* Si se tiene páginas chicas implica que, cuando se haga paginación por demanda pura, van a haber mayor cantidad de fallos de páginas, porque las páginas van a tener pocas instrucciones. Entonces, cuanto más chica sea la página nos implica mayor número de fallos de páginas cuando un proceso comienza, por lo tanto mayor cantidad de marcos.
* La ventaja de tener páginas chicas es que habrá menos fragmentación interna.
* Favorece a la localidad de un proceso, más precisamente a determinarla, pero esto implica que el tiempo que vamos a necesitar para determinar el modelo del conjunto de trabajo, va a tener que ser más grande.
* Si se utilizan páginas grandes implica que hay menos fallos de páginas, el modelo de conjunto de trabajo va a perdurar en el tiempo, porque vamos a estar referenciando a la misma página por más tiempo porque tenemos más instrucciones.
* La desventaja fundamental es que tiene mayor Fragmentación Interna.

Por eso es que se ha adoptado hoy en día en usar páginas de un tamaño medio (4kb a 8kb) y normalmente lo que se intenta es que sea equivalente al tamaño del bloque del disco, así ahorrar tiempo.

**Mapear en Memoria**

¿Qué significa mapear en memoria?

Que haya una correspondencia directa de lo que está en Disco, a lo que está en Memoria.

* Como se Programa: Es fundamental elegir buenas estructuras de datos, y que sean como máximo del tamaño de un marco, o división exacta del marco, para evitar la fragmentación.

Es importantísimo elegir un buen lenguaje de programación que optimice el uso de los marcos. Por ejemplo, tenemos una matriz:



En la memoria principal, el marco contiene 4 palabras, entonces en ese marco entra 4 filas de la matriz.

Si se programa que primero se va a leer en columna, cuando llegamos a la columna 4, vamos a tener que ir a un marco N donde van a estar las 4 siguientes columnas, entonces vamos a estar saltando de marco en marco. Ahora, si se dice que el programa no guarda la estructura de datos en filas si conviene leer en columna primero porque toda la columna va a estar ahí.

* Cuando se programa la estructura de Datos debe hacer referencia o como guarda el marco para tratar de no estar saltando de marco a marco.
* Por eso es importante la Estructura de Datos.
* El tamaño de las Estructura de Datos y como acceder a la E.D, en referencia al tamaño de marco o de página que maneja esa computadora, para optimizar el uso de la memoria y para que el proceso sea más rápido.

**Uso de Memoria Virtual de la Tabla de Página Invertida**

La tabla de página invertida para poder usar memoria virtual tiene un grave problema, como es secuencial para leer y es lenta, entonces se le agrega una tabla de páginas a cada proceso. Y es imposible usar paginación por demanda si no tiene una tabla cada proceso.

O sea que la paginación invertida solo va a poder usar memoria virtual, si además de la página invertida, cada proceso tiene una tabla de páginas y esto hace que se gaste demasiada memoria, y por esta razón es que no se la usa.

🡪En un Sistema de Tiempo Real ¿Se puede usar paginación por demanda? ¿Se usa Memoria Virtual? Un sistema de tiempo real no usa la memoria virtual. Es muy poco probable que suceda, porque la memoria virtual conlleva una pérdida de tiempo en fallos de páginas.

Si existiera la **Segmentación por Demanda**.

**Asignación de Memoria del Kernel**

* El Kernel o núcleo del S.O individualmente tiene que tener una forma de manejar la memoria especial, porque el S.O se tiene que ejecutar lo más rápido posible y entregar el servicio al proceso que lo pide, ese es el objetivo del S.O.
* El núcleo lo maneja de dos formas:

1. **Descomposición Binaria**: (Algoritmo de colegas) Trabaja de la siguiente forma, el S.O toma su memoria como un solo bloque y según la estructura de datos que quiera crear, va dividiendo binariamente. Supongamos que le soliciten 64 kb.

Cuando lo use, si lo deja de usar y los libera va a compactar y siempre va a tratar de mantenerlo al bloque más grande y siempre en forma contigua.

* **Consolidación:** Cuando junto los bloques, tratando de tener siempre como un solo bloque lo más grande posible.

1. **Asignación en Franjas**: (Asignador de losas) Lo que intenta es agrupar, todas las estructuras de datos los maneja como objetos.

* Trata de poner uno tras de otro para evitar la fragmentación interna y para tenerlos contiguos.