CSO Final

Age	2025

▼ Sistema Operativo

El SO es el encargado de administrar la ejecucion de procesos utilizando y gestionando los recursos del HW. El SO actua como intermediario entre el usuario y el HW a traves de una interfaz grafica.

Objetivos de un SO:

- Comodidad: Hacer facil el uso del HW.
- Eficiencia: Hacer uso eficiente de los recursos del HW.
- Evolucion: Permitir que el diseño de un SO tenga en cuenta la evolucion del HW y/o nuevos procesos sin afectar a lo que ya se tenia.

Funciones de un SO:

- Brindar abstraccion de alto nivel a los procesos del usuario.
- Administrar eficientemente el uso de los recursos del HW.
- Brindar asistencia para los procesos que requieran E/S.

Componentes de un SO: Al SO se lo puede ver como un conjunto de componentes que hacen a la solucion final.

- Kernel: Es el encargado de implementar y gestionar todo lo necesario para poder utilizar el HW.
- Shell: Es la forma de interactuar con el SO, en windows son las ventanas, en linux la consola.
- Herramientas: Conjunto de herramientas basicas como editores, compiladores, librerias, etc.

Kernel: Es una porcion de codigo que se encuentra en memoria principal, no es un proceso (ya que no tiene PCB, no sigue ninguna planificacion ni ciclo de vida y se ejecuta en modo privilegiado).

Es el encargado de la administracion de los recursos, entre ellos:

- Memoria
- CPU
- Procesos
- Comunicacion y concurrencia
- E/S

Problemas a evitar y obligaciones del SO:

- Gestionar la apropiacion de un proceso sobre la CPU.
  - Uso de la interrupcion por clock para evitar que un proceso se apropie de la CPU, usualmente se implemente con un clock y un contador.
- Detectar la ejecucion de instrucciones (por ej E/S) por parte de un proceso.
- Proteger el acceso ilegal a memoria.

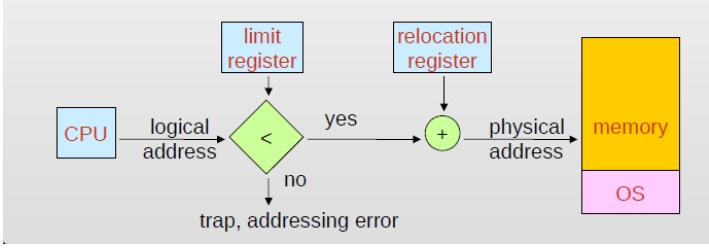
Para poder cumplir estos objetivos, es necesario el apoyo del HW, algunos de los conceptos son:

- **Modos de ejecucion**: La CPU cuenta con un bit indicando el modo en el que se encuentra, dando lugar a instrucciones privilegiadas que solo pueden ejecutarse en modo kernel. Esto significa que si la CPU esta en este modo, le permite a un proceso hacer todo lo que sea en cuanto a instrucciones y recursos del HW. Mientras que en el modo usuario, un proceso solo puede acceder a su propio espacio de direcciones.

  Cuando se inicia el sistema, se inicia en modo kernel y cada vez que se quiera ejecutar un proceso de usuario, el bit se debe poner en modo usuario. El cambio de modo se hace a traves de una interrupcion o trap, ya que estos alteran el flujo normal de instruccion.

  No es el proceso de usuario el que hace esta interrupcion, sino que el proceso ejecuta una syscall y el SO se encarga de gestionar esa solicitud y cambiar de modo.

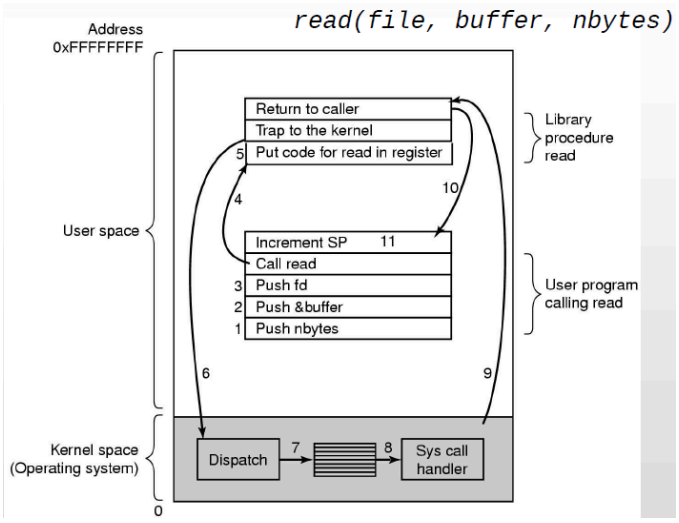
- **Proteccion de CPU:** Interrupcion por clock → Modo kernel → Vector de interrupciones → Ejecuta rutina de interrupcion → Vuelve a modo usuario.
- **Proteccion de memoria:** El SO delimita el espacio de direcciones del proceso. Al momento de ejecucion, el SO “no esta atento” a lo que accede el proceso, entonces nuevamente se necesita un apoyo del HW (mmu) para el control de memoria.



Proteccion de memoria

- **Proteccion de E/S:** Todas las instrucciones de E/S se definen como privilegiadas, lo cual solo podran ejecutarse si la CPU se encuentra en modo kernel. Estas llamadas se realizan a traves de las System Calls.

System Calls (servicios): Es la forma en que los programas de usuario acceden a los servicios del SO. Estas llamadas son realizadas por los procesos de usuarios pero son gestionadas por el kernel, permitiendo un cambio de modo. Los parametros a estas funciones pueden pasarse por registros, bloques de memoria o pilas.



Ejemplo de system call

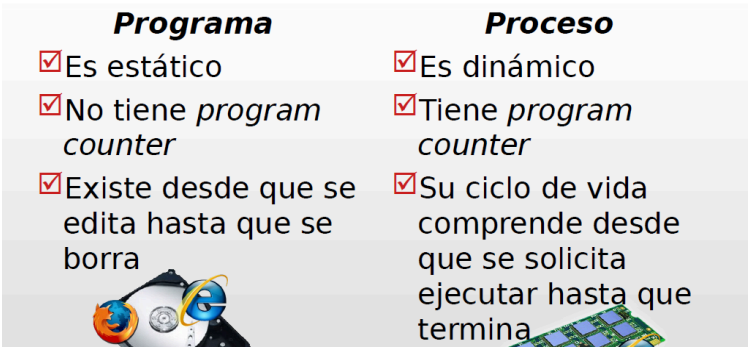
Proceso de atencion de System Call:

- Un proceso de usuario solicita un servicio del SO y realiza una syscall.
- Se genera un trap para cambiar el modo de CPU de usuario a kernel.
- Se busca en el vector de interrupciones mediante un id para poder manejar la rutina especifica para esa llamada. Esta rutina se ejecuta en modo kernel.
- Al finalizar, el SO vuelve a poner la CPU en modo usuario y retoma desde la siguiente instruccion a la syscall.

Aca no hay context switch ya que el proceso no se suspende ni se tranfiere el flujo a otro proceso, sino que el control pasa al kernel momentaneamente.

▼ Procesos

Un proceso es un programa en ejecucion, sinonimo de tarea, job. Puede pensarse como 'entidad viva', distinto a la definicion de programa. Un proceso cuenta con tres elementos esenciales, codigo de programa, conjunto de datos y su PCB.



Diferencia entre programa y proceso

Un proceso como minimo debe contener tres secciones:

- Codigo (texto)
- Datos (variables)
- Stack(s): usadas para datos temporarios como parametros, variables temporales, direcciones de retorno. En general se utilizan dos para los distintos modos, kernel y usuario, que se crean automaticamente. Esta formado por stack frames que son pushed y popped, teniendo como parametros la rutina y datos necesarios para recuperar el stack frame anterior (PC y SP).

PCB (Process Control Block): Es una estructura de datos asociada al proceso (abstraccion), solo existe una PCB asociada a cada proceso y es lo primero que se crea cuando se crea un proceso y lo ultimo que se borra.

Cada proceso tiene su propia PCB que contiene informacion especifica acerca de ese proceso.

Los procesos contienen atributos para ser identificados por el SO:

- ID del proceso y del padre.
- ID del usuario que lo "disparo".
- Si hay estructura de grupos, grupo que lo "disparo".
- En ambientes multiusuario, desde que terminal y quien lo ejecuto.

Las PCBs activas son almacenadas en una tabla de procesos en la memoria ram.

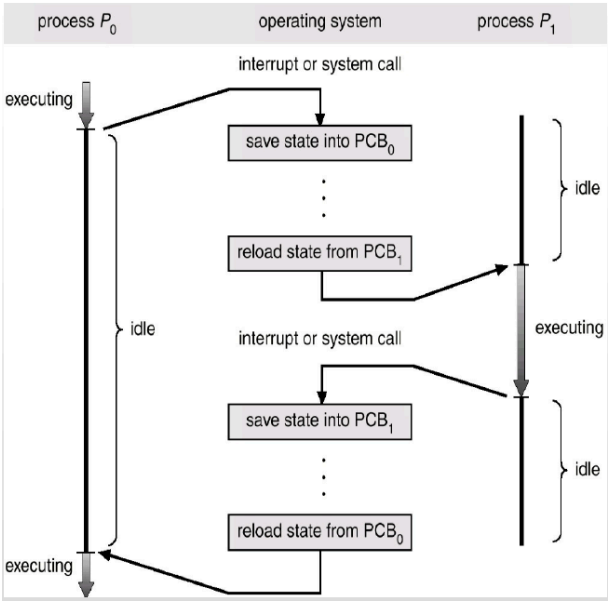
Espacio de direcciones de un proceso: Es el conjunto de direcciones de memoria que ocupa un proceso, incluyendo su stack, text y datos. No incluye la PCB del proceso. Hay que tener en cuenta que en modo kernel,

este espacio no se “respeta” ya que el proceso puede acceder a cualquier espacio de memoria.

Context Switch: Se produce a traves del dispatcher (activador) cuando la cpu cambia de un proceso a otro.

Para ello, primero el dispatcher debe resguardar la PSW del proceso saliente en la PCB del proceso, para que luego el short term pueda seleccionar un nuevo proceso y el dispatcher nuevamente cargar la PCB del nuevo proceso y comenzar desde la instruccion siguiente a la ultima ejecutada en dicho contexto. El proceso saliente pasara al estado de listo o listo/suspendido y comenzara a ejecutarse el proceso cargado.

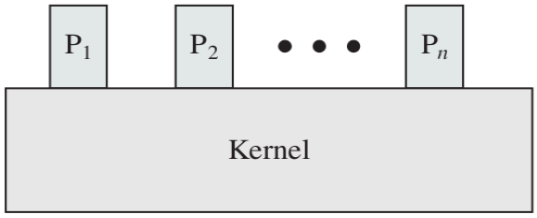
Se dice que el tiempo no es productivo para la cpu ya que no se esta maximizando el uso de la cpu para los procesos del usuario, sino para tareas de SO.



Context Switch. Se ve claramente como el tiempo de “espera” es alto para la cpu.

El kernel no es un proceso por definicion porque sino deberia estar controlado por un SO y para eso deberia existir un mega kernel y asi siguiendo. Entonces se le dan dos enfoques diferentes:

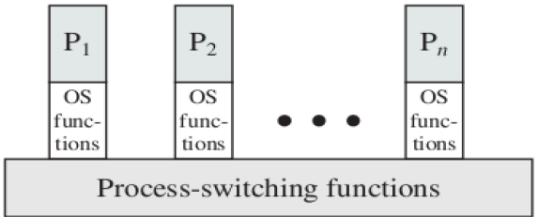
1. Kernel como **entidad independiente**: El kernel se ejecuta fuera de todo proceso, teniendo su propia region de memoria y su propio Stack. Se utiliza este enfoque para que al producirse una interrupcion o system call, el control pase al kernel y finalizada la actividad, le devuelve el control al proceso. La desventaja es justamente el context switch si el programa necesita interactuar mucho con el SO, ya que requiere un cambio de modo usuario a privilegiado constantemente a la vez que una carga y descarga del contexto.



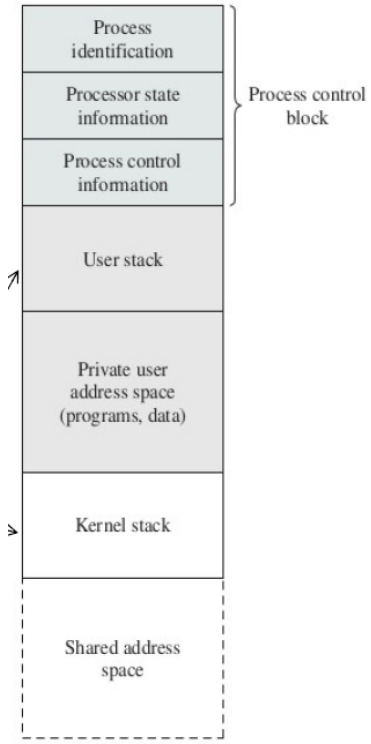
Kernel como entidad independiente

2. Kernel **“dentro” del proceso**: El codigo del kernel se encuentra dentro del espacio de direcciones de cada proceso, para poder ejecutarse en el mismo contexto que algun proceso de usuario. Puede verse como una coleccion de rutinas que el proceso utiliza. En este modelo, una interrupcion no generaria un context switch

completo, ya que las funciones del SO ya estan dentro del espacio de direcciones del proceso. Para este enfoque, necesito 2 stacks, una para el modo usuario, y otra para el modo kernel, ya que no puedo mezclar el espacio de direcciones.

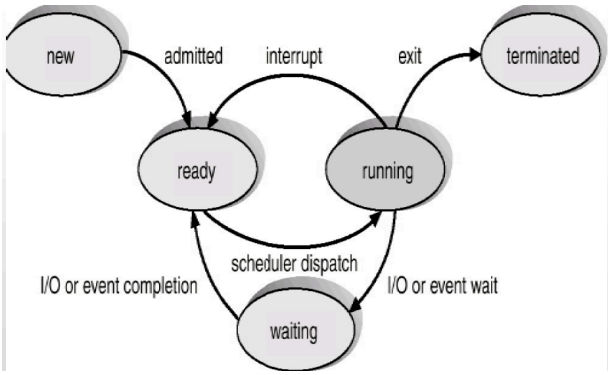


Kernel incluido en el proceso



Al finalizar, el proceso vuelve al modo usuario y continua.

Ciclo de vida de un proceso



1. New: Es el estado de inicializacion del proceso, donde se crean todas las estructuras de datos, ademas de reservar y cargar el espacio de direcciones en memoria para su posterior ejecucion. Una vez cargado, el proceso es admitido. La creacion de una proceso tiene los siguientes pasos:
    a. Asignar PID unico
    b. Asignarle memoria para regiones stack, text, datos
    c. Crear la PCB
    d. Crear estructuras de datos asociadas

    El espacio de direcciones del proceso hijo depende del sistema que estemos utilizando:

    - Caso Unix: se crea un nuevo espacio de direcciones y se copia el espacio del padre. fork() crea el proceso, execve() carga un nuevo programa en el espacio de direcciones. La funcion fork() devuelve un numero positivo si se creo, a su vez ese numero es el PID del hijo, esto me sirve para comprobar si es hijo o padre.
    - Caso Windows: se crea un nuevo espacio de direcciones vacio. CreateProcess() crea un proceso y carga el programa para ejecutar.

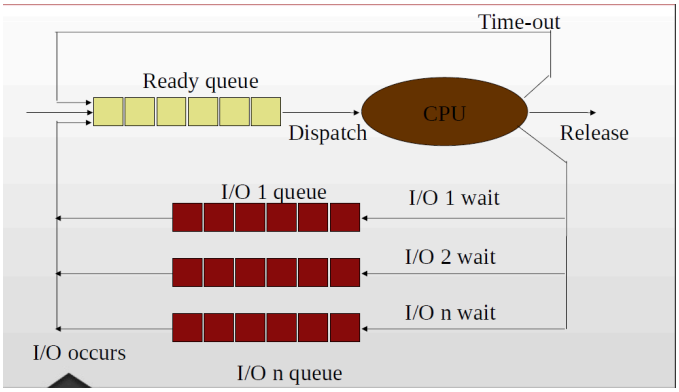
    En cualquiera de los dos casos, el espacio de direcciones logico esta restringido para cada uno pero el espacio fisico es el mismo.
2. Ready to run: El proceso tiene todo lo necesario para ejecutarse pero necesita de la cpu para ello, recien lo hara cuando el short term scheduler lo seleccione y pasara al estado de ejecucion mediante un context switch gestionado por el dispatcher.
3. Running: El proceso es ejecutado y tiene dos opciones:
    a. Terminated: Al finalizar su tarea, se liberan todas las estructuras del proceso y se libera su espacio de direcciones.
    b. Waiting: El proceso puede ser interrumpido mediante algun tipo de evento para posteriormente volver al estado de Ready to run y volver a competir para la ejecucion de la cpu.
4. Suspendido: El SO mueve parte o todo el proceso en estado de bloqueado (waiting) de memoria principal al disco, donde existe un espacio de direcciones llamado swap usado para almacenar una lista de procesos suspendidos. Es usado para liberar memoria principal y poder aceptar mas procesos en estado de ready, aumentando el grado de multiprogramacion.

Terminacion de procesos

- Exit: Retorna el control al sistema operativo, el padre puede esperar recibir un codigo de retorno (wait). Generalmente se lo usa cuando se requiere que el padre espere a los hijos.
- Kill: El padre puede terminar la ejecucion de sus hijos. Si el padre termina su ejecucion, usualmente no se permite a los hijos continuar, pero existe la opcion. Concepto de terminacion en cascada.

Colas de planificacion

Son estructuras de datos que relacionan PCBs en funcion del estado en los que se encuentran los procesos. Se pueden tener varias colas dependiendo de su uso, por ejemplo, de estado, de I/O, de prioridad, etc. Esto me permite una busqueda mas eficiente de las PCBs.

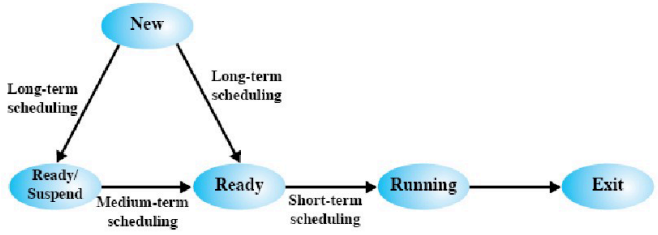


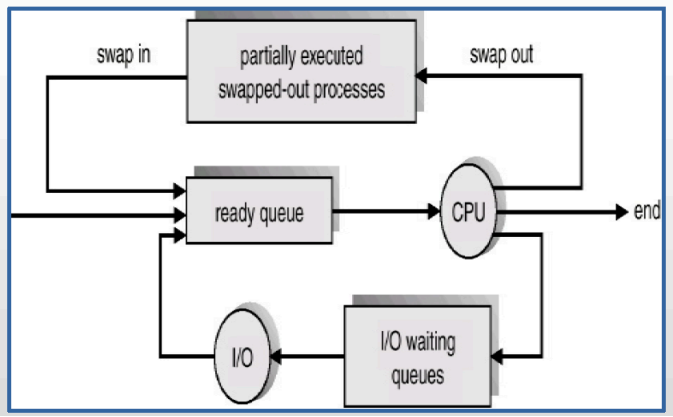
Ejemplo colas de planificacion

Modulos de planificacion

Son porciones de codigo del kernel que realizan distintas tareas asociadas a la planificacion. Son los encargados de cambiar de estado a un proceso. Existen varios planificadores que dependen de la frecuencia de ejecucion:

- **Long term**: Es el que realiza la actividad de admitir procesos al estado de Ready to run. Ligado a este planificador esta el Loader, que es el encargado de cargar el espacio de direcciones en memoria para que el proceso pueda competir por ejecucion. El long term controla el grado de multiprogramacion, osea, la cantidad de procesos en memoria.
- **Short term**: Es el que selecciona entre los procesos que estan en estado de Ready para tomar la cpu. Este planificador actuara dependiendo que tipo de ordenamiento o prioridad tiene la ejecucion de procesos. Ligado a este planificador esta el Dispatcher, que es el que realiza la tarea de context switch.
- **Medium term:** En algunas circunstancias es necesario reducir el grado de multiprogramacion. Lo que hace este planificador es sacar temporalmente de memoria los procesos que sea necesario paa mantener el equilibrio del sistema. Swap Out (sacar de memoria) y Swap In (meter en memoria).





Cola de planificacion actualizada

Planificacion

Hay una necesidad de determinar cual de todos los procesos que estan en Ready se ejecutara, para esto se utilizan distintos algoritmos de planificacion. Lo ideal seria que los procesos que son I/O bound mantengan a los dispositivos ocupados, mientras los que son CPU bound mantengan productiva a la CPU. Los algoritmos se pueden clasificar en dos tipos:

- Apropiativos: Si la politica de planificacion es apropiativa, se puede expulsar al proceso mas alla de que se este ejecutando.
- No apropiativos: Si la politica es no apropiativa, los procesos que se ejecutan lo hacen hasta dejar por cuenta propia la CPU, ya sea porque finalia su ejecucion o porque se bloquea por E/S.

Algoritmos de planificacion

Las metas de los algoritmos de planificacion son otorgar una parte justa de la CPU a cada proceso (equidad) y mantener ocupada todas las partes del sistema (balance). Existen varios mecanismos de planificacion:

- Procesos **B**atch: no requiere de un usuario esperando una espera, se pueden utilizar algoritmos no apropiativos. Ejemplos:
  - **FCFS - First come first serve**: Es la planificacion mas simple, cuando el proceso pasa a estado de listo se coloca en la cola y espera su turno.
    - Funciona mejor para procesos largos que cortos.
    - Tiende a favorecer a los CPU bound, puede ser ineficiente.
  - **SJF - Shortest job first:** Se selecciona el proceso mas corto. Puede causar inanicion en los procesos largos. Tambien es dificil estimar el tiempo que llevara la ejecucion de un proceso.
- Procesos **Interactivos**: puede ser interaccion con usuario o con servidores, son necesarios algoritmos apropiativos para evitar que un proceso acapre la CPU. Ejemplos:
  - **Round Robin**: Se basa en la utilizacion de la CPU durante un tiempo determinado. El proceso en ejecucion expulsado vuelve a competir por la cpu a la cola de listos.
    - La eleccion del quantum es un factor a ver, si es pequeño, el proceso se movera en el sistema relativamente rapido, pero si el quantum es muy largo, se pareceria mas al FCFS.
    - Tienen desventaja los procesos que son I/O bound, ya que su tiempo de cpu es mas corto. La solucion a esto es el virtual round robin, donde se crea una cola auxiliar de listos de los procesos que estuvieron bloqueados por E/S. El dispatcher tiene preferencia por ellos.
  - Prioridades

- **Colas multinivel:** Consiste en varias colas donde el proceso ira degradandose por una jerarquia de colas, donde los procesos mas largos descenderan si han pasado cierto tiempo en ejecucion, a mas tiempo en cpu, mas abajo en la jerarquia.
- **SRTF - Shortest remaining time first:** Es la version apropiativa del SJF. Se deben almacenar los tiempos restantes de los procesos expulsados.

Tiempo de retorno: tiempo desde que el proceso se crea hasta que finaliza su ejecucion.

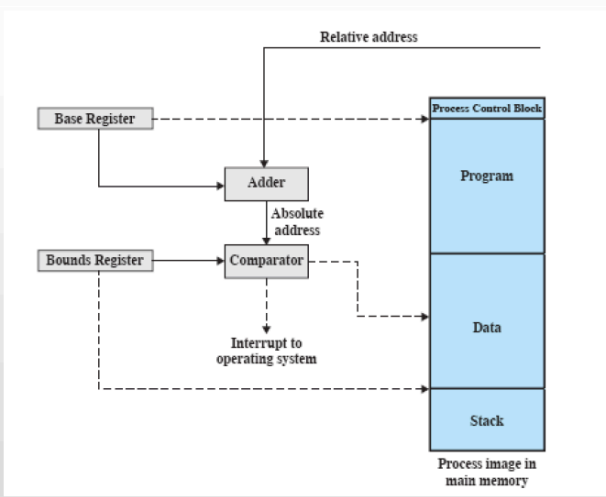
▼ Memoria

Para lograr un alto grado de multiprogramacion el uso de la memoria debe ser eficiente, por lo que el SO debe:

- Llevar un registro de las partes de memoria que se estan usando y de las que no. Ademas debe saber QUE esta ocupando ese espacio y toda la informacion que contiene.
- Asignar espacio en memoria a los procesos cuando estos la necesiten.
- Liberar espacio en memoria asignada a procesos que han terminado.
- Lograr abstraccion de la memoria al programador.
- Brindar seguridad entre procesos y controlar los accesos a memoria.
- Brindar la posibilidad de acceso compartido a memoria.

La memoria fisica contiene ya ciertos mecanismos de organizacion para que los desarrolladores la usen eficientemente para contener la mayor cantidad posible de procesos (productividad de cpu) garantizando seguridad y proteccion de la misma. Hay ciertos requisitos que el HW debe cumplir:

- Reubicacion
- Proteccion
- Comparticion



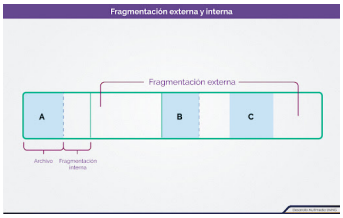
Conversion de direcciones en HW es en ejecucion, no en compilacion.

Mecanismos de asignacion en memoria

- Particiones fijas: El SO hace una division de la memoria en tamaño fijo, donde cada parte aloca un proceso o una parte de el mismo. Limita la cantidad de procesos activos.
- Particiones dinamicas: Las particiones varian en tamaño y numero, alocando tambien un proceso en cada una, y se divide dependiendo lo que el proceso necesite.

Al tener un esquema de particiones, algunos espacios de memoria quedan sin usarse debido a la alocacion o finalizacion de los procesos por no encontrarse de forma contigua, lo que producia dos tipos de **fragmentacion:**

- Fragmentacion interna: Se produce en el esquema de particiones fijas, es la porcion de particion que queda sin usar.
- Fragmentacion externa: Se produce en el esquema de particiones dinamicas, son los huecos que van quedando en memoria a medida que los procesos finalizan y que al no encontrarse de forma contigua no podamos utilizarla para otro proceso. La solucion a esto es la compactacion, pero es muy costosa en tiempo de cpu.



El problema de usar estos metodos, es que necesitariamos almacenar todas los registros base y limite de todos los procesos y que se mantengan de manera contigua. La solucion actual es la paginacion y la segmentacion.

Paginacion

El SO se encarga de mantener una tabla de paginas para cada proceso donde cada entrada contiene el marco en la que se aloca cada pagina, ademas de tener una tabla de marcos libres. Cada direccion logica esta representada por un numero de pagina mas un desplazamiento.

La direccion logica se interpreta como un numero de pagina y un desplazamiento dentro de la misma.

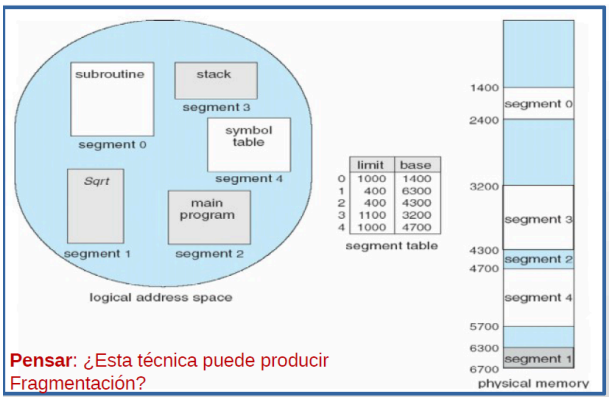
Paginación sencilla	La memoria principal se divide en marcos del mismo tamaño. Cada proceso se divide en páginas del mismo tamaño que los marcos. Un proceso se carga a través de la carga de todas sus páginas en marcos disponibles, no necesariamente contiguos.	No existe fragmentación externa.	Una pequeña cantidad de fragmentación interna.

Es invisible al programador

Segmentacion

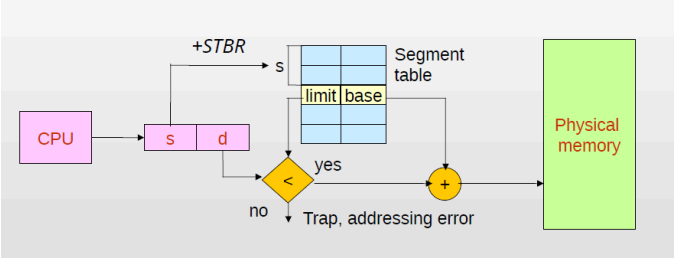
Segmentación sencilla	Cada proceso se divide en segmentos. Un proceso se carga cargando todos sus segmentos en particiones dinámicas, no necesariamente contiguas.	No existe fragmentación interna; mejora la utilización de la memoria y reduce la sobrecargada respecto al particionamiento dinámico.	Fragmentación externa.

Es visible al programador



La tabla de segmentos ademas de guardar la direccion base guarda la dir limite, ya que los segmentos son de diferentes tamaños.

El SO mantiene una tabla de segmentos, que contiene la dir logica inicial del principio de un segmento y su longitud, ademas de tener una tabla de marcos libres.



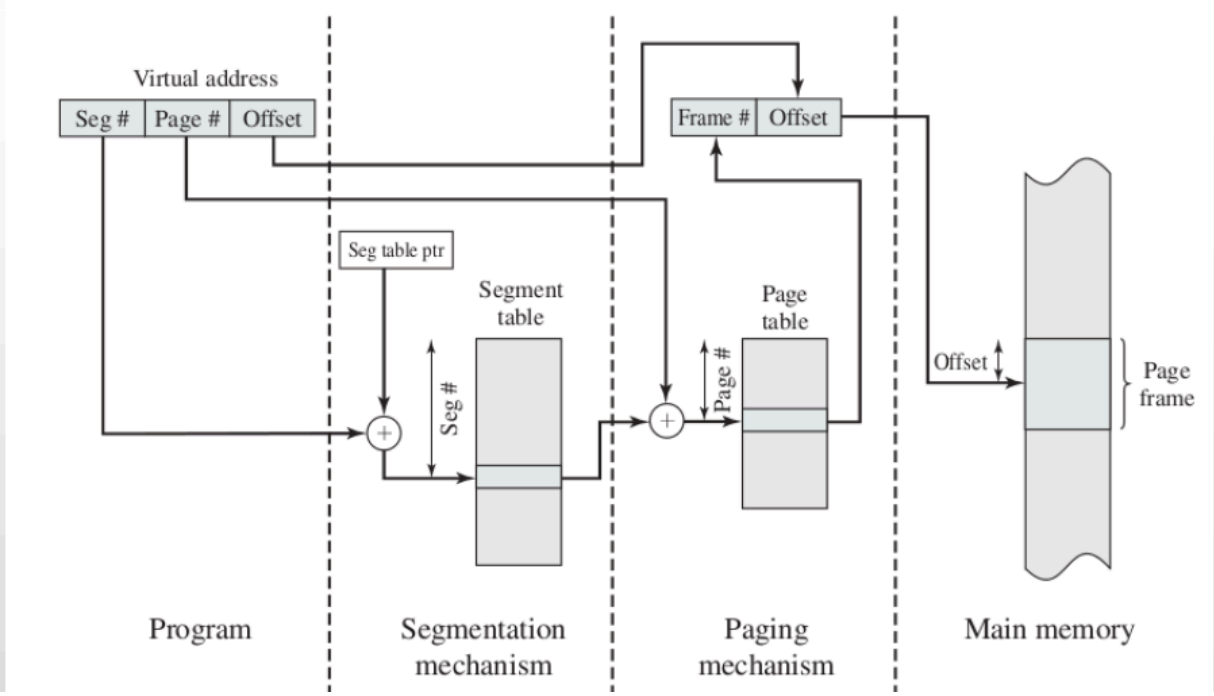
La direccion consiste en numero de segmento y desplazamiento

La ventaja de la segmentacion con respecto a la paginacion es que me permite compartir una direccion entre dos o mas procesos que utilicen la misma informacion, mientras que en paginacion deberia guardar tantas entradas de paginas como eso ocupe.

Segmentacion paginada

Esta tecnica consiste en dividir el espacio de direcciones del proceso en segmentos de tamaño variable y luego dividirlo en paginas de tamaño fijo (tamaño de los marcos) para alocarlas en memoria principal. Asi se evita la fragmentacion externa, igualmente puede haber interna debido al tamaño de la pagina. Esta tecnica presenta las ventajas de los dos mecanismos, compartir, proteger y fragmentacion.

La direccion virtual consiste en un numero de segmento, un numero de pagina y un desplazamiento. Con el numero de segmento me voy a la tabla de segmentos, donde estara alocada la direccion de la tabla de paginas correspondiente a ese segmento.



Ejemplo segmentacion paginada

Tecnica Memoria virtual - Memoria Swap

La tecnica de memoria virtual o memoria swap, se define como una memoria potencialmente mucho mas grande que la memoria real, localizada en el disco. No hay necesidad que la totalidad de la imagen del proceso sea almacenada en la memoria fisica. El SO puede llevar a memoria las partes de un proceso a medida que se necesiten.

El **conjunto residente (working set)** se define como la porcion del espacio de direcciones del proceso que se encuentra en memoria.

Con el apoyo del HW se detecta cuando se necesita una porcion de proceso que no esta en el working set y en ese caso, el SO pone al proceso en estado de bloqueado y se encarga de llevar el bloque que se necesita a memoria.

Esta tecnica tiene varias ventajas:

- Aumenta el grado de multiprogramacion, ya que mas procesos pueden estar almacenados en memoria principal.
- Un proceso puede ser mas grande que la memoria principal, esta limitacion la impone el HW.

Hay varios puntos que se necesitan para usar memoria virtual

- Soporte del HW para que soporte paginacion o segmentacion por demanda.
- Un dispositivo de memoria secundaria que de apoyo para almacenar las secciones del proceso que no estan en memoria princiapal (memoria swap).
- El SO debe ser capaz de manejar el movimiento de las paginas o segmentos entre la memoria principal y secundaria.

Memoria virtual con paginacion por demanda

Paginación con memoria virtual	Exactamente igual que la paginación sencilla, excepto que no es necesario cargar todas las páginas de un proceso. Las páginas no residentes se traen bajo demanda de forma automática.	No existe fragmentación externa; mayor grado de multiprogramación; gran espacio de direcciones virtuales.	Sobrecarga por la gestión compleja de la memoria.

Cada entrada en la tabla de paginas tiene bits de control

- Bit V indica si la pagina esta en memoria. Si esta desactivado la mmu no puede hacer la traduccion de logica a fisica. El SO es el que lo activa ya que se encarga de la memoria y el HW lo usa.
- Bit M indica si la pagina fue modificada y debe reflejar los cambios en memoria secundaria. Lo activa el HW ya que es el que controla si hubo L/E y lo usa el SO.

Memoria virtual con segmentacion por demanda

Segmentación con memoria virtual	Exactamente igual que la segmentación, excepto que no es necesario cargar todos los segmentos de un proceso. Los segmentos no residentes se traen bajo demanda de forma automática.	No existe fragmentación interna; mayor grado de multiprogramación; gran espacio de direcciones virtuales; soporte a protección y compartición.	Sobrecarga por la gestión compleja de la memoria.

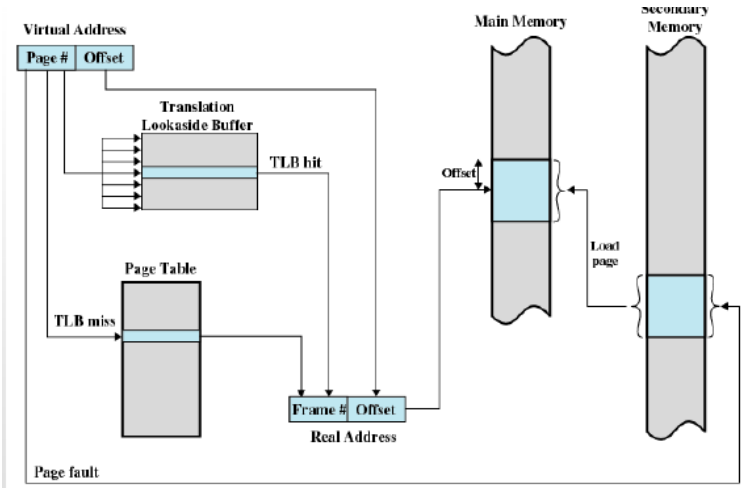
MMU

Es un dispositivo HW que mapea direcciones virtuales a fisicas. Al ser parte del procesador, reprogramar el MMU es una operacion privilegiada. Esto sirve para que los procesos trabajen siempre con direcciones virtuales y no fisicas.

Ejemplo uso de MMU

TLB

La TLB es un buffer que sirve como “cache” para poder guardar las direcciones de entradas a la tabla de paginas que han sido usadas recientemente. De esta forma, si la MMU recibe una direccion y la misma esta almacenada en la TLB, se tiene el numero de marco y directamente se busca esa direccion en memoria. Si no se encuentra, la MMU traducira la direccion desde la tabla de paginas e ira a memoria, luego actualizara la TLB.



Page fault

Ocurre cuando una direccion logica al ser traducida no se encuentra en la memoria principal, generando un trap. El SO deberia colocar al proceso en estado de waiting mientras se gestiona que la pagina que se necesite se cargue a memoria.

El SO debe buscar un frame libre (a traves de la tabla de marcos libres) y traer desde la memoria secundaria la pagina que busca almacenar mientras la CPU esta atentiendo otro proceso, ya que estamos haciendo operacion de E/S.

Cuando se carga en memoria, el SO actualiza la tabla de paginas del proceso, colocando el bit V en 1 y la direccion base donde se coloco la pagina. El proceso que genero el page fault vuelve a estado de Ready y cuando se ejecute volvera a ejecutar la instruccion que genero el page fault.

Perfomance

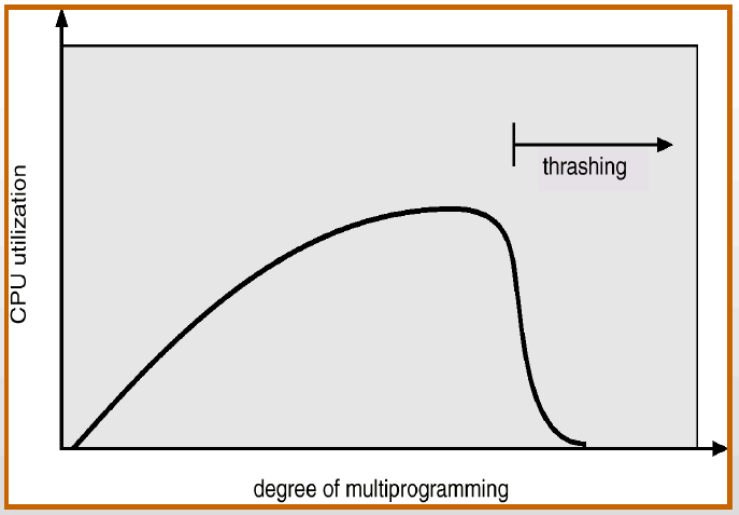
Si los PF son excesivos, la perfomance del sistema decae, la tasa de PF $0 \leq p \leq 1$, p=0 no hay PF, p=1 cada acceso a memoria genera un PF.

Trashing

Se dice que el sistema esta en trashing cuando se pasa mas tiempo trayendo y llevando porciones de procesos a memoria swap que ejecutando esos propios procesos, lo que hace una baja en el rendimiento de la perfomance en el sistema. Usualmente ocurre cuando el grado de multiprogramacion es demasiado alto.

Ciclo del trashing:

- El kernel monitorea el uso de la cpu, si detecta baja utilizacion, entonces decide aumentar el grado de multiprogramacion.
- Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos.
- Si un proceso necesita mas frames, comienzan los PF y robo de frames a otros procesos.
- Por swapping y encolamiento, baja el uso de la cpu.



Lo que causa trashing no son los procesos, sino la forma en la que el kernel administra la paginacion y la multiprogramacion.

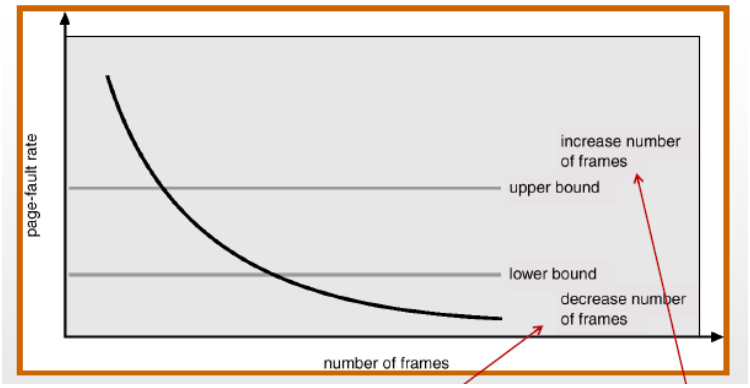
Control de trashing:

1. Bajar el grado de multiprogramacion, algo que no se busca.
2. Algoritmos de reemplazo local, si un proceso entra en trashing no roba frames a otros procesos. Perjudica la performance, pero es controlable.

Estrategia de working set / modelo de localidad

El principio de localidad se basa en que las direcciones a las que un proceso referencia suelen estar agrupadas. Con esto se busca evitar la hiperpaginacion ya que el proceso tendra mas a mano las direcciones que referenciara proximamente y evitaria los PF.

Lo ideal es que el SO le de al proceso la cantidad de frames necesarios para el tamaño de su conjunto residente.



Funcionamiento:

- El SO con apoyo del HW determina si una pagina fue accedida en un periodo reciente. Cada cierto tiempo, se verifica cuales paginas han sido referenciadas en un intervalo delta.
- Las paginas que fueron referenciadas dentro del intervalo se mantienen en el working set, mientras que las que no son removidas del conjunto.
- El SO asigna suficiente espacio en memoria para mantener el working set de cada proceso activo, si la memoria no es suficiente, se intenta bajar el grado de multiprogramacion. Si se necesita liberar memoria, se eliminan las paginas que estan fuera del working set

La desventaja de usar la estrategia de working set son:

- El pasado no siempre predice el futuro.
- Una medicion verdadera del working set no es practicable, ya que el valor optimo es desconocido y puede variar.

PFF - Page Fault Frecuency

La tecnica de PFF utiliza la frecuencia de los PF para poder calcular el tamaño del conjunto residente del proceso.

Funcionamiento:

- El SO mide la performance de cada proceso. Si el PFF es alto, es que el proceso necesita de mas frames, sino, al proceso le sobran frames.

Esta estrategia es mas sencilla de implementar que el working set, y permite ajustar dinamicamente la asignacion de paginas a cada proceso, sin embargo, puede llevar a problemas si la performance cambia bruscamente, y no garantiza que el proceso tenga siempre todas las paginas necesarias en memoria.

Tabla de Páginas de Dos Niveles

Cada proceso tiene su propia tabla de páginas, cuyo tamaño depende del espacio de direcciones del proceso. El problema de pensar en la tabla de páginas como un array por cada página que tenga un proceso es que las tablas pueden alcanzar un tamaño considerable, especialmente con espacios de direcciones grandes.

Para resolver este problema, se utilizan **tablas de páginas multinivel**, que dividen la tabla de páginas en múltiples tablas más pequeñas. La idea principal es que cada proceso tiene una **Tabla Principal** que apunta a **Tablas Secundarias**. La búsqueda de una dirección se realiza primero en la Tabla Principal y luego en la Tabla Secundaria correspondiente.

Ventajas:

1. Permite que la tabla de páginas se pueda **paginar**, es decir, almacenar partes de ella en memoria secundaria y cargar solo lo necesario en memoria principal.
2. Solo la tabla de primer nivel debe permanecer en memoria principal, lo cual ahorra espacio y permite almacenar más páginas de diferentes procesos, aumentando el grado de multiprogramación.

Desventaja:

1. La **MMU (Unidad de Gestión de Memoria)** necesita realizar múltiples accesos a memoria para traducir direcciones, lo cual puede disminuir la eficiencia si no se utiliza un mecanismo de aceleración como la TLB (Translation Lookaside Buffer).

Tabla de Páginas Invertida

En lugar de tener una tabla de páginas independiente para cada proceso, la **Tabla de Páginas Invertida** utiliza una única tabla global para todo el sistema. Cada entrada de esta tabla representa un **frame de la memoria física** y almacena información sobre qué página lógica de qué proceso se encuentra en ese frame.

Funcionamiento:

- Cada entrada contiene:
    1. **PID (Identificador del Proceso):** Identifica a qué proceso pertenece la página almacenada.
    2. **Número de Página:** Número de la página lógica que está almacenada en ese frame.
    3. **Información de Control:** Puede incluir bits de uso, de modificación, etc.
- Cuando un proceso quiere acceder a una dirección lógica, la MMU debe buscar en la Tabla Invertida la entrada que coincida con el **PID** y el **Número de Página**. Este proceso puede ser ineficiente si no se utiliza una estructura eficiente de búsqueda, como un **hash table**.

Ventajas:

1. Reduce significativamente el uso de memoria para almacenar tablas de páginas, especialmente en sistemas con un gran número de procesos.
2. Permite que la tabla de páginas sea de tamaño fijo, proporcional al tamaño de la memoria física, no al espacio de direcciones de los procesos.

Desventajas:

1. El proceso de búsqueda puede ser lento si no se utiliza un esquema eficiente como hashing.
2. No permite un intercambio sencillo de páginas entre procesos (swapping) porque la tabla es global y no independiente por proceso.

Tamaño de pagina

- Pequeño
  - Menor fragmentacion interna
  - Mas paginas requeridas por proceso → tablas de paginas mas grandes
  - Mas paginas pueden residir en memoria (como la direccion de n bits no cambia, al achicar desplazamiento tengo mas psobilidades de pagina)
- Grande
  - Mayor fragmentacion interna
  - La memoria secundaria esta diseñada para transferir grandes bloques de datos mas eficientemente → mas rapido mover paginas hacia la memoria principal → se ajusta mejor a la E/S porque los mayores costos de almacenamiento estan en latencia y busqueda, entonces en una sola llamada puedo mover mas datos.

Asignacion de marcos

La cantidad de paginas que se pueden guardar en memoria de un proceso es el tamaño del conjunto residente.

- Asignacion fija: Numero fijo de marcos para cada proceso. No es la mejor opcion, ya que la cantidad de procesos que hay en memoria van variando, por lo que se tendria que estar redistribuyendo constantemente los frames, osea, una sobrecarga del SO y tiempo no productivo.
  - Asignacion equitativa: frames disponibles/cant procesos. Tiene una desventaja importante ya que no siempre todos los procesos ocuparan la totalidad de lo asignado, pero es facil de implementar.
  - Asignacion proporcional: Se asigna acorde al tamaño del proceso.
- Asignacion dinamica: El numero de marcos varia para cada proceso. Seria una desventaja si por ejemplo se implementa una cola de prioridades, ya que los procesos de menos prioridad podrian no tener sus paginas disponibles debido a la sobrecarga de los de mayor prioridad y/o tiempo en cpu.

Hoy en dia, todo proceso inicia con una cantidad fijas de frames, pero si se detecta que con esos frames no le esta alcanzando, el SO le puede aumentar la cantidad de frames “robandole” ese espacio a otro proceso que no lo este utilizando. Esto es una medida que se hace usando la performance del sistema, que evalua la tasa de PF.

Politicas de reemplazo de paginas

Si se produce un PF y todos los marcos estan ocupados, se debe buscar un frame seleccionando una pagina victima. El reemplazo mas optimo sera que la pagina a ser removida no sea referenciada en un futuro proximo; en algunos casos esto se puede lograr mirando el comportamiento pasado.

Alcance del reemplazo

En caso de tener que seleccionar una victima, hay que decidir si se libera espacio del mismo proceso que genera el PF (reemplazo local) o si la victima puede ser cualquier otro proceso cargado en memoria (reemplazo global).

- Reemplazo global: Con esta tecnica, el SO no controla la tasa de PF de cada proceso (imposible medir la performance, ya que nose si es porque le robaron los frames o porque tiene p→1). Esto permite a un proceso de alta prioridad tomar los frames de un proceso con menor prioridad. Ademas, puede tomar frames de otro proceso y asignarselos.
- Reemplazo local: El reemplazo se da solo de su conjunto residente, por lo que no cambia la cantidad de frames asignados. El SO si puede medir la performance de cada proceso y si p→1 no le estan alcanzando los frames que le di. La desventaja es que puede tener frames sin utilizar o que no le alcance.

	Local Replacement	Global Replacement
Fixed Allocation	• Number of frames allocated to a process is fixed. • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Relacion entre asignacion y reemplazos

Algoritmos de reemplazo de paginas

- Optimo: es solo teorico, no aplicable.
- FIFO: es el mas sencillo.
- LRU (least recently used): Requiere soporte del HW para mantener timestamps con el acceso a las paginas. Favorece a las paginas menos recientemente accedidas. Tambien puede ser usado con el bit de referencia, pero no es tan exacto.
- 2da chance: Es igual al FIFO, pero utilizando el bit de referencia (pasa de 1 a 0) que indica que esa pagina se utillizo, por lo que beneficia a las paginas mas frecuentadas.
- NRU (non recently used): Utiliza los bits de referencia y modificada, favorece a las paginas que fueron usadas recientemente.

▼ I/O

El SO debe tener estructuras que permitan las operaciones con los dispositivos, por lo que tambien debe administrar errores ocurridos.

Buffering: Almacenamiento de los datos en memoria mientras se transfieren.

Caching: Mantener en memoria copia de los datos de reciente acceso para mejorar performance (lo que se accedio recientemente es probable que vuelva a ser accedido).

Spolling: Es un mecanismo para administrar la cola de requeriemientos de un dispositivo.

Formas de realizar I/O:

- Bloqueante: El proceso se suspende hasta que el requerimiento no se completa.
- No bloqueante: El requerimiento de I/O retorna en cuanto es posible.

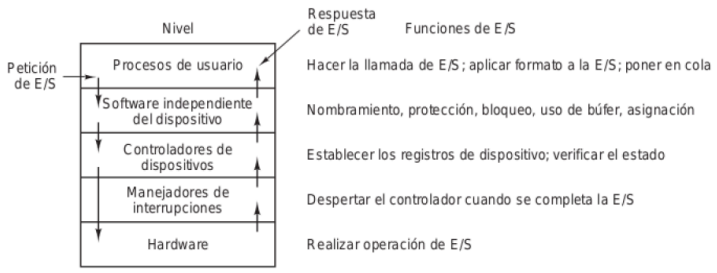
Ciclo de atencion de un Requerimiento

El ciclo de vida de un requerimiento de Input/Output (I/O) en un sistema operativo sigue varios pasos, desde la solicitud del proceso hasta la ejecuci´on final en el dispositivo correspondiente.

Aqui se presenta un resumen breve de los pasos en este ciclo:

- Un proceso en **modo usuario** emite una operación de I/O mediante una **llamada al sistema (syscall)**.

- Se genera una **interrupción por software (trap)** para cambiar a **modo kernel** y delegar la solicitud al SO.
- Se valida si la solicitud es posible (parametros, permisos, disponibilidad de recursos, etc.).
- Si los datos requeridos están disponibles en **buffer cache**, se devuelve la respuesta al proceso sin realizar I/O físico.
- Si es necesario realizar una I/O física, el proceso es colocado en la **cola de espera del dispositivo**.
- El **planificador de corto plazo** expulsa al proceso de la CPU.
- El **módulo de I/O independiente del dispositivo** gestiona la solicitud y la reenvía al **driver específico del hardware**. El **driver del dispositivo** asigna buffers en el espacio del kernel y planifica la ejecución de la operación. El driver comunica la solicitud al **controlador del dispositivo** y este inicia la operación. Si el dispositivo admite **DMA (Acceso Directo a Memoria)**, el **DMA controller** transfiere datos sin intervención de la CPU.
- Cuando el hardware completa la operación, se genera una **interrupción** para notificar al SO.
- El **manejador de interrupciones del SO** atiende la interrupción y actualiza estructuras de control (buffers, colas de espera, etc.).
- El **planificador de mediano plazo** puede decidir traer el proceso de vuelta a memoria principal.
- El **planificador de corto plazo** lo coloca en la cola de ready para competir por la CPU.
- El kernel transfiere los datos o códigos de retorno al **espacio de direcciones del proceso**.
- Cuando el **planificador de corto plazo** asigna la CPU al proceso, este retoma su ejecución desde donde quedó interrumpido.



Atencion de un requerimiento

▼ FileSystem

El filesystem es el conjunto de unidades de software que proveen los servicios necesarios para la utilizacion de archivos. Su objetivo es brindar espacio en disco a los archivos del usuario y del sistema, manteniendo un registro del espacio libre. Permite la abstraccion al programador en cuanto al acceso de bajo nivel.

Los objetivos del SO en cuanto a los archivos son:

- Cumplir con la gestion de datos y las solicitudes del usuario.
- Minimizar / eliminar la posibilidad de perder o destruir datos, garantizando la integridad del contenido de los archivos.
- Dar soporte de E/S a distintos dispositivos.
- Brindar un conjunto de interfaces de E/S para tratamiento de archivos.

Tipos de archivos

- Archivos regulares
  - Texto plano: Abro el archivo y puedo ver sus datos.

    - Binarios: No puedo ver su contenido, estan preparados para ser ejecutados.
- Directorios

Atributos de un archivo

- Nombre, Id, tipo, localizacion, tamaño.
- Proteccion, seguridad y monitereo.

Directorios

Contiene informacion acerca de archivos y subdirectorios dentro de el, siendo el mismo un archivo. El uso dedirectorios ayuda para:

- Localizacion rapida de archivos.
- Repeticion de nombres de archivos.
- Agrupacion logica de archivos en propiedades/funciones.

Los archivos pueden ubicarse siguiendo un path desde el directorio raiz y sucesivas referencias. Distintos archivos pueden tener el mismo nombre pero el fullpathname es unico. Al directorio actual se lo llama working directory, donde se pueden referenciar archivos tanto por su path absoluto (directorio completo) o relativo(el nombre se calcula relativamente al directorio donde se este), indicando solamente la ruta al archivo.

Conceptos del disco

- Sector: Unidad de almacenamiento usada en los discos rigidos.
- Bloque/Cluster: Conjunto de sectores consecutivos.
- FileSystem: Define la forma en que los datos son almacenados.
- Fat (File Allocation Table): Contiene informacion sobre en que lugar estan alocados los distintos archivos.

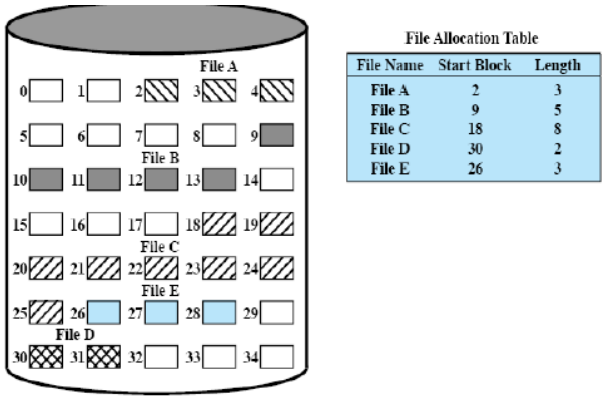
Asignacion de ficheros

Por reserva de espacio

- **Preasignacion**: Se reserva un espacio especifico para un archivo al crearlo. A menudo, se tiende a asignar espacios mas grandes de los necesarios para prever futuras expansiones del archivo. Esto puede llevar a un uso ineficiente del espacio en disco, ya que el espacio asignado no se utiliza completamente. Si un archivo supera el espacio asignado en la preasignacion, es necesario realizar una operacion de expansion para aumentar su tamaño, lo cual implica un costo adicional.
- **Asignacion dinamica**: El espacio se solicita a medida que se necesita, pero los bloques de datos pueden quedar de manera no contigua.

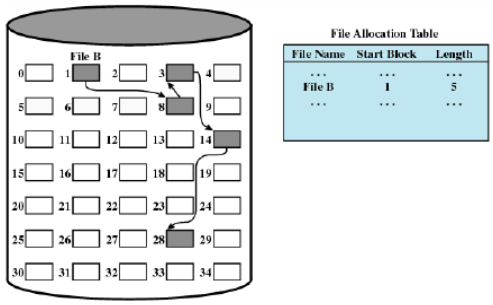
Por organizacion de bloques

- **Contigua:** Se asigna un espacio de bloques contiguo en memoria al momento de su creacion, asignandole al fichero solo el bloque del comienzo y la cantidad de bloques.
    - La ventaja es que la informacion se encuentra contigua, lo que hace facil la busqueda ya que se pueden traer un conjunto de bloques en un solo llamado.
    - La desventaja es que puede causar fragmentacion externa. Ademas, en caso de querer incrementar o agragar otro archivo hay que ver si hay bloques contiguos para asignar.



Asignacion contigua

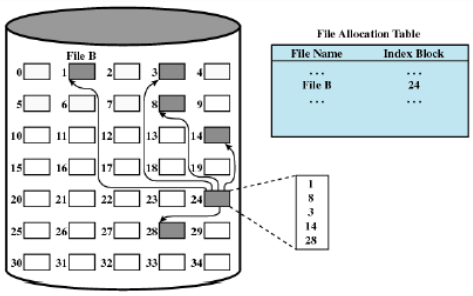
- **Encadenada**: Se asigna en base a bloques individuales, donde cada bloque tiene un puntero al proximo bloque del archivo.
  - Ventaja: No existe la fragmentacion externa. Es util para el acceso secuencial. Permite la extension del archivo ya que no requiere bloques contiguos.
  - Desventaja: Si es necesario traer bloques individuales, se requieren de varios accesos diferentes a las partes del disco.



Asignacion encadenada

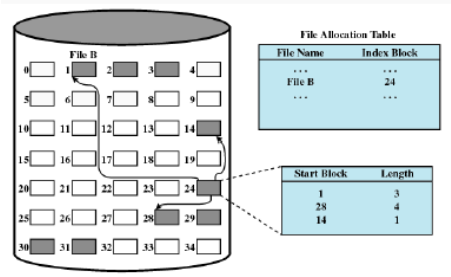
- **Indexada:** La FAT contiene un puntero al bloque indice, el cual no contiene datos propios del archivo, sino que contiene un indice a los bloques que lo componen.

  Existen dos tipos de asignacion indexada:
  - Por bloques: Elimina la fragmentacion externa. Acceso ‘Random’ eficiente.



Asignacion indexada por bloques

  - Por secciones: A cada entrada del campo indice se le agrega el campo longitud, mientras que el indice apunta al primer bloque de un conjunto almacenado de manera contigua.

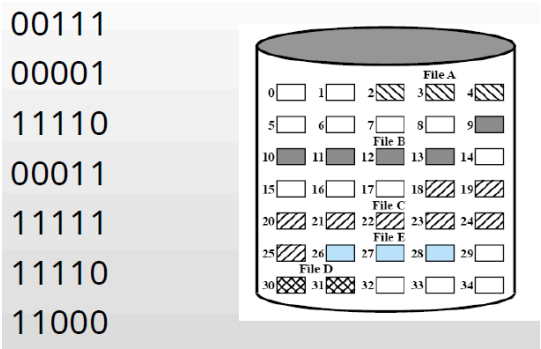


Asignacion indexada por secciones

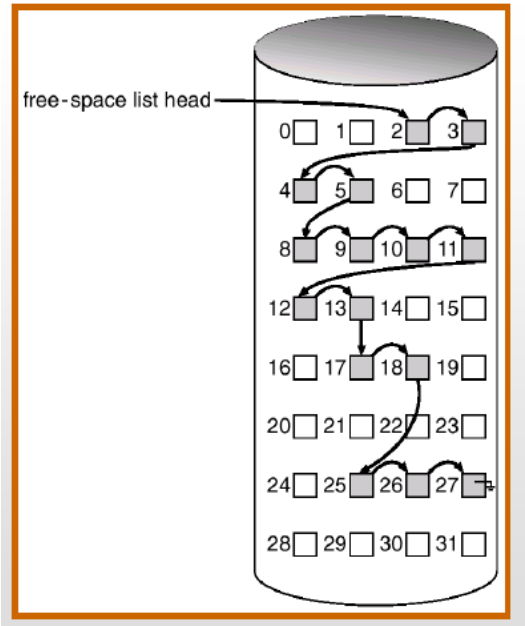
Gestion del espacio libre

Al igual que para los bloques usados, se debe tener un control sobre los bloques libres del disco. Hay varias alternativas:

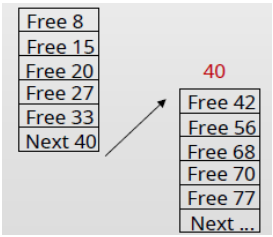
- **Tabla de bits**: Consta de un vector de bits de longitud n donde cada bit corresponde a un bloque de disco, que su valor dependera si esta libre o ocupado. Es facil de encontrar un bloque o grupo de bloques libres, pero el tamaño en memoria no es despreciable.



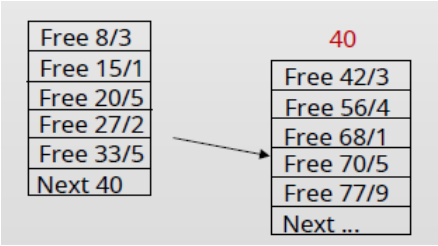
- **Bloques encadenados:** Se tiene un puntero al primer bloque libre, donde cada bloque libre tiene un puntero al siguiente bloque libre. Es ineficiente para la busqueda de bloques libres o un grupo de ellos consecutivos.



- **Indexacion:** El primer bloque libre funciona como una pagina donde contiene las direcciones de N bloques libres, la ultima posicion cuenta con la referencia a otro bloque con n direcciones de bloques libres.



- **Recuento:** Consiste en que varios bloques contiguos pueden ser solicitados o liberados a la vez, entonces en lugar de tener N direcciones libres con indice se tiene la direccion del primer bloque, y la longitud de los n bloques libres.



▼ Buffer cache

Una cache de disco es un buffer en memoria principal usado para almacenamiento temporario de sectores del disco, con esto se logra minimizar la frecuencia de acceso al disco. Cuando se hace una peticion de E/S solicitando un determinado sector esta en la cache del disco. En caso afirmativo, se sirve la peticion desde la cache. Es probable por el principio de proximidad, que haya referencias a ese mismo bloque en el futuro.

Si un proceso quiere acceder a un bloque de la cache hay dos alternativas:

- Se copia el bloque al espacio de direcciones del usuario → no permite compartir el bloque.

- Se trabaja como memoria compartida → permite acceso a varios procesos, aunque el area de memoria debe ser limitada con lo cual se usan algoritmos de reemplazo.

El algoritmo de reemplazo mas utilizado es el LRU, ya que un bloque se ubica en lo mas alto de la pila al ser referenciado. No se mueven los bloques dentro de la memoria principal, se usa una pila de punteros a la cache. Tambien se usa el LFU, que utiliza un contador que almacenara las referencias que tuvo ese bloque, aunque por el prinpicio de proximidad, puede que este contador no lleve una correcta cuenta.

Buffer cache en UNIX - System V

El kernel es el que le asigna un espacio en la memoria principal durante la inicializacion para esta estructura, por lo tanto el buffer cache es independiente del HW, ya que es un servicio gestionado por el SO.

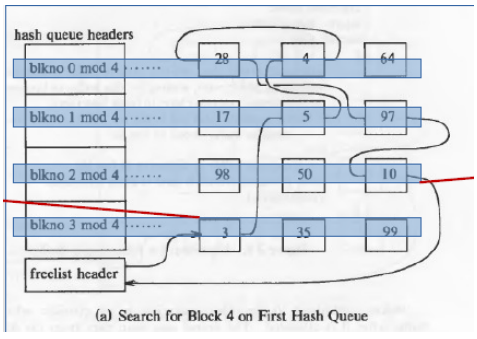
Un buffer cache consta de dos partes:

- Headers: Contienen informacion para modelar, como el nro de dispositivo y nro de bloque, estado (free, busy o Delayed Write modificados en memoria pero no en bloque original de disco), y cuenta con punteros a:
  - 2 punteros a Hash Queue: Son colas para optimizar la busqueda de un buffer en particular. Los headers se organizan segun una funcion de hash usando (dispositivo,#bloque). Al numero de bloque se le aplica hash que permite agrupar los buffers cuyo resultado dio igual para hacer que las busqueadas sean mas eficientes.
  - 2 punteros a la free list: La free list organiza los headers de los buffers disponibles para ser utilizados para cargar nuevos bloques de disco, no necesariamene estan vacios, puede haber liberado el bloque pero sigue en estado DW. Se ordenan segun LRU de menor a mayor.
  - 1 puntero al bloque en memoria
- El buffer en si: El lugar fisico en la ram referenciado por ek header donde se almacena realmente el bloque del dispositivo llevado a memoria.

Cuando un proceso quiere acceder a un archivo, se utiliza su i-nodo para localizar los bloques de datos donde se encuentra este. Luego, el requerimiento llega al buffer cache quien evalua si puede satisfacer el requerimiento o si debe realizar la E/S, donde se pueden dar 5 escenarios.

1er escenario

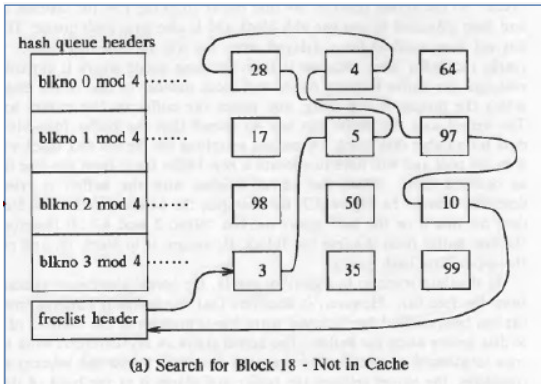
El kernel encuentra el bloque en la hash queue y esta diponible en la free list. Se remueve el bloque de la free list y el buffer pasa a estado de busy. Se usa el bloque y se reacomodan los punteros de la free list.



(a) Search for Block 4 on First Hash Queue

2do escenario

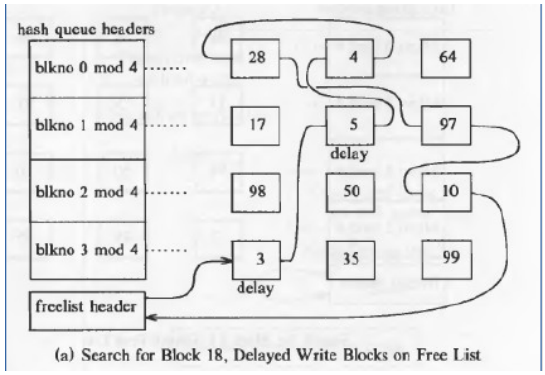
El bloque buscado no esta en la hash queue, se debe buscar un bloque libre. Se toma el primer buffer libre de la free list y se lee de disco el bloque deseado en el buffer obtenido. Se ubica el header en la hash queue correspondiente (solo punteros, no se cambian las ubicaciones de memoria)



(a) Search for Block 18 - Not in Cache

3er escenario

El kernel no encuentra el bloque buscado en la hash queue, debe tomar el primero de la free list pero se encuentra marcado DW, entonces manda a escribir a disco ese bloque y tomar el siguiente buffer de la free list.



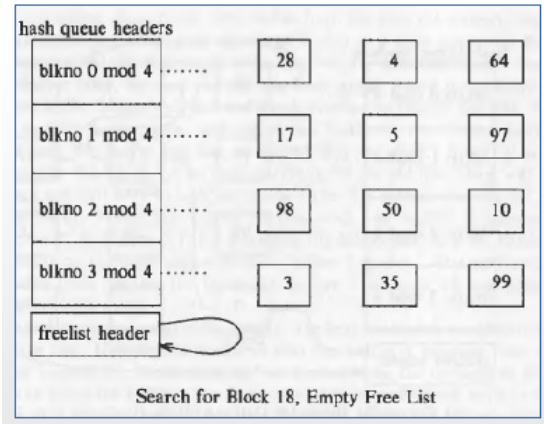
(a) Search for Block 18, Delayed Write Blocks on Free List

Estado de delay pasa a waiting

Si el siguiente tambien esta en DW, se sigue hasta encontrar otro. Una vez escritos en disco los bloques DW, se ubican al principio de la free list.

4to escenario

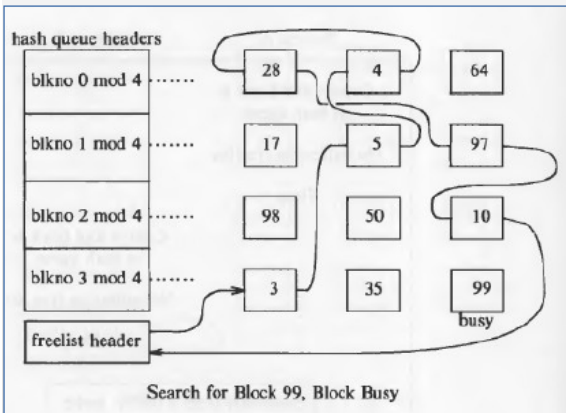
El kernel no encuentra el bloque en la hash queue y la free list esta vacia. El proceso queda bloqueado en espera a que se “libere” algun buffer. Cuando despierta, se debe verificar que el bloque no este en la hash queue (algun proceso pudo haberlo pedido mientras este dormia).



Search for Block 18, Empty Free List

5to escenario

El kernel busca un bloque y el buffer que lo contiene esta marcado como busy, entonces el proceso se bloquea a la espera de que el buffer se desbloquee.



Si el proceso que tenia ese buffer lo libera, entonces todos los que estaban a la espera de algun buffer se despiertan y reclaman el que estaban buscando en la hash queue y en la free list.