# PRACTICA 4

**Programa**

- Es estático
- No tiene program cunter
- Existe desde que se edita hasta que se borra

**Proceso**

- Es dinámico
- Tiene program counter
- Su ciclo de vida comprende desde que se lo ejecuta hasta que termina
- Según su historial de ejecución, los podemos clasificar en:
    - CPU Bound (ligados a la CPU): significa que el proceso requiere principalmente de tiempo de procesador para su ejecución
    - I/O Bound (ligados a entrada/salida): significa que el proceso depende en gran medida de operaciones de E/S para su ejecución. Lo ideal es alta prioridad y quantum chico

Performance - CPU Bound - Procesos BATCH

Respuesta - I/O Bound - Interactivos (con el Usuario)

**Tiempo Retorno (TR):** tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución

**Tiempo Promedio de Retorno (TPR):** es el promedio de tiempos de retorno entre todos los procesos pertenecientes al lote. Se calcula como la suma del TR de cada proceso, dividida entre la cantidad de procesos.

**Tiempo Espera (TE):** tiempo que el proceso se encuentra en el sistema esperando, es decir el tiempo que pasa sin ejecutarse. Es básicamente la diferencia entre el tiempo de retorno y el tiempo de ocupación del procesador

**Tiempo Promedio de Espera (TPE):** es el promedio de tiempos de espera calculado entre todos los procesos involucrados en el lote. Se calcula como la suma del TE de todos los procesos, dividida entre la cantidad de procesos.

**Nonpreemptive**: una vez que un proceso está en estado de ejecución, continua hasta que termina o se bloquea por algún evento (e.j. I/O)

**Preemptive**: el proceso en ejecución puede ser interrumpido y llevado a la cola de listos:

- Mayor overhead pero mejor servicio
- Un proceso no monopoliza el procesador

Planificadores:

- Es la clave de la multiprogramación.
- Esta diseñado de manera apropiada para cumplir las metas de:
    - Menor Tiempo de Respuesta
    - Mayor rendimiento
    - Uso eficiente del procesador

Tipos

**Short Term Scheduler:** determina que proceso pasar a ejecutarse. Es el que selecciona, de entre los procesos en estado ready, aquel que va a pasar a ser ejecutado. Basicamente elige entre los procesos que estan listos en memoria para darles CPU

**Long Term Scheduler:** admite nuevos procesos a memoria (controla el grado de multiprogramación). Se encarga del paso de estado nuevo a listo

**Medium Term Scheduler:** realiza el swapping (intercambio) entre el disco y la memoria cuando el SO lo determina (puede disminuir el grado de multiprogramación).

**Dispacher:** hace cambios de contexto, cambio del modo de ejecución, despacha el proceso elegido en Short Term → salta a la instrucción a ejecutar.

Descarga los registros del procesador y otros datos relevantes a las estructuras de datos que se utilizan para controlar el proceso (PCB) saliente, y la carga en el procesador y otros puntos de los mismos datos del proceso entrante.

**Información mínima que el SO debe tener sobre un proceso**

- Es una estructura de datos asociada al proceso.
- Existe una PCB por proceso.
- Contiene información del proceso
- Tiene referencias a memoria.
- Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina.
- El PCB lo podemos pensar como un gran registro en el que se guardan los atributos anteriormente mencionados, también guarda punteros y direcciones de memoría relacionadas al proceso.
    - Información asociada a cada proceso:
        - PID, PPID, etc.
        - Valores de los registros de la CPU (PC, AC, etc).
        - Planificación (estado, prioridad y tiempo consumido del proceso, etc).
        - Ubicación (donde está el proceso) en memoria.
        - Accounting (cantidades, cuanta memoria ocupó, cuanta entrada salida ocupó).

FCFS

- Cuando hay que elegir un proceso para ejecutar, se selecciona el más viejo
- No favorece a ningún tipo de procesos, pero en principio podíamos decir que los CPU Bound terminan al comenzar su primer ráfaga, mientras que los I/O Bound no

SJF

- Política nonpreemptive que selecciona el proceso con la ráfaga más corto
- Calculo basado en la ejecución previa
- Procesos cortos se colocan delante de procesos largos
- Los procesos largos pueden sufrir starvation (inanición)

Round Robin

- Politica basada en un reloj
- Quantum (Q): medida que determina cuanto tiempo podrá usar el procesador cada proceso. Medida de tiempo pasada, por la cual el scheduler deberá elegir el próximo proceso a tomar la CPU:
    - Pequeño: overhead de context switch. Promedios más grandes.
    - Grande: se asimila a FCFS. Promedios menores.
      Al ser el Quantum tan largo, va a llegar un punto donde los procesos saldrán por su propia cuenta (finalizaron o por E/S), antes de ser expulsado. De esta forma se eliminaría por completo la rotación de procesos, ya que cada proceso encolado acabaría por sí mismo sin regresar a la cola de ejecución a esperar su siguiente turno (porque no lo necesitaría, salvo que se abandone el procesador por E/S).

- Cuando un proceso es expulsado de la CPU es colocado al final de la Ready Queue y se selecciona otro (FIFO circular)

Prioridades

- Cada proceso tiene un valor que representa su prioridad → menor valor, mayor prioridad
- Se selecciona el proceso de mayor prioridad de los que se encuentran en la Ready Queue
- Existe una Ready Queue por cada nivel de prioridad
- Procesos de baja prioridad pueden sufrir starvation (inanición)
- Solución: permitir a un proceso cambiar su prioridad durante su ciclo de vida → Aging o Penalty
- Puede ser un algoritmo preemptive o no

**Starvation**: imposibilidad de ejecución de un proceso. Es provocada por SJF y SRTF

La **inanición** es un problema relacionado con los sistemas multitarea, donde a un proceso o un hilo de ejecución se le deniega siempre el acceso a un recurso compartido (en este caso la CPU). Sin este recurso, la tarea a ejecutar no puede ser nunca finalizada.

Cualquier algoritmo que involucre algún tipo de diferenciación entre procesos y asigne recursos con base en ese criterio es susceptible de generar inanición, traducido a tipos de algoritmos, implica que los tipos SJF, SRTF y por prioridades, todos pueden generar inanición; los primeros dos en aquellos procesos que sean más largos que, y el tercero en aquellos procesos de menor prioridad

La solución es permitir a un proceso cambiar su prioridad durante su ciclo de vida → Aging o Penalty

**FCFS** → No favorece a ningún tipo de procesos, pero en principio podíamos decir que los CPU Bound terminan al comenzar su primer ráfaga, mientras que los I/O Bound no.

**SJF** → I/O Bound porque los CPU Bound al ser (casi siempre) los mas largos, podrian sufrir inanición.

**Round Robin** → Mas adecuado para I/O Bound.

**Prioridades** → No favorece a ningún tipo de procesos.

**Variantes del RR**

Timer Variable : El "contador" se inicializa en Q (contador := Q) cada vez que un proceso es asignado a la CPU

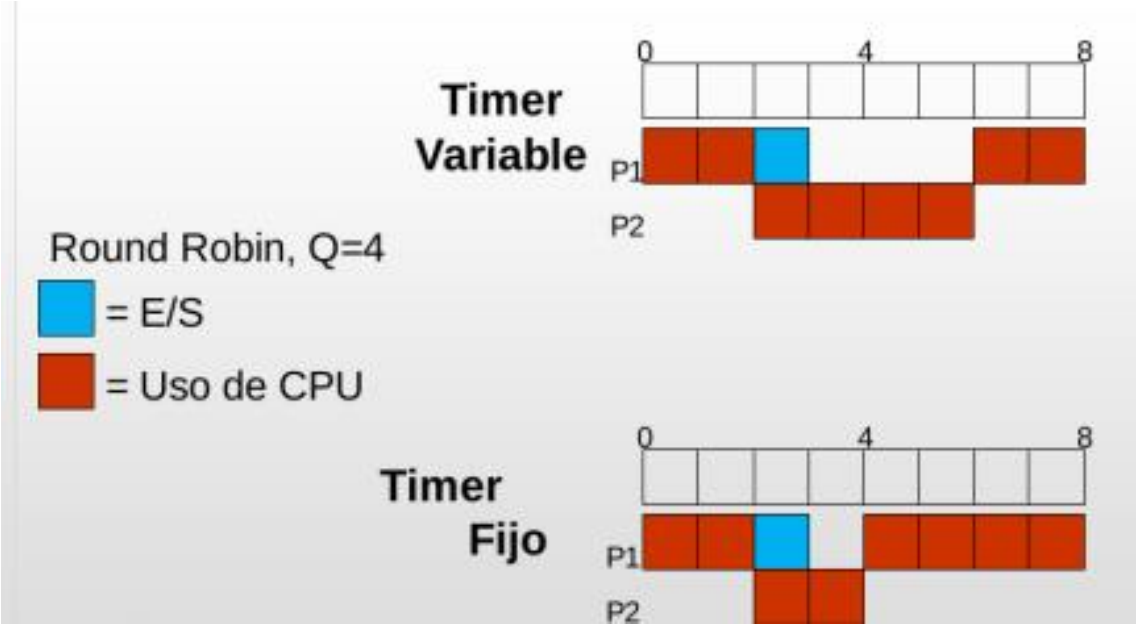
- Es el más utilizado
- Utilizado por el simulador

Timer Fijo : El "contador" se inicializa en Q cuando su valor es cero

- if (contador == 0) contador = Q;
- Se puede ver como un valor de Q compartido entre los procesos

En el caso de Timer Variable, cada proceso debe almacenar la información de su Quantum en su PCB, mientras que en el caso de Timer Fijo, se debe almacenar en una estructura o espacio accesible directamente por el SO

Los algoritmos no apropiativos tienen promedios más pequeños, pero hay demasiada diferencia en los tiempos de cada proceso.




Utilizaria Quantum alto para los procesos que requieren de bastante tiempo de CPU para que puedan terminar su tarea de manera rápida con un tiempo de retorno bajo.

La desventaja es que si hay procesos con pocas unidades de CPU, estarían esperando varios ciclos para usar la CPU por poco tiempo y darse por terminados.

**(a) Round Robin**

Beneficia a los procesos ligados a CPU. Generalmente, un proceso limitado por la E/S tiene ráfagas de procesador más cortas (cantidad de tiempo de ejecución utilizada entre operaciones de E/S) que los procesos limitados por el procesador. Si hay una mezcla de los dos tipos de procesos, suceder lo siguiente:

1. Un proceso limitado por la E/S utiliza el procesador durante un periodo corto y luego se bloquea; espera a que complete la operación de E/S y a continuación se une a la cola de listos.
2. Un proceso limitado por el procesador generalmente utiliza la rodaja de tiempo completa mientras ejecuta e inmediatamente vuelve a la cola de listos.

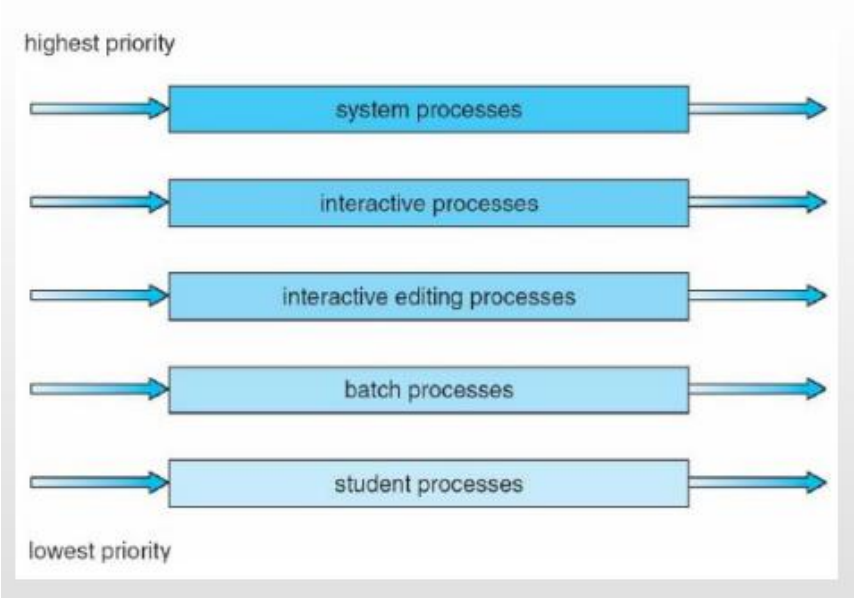
De esta forma, los procesos limitados por el procesador tienden a recibir una rodaja no equitativa de tiempo de procesador, lo que conlleva un mal rendimiento de los procesos limitados por la E/S, uso ineficiente de los recursos de E/S y un incremento en la varianza del tiempo de respuesta

**(b) SRTF (Shortest Remaining Time First)**

Favorece a los procesos ligados a E/S, ya que estos por principio hacen menos uso de CPU, con lo que desde el momento en que llegan a la cola de listos tienen más posibilidades de acceder al procesador. Por otro lado, al requerir ráfagas cortas de CPU, cuentan con más chances de hacer uso de la totalidad de su ráfaga y esperar al nuevo proceso de E/S (o terminarse) antes de ser expulsados.

**Colas Multinivel**

- Schedulers actuales → combinación de algoritmos vistos
- La ready queue es dividida en varias colas (similar a prioridades)
- Los procesos se colocan en las colas según una clasificación que realice el sistema operativo
- Cada cola posee su propio algoritmo de planificación → planificador horizontal
- A su vez existe un algoritmo que planifica las colas → planificador vertical
- Retroalimentación → un proceso puede cambiar de una cola a la otra




**Retroalimentación**

- Un proceso se puede mover de una cola a otra dependiendo de su comportamiento en tiempo de ejecución.
- Se separan los procesos en grupos pero dependiendo de las características de su ráfaga de CPU.
- Los procesos con ráfagas cortas irán a una cola más prioritaria de procesos preparados que los procesos con ráfagas largas.
- Funcionamiento:
    - Ejecutar los procesos de la cola de prioridad más alta, a continuación se pasan a ejecutar los procesos de la siguiente cola y así sucesivamente.
    - Con esta distribución, los procesos con ráfagas cortas se ejecutarán de forma rápida sin necesidad de llegar muy lejos en la jerarquía de colas de listos.
    - Mientras que los procesos con ráfagas largas irán degradándose gradualmente.

- Para gestionar a los procesos de la forma más justa, es necesario conocer:

- su longitud, si están limitados por entrada/salida o por el procesador, la memoria que van a necesitar, etc.

Forma óptima de atenderlos es:

- Preferencia para los trabajos más cortos y penalizar a los que se han estado ejecutando durante más tiempo.
- Favorecer a los trabajos limitados por entrada/salida, para mantener los recursos ocupados y dejar el procesador libre el mayor tiempo posible.

A un proceso se le concede un tiempo T de permanencia en una cola, cuando lo supera, pasará a la cola inmediatamente inferior con menor prioridad

Se pueden usar mecanismos de envejecimiento para evitar el bloqueo indefinido de un proceso

Mayor prioridad → menos Quantum. Beneficia a E/S ya que los CPU Bound pueden llegar a sufrir inanición si hay muchos de E/S → Tratarlo con aging

**(NO SE PORQUE EN LA DIAPO DICE QUE FAVORECE A CPU BOUND??????????)**

**Diapositiva:**

El sistema consta de tres colas:

- Q0: se planifica con RR, q=8
- Q1: se planifica con RR, q=16
- Q2: se planifica con FCFS

Para la planificación se utilizan los siguientes criterios:

- Los procesos ingresan en la Q0. Si no se utilizan los 8 cuantos, el job es movido a la cola Q1
- Para la cola Q1, el comportamiento es similar a Q0. Si un proceso no finaliza su ráfaga de 16 instantes, es movido a la cola Q2

¿A que procesos beneficia el algoritmo? → CPU Bound

¿Puede ocurrir inanición? → Si, con los procesos ligados a E/S si siempre llegan procesos ligados a CPU

**Actividades con más prioridad**

Las actividades que conllevan poco uso de la CPU, las que son inherentes al procesador y requieren una interacción. Priorizar estas tareas permite que aprovechen al máximo el poco uso de CPU que puedan necesitar y poder minimizar el tiempo de espera.

Si quiero escribir este trabajo en Word, no quiero que la tecla que presiono me aparezca 10 segundos después. Los que son CPU Bound, son más invisibles al usuario.

a) Cortos acotados por CPU ✔✔
b) Cortos acotados por E/S ✔
c) Largos acotados por CPU ✔
d) Largos acotados por E/S ✔

Benefician las siguientes estrategias de administración:

a) Prioridad determinada estáticamente con el método del más corto primero (SJF).
b) Prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última operación de E/S.

Los sistemas multiprocesador pueden clasificarse en:

**Homogéneos**: Los procesadores son iguales. Ningún procesador tiene ventaja física sobre el resto.

**Heterogéneos**: Cada procesador tiene su propia cola, su propio clock y algoritmo de planificación.

Otra clasificación posible puede ser:

**Multiprocesador débilmente acoplados**: Cada procesador tiene su propia memoria principal y canales.

**Procesadores especializados:** Existe uno o más procesadores principales de propósito general y varios especializados controlados por el primero (ejemplo procesadores de E/S, procesadores Java, procesadores Criptográficos, etc.).

**Multiprocesador fuertemente acoplado:** Consta de un conjunto de procesadores que comparten una memoria principal y se encuentran bajo el control de un Sistema Operativo.

**¿Con cuál/es de estas clasificaciones asocia a las PCs de escritorio habituales?**

Homogéneos, fuertemente acoplados

En el multiprocesamiento simétrico, los procesadores comparten la misma memoria.

| | MULTIPROCESAMIENTO SIMÉTRICO | MULTIPROCESAMIENTO ASIMÉTRICO |
|---|---|---|
| Trabajo | Cada procesador ejecuta las tareas en el sistema operativo. | Solo el procesador maestro ejecuta las tareas del sistema operativo. |
| Proceso | El procesador toma los procesos de una cola preparada común, o puede haber una cola preparada privada para cada procesador. | Procesador maestro asigna procesos a los procesadores esclavos, o tienen algunos procesos predefinidos. |
| Arquitectura | Todo el procesador en multiprocesamiento simétrico tiene la misma arquitectura. | Todos los procesadores en multiprocesamiento asimétrico pueden tener la misma o diferente arquitectura. |
| Comunicación | Todos los procesadores se comunican con otro procesador mediante una memoria compartida. | Los procesadores no necesitan comunicarse ya que están controlados por el procesador maestro. |
| Al fallar | Si un procesador falla, la capacidad de computación del sistema se reduce. | Si falla un procesador maestro, se envía un esclavo al procesador maestro para continuar la ejecución. Si un procesador esclavo falla, su tarea cambia a otros procesadores. |
| Dificultad | El multiprocesador simétrico es complejo ya que todos los procesadores deben sincronizarse para mantener el equilibrio de carga. | El multiprocesador asimétrico es simple ya que el procesador maestro accede a la estructura de datos. |

**¿Qué significa que se trabaje bajo un esquema Maestro/esclavo?**

- El núcleo del sistema operativo siempre ejecuta en un determinado procesador.
- El resto de los procesadores sólo podrían ejecutar programas de usuario y, a lo mejor, utilidades del sistema operativo.
- El maestro es responsable de la planificación de procesos e hilos.
    - Una vez que un proceso/hilo esta activado, si el esclavo necesita servicios debe enviar una petición al maestro y esperar a que se realice el servicio.
    - La resolución de conflictos se simplifica porque un procesador tiene el control de toda la memoria y recursos de E/S.

- Las desventajas de este enfoque son las siguientes:
    - Un fallo en el maestro echa abajo todo el sistema.
    - El maestro puede convertirse en un cuello de botella desde el punto de vista del rendimiento, ya que es el único responsable de hacer toda la planificación y gestión de procesos.

Procesadores homogéneos. Método de planificación más sencillo para asignar CPUs a los procesos

**Compartición de carga**

Los procesos no se asignan a un procesador particular. Se mantiene una cola global de hilos listos, y cada procesador, cuando está ocioso, selecciona un hilo de la cola.

Tiene algunas ventajas:

- La carga se distribuye uniformemente entre los procesadores.
- No se precisa un planificador centralizado.
- La cola global puede organizarse y ser accesible usando cualquiera de los esquemas de planificación.

Tres versiones diferentes de compartición de carga:

    - FCFS: cuando llega un trabajo, cada uno de sus hilos se disponen consecutivamente al final de la cola compartida.
    - Menor número de hilos primero: la cola compartida de listos se organiza como una cola de prioridad, con la mayor prioridad para los hilos de los trabajos con el menor número de hilos no planificados. Los trabajos con igual prioridad se ordenan como FCFS
    - Menor número de hilos primero con expulsión: se le da mayor prioridad a los trabajos con el menor número de hilos no planificados. Si llega un trabajo con menor número de hilos que un trabajo en ejecución se expulsarán los hilos pertenecientes al trabajo planificado.

La compartición de carga tiene varias desventajas:

- La cola central ocupa una región de memoria a la que debe accederse de manera que se cumpla la exclusión mutua. De manera que puede convertirse en un cuello de botella si muchos procesadores buscan trabajo al mismo tiempo.
- Es poco probable que los hilos expulsados retomen su ejecución en el mismo procesador. Si cada procesador está equipado con una cache local, ésta se volverá menos eficaz.
- Si todos los hilos se tratan como un conjunto común de hilos, es poco probable que todos los hilos de un programa ganen acceso a procesadores a la vez.

Planificación con múltiples procesadores: **Criterios**

- Planificación temporal → que proceso y durante cuanto
- Planificación espacial → en que procesador ejecutar:
    - Huella: estado que el proceso va dejando en la cache de un procesador
    - Afinidad: preferencia de un proceso para ejecutar en un procesador
- La asignación de procesos a un procesador puede ser:

    - Estática: existe una afinidad de un proceso a una CPU.
    - Dinámica: la carga se comparte → balanceo de carga.
- La política puede ser:
    - Tiempo compartido: se puede considerar una cola global o una cola local a cada procesador
    - Espacio compartido:
        - Grupos (threads)
        - Particiones

# PRACTICA 5

**Dirección Lógica o Virtual**

- Es una dirección que enmascara o abstrae una dirección física
- Referencia a una localidad en memoria
- Se la debe traducir a una dirección física

**Dirección Física**

- Es la dirección real. Es con la que se accede efectivamente a memoria.
- Representa la dirección absoluta en memoria principal.

**En la técnica de Particiones Múltiples, la memoria es divida en varias particiones y los procesos son ubicados en estas, siempre que el tamaño del mismo sea menor o igual que el tamaño de la partición.**

**Al trabajar con particiones se pueden considerar 2 métodos (independientes entre si):**

**Particiones fijas**: en la técnica de particiones fijas, la memoria se divide lógicamente en sectores que pueden ser de diferente o igual tamaño que alojan un único proceso

Ventajas:

- Fácil implementación.
- Baja sobrecarga del CPU en el manejo de memoria y resolución de direcciones

Desventajas:

- Es muy probable que se genere un alto desperdicio de memoria en forma de fragmentación interna, puesto que no se puede saber con certeza el tamaño que ocuparán los procesos antes de que éstos sean ejecutados.
- La cantidad de particiones disponibles impone un límite a la cantidad de procesos que pueden encontrarse en memoria.
- La imposibilidad de prever el tamaño de un proceso también es aplicable a su crecimiento o decrecimiento al momento de ser ejecutado, el segundo caso no trae más problemas que el aumento de la fragmentación interna, pero el primero puede llevar a que la partición asignada no basta ya para almacenar al proceso, lo cual requeriría un bloque del proceso para ser desplazado a otra área de memoria en la que quepa, o su suspensión para ser llevado al área de intercambio.

**Particiones dinámicas**: Las particiones varían en tamaño y número. Alojan un proceso cada una. Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso.

Ventajas:

- La asignación de memoria es más eficiente por proceso (no hay fragmentación interna) y en general.
- Su costo continúa siendo relativamente bajo.
- El grado de multiprogramación puede crecer o decrecer de forma dinámica (aunque hasta cierto límite)

Desventajas:

- Las sucesivas cargas y descargas de procesos en y desde memoria principal pueden generar espacios de memoria libres entre procesos que sean cada uno demasiado pequeño para ser ocupado por un proceso, pero en conjunto suficientemente grandes como para alojar uno o más procesos adicionales (fragmentación externa).
    - Esto es susceptible de resolverse a través de lo que se denomina desfragmentación→ muy costosa
- Difícil conocer con anticipación el espacio que un proceso dado requerirá a lo largo de su historia de ejecución, por lo que pueden surgir problemas en los que un proceso no puede seguir creciendo por encontrarse con otro (cada proceso es asignado a un espacio continuo de memoria).

Cada proceso se coloca en alguna partición de acuerdo a algún criterio:

Primer ajuste: se toma el primero en el que quepa el proceso entrante. Generará una tendencia a ocupar siempre los primeros espacios de la memoria.

Siguiente ajuste: se mantiene un puntero en la lista de espacios disponibles, cuando se asigna espacio a un determinado proceso, el puntero queda apuntando al siguiente inmediato, y en la próxima búsqueda se comenzará desde ese punto. Evita la preferencia por una región específica de la memoria.

Mejor ajuste: se recorre la totalidad de espacios disponibles, buscando aquel que más se aproxime al tamaño del proceso entrante (siendo siempre el proceso de igual tamaño o menor que el espacio).

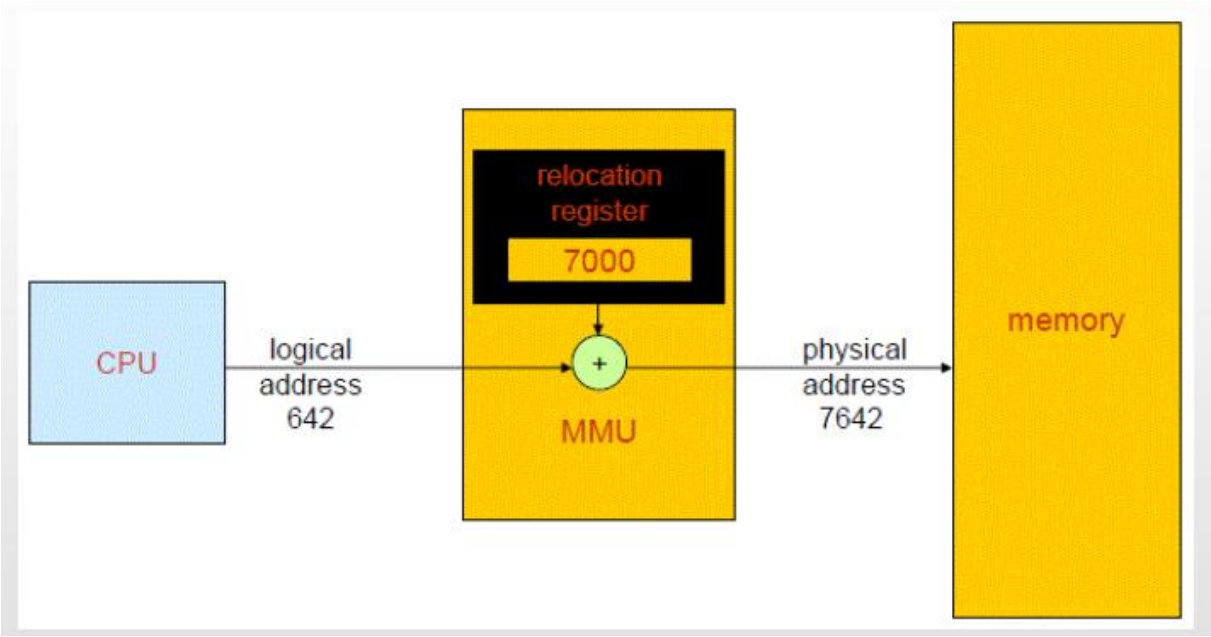
Peor ajuste: intenta resolver el problema de la fragmentación externa por exceso, procurando que las particiones libres resultantes de la asignación de espacio sean suficientemente grandes como para albergar a otros procesos. Pretende lograr esto asignando a cualquier proceso entrante la partición más grande disponible. *En la práctica, no resulta un método que incorpore ventajas significativas sobre la técnica de mejor ajuste.*

**¿Qué información debe disponer el SO para poder administrar la memoria con estos métodos?**

- Para cada partición el S.O debe conocer el registro base (donde comienza la memoria asignada al proceso) y el registro límite (donde termina la memoria asignada al proceso).
  Esto genera problemas:
    - Necesidad de almacenar el espacio de direcciones de forma continua en memoria física.

Se mantienen partes del proceso que pueden ser innecesarias en el momento

El mapeo entre direcciones virtuales y físicas se realiza mediante hardware → MMU (Memory Management Unit)




- **Particiones de igual tamaño.**
  - Fácil de implementar.
  - Pueden generar mucha fragmentación interna
  - El uso de algoritmos implica una carga más para el procesador
  - Límite de espacio máximo aunque haya espacio libre
- **Particiones de diferente tamaño.**
  - Hacen mejor uso del espacio
  - Genera fragmentación externa
  - El uso de algoritmos implica una carga más para el procesador

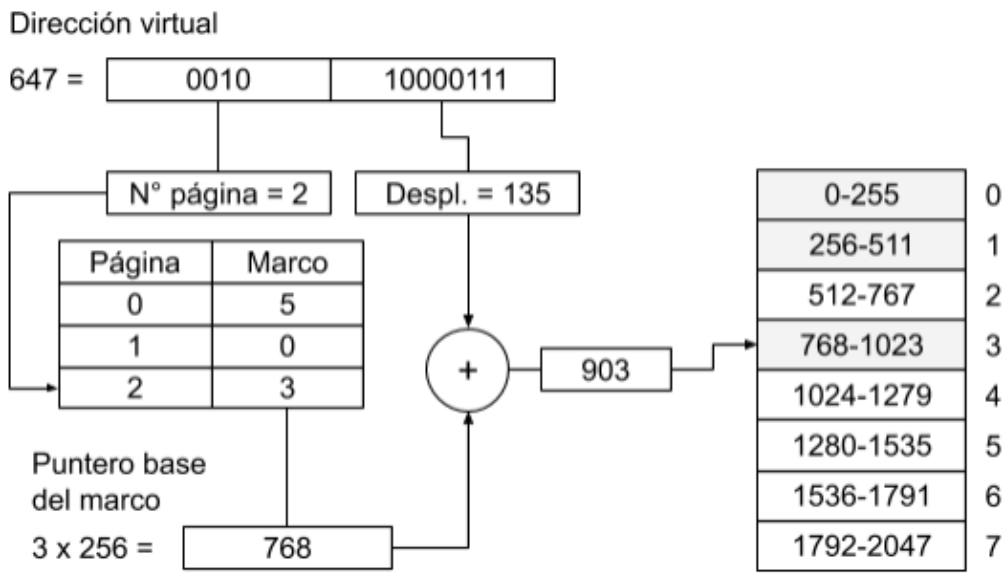
    ***a.*** Fragmentación interna:
      ***i.*** Producido en particiones fijas.
      ***ii.*** Porción de partición que queda sin utilizar.

    ***b.*** Fragmentación externa:
      ***i.*** Producido en particiones dinámicas.
      ***ii.*** Son huecos que van quedando en la memoria a medida que los procesos terminan.
      ***iii.*** Estos huecos si bien son memoria libre, al no estar contiguos entre sí un proceso no puede ser alocado.

**Paginación**

- ➔ La memoria física es dividida lógicamente en pequeños trozos de igual tamaño (marcos).
- ➔ La memoria lógica (de los procesos) es dividida en trozos de igual tamaño que los marcos (esto genera paginas).
- ➔ El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada de dicha tabla contiene el marco en la que se coloca dicha página. Se guarda la base del marco que se cargó en la PCB. Hay puntero a la tabla.
- ➔ La dirección lógica es interpretada como:
    - Un numero de página y un desplazamiento dentro de la misma.
- ➔ Como se puede observar, se rompe la continuidad.
- ➔ Puede causar fragmentación interna (menos rompe bolas que la externa).

El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada de dicha tabla contiene el marco en la que se coloca dicha página. Se guarda la base del marco que se cargó en la PCB




Se puede producir fragmentación interna, pero no tan importante como en las particiones fijas.

**Similitudes y diferencias entre la técnica de paginación y la de particiones fijas.**

Similitudes:

- Ambas pueden generar fragmentación interna.
- Ambas dividen a la memoria en fracciones de tamaño fijo.

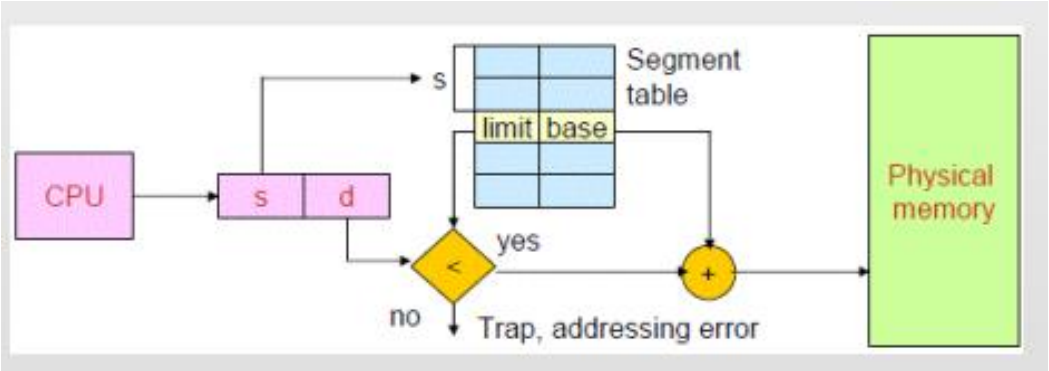
Diferencias:

- La paginación divide al proceso en varias particiones.
- La partición fija coloca todo un proceso de forma continua en una sóla partición.

**Segmentación**

- ➔ Esquema que se asemeja a la visión del usuario. El programa es dividido en partes/secciones, donde en cada sección se guardan datos similares.
- ➔ Un programa es una colección de segmentos. Un segmento es una unidad lógica como:
  - Programa Ppal, procedures, funciones, variables locales y globales, stack, etc.
  - Cada segmento tiene un registro base y un registro límite.
  - Por lo que se genera una tabla de segmentos con esos dos valores para cada segmento por proceso.
- ➔ Puede causar fragmentación EXTERNA.
- ➔ Todos los segmentos de un programa pueden NO tener el mismo tamaño (códigos, datos, rutinas). La base y límite del segmento son dinámicos.
- ➔ Las direcciones lógicas consisten en 2 partes:
  - Selector de segmento.
  - Desplazamiento dentro del segmento (sobre registro base y límite).
- ➔ La segmentación posee ventaja sobre la paginación respecto de: la protección de espacios de memoria y la compartición de bloques de memoria.

El SO debe mantener una tabla de segmentos, donde por cada segmento hay dos valores: registro base y registro límite.




Se puede producir fragmentación externa

**Similitudes y diferencias entre la técnica de segmentación y la de particiones dinámicas.**

Similitudes:

- Ambas pueden generar fragmentación externa
- Ambas se basan en la idea de distribuir espacios de tamaño variable a los procesos en forma dinámica (según la necesidad particular)

Diferencias:

- El particionamiento dinámico requiere que todo el proceso se encuentre cargado de forma continua en memoria.
- La segmentación subdivide y distribuye las divisiones de forma indiferente a su orden real, aunque sí mantiene la secuencia dentro de cada segmento

**Similitudes y diferencias entre la técnica de paginación y segmentación.**

Similitudes:

- Ambas técnicas permiten la distribución en sectores no contiguos de memoria, y tampoco requieren que dichos sectores sean asignados en el mismo orden que en el programa principal
- Se requiere de estructuras adicionales

Diferencias:

- La paginación ttransparente al programador.
- La paginación elimina fragmentación externa.
- La segmentación es visible al programador.
- La segmentación facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección.

**Memoria Virtual**

- Ejecutar procesos que requieren más memoria que la disponible en el sistema, manteniendo en memoria principal solo aquella memoria que el proceso este utilizando y el resto en disco.
- El SO se apoya en el hardware:
    - Capaz de detectar cuándo una instrucción está tratando de acceder a una dirección que no está en el momento cargada en memoria.
    - A partir de allí recién el SO podría generar las instrucciones necesarias para atender el fallo.
- El SO debe dar al hardware la información requerida para llevar a cabo su tarea
- Las tablas de páginas de un proceso deben contar con información adicional además del marco donde se encuentra la página
- Un bit que indique la presencia de la página en memoria.
    - A partir de ese dato es que el hardware puede generar la interrupción necesaria para resolver el fallo de página.
- Debe contarse con información sobre la presencia de modificaciones o no sobre las páginas cargadas.
- Otros bits de control pueden resultar útiles para definir el historial de uso del conjunto de páginas o segmentos de un proceso, o de todos los cargados → mayor grado de precisión para elegir a la víctima.

**Fallos de Página**

Se producen cuando una instrucción ejecutada hace referencia a una dirección lógica cuya página no está cargada en memoria.

**¿Quién es responsable de detectar un fallo de página?**

El hardware, generando una interrupción. Al momento de resolver la dirección, cuando recupera la entrada en la tabla de páginas correspondiente, debe analizar el bit de control V, que indica si la página se encuentra o no cargada en memoria principal

**Acciones que emprende el SO cando se produce un fallo de página.**

1. Se genera el trap.
2. El SO bloquea al proceso (la CPU toma otro proceso).
3. El SO busca un marco libre en la memoria y genera una operación de E/S que le pide al disco, para copiar en dicho marco la página deseada.
4. La E/S avisa por interrupción cuando finaliza.
5. El SO actualiza la tabla de páginas del proceso.
   - Colocando el bit V en 1, en la página correspondiente.
   - Coloca la dirección base del marco donde se colocó la página.
6. El proceso pasa del estado bloqueado a listo para ejecutar
7. Cuando vuelva a ser asignado al CPU, el proceso comenzará desde la instrucción que generó el fallo en primera instancia

La selección del tamaño de la página influye de manera directa sobre el funcionamiento de la memoria virtual.

**Tamaño de la pagina**

- ➔ Pequeño
  - Menor fragmentación interna (menos probabilidad de que queden espacios libres que sean muy grandes).
  - Más paginas requeridas por proceso → tablas de páginas más grandes.
  - Más paginas pueden residir en memoria.
- ➔ Grande
  - Mayor fragmentación interna.
  - La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente → es más rápido mover páginas hacia la memoria ram.
- ➔ A mayor tamaño de página se transfiere mejor

**Asignación de marcos a un proceso (Conjunto de trabajo o Working Set)**

**Working Set:** es el conjunto de páginas que tienen las Δ referencias a páginas más recientes

- Δ chico: no cubrirá la localidad. Toma muy pocas referencias
- Δ grande: puede tomar varias localidades.

No se requiere que todas las páginas de un proceso se encuentren en memoria. El SO debe controlar cuantas páginas de un proceso puede tener en la memoria principal. Existen 2 políticas que se pueden utilizar:

- Asignación Fija: a cada proceso se le asigna una cantidad arbitraria de marco. A su vez para el reparto se puede usar:
  - Reparto equitativo: se asigna la misma cantidad de marcos a cada proceso → m div p .
  - Reparto proporcional: se asignan marco en base a la necesidad que tiene cada proceso → Vp . m / Vt .

- Asignación Dinámica: los procesos se van cargando en forma dinámica de acuerdo a la cantidad de marcos que necesiten

**Algoritmos selección de una victima**

- Optimo: es sólo teórico, es el mejor. Selecciona como víctima a la página cuyo próxima referencia se encuentra más lejana a la actual. Es imposible de implementar ya que no se conoce los futuros eventos
- FIFO: el más sencillo, la página más vieja en la memoria es reemplazada y esta puede ser necesitada pronto, debido a esto es el peor en cuanto a la tasa de fallos de página. Trata a los frames en uso como una cola circular
- LRU: requiere soporte del hardware para mantener timestamps de acceso a las páginas. Favorece a las páginas menos recientemente accedidas. Entre aquellos más susceptibles de ser implementados, es el que más se aproxima a la idea de trabajo con el conjunto residente, por lo que podría esperarse una tasa de fallos de página relativamente baja respecto a otros métodos. Cada página debe tener información del instante de su ´ultima referencia, por lo tanto, el overhead es mayor
- Segunda Chance: un avance del FIFO tradicional que beneficia a las páginas más referenciadas, aproximándose en mayor medida a la idea de trabajo con el conjunto residente.
    - Se utiliza un bit adicional → bit de referencia
    - Cuando la página se carga en memoria, el bit R se pone a 0.
    - Cuando la página es referenciada el bit R se pone en 1
    - La víctima se busca en orden FIFO. Se selecciona la primer página cuyo bit R está en 0
    - Mientras se busca la víctima cada bit R que tiene el valor 1, se cambia a 0

Si la página a ser reemplaza puede estar modificada el sistema operativo trata esto reservando uno o varios marco para la **descarga asincrónica** de páginas.

Cuando es necesario descargar una página modificada:

1) La página que provoco el fallo se coloca en un frame designado a la descarga asincrónica.
2) El SO envía la orden de descargar asincrónicamente la página modificada mientras continua la ejecución de otro proceso.
3) El frame de descarga asincrónica pasa a ser el que contiene a la página víctima que ya se descargó correctamente.

**Alcance del reemplazo**

<u>Reemplazo Global</u>

- El fallo de página de un proceso puede reemplazar la página de cualquier proceso.
- El SO no controla la tasa de page-faults de cada proceso.
- Puede tomar frames de otro proceso aumentando la cantidad de frames asignados a él.

- Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad.

No es posible usar la asignación fija, ya que esta determina un conjunto específico de marcos que son exclusivos de cada proceso. Si se implementara un reemplazo global, un fallo de página en el proceso A, podría generar la selección de una página víctima del proceso B, con lo que se le estaría quitando un marco a B y asignándoselo a A, rompiendo por completo con la idea de asignación fija de marcos.

Reemplazo Local

- El fallo de página de un proceso solo puede reemplazar sus propias páginas (de su conjunto residente)
- No cambia la cantidad de frames asignados.
- El SO puede determinar cuál es la tasa de page-faults de cada proceso.
- Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.

**Hiperpaginación**

Es cuando un sistema pasa más tiempo paginando que ejecutando procesos

- Sucede cuando el total de marcos disponibles en el sistema es inferior a la suma de los conjuntos de trabajo de cada proceso en ejecución. En sí, la hiperpaginación sucede cuando los procesos en ejecución no tienen suficientes marcos asignados como para mantener su conjunto de trabajo.
    - Cuando una referencia a memoria genera un fallo de página, se seleccionaría una víctima que forma parte del conjunto de trabajo, la ausencia de esta víctima genera otro fallo de página, que al ser atendido selecciona a otra página del conjunto de trabajo, y así sucesivamente.
- Técnica más utilizada para detectarlo es la del control de frecuencia de fallos de página.
    - Se establecen ciertas cotas (máxima y mínima) a partir de las cuales se define el estado de un proceso y, ante la evaluación de la situación general de todos los procesos, el estado del sistema.
    - Tasa de page faults $0 < p < 1$
        - Si $p = 0$, no hay page faults
        - Si $p = 1$, cada referencia es un page fault
    - Para eliminar este problema se puede reducir la cantidad de marcos asignados a un proceso con $p \rightarrow 0$, para reasignarlos a los procesos con $p \rightarrow 1$.
    - En caso de tratarse de un reemplazo local, o de no haber procesos con p debajo de la cota mínima, se pueden seleccionar uno o más procesos para suspender.

- Effective Access Time: describe el tiempo de acceso efectivo a la memoria al utilizar paginación para la implementación de la memoria virtual:
    - Am = tiempo de acceso a la memoria real
    - Tf = tiempo de atención de un fallo de pagina

- At = tiempo de acceso a la tabla de páginas. Es igual al tiempo de acceso a la memoria (Am) si la entrada de la tabla de páginas no se encuentra en la TLB (cache donde residen las traducciones de direcciones realizadas)
- p = taza de fallo de pagina (0 <= p <=1)

$$TAE = At + ( 1 - p ) * Am + p * ( Tf + Am )$$

**Anomalía de Belady**

Es posible tener más fallos de página al aumentar el número de marcos en la memoria física utilizando el método FIFO como algoritmo de reemplazo de páginas en sistemas de gestión de memoria virtual con paginación. Antes de esta fecha, se creía que incrementar el número de marcos físicos siempre llevaría a un descenso del número de fallos de página o, en el peor de los casos, a mantenerlos.

# PRACTICA 6

## Administración E/S

**Dispositivos**

- **Orientado a bloques**: estos dispositivos transmiten datos en bloques (paquetes) y por esa razón son usados a menudo para la transmisión paralela de datos utilizando el buffer de datos del S.O.
  - Por ejemplo: Disco magnético.
- **Orientado a flujos (de caracteres)**: utilizan transmisión en secuencia de datos sin usar buffer, transmitiendo un bit o un byte a la vez.
  - Por ejemplo: Impresora, terminales, cintas de papel, interfaz de redes.

**Diferencias que existen entre los dispositivos de E/S y que el SO debe considerar.**

- Heterogeneidad de dispositivos
- Características de los dispositivos
- Velocidad
- Nuevos tipos de dispositivos
- Diferentes formas de realizar E/S

**Técnicas de E/S**

- **E/S programada**: El procesador envía un mandato de E/S, a petición de un proceso, a un módulo de E/S; a continuación, ese proceso realiza una espera activa hasta que se complete la operación antes de continuar.

- **E/S dirigida por interrupciones**: El procesador emite un mandato de E/S a petición de un proceso y continúa ejecutando las instrucciones siguientes, siendo interrumpido por el módulo de E/S cuando éste ha completado su trabajo. Las siguientes instrucciones pueden ser del mismo proceso, en el caso de que ese proceso no

necesite esperar hasta que se complete la E/S. En caso contrario, se suspende el proceso en espera de la interrupción y se realiza otro trabajo.

- **DMA (Acceso Directo a Memoria)**: Un módulo de DMA controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador manda una petición de transferencia de un bloque de datos al módulo de DMA y resulta interrumpido sólo cuando se haya transferido el bloque completo.

**La tecnica de E/S programa puede trabajar de dos formas:**

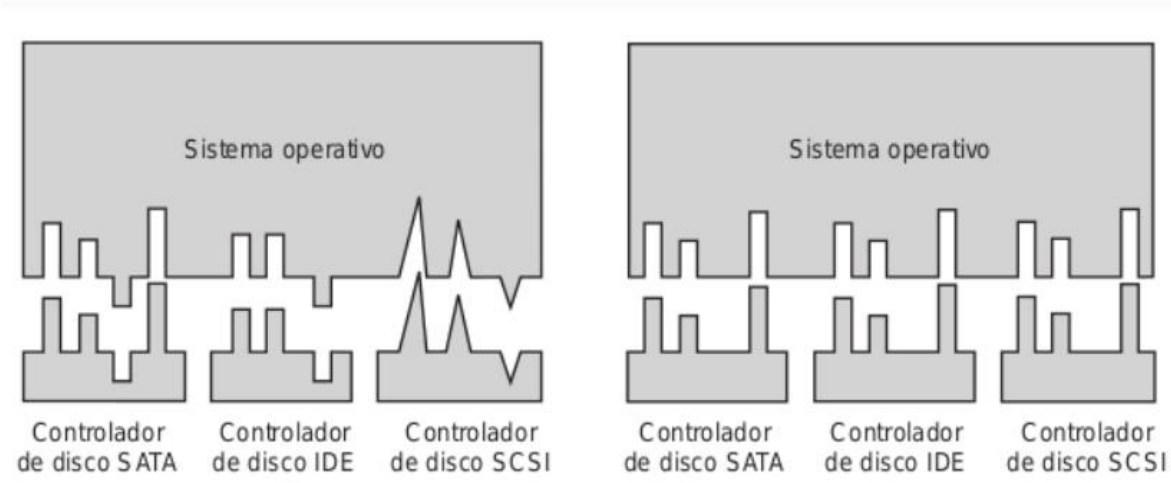
- **E/S mapeada**: Los dispositivos y memoria comparten el espacio de direcciones. I/O es como escribir/leer en la memoria por lo que no hay instrucciones especiales para I/O (ya se dispone de muchas instrucciones para la memoria)
- **E/S aislada**: Hay un espacio separado de direcciones para I/O. Se necesitan líneas de I/O. Puertos de E/S. Instrucciones especiales (conjunto limitado)

**Metas que debe perseguir un SO para la administración de la entrada salida.**

**Generalidad:**

- Es deseable manejar todos los dispositivos de I/O de una manera uniforme, estandarizada.
- Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles más “bajos” para que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock.

**Interfaz Uniforme:**




**Eficiencia:**

- Los dispositivos de I/O pueden resultar extremadamente lentos respecto a la memoria y la CPU.
- El uso de la multi-programación permite que un procesos espere por la finalización de su I/O mientras que otro proceso se ejecuta.

**Planificación:**

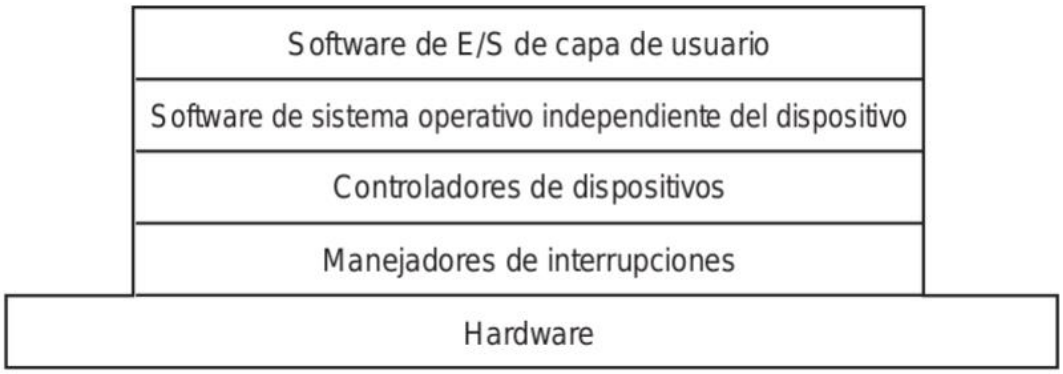
- Organización de los requerimientos a los dispositivos
    - Ej: Planificación de requerimientos a disco para minimizar tiempos

**Drivers**

Consiste en una interfaz entre el SO y el HARDWARE cargadas como módulos en el espacio de memoria del kernel conteniendo el código dependiente del dispositivo para manejar el mismo y traducir requerimientos abstractos en los comandos para el dispositivo manejado.

**Funciones minimas determinadas por el fabricante**

- init_module: Para instalarlo
- cleanup_module: Para desinstalarlo.




**Pasos mínimos que se suceden desde que un proceso genera un requerimiento de E/S hasta que el mismo llega al dispositivo**

- Determinar el dispositivo que almacena los datos
    - Traducir el nombre del archivo en la representación del dispositivo.
- Traducir requerimiento abstracto en bloques de disco (Filesystem)
- Realizar la lectura física de los datos (bloques) en la memoria
- Marcar los datos como disponibles al proceso que realizo el requerimiento
- Desbloquearlo
    - Retornar el control al proceso

**Servicios provee el SO para la administración de E/S.**

**Buffering** – Almacenamiento de los datos en memoria mientras se transfieren.

- Solucionar problemas de velocidad entre los dispositivos.
- Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos.

**Caching** – Mantener en memoria copia de los datos de reciente acceso para mejorar performance.

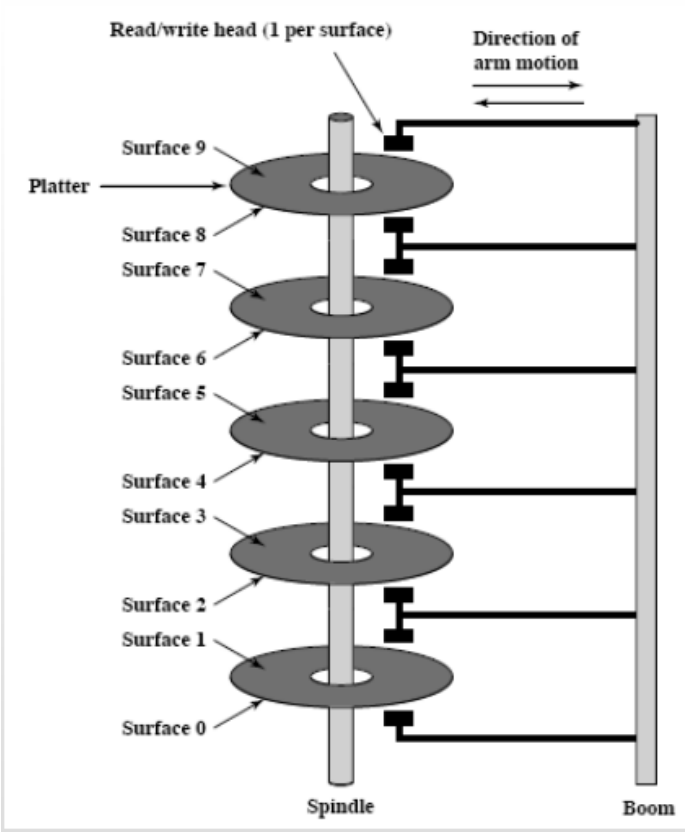
**Spooling** – Administrar la cola de requerimientos de un dispositivo.

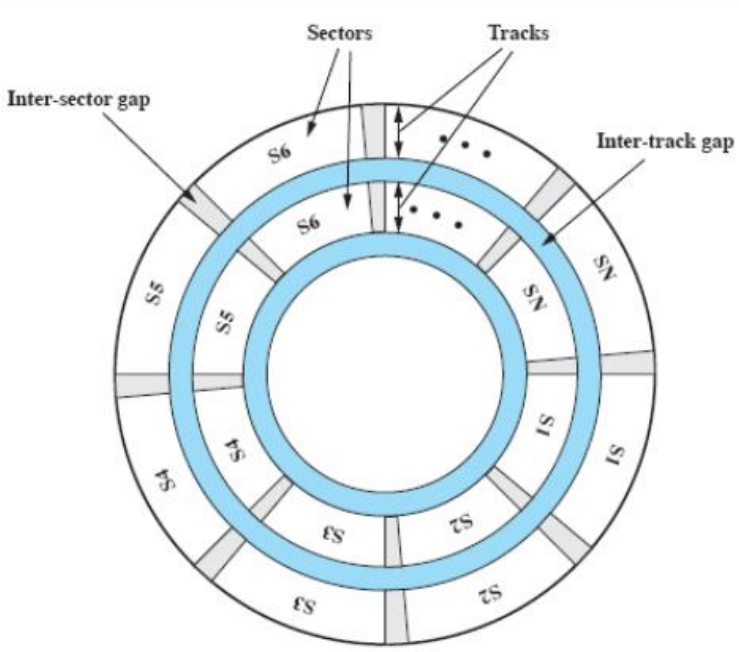
- Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo: Por ej. Impresora.

- Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo.

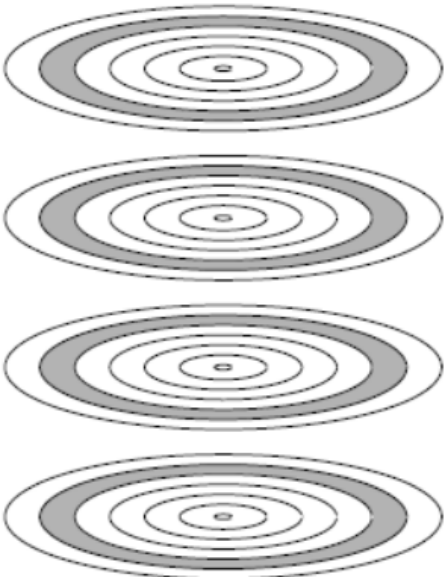
## Administración de Discos

### Organización física de un disco



- Cilindro N: todas las n-esimas pistas de todas las caras



La capacidad de un disco está dada por el producto de:

- Cantidad de caras: W
- Cantidad de pistas: X
- Cantidad de sectores por pista: Y
- Tamaño de sector: Z

capacidad = W * X * Y * Z

**La velocidad promedio para la obtención de datos de un disco está dada por la suma de los siguientes tiempos:**

- **Seek Time** (posicionamiento): tiempo que tarda en posicionarse la cabeza en el cilindro
- **Latency Time** (latencia): tiempo que sucede desde que la cabeza se posiciona en el cilindro hasta que el sector en cuestión pasa por debajo de la misma
- **Transfer Time** (transferencia): tiempo de transferencia del sector (bloque) del disco a la memoria

Para realizar una E/S, por ejemplo un acceso a disco, se requiere de una llamada al sistema (System Call). En la misma se especifica:

• Tipo de operación (E o S)

• Dirección en disco para la transferencia (file descriptor que se obtuvo al abrir un archivo)

• Dirección en memoria para la transferencia (de donde se lee o escribe)

• Número de bytes a transferir

• Este requerimiento es pasado, por el kernel, al subsistema de E/S quien lo traduce en: (#Cara, #Cilindro, #Sector)

**Tiempo de acceso a un HDD**

Latency time → si este tiempo no se conoce, se considera que es igual a lo que tarda el disco en dar media vuelta

(Tres simples)

5400 vueltas → 1' = 60" = 60000 ms

1/2 vuelta → x = 5,5 ms

**Almacenamiento secuencial:** seek + latency + (tiempo transferencia bloque * #bloques)

**Almacenamiento aleatorio:** (seek + latency + tiempo transferencia bloque) * #bloques

Unidades básicas de información (en bytes)

| Prefijos del Sistema Internacional | | | **Prefijo binario** | |
|---|---|---|---|---|
| Múltiplo - (Símbolo) | Estándar SI | Binario | **Múltiplo - (Símbolo)** | **Valor** |
| kilobyte (kB) | $10^3$ | $2^{10}$ | **kibibyte (KiB)** | **$2^{10}$** |
| megabyte (MB) | $10^6$ | $2^{20}$ | **mebibyte (MiB)** | **$2^{20}$** |
| gigabyte (GB) | $10^9$ | $2^{30}$ | **gibibyte (GiB)** | **$2^{30}$** |
| terabyte (TB) | $10^{12}$ | $2^{40}$ | **tebibyte (TiB)** | **$2^{40}$** |

Kibibyte (KiB) ≠ Kibibit (Kib)

Kibibyte (KiB) = 1024 Bytes

Kibibit (Kib) = 1024 Bits

- Supongamos un disco con 6 platos, 2 caras útiles, 1500 pistas por cara y 700 sectores por pista de 256 bytes cada uno
- Si queremos cuantas caras ocupara un archivo de 513 Mibytes almacenado de manera contigua a partir del primer sector de la primer pista de una cara determinada:
    - Calculamos la capacidad de 1 cara: 1500 * 700 * 256 bytes = 268800000 bytes
    - Dividimos el tamaño del archivo por la capacidad de una cara: 513 MiB= 537919488 bytes 537919488 / 268800000 = 2,00118 → 3 caras

- Supongamos un disco con 6 platos, 2 caras útiles, 1500 pistas por cara y 700 sectores por pista de 256 bytes cada uno. El disco gira a 12600 RPM , tiene un tiempo de posicionamiento (seek) de 2 milisegundos y una velocidad de transferencia de 15 Mib/s (Mebibits por segundo )
- Si queremos saber cuántos milisegundos se tardarían en transferir un archivo almacenado de manera contigua y aleatoria de 4500 sectores
- Calculamos los datos que faltan:
    - Latencia: 12600 vueltas → 1' = 60 s = 60000 ms
      0,5 vueltas → x = 2,3809 ms
    - Transferencia: 15 Mibits → 1 s = 1000 ms
      256 bytes → x
    - Unificamos unidades: 15728640 bits → 1000 ms
      2048 bits → x = 0,1302 ms

Datos obtenidos:

- Seek time: 2 ms
- Latency time: 2,3809 ms
- Tiempo transferencia bloque: 0,1302 ms
- #bloques: 4500 → eventualmente se tienen que calcular
- Resultados:
    - Almacenamiento secuencial: seek + latency + tiempo transferencia bloque * #bloques
      2 + 2,3809 + 0,1302 * 4500 = 590,2809 ms
    - Almacenamiento aleatorio: (seek + latency + tiempo transferencia bloque) * #bloques
      (2 + 2,3809 + 0,1302) * 4500 = 20299,95 ms

## Administración de Archivos

**Asignación Contigua**: en la asignación contigua para agregar un nuevo archivo, se utiliza un conjunto continuo de bloques para los cuáles se necesita una pre-asignación la cual consiste en conocer el tamaño del archivo durante su creación.

- Ventajas: el archivo puede ser leído con una única operación, aplicar el File Allocation Table para ésta técnica es SIMPLE ya que tiene una sola entrada que indica el bloque de inicio y la longitud del archivo.
- Desventajas: es difícil hallar bloques libres continuos en el disco, el incremento del tamaño del archivo puede generar problemas. Genera fragmentación externa.

**Asignación Enlazada**: asignación realizada en base a bloques individuales los cuales no necesitan estar contiguos. Cada bloque posee un puntero al próximo bloque del archivo.

- Ventajas: no hay fragmentación externa; se potencia el acceso secuencial; los archivos pueden crecer bajo demanda ; no se requieren bloques contiguos; es posible consolidar bloques de un mismo archivo para garantizar cercanía de bloques.
- Desventajas: no existe principio de proximidad, si es necesario traer varios bloques de fichero a la vez, como en el procesamiento secuencial, se requiere una serie de accesos a diferentes partes del disco

**Asignación Indexada**: la asignación es usando bloques individuales, para aplicar el método se usa un bloque que guarda punteros a los otros bloques que poseen los datos del archivo.

- Ventajas: los accesos random a un archivo son eficientes; no hay fragmentación externa.

**Gestión de espacio libre**

**Tabla de bits**: vector que por cada bloque de disco tiene 1 bit, bit 0 = bloque libre, bit 1 = bloque en uso.

- Ventajas: fácil de hallar un bloque o grupo de bloques libres.
- Desventajas: el tamaño de vector en la memoria.

**Lista Ligada**: se tiene un puntero al primer bloque libre, cada bloque libre tiene un puntero al siguiente bloque libre.

- Ventajas: apropiado para todos los métodos de asignación de ficheros. Si se asigna un bloque cada vez, simplemente hay que escoger el bloque libre en la cabeza de la cadena y ajustar el primer puntero o el valor de la longitud.
- Desventaja: ineficiente para búsqueda de bloques libres (se realizan varias operaciones de E/S para obtener grupo libre) ; problemas si se pierde un enlace ; es difícil hallar bloques libres consecutivos. Después de cierto uso, el disco se quedará bastante fragmentado y muchas porciones serán de la longitud de un único bloque.

**Agrupamiento**: variante de la lista ligada, donde el primer bloque libre contiene N direcciones de bloques libres, donde las N-1 primeras direcciones son bloques libres y la N-ésima dirección referencia a otro bloque con N direcciones de bloques libres.

**Recuento**: A cada bloque se le asigna un número secuencialmente y la lista de los números de todos los bloques libres se mantiene en una porción reservada del disco.

1. El espacio en disco dedicado a la lista de bloques libres es menor que el 1% del espacio total de disco.
2. Aunque la lista de bloques libres es demasiado grande para almacenarla en memoria principal, hay dos técnicas efectivas para almacenar una pequeña parte de la lista en memoria principal.
    a. La lista se puede tratar como una pila.
    b. La lista se puede tratar como una cola FIFO.

**Algoritmos de planificación en un HDD**

- Objetivo: minimizar el movimiento de la cabeza
- Como: ordenando lógicamente los requerimientos pendientes (que estan en la cola) al disco, considerando el número de cilindro de cada requerimiento. En cualquier momento se pueden encolar nuevo movimientos
- La atención de requerimientos a pistas duplicadas se resuelve según el algoritmo de planificación:
    - FCFS: se atienden de manera separada (tantas veces como se requieran). Por ejemplo, si tengo {10, 40, 70, 10}, al 10 lo atiendo 2 veces
    - SSTF/SCAN/LOOK/C-SCAN/C-LOOK: se atienden de manera consecutiva
- FCFS: atiende los requerimientos por orden de llegada
- SSTF: selecciona el requerimiento que requiere el menor movimiento del cabezal
- SCAN: barre el disco en una dirección atendiendo los requerimientos pendientes en esa ruta hasta llegar a la ´ultima pista del disco y cambia la dirección. Es importante saber en qué pista se está y de que pista se viene para determinar el sentido del cabezal
- LOOK: se comporta igual que el SCAN pero no llega hasta la ´ultima pista del disco sobre la dirección actual sino que llega hasta el ´ultimo requerimiento de la dirección actual. Es importante saber en qué pista se está y de que pista se viene para determinar el sentido del cabezal
- C-SCAN: se comporta igual que el SCAN pero restringe la atención en un solo sentido. Al llegar a la ´ultima pista del disco en el sentido actual vuelve a la pista del otro extremo (salto → no se cuentan los movimientos) y sigue barriendo en el mismo sentido

- C-LOOK: se comporta igual que el LOOK pero restringe la atención en un solo sentido. Al llegar a la ´ultima pista de los requerimientos en el sentido actual vuelve a la primer pista más lejana del otro extremo (salto → no se cuentan los movimientos) y sigue barriendo en el mismo sentido

Existen requerimientos especiales que deben atenderse con urgencia. Los fallos de página indican simplemente que tienen mayor prioridad con respecto a los requerimientos convencionales, por lo tanto deben ser atendidos inmediatamente después del requerimiento que se está atendiendo actualmente

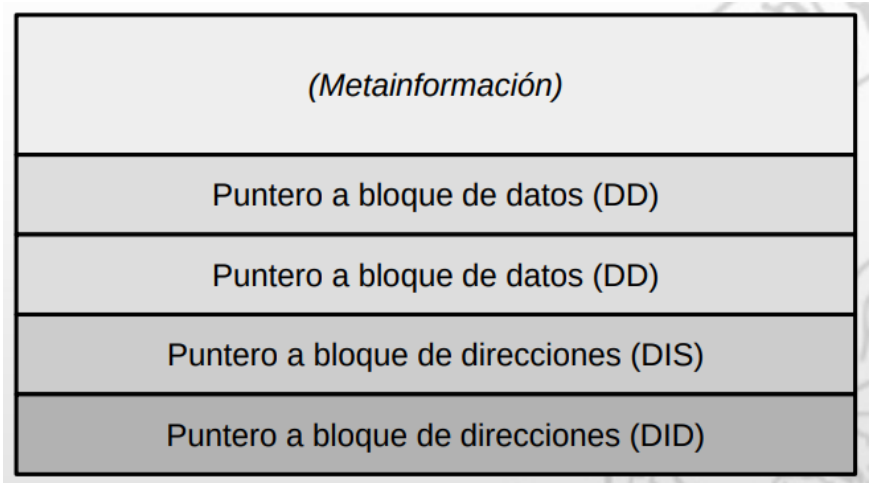
- FCFS: en orden FCFS
- SSTF: en orden SSTF
- SCAN: con el sentido que determina la atención de los últimos dos requerimientos → puede cambiar de sentido
- C-SCAN: con el sentido original → el sentido no cambia
- LOOK: del mismo modo en que lo hace el SCAN
- C-LOOK: del mismo modo en que lo hace el C-SCAN

I-NODOS

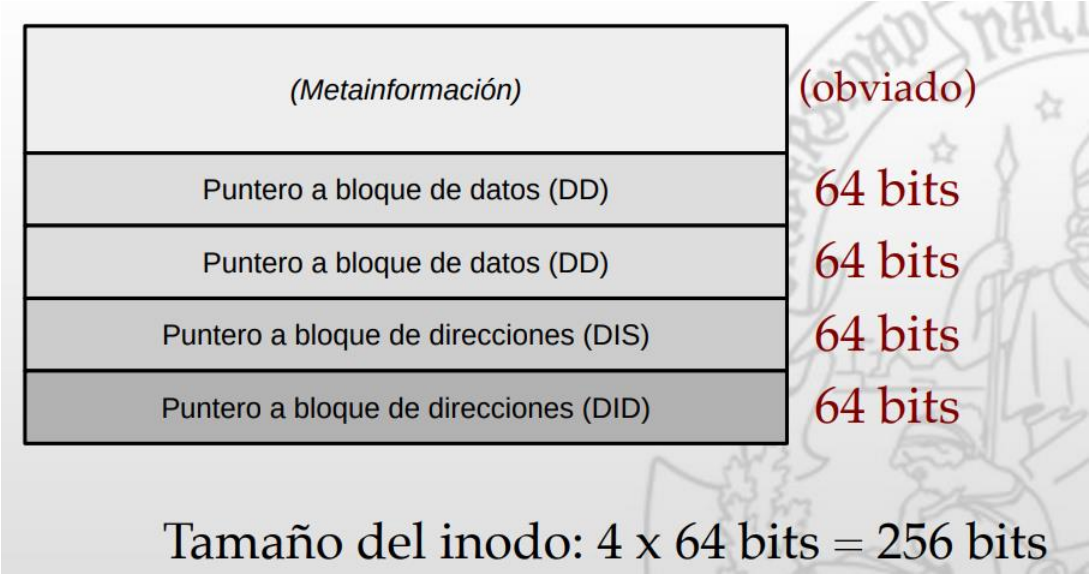
Un disco puede estar lleno y tener espacio libre a la vez. Puede quedarse sin inodos libres, por lo que no podrían crearse nuevos archivos, pero aún así tener espacio libre.

Estructura imaginaria de un inodo

- Metainformación (la obviaremos por simplicidad).
- 4 direcciones a los bloques de datos:
  – 2 de direccionamiento directo (DD).
  – 1 de direccionamiento indirecto simple (DIS).
  – 1 de direccionamiento indirecto doble (DID).




**Cada dirección es de 64 bits.**



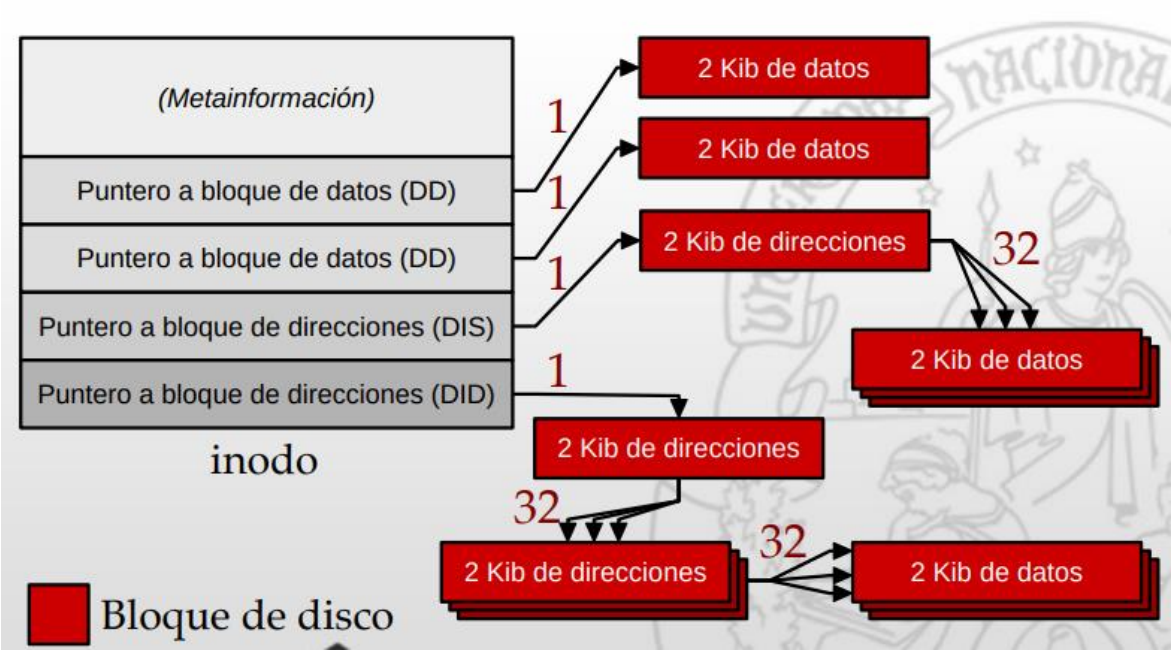

Cada bloque es de 2 Kib.

**¿Cuántos inodos puede contener un bloque de disco?**

2 Kib ÷ 256 bits = 8 inodos por bloque.

Cada dirección es de 64 bits.

2 Kib ÷ 64 bits = 32 direcciones por bloque.




**¿Cuál es el tamaño de archivo máximo que admite esta estructura?**

2 punteros a bloque de datos (DD) + 1 puntero a bloque de direcciones que apuntan a bloques de datos (DIS) + 1 puntero a bloque de direcciones que apuntan a bloques de direcciones, que apuntan a bloques de datos (DID)

2 x 2 Kib (DD) + 1 x 32 x 2 Kib (DIS) + 1 x 32 x 32 x 2 Kib (DID)