

RESUMEN 2DA PARTE TEÓRICO ISO

Administración de Memoria - II (Mr. Francinio):

Palabras Claves: Procesos, Espacio de Direcciones, Memoria, Seguridad, Página, Memoria Virtual, Tablas de Páginas.

Retomando lo visto sabemos que:

1. La memoria de un proceso no tiene porqué estar contigua en la memoria.
2. Se usa una tabla de páginas para manejarlo.
3. El SO es quien la administra.
4. El hardware traduce las direcciones lógicas a físicas de la tabla.

La razón para tener memoria virtual es que:

1. No toda la memoria del proceso es necesaria a la vez.
2. Algunas cosas no se vuelven a utilizar.
3. Otras son alocadas y liberadas dinámicamente.

Conjunto residente: Es el espacio de direcciones de un proceso que está en memoria. (Working Set).

El hardware detecta cuando una porción del proceso no está en el working set. Se debe cargar en memoria.

Ventajas de la memoria virtual:

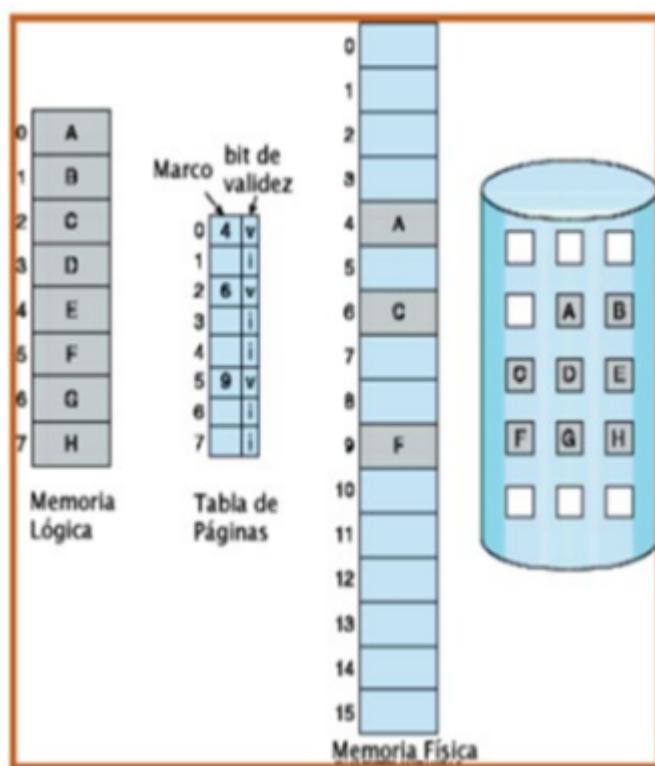
1. Más procesos en memoria principal porque solo se cargan las partes que el proceso necesita, no su totalidad, haciendo que sea más probable que haya más procesos en estado ready.
2. Si el proceso es más grande que la memoria no se debe preocupar por el tamaño de sus programas. Limitado por HW y bus de direcciones.

Para utilizar memoria virtual se requiere:

1. Soporte para paginación por demanda (y/o segmentación por demanda).
2. Memoria secundaria para área SWAP.
3. El SO debe ser capaz de manejar el movimiento de páginas/segmentos entre memoria principal y secundaria.

Memoria virtual con paginación:

1. Cada proceso tiene una tabla de páginas.
2. Cada entrada tiene una referencia a la base del marco/frame donde está la página en memoria principal.
3. Cada entrada tiene bits de control:
 - a. Bit V: Si está en memoria, manejado por el SO porque este es el encargado de subir y bajar páginas a memoria principal y consultado por el HW.
 - b. Bit M: Indica si fue modificada (se debe reflejar en memoria secundaria, específicamente en el área de SWAP), manejado por el HW porque este es el que detecta que hay una escritura en una dirección de memoria, consultado por SO.

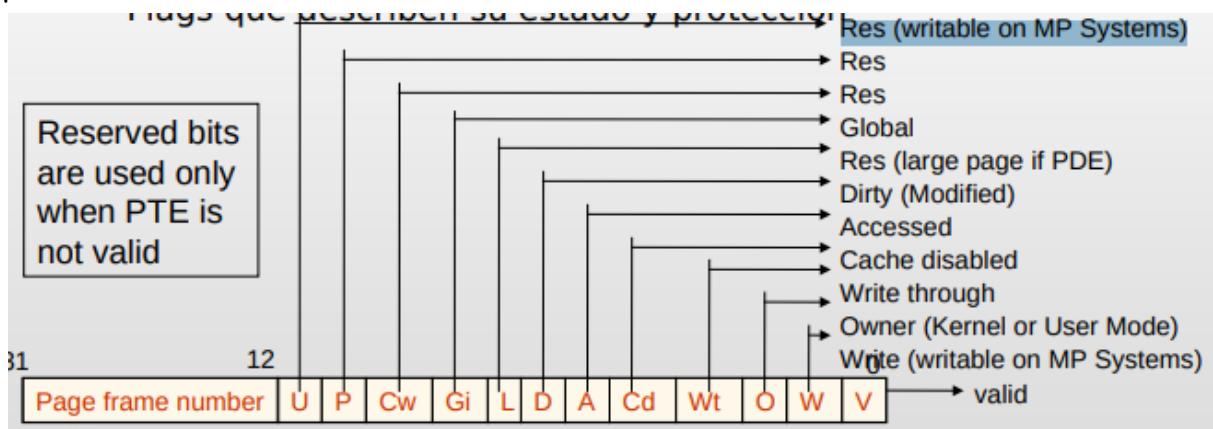


El HW define el formato de la tabla de páginas. El SO se adapta.

Una entrada válida tiene:

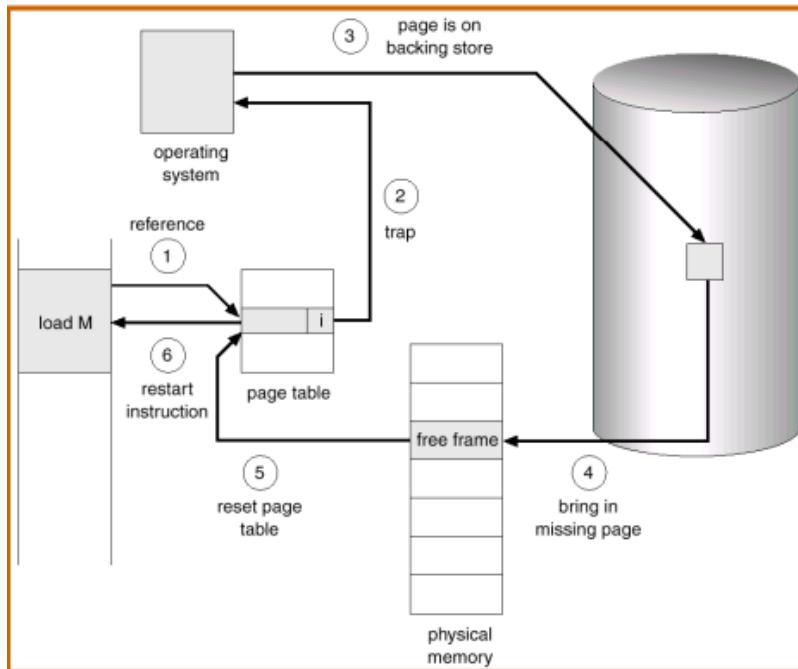
1. Bit V = 1
2. Page Frame Number (PFN) (marco de memoria asociado).
3. Flags.

Ejemplo:



Page Fault:

1. Ocurre cuando la dirección que el proceso quiere acceder no está en la memoria principal. (No está en el working set).
2. El HW detecta y genera un trap al S.O.
3. El SO coloca el proceso en estado "Blocked".
4. El SO busca un marco libre y si no hay, toma una víctima para reemplazar, y luego se genera una operación de E/S al disco para copiar al marco la página a usar. (Mientras se hace la E/S el SO puede prestarle la CPU a otro proceso), al terminar la E/S hará una interrupción.
5. Al finalizar la E/S actualiza la tabla de páginas con el bit V en 1 y pone la dirección base del frame donde quedó la página.
6. El proceso vuelve a estado Ready.
7. Cuando se ejecute el proceso se ejecutará desde la instrucción que generó PF.



Resumen de FP:

1. La técnica de paginación intenta alocar la mayor cantidad de páginas necesarias posibles.
2. Cada vez que hay que alocar una página en un marco, se produce un fallo de página
3. ¿Y si no hay marcos disponibles?
 - a. Se debe seleccionar una página víctima, para lo cual existen diversos algoritmos (FIFO, Óptimo, LRU, etc).
 - b. ¿Cuál es el mejor algoritmo?:
El que seleccione como página víctima aquella que no vaya a ser referenciada en un futuro.
4. Si los page faults son excesivos, la performance del sistema decae:
Tasa de Page Faults $0 \leq \rho \leq 1$
Si $\rho = 0$ no hay page faults
Si $\rho = 1$, cada acceso memoria genera un page fault
Effective Access Time (EAT) $EAT = (1 - \rho) \times \text{memory access} + \rho \times (\text{page_fault_overhead} + [\text{swap_page_out}] + \text{swap_page_in} + \text{restart_overhead})$

La tabla de páginas:

1. Su tamaño depende del espacio de direcciones del proceso.
2. Puede ser muy grande.

Formas de organizar:

Tabla de 1 nivel: Tabla única lineal

Tabla de 2 niveles (o más, multinivel)

Tabla invertida: Hashing

(La forma de organizarla depende del HW subyacente).

Si tenemos una dirección de 32 bits:

1. 20 son el número o índice de página, y 12 el desplazamiento.
2. La cantidad máxima de páginas contenidas en la tabla entonces sería 2^{20} ó 1.048.576.
3. Y el tamaño del marco en sí son 2^{12} o 4KB.
4. Si cada page table entry (PTE o puntero a la pagina), pesa 4B -> Un marco puede guardar 4KB/4B de PTE, o 2^{10} .
5. Para guardar todos los PTE's hacemos los siguiente: 2^{20} PTE/ 2^{10} PTE (2^{10} siendo los PTE's que entran en un marco) -> Necesitamos 2^{10} marcos para todas las páginas.
6. Como cada marco pesa 4KB, necesitamos $2^{10} * 4KB$ memoria o 4MB por proceso.

Misma lógica con 64 bits:

Tabla de 1 nivel - 64 bits

<input checked="" type="checkbox"/> Direcciones de 64bits	52 bits	12 bits
	Número de página	Desplazamiento

Ejemplo

- ✓ Cantidad de Page Table Entries (PTEs) máximas que puede tener un proceso = 2^{52}
- ✓ El tamaño de cada página es de 4KB
- ✓ El tamaño de cada PTE es de 4 bytes
 - ◆ Cantidad de PTEs que entran en un marco: $4KB/4B = 2^{10}$

Tamaño de tabla de páginas

- ◆ Cantidad de marcos necesarios para todas las PTEs de la tabla de páginas de un proceso = $2^{52}/2^{10} = 2^{42}$
- ◆ Tamaño tabla de páginas del proceso = $2^{42} * 4\text{bytes} = 2^{54}$

Más de 16.000GB por proceso!!!

ACLARACIÓN: El resultado final es correcto, el procedimiento está mal.

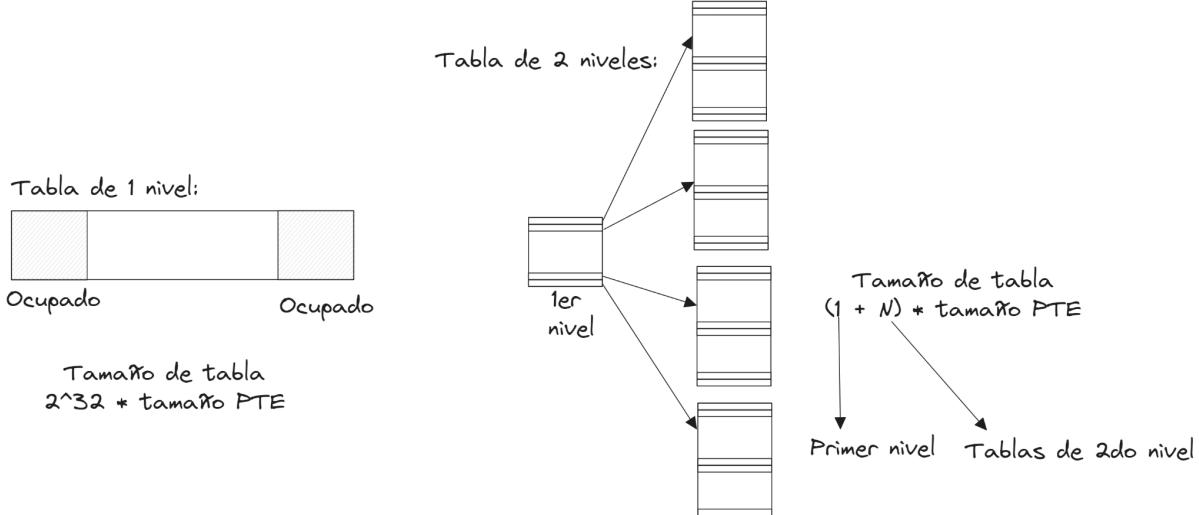
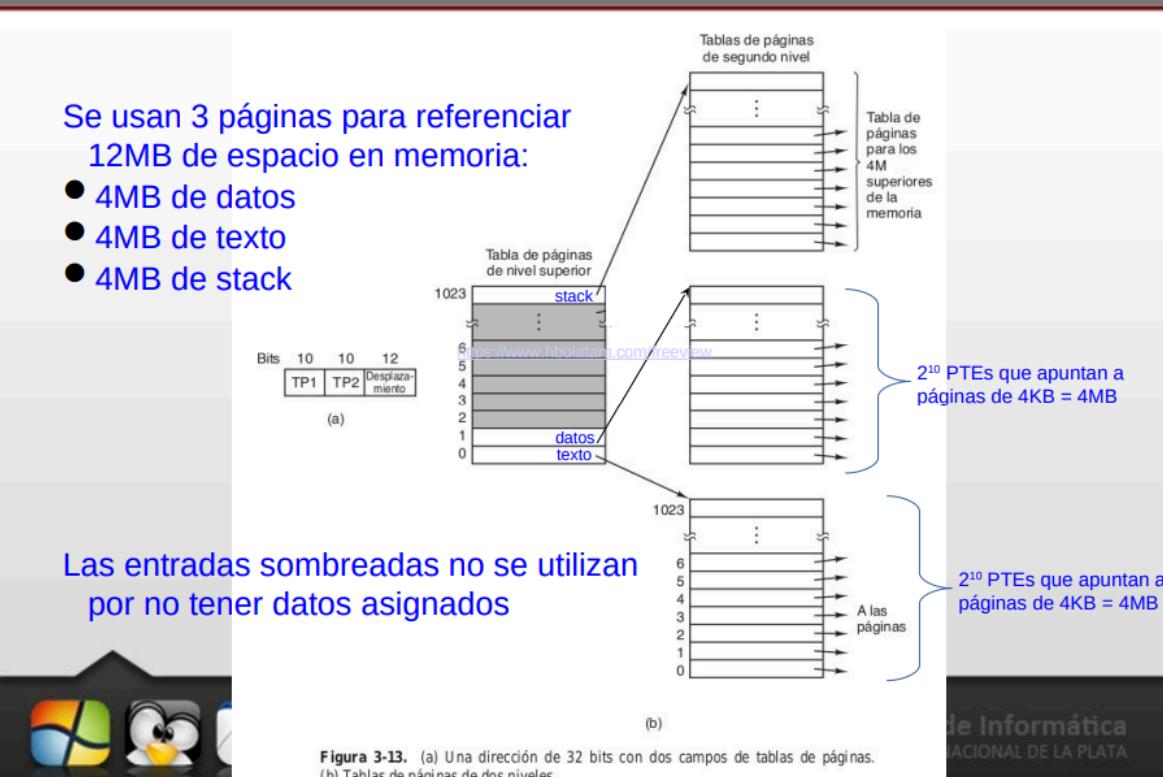
Tabla de páginas - Tabla de 2 niveles:

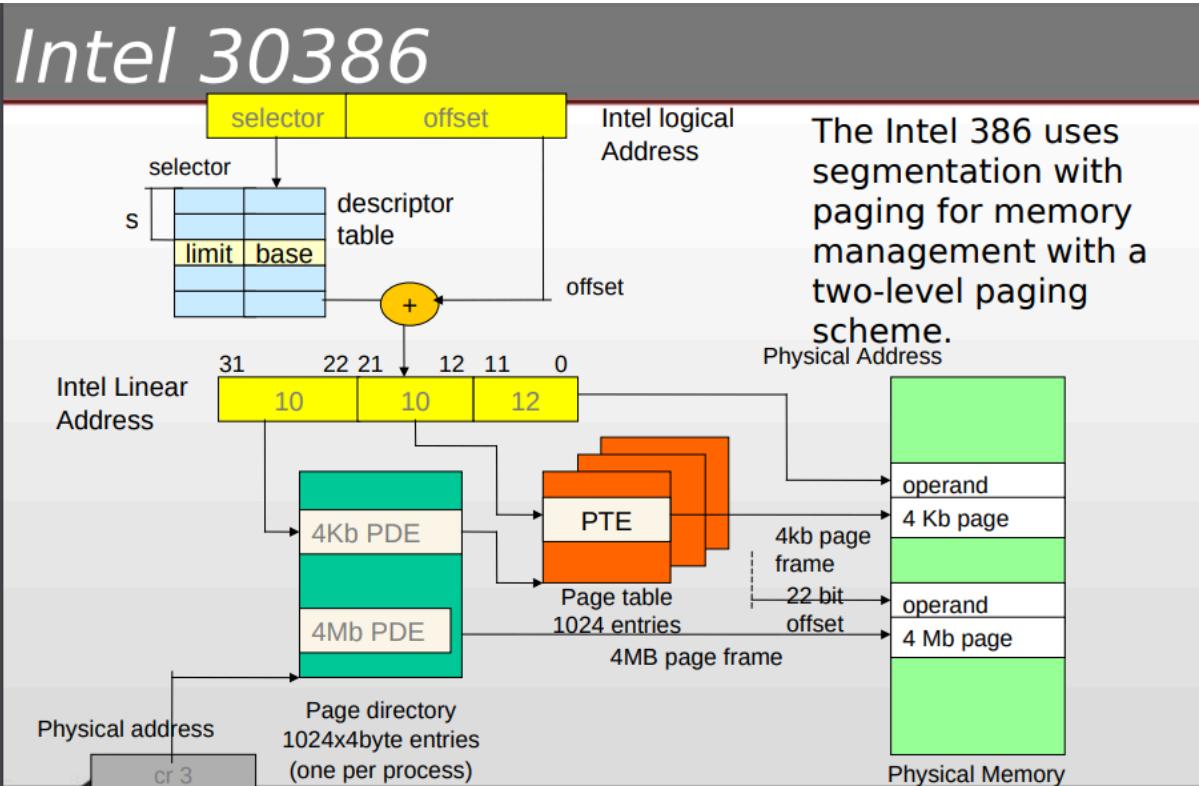
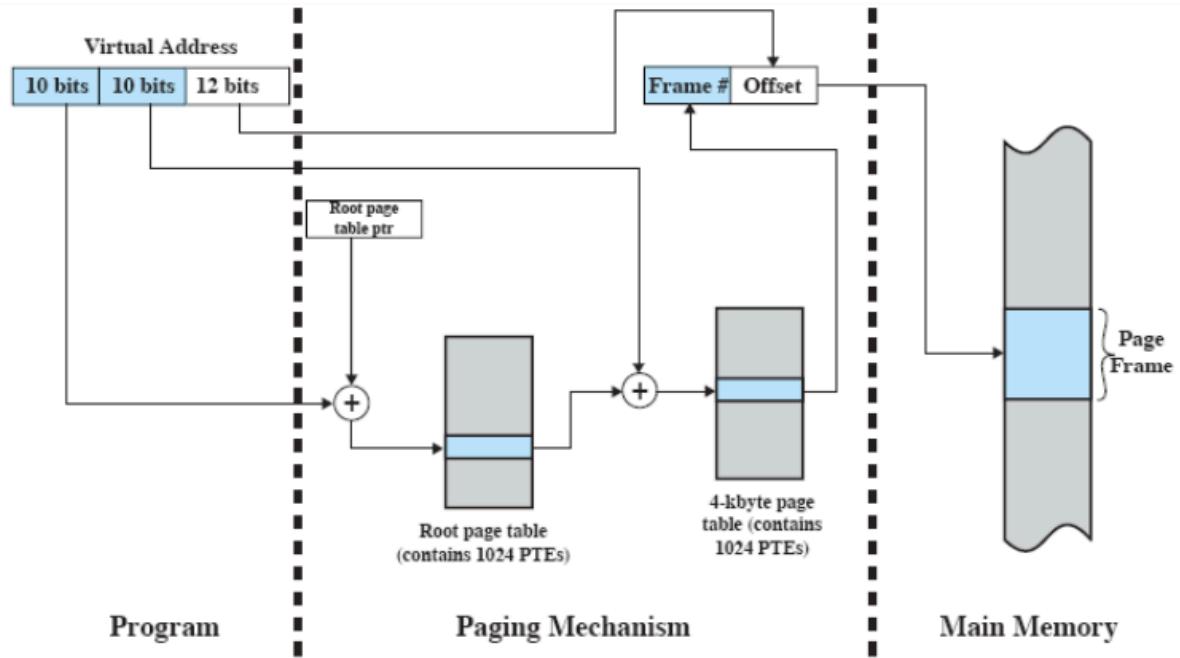
- El propósito de la tabla de páginas multinivel es dividir la tabla de páginas lineal en múltiples tablas de páginas.
- Cada tabla de páginas suele tener el mismo tamaño pero se busca que tengan un menor número de páginas por tabla.
- La idea general es que cada tabla sea más pequeña. Se busca que la tabla de páginas no ocupe demasiada memoria RAM.
- Además solo se carga una parcialidad de la tabla de páginas (solo lo que se necesite resolver).
- Existe un esquema de direccionamientos indirectos.
- Como beneficio se puede ver que las tablas de segundo nivel se pueden paginar, es decir, se pueden sacar de memoria principal, la única que debe permanecer en

memoria principal es la de primer nivel que contiene las direcciones bases a las de segundo nivel (video memoria 2, 37.44min).

- Como desventaja podemos ver que se necesitan más accesos a memoria para poder traducir una dirección.

Ejemplo: mapeo en memoria de tabla de páginas de 2 niveles





Tablas de Páginas - x64

- Se usan 47 bits para direccionar
- Usa tabla de páginas de 4 niveles

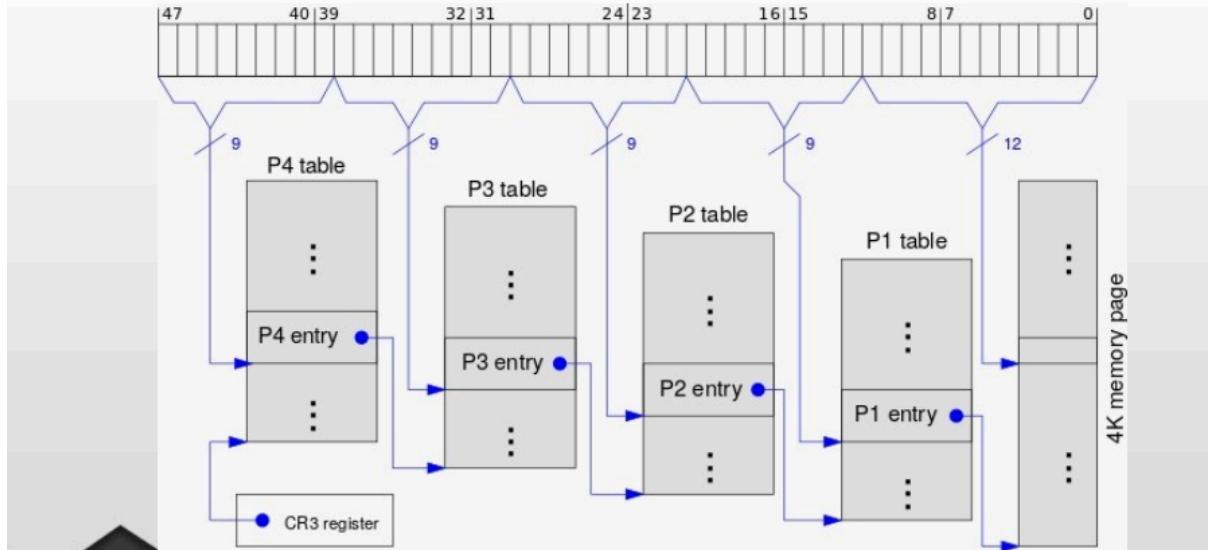
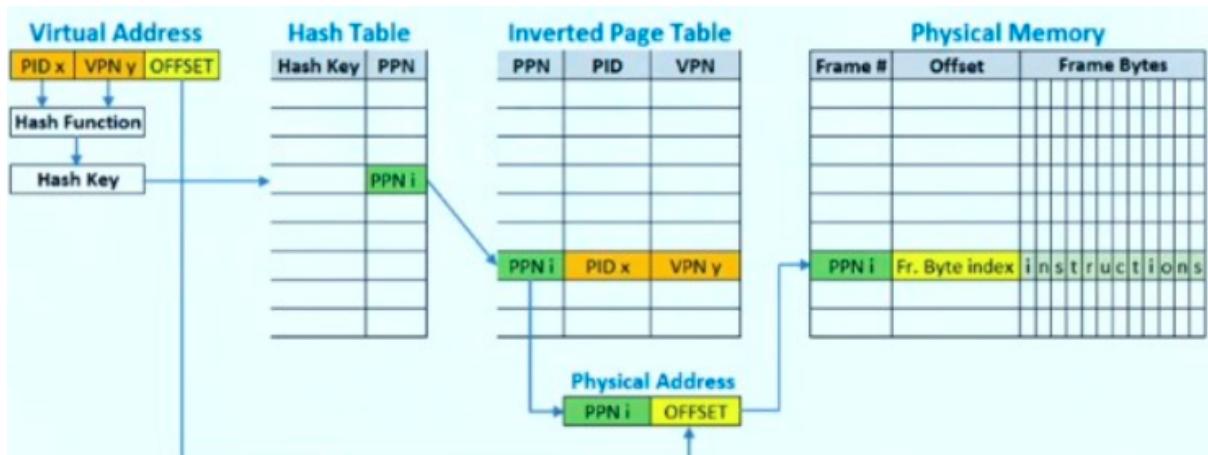


Tabla invertida:



1. Para aquellas arquitecturas donde el tamaño de página es muy grande. Las tablas multinivel requieren muchos niveles y no sería eficiente.
2. Si el espacio de direcciones es de 2^{64} bytes, con páginas de 4KB, necesitamos una tabla de páginas con 2^{52} entradas. Si cada entrada es de 8 bytes, la tabla es de más de 30 millones de Gigabytes (30 PB).
3. La tabla de páginas sólo contiene los PTE/punteros a las páginas que están cargadas en la memoria principal.

4. Una entrada para cada marco de memoria real.
5. Una tabla para todo el sistema.
6. El espacio de direcciones se refiere a la memoria física y no a la virtual.
7. Usada en PowerPC, UltraSPARC, y IA-64
8. El número de página es transformado en un valor de HASH. Para luego ser utilizado como índice y encontrar el marco correspondiente.
9. Mecanismo de encadenamiento para solucionar colisiones.

Tamaños de páginas:

- **Pequeño:** Menos fragmentación interna, más páginas por proceso, por lo tanto tablas de páginas más grandes. Más páginas en la memoria.
- **Grande:** Mayor fragmentación interna (no necesariamente en la última página). Ya que la memoria secundaria está diseñada para mover grandes bloques de datos, es más rápido escribirla en principal. Menos páginas por proceso y menos páginas en memoria al mismo tiempo.

Relación con la E/S:

- Vel. De transferencia: 2 Mb/s
- Latencia: 8 ms
- Búsqueda: 20 ms

Página de 512 bytes

- 1 página → total: 28,2 ms
- Sólo 0,2 ms de transferencia (1%)
- 2 páginas → 56,4 ms

Página de 1024 bytes

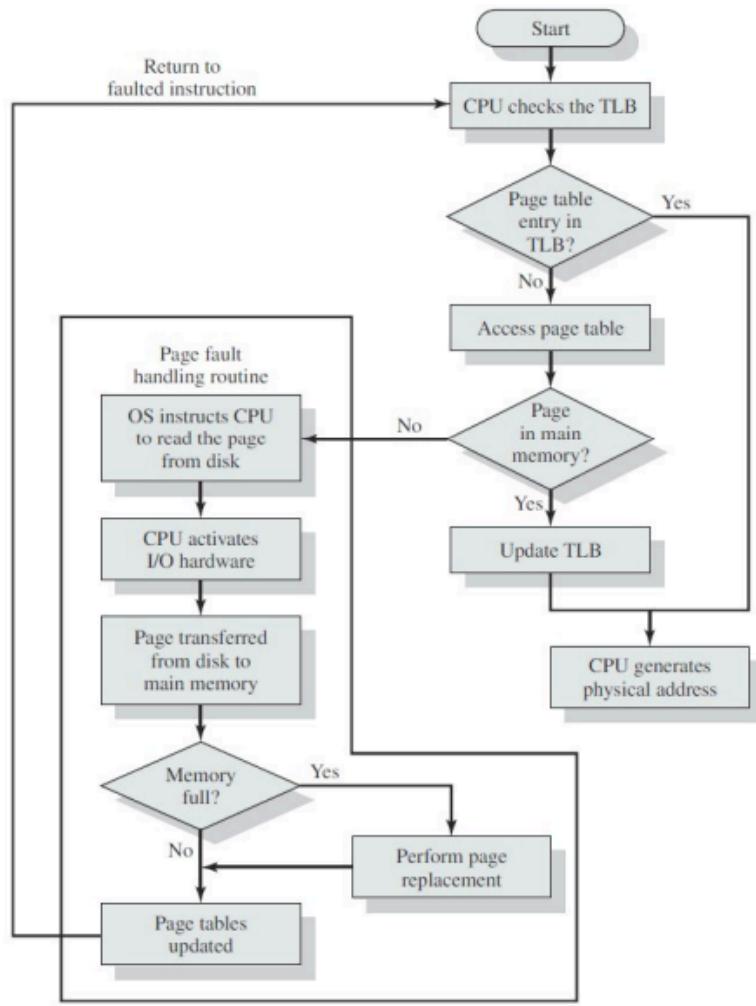
- total: 28,4 ms
- Sólo 0,4 ms de transferencia

Translation Lookaside Buffer (buffer de traducción adelantada):

Dado que hay que hacer 2 accesos a memoria para acceder

a la memoria física (primero en la tabla y segundo para los datos), se crea esta caché para entradas de páginas.

1. Funciona como la caché típica, primero se busca en la misma y si esta se produce un hit, se obtiene el marco y se arma la dirección física.
2. De no estar acá hay que usar el número de página para buscar en la tabla de páginas. De no estar ahí será PF.
3. Una vez recuperada se pone esta entrada en la TLB.
4. El cambio de contexto invalida las entradas de la TLB, por eso por ejemplo no sería recomendable su uso si vamos a utilizar un algoritmo Round Robin con Quantum chico, ya que los constantes cambios de contexto por parte de los procesos invalidarían frecuentemente a la TLB.



Parece que hay un MMU por core de procesador y cada MMU tiene una TLB (sacado de interne').

Políticas que el sistema operativo debe tener para manejar la memoria virtual:

Fetch Policy	<i>Cuando una página debe ser llevada a la memoria</i>	Resident Set Management
Demand paging		Resident set size
Prepaging		Fixed
	<i>Donde ubicarla (best-fit, first-fit, etc...)</i>	Variable
Placement Policy		<i>Cuántas páginas se traen a memoria</i>
Replacement Policy		Replacement Scope
Basic Algorithms		Global
Optimal	<i>Elección de víctima</i>	Local
Least recently used (LRU)		
First-in-first-out (FIFO)		
Clock		
Page Buffering		
Cleaning Policy	<i>Cuando una página modificada debe llevarse a disco</i>	
		Load Control
		# de procesos en memoria
		Degree of multiprogramming

Asignación de marcos a los procesos:

1. Un proceso puede tener una cantidad de páginas igual al tamaño del conjunto residente en memoria.
2. Si la asignación es dinámica la cantidad de marcos varía según la necesidad.
3. De ser asignación fija, esta es estática.

Tipos de asignación fija:

- **Asignación equitativa:** Ejemplo: si tengo 100 frames y 5 procesos, 20 frames para cada proceso. Es fácil de implementar pero podría pasar que algunos procesos se queden con frames de sobra porque no los usan, generando un desperdicio y mal uso de la memoria.
- **Asignación Proporcional:** Se asigna acorde al tamaño del proceso.

Alcance de reemplazo: Cuando hay que seleccionar una víctima para cargar una página en un marco, el alcance de

reemplazo puede ser:

- **Global:**
 - Puede reemplazar páginas de cualquier proceso.
 - El SO no controla la tasa de PF por proceso porque no puede saber si es el propio proceso el que se está reemplazando sus páginas o si es otro que le está robando marcos disponibles.
 - Los procesos pueden cambiar la cantidad total de marcos.
 - Procesos de alta prioridad pasan por arriba a los de baja.
- **Local:**
 - Un proceso solo puede reemplazar páginas con su conjunto residente.
 - No cambia la cantidad de frames asignados.
 - Permite contabilizar la tasa de PF por proceso.
 - Un proceso puede tener frames asignados sin usar.

	Local Replacement	Global Replacement
Fixed Allocation	<ul style="list-style-type: none">• Number of frames allocated to a process is fixed.• Page to be replaced is chosen from among the frames allocated to that process.	<ul style="list-style-type: none">• Not possible.
Variable Allocation	<ul style="list-style-type: none">• 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.	<ul style="list-style-type: none">• 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.

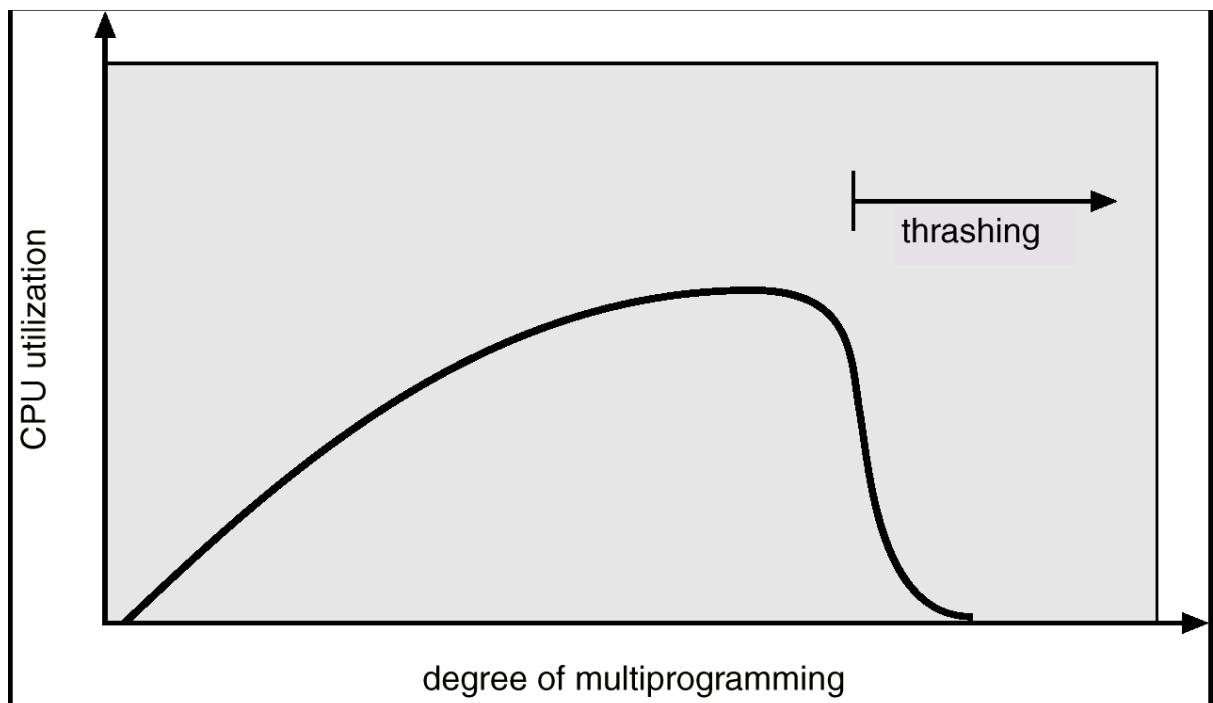
Algoritmos de reemplazo:

1. **ÓPTIMO:** Es solo teórico.
2. **FIFO:** Es el más sencillo.
3. **LRU (Least Recently Used):** Requiere soporte del hardware para mantener timestamps de acceso a las páginas. Favorece a las páginas más recientemente accedidas.

4. 2da. Chance: Un avance del FIFO tradicional que beneficia a las páginas más referenciadas.
5. NRU (Non Recently Used): Utiliza bits R y M Favorece a las páginas que fueron usadas recientemente.

Memoria III (Ivancinho)

- **Thrashing (hiperpaginación)**
 - Un sistema está en thrashing cuando pasa más tiempo paginando que ejecutando procesos.
 - Como consecuencia, hay una baja importante de performance en el sistema.
- **Ciclo del thrashing**
 1. El kernel monitorea el uso de la CPU.
 2. Si hay baja utilización aumenta el grado de multiprogramación.
 3. Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos.
 4. Un proceso necesita más frames. Comienzan los page-faults y robo de frames a otros procesos.
 5. Por swapping de páginas, y encolamiento en dispositivos, baja el uso de la CPU.
 6. Vuelve a 1).



- **El scheduler de CPU y el thrashing**

1. Cuando se decrementa el uso de la CPU, el scheduler long term aumenta el grado de multiprogramación.
2. El nuevo proceso inicia nuevos page-faults, y por lo tanto, más actividad de paginado.
3. Se decrementa el uso de la CPU
4. Vuelve a 1).

- **Control del thrashing**

- Se puede limitar el thrashing usando algoritmos de reemplazo local.
- Con este algoritmo, si un proceso entra en thrashing no roba frames a otros procesos.
- Si bien perjudica la performance del sistema, es controlable.

- **Conclusión sobre thrashing**

- Si un proceso cuenta con todos los frames/páginas que necesita, no habría thrashing.

- Una manera de abordar esta problemática es utilizando la estrategia de Working Set, la cual se apoya en el modelo de localidad
- Otra estrategia con el mismo espíritu es la del algoritmo PFF (Frecuencia de Fallos de Página)

- ***El modelo de localidad***

- Cercanía de referencias o principio de cercanía
- Las referencias a datos y programa dentro de un proceso tienden a agruparse
- La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento.
- En cortos períodos de tiempo, el proceso necesitará pocas “piezas” del proceso (por ejemplo, una página de instrucciones y otra de datos...)

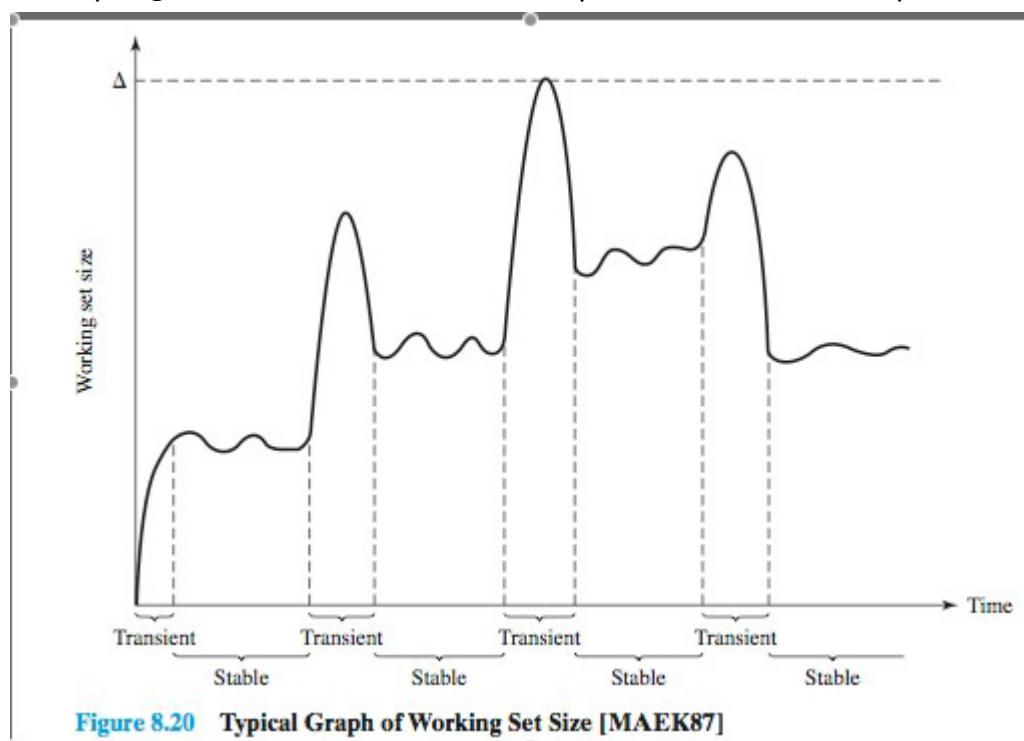


Figure 8.20 Typical Graph of Working Set Size [MAEK87]

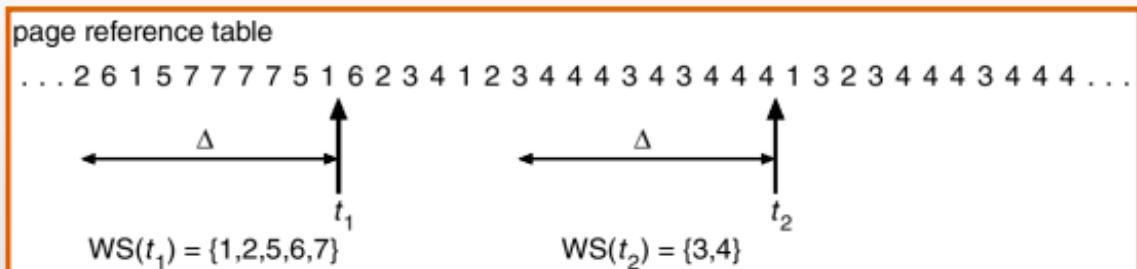
- Un programa se compone de varias localidades.
- Ejemplo: Cada rutina será una nueva localidad: se refieren sus direcciones (cercanas) cuando se está ejecutando.

- Para prevenir la hiperpaginación, un proceso debe tener en memoria sus páginas más activas (menos page faults).

- **El modelo de working set**

- Se basa en el modelo de localidad.
- **Ventana del working set (Δ)**: las referencias de memoria más recientes.
- **Working set**: es el conjunto de páginas que tienen las más recientes referencias a páginas (Δ).
- Es costosa esta técnica ya que el SO tiene que implementar una lógica para analizar toda la estructura de la tabla de páginas, mantener en otra estructura de datos las páginas que aparecen con el bit R = 1, es decir, las que están siendo utilizadas en la localidad y limpiar los bits de R en la tabla de páginas.

$\Delta=10$



- **La selección del Δ**

- Δ chico: no cubrirá la localidad y haría que se generen posibles fallos de página.
- Δ grande: puede tomar varias localidades y hacer un mal uso de la memoria principal cargando páginas de una localidad que ya no se usa.

- **Medida del working set**

- m = cantidad frames disponibles
- WSS_i = tamaño del working set del proceso π_i .
- $WSS_i=D$;
- D = demanda total de frames por parte de todos los procesos.
- Si $D > m$, habrá thrashing.
- **Conclusión:** Si la demanda de frames que tienen todos los procesos juntos, supera la cantidad de frames disponibles de la RAM, entonces estamos frente a una situación de hiperpaginación. La solución de esta situación es bajar el grado de multiprogramación en pos de achicar la demanda de frames.

- **Prevención del thrashing**

- SO monitorea c/ proceso, dándole tantos frames hasta su WSS_i (medida del working set del proceso π_i)
- Si quedan frames, puede iniciar otro proceso.
- Si D crece, excediendo m , se elige un proceso para suspender, reasignándose sus frames...
- Así, se mantiene alto el grado de multiprogramación optimizando el uso de la CPU.

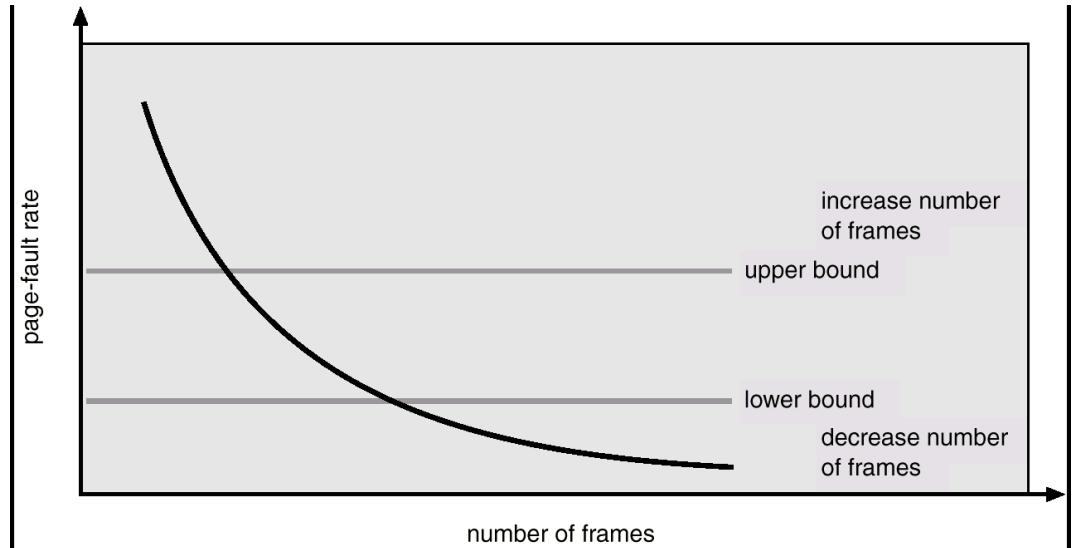
- **Problema del modelo del WS**

- Mantener un registro de los WSS_i (medida del working set del proceso π_i)
- La ventana es móvil

- **Prevención del thrashing por PFF**

- La técnica PFF (Page Fault Frequency o Frecuencia de Fallo de Páginas), en lugar de calcular el WS de los procesos, utiliza la tasa de fallos de página para estimar si el proceso tiene un conjunto residente que representa adecuadamente al WS.
- Es más factible de implementar que el Working Set.

- Requiere del uso de reemplazo local.
- PFF: Frecuencia de page faults
- PFF alta: Se necesitan más frames
- PFF baja: Los procesos tienen frames asignados que le sobran.



- Establecer tasa de PF aceptable
 - Si la tasa actual es baja, el proceso puede perder frames
 - Si la tasa actual es alta, el proceso gana frames.
- Establecer límites superior e inferior de las PFF's deseadas.
- Si se Excede PFF máx. Se le asignan más frames al proceso, ya que el mismo genera muchos PF y probablemente los esté necesitando.
- Si la PFF está por debajo del mínimo se le pueden quitar frames al proceso para ser utilizados en los que necesitan más.
- Se puede llegar a suspender un proceso si no hay más frames libres y todos los procesos están por arriba de PFF Máx

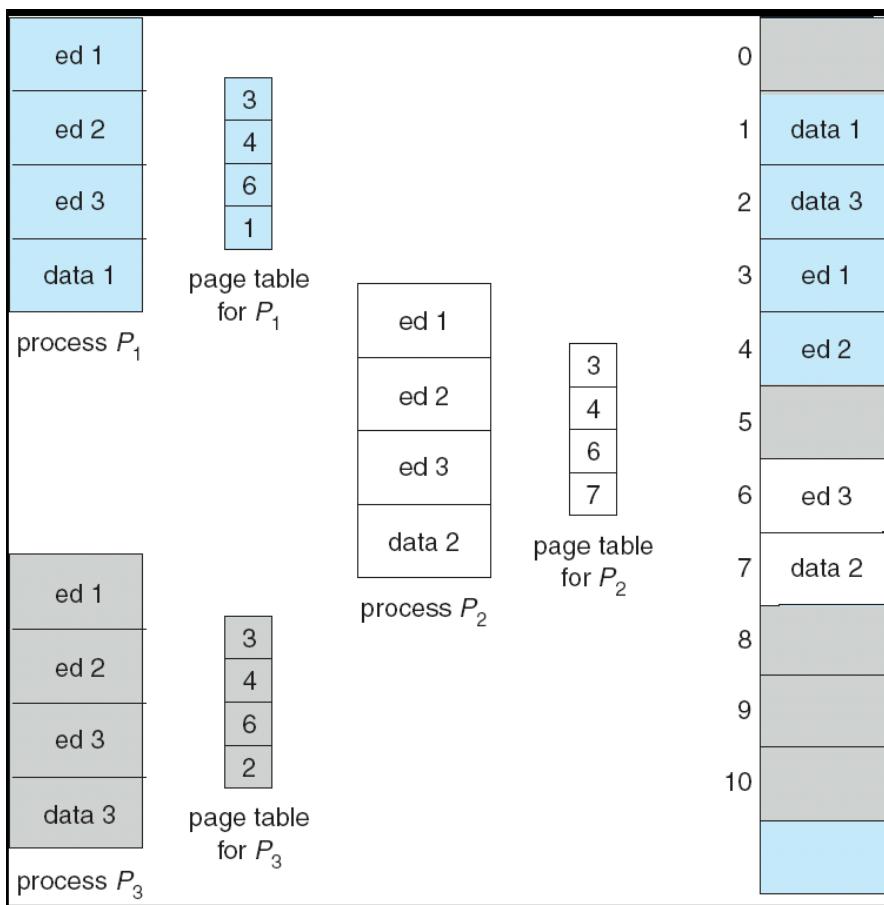
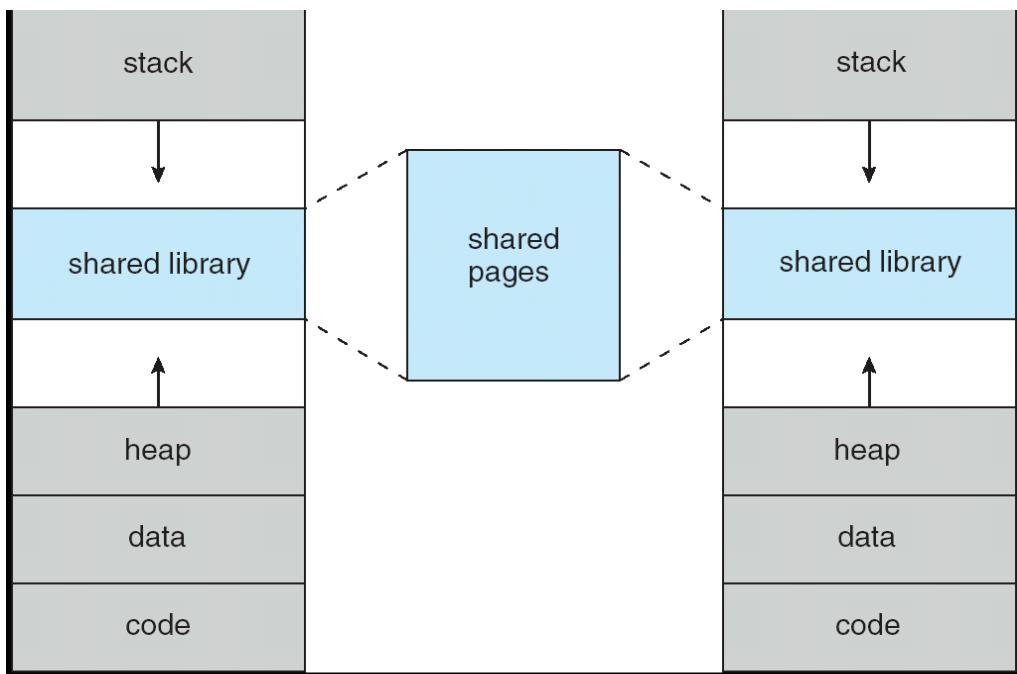
- **Demonio de paginación**

- Proceso creado por el SO durante el arranque que apoya a la administración de la memoria

- Se ejecuta cuando el sistema tiene una baja utilización o algún parámetro de la memoria lo indica
 - Poca memoria libre
 - Mucha memoria modificada
- Tareas:
 - Limpiar páginas modificadas sincronizándolas con el swap (evita que estas se guarden únicamente cuando las páginas modificadas son seleccionadas como víctimas, mejorando performance del intercambio de páginas)
 - Reducir el tiempo de swap posterior ya que las páginas están “limpias”
 - Reducir el tiempo de transferencia al sincronizar varias páginas contiguas.
 - Mantener un cierto número de marcos libres en el sistema.
 - Demorar la liberación de una página hasta que haga falta realmente
 - Regula el crecimiento o decrecimiento de los working sets

- **Memoria compartida**

- Gracias al uso de la tabla de páginas varios procesos pueden compartir un marco de memoria; para ello ese marco debe estar asociado a una página en la tabla de páginas de cada proceso
- El número de página asociado al marco puede ser diferente en cada proceso
- Código compartido
 - Los procesos comparten una copia de código (sólo lectura) por ej. editores de texto, compiladores, etc
 - Los datos son privados a cada proceso y se encuentran en páginas no compartidas



• **Mapeo de Archivo en Memoria**

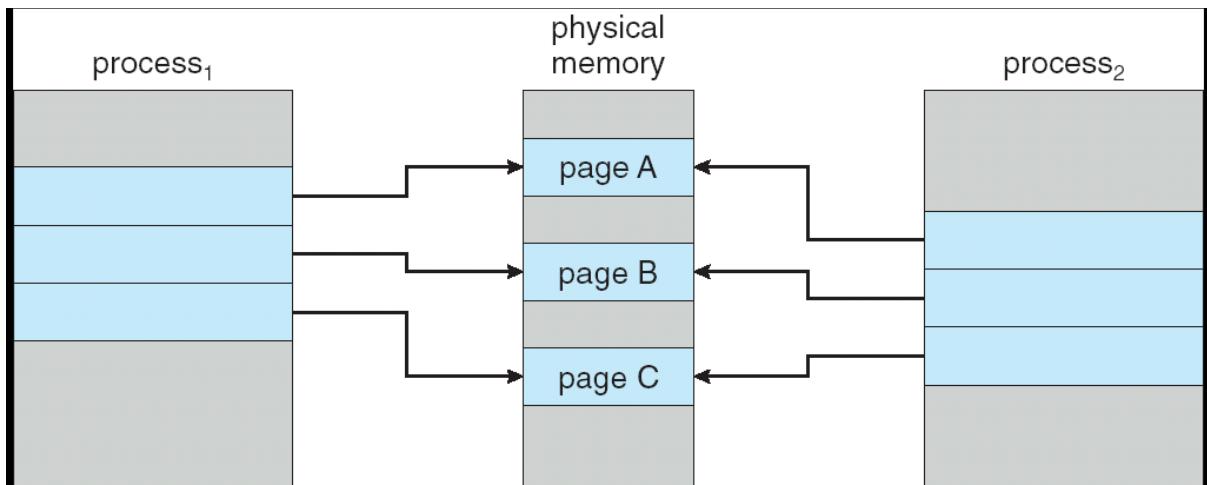
- Técnica que permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtuales

- El contenido del archivo no se sube a memoria hasta que se generan Page Faults
- El contenido de la página que genera el PF es obtenido desde el archivo asociado
 - No del área de intercambio
- Cuando el proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo correspondiente
- Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el Sistema de Archivos
- Es utilizado comúnmente para asociar bibliotecas compartidas o DLLs

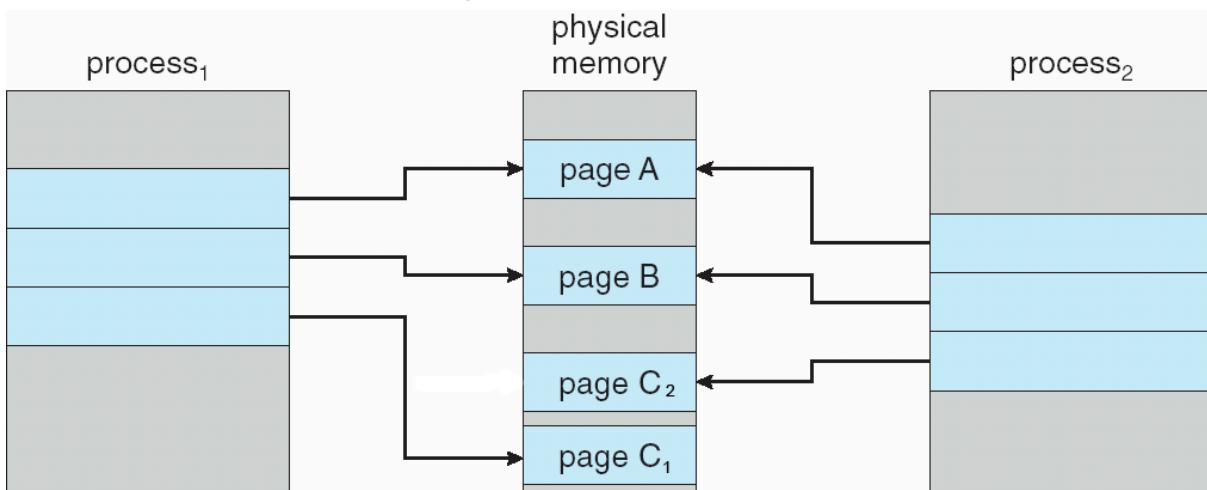
- **Copia en Escritura**

- La copia en escritura (Copy-on-Write, COW) permite a los procesos compartir inicialmente las mismas páginas de memoria
 - Si uno de ellos modifica una página compartida la página es copiada
 - En fork, permite inicialmente que padre e hijo utilicen las mismas páginas sin necesidad de duplicación.
- COW permite crear procesos de forma más eficiente debido a que sólo las páginas modificadas son duplicadas

El Proceso 1 Modifica la Página C (Antes)



El Proceso 1 Modifica la Página C (Despues)

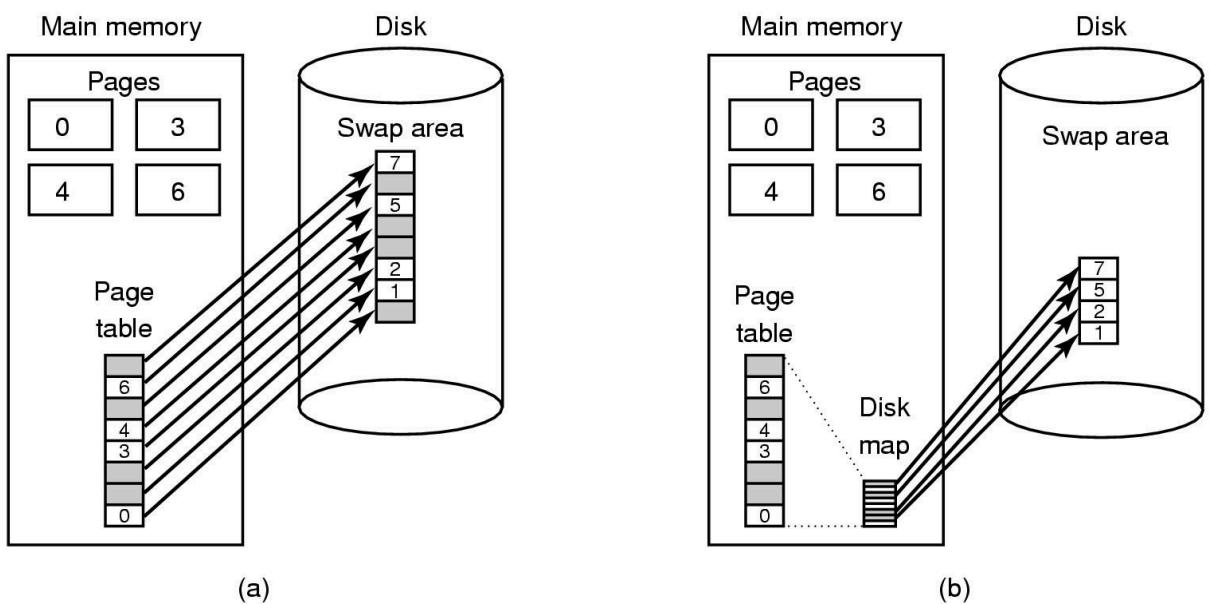


- **Área de Intercambio**

- Sobre el Área utilizada
 - Área dedicada, separada del Sistema de Archivos (Por ejemplo, en Linux)
 - Un archivo dentro del Sistema de Archivos (Por ejemplo, Windows)
- Técnicas para la Administración:
 - a. Cada vez que se crea un proceso se reserva una zona del área de intercambio igual al tamaño de imagen del proceso. A cada proceso se le asigna la dirección en disco de su área de intercambio. La lectura se realiza sumando el número de página virtual a la

dirección de comienzo del área asignada al proceso.

- b. No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a intercambiar, y el espacio se libera cuando la página vuelve a memoria.
Problema: se debe llevar contabilidad en memoria (página a página) de la localización de las páginas en disco.



- Cuando una página no está en memoria, sino en swap, como podríamos saber en qué parte del área de intercambio está?
 - Rta: El PTE de dicha página tiene el bit V=0 y todos los demás bits sin usar!

Anexo Administración de memoria (*Felipinho*)

Mapa de Memoria

Un mapa de memoria, (memory map o memory layout) es una estructura de datos que indica al sistema operativo cómo está distribuida la memoria de un proceso, los segmentos que la componen, y los datos almacenados en cada uno de ellos.

Cuando se ejecuta un programa en GNU/Linux, se crea un mapa de memoria en la que se carga variables inicializadas, variables no inicializadas, segmentos de memoria dinámica, la pila, y el código binario de la aplicación (o una parte de él).

GNU/Linux utiliza gestión de memoria basada en segmentación y paginación

Las páginas de memoria son de tamaño fijo, por ejemplo, 4 KiB (podemos leer el tamaño de la página con el comando `getconf PAGESIZE`), mientras que la segmentación implica segmentos de tamaño variable.

GNU/Linux divide el mapa de memoria de un proceso en segmentos de diferentes tamaños, cada uno dividido, a su vez, en páginas de tamaño fijo.

El mapa de memoria de un proceso se divide generalmente en 6 segmentos:

1-Argumentos de la línea de comandos y variables de entorno

2-Stack o pila del proceso: En este segmento se almacenan las variables locales de las funciones, los argumentos pasados a cada función llamada, y los punteros de retorno de las mismas.

Cuando se llama a una función se crea un marco de pila, se almacenan variables locales, argumentos y la dirección de retorno de la función. Cuando la función termina, se carga como siguiente instrucción la dirección de retorno, de modo que la ejecución continúe por donde iba antes de llamar a la función, y se procede a eliminar el marco de pila

3-Heap o espacio para almacenar segmentos de memoria dinámica: En este segmento se almacena la memoria dinámica creada por el proceso (`malloc()`, `calloc()`, `realloc()` por ejemplo).

Cuando se reserva una posición para memoria dinámica el programa adquiere dicho espacio desde el heap. Si se libera dicha memoria (`free()`), se reduce el espacio ocupado dentro del heap.

Cuando se libera espacio de memoria dinámica se devuelve al heap, pero no a la memoria del sistema operativo, por lo que el heap puede que comience a fragmentarse.

4-Datos no inicializados (BSS): Este segmento, también conocido como BSS (heredado de lenguaje ensamblador) almacena todas las variables globales y estáticas que no están inicializadas a cero o no tienen un valor de inicialización en el código fuente

5-Datos inicializados: Contiene las variables globales y estáticas del programa, que a su vez fueron inicializadas con valores distintos de cero. Este segmento puede clasificarse como de sólo lectura y de lectura-escritura.

6-Segmento de texto (código binario de un programa): Este segmento (de solo Lectura) almacena las instrucciones ejecutables del programa, por lo que también se lo denomina segmento de código.

Almacena la representación en código de máquina de las instrucciones del programa. Este segmento en general puede ser compartido entre diferentes procesos, ya que no se modifica, y si contiene código de bibliotecas compartidas no es necesario duplicarlo en memoria.

File System 1 (Coco)

- **¿Qué es un archivo?**

- ❖ Entidad abstracta con nombre.
- ❖ Espacio lógico continuo y direccionable.
- ❖ Provee a los programas con datos (entradas).
- ❖ Permite a los programas guardar datos (salida).

- ❖ El programa en sí mismo es información que necesita guardarse.

- ***¿Para qué se necesitan los archivos?***

- ❖ Almacenar grandes cantidades de datos
- ❖ Almacenamiento a largo plazo
- ❖ Permitir que varios procesos accedan al mismo conjunto de información.

- **Archivos**

- *Punto de vista del diseño/SO*
 - ❖ Implementar archivos.
 - ❖ Implementar directorios.
 - ❖ Manejo del espacio en disco.
 - ❖ Manejo del espacio libre.
 - ❖ Eficiencia y mantenimiento.
- *Punto de vista del usuario*
 - ❖ Operaciones que se pueden realizar.
 - ❖ Cómo renombrar un archivo.
 - ❖ Cómo asegurar la protección/integridad del archivo.
 - ❖ Cómo compartir archivos ya sea con otras personas o entre procesos en ejecución.
 - ❖ No tratar aspectos físicos

- **Sistema de manejo de archivos (FileSystem)**

- ❖ Se puede ver como la parte del SO encargada del manejo de los archivos.
- ❖ Conjunto de unidades de software que proveen los servicios necesarios para la utilización de archivos
 - ➔ Crear
 - ➔ Borrar
 - ➔ Buscar
 - ➔ Copiar
 - ➔ Leer
 - ➔ Escribir
 - ➔ etc.

- ❖ Facilita el acceso a los archivos por parte de las aplicaciones
- ❖ Permite abstracción al programador en cuanto al acceso a bajo nivel, ya que no es él quien desarrolla el FileSystem

- ***Objetivos del sistema operativo con los archivos***

- ❖ Cumplir con la gestión de datos
- ❖ Cumplir con las solicitudes del usuario (leer, escribir, etc).
- ❖ Minimizar / eliminar la posibilidad de perder o destruir datos (Garantiza su integridad).
- ❖ Dar soporte de E/S a distintos dispositivos con accesos uniformes a la información en pos de una estandarización de los mismos.
- ❖ Brindar un conjunto de interfaces de E/S para el tratamiento de archivos.

- ***Tipos de archivos***

- Archivos regulares:**

- ❖ Texto plano:
→ Source File.
 - ❖ Binarios:
→ Object File.
→ Executable File.

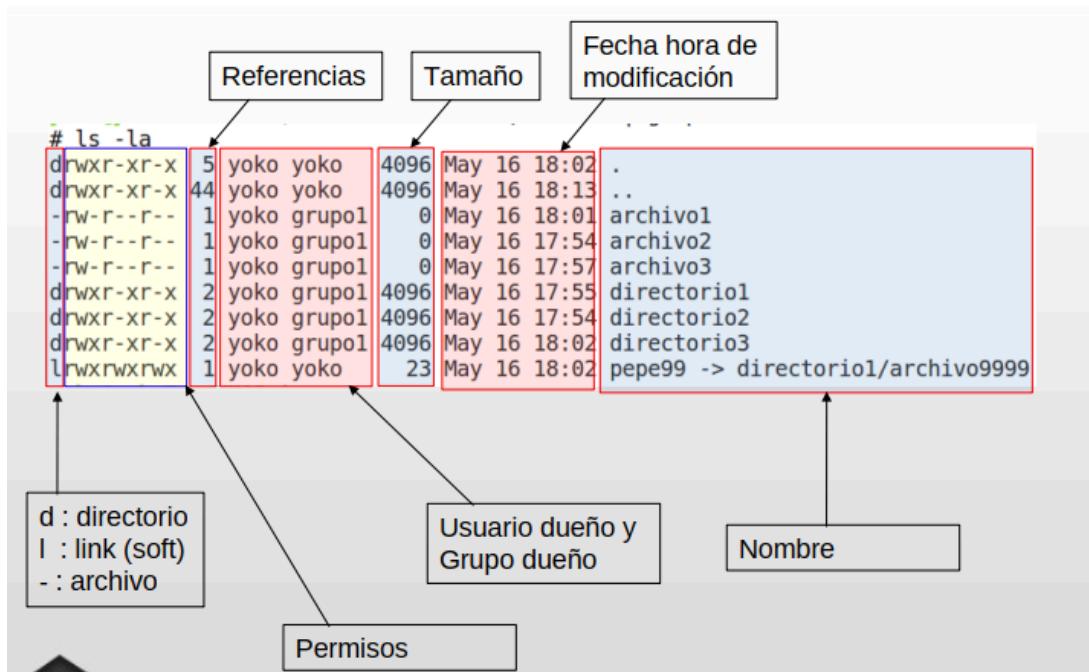
- Directarios:**

- ❖ Se consideran archivos porque son un conjunto de información que está relacionada e individualizada.
 - ❖ Contiene información sobre los archivos que contiene en su interior.
 - ❖ Permite mantener la estructura en el FileSystem (árbol de directorios).

- ***Atributos de un Archivo***

- ❖ Nombre (identificador para el usuario).

- ❖ Identificador (identificador para el sistema operativo, no usa el nombre porque este puede ser muy grande y el kernel necesitaría reservar bastante almacenamiento para nombres largos).
- ❖ Tipo.
- ❖ Localización.
- ❖ Tamaño.
- ❖ Protección, seguridad y monitoreo.
 - Owner, Permisos, Password.
 - Momento en que el usuario lo modifico, creo, accedió por última vez.
 - ACLs “Access control list/Lista de Control de Acceso”.

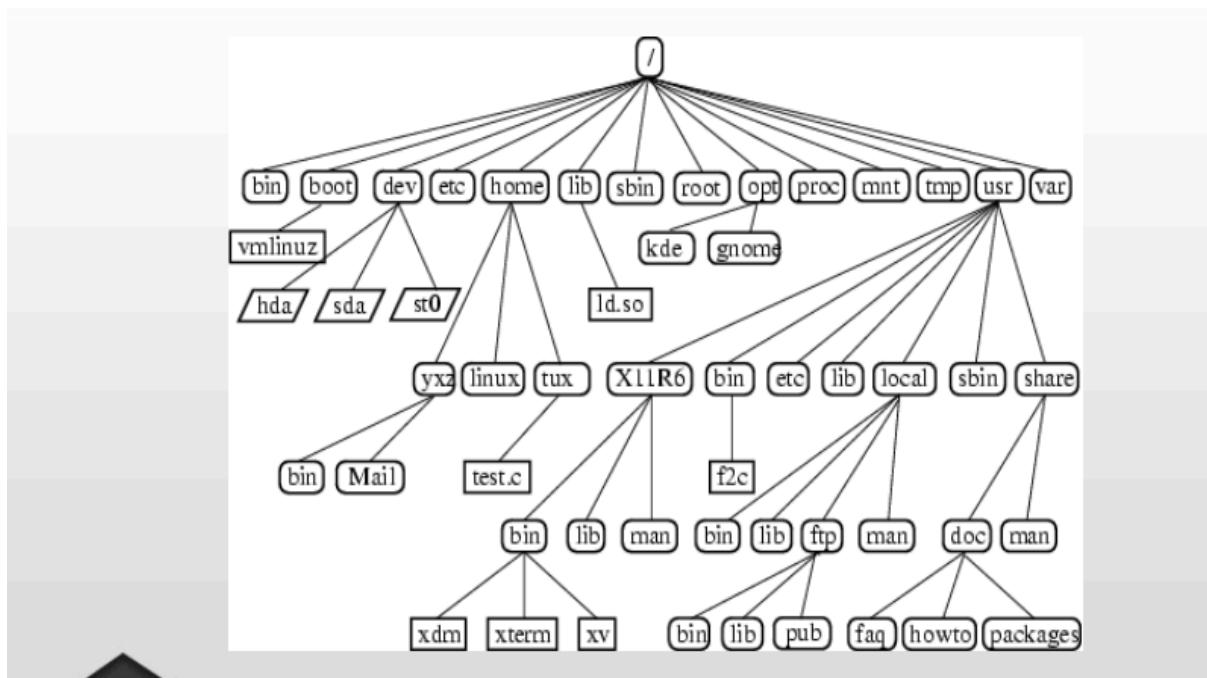


• Directrios

- ❖ Contiene información acerca de archivos y directorios que están dentro de él.
- ❖ El directorio es, en sí mismo, un archivo.
- ❖ Interviene en la resolución entre el nombre y el archivo mismo.
- ❖ Operaciones en directorios:
 - Buscar un archivo

- ➔ Crear un archivo (entrada de directorio)
 - ➔ Borrar un archivo
 - ➔ Listar el contenido
 - ➔ Renombrar archivos
 - ➔ etc.
- ❖ El uso de directorios ayuda con:
- ➔ La eficiencia: localización rápida de archivos
 - ➔ Uso del mismo nombre de archivos
 - ➔ Agrupación lógica de archivos por propiedades / funciones (JAVA, juegos, libreria)

Estructura de Dir. Jerárquica o Arbol



- ***Estructura de directorios***

- ❖ Los archivos pueden ubicarse siguiendo un path desde el directorio raíz y sus sucesivas referencias (path absoluto).
- ❖ Distintos archivos pueden tener el mismo nombre pero el full path name es único.
- ❖ El directorio actual del proceso se llama **working directory**.
- ❖ Dentro del directorio de trabajo se pueden referenciar archivos tanto por su **PATH absoluto**

como por su PATH relativo indicando solamente la ruta al archivo DESDE el working directory.

- **Absolute**: el path incluye todo el camino del archivo.
- **Relativo** (al working directory actual): el nombre se calcula relativamente al directorio en el que se esté.

- **Compartir archivos**

- ❖ En ambientes multiusuario es necesario que varios usuarios puedan compartir archivos.
- ❖ La compartición se debe realizar bajo un esquema de protección donde el usuario / administrador es capaz de controlar que se puede hacer (derechos de acceso) y quién puede hacerlo.

- **Derechos de acceso**

- Los directorios también tienen permisos, los cuales pueden permitir el acceso al mismo para que el usuario pueda usar el archivo siempre y cuando tenga los permisos necesarios.
- Execution - Permiso de ejecución.
- Reading - Permiso de lectura.
- Appending - Permiso de agregar datos (no está permitido modificar ni borrar el contenido del archivo, por eso cada vez que se agregan datos, se crean nuevas versiones del archivo original como archivos nuevos, no es muy usado actualmente).
- Updating - Permiso de actualización el usuario puede modificar, borrar y agregar datos. Incluye crear archivos, sobreescibirlos, remover datos de ellos.
- Changing Protection - Permiso de cambio de protección, el usuario puede modificar los derechos de acceso al archivo.

→ **Deletion** - Permiso de borrado el usuario puede borrar el archivo.

- **Owners**

- ❖ Posee todos los derechos sobre el archivo.
- ❖ Puede dar derechos a otros usuarios. Se determinan clases:
 - ➔ Derechos para un usuario específico.
 - ➔ Derechos para grupos de usuarios.
 - ➔ Derechos para todos (archivos públicos).

SISTEMAS DE ARCHIVOS 2 (Tomeik125):

Metas: Brindar espacio en disco manteniendo un registro de cantidad y ubicación de ese espacio libre

Conceptos:

- **Sector:** Unidad de almacenamiento del disco.
- **Bloque:** Conjunto de sectores.
- **File system:** Forma en que se almacenan los datos, también se le dice File System a la partición del disco.
- **FAT:** Contiene información sobre la ubicación de los archivos, esta estructura se almacena en disco y cada vez que se enciende el sistema, se trae a memoria para poder trabajar con los archivos, no se crea dinámicamente (File allocation table).

ASIGNACIONES DE ESPACIO A LOS ARCHIVOS:

Pre-asignación:

- Este método de asignación se utilizaba cuando los archivos se tenían que almacenar de forma contigua en disco. Requiere saber cuánto espacio ocupa el archivo al crearse, tiende a definir espacios más grandes de lo

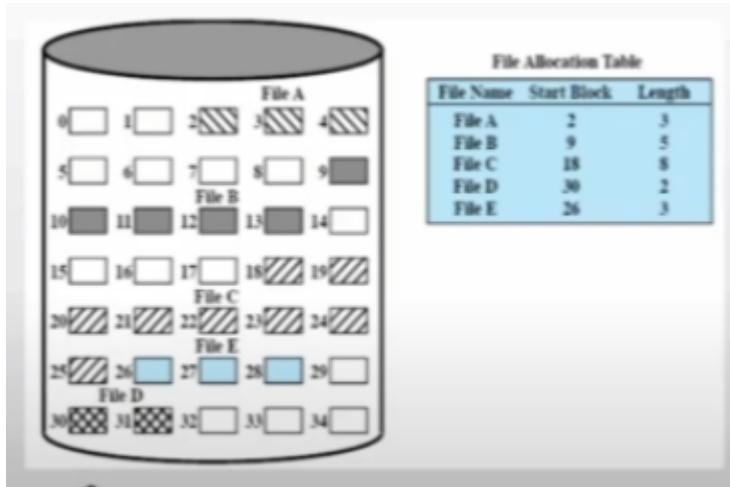
necesario pensando en que el archivo puede crecer de tamaño (puede generar **fragmentación interna** porque el espacio de sobra no puede ser utilizado por otros archivos).

Asignación dinámica:

- El espacio se pide a medida que se necesita y los bloques de datos pueden quedar no contiguos ya que no se exige continuidad al almacenar. Se tiene que llevar un registro de los bloques de datos que conforman el archivo y en qué orden fueron asignados por si se quiere acceder al archivo en forma secuencial.

Asignación continua:

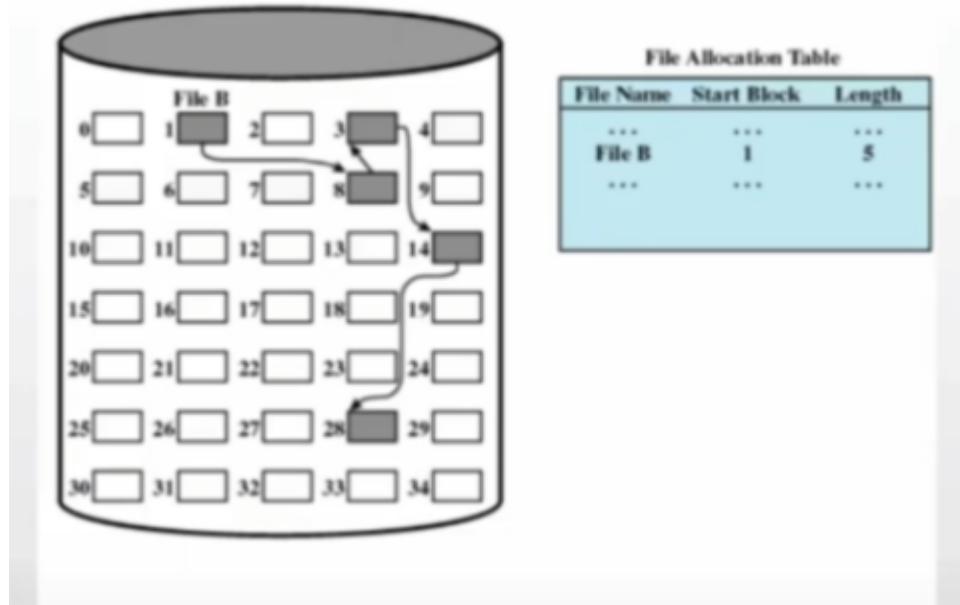
- Características:
 - Asigna en conjuntos continuos de bloques.
 - Se requiere una pre-asignación para no quedarse corto en cuanto a la cantidad de bloques asignados, aunque, normalmente se sobredimensiona el archivo.
 - Puede generar fragmentación interna.
 - Su FAT es simple ya que contiene una sola entrada con un bloque de inicio y longitud.
 - Puede leer archivos con una única operación, (más eficiente para cosas que solo requieren lectura ya que se da una secuencialidad en la información).
 - Puede generar fragmentación externa generada por la creación y borrado de archivos.
 - Utiliza la compactación para mitigar la fragmentación externa (Se necesita otro disco para poder copiar toda la información y que quede contigua para después volver a copiarla en el disco original ya compactada).



Asignación encadenada:

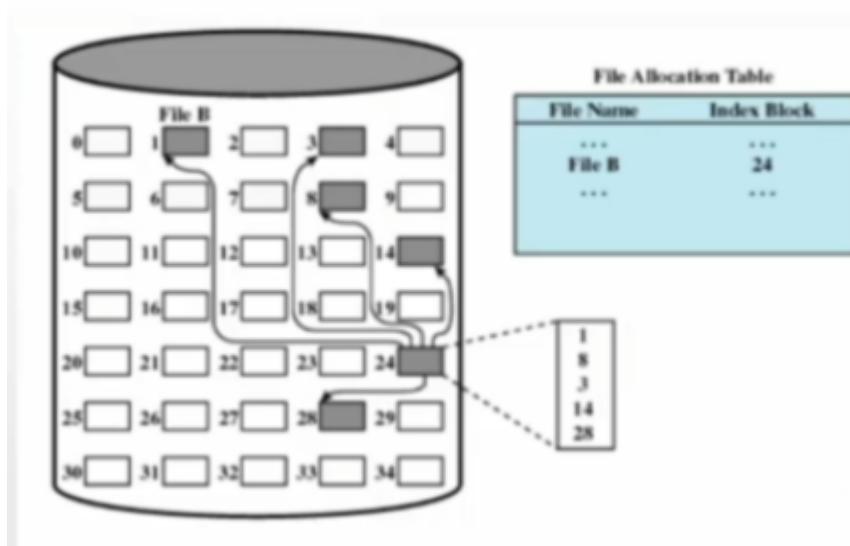
- **Características:**

- Asigna en bloques individuales, es decir, no hay exigencia de continuidad, pueden estar contiguos como no.
- **Los bloques tienen un puntero al siguiente** por lo que facilita el acceso secuencial
- Su FAT sigue siendo simple, es una sola entrada por archivo (bloque de inicio y tamaño del archivo)
- **No tiene fragmentación externa ni interna.**
- Los archivos pueden crecer bajo demanda.
- Puede consolidar los bloques de un archivo para garantizar su cercanía entre los bloques del archivo.
- Como problema tenemos que si un bloque de la cadena se daña y perdemos los últimos bytes que contienen el puntero al siguiente bloque, el archivo se ve cortado ya que no se puede avanzar en la cadena.



Asignación indexada simple:

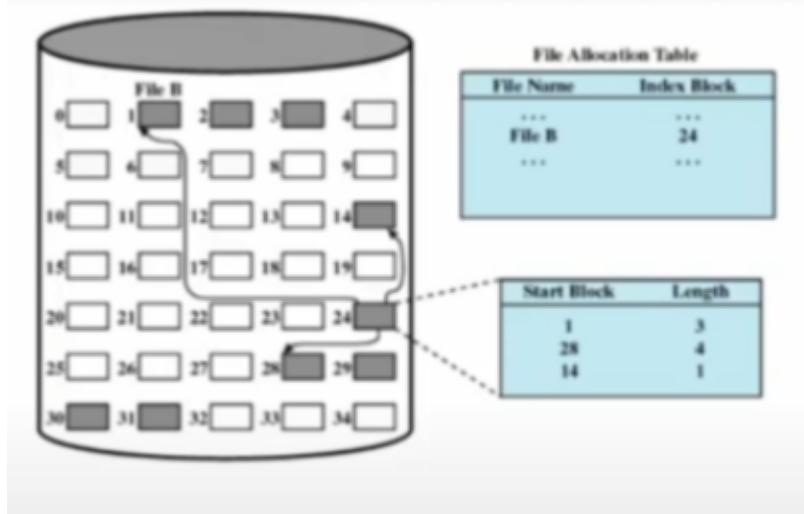
- Características:
 - Su FAT es simple, tiene una entrada con un puntero al bloque índice
 - El archivo contiene un **bloque índice** que contiene los índices a los bloques que componen al archivo de manera ordenada.
 - El **bloque índice** no almacena contenido del archivo.
 - Asigna en bloques individuales.
 - **No tiene fragmentación externa ni interna.**
 - **Facilita el acceso random/directo** a un archivo a diferencia de la encadenada.



Asignación indexada por secciones:

- Características:

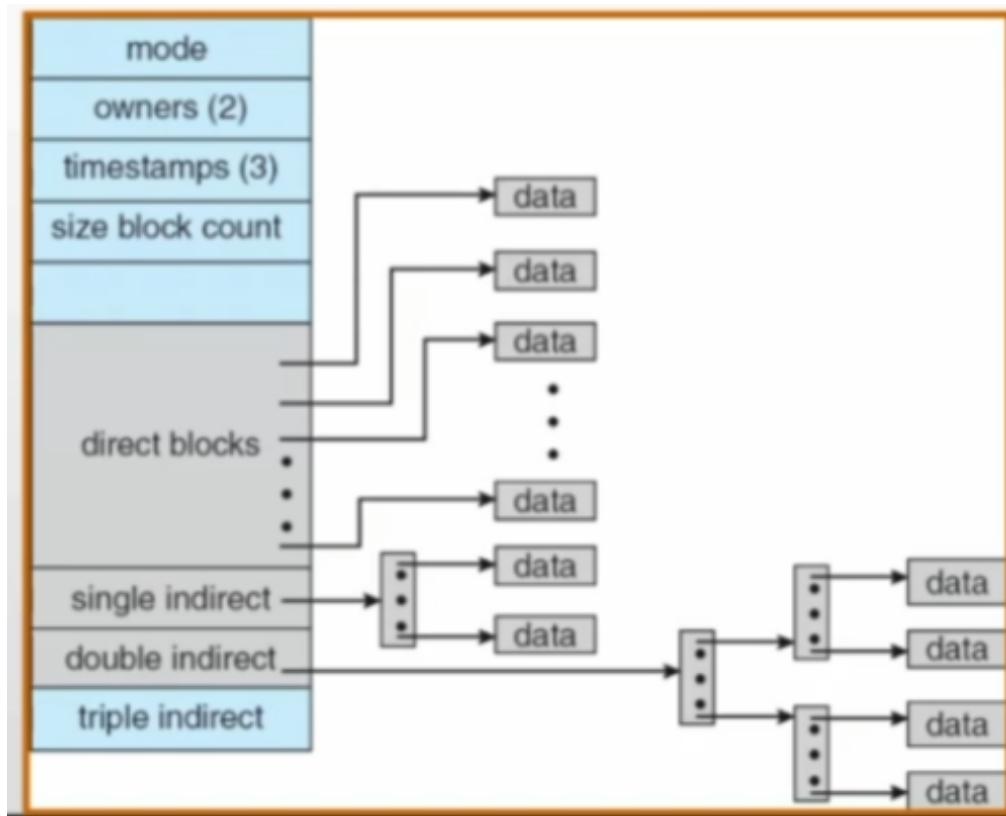
- Cada entrada del bloque índice contiene además el campo longitud.
- El índice apunta al primer bloque de un conjunto ordenado contiguamente.



Asignación indexada por niveles de indirección:

- Tiene bloques directos de datos.
- Tiene bloques índices (apuntan a bloques de datos).

- Puede tener varios niveles de indirección.
- Son los I-NODOS básicamente.
- No tiene fragmentación externa ni interna.



GESTIONES DE ESPACIO LIBRE:

Gestión de espacio libre:

- Gestiona cuáles de los bloques de disco están disponibles.
- Tiene como alternativas a la Tabla de Bits, Bloques libres encadenados e Indexación.

Tabla de bits:

- Tiene un bit por cada bloque de disco.
- 0 es libre y 1 es ocupado.
- Facilita encontrar bloques o grupos de bloques libres

- Su tamaño en memoria se calcula así: tamaño del disco en bytes / tamaño del bloque en el sistema de archivos.
- Problemático porque ocupa mucho espacio y hay que mantenerlo.

Bloques libres encadenados:

- Tiene un puntero al primer bloque libre
- Cada bloque libre tiene un puntero al siguiente bloque libre
- Es ineficiente para encontrar bloques libres (hace varias operaciones de E/S para buscar)
- Tiene problemas con la pérdida del enlace.
- Es difícil encontrar bloques libres consecutivos porque se tendría que tener la lista encadenada ordenada.

Indexación:

- El primer bloque tiene las direcciones de N bloques libres
 - Las N-1 primeras direcciones son bloques libres.
 - La N dirección es a otro bloque de N direcciones.
 - Difícil mantenimiento, si por ejemplo tenemos una tabla que tiene: 10-15 y 17-20 y se liberan del 15 al 16, no insertamos en la tabla sino que hay que rearmar para que quede 10-20.

File System 3 (Joaquinho)

UNIX - Manejo de Archivos

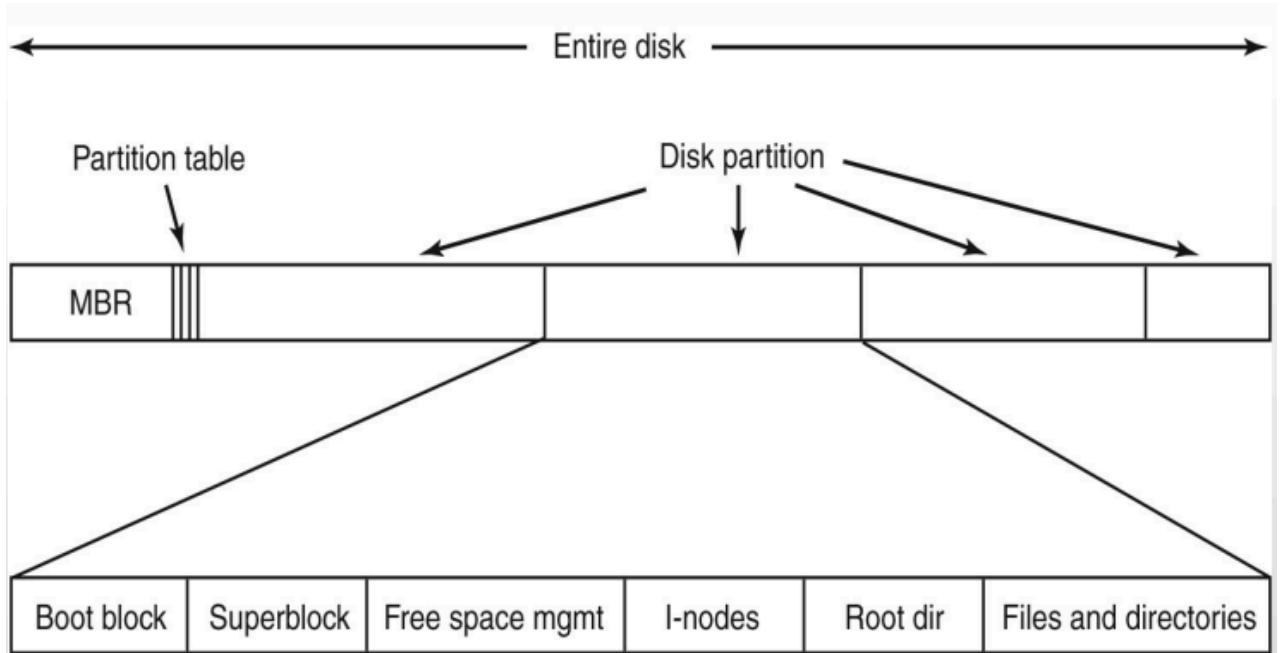
- Tipos de Archivos:
 - ❖ Archivos Comunes.
 - ❖ Directorio.
 - ❖ Archivos Especiales (dispositivos /dev/sda).

- ❖ **Named Pipes** (Comunicación entre procesos).
- ❖ **Links** (Comparten el I-Nodo, únicamente dentro del mismo File System):
- ❖ **Links Simbólicos** (Tienen I-Nodo propio, para File System diferentes).

UNIX - Estructura de Volumen/Partición

- Cada disco físico se puede dividir en uno o varios **volúmenes/particiones**. Cada una de estas particiones tiene un **File System** que cuenta con las siguientes partes:
 - ❖ **Boot Block “Bloque de Booteo”**: Código para bootear el S.O.
 - ❖ **SuperBlock**: Atributos sobre el File System, como por ejemplo Bloques/Clusters Libres.
 - ❖ **I-Node Table “Tabla de I-Nodos”**: Tabla que contiene todos los I-Nodos.
 - **I-Nodo**: Es una estructura de control que contiene la estructura de un archivo.
 - ❖ **Data Blocks “Bloques de Datos”**: Bloques de datos de los archivos.

UNIX - Estructura de Una Partición



UNIX - Información de un I-Nodo

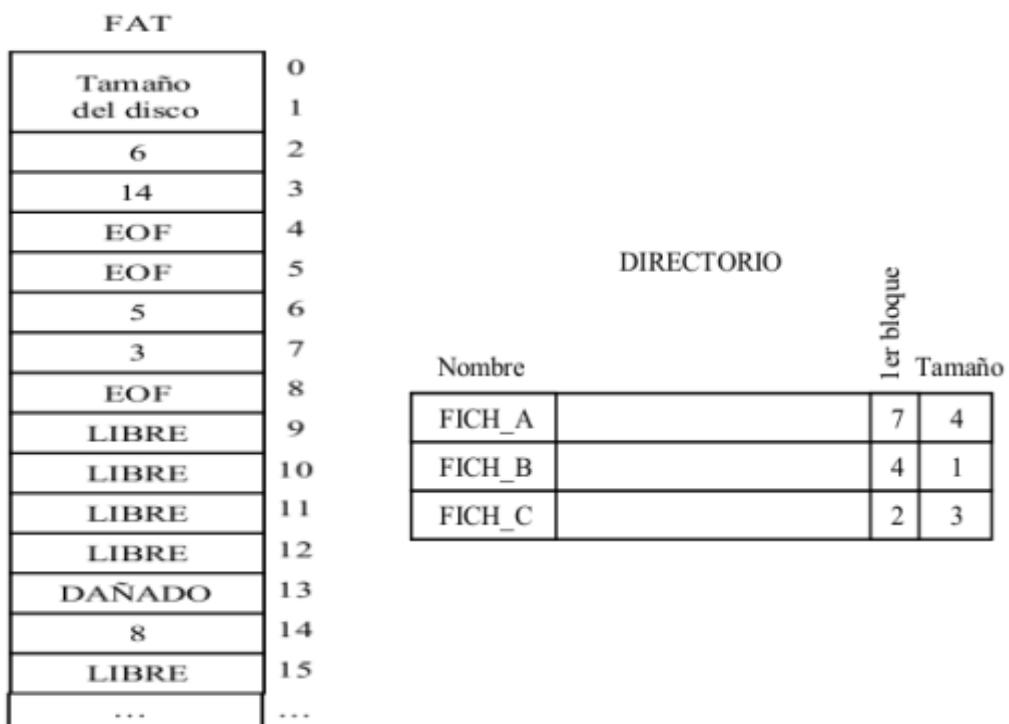
File Mode	16-bit flag that stores access and execution permissions associated with the file. 12-14 File type (regular, directory, character or block special, FIFO pipe 9-11 Execution flags 8 Owner read permission 7 Owner write permission 6 Owner execute permission 5 Group read permission 4 Group write permission 3 Group execute permission 2 Other read permission 1 Other write permission 0 Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

File Systems que soporta Windows

- CD-ROM File System (CDFS): Para CD.
- Universal Disk Format (UDF): DVD, Blu-Ray.
- File Allocation Table (FAT)
 - ❖ FAT 12: MS-DOS v3.3 a 4.0, floppy disk “diskette”.
 - ❖ FAT 16: MS-DOS 6.22, nombres cortos de archivo.
 - ❖ FAT 32: MS-DOS 7.10, nombres largos pero no soportados en MS-DOS.
- New Technology File System (NTFS).

Windows - FAT

- Utiliza un esquema de ASIGNACIÓN ENCADENADA (La FAT contiene un puntero que apunta al siguiente bloque de datos en la secuencia. Esta secuencia de bloques de datos forma una cadena que representa el archivo.).
- Bloques libres y dañados tienen códigos especiales.



Windows - FAT12

- Al utilizarse 12 bits para la identificación del sector, la misma se limita a 2¹² (4096) sectores en un volumen/partición.
- Utiliza tamaños de sector desde los 512 bytes hasta 8 KB, lo que limita a un tamaño total de volumen de 32 MB -> 2¹² * 8 KB.
- Se usaba como Sistema de archivos de los diskettes de 3,5 y 12 pulgadas que pueden almacenar hasta 1,44 MB de datos.

Windows - FAT16

- Al utilizar 16 bits para identificar cada sector puede tener hasta 2¹⁶ (65.536) sectores en un volúmen/partición.
- El tamaño de sector en FAT16 varía desde los 512 bytes hasta los 64 KB, lo que limita a un tamaño máximo de volumen de 4 GB.
- El tamaño de sector dependía del tamaño del volumen/partición al formatearlo.

Windows - FAT32

- Fue el más reciente de la línea hasta la salida del exFAT o FAT64.
- Utiliza 32 bits para la identificación de sectores, pero reserva los 4 bits superiores, con lo cual efectivamente solo se utilizaban 28 bits para la identificación.
- El tamaño de sector en FAT32 puede ser de hasta 32 KB, con lo cual tiene una capacidad teórica de direccionar volúmenes de hasta 8 TB.
- El modo de identificación y acceso de los sectores lo hace más eficiente que FAT16.

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Windows - NTFS

- Es el filesystem nativo de Windows desde Windows NT.
- Usa 64-bit para referenciar sectores, teóricamente permite tener volúmenes/particiones de hasta 16 Exabytes (16 billones de GB).

Windows - NTFS vs FAT

- Si bien FAT es simple y más rápido para ciertas operaciones, NTFS soporta:
 - ❖ Tamaños de archivo y de discos mayores.
 - ❖ Mejora la performance en discos grandes.
 - ❖ Nombres de archivos de hasta 255 caracteres.
 - ❖ Atributos de seguridad.
 - ❖ Transaccional.

I/O (Milenariha)

El SO es responsable de controlar los dispositivos de E/S:

- Generar comandos

- Manejar interrupciones
 - Manejar errores
- y proporcionar una **interfaz** para utilizarlos.

Problemas

- Heterogeneidad de dispositivos
- Características de los dispositivos
- Velocidad
- Nuevos tipos de dispositivos (poder adaptarse sin que haya cambios bruscos)
- Diferentes formas de realizar E/S

Aspectos de los dispositivos de E/S

Unidad de Transferencia

- Dispositivos por bloques (discos):
Operaciones: read, write, seek
- Dispositivos por carácter (keyboards, mouse, serial ports)
Operaciones: get, put

Formas de acceso (secuencial o aleatorio)

Tipo de acceso

- Compartido (disco rígido)
- Exclusivo (impresora)

Tipo de acceso

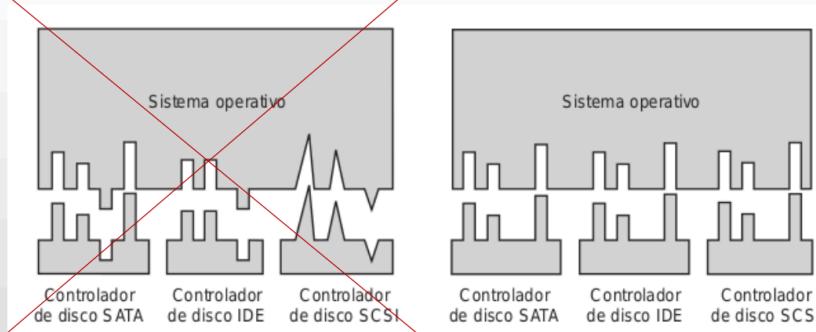
- Read only (CDROM)
- Write only (Pantalla)
- Read/Write (Disco)

Velocidad

Dadas estas características y problemas nombrados surgen **Metas, Objetivos y Servicios** que el SO deberá brindar en relación a la E/S:

- Generalidad
 - manejar todos los dispositivos de I/O de una manera **uniforme** y estandarizada en cuanto a conectores y las operaciones de los mismos.
 - Ocultar la mayoría de los **detalles** del dispositivo en las rutinas de niveles más “bajos”

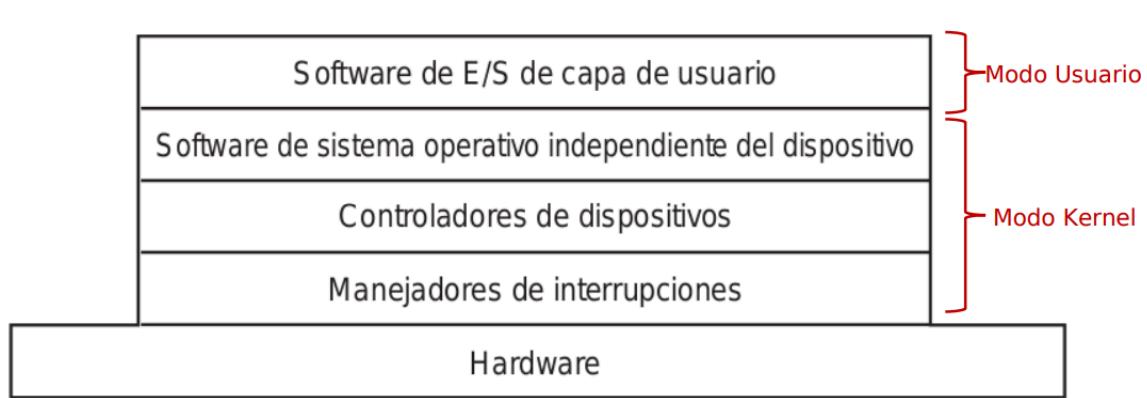
Interfaz Uniforme



- 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 proceso espere por la finalización de su I/O mientras que otro proceso se ejecuta en la CPU, generando un uso más eficiente de la misma
- Planificación
 - El SO debería ser capaz de organizar los requerimientos que hay a un dispositivo para que se utilice de manera eficiente
 - Ej: Planificar movimientos de la cabeza del disco
- Buffering - Almacenamiento de los datos en memoria mientras se transfieren
 - Los buffers solucionan los problemas de **velocidad** entre dispositivos ya que almacenan los datos para

- transferirlos a la velocidad que soporta el dispositivo
 - También actúan como reguladores de formatos y tamaño de la información 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
 - mecanismo para coordinar el acceso concurrente al dispositivo (armar cola)
 - Ej: La impresora tiene acceso exclusivo y no puede atender dos cosas a la vez
- **Reserva de dispositivos:** acceso exclusivo
- **Manejo de Errores**
 - El S.O. debe administrar errores ocurridos y tomar las medidas necesarias (S.O debe abstraer todos los detalles del error)
 - La mayoría retorna un número de error o código cuando la I/O falla.
 - Logs de errores
- **Formas de realizar la E/S**
 - **Bloqueante:** El proceso se suspende hasta que el requerimiento de I/O se completa
 - Fácil de usar y entender
 - No es suficiente bajo algunas necesidades.
 - **No Bloqueante:** El requerimiento de I/O retorna en cuanto es posible
 - Ejemplo: Interfaz de usuario que recibe input desde el teclado/mouse y se muestra screen.
 - Ejemplo: Aplicación de video que lee frames desde un archivo mientras va mostrandolos en pantalla.

Diseño (en capas mdc)



1. Software capa del usuario

- Librerías: permiten acceso a SysCalls e implementan servicios que no dependen del kernel
- Procesos de apoyo: spooling

2. Software independiente SO: brinda los principales servicios de E/S (no hace detalle fino, es **genérica** e independiente)

- Interfaz uniforme
- Manejo de errores
- Buffer
- Asignación de Recursos
- Planificación

Para realizar tantas tareas, el Kernel mantiene un montón de estructuras de datos con la información de estado de cada dispositivo o componente en pos de una mejor toma de decisiones.

3. Controladores (Drivers)

- Software que funciona como traductor de operaciones entre el SO y el dispositivo.
- Contienen las instrucciones específicas para ver cómo interactuar con el dispositivo
- Manejan un tipo dispositivo
- Traducen los requerimientos abstractos en los comandos para el dispositivo
 - Escribe sobre los registros del controlador
 - Acceso a la memoria mapeada

- Encola requerimientos
- Comúnmente las interrupciones generadas por los dispositivos son atendidas por funciones provistas por el driver
- Interfaz entre el SO y el HARD
- Forman parte del espacio de memoria del Kernel (módulos)
- Los fabricantes de HW implementan el driver en función de una API especificada por el SO
 - open(), close(), read(), write(), etc
- Para agregar nuevo HW sólo basta indicar el driver correspondiente sin necesidad de cambios en el Kernel

EJEMPLO LINUX

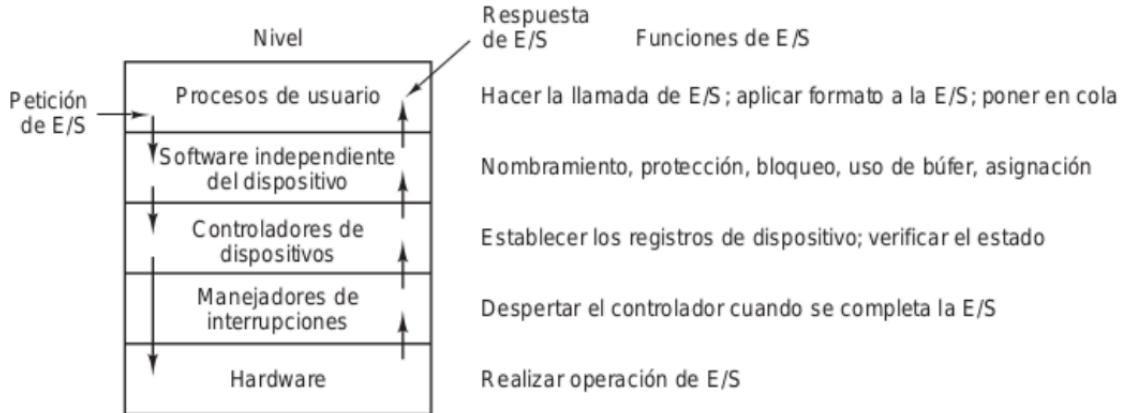
- Linux distingue 3 tipos de dispositivos (Carácter, bloque, red)
- Drivers se implementan como módulos
- Debe tener al menos estas operaciones
 - init_module: Para instalarlo
 - cleanup_module: Para desinstalarlo.
- Operaciones que debe contener para I/O
 - open
 - release
 - read
 - write
 - ioctl: orden de control sobre el dispositivo
- Por convención, los nombres de las operaciones comienzan con el nombre del dispositivo
- Acceso al hardware
- Leen o Escriben un byte en el puerto de E/S indicado

4. Gestor de interrupciones

- Atiende todas las interrupciones del HW Deriva al driver correspondiente según interrupción
- Resguarda información

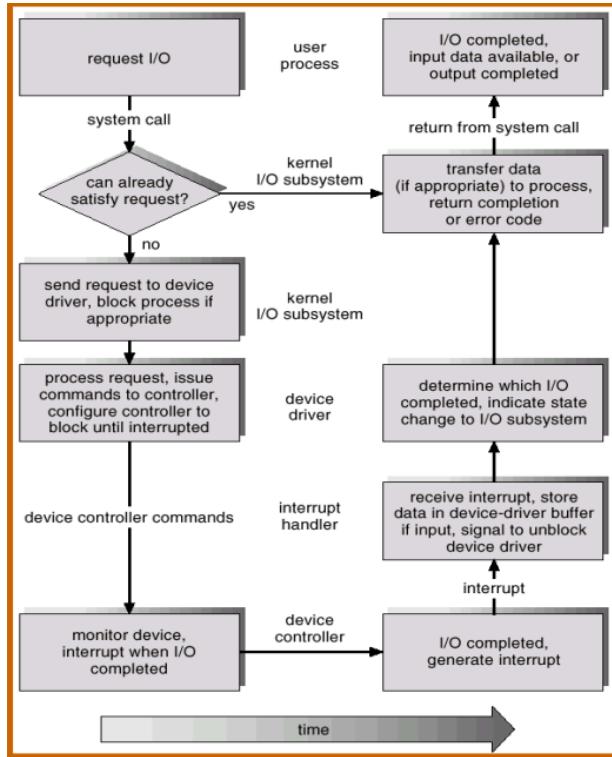
- Independiente del Driver

Ciclo de atención de un requerimiento



(paso a paso desde el requerimiento hasta el hardware)

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



Performance

I/O es uno de los factores que más afectan a la performance del sistema:

- Utiliza mucho la CPU para ejecutar los drivers y el código del subsistema de I/O
- Provoca Context switches ante las interrupciones y bloqueos de los procesos
- Utiliza el bus de memoria en copia de datos:
 - Aplicaciones (espacio usuario) – Kernel
 - Kernel (memoria física) - Controladora

Mejorar performance

- Reducir el número de context switches
- Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación
- Reducir la frecuencia de las interrupciones, utilizando:
 - Transferencias de gran cantidad de datos
 - Controladoras más inteligentes
 - Polling, si se minimiza la espera activa.
- Utilizar DMA

Anexo I/O (Felipinho)

Existen diversos dispositivos de entrada/salida, basados en la función que cumplen para:

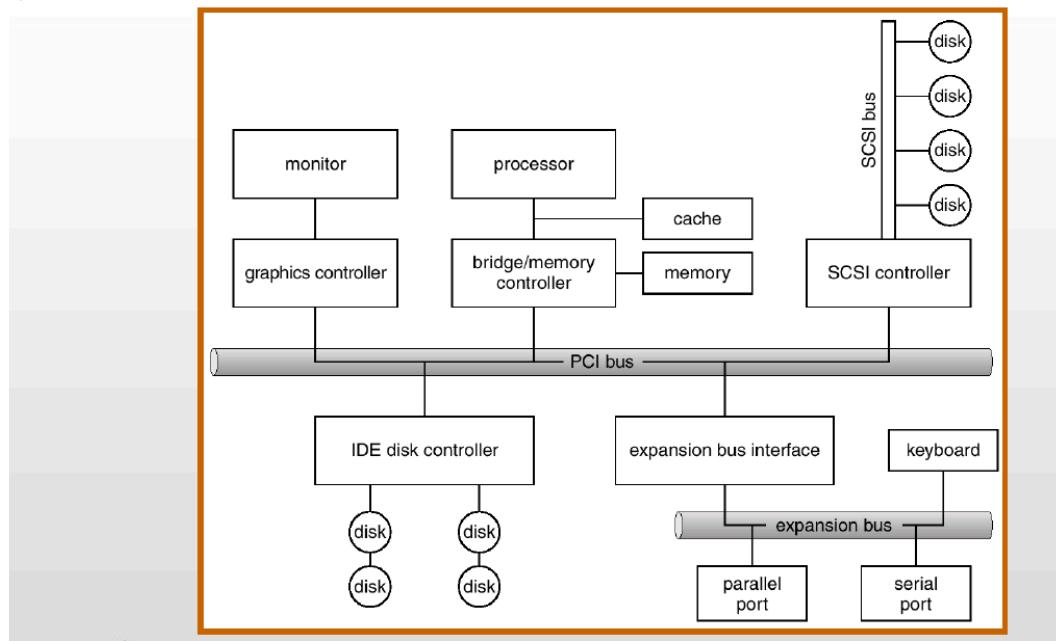
- Comunicación con el usuario: impresoras, pantalla, teclado ...
- Comunicación intra maquina (comunica procesador con elementos que componen la maquina): discos, sensores...
- Comunicación externa/remota: modems, interfaces de red...

* **Problemas que surgen:** Todos estos dispositivos de I/O manejan datos, estos datos no tienen porqué ser iguales ya sea en cantidad, formato y velocidades de transferencia, haciendo que la comunicación con elementos de mayor velocidad (procesador y ram) sea compleja en términos de bloqueo de recursos, como en la identificación de quien genera la información (YA QUE TODOS LOS MÓDULOS DE I/O COMPARTEN LOS BUSES).

Hardware y software involucrados

- Buses
- Controladores
- Dispositivos
- Puertos de E/S – Registros
- Drivers
- Comunicación con controlador del dispositivo: I/O Programada, Interrupciones, DMA

Estructura de Bus de una PC



Comunicación: CPU - Controladora

- ¿Cómo puede la CPU ejecutar comandos o enviar/recibir datos de una controladora de un dispositivo?

La controladora tiene uno o más registros:

- Registros para señales de control
- Registros para datos

La CPU se comunica con la controladora escribiendo y leyendo en dichos registros.

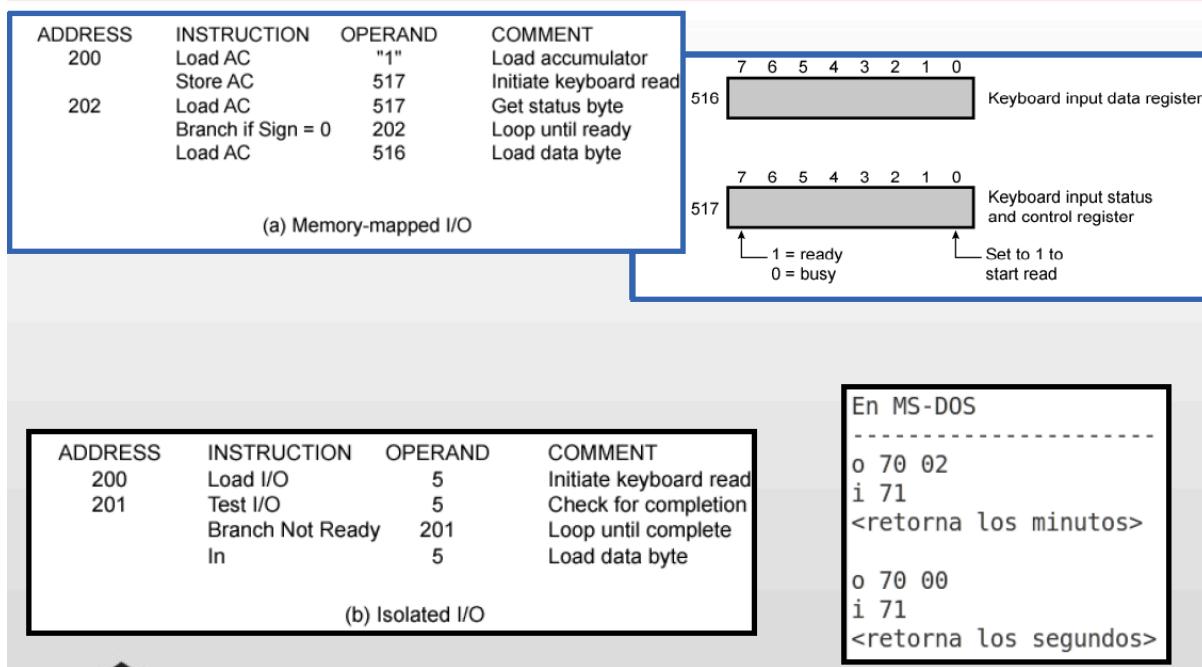
Comandos de I/O

- CPU emite direcciones para identificar el dispositivo que genera la información/ hacia donde enviamos info.
- CPU emite comandos
 - Control - Qué hacer? Ej. Girar el disco
 - Test - Controlar el estado Ej. power? ¿Error?
 - Read/Write - Transferir información desde/hacia el dispositivo

Mapeo de E/S y E/S aislada

- Correspondencia en memoria (Memory mapped I/O)
 - Dispositivos y memoria comparten el espacio de direcciones.
 - I/O es como escribir/leer en la memoria.
No hay instrucciones especiales para I/O Ya que se dispone de muchas instrucciones para la memoria
- Isolated I/O (Aislada, uso de Puertos de E/S)
 - Espacio separado de direcciones
 - Se necesitan líneas de I/O. Puertos de E/S
 - Posee un conjunto limitado de instrucciones especiales

Memory Mapped and Isolated I/O



Técnicas de I/O

Estas técnicas tienen como fin realizar una correcta vinculación con los módulos de E/S, dividiéndose en I/O... :

- **Programada**

La CPU tiene control directo sobre la I/O, posibilitando :

- Controlar el estado del módulo
- Utilizar Comandos para leer y escribir datos

- Transferir los datos

La CPU espera que el componente de I/O complete la operación, generando durante el tiempo que se realiza la tarea que se desperdicie ciclos de CPU (ineficiente).

- **Polling**

Es utilizada para verificar repetidamente de manera activa el estado de un dispositivo, para determinar si está listo para ser leído o escrito. En lugar de esperar pasivamente una interrupción o evento que notifique el cambio de estado, el sistema realiza consultas periódicas para verificar si ha habido algún cambio o si se ha completado alguna operación.

- Este enfoque puede ser útil en situaciones donde no se dispone de interrupciones hardware o donde el sistema no puede permitirse el costo de esperar pasivamente. Sin embargo, el polling continuo puede ser ineficiente en términos de uso de CPU, especialmente si se realizan consultas frecuentes y el dispositivo rara vez cambia de estado.

- **Manejada por Interrupciones**

Este enfoque soluciona el problema de la espera de la CPU.

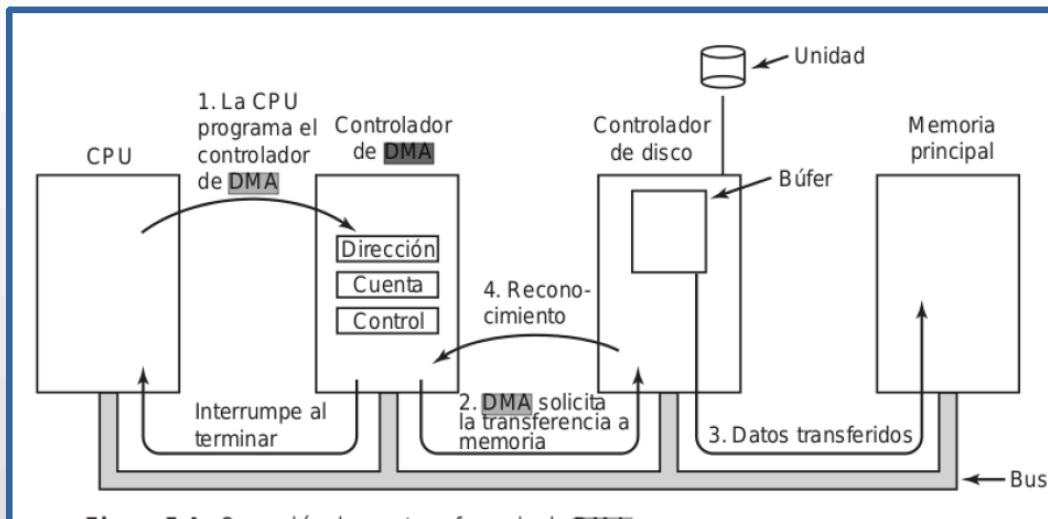
La CPU no repite el chequeo sobre el dispositivo permitiendo que el procesador continúe la ejecución de instrucciones hasta que una interrupción (como "select" o "epoll" en sistemas tipo UNIX) informa el cambio de estado de un dispositivo o módulo, en lugar de requerir una verificación constante del estado del mismo.

- **DMA (Direct Memory Access)**

Controla el intercambio de datos entre la memoria principal y el dispositivo

El procesador es interrumpido luego de que el
El bloque entero fue transferido.

Pasos para una transferencia DMA

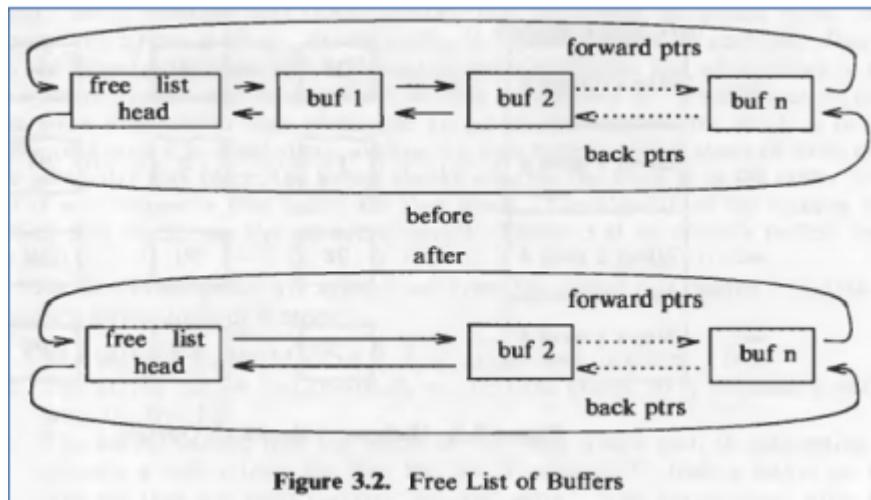


Cache (Coco)

- Disk Cache
 - ❖ Buffers en memoria RAM para almacenamiento temporal de bloques de disco.
 - ❖ Objetivo minimizar la frecuencia de acceso al disco.
- Observaciones
 - ❖ Cuando un proceso quiere acceder a un bloque de la caché hay dos alternativas:
 - ➔ Se copia el bloque al espacio de direcciones del usuario no permite compartir el bloque.
 - ➔ Se trabaja el bloque como memoria compartida permite acceso a varios procesos
 - Dicha área compartida debe ser limitada, con lo cual debe existir un algoritmo de reemplazo.

- Estrategia de reemplazo
 - ❖ Cuando se necesita un buffer para cargar un nuevo bloque, se elige el bloque que hace más tiempo no es referenciado.
 - ❖ Consiste en una lista de bloques donde el último es el más recientemente usado (LRU).
 - ❖ Cuando un bloque es referenciado o entra en la caché se queda al final de la lista.
 - ❖ No se mueven los bloques en la memoria: se asocian punteros.
 - ❖ Otra alternativa: LFU, se reemplaza el que tenga menor número de referencias.
- Objetivo y estructura de un Buffer Cache
 - ❖ Minimizar la frecuencia de acceso a disco.
 - ❖ Estructura que se forma por buffers.
 - ❖ El kernel asigna un espacio en la memoria durante la inicialización de dicha estructura.
 - ❖ El buffer se compone de dos partes:
 - Header: contiene información del bloque, número del bloque, estado, relación con otros buffers, etc.
 - Buffer en sí: el lugar donde se almacena el bloque de disco traído a memoria.
- Header
 - ❖ Nro de dispositivo, nro de bloque.
 - ❖ Estado.
 - ❖ Punteros:
 - 2 punteros de hash queue.
 - 2 punteros para la free list.
 - 1 puntero al bloque en memoria (donde se ubica).
- Estados de los buffers
 - ❖ Free (disponible).

- ❖ Busy (no disponible, en uso por algún proceso).
 - ❖ Escribiendo o leyendo del disco.
 - ❖ Delayed Write(DW): buffers que fueron modificados en memoria, pero los cambios no se han reflejado en el bloque original de disco.
- Free list
 - ❖ Organiza los buffers disponibles para ser utilizados para cargar nuevos bloques de disco.
 - ❖ No necesariamente dichos buffers disponibles están vacíos (el proceso puede que haya terminado y liberado el bloque, pero el buffer puede seguir en estado DW).
 - ❖ Es ordenada según LRU.



Hash queues

- ❖ Colas que buscan optimizar la búsqueda de un buffer en particular.
- ❖ Los headers de los buffers se organizan según una función de Hash (Dispositivo, #bloque).

- ❖ A esto se le aplica una función de hash que permite agrupar los buffers cuyo resultado dio igual. Esto permite la búsqueda eficiente.
- ❖ La agrupación y enganche de cada buffer con otro en cada cola se realiza mediante los punteros que se almacenan en el header

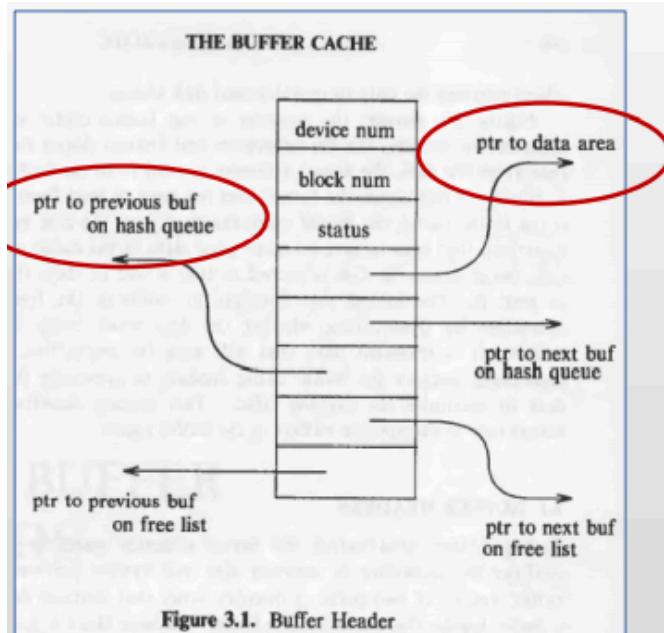


Figure 3.1. Buffer Header

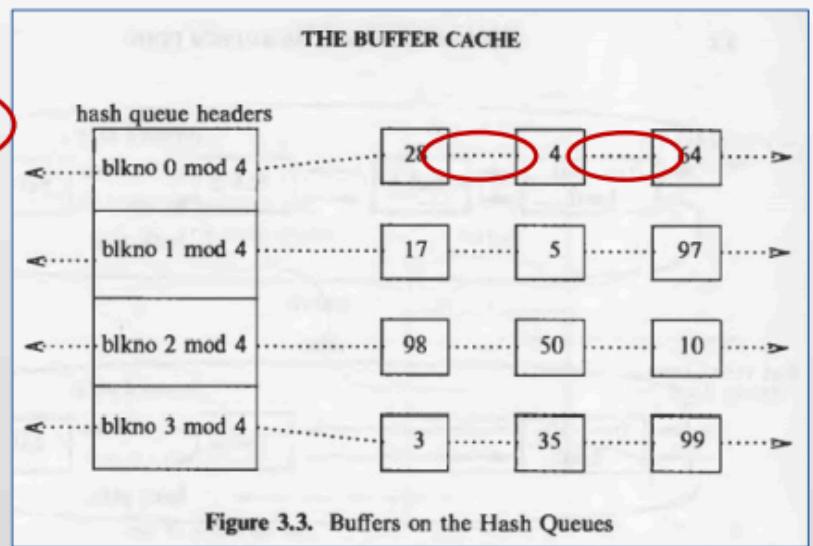


Figure 3.3. Buffers on the Hash Queues

- *Free list (continuación)*

- ❖ Sigue el mismo esquema de la Hash queue pero contiene los headers de los buffers de aquellos procesos que ya han terminado.
- ❖ El header de un buffer siempre está en la Hash Queue.
- ❖ Si el proceso que lo referenciaba terminó, va a estar en la Hash Queue y en la Free list.

- *Funcionamiento del buffer cache*

- ❖ Cuando un proceso requiere el acceso a un archivo, utiliza su inodo para localizar los bloques de datos donde se halla el mismo.

- ❖ El requerimiento llega al buffer cache quien evalúa si puede satisfacer el requerimiento o si debe realizar la E/S.
- ❖ Se pueden dar 5 escenarios:
 1. El kernel encuentra el bloque en la hash queue y el buffer está libre (en la free list).
 2. El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
 3. Idem 2, pero el bloque libre está marcado como DW.
 4. El kernel no encuentra el bloque en la hash queue y la free list está vacía.
 5. El kernel encuentra el bloque en la hash queue pero está BUSY

- Algoritmo de asignación

- ❖ Escenarios:
 1. El kernel encuentra el bloque en la hash queue y el buffer está libre.
 2. El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
 3. Idem 2, pero el bloque libre está marcado como DW.
 4. El kernel no encuentra el bloque en la hash queue y la free list está vacía.
 5. El kernel encuentra el bloque en la hash queue pero está BUSY.

Explicación correcta de que pasa en cada escenario:

1. El kernel lo busca en el hash queue, está en la misma, se busca en la free list, está entonces se puede utilizar, pero hay que reacomodar los punteros de la free list para que el anterior apunte al siguiente del removido.
2. El kernel no tiene cargado el buffer en la hash queue así que se reutiliza uno de los header que está en la free list

(sin Delayed Write), y lo agrega a la cola correspondiente al final.

3. Idem que el 2, pero cada uno de los buffers que se vayan encontrando como DW se ponen en modo writing (escribiendo al disco), y cuando terminan se agregan al inicio de la free list en vez de al final como los que se liberan normalmente.
4. El proceso queda bloqueado ya que no hay ningún espacio libre, necesita esperar que se libere un buffer para así reemplazarlo.
5. El proceso queda bloqueado ya que tiene que esperar que termine de usarse.

Preguntas de Repaso para el Parcial

Memoria

1. En paginación, si disminuyo la cantidad de bits del desplazamiento de una dirección de memoria, los frames serán más chicos.
 - Verdadero.
2. La cantidad máxima de páginas en memoria depende sólo del tamaño del proceso.
 - Falso, depende de la cantidad máxima de frames en la RAM.
3. Analice tamaños de página y page fault.
 - Tamaños de Páginas:
 - ❖ Si es chica, hay menos fragmentación interna, más páginas por proceso y más páginas en memoria principal al mismo tiempo.

- ❖ Si es grande, hay más fragmentación interna, menos páginas por proceso y menos páginas en memoria principal al mismo tiempo.
 - Page Fault:
 - ❖ Ocurre cuando la dirección que el proceso quiere acceder no está en la memoria principal. El Hardware es el encargado de detectarlo y el Sistema Operativo es el encargado de solucionarlo. Si son excesivos, reducen la performance del sistema.

Cuando el Hardware detecta un **page fault**, lanza un trap para avisarle al Sistema Operativo, este bloquea al proceso y busca un marco libre para poder realizar una operación de E/S y copiar la página que el proceso está necesitando desde memoria secundaria, una vez que termina esta operación, lanza una interrupción y se actualiza la tabla de páginas con el bit V en 1 y además, se cambia la dirección base del frame donde quedó la página. Una vez terminado este procedimiento, el proceso conflictivo vuelve a estar en Ready y cuando pase a Running, lo hará desde la instrucción que generó el **page fault**.
4. Relación de tamaño de página, de proceso, de tabla de páginas según la arquitectura de la dirección.
 5. La tabla invertida proporciona acceso directo al marco buscado.
 - Verdadero, a partir del número de página que se convierte en una clave hasheada, podemos encontrar el marco correspondiente.
 6. Qué consecuencias puede tener la hiperpaginación.

- Hay una baja importante de performance en el sistema, ya que, la CPU pasa más tiempo resolviendo page fault que ejecutando procesos.

7. En la técnica del Conjunto Trabajo ¿Qué pasa si el delta elegido es muy chico? ¿Y si es muy grande?

- Δ chico: no cubrirá la localidad (podes no tomar todas las páginas que el proceso está utilizando actualmente y por ende generar más page faults porque el proceso no tiene todas las páginas que está utilizando).
- Δ grande: puede tomar varias localidades (podes tomar páginas que el proceso ya no utiliza más o por lo menos no está utilizando en ese instante de tiempo).

8. Diferencia entre reemplazo de páginas global y local.

- Global:

- Puede reemplazar páginas de cualquier proceso.
- El SO no controla la tasa de PF por proceso.
- Los procesos pueden cambiar la cantidad total de marcos.
- Procesos de alta prioridad pasan por arriba a los de baja.

Local:

- Un proceso solo puede reemplazar páginas con su conjunto residente.
- No cambia la cantidad de frames asignados.
- Permite contabilizar la tasa de PF por proceso.
- Un proceso puede tener frames asignados sin usar.

9. Con el reemplazo local no cambia la cantidad de frames asignados al proceso.

- Verdadero.

10. Diferencia entre asignación equitativa y proporcional.

- **Asignación equitativa:** Ejemplo: si tengo 100 frames y 5 procesos, 20 frames para cada proceso.
Asignación Proporcional: Se asigna acorde al tamaño del proceso.

11. Secuencia de resolución de un page fault (incluyendo TLB)

- Se busca la página en la TLB, como la página no está allí se va a buscar a la tabla de páginas, al tampoco estar allí, el Hardware detecta un page fault, lanza un trap para avisarle al Sistema Operativo, este bloquea al proceso y busca un marco libre para poder realizar una operación de E/S y copiar la página que el proceso está necesitando desde memoria secundaria, una vez que termina esta operación, lanza una interrupción y se actualiza la tabla de páginas con el bit V en 1 y además, se cambia la dirección base del frame donde quedó la página y se agrega la misma a la TLB. Una vez terminado este procedimiento, el proceso conflictivo vuelve a estar en Ready y cuando pase a Running, lo hará desde la instrucción que generó el page fault.

12. En cuanto a los estados del bit M y R: ¿Cuál sería la página ideal para elegir como página víctima?

- La ideal sería la página con bit M = 0 y el bit R = 0 ya que la página no habrá sido referenciada recientemente y además no está modificada, por lo tanto, no hay que descargarla en memoria secundaria para que las modificaciones en la misma se vean aplicados.
- 1) Luego de cargar una página a memoria por un fallo de página el responsable de poner el bit de validez en 1 es:
- El Kernel.

- 2) Dado un SO que se encuentra con hiperpaginación (trashing), aumentar el grado de multiprogramación ayudará a que el problema se resuelva.
- Falso.
- 3) Cuál de las siguientes técnicas de reemplazo de páginas favorece al control de tasa de fallo de páginas de un proceso
- Reemplazo Local.
- 4) En la técnica de administración de páginas por el método de tabla invertida solo son cargadas en dicha tabla la información de las páginas que se encuentran cargadas en memoria principal.
- Verdadero.
- 5) En la técnica de particiones fijas, la partición donde se cargará el espacio de direcciones de un proceso lo determina:
- El Kernel.
- 6) En la técnica de administración de memoria con paginación donde se utiliza una Tabla de Páginas Invertida existe:
- Una tabla invertida para todas las páginas que se encuentran cargadas en memoria principal.
- 7) Para una gestión eficiente del área de intercambio (Swap Area) indique cual/cuales de las siguientes opciones deberán ser guardadas en la misma:
- Páginas de datos (variables) y páginas de los stacks.

8) ¿Cuál/Cuáles de la siguientes opciones es correcta acerca de las tablas de páginas multinivel?

- Tiene como objetivo tener múltiples tablas de páginas pero de menor tamaño.
- La resolución de una dirección, podría causar varios accesos a la memoria.
- Busca que la tabla de páginas ocupe menor cantidad de memoria RAM cuando corresponda.
- La tabla de páginas puede no residir completa en memoria y se sea cargada bajo demanda

9) ¿Cuál/Cuáles de las siguientes opciones es correcta acerca del Translation Lookaside Buffer (TLB)?

- Si todas las entradas buscadas generan un TLB MISS, su uso perjudica al tiempo de resolución.
- Es una memoria caché que contiene las entradas de la tabla de páginas usadas recientemente.
- Los cambios de contexto invalidan la TLB.

10) ¿Cuál/Cuáles de las siguientes características de la técnica de Segmentación Paginada es correcta?

- Elimina el problema de la fragmentación externa introducido por la segmentación pura.
- Las estructuras de datos necesarias en memoria para llevar adelante la técnica son mayores que en la segmentación o paginación pura.
- La técnica garantiza la facilidad de implementar compartición.

11) En el modelo del Working Set la elección de un Δ (delta) demasiado grande en un momento dado causará que los procesos no cuenten con todas las páginas necesarias.

- Falso.

12) ¿Cuál/Cuáles de las siguientes opciones es correcta acerca de la técnica PFF (Page Fault Frequency)?

- Ajusta sus parámetros en base al comportamiento de cada proceso.
- Se utiliza para prevenir la hiperpaginación.
- Su costo de implementación es sencillo y no requiere cálculos previos para determinar la necesidad de frames de un proceso.

13) El concepto de “Memoria Compartida” es posible implementarlo con la técnica de Segmentación pero no con la técnica de Paginación.

- Falso.
- 14) En la técnica de tabla invertida para la administración de memoria por paginación, si la función de hash utilizada no provee una buena dispersión, entonces la resolución de una dirección podría requerir más de un acceso a la memoria.
- Verdadero.
- 15) Si un proceso quiere modificar datos en un área compartida con otro proceso, puede hacerlo directamente si el otro proceso sólo accede en modo lectura
- Falso.
- 16) Las estructuras que mantienen la información de la ubicación de los sectores de datos de los archivos son creadas cada vez que el SO comienza su ejecución.
- Falso, pensar por ejemplo en el File System.

Archivos

13. En qué momento se hace el chequeo sobre si el usuario puede acceder a un archivo: en el open? en cada read? en cada write?
- El chequeo se realiza en el momento del open.
14. En Unix System V: ¿puede modificarse el i-nodo del archivo sin modificar el contenido del archivo en sí?
- Si, por ejemplo yo podría modificar los permisos del archivo sin haber modificado el contenido del mismo, haciendo que el i-nodo se vea modificado pero el contenido del archivo en sí, no.
15. En Unix System V: ¿puede modificarse el contenido del archivo sin modificar su i-nodo?
- No, el contenido del archivo si es modificado, también se verá modificado el i-nodo (File size, File addresses, Last accessed y Last modified).
16. En Unix System V se verán beneficiados en performance los archivos cuyo contenido pueda ser referenciado por las 10 primeras direcciones de bloque que están en su i-nodo.
- Verdadero. Cuando los datos del archivo se encuentran en los primeros 10 bloques de datos que pueden ser referenciados directamente desde el i-nodo, el sistema operativo puede acceder a esos datos con un mínimo de indirección, lo que resulta en un mejor rendimiento de lectura y escritura.
17. En Unix System V, el acceso random a un archivo puede realizarse accediendo directamente al bloque que necesito, sin leer los precedentes.

- Verdadero.

18. En Unix System V, ¿Puede asignarse un bloque a un archivo sin acceder previamente al superblock?

-> No.

19. En Unix System V, Se puede acceder a un archivo sin acceder a su i-nodo.

-> Falso. "el inodo es la clave de todo"

20. En Unix System V, al crear un archivo en un filesystem, indique qué se modifica: -directorio al que pertenece -superblock -tabla de i-nodos.

-> Todo.

directorio al que pertenece : modifica el path

superblock : la cantidad de bloques libres y bloques ocupados ya que almacena esos tipos de datos del filesystem. 😊<3

tabla de i-nodos : se modifica el contador del inodo que está en la tabla (cambia el contenido del inodo)

21. En Unix System V, puedo crear un archivo en un filesystem no montado?

-> No. se debe poder llegar por el path

22. Todos los filesystems de un disco deben tener el mismo tamaño de bloque.

-> VERDAD. Puede cambiar el cluster(**Conjunto de bloques consecutivos**) entre filesystems, cluster != de bloque de disco.

23. Cuando un archivo se borra, se ponen en cero los bloques?

-> No necesariamente. DIJO Q NO LE DEMOS BOLA

24. La estructura del filesystem define el tamaño máximo del archivo.

-> Verdadero. Los archivos son una construcción lógica que representa los datos en disco, depende de las estructuras de datos usadas en el filesystem.

25. La estructura del filesystem define la longitud máxima del nombre.

-> Zi.

1) Para cada una de las siguientes técnicas de administración de espacio de los archivos indique que tipo de fragmentación pueden causar:

Continua: EXTERNA

Enlazada: NI EXTERNA NI INTERNA

Indexada Simple: NI EXTERNA NI INTERNA

Indexada con niveles de indirección: NI EXTERNA NI
INTERNA

2) En la técnica de administración de archivo indexada, el tamaño máximo de un archivo lo determina:

- La cantidad de índices definidos en la estructura de datos utilizada.

3) Indique cual/cuales de las siguientes afirmaciones son correctas respecto al sistema de archivos de Unix System V visto en la teoría:

- La cantidad de índices de sectores/clusters que se pueden utilizar por archivos es limitada.

4) Para los medios de almacenamiento que son solo lectura, como un CD-ROM, la mejor técnica para manejar el espacio de datos de los archivos es:

- Continua, lee todo de una.

5) En Unix System V, si se produce una modificación en el nombre de un archivo:

- a) Cambia el i-nodo del archivo
- b) Cambia el i-nodo del directorio en el que esta el archivo
- c) Cambia el contenido del archivo
- d) Cambia el contenido del directorio en el que esta el archivo

Entrada/Salida

Tener en cuenta que todo es un archivo, como se vincula la cpu con los módulos etc.

1. En la I/O bloqueante el proceso se suspende hasta que el requerimiento de I/O se complete.

- Verdadero.

2. Complete la siguiente oración relacionada a E/S:

- En el diseño de la E/S en un SO se busca manejar los diferentes dispositivos de E/S de una manera **UNIFORME**.

Para ello se definen un conjunto de funciones **COMUNES** para cada tipo de dispositivo que pueda administrarse.

3. Entre los servicios que se brindan en el diseño de la E/S de un SO se encuentran la planificación de requerimientos y buffering de datos. En cambio, el manejo de errores de los dispositivos es delegado a los procesos de usuario que quieran utilizar dichos dispositivos.
- Falso.
4. En el diseño de Entrada/Salida el código de los Drivers correspondientes a cada tipo de dispositivo se ejecutan en modo Kernel
- Verdadero.
5. Los drivers de un dispositivo se ejecutan en modo kernel, aún cuando los mismos fueron diseñados por personas que no participan en el desarrollo del kernel de SO.
- Verdadero.
6. En Unix System V, cuando a un archivo se le modifican los permisos se modifica:
- El i-nodo del archivo.
7. Dados los siguientes servicios a implementar en el diseño del sub-sistema de E/S en un SO , seleccione la finalidad de los siguientes:
- Spooling: Mantener una cola de requerimientos a dispositivos de acceso exclusivo
 - Buffering: Tratar con problemas de tamaños o formatos de los datos de los dispositivos
 - Caching: Mantener copia de los datos en memoria

Buffer Cache

26. En la estructura de Buffer cache vista, un buffer puede estar ocupado y delayed write a la vez.

-> VERDADERO. La combinación de "ocupado" y "delayed write" puede ocurrir cuando el buffer cache está siendo utilizado activamente para almacenar datos, pero el sistema operativo ha decidido retrasar la escritura real en el dispositivo de almacenamiento.

27. El i-nodo de un archivo que se está usando debe estar en algún buffer del buffer cache.

-> No, no suelen mezclarse.

En general el buffer caché se usa para bloques de datos no bloques de control, aunque los i-nodos si están en bloque y por tanto podrían traerse al mismo.

28. Un buffer delayed write puede volver a estar ocupado, si lo pide un proceso, pero antes debe grabarse a disco. (importa el bloque en el buffer).

-> No necesariamente necesita escribirlo a disco, si se lo puede asignar directamente.

29. Para asignar un bloque a un archivo es necesario contar con el superblock en el buffer cache.

-> FALSO Guarda bloques, no superblock. No conviene combinar estas estructuras de gestión.

30. Un proceso esperando por un buffer delayed write y ocupado, deberá esperar la escritura a disco antes de que se le asigne a él. (importa el bloque en el buffer).

-> Falso.

31. Las hash queues sirven para buscar por un bloque o buffer en particular .

-> Solo para buscar por bloque. Buscan por bloque, se aplica el hash al número de bloque, no al número de buffer.

32. La free list sirve para buscar por cualquier buffer.

-> No, porque contiene los headers de los buffers de aquellos procesos que ya han terminado (Buffers libres).

33. No puede haber más de un proceso esperando por un buffer.

-> Falso (no tiene por que haber un limite).

34. Cuando un proceso libera un delayed write, es escrito a disco antes de ponerlo en la free list.

-> No necesariamente tiene porque escribirlo, al ser diferida la escritura puede escribirlo cuando lo vea necesario, no depende del proceso.

35. ¿Puede un buffer en la free list, estar ocupado?

-> No. Si está en la free list esta disponible (puede no estar vacio de contenido)

36. Si un buffer está primero en la free list y en ese momento lo pide un proceso: donde va cuando se libere?

-> Al final de la free list, (LRU).

37. Si un proceso necesita un buffer y la free list está vacía, el proceso se aborta.

- No, espera bloqueado.

38. Si una hash queue está vacía, se toma un buffer de otra cola.

- FALSO, puede que una hash queue esté vacía, si no esta el buffer hay que cargarlo y se busca un buffer de la free list (no en cualquier otra cola o una cola de otra hash queue).

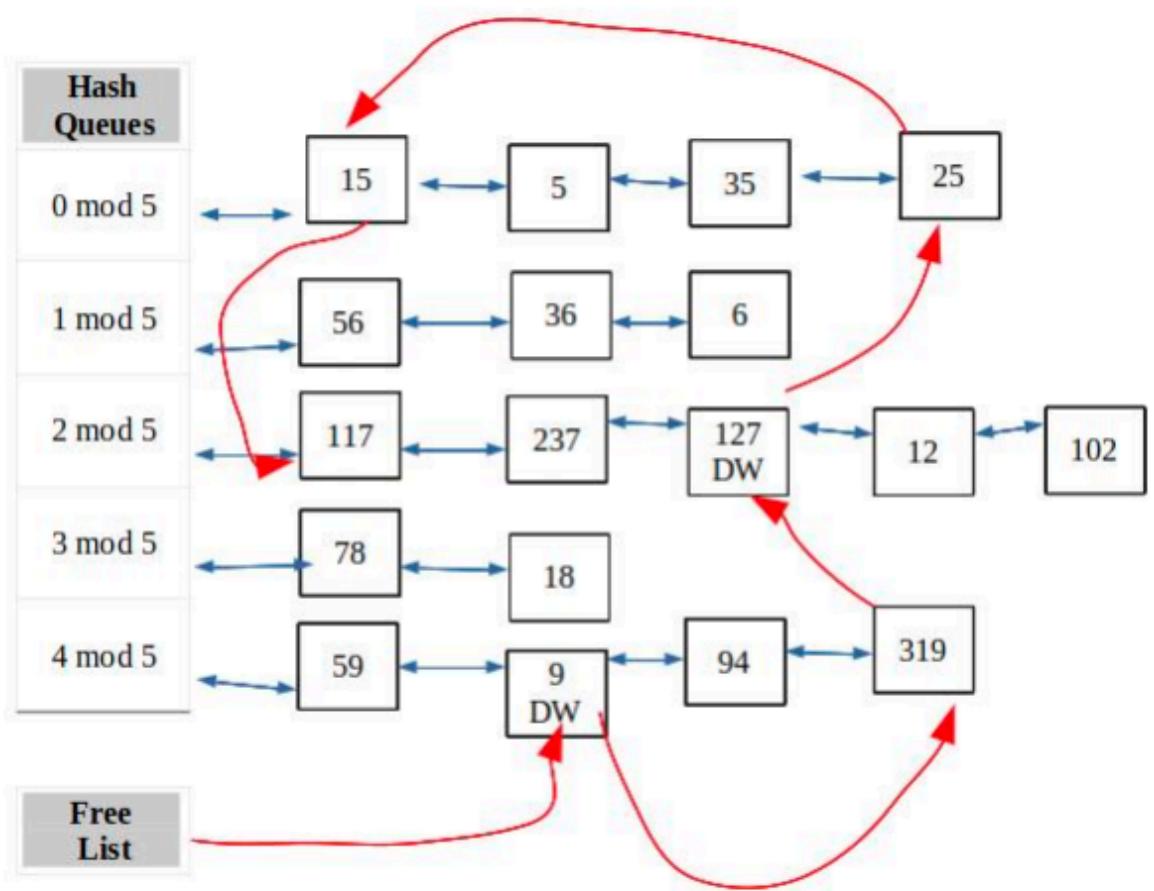
39. Para acceder a la tabla de páginas, ésta debe estar completa en el buffer cache.

- Falso. Es una estructura de control, no debería combinarse con datos, además la tabla de páginas está en RAM, no es algo de bloques de disco como el buffer cache.

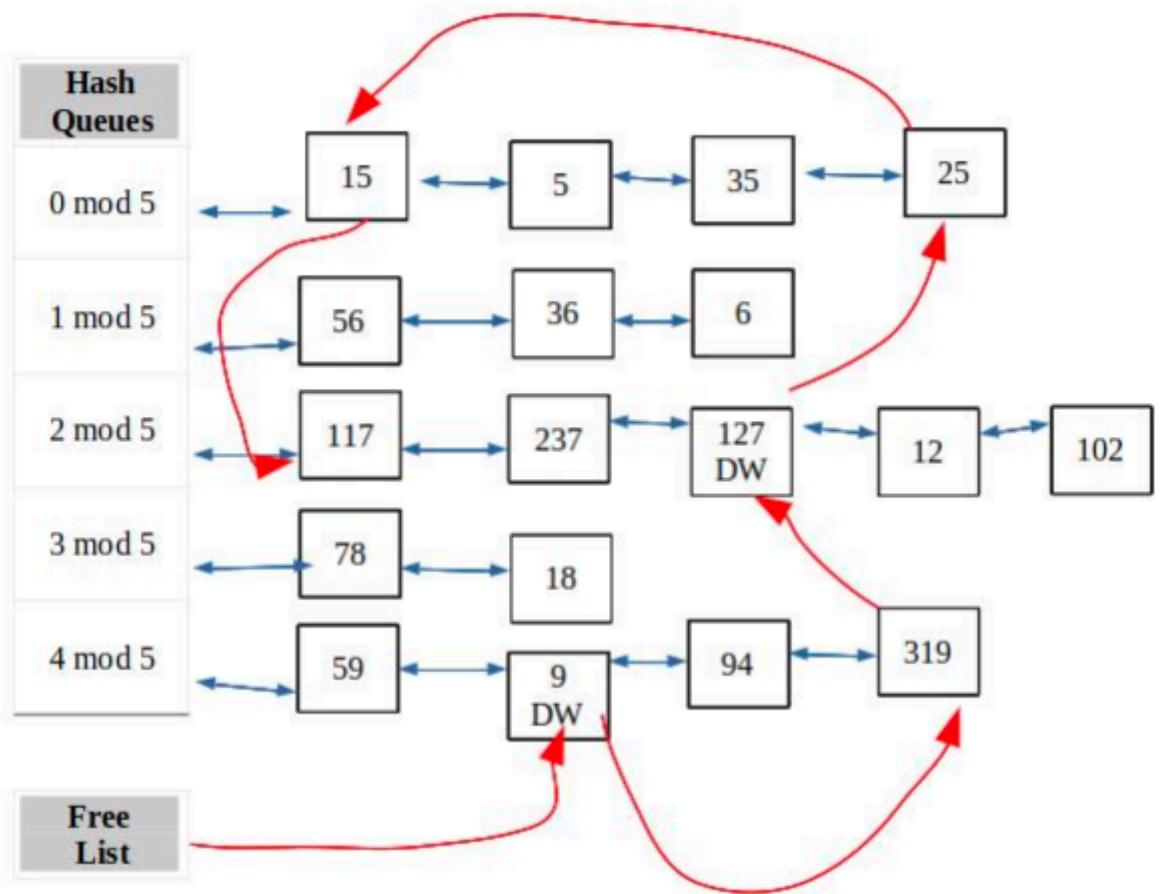
40. ¿Qué conviene más? más cantidad de hash queues con pocos elementos, o menos cantidad de hash queues con más elementos.

- MÁS hash queues con pocos elementos. La sobrecarga surge de la cantidad de hash, si son pocos hasheos más lineal la búsqueda y por ende menos beneficiosa.

1. El siguiente gráfico representa la situación de buffer cache en Unix System V. Seleccione la/las opciones que sucederá/n si un proceso P1 requiere el bloque 237.



- I. El proceso deberá esperar a que se libere el bloque.
- II. El proceso podrá utilizar el bloque directamente porque el mismo ya se encuentra en el buffer cache.
- III. El header que contiene el bloque 237 pasará a estar ocupado.
8. El siguiente gráfico representa la situación de buffer cache en Unix System V. Seleccione la/las opciones que sucederá/n si un proceso P1 requiere el bloque 61



I. Se toma el header que contiene el bloque 319 y se lo acomoda en la hash queue correspondiente al bloque 61 (la hash queue, 1, porque es el resto de calcular $61/5$ (la función módulo es 5)).

II. El proceso deberá esperar porque la hash queue correspondiente al bloque que solicita no tiene un header en estado libre

III. El header que contiene el bloque 9 pasa a estar en estado writing

IV. En la free list, como primer header queda el que contiene al bloque 9, en estado DW, y a este le sigue header que contiene al bloque 127, también en estado DW

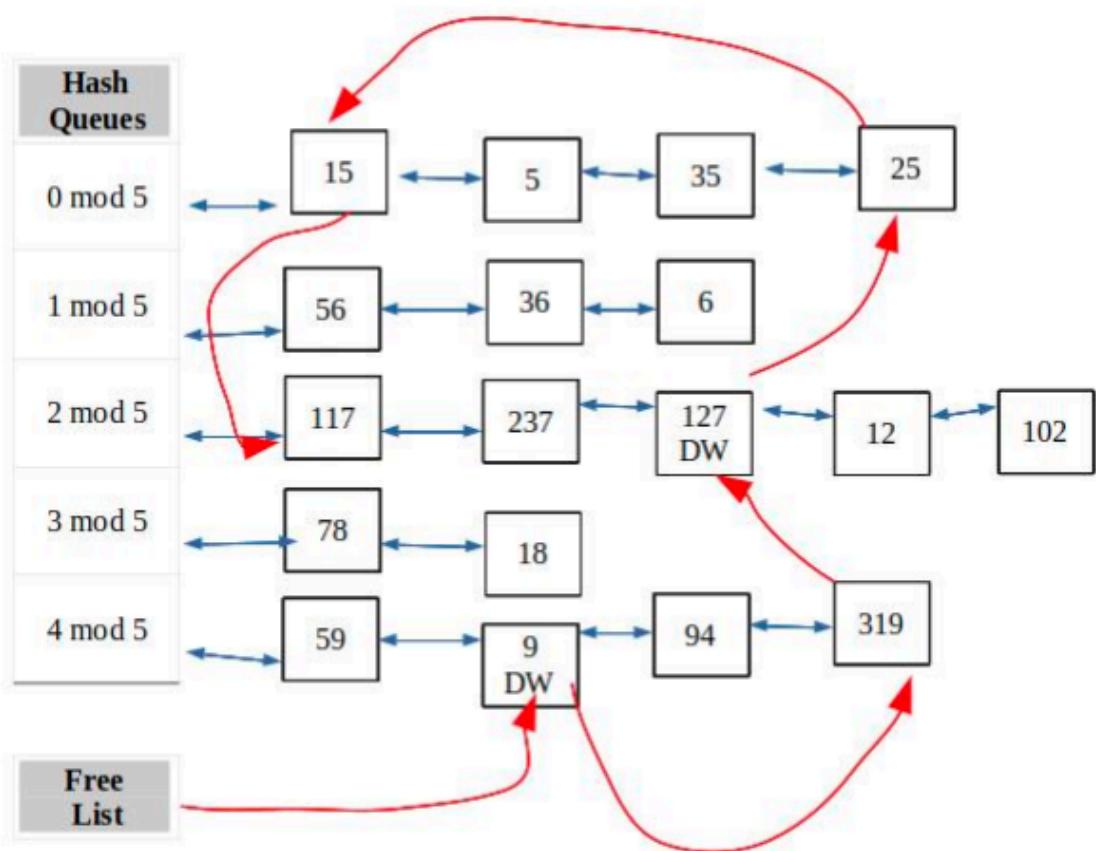
9. El uso de Clusters de gran tamaño podrá producir:

I. Mayor fragmentación interna

II. Mayor fragmentación interna y externa

III. Mayor fragmentación externa

10. El siguiente gráfico representa la situación de buffer cache en Unix System V. Seleccione la/las opciones que sucederá/n si un proceso P1 requiere el bloque 9.



- I. El bloque es otorgado al proceso pero recién lo podrá usar luego de que se mande a escribir a disco
- II. El header del bloque correspondiente debe permanecer en la free list porque está marcado como DW
- III. El header correspondiente sale de la free list**
- IV. El bloque es otorgado al proceso sin necesidad de escribir el mismo al disco**
- V. El proceso deberá esperar porque el header esta marcado como DW