

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

### Memoria:

- La organización y la administración de la “memoria principal” resultan ser uno de los factores MÁS importantes en el diseño de un sistema operativo.
- Todo programa y datos de estos deben estar en el almacenamiento principal (RAM) para:
  - Poder ejecutar dichos programas.
  - Referenciarlos de forma directa.
- El kernel se encarga de administrar los espacios de direcciones de la memoria RAM.
  - Administración eficiente → implica mejor funcionamiento.

### Tarea del sistema operativo:

- Debe llevar el registro de las partes de memoria que están en uso y aquellas las cuales no están en uso.
- Debe asignar espacio en memoria principal (RAM) a los procesos cuando estos necesitan el espacio.
- Debe liberar espacio de memoria asignada a procesos que han terminado.
- Se espera que el sistema operativo haga un uso eficiente de la memoria con la finalidad de alojar la mayor cantidad de procesos posible.
- Adicionalmente:
  - El SO debe lograr que el programador se **abstraiga** de la alocación de los programas.
  - Debe brindar **seguridad** entre procesos para que no haya accesos indebidos (un proceso violando el espacio privado de otro).
  - Brindar la posibilidad de **acceso compartido** a determinadas partes de la memoria (librerías, código en común, etc).
  - Garantizar la **performance**.

---

### **Administración de memoria**

- Hay una división lógica de la memoria física para alojar múltiples procesos:
  - Se garantiza protección.
  - Depende del mecanismo de administración provisto por el Hardware.
  - ¿Cómo va a parar lo lógico a lo físico? → depende del hardware y de un conjunto de estructuras utilizadas por el mismo.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

- Asignación eficiente:
  - Contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismos.
  - Asignar de manera eficiente la RAM.

### Requisitos para la administración de la RAM:

#### → Reubicación

- El programador no debe ocuparse de saber donde será colocado el programa en la RAM. Es decir, el programa es indiferente a la ubicación física.
- Mientras un proceso es ejecutado, puede ser sacado y traído a la memoria (swap), y posiblemente ser colocado en diferentes direcciones → el proceso debe ser independiente de la ubicación.
- Las referencias a la memoria se deben “traducir” según la ubicación actual del proceso.

#### → Protección

- Los procesos NO deben referenciar o acceder a direcciones de memoria de otros procesos. (SALVO si tienen permiso para ello).
- El chequeo se debe realizar durante la ejecución.
  - No es posible anticipar todas las referencias a memoria que un proceso puede realizar (es dinámico), por eso se realiza en run-time.

#### → Compartición

- Consiste en permitir que varios procesos accedan a la misma porción de memoria.
  - Ej: rutinas comunes, bibliotecas, espacios explícitamente compartidos para los procesos, etc.
- Permite un mejor aprovechamiento de la memoria RAM, evitando copias repetidas de instrucciones que son innecesarias.

---

### Abstracción – espacio de direcciones

- El espacio de direcciones es el rango de direcciones a memoria posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.
- El rango de direcciones lógicas esta definida por la arquitectura.
  - Procesador de 32 bits: 0..  $2^{32}$  -1
  - Procesador de 64 bits: 0..  $2^{64}$  -1
- El direccionamiento es independiente de la ubicación “real” del proceso en la memoria RAM, la dirección 100 en el rango puede no ser igual a la dirección 100 en la física.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

### Direcciones

- Lógicas (la que usan los procesos)
  - Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.
  - Representa una dirección en el “Espacio de direcciones del proceso”
- Físicas
  - Referencia a una localidad en la memoria física (RAM).
  - Dirección absoluta.
- **Cuando uso direcciones lógicas, necesito algún tipo de conversión a direcciones físicas para hallar el dato.**

### Conversión de direcciones

- ➔ Cada dirección lógica que se genera se debe convertir a una dirección física.
- ➔ Una forma simple de hacer esto es utilizando registros auxiliares.
  - Registro base:
    - Dirección de comienzo del espacio de direcciones del proceso en la RAM.
  - Registro límite:
    - Dirección final del proceso o tamaño de su espacio de direcciones.
  - A partir de los valores de esos dos registros, podemos tomar la dirección lógica como un desplazamiento desde la base, y controlar que dicha suma (registro base + dirección lógica=dirección física) no exceda el registro límite. Si la dirección supera el límite ➔ excepción.
- ➔ Los valores de ambos registros se fijan cuando el espacio de direcciones del proceso se carga en la memoria.

---

### Direcciones lógicas vs Físicas

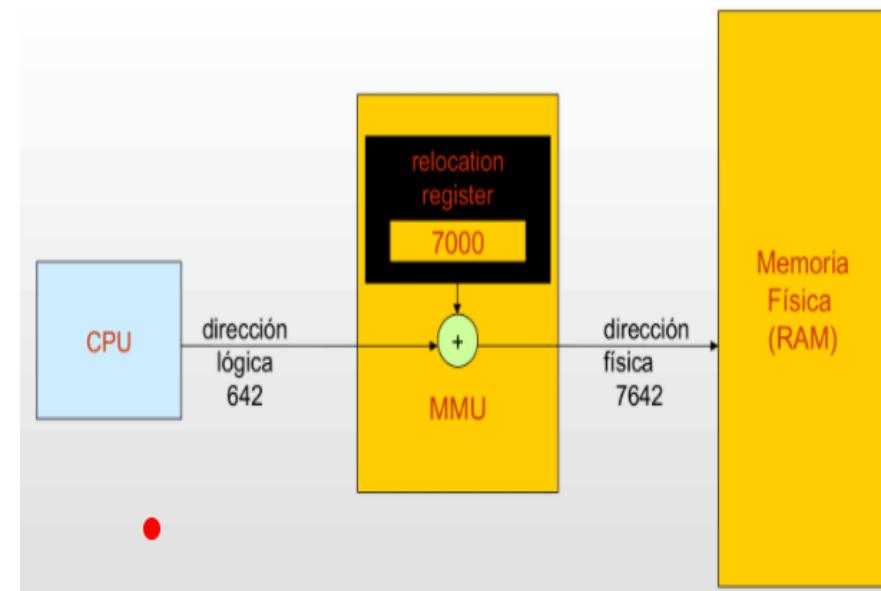
- Si la CPU trabaja con direcciones lógicas, para acceder a la memoria principal, se deben transformar en direcciones físicas.
  - Resolución de direcciones (address-binding): transformar la dirección lógica en la dirección física que corresponde.
- Resolución en momento de compilación o en tiempo de carga del proceso (no usado en la actualidad):

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

- Direcciones lógicas y físicas son idénticas.
- Para reubicar un proceso es necesario recompilarlo o recargarlo.
- Resolución en tiempo de ejecución
  - Direcciones lógicas y físicas son diferentes.
  - Las lógicas son llamadas direcciones virtuales.
  - La reubicación se puede realizar fácilmente.
  - La resolución debe ser rápida en velocidad, a la par del CPU.
  - El mapeo entre “virtuales” y “físicas” es realizado por hardware.
    - Memory Management Unit (MMU).
      - Es un dispositivo de hardware que mapea direcciones virtuales a físicas (convierte.)
        - Es parte del procesador. (Hardware)
        - Re-programar el MMU es una operación privilegiada.
          - SOLO SE REALIZA EN MODO KERNEL.
      - El valor en el “registro de realocación” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria.
        - Los procesos nunca usan direcciones físicas.

**Mecanismos de asignación de memoria:**

➔ Particiones fijas:  
el primer  
esquema que se implementó.



- La memoria se divide en particiones o regiones de tamaño fijo (todos del mismo tamaño o no).
- Cada partición aloja un proceso.
- Cada proceso se coloca de acuerdo a algún criterio (first fit, best fit, worst fit, next fit) en alguna partición (son algoritmos).

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

- Al ser fijo podría ocurrir fragmentación interna (espacio desperdiciado).
- ➔ Particiones dinámicas: es la evolución del esquema anterior.
  - Las particiones varían en tamaño y en número.
  - Cada una aloja un proceso.
  - Cada partición es generada de forma dinámica con el tamaño justo que necesita el proceso.
  - Generación fragmentación externa.
  - Se recomienda peor ajuste, ya que la que me va a quedar libre es probable que sea la más grande.
- ➔ Fragmentación (problemas de las particiones)
  - Producida cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua.
  - Fragmentación interna:
    - Producido en particiones fijas.
    - Porción de partición que queda sin utilizar.
  - Fragmentación externa:
    - Producido en particiones dinámicas.
    - Son huecos que van quedando en la memoria a medida que los procesos terminan.
    - Estos huecos si bien son memoria libre, al no estar contiguos entre sí un proceso no puede ser alocado.
    - El problema se soluciona compactando → pero es costosa.

### ***Problemas del esquema registro base y límite***

- El esquema de registro base + registro límite presenta problemas:
  - Necesidad de almacenar el espacio de direcciones de forma **continua** en memoria física.
  - Los primeros SO definían particiones fijas de memoria hasta evolucionar en particiones dinámicas.
  - Fragmentación.
  - Mantener partes del proceso que sean innecesarias.
  - Los esquemas de particiones fijas y dinámicas NO son usados hoy en día.
- Solución:
  - Paginación

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

### ○ Segmentación

#### Paginación

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



- ➔ Como se puede observar, se rompe la continuidad.
- ➔ Puede causar fragmentación interna (menos rompe bolas que la externa).
- ➔ No es muy importante esa fragmentación interna.

#### Segmentación

- ➔ Esquema que se asemeja a la visión del usuario. El programa es dividido en partes/secciones, donde en cada sección se guardan datos similares.
- ➔ Un programa es una colección de segmentos. Un segmento es una unidad lógica como:
  - Programa Ppal, procedures, funciones, variables locales y globales, stack, etc.

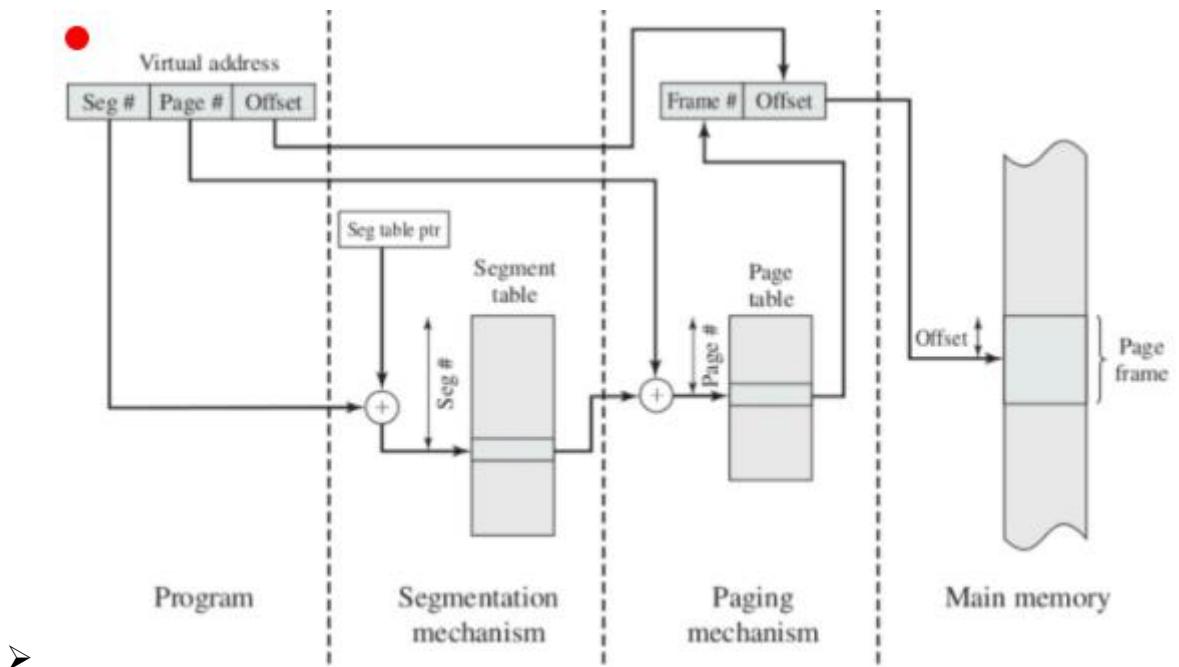
## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

- Cada segmento tiene un registro base y un registro límite.
  - Por lo que se genera una tabla de segmentos con esos dos valores para cada segmento por proceso.
- ➔ Puede causar fragmentación EXTERNA.
- ➔ Todos los segmentos de un programa pueden NO tener el mismo tamaño (códigos, datos, rutinas). La base y límite del segmento son dinámicos.
- ➔ Las direcciones lógicas consisten en 2 partes:
- Selector de segmento.
  - Desplazamiento dentro del segmento (sobre registro base y límite).
- ➔ La segmentación posee ventaja sobre la paginación respecto de: la protección de espacios de memoria y la compartición de bloques de memoria.
- ➔ Cuando uno compila ➔ el compilador deja huellas del segmento.
- 

### Segmentación paginada (mix de las dos anteriores):

- Paginación
  - Transparente al programador.
  - Elimina fragmentación externa.
- Segmentación
  - Visible al programador.
  - Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección.
- Segmentación paginada: cada segmento es dividido en páginas de tamaño fijo.
  - Cada segmento tiene una tabla de páginas que le corresponde.
  - Toma las ventajas de ambas: compartición, protección(de parte de la segmentación), evitar fragmentaciones (de parte de la paginación).
  - Se sigue guardando en páginas
  - La unidad de trabajo para subir o bajar de la RAM es la página

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA



## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

### Motivación para Memoria Virtual

- Podemos pensar también que no todo el espacio de direcciones del proceso se necesita en todo momento.
  - Rutinas o librerías que se ejecutan una única vez (o nunca).
  - Partes del programa que no vuelven a ejecutarse.
  - Regiones de memoria alocadas dinámicamente y luego liberadas.
- No hay necesidad que la totalidad de la imagen del proceso sea cargada en memoria.

### Como lo podemos trabajar...

- El SO puede traer a memoria las “piezas” de un proceso a medida que éste las necesita.
- Definiremos como “**Conjunto residente**” a la porción del espacio de direcciones del proceso que se encuentra en memoria RAM. (también como working set)
- Con el apoyo del HW:
  - Se detecta cuando se necesita una porción del proceso que no está en su Conjunto Residente.
  - Se debe cargar en memoria dicha porción para continuar con la ejecución.
- Ventajas:
  - Más procesos pueden ser mantenidos en memoria.
    - Solo se cargan algunas secciones de cada proceso.
    - Con más procesos en memoria RAM es más probable que existan más procesos Ready.
  - Un proceso puede ser mas grande que la memoria RAM (la cantidad requerida de RAM para el proceso total se excede).
    - El usuario no se debe preocupar por el tamaño de sus programas.
    - La limitación la impone el HW y el bus de direcciones.

### ¿Qué se necesita para aplicar la técnica de Memoria Virtual?

- El hardware debe soportar paginación por demanda (y/o segmentación por demanda), es decir que se meten páginas a medida que se necesiten.
- Un dispositivo de memoria secundaria (disco) que dé el apoyo para almacenar las secciones del proceso que no están en memoria principal (área de intercambio - swap).
- El SO debe ser capaz de manejar el movimiento de las páginas (o segmentos) entre la memoria principal y la secundaria.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

### Memoria virtual con paginación

- ➔ Cada proceso tiene su tabla de páginas.
- ➔ Cada entrada en la tabla referencia al frame o marco en el que se encuentra la página en RAM.
- ➔ Cada entrada en la tabla de páginas tiene bits de control:
  - Bit V: indica si la página está en memoria. (cargada), el SO modifica el valor de este bit ya que es el encargado de subir o bajar páginas.
  - Bit M: indica si la página fue modificada. Si se modificó, en algún momento se deben reflejar los cambios en Memoria Secundaria (disco) para mantener una coherencia de datos. El bit modificado lo va cargando el hardware en sí (ya que este sabe sobre operaciones de escritura), el bit lo usa el SO.

### Fallo de páginas(page fault)

- Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en RAM. Bit V = 0.
  - La página no se encuentra en su conjunto residente.
- El HW detecta la situación y genera un trap al S.O
- El S.O podrá colocar al proceso en estado de “Blocked” (espera) mientras gestiona que la página que se necesite se cargue.
  - La carga consiste en:
    - El S.O busca un marco libre en la memoria y genera una operación de E/S al disco para copiar en dicho marco la página del proceso que se necesita usar.
    - El SO puede asignarle la CPU a otro proceso mientras se completa la E/S
      - La E/S avisará por interrupción cuando finalice.
  - Cuando la E/S finaliza, se notifica al SO y este:
    - Actualiza la tabla de páginas del proceso.
      - Coloca el Bit V en 1 en la página correspondiente.
      - Coloca la dirección base del frame donde se colocó la página.
    - El proceso que generó el fallo de página vuelve al estado de Ready.
    - Cuando el proceso, se ejecute se volverá a ejecutar la instrucción que provocó el fallo.
- **PERFORMANCE:**
  - Si los page faults son excesivos, la performance del sistema decae.
  - Tasa de page faults entre 0 y 1.  $0 \leq p \leq 1$

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

- Si  $p=0$  no hay page faults.
- Si  $p=1$ , en cada acceso a memoria hay un page fault (caos).
- $P$  tiene que tender a 0, nunca va a ser 0 por que la técnica se basa en el fallo de pagina.
- Effective Access time (EAT) consiste en el tiempo efectivo de acceso:

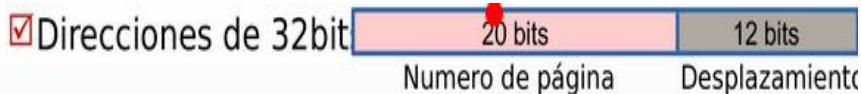
$$\text{EAT} = \left( (1 - p) \times \text{memory access} \right) + p \times (\text{page\_fault\_overhead} + [\text{swap\_page\_out}] \cdot \text{swap\_page\_in} + \text{restart overhead})$$

- 
- Si no hay un frame libre, habrá que descargar otro para lograr espacio para la nueva pagina ( $\text{swap\_page\_out}$ )

### Tabla de paginas

- Cada proceso tiene su tabla de páginas.
- El tamaño de la tabla de páginas depende del espacio de direcciones del proceso.
- Puede alcanzar un tamaño considerable.
- Formas de organizar:
  - Tabla de 1 nivel: tabla única lineal
  - Tabla de 2 niveles: o más, multinivel
  - Tabla invertida: Hashing
- La forma de organizar la tabla de paginas depende del HW subyacente.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

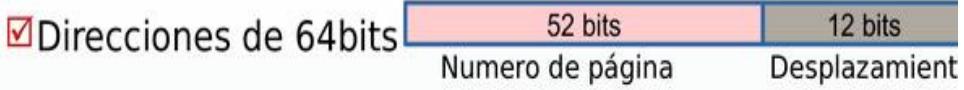


### ✓ Ejemplo

- ✓ Cantidad de Page Table Entries (PTEs) máximas que puede tener un proceso =  $2^{20}$
- ✓ 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 =  $2^{20}/2^{10} = 2^{10}$
- ♦ Tamaño tabla de páginas del proceso:  
 $2^{10} * 4bytes = \textbf{4MB por proceso}$



### ✓ 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:  $4KB/4B = 2^{10}$

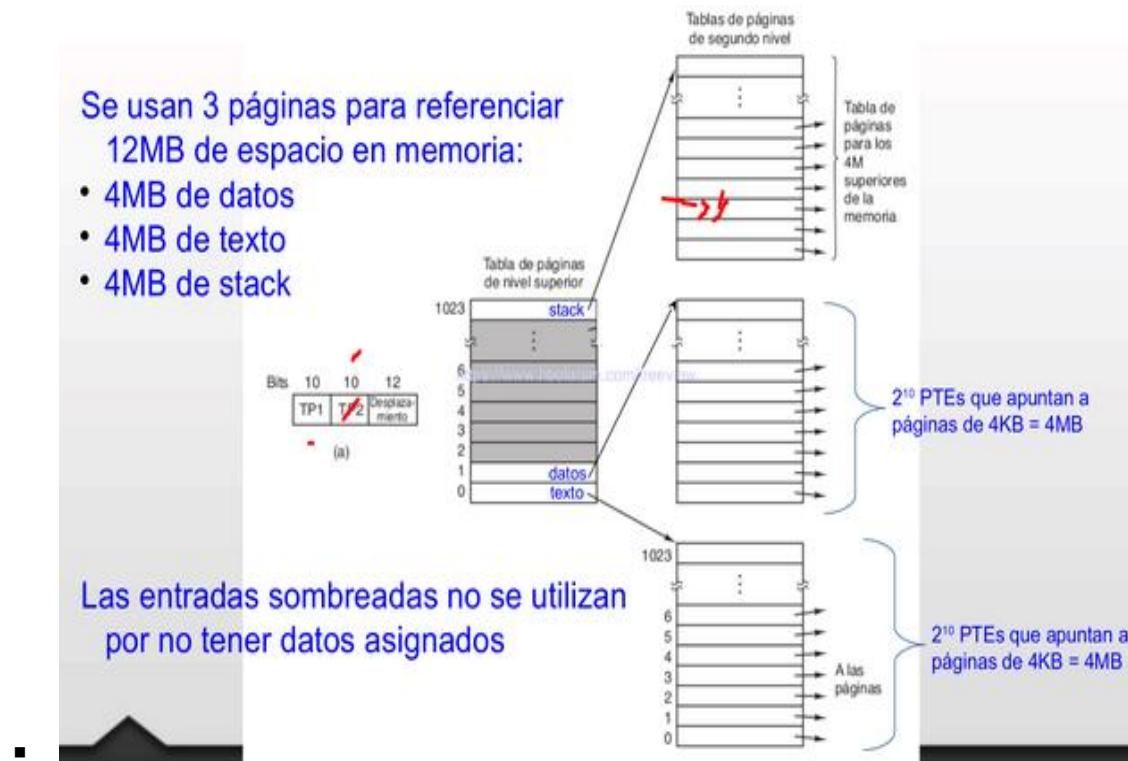
### ✓ Tamaño de tabla de páginas

- ♦ 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 =  $2^{42} * 4bytes = 2^{54}$

**Más de 16.000GB por proceso!!!**

- Eso para tablas de 1 nivel.
- Existen las tablas de 2 niveles.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA



- 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 RAM.
- Además solo se carga una parcialidad de la tabla de páginas (la que se necesite usar).
- Existe un esquema de direccionamientos indirectos.
- Beneficios?
  - Las tablas de segundo nivel (o más) se pueden llevar a memoria secundaria, liberando RAM.
- Desventaja?
  - Más de un acceso a memoria para obtener un dato.
- -----

### - Tabla invertida:

- Utilizada en arquitecturas donde el espacio de direcciones es muy grande.
  - Las tablas de páginas ocuparían muchos niveles y la traducción es costosa.
  - Por esa razón se usa ésta técnica.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

- Por ejemplo, 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.
- Hay una entrada por cada marco de página en la memoria real. Es la visión inversa a la que veníamos viendo.
- Hay una sola tabla para todo el sistema.
- El espacio de direcciones de la tabla se refiere al espacio físico de la RAM (todo lo que esté cargado), en vez del espacio de direcciones virtuales de un proceso.
- El numero de página es transformado en un valor de HASH.
- El HASH se usa como índice de la tabla invertida para encontrar el marco asociado.
- Se define un mecanismo de encadenamiento para solucionar colisiones (el hash da igual para dos direcciones virtuales).
- Solo se mantienen las entradas de tablas de pagina (PTE) de páginas presentes en memoria ram.
  - La tabla invertida se organiza como tabla hash en RAM.
    - Se busca indexadamente por número de página virtual.
    - Si está presente en tabla, se extrae el marco de página y sus protecciones.
    - Si no está presente en tabla, corresponde a un page fault.

### Tamaño de la pagina

#### ➔ Pequeño

- Menor fragmentación interna (menos probabilidad de que queden espacios libres que sean muy grandes).
- Más paginas requeridas por proceso → tablas de paginas más grandes.
- Más paginas pueden residir en memoria.

#### ➔ Grande

- Mayor fragmentación interna.
- La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente → es más rápido mover páginas hacia la memoria ram.

#### ➔ Relación con la E/S

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

- Vel de transferencia: 2 MB/s
  - Latencia: 8 ms
  - Búsqueda: 20.
  - Búsqueda y latencia tienen que ver con el cabezal del disco.
  - ➔ Pagina de 512 bytes.
    - 1 pagina → total: 28,2 ms
    - Solo 0,2 ms de transferencia (1%) → transferencia real (sin búsqueda ni latencia).
    - 2 paginas → 56,4ms
  - ➔ Pagina de 1024 bytes
    - Total: 28,4 ms
    - Solo 0,4 ms de transferencia.
  - ➔ Vemos que a mayor tamaño de página se transfiere mejor
- 
- 

### Translation Lookaside Buffer (TLB)

- Cada referencia en el espacio virtual puede causar 2 (o más) accesos a la memoria física RAM.
    - Uno (o más) para obtener la entrada en tabla de páginas.
    - Uno para obtener los datos.
  - Para solucionar este problema, una memoria cache de alta velocidad es usada para almacenar entradas de páginas.
    - TLB.
  - Contiene las entradas de la tabla de páginas que fueron usadas más recientemente.
  - Dada una dirección virtual, el procesador examina la TLB.
    - Si la entrada de la tabla de páginas se encuentra en la TLB (hit), es obtenido el frame y armada la dirección física.
    - Si la entrada no es encontrada en la TLB (miss), el número de página es usado como índice en la tabla de páginas del procesos.
  - Se controla si la pagina está en la memoria
    - Si no está, se genera un Page Fault.
  - La TLB es actualizada para incluir la nueva entrada.
  - El cambio de contexto genera la invalidación de las entradas de la TLB.
-

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

### Asignación de Marcos:

- ¿Cuántas páginas de un proceso se pueden encontrar en memoria?
  - Tamaño del Conjunto Residente.
- Asignación dinámica
  - El número de marcos para cada proceso es variable, los procesos de alta prioridad pueden usar muchas páginas (y esto puede dejar que procesos con menos prioridad tengan menos páginas disponibles).
- Asignación fija:
  - Número fijo de marcos para cada proceso.
- Asignación equitativa: Ejemplo: si tengo 100 frames y 5 procesos, 20 frames para cada proceso, como desventaja: cada proceso puede requerir más o menos frames, no es eficiente.
- Asignación proporcional: Asignado acorde al tamaño del proceso.

$$s_i = \text{size of process } p_i$$

$$S = \sum s_i$$

$m$ = total number of frames

$$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$$

$$m=64$$

$$s_1=10$$

$$s_2=127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

- El SO hace juego entre asignación fija y dinámica.

- Primero al proceso se le asigna un número fijo de frames.
  - Si necesita más se le asigna más.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

### Reemplazo de páginas:

- ¿Qué sucede si ocurre un fallo de página y todos los marcos están ocupados → “Se debe seleccionar una pagina víctima” (la cual será sacrificada xd).
- ¿Cuál sería un reemplazo optimo?
  - Que la pagina a ser removida no sea referenciada en un futuro próximo.
  - Es perfecto, pero sólo teórico, ósea, inaplicable.
- Por eso el sistema operativo trata: la mayoría de los reemplazos predicen el comportamiento futuro mirando el comportamiento pasado.

### Alcance del reemplazo

- Reemplazo global
  - El fallo de página de un proceso puede reemplazar la página de **cualquier** proceso.
  - El SO no controla la tasa de page-faults de cada proceso.
  - Puede tomar frames de otro proceso aumentando la cantidad de frames asignados a él.
  - Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad.
- Reemplazo local
  - El fallo de página de un proceso solo puede reemplazar sus propias páginas – de su conjunto residente.
  - No cambia la cantidad de frames asignados.
  - El SO puede determinar cual es la tasa de page-faults de cada proceso.
  - Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.

### Algoritmos de reemplazo

- Óptimo: es sólo teórico.
- FIFO: el más sencillo
- LRU (Least Recently Used): requiere soporte del hardware para mantener timestamps de acceso a las páginas. Favorece a las páginas menos recientemente accedidas.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

- 2da chance: un avance del FIFO tradicional que beneficia a las páginas más referenciadas.
- NRU (Non Recently Used):
  - Utiliza Bits R y M.
  - Favorece a las páginas que fueron usadas recientemente.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

### Thrashing (Hiperpaginación)

- Concepto: decimos que un sistema está en thrashing cuando pasa más tiempo paginando (atendiendo page faults) que ejecutando procesos.
- Como consecuencia, el sistema sufre una baja significativa de performance.

### Ciclo de Thrashing

1. El kernel monitorea el uso de la CPU.
  2. Si hay baja utilización → aumenta el grado de multiprogramación (la misma debe tener un límite).
  3. Si el algoritmo de reemplazo es global, pueden sacarse frames de 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
- ➔ Una de las soluciones cuando entra en hiperpaginación (o la CPU se malgasta en administración de memoria) consiste en bajar el grado de multiprogramación.

### Control de thrashing: soluciones propuestas

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

### Conclusión sobre la hiperpaginación

- Si un proceso cuenta con todos los frames que necesita, no habría thrashing (esto es, que el SO sepa siempre que frames necesita un proceso en el momento adecuado).
- Una manera de abordar esta problemática es utilizando la estrategia de Working Set, la cual se apoya en el modelo de localidad.
- Otra estrategia para afrontar el thrashing con el mismo espíritu es el algoritmo PFF (Frecuencia de Fallos de Pagina).

### El modelo de localidad:

- Cercanía de referencias o principio de referencias.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

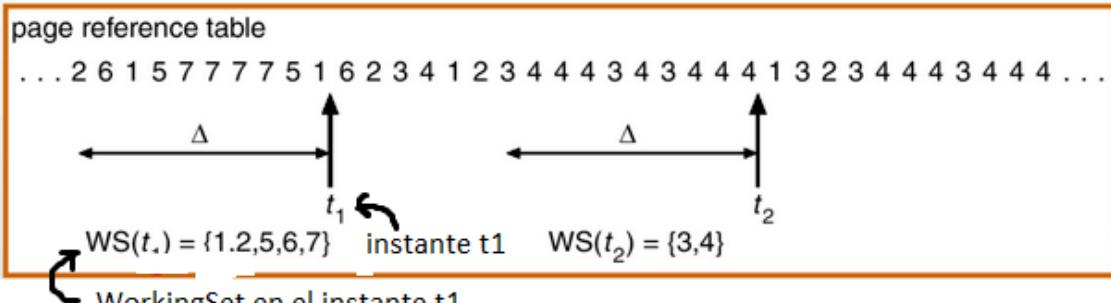
- Las referencias a datos y programas dentro de un proceso tienden a agruparse.
- La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento.
- En cortos períodos de tiempo, el proceso necesitará pocas “piezas” del proceso (ej, una página de instrucciones y otra de datos).
- En la vida del proceso, el mismo va ejecutándose en diferentes localidades, cada localidad con su conjunto de instrucciones próximas entre sí. Cuando ocurre el cambio de localidad el proceso requiere más frames para seguir sus tareas.
- Continuando:
  - Un programa se compone de varias localidades.
  - Ej: cada rutina será una nueva localidad: se mencionan sus direcciones (cercanas) cuando se está ejecutando.
  - Para prevenir la hiperactividad, un proceso debe tener en memoria (conjunto residente) sus páginas más activas (esto implica menos page faults).

### El modelo de working set:

- Basado en el modelo de localidad definido anteriormente.
- Ventana del working set ( $\Delta$ ): las referencias de memoria más recientes.
- Working set: consiste en el conjunto de páginas que tienen las más recientes  $\Delta$  que mencionan a páginas.

$\Delta=10$

Si en  $T_1$  debo elegir una página víctima, y tengo mi conj. residente y mi working set. Voy a elegir aquella página que no se encuentre en el working set.



- WorkingSet en el instante  $t_1$
- Si el valor de  $\Delta$  es MUY chico: lo que puede suceder es que el WS que arme no me cubra la localidad real del proceso. Chica no me sirve.
- Si el valor de  $\Delta$  es MUY grande: puede que me quede en mi WS páginas que ya no tenga que usar.
- Surge el problema, qué valor correcto darle a  $\Delta$  (es solucionable).

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

- El problema mayor, ¿Cómo es que el sistema operativo tiene registro de las páginas que va usando?, se debería apoyar en el hardware
- El SO puede (como solución), ver la tabla de páginas de cada proceso y analizar cada bit de Referencia (si está en 1 implica que la página se estuvo usando). Entonces, los bits de R que están en 1 son puestos a 0. En la próxima vez que el SO aparezca a ver la tabla de páginas, va a ver cuales páginas tienen de nuevo el bit en 1 para empezar a analizar y comparar con la vez anterior que miró la tabla. Como para determinar cuál página está siendo usada frecuentemente, (resulta en una determinación que se aproxima). El problema es que ésta opción resulta costosa, por lo que el modelo de Working Set implica mucho CPU.
- En resumen, la técnica pinta buena, pero es costosa de implementar (consumo de CPU).
- Medida del working set:
  - Si la demanda total de frames es mayor a la cantidad de frames disponibles. Entonces hay THRASHING.

**m = cantidad frames disponibles**

**WSS<sub>i</sub>= medida del working set del proceso p<sub>i</sub>.**

**$\sum WSS_i = D$ ;**

**D= demanda total de frames.**

**Si D>m, habrá **thrashing**.**

- Para achicar la demanda, debo bajar el grado de multiprogramación.

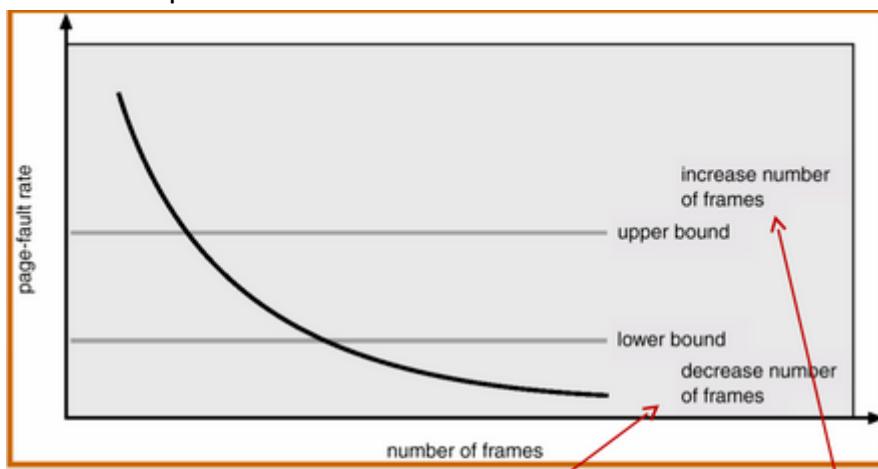
### Prevención del thrashing:

- SO monitoreando cada proceso, dándole tantos frames hasta su WSS(medida del working set del proceso Pi).
- 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.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

### Prevención del thrashing por PFF:

- La técnica PFF (Page Fault Frequency o Frecuencia de Fallo de Páginas), en lugar de calcular el Working Set de los procesos, utiliza la tasa de fallos de pagina para estimar si el proceso tiene un conjunto residente que representa adecuadamente al WS.
- Hace que el Page Fault quede en un valor “manejable”
- PFF: Frecuencia de page faults.
- PFF alta → se necesitan más frames para el proceso.
- PFF baja → los procesos tienen muchos frames asignados (el proceso tiene frames de sobra).
- Si o sí reemplazo local.



#### Establecer tasa de PF aceptable

- ✓ Si la tasa actual es baja, el proceso pierde frames
- ✓ Si la tasa actual es alta, el proceso gana frames.
- Continuando:
  - PFF es la técnica usada por los sistemas operativos hoy en día.
  - Se deben establecer límites superior e inferior de las PFF's deseadas.
  - Si se excede PFF máx → se le asigna un frame más al proceso, ya que el mismo genera muchos page fault y probablemente lo esté necesitando.
  - Si la PFF está por debajo del mínimo → se le quita un frame al proceso, asumiendo que el mismo tiene frames de más.
  - Puede llegar a ocurrir que se suspenda un proceso si no hay más frames disponibles. Sus frames se reasignan a procesos de alta PFF.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

### Demonio de paginación

- Proceso creado por el SO durante el arranque que apoya a la administración de la memoria.
- Es ejecutado cuando el sistema tiene una baja utilización o algún parámetro de la memoria lo indica.
  - Poca memoria libre.
  - Mucha memoria modificada.
- Tareas del demonio
  - Limpiar páginas modificadas sincronizándolas con el swap.
    - Esto reduce el tiempo de swap posterior ya que las páginas están limpias.
    - Reduce también el tiempo de transferencia al sincronizar varias páginas contiguas.
  - Mantiene un cierto número de frames libres en el sistema, viendo que procesos tienen un PFF alto o bajo.
  - Demora la liberación de una página hasta que la misma haga falta realmente.
- En Windows → proceso system.
- En Linux → kswapd

### Memoria compartida

- Gracias al uso de la tabla de páginas, varios procesos pueden compartir un marco de memoria; para ello ese frame 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 (read only), por ej: editores de texto, compiladores, etc.
  - Los datos son privados a cada proceso y se encuentran en páginas no compartidas.
- El frame de memoria que es compartido, físicamente está en un solo lugar.

### Mapeo de archivo en memoria

- ➔ Es una técnica que permite a un proceso asociar el CONTENIDO de un archivo a una región del propio espacio de direcciones virtuales.
- ➔ El contenido del archivo no es subido a RAM hasta que se generan Page Faults.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

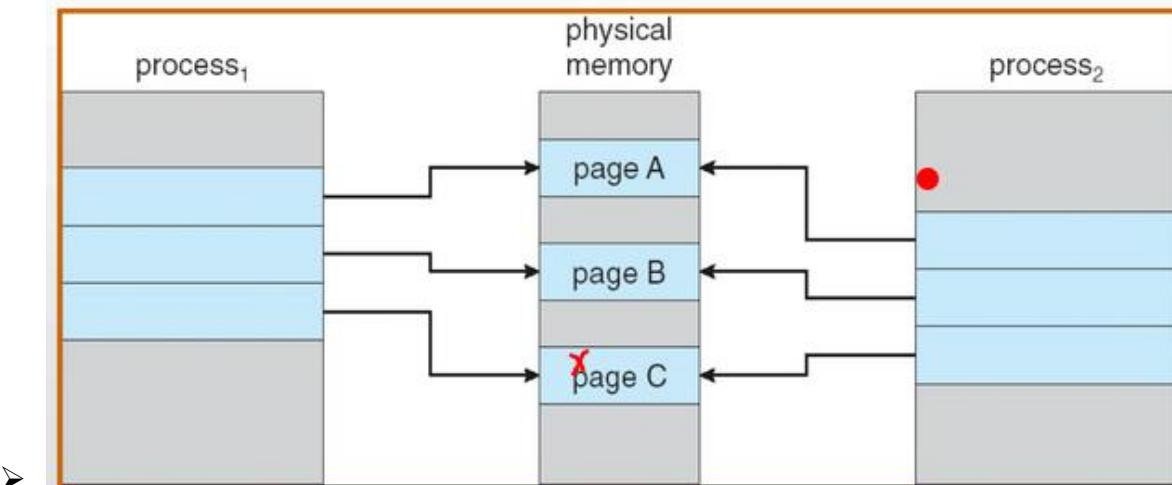
- ➔ Básicamente es alinear el contenido de un archivo en un propio espacio de direcciones, por lo que si se accede a la porción que posee la alineación con dicho archivo, es como acceder al archivo.
- ➔ Continuando...
  - El contenido de la pagina que genera el PF es obtenido desde el archivo asociado (no del área de SWAP)
  - Cuando el proceso termina o el archivo es liberado, las paginas modificadas son escritas en el archivo correspondiente.
  - Permite la realización de E/S de una manera alternativa (sin usar funciones del SO como read o write) a usar operaciones directas sobre el sistema de archivos.
  - Es utilizado comúnmente para asociar librerías compartidas o DLLs.

### Copia en Escritura

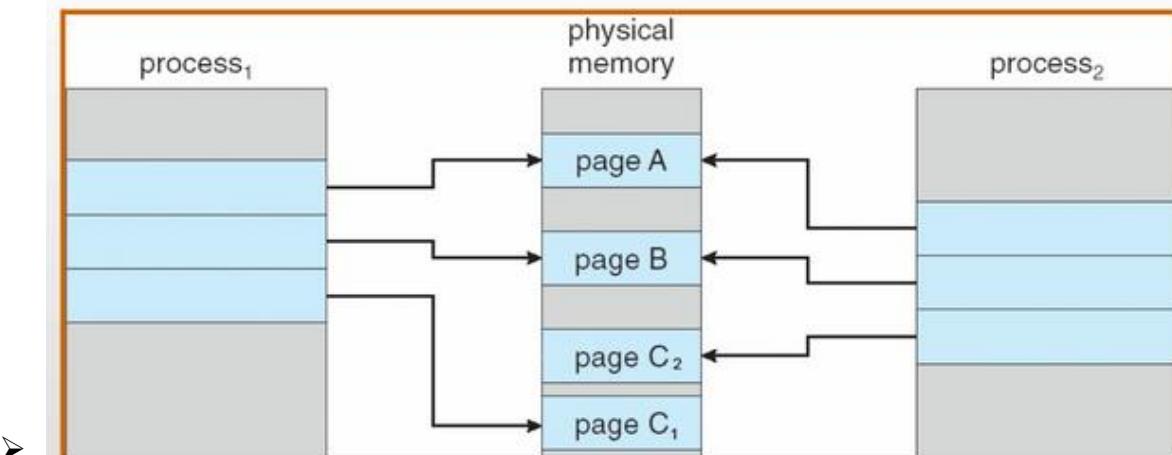
- La copia en escritura (Copy-on-Write, COW), permite a los procesos padre e hijo compartir inicialmente las mismas páginas de memoria (apuntan a los mismos frames).
  - Si uno de ellos modifica una pagina que comparten, entonces la página es copiada.
  - Usada mucho en Linux.
- COW permite la creación de procesos de forma más eficientemente debido a que sólo las páginas modificadas son duplicadas.
- Las páginas de datos son de SOLO lectura, (ya que tanto hijo como padre deberían tener una técnica de trabajo que les permita manejar datos distintos). Entonces al escribir una página de datos se aplica el COPY-on-WRITE.
- La pagina copiada va a partirse en dos, y tanto padre va a apuntar a copia 1 y el hijo a copia2.

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

### El Proceso 1 Modifica la Página C (Antes)



### El Proceso 1 Modifica la Página C (Después)



### Área de Intercambio (swap)

- En los diferentes SO
  - Linux: área dedicada, separada del sistema de archivos.
  - Windows: un archivo dentro del sistema de archivos.
- Técnicas para la administración de la SWAP:
  - Cada vez que se crea un proceso es reservada una zona del área de intercambio que tiene el mismo tamaño del proceso. A cada proceso se le asigna la dirección en disco de su área de swap correspondiente. La lectura de dicha área se realiza sumando: número de página virtual + dirección de comienzo del área de swap asignada al proceso, el problema es que **desperdicio espacio**.
  - No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a intercambiar, y el espacio es liberado cuando la página vuelve a memoria. Problema: se

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS: TEMA 3 - MEMORIA

debe llevar contabilidad en memoria (página a página) de la localización de las páginas en disco, es la **técnica más usada ya que optimiza el espacio usado en área de swap**.

- Continuando:
  - Cuando una página no está en memoria, sino en disco, como se puede saber en qué parte del área de swap está?
    - Respuesta: el PTE de dicha página tiene el bit V=0 y todos los demás bits sin usar (toda la información guardada no tiene sentido).

# ISO, TEORÍA 5: FILESYSTEM

## Que es un archivo?

- Entidad abstracta con nombre.
- Espacio lógico continuo y que es direccionable.
- Provee a los programas con datos (entrada).
- Permite a los programas guardar datos (salida).
- El programa mismo es información que debe ser guardada.

## Punto de vista del usuario sobre un archivo

- Que operaciones puedo llevar a cabo sobre el archivo.
- Como debo nombrarlo.
- Como asegurar la protección del mismo.
- Como compartirlo.
- El usuario no trata con aspectos físicos del HW.

## Punto de vista sobre el diseño

- Como implementamos la construcción lógica para crear un archivo.
- Como implementamos directorios para contener y organizar archivos.
- Manejo del espacio en disco.
- Manejo del espacio libre.
- Eficiencia y mantenimiento.

## Que es un sistema de archivos

- Conjunto de unidades de software (que requieren CPU) que proveen los servicios necesarios para la utilización de archivos. La idea es que cada operación sea eficiente ya que la CPU no puede ser malgastada.
  - Crear.
  - Borrar.
  - Buscar.
  - Copiar.
  - Leer.
  - Escribir.
  - Etc.
- Permite la facilitación el acceso a los archivos por parte de las aplicaciones.
- Permite la abstracción al programador sobre el bajo nivel (el programador no desarrolla el software de administración de archivos).

## Objetivos del SO en cuanto a archivos

- Cumplir con la gestión de datos.
- Cumplir las solicitudes del usuario.

## ISO, TEORÍA 5: FILESYSTEM

- Minimizar o eliminar la posibilidad de perder o destruir datos de forma que sea garantizada la integridad del contenido de los archivos.
- Dar soporte de E/S a distintos dispositivos.
- Brindar un conjunto de interfaces de E/S para tratar archivos.

### Tipos de archivos

- Archivos de tipo regular
  - Texto plano
    - Source file
  - Binarios
    - Object file
    - Executable file
- Directories (los directorios son archivos)
  - Su contenido son relaciones con otros archivos y mantienen la estructura en el Filesystem

### Atributos de un archivo

- Nombre.
- Identificador único.
- Tipo.
- Localización, toda la información necesaria para ubicar el contenido del archivo en disco.
- Tamaño.
- Protección, seguridad y monitoreo
  - Dueño, permisos, contraseña.
  - Momento en que el usuario lo modificó, creó, accedió por última vez (FECHAS)
  - ACLs → Access control list.

### Directories

- Archivos que contienen 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, ósea que proporciona la ruta concreta para el archivo específico (el cuál puede estar en otros directorios).
- Operaciones en directorios:
  - Buscar un archivo.
  - Crear un archivo.
  - Borrar un archivo.

## ISO, TEORÍA 5: FILESYSTEM

- Listar el contenido.
- Renombrar.
- El uso de los directorios ayuda con:
  - Eficiencia -> ubicar rápidamente un archivo.
  - Uso del mismo nombre de archivo:
    - Diferentes usuarios pueden tener el mismo nombre de archivo.
  - Agrupar: se agrupa un conjunto de archivos de forma lógica en base a propiedades o funciones:
    - Ejemplo: programas java, juegos, librerías.

### 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 lo 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.
  - Absoluto: 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:
  - 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.
    - Permiso de ejecución → el usuario puede ejecutar.
    - Permiso de lectura → el usuario puede leer el archivo.
    - Permiso de agregar → el usuario puede agregar datos al archivo (no modificar).
    - Permiso de actualización → el usuario puede modificar, borrar y agregar datos. Incluye crear archivos, sobreescribirlos, remover datos de ellos.

## ISO, TEORÍA 5: FILESYSTEM

- Permiso de cambio de protección → el usuario puede modificar los derechos de acceso al archivo.
- 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).
  - Manejo de accesos simultáneos.
  - El esquema de protección:
    - El propietario/administrador del archivo debe ser capaz de controlar qué se puede hacer (derechos de acceso) y quién puede hacer.

## **ISO Tema 5, FILESYSTEM**

### **Metas del Sistema de archivos**

- Brindar espacio en disco a los archivos del usuario y del sistema.
- Mantener un registro del espacio libre. Cantidad de espacio libre y donde se ubica dentro del disco.

### **Conceptos**

- Sector
  - Unidad de almacenamiento utilizada en los discos rígidos.
- Bloque o Cluster
  - Conjuntos de sectores consecutivos.
  - Nos sirve para archivos que necesitan más de un sector.
- File system
  - Define la forma en que los datos se almacenan.
- FAT: File Allocation Table
  - Tabla que contiene información sobre en que lugar están alojados los distintos archivos.
- Se puede provocar fragmentación interna en un disco.

### **Pre-asignación: forma de asignar espacio**

- Es necesario saber cuánto espacio va a ocupar el archivo en el momento de su creación.
- Se tiende a definir espacios mucho más grandes que lo necesario.
- Es posible usar sectores contiguos para almacenar los datos de un archivo.
- Y si el archivo supera el espacio que le asignaron?
- Asignación continua aplica la pre-asignación.

### **Asignación dinámica: forma de asignar espacio**

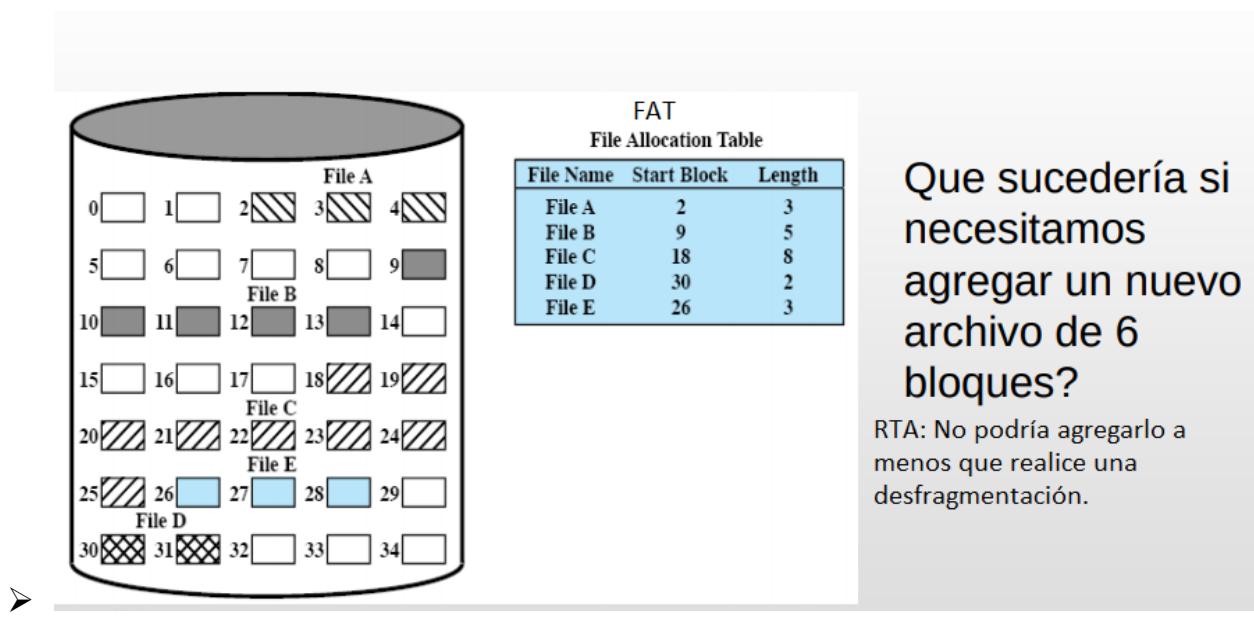
- El espacio es solicitado a medida que se necesita.
  - Los bloques de datos pueden quedar de forma no contigua.
  - La tendencia de hoy día es para asignaciones dinámicas.
- 
-

## ISO Tema 5, FILESYSTEM

Veremos formas de asignación

### Forma continua

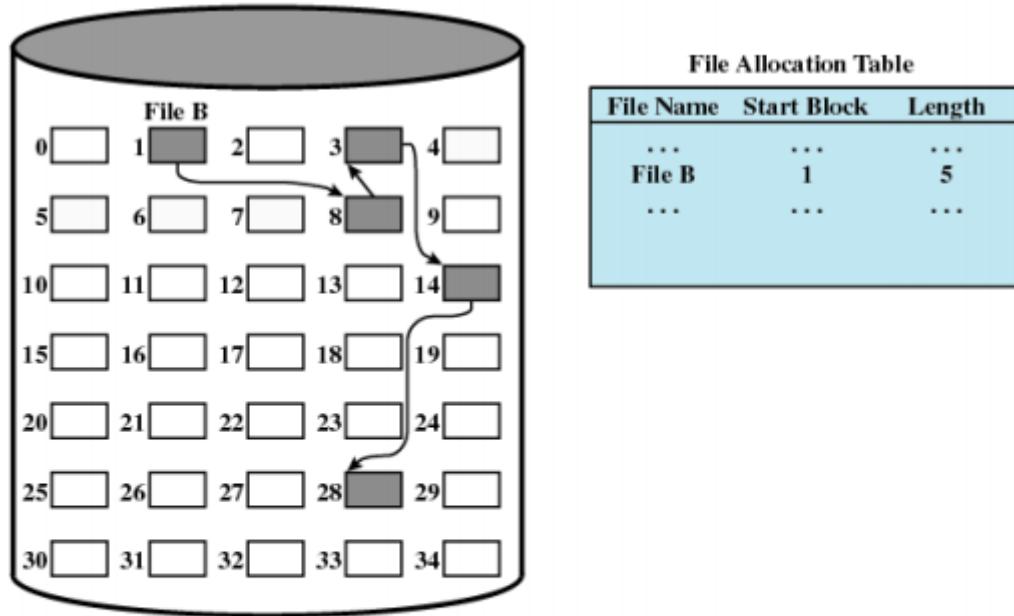
- Para guardar un archivo se utiliza un conjunto contiguo de bloques.
- Es necesario preasignar, ósea conocer el tamaño del archivo al ser creado.
- El formato de la FAT es simple, ya que utiliza una sola entrada que incluye el bloque en el cuál empieza el archivo y la longitud (bloques usados).
- El archivo puede ser leído con una única operación.
- Puede existir fragmentación externa que se puede resolver compactando aunque esto es MUY costoso.
- La técnica presenta problemas:
  - Encontrar bloques libres continuos en el disco (y que tengan el tamaño necesario).
  - El incremento del tamaño de un archivo.



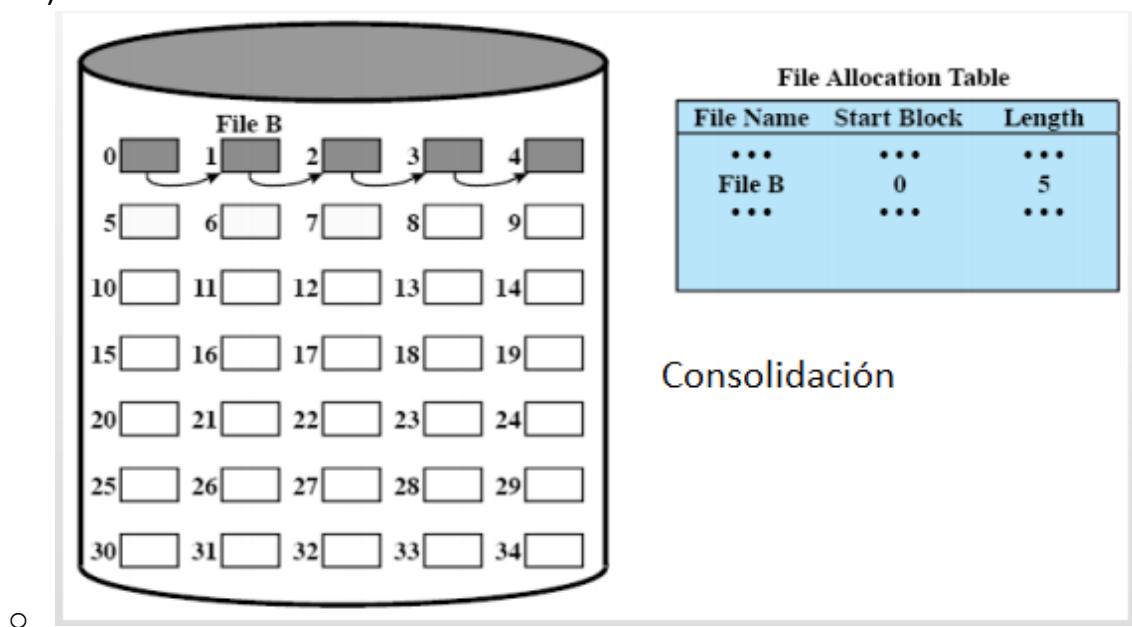
### Forma encadenada

- Se asigna en base a bloques individuales.
- Cada bloque tiene un puntero a la ubicación del próximo bloque del archivo → se debe implementar una estructura de enlaces.
- FAT tiene una única entrada, bloque inicial y tamaño del archivo.
- No hay fragmentación externa al usar esta forma.
- Útil para acceso secuencial.
- Los archivos pueden crecer bajo demanda (asignación dinámica).
- No se requieren bloques contiguos.

## ISO Tema 5, FILESYSTEM



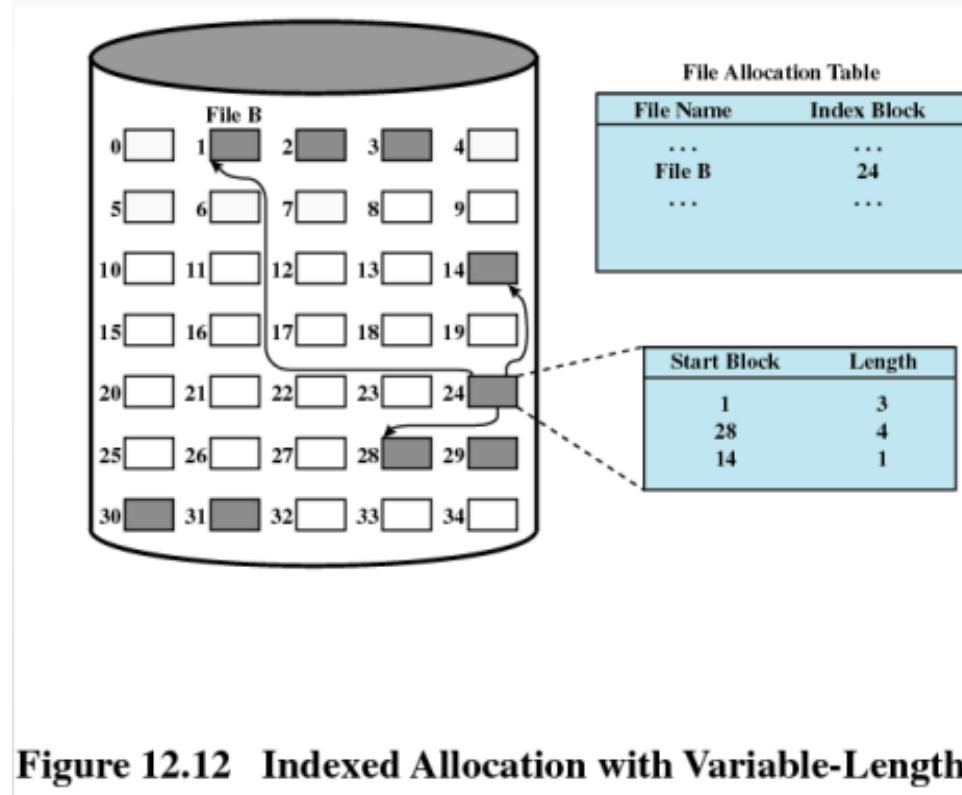
- **Figure 12.9 Chained Allocation**
- Es posible mover los bloques de un mismo archivo de forma que queden contiguamente, esto ayuda a que haya una cercanía de bloques para agilizar operaciones a la hora de leer el disco (desfragmentadores de discos).



## ISO Tema 5, FILESYSTEM

### Forma indexada:

- Se asigna en base a bloques individuales.
- No se produce fragmentación externa.
- El acceso random a un archivo se vuelve eficiente.
- FAT
  - Una sola entrada con la dirección del bloque de índices (puntero).
  - El bloque índice NO contiene datos del archivo, sino que contiene una lista de punteros a los bloques que componen el archivo.  
(índice)
- Tiene dos variantes
- **ASIGNACIÓN POR SECCIONES**
  - A cada entrada del bloque índice se agrega el campo longitud.
  - Cada índice del bloque índice apunta al primer bloque de un conjunto que se almacena sí o sí de manera contigua.

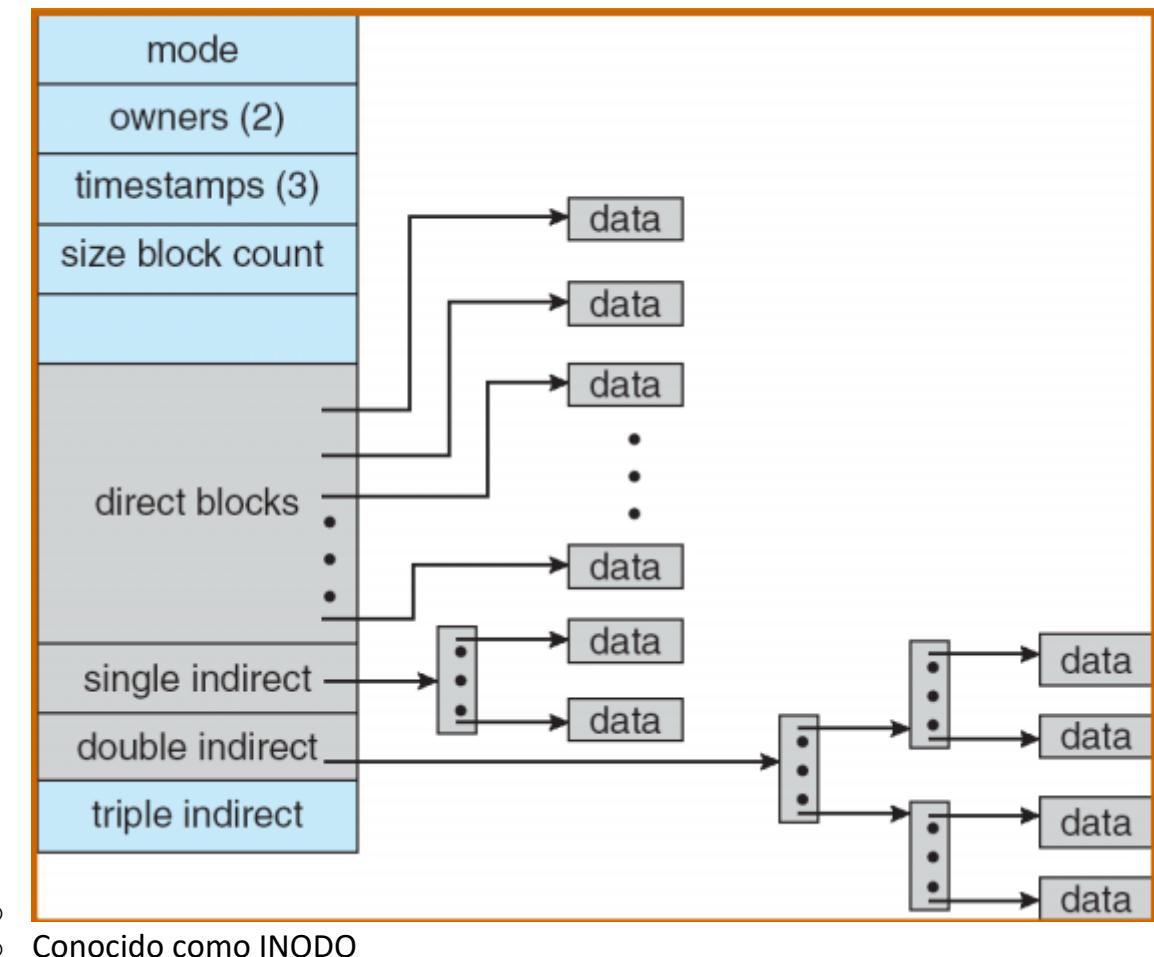


**Figure 12.12 Indexed Allocation with Variable-Length Portions**

- **NIVELES DE INDIRECCIÓN**
  - Existen bloques directos con datos guardados dentro.
  - Otros bloques son considerados como bloque índice (apuntan a varios bloques de datos).
    - Direcciónamiento indirecto simple: un bloque índice solo que apunta a los bloques de datos.

## ISO Tema 5, FILESYSTEM

- Direccionamiento indirecto doble: un bloque índice que apunta a otro bloque índice y este último apunta a los bloques de datos.
- Pueden haber varios niveles de indirección



### Gestión de espacio libre

- Consiste en el control sobre cuáles de los bloques de disco están disponibles.
- Tres alternativas:
  - Tablas de bits.
  - Bloques libres encadenados.
  - Indexación.

### Tablas de bits

- Vector con 1 bit por CADA bloque de disco.
  - 0 = bloque libre, 1 = bloque ocupado.
- Ventaja
  - Es fácil hallar un bloque o grupo de bloques que estén libres.

## ISO Tema 5, FILESYSTEM

- Desventaja
  - Tamaño que llega a ocupar el vector en la memoria.
    - Se calcula: tamaño del disco bytes / tamaño bloque en sistema de archivos.
    - Ej: Disco de 16 GB con bloques de 512 bytes cada uno → el vector pesa 32 MB.

### ✓ Ejemplo

00111

00001

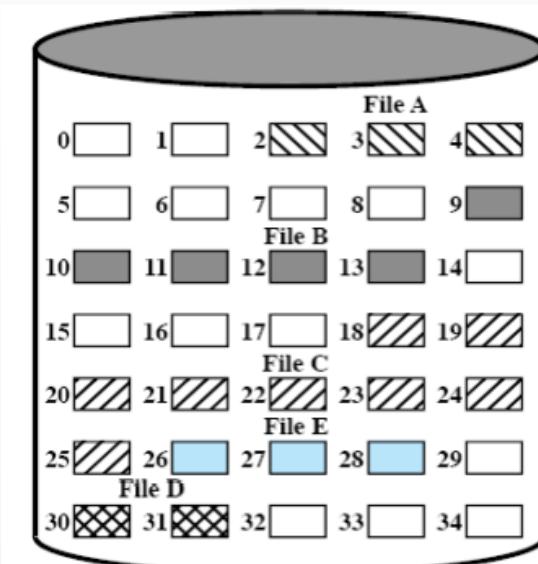
11110

00011

11111

11110

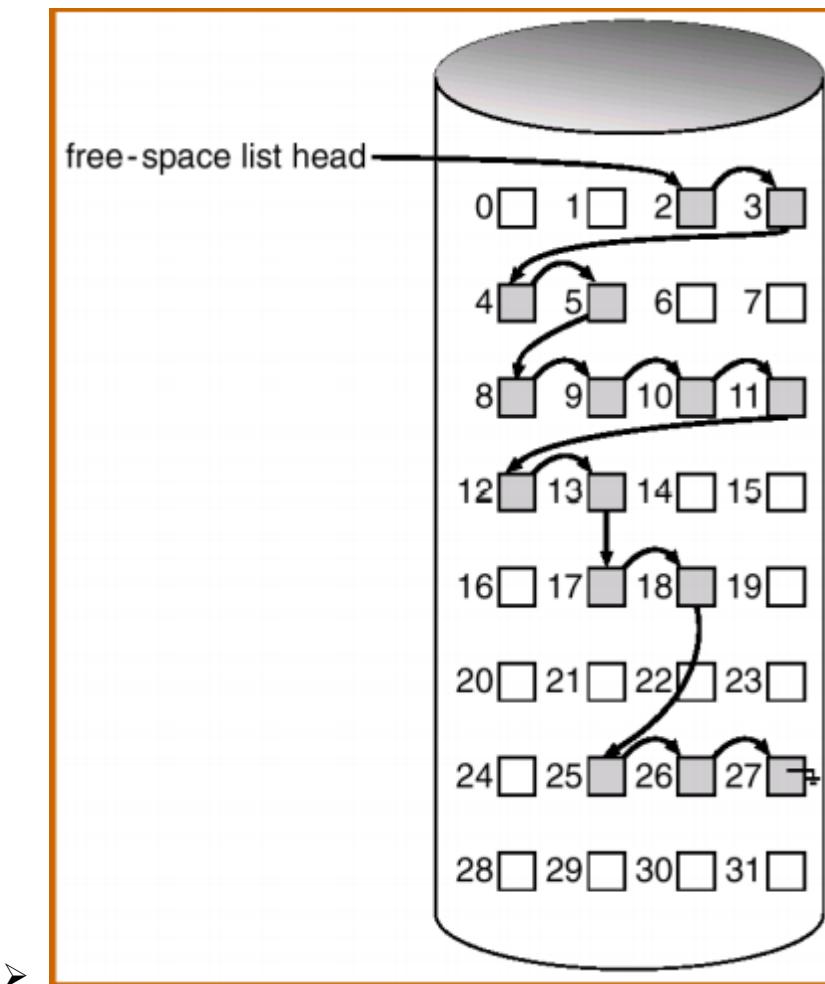
11000



### Bloques encadenados

- Se tiene un puntero al primer bloque que esté libre.
- Cada bloque libre tiene un puntero al siguiente bloque libre.
- Resulta ser ineficiente para la búsqueda de bloques libres ya que hay que realizar varias operaciones de E/S para obtener un grupo libre.
- Si se pierde un enlace se generan problemas.
- Es difícil hallar bloques libres consecutivos.

## ISO Tema 5, FILESYSTEM



### Indexación (agrupamiento)

- Variante de bloques libres encadenados.
- El primer bloque libre contiene las direcciones de N bloques libres.
- Las N-1 primeras direcciones son bloques libres.
- La N-ésima dirección referencia otro bloque con N direcciones de bloques libres.

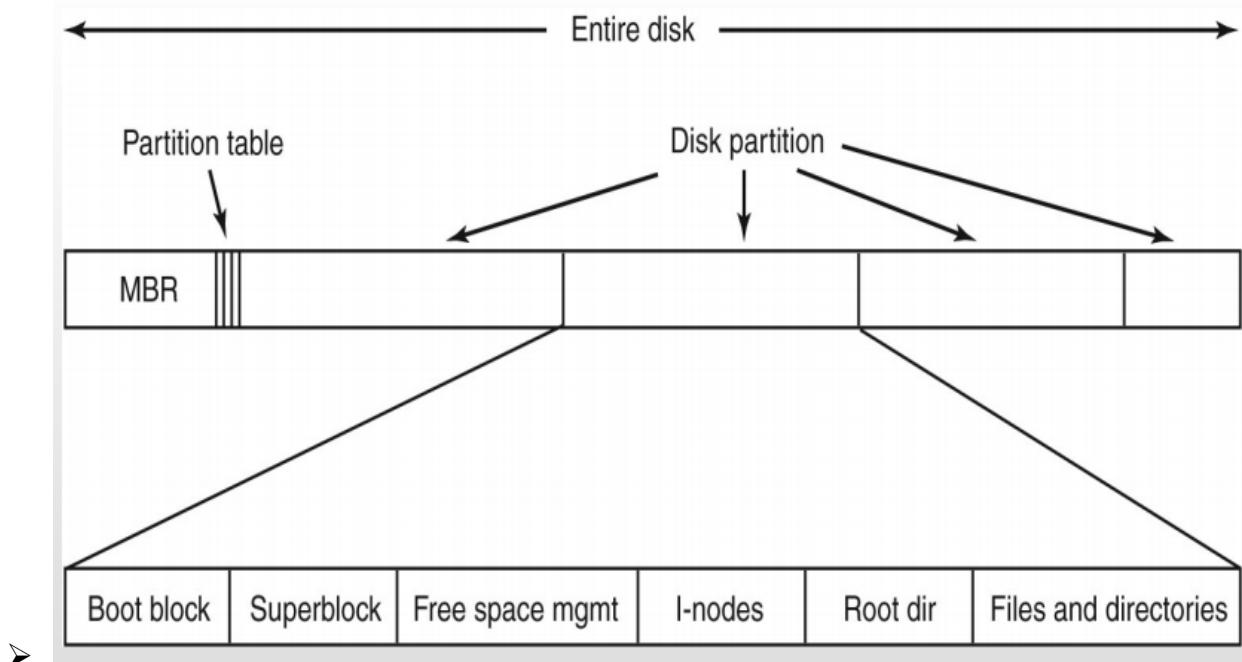
## ISO TEMA 5- FILESYSTEM

### Unix – Manejo de archivos

- Tipos de archivos
  - Archivo común.
  - Directorio.
  - Archivos especiales (dispositivos).
  - Named pipes (comunicación entre procesos).
  - Links (comparten el i-nodo, solo dentro del filesystem).
  - Links simbólicos (como si fueran accesos directos, tiene i-nodo propio, para filesystems diferentes).

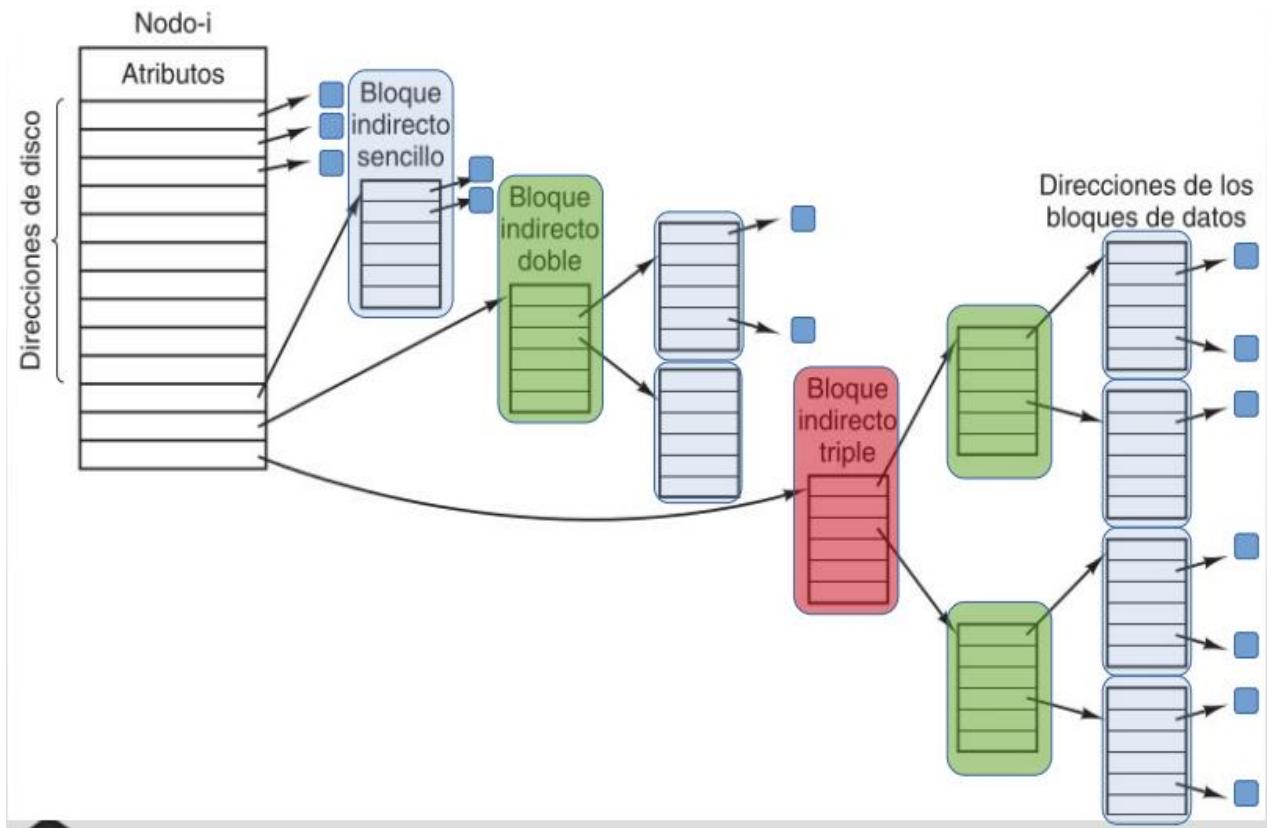
### Unix – Estructura del volumen

- Cada disco físico puede ser dividido en uno o más volúmenes (particiones). Cada volumen contiene un filesystem. Cada filesystem contiene.
  - Boot block: código para bootear el S.O
  - Superblock: atributos sobre el filesystem que incluyen qué bloques/clusters están libres.
  - Tabla de Inodos: tabla que contiene todos los inodos.
    - Inodo: estructura de control que contiene la información clave de un archivo.
  - Bloques de datos: de los archivos.

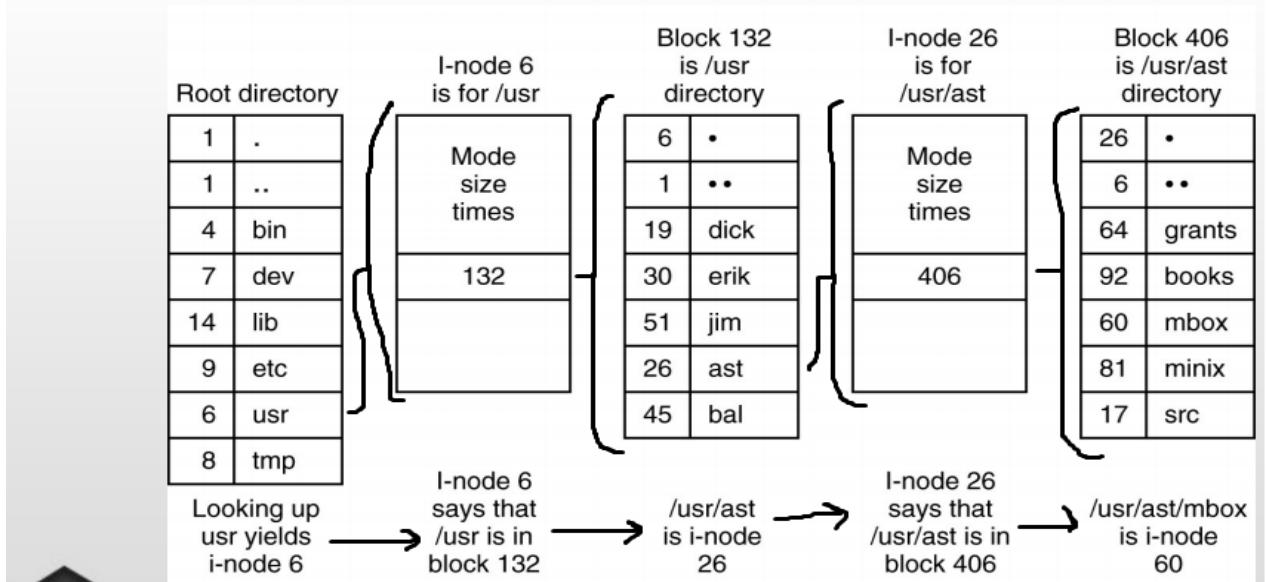


### Unix - INODO (Como se ve y un ejemplo de búsqueda)

## ISO TEMA 5- FILESYSTEM



Buscar el i-nodo del archivo /usr/ast/mbox



Las tablas que tienen "Mode size times" especifican en que bloque se encuentra la información.

## ISO TEMA 5- FILESYSTEM

### Windows – File Systems soportados

- CD-ROM filesystem (CDFS) → CD
  - Universal Disk Format (UDF) → DVD, Blu-Ray
  - File Allocation Table
    - FAT12
    - FAT16 → nombres cortos de archivos
    - FAT32 → nombres largos de archivos pero no soportados en MS-DOS
  - New Technology File System (NTFS)
- 

### Windows – FAT

- FAT es un sistema de archivos utilizado originalmente por DOS y Windows 9x
- Por qué Windows aun soporta FAT?
  - Proporciona compatibilidad con otro SO en sistemas multiboot.
  - Permite upgrades desde versiones anteriores.
  - Para formato de dispositivos como diskettes.
- Las distintas versiones de FAT son diferenciadas por un número que indica la cantidad de bits que son usados para identificar diferentes bloques o clusters.
  - FAT12
  - FAT16
  - FAT32
- Es utilizado un mapa de bloques del FILESYSTEM, llamado FAT.
- La FAT tiene cantidad de entradas = cantidad de bloques.
- La FAT, su duplicado, y el directorio raíz se almacenan en los primeros sectores de la partición.



- En FAT se utiliza la asignación encadenada.
- El puntero al próximo bloque está en la FAT y no en los bloques.
- Los bloques libres y dañados tienen códigos especiales.

### Windows – FAT12

- FAT12 utilizan 12 bits para la identificación del sector, se limita a  $2^{12}$  (4096) sectores.
  - Windows utiliza tamaños de sector que van desde los 512 bytes hasta los 8kb, esto limita el tamaño total del volumen a 32 MB → 4096 sectores \* 8 KB

## **ISO TEMA 5- FILESYSTEM**

- FAT12 se usa como filesystem para diskettes de 3,5 y 12 pulgadas que son capaces de almacenar hasta 1,44 MB de datos (probablemente lo que pesen 3 o 4 resumenes mios de la materia).

### **Windows – FAT16**

- FAT16 usa 16 bits para identificar cada sector. Puede haber  $2^{16}(65.536)$  sectores en un volumen.
  - El tamaño del sector varía entre 512 bytes hasta los 64 KB lo que limita el tamaño máximo de la partición a 4 GB (64 KB \* 65.536)
  - El tamaño de sector dependía del tamaño de la partición al formatearla.

### **Windows – FAT32**

- FAT32 fue el filesystem más reciente de la línea esa (posteriormente salió exFAT que algunos lo conocen como FAT64).
- FAT32 usa 32 bits para la identificación de sectores, PERO reserva los 4 bits superiores. Efectivamente se usan 28 bits para identificar.
  - El tamaño de sector en FAT32 puede ser de hasta 32KB con lo cual tiene una capacidad teórica de direccionar particiones de hasta 8 TB.
  - El modo de identificación y acceso a los sectores lo hace de forma más eficiente que FAT16. Con tamaño de sector de 512 bytes, puede direccionar volúmenes de hasta 128 gb.

### **Windows – NTFS**

- NTFS es el filesystem nativo de Windows desde Windows NT.
- Usa 64 bits para referenciar sectores.
  - Teóricamente permite tener volúmenes de hasta 16 Exabytes (16 billones de GB).
- Porqué usar NTFS en lugar de FAT?. A pesar de que FAT es simple y más rápido para ciertas operaciones, NTFS soporta:
  - Tamaños de archivo y de disco mayores.
  - Mejora performance en discos grandes.
  - Nombres de archivos de hasta 255 caracteres.
  - Atributos de seguridad.
  - Es transaccional.

## Resumen ISO → Tema IO( entrada/salida)

### Responsabilidades del SO

- Controlar dispositivos de E/S
  - Generar comandos.
  - Manejar interrupciones.
  - Manejar errores.
- Proporcionar una interfaz para utilizar los dispositivos de E/S

### Problemas en E/S

- Dispositivos diferentes entre sí.
  - Características únicas de cada uno.
  - Velocidad de cada dispositivo (variante).
  - Surgen nuevos tipos de dispositivos que el SO debe adaptar para utilizarlos (siempre y cuando no se haga un cambio muy fuerte del diseño del SO).
  - Existen diferentes formas de realizar E/S (ver anexo).
- 

### Aspectos de los dispositivos de E/S

- Unidad de transferencia:
    - Dispositivos que transfieren bloques (discos):
      - Operaciones → Read, Write, Seek
    - Dispositivos que transfieren caracteres (teclados, mouse, serial ports):
      - Operaciones → Get, Put.
  - Forma de acceder al dispositivo:
    - Secuencialmente.
    - Aleatoriamente.
  - Tipo de acceso al dispositivo:
    - Acceso compartido (más de un proceso usando el mismo dispositivo): Disco rígido.
    - Acceso exclusivo (un solo proceso por vez): Impresora.
  - Tipo de operaciones:
    - Read only: CDROM.
    - Write only: Pantalla.
    - Read/write: Disco.
- 

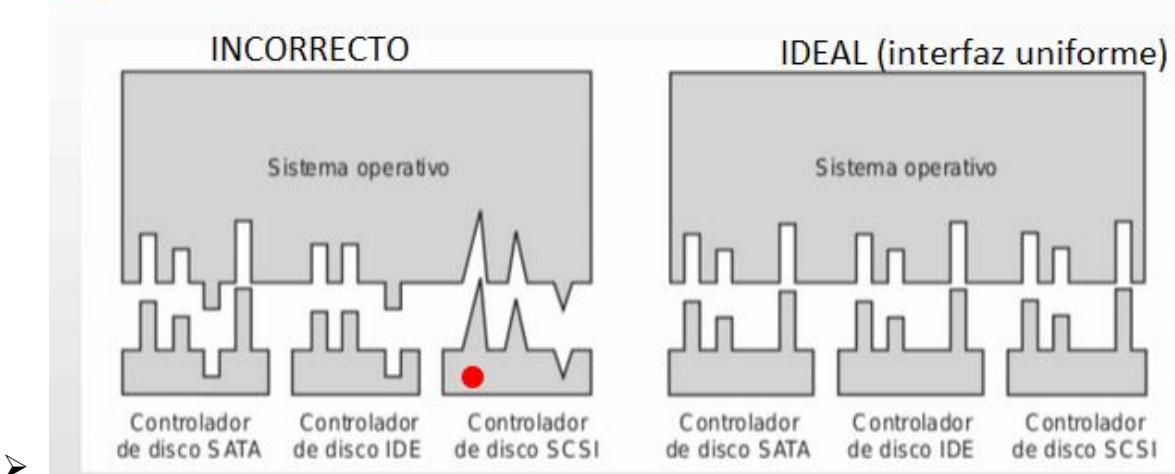
### Metas, objetivos y servicios

- Generalidad:
  - Es deseable manejar todos los dispositivos de una manera estandarizada (general).

## Resumen ISO → Tema IO( entrada/salida)

- Se deben ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles más “bajos” de forma que los procesos vean a los dispositivos en forma de operaciones comunes → read, write, open, close, lock, unlock.

## ☒ Interfaz Uniforme



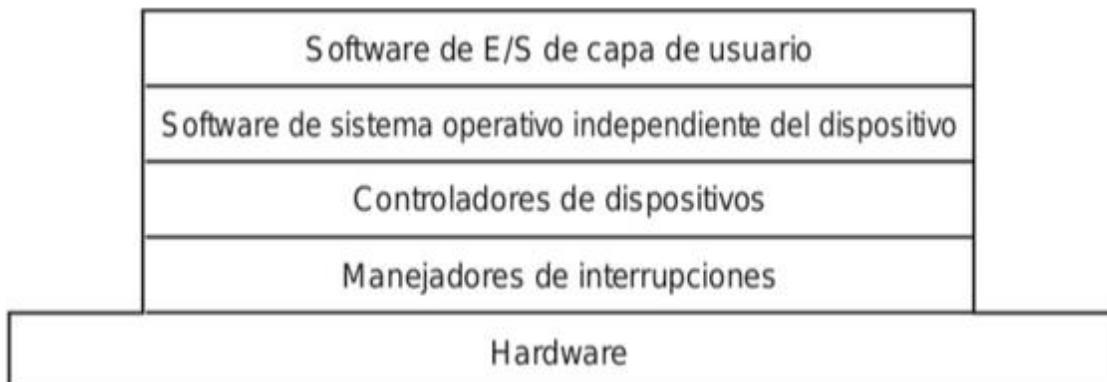
- Eficiencia:
  - Los dispositivos de E/S son mucho más lentos respecto a la memoria RAM y la CPU.
  - Aprovechando la idea de la multi-programación permite que un proceso espere por la finalización de su E/S mientras que otro proceso se ejecuta (no desperdiciamos uso de CPU)
- Planificación:
  - Se deben organizar los requisitos de los dispositivos.
  - Ejemplo: planificar requerimientos a disco para minimizar tiempos.
- Buffering: almacenamiento de los datos en memoria mientras se transfieren.
  - Soluciona problemas de velocidad entre todos los dispositivos.
  - Soluciona problemas de tamaño y/o forma de los datos entre los dispositivos.
- Caching: se mantiene en memoria una copia de los datos de reciente acceso para mejorar la performance.
- Spooling: administrar la cola de requerimientos de un dispositivo.
  - Mecanismo que arma una cola para coordinar el acceso concurrente a un dispositivo.
  - Esto se usa ya que algunos dispositivos de acceso **exclusivo**, no pueden atender distintos requerimientos al mismo tiempo. Por ej: impresora.
- Reserva de dispositivos: para dispositivos de acceso exclusivo.

## Resumen ISO → Tema IO( entrada/salida)

- Manejo de errores:
  - El S.O administra los errores ocurridos(lectura de un disco, dispositivo no disponible, errores de escritura).
  - La mayoría retorna un número de error o código cuando la I/O falla.
  - Logs de errores.
- Formas de realizar E/S:
  - Bloqueante: el proceso es suspendido hasta que el requerimiento de E/S es completado.
    - Fácil de usar y entender.
    - No es suficiente bajo algunas necesidades.
  - No bloqueante: el requerimiento de E/S retorna en cuanto es posible (el proceso sigue ejecutándose).
    - Ejemplo: aplicación de video que lee frames desde un archivo mientras va mostrándolo en pantalla.

---

**Diseño del software:** (diseño de programación en capas, cada capa tiene objetivos muy específicos. La implementación de cada capa es para sí misma → se ocultan detalles de implementación)



### Software de capa de usuario (la única en modo usuario)

- Tiene un conjunto de librerías de funciones que no requieren ejecución en modo kernel.
  - Permiten acceso a SysCalls.
  - Implementan servicios que no dependen del kernel.
- Procesos de apoyo.
  - Demonio de impresión (realiza el spooling): la cola se maneja por un proceso en **modo usuario**.

### Software de SO independiente del dispositivo (ya en modo kernel)

## Resumen ISO → Tema IO( entrada/salida)

- Brinda los principales servicios de E/S antes vistos.
  - Interfaz uniforme.
  - Manejos de errores.
  - Buffering de datos.
  - Asignación de recursos.
  - Planificación.
- Es la capa independiente de los dispositivos → brinda la generalidad.
  - Explotable en sub-capas para realizar un diseño más modular.
- El kernel debe mantener la información de estado de cada dispositivo o componente.
  - Archivos abiertos.
  - Conexiones de red.
  - Etc.
- Hay varias estructuras complejas que representan buffers, utilización de la memoria, disco, etc.

## Controladores de dispositivos (drivers)

- Los drivers contienen el código dependiente del dispositivo.
- Manejan un tipo dispositivo.
- Traducen los requerimientos abstractos recibidos de capas más altas en los comandos necesarios para el dispositivo.
  - Escribe sobre los registros del controlador.
  - Acceso a la memoria mapeada.
  - Encola requerimientos.
- Las interrupciones generadas por los dispositivos son atendidas por funciones provistas por el driver.
- En resumen, es el que sabe como manejar el dispositivo.
- Constituye una interfaz entre el SO y el Hardware.
- Los drivers forman parte del espacio de memoria del kernel.
  - Se cargan como 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.
  - El kernel es modular → es decir tiene su parte principal y se le puede agregar módulos (drivers).
- Ejemplo en **LINUX**:

Linux distingue 3 tipos de dispositivos

- ✓ Carácter: I/O programada o por interrupciones
- ✓ Bloque: DMA
- ✓ Red: Ports de comunicaciones

Los Drivers se implementan como módulos

- ✓ Se cargan dinámicamente

Debe tener al menos estas operaciones:

- ✓ `init_module`: Para instalarlo
- ✓ `cleanup_module`: Para desinstalarlo.

➤

Operaciones que debe contener para I/O

- ✓ `open`: abre el dispositivo
- ✓ `release`: cerrar el dispositivo
- ✓ `read`: leer bytes del dispositivo
- ✓ `write`: escribir bytes en el dispositivo
- ✓ `ioctl`: orden de control sobre el dispositivo

➤

Otras operaciones menos comunes

- ✓ `llseek`: posicionar el puntero de lectura/escritura
- ✓ `flush`: volcar los búferes al dispositivo
- ✓ `poll`: preguntar si se puede leer o escribir
- ✓ `mmap`: mapear el dispositivo en memoria
- ✓ `fsync`: sincronizar el dispositivo
- ✓ `fasync`: notificación de operación asíncrona
- ✓ `lock`: reservar el dispositivo
- ✓ .....

➤

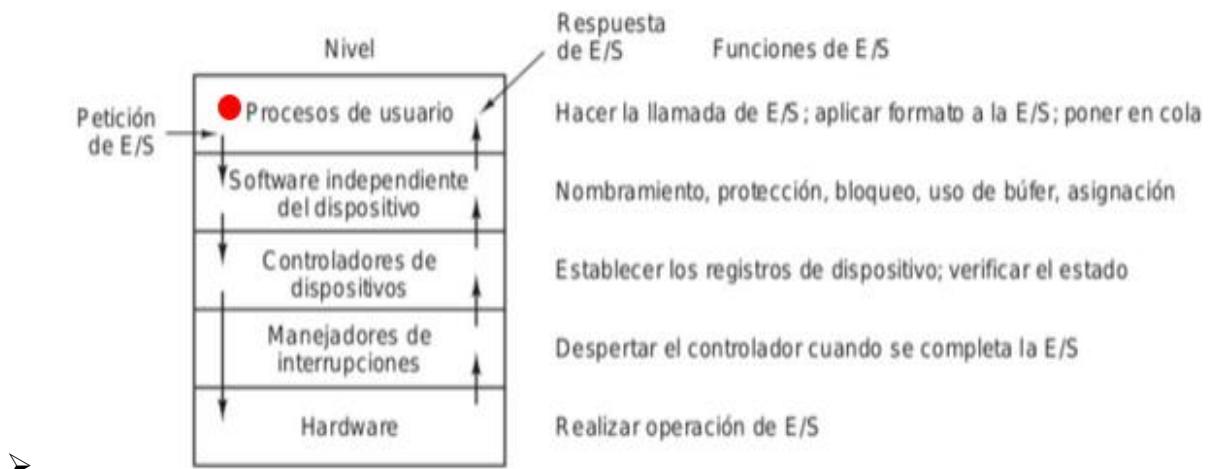
## Resumen ISO → Tema IO( entrada/salida)

- ☒ Por convención, los nombres de las operaciones comienzan con el nombre del dispositivo
- ☒ Por ejemplo, para `/dev/ptr`

```
int ptr_open (struct inode *inode, struct file *filp)
void ptr_release (struct inode *inode, struct file *filp)
ssize_t ptr_read (struct file *filp, char *buff,
                  size_t count, loff_t *offp)
ssize_t ptr_write (struct file *filp, const char *buff,
                   size_t count, loff_t *offp)
int ptr_ioctl (struct inode *inode, struct file *filp,
               unsigned int cmd, unsigned long arg)
```

## Manejadores de interrupciones (gestor)

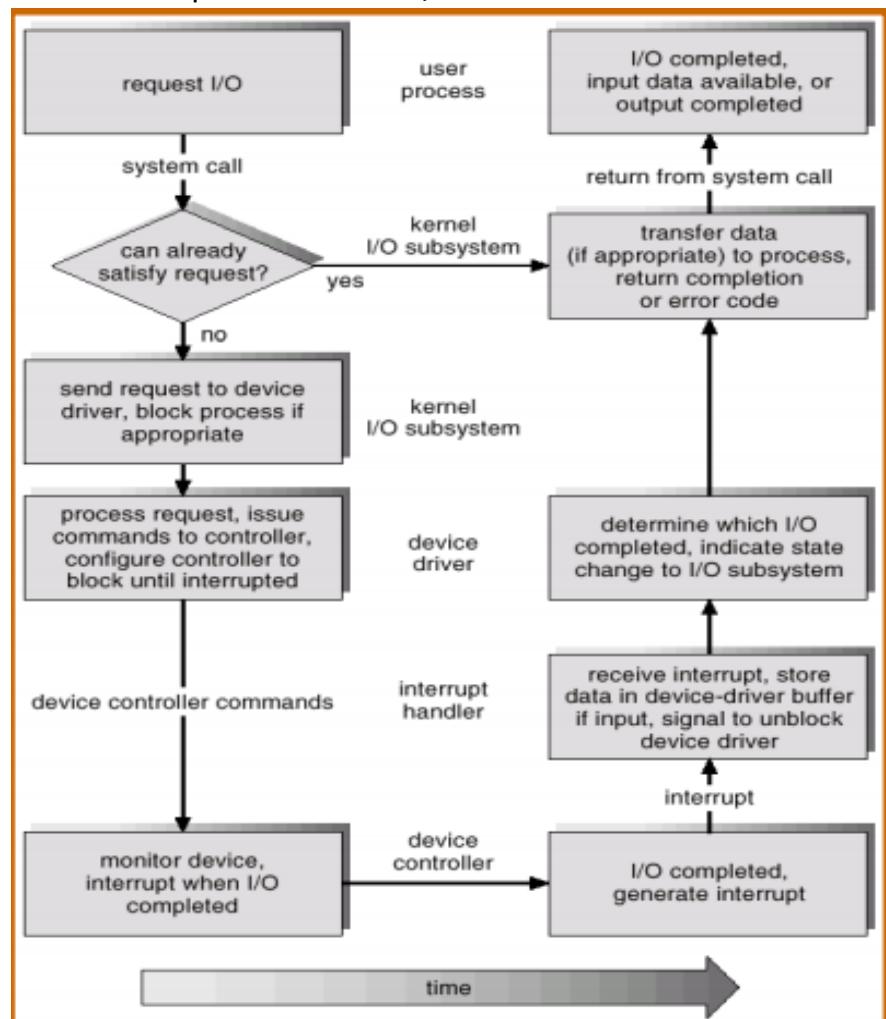
- Atiende todas las interrupciones del HW.
- Deriva al driver correspondiente según interrupción.
- Resguarda información.
- Independiente del driver.
- Puente entre interrupciones y driver que las resuelve.
- Ciclo de atención de un requerimiento:



- - Ciclo explicado:
    - Se determina el dispositivo que almacena los datos.
      - Se traduce el nombre del archivo en la representación del dispositivo.
    - Se traduce el requerimiento abstracto en bloques de discos del filesystem.
    - Se realiza la lectura física de los datos en la memoria.

## Resumen ISO → Tema IO( entrada/salida)

- Se marcan los datos como disponibles al proceso que realizó el requerimiento.
  - Se desbloquea el proceso.
- Se retorna el control al proceso.
- Ciclo de vida del requerimiento de E/S



## Performance

- La I/O es uno de los factores que más afectan a la performance del sistema.
  - Utilizan mucho 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) – controlador

## **Resumen ISO → Tema IO( entrada/salida)**

### **Como mejoro la performance**

- Reducir el número de context switches.
- Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación (los datos se pasan entre capas).
- Reducir la frecuencia de las interrupciones:
  - Transfiriendo gran cantidad de datos de un saque.
  - Controladores más inteligentes.
  - Polling.
- Utilizar DMA → resuelve las interrupciones, sin tener que usar CPU.

## ISO TEMA 6 – BUFFER CACHE

### 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 cache 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 cuál 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 cache 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.

# ISO TEMA 6 – BUFFER CACHE

## Header

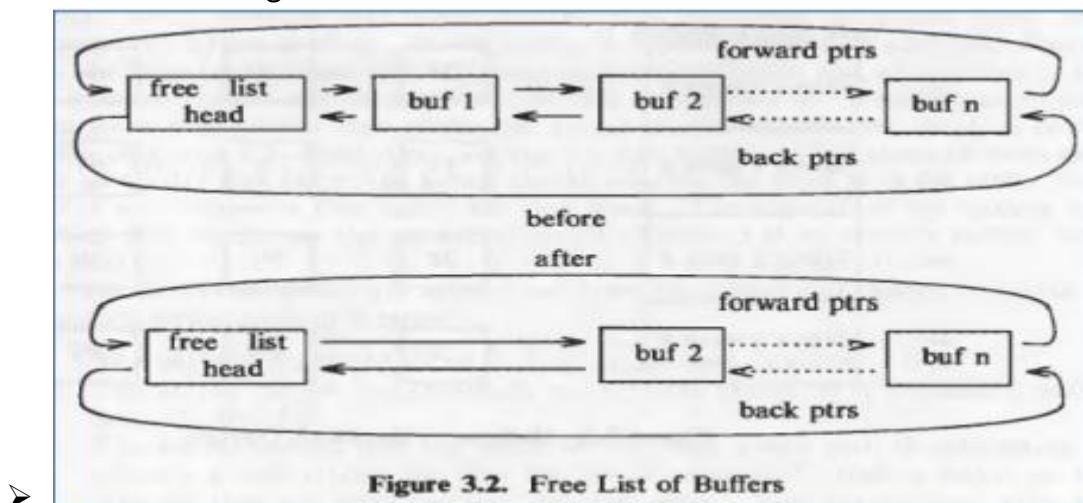
- Nro de dispositivo, nro de bloque.
- Estado.
- Punteros:
  - 2 punteros de hash queue.
  - 2 punteros para la free list.
  - 1 puntero al bloque en memoria (donde se ubica).

## 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.

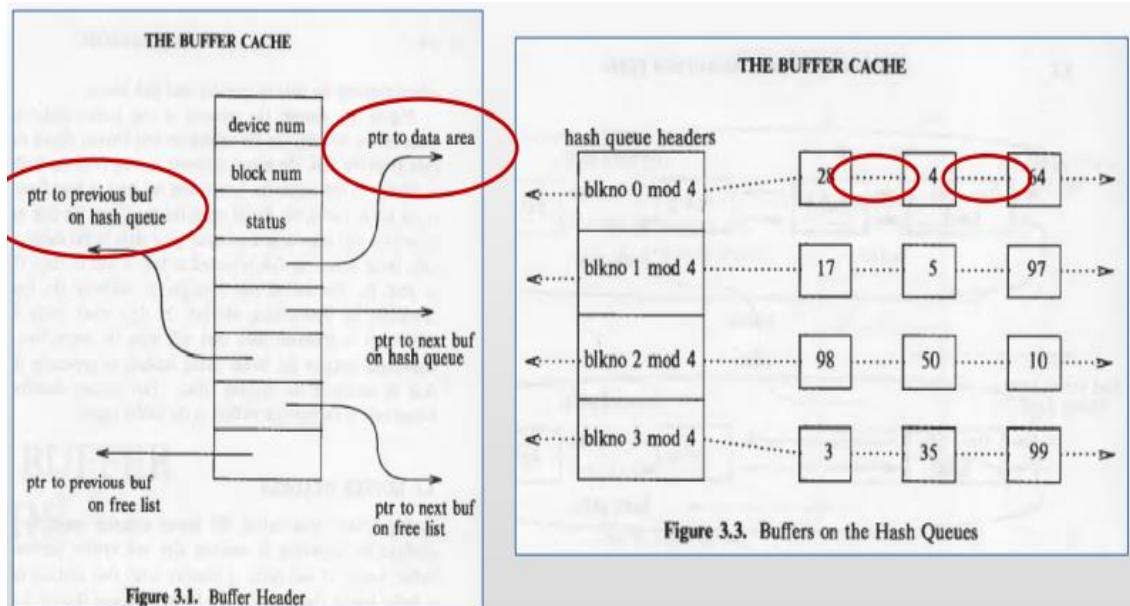


## Hash queues

- 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.

## ISO TEMA 6 – BUFFER CACHE

- La agrupación y enganche de cada buffer con otro en cada cola se realiza mediante los punteros que se almacenan en el header



### Free list (continuando)

- 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.