Memoria – Tema 3

**Repaso:**

Con paginación vimos que el espacio de direcciones de un proceso no necesariamente debe estar “contiguo” en la memoria para poder ejecutarse.

El hardware traduce las direcciones lógicas a físicas utilizando las tablas de páginas que el Sistema Operativo administra.

**Memoria Virtual**

Se puede pensar también que, no todo el espacio de direcciones de proceso es necesario en todo momento. Por ejemplo, existen rutinas o librerías que se ejecutan una única vez (o nunca), partes del programa que no vuelven a ejecutarse, etc.

**¿Soluciones?**

El SO puede traer a memoria las “piezas” de un proceso a medida que este las necesita**,** se le define como “**Conjunto Residente”** a la porción del espacio de direcciones de proceso que se encuentra en memoria. También llamado “Working Set”

Trabajando con el apoyo de hardware, se trae una nueva porción a memoria cuando se necesita una que no se encuentra en el conjunto residente.

**Ventajas:**

* Más procesos pueden ser mantenidos en Memoria
  + Solo son cargadas algunas secciones de cada proceso
  + Con más procesos en memoria principal más probable que existan más procesos Ready
* Un proceso puede ser más grande que la memoria principal
  + El usuario no se debe preocupar por el tamaño de sus programas
  + La limitación la impone el hardware y el bus de direcciones

**¿Qué hace falta para la Memoria Virtual?**

* El hardware **debe** soportar paginación por demanda (y/o segmentación)
* 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)
* El SO debe ser capaz de manejar el movimiento de las páginas (o segmentos) entre la memoria principal y la secundaria.

**Memoria virtual + Paginación**

* Cada proceso tiene su tabla de páginas
* Cada entrada en la tabla referencia al frame (marco) en el que se encuentra la página en la memoria principal.
* Cada entrada en la tabla de páginas tiene bits de control
  + Bit V: indica si la página esta en memoria
  + Bit M: indica si la página fue modificada, si se modificó, estos cambios deben estar reflejados en memoria secundaria.

**Fallo de páginas (Page Fault)**

Ocurre cuando el proceso intenta usar una dirección que reside en una página que no se encuentra en la memoria principal. (Bit V = 0)

* La página no se encuentra en su conjunto residente
* el bit V es controlado por el hardware.

Para contrarrestar este suceso, el Hardware detecta la situación y genera un **trap** al SO, este podrá colocar al proceso en estado de “Blocked” (espera) mientras gestiona que la pagina que se necesita se cargue.

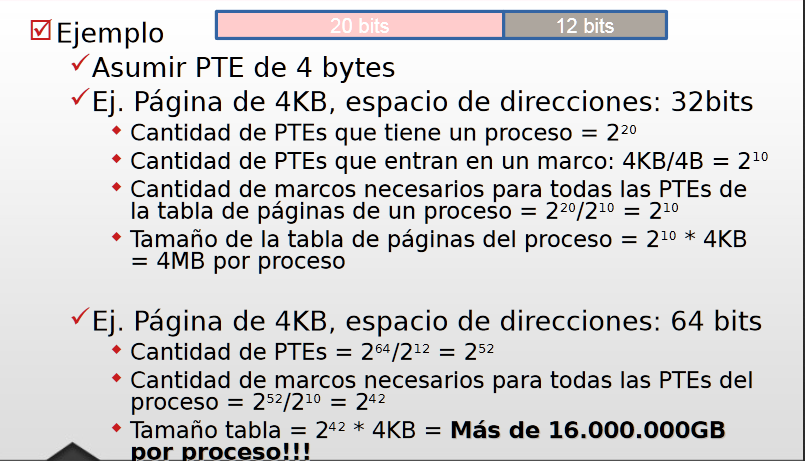
Para esto el SO busca un frame libre en la memoria y genera una operación de E/S al disco para copiar dicho frame en memoria principal. Mientras tanto el SO puede asignarle la CPU a otro proceso. Cuando la E/S finalice, informará al SO mediante una interrupción.

Cuando esto ocurre, el Sistema Operativo actualiza la tabla de páginas del proceso (Coloca el Bit V en 1 de la página en cuestión y también coloca la dirección base del frame (marco) donde se colocó la página)

El proceso que generó el Page Fault vuelve a estado de Ready, y cuando este se ejecute se volverá a ejecutar la instrucción que antes generó el fallo de página.

**Tabla de páginas**

Cada proceso tiene su tabla de páginas, el tamaño de la misma depende del espacio de direcciones del proceso. Esta puede llegar a alcanzar un tamaño considerable.



Existen distintas formas para organizar una tabla de páginas:

* Tabla de 1 nivel: Tabla única lineal
* Tabla de 2 niveles (o más, multinivel)
* Tabla invertida: Hashing

La forma de organizarla depende del Hardware subyacente.

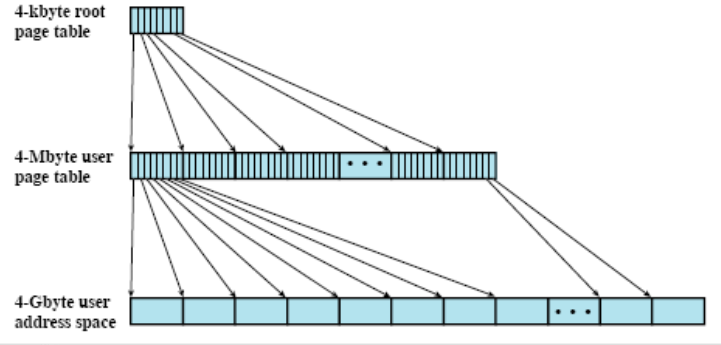


Tabla multinivel

**Tabla invertida:**

Se utiliza en arquitecturas donde el espacio de direcciones es muy grande. (Las tablas de páginas ocupan muchos niveles y la traducción es costosa)

Existe una entrada por cada frame, hay una sola tabla para todo el sistema.

El número de página es transformado en un valor de HASH, este se usa como índice de la tabla invertida para encontrar el marco asociado. Sólo se mantienen los **PTEs (**¿punteros?) de páginas presentes en memoria física.

Existe una tabla invertida organizada como tabla de hash en memoria principal.

* 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, page fault.

**Tamaños de Página:**

|  |  |
| --- | --- |
| **Pequeño** | **Grande** |
| * Menor fragmentación interna. * Más paginas requeridas por Proceso (Tablas de páginas más grandes) * Más páginas pueden residir en memoria | * Mayor fragmentación interna * La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente * Más rápido mover páginas hacia la memoria principal. |

**Translation lookaside Buffer**

Cada referencia en el espacio virtual puede causar 2 (o más) accesos a la memoria física.

* 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.

Esta 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.

En caso de que la entrada no se encuentra en la TLB (miss), el número de página es usado como índice en la tabla de páginas de proceso.

Se controla si la página está en memoria. (Si no está, se genera un page fault)

La TLB es actualizada para incluir la nueva entrada.

Resumiendo: Cuando se busca una página, primero se consulta en la TLB, luego en la memoria principal. Si hay doble miss, se genera Page fault.

**Asignación de marcos**

* ¿Cuántas páginas de un proceso se pueden encontrar en memoria?

Depende de los tamaños del conjunto residente.

* Tipos de asignación
  + Asignación Dinámica:
    - El número de marcos para cada proceso varía.
  + Asignación Fija:
    - Número fijo de marcos para cada proceso.

**Reemplazo de páginas**

* ¿Qué sucede si ocurre un page fault y todos los marcos están ocupados?

Se debe seleccionar una página victima

* ¿Cuál sería el reemplazo óptimo?

Es imposible de implementar, pero óptimo sería que la pagina a ser removida no sea referenciada en un futuro **próximo**.

La mayoría de las técnicas de reemplazo existentes predicen el comportamiento futuro mirando el comportamiento pasado.

**Algoritmos de reemplazo:**

1. **Optimo**
2. **FIFO**
3. **LRU (Least recently Used)**
4. **2da. Chance**
5. **NRU (Non recently Used)**
   1. **Utiliza bits R y M**
   2. **~R, ~M > ~R, M > R,~M > R, M**

**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 Working Set(conjunto residente)
  + No cambia la cantidad de frames asignados
  + El SO puede determinar cuál es la tasa de page-faults de cada proceso
  + Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.

**Memoria – Clase 4**

**Thrashing (Hiperpaginación)**

Decimos que 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 S.O 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)

**¿Cómo controlamos el Thrashing?**

Se lo puede limitar utilizando algoritmos de reemplazo local. Con estos algoritmos, si un proceso entra en thrashing no roba frames a otros procesos.

Si bien perjudica la performance del sistema, es controlable.

**¿Conclusión?**

Si un proceso cuenta con todos los frames que necesita, no **habría** thrashing. A continuación veremos algunas técnicas como la estrategia de Working Set, con el modeo de localidad y la estrategia de **PFF** (Frecuencia de Fallos de página)

**El Modelo de localidad**

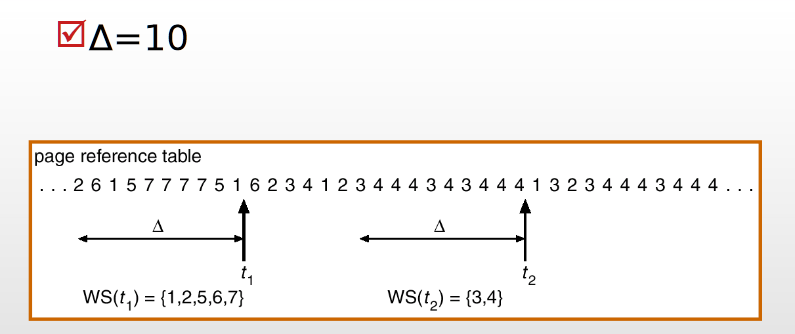
* Cercanía de referencias o principio de cercanía
* Las referencias a datos y programa dentro de un proceso tienden a agruparse
* La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento.
* En cortos períodos de tiempo el proceso necesitara pocas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

Un programa se compone de varias localidades. Por ejemplo: Cada rutina será una nueva localidad: se referencian sus direcciones (cercanas) cuando se está ejecutando.

Para prevenir la hiperactividad, 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 la más recientes Δ referencias a páginas.



**La selección del** Δ

* Δ chico: No cubrirá la localidad
* Δ grande: puede tomar varias localidades.

**Medida del working set**

* **m =**  cantidad de frames disponibles
* **WSS**i = medida del working set del proceso **p**i.
* **∑ WSS = D**
* D = demanda total de frames.
* Si D>m, habrá **thrashing.**

**Prevención de thrashing (parte dos)**

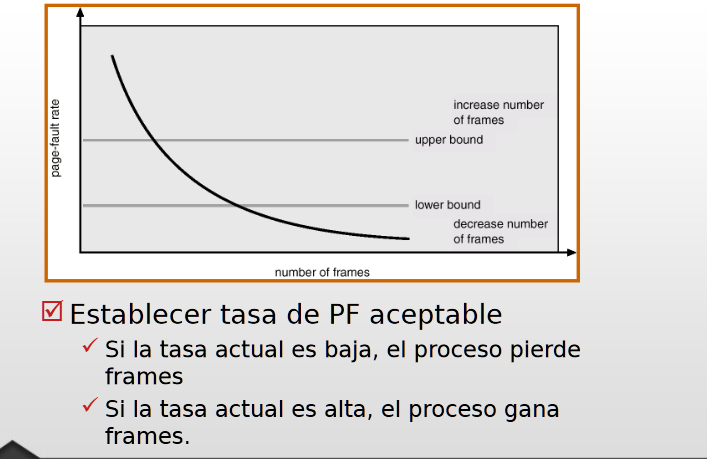
* SO monitorea cada proceso, dándole tantos frames hasta su WSS.
* Si quedan frames, puede iniciar otro proceso.
* Si D crece, excediendo m, se elige un proceso apra suspender, reasignándose sus frames… Así se mantiene alto el grado de multiprogramación utilizando el uso de la CPU.

**Problemas del modelo del Working Set**

* Mantener un registro de las medida del working set para cada proceso (WSS)
* La ventana es móvil.

**Prevencion del thrashing por PFF**

* **PFF alta,** se necesitan más frames
* **PFF baja,** los procesos tienen muchos frames asignados.

****

* Establecer límites superior e inferior de las PFF’s deseadas.
  + Excede PFF máx. entonces le doy un frame más.
  + Por debajo del PFF mínimo, entonces le saco un frame.
* Puede llegar a suspender un proceso si no hay más frames. Sus frames se reasignan a procesos de alta PFF.

**Demonio de paginación**

Proceso creador por el SO durante el arranque que apoya a la administración de la memoria. Este se ejecuta cuando el sistema tiene una baja utilización o algún parámetro de la memoria lo indica.

Estos parámetros pueden mandarse al tener, poca memoria libre, o mucha memoria modificada.

Tiene diferentes tareas:

* Limpia las páginas modificadas sincronizándolas con el swap.
* Reduce el tiempo de swap posteriormente ya que las páginas están “limpias”
* Puede sincronizar varias páginas contiguas reduciendo el tiempo total de transferencia
* Mantener el número de páginas libres en el sistema a un cierto número
* No liberarlas del todo hasta que haga falta realmente.

**En Linux proceso “kswapd”, en Windows, proceso “system”**

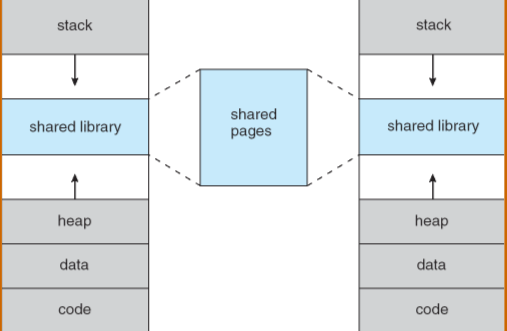
**Memoria compartida**

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.

***Codigo compartido***

Los procesos comparten una copia de código (solo lectura) por ej. Editores de texto, compiladores, etc. Los datos son privados a cada proceso y se encuentran en páginas no compartidas.



**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.

Si uno de ellos modifica una página compartida, la página es copiada.

COW permite crear procesos de forma más eficiente debido a que sólo las páginas modificadas son duplicadas.

**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 Page-fault es obtenido desde el archivo asociado. (No del área de intercambio)

Cuando el proceso termina o el archivo se libera, las paginas 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 librerías compartidas

**Area de Intercambio (swap)**

Sobre el área utilizada:

* Area dedicada, separada del sistema de Archivos(Por ejemplo, en Linux)
* Un archivo dentro del sistema de archivos (Por ejemplo, Windows)

**Técnicas para la administración**

* 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.
* 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 disco, ¿cómo podemos saber en qué parte del área de intercambio está?**

El PTE de dicha página tiene el bit V=0 y todos los demás bits sin usar.

**Area de intercambio – Linux**

Cada área es dividida en un número fijo de slots según el tamaño de la página.

Cuando una página es llevada a disco, Linux utiliza el PTE para almacenar 2 valores:

* El número de área
* El desplazamiento en el área (24 bits, lo que limita el tamaño máximo del área a 64 gb)

**Entrada Salida – Clase 1**

**Variedad en los dispositivos de E/S**

* Legible para el usuario
  + Usados para comunicarse con el usuario
    - Impresoras, Terminales: Pantalla, Teclado, Mouse
* Legible para la máquina
  + Utilizados para comunicarse con los componentes electrónicos
    - Discos, Cintas, sensores, etc.
* Comunicación
  + Utilizados para comunicarse con dispositivos remotos.
    - Líneas digitales, módems, interfaces de red, etc.

**Problemas que surgen:**

* Amplia variedad
  + Manejan diferentes cantidad de datos
  + En velocidades diferentes
  + En formatos diferentes
* La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM

**Hardware y software involucrado**

1. Buses
2. Controladores
3. Dispositivos
4. Puertos de E/S – Registros
5. Drivers
6. Comunicación con controlador del Dispositivo: E/S programada, Interrupciones, DMA

**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 E/S**

* CPU emite direcciones
  + Para identificar el dispositivo
* CPU emite comandos
  + Control
  + Test
  + Read/Write

**¿E/S mapeada o aislada?**

* **Correspondencia en memoria** (Memory mapped I/O)
  + Dispositivos y memoria comparten el espacio de direcciones
  + E/S es como escribir/leer en la memoria
  + No hay instrucciones especiales para E/S
    - Ya se dispone de muchas instrucciones para la memoria.
* **E/S aislada** (Isolated I/O)
  + Espacio separado de direcciones
  + Se necesitan líneas de E/S, puertos de E/S.
  + Instrucciones especiales
    - Conjunto limitado.

**Problemas:**

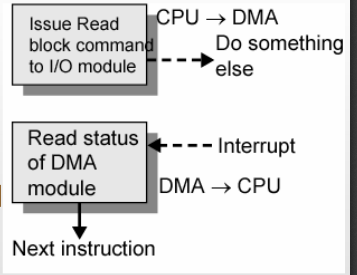
* + La CPU tiene control Directo sobre la E/S
    - Controla estado
    - Comandos para leer/escribir
    - Transfiere los datos
  + La CPU espera que el componente de E/S complete la operación
  + Se desperdician ciclos de CPU.
  + Es necesario hacer polling (consultar estado del dispositivo E/S)
    - Ciclo de “busy-wait” para realizar la E/S
    - Puede ser muy costoso si la espera es muy larga.

**Solucion:**

* **Agregar interrupciones**
  + Soluciona el problema de la espera de la CPU
  + La CPU no repite el chequeo sobre el dispositivo (polling)
  + El procesador continua la ejecución de instrucciones
  + El componente de I/O envía una interrupción cuando termina.

**DMA**

**Direct Memory Acces,** es un componente que controla el intercambio de datos entre la memoria principal y el dispositivo. El procesador es interrumpido luego de que el bloque entero fue transferido.



**Interfaz de E/S – Metas**

* Generalidad
  + Es deseable manejar todos los dispositivos de E/S de una manera uniforme, estandarizada
  + Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles “bajos”, para que los procesos vean a los dispositivos en términos de operaciones comunes como: read, write, open, close, lock, unlock.
* Eficiencia
  + Los dispositivos de E/S pueden resultar extremadamente lentos respecto a la memoria
  + El uso de la multiprogramación permite que los procesos esperen por la finalización de la E/S mientras que otro se ejecuta.
  + E/S no puede alcanzar la velocidad de la CPU.

**Aspectos de los dispositivos de E/S**

* Unidad de Transferencia
  + Dispositivos por bloques(discos):
    - Operaciones: Read, write, seek
  + Dispositivos por Carácter (keyboards, ouse, serial ports)
    - Operaciones: get, put
* Formas de acceso
  + Secuencial o Aleatorio
* Tipo de acceso
  + Acceso Compartido: Disco rígido.
  + Acceso Exclusivo: Impresora
* Tipo de acceso:
  + Read only: CDROM
  + Write only: Pantalla
  + Read/Write: Disco

**Subsistema de E/S – Servicios**

* **Planificación**
  + Organización de los requerimientos a los dispositivos.
  + Ej: planificación de requerimientos a disco para minimizar movimientos.
* **Buffering** – Almacenamiento de los datos en memoria mientras se transfieren
  + Solucionar problemas de velocidad entre los dispositivos
  + Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos.
* **Caching** – Mantener en memoria copia de los datos de reciente acceso para mejorar performance.
* **Spooling** – Administrar la cola de requerimientos de un dispositivo.
  + Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo: Por ej. Impresora
  + Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo.
* **Reserva de Dispositivos:** Acceso exclusivo.

**Manejo de Errores:**

* + El S.O debe administrar errores ocurridos (lectura de un disco, dispositivo no disponible, errores de escritura)
  + La mayoría retorna un número de error o código cuando la E/S falla.
  + Logs de errores.

**Formas de realizar la E/S**

* + **Bloqueante:** El proceso se suspende hasta que el requerimiento de E/S se completa
    - **Facil de usar y entender**
    - **No es suficiente bajo algunas necesidades**
  + **No Bloqueante**: El requerimiento de E/S retorna en cuanto es posible
    - **Ejemplo**: Interfaz de usuario que recibe input desde el teclado/mouse y se muestra en el screen.
    - **Ejemplo**: Aplicación de video que lee frames desde un archivo mientras va mostrándolo en pantalla.

**Estructuras de Datos – E/S**

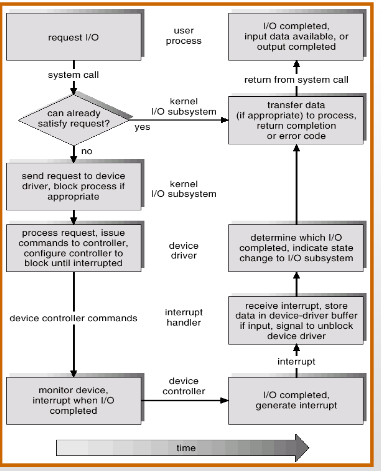
El kernel mantiene la información de estado de cada dispositivo o componente.

* Archivos abiertos
* Conexiones de red
* Etc.

Existen varias estructuras complejas que representan buffers, utilización de la memoria, disco, etc.

**Consideremos la lectura sobre un archivo en un disco:**

* Determinar el dispositivo que almacena los datos
  + Traducir el nombre del archivo en la representación del dispositivo
* Lectura física de los datos en la memoria
* Marcar los datos como disponibles al proceso que realizo el requerimiento
  + Desbloquearlo
* Retornar el control al proceso



**Drivers**

* Contienen el código dependiente del dispositivo
* Manejan un tipo de 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 de los dispositivos están asociadas a una función del driver.
* Actúan como la interfaz entre el SO y el Hardware
* Forman parte del espacio de memoria del Kernel
  + En general se cargan como módulos
* Los fabricantes de Hardware implementan el driver en función de na API especificada por el SO
* Para agregar nuevo Hardware solo basta indicar el driver correspondiente sin necesidad de cambios en el Kernel

**Performance**

E/S 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 E/S
* Provoca context switches ante las interrupciones y bloqueos de los procesos
* Utiliza el bus de mem. En copia de datos:
  + Aplicaciones (espacio usuario) – Kernel
  + Kernel (memoria física) – controladora

**¿Cómo mejorar la 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.

**File System – Clase 1**

**¿Por qué necesitamos archivos?**

Para almacenar grandes cantidades de datos, tener almacenamiento a largo plazo, permitir a distintos procesos acceder al mismo conjunto de información.

**Características de un archivo**

* Entidad con nombre
* Espacio lógico continuo y direccionable
* Provee a los programas de datos (entrada)
* Permite a los programas datos (salida)
* El programa mismo es información que debe guardarse.

**Punto de vista del Usuario**

* ¿Qué operaciones se pueden llevar a cabo?
  + Nombrar un archivo
  + Asegurar la protección
  + Compartir archivos
  + No tratar con aspectos físicos
  + Etc.

**Punto de vista del Diseño**

* Implementar archivos
* Implementar directorios
* Manejo de espacio en disco
* Manejo del espacio libre
* Eficiencia y mantenimiento

**Filesystem**

Es el conjunto de unidades de software que proveen los servicios necesarios para la utilización de archivos. Facilita el acceso a los archivos por parte de las aplicaciones y permite la abstracción al programador, en cuanto al acceso de bajo nivel (El programador no desarrolla el soft de administración de archivos)

**Objetivos**

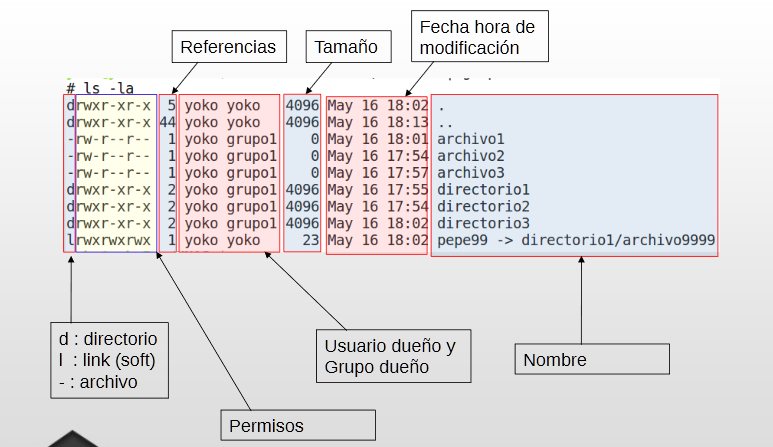
* Cumplir con la gestión de datos
* Cumplir con las solicitudes del usuario
* Minimizar/eliminar posibilidad de perder o destruir Datos
  + Garantizar la integridad del contenido de los archivos
* Dar soporte de E/s a distintos dispositivos
* Brindar un conjunto de interfaces de E/S para tratamiento de archivos.
* Proveer soporte de E/S para múltiples usuarios.

**Tipos de archivos:**

* **Regulares**
  + Texto plano
    - Source file
  + Binarios
    - Object File
    - Executable File
* **Directorios**
  + Archivos que mantienen la estructura en el fileSystem

**Atributos de un Archivo**

* Nombre
* Identificador
* Tipo
* Localización
* Tamaño
* Protección, seguridad y monitoreo
  + Owner, permisos, password
  + Momento en el que usuario lo modifico, creo, accedió por ultima vez
  + ACLs



**Directorios:**

Contienen información acerca de archivos y directorios que están dentro de él. El directorio es, en si mismo, un archivo.

Interviene en la resolución entre el nombre y el archivo mismo.

Operaciones en directorios:

* Search for a file
* Create a file (directory entry)
* Delete a file (directory entry)
* List a directory
* Rename a file

El uso de directorios ayuda con:

* La eficiencia: Localización rápida de archivos
* Uso del mismo Nombre de archivo:
  + Diferentes usuarios pueden tener el mismo nombre de archivo
* Agrupación: Agrupación lógica de archivos por propiedades/funciones:
  + Ejemplo: Programas, juegos, librerías, etc.

**Estructura de Directorios**

Los archivos pueden ubicarse siguiendo un path desde el directorio raíz y sus sucesivas referencias (**full pathname** del archivo)

Distintos archivos pueden tener el mismo nombre pero el fullpathname es único.

El directorio actual se lo llama “directorio de trabajo”

Dentro del mismo, se puede referenciar los archivos tanto por su **path absoluto** como por su **path relativo** indicando solamente la ruta al archivo desde el directorio de trabajo.

**Compartir Archivos**

En un ambiente multiusuario se necesita que varios usuarios puedan compartir archivos. Esto debe ser realizado bajo un esquema de protección: **Derechos de acceso y Manejo de accesos simultáneos**.

**Protección**

El propietario/administrador debe ser capaz de controlar:

* Que se puede hacer
  + Derechos de acceso
* Quien lo puede hacer

Los directorios también tienen permisos, los cuales pueden permitir el acceso al mismo para que el usuario pueda usar el mismo archivo siempre y cuando tenga permisos.

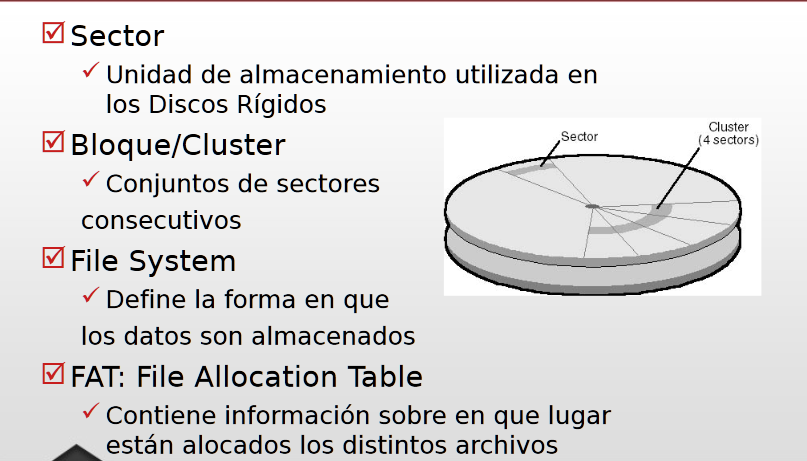
**Tipos de derechos**

* **Execution**
  + El usuario puede ejecutar
* Reading
  + El usuario puede leer el archivo
* Appending
  + El usuario puede agregar datos pero no modificar o borrar el contenido del archivo.
* Updating
  + El usuario puede modificar,borrar y agregar datos. Incluye la creación de archivos, sobreescribirlo y remover datos.
* Changing protection
  + El usuario puede modificar los derechos de acceso
* Deletion
  + El usuario puede borrar el archivo
* Owners (propietarios)
  + Tienen todos los derechos
  + Pueden dar derechos a otros usuarios. Se determinan clases:
    - Usuario específico
    - Grupos de usuarios
    - Todos(archivos públicos)

**Filesystem – clase 2**

**Metas del sistema de archivos**

* Brindar espacio en disco a los archivos de usuario y del sistema.
* Mantener un registro del espacio libre, cantidad y su ubicación del mismo dentro del disco



**Pre-asignación**

* Se necesita 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.
* Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo.

**Asignación dinámica**

* El espacio se solicita a medida que se necesita
* Los bloques de datos pueden quedar de manera no contigua

**Asignación continúa**

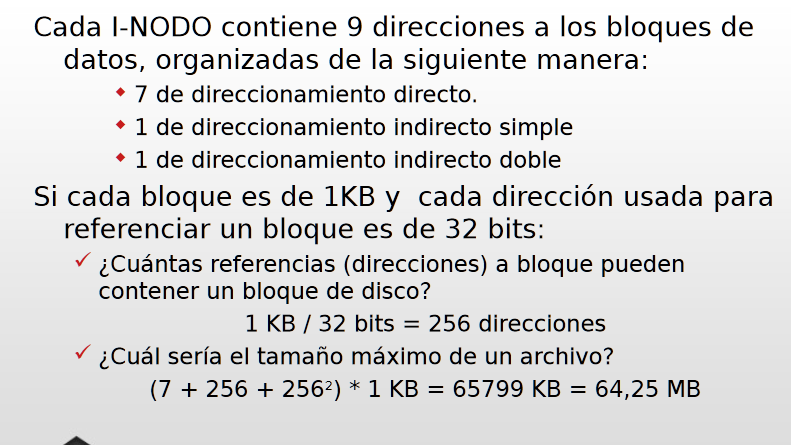
* Conjunto continuo de bloques son utilizados
* Se requiere una pre-asignación.
  + Se debe conocer el tamaño del archivo durante su creación
* File Allocation Table (FAT) es simple
  + Solo una entrada que incluye bloque de inicio y longitud
* El archivo puede ser leído con una única operación.
* Puede existir fragmentación externa.
  + Se deberá implementar algún tipo de compactación.
* **Problemas:**
  + Se deben encontrar bloques libres continuos en el disco
  + Incremento del tamaño de un archivo.

**Asignación Encadenada**

* Asignación en base a bloques individuales
* Cada bloque tiene un puntero al próximo bloque del archivo
* File allocation table (FAT)
  + Unica entrada por archivo: Bloque de inicio y tamaño de archivo
* No hay fragmentación externa
* Util para acceso secuencial (no Random)
* Los archivos pueden crecer bajo demanda
* No se requieren bloques contiguos.

**Asignación Indexada**

* Asignación en base a bloques individuales
* No se produce fragmentación externa
* El acceso “random” a un archivo es eficiente
* FAT
  + Unica entrada con la dirección del bloque de índices (index node / i-node)



**Gestión de Espacio Libre**

* Control sobre cuáles de los bloques de disco están disponibles
* **Alternativas**
  + Tablas de bits
  + Bloques libres encadenados
  + Indexación

**Tabla de bits**

Tabla(vector) con 1 bit por cada bloque de disco

**Cada entrada:**

0 = Bloque libre 1 = Bloque en uso

**Ventaja**

Facil encontrar un bloque o grupo de bloques libres.

**Desventaja**

Tamaño del vector en memoria (tamaño disco bytes / tamaño bloques en sistema archivo)

**Bloques encadenados**

Se tiene un puntero al primer bloque libre. Cada bloque libre tiene un puntero al siguiente bloque libre.

**Es ineficiente** para la búsqueda de bloques libres, ya que hay que realizar varias operaciones de E/S para obtener un grupo libre.

Genera problemas si se pierde un enlace, también es difícil encontrar bloques libres consecutivos.

**Indexación**

Es una 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.

**Recuento**

Variante de indexacion

* Esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o liberados a la vez (en especial con asignación contigua)
* En lugar de tener N direcciones libres(índice) se tiene
  + La dirección del primer bloque libre
  + Los N bloques libres contiguos que le siguen.
  + (#bloque, N siguientes bloques libres)

**Filesystem – Clase 3**

**UNIX – Manejo de archivos**

* Tipos de Archivos:
  + Archivo Comun
  + Directorio
  + Archivos especiales (dispositivos /dev/sda por ejemplo)
  + Named pipes (comunicación entre procesos)
  + Links (comparten el i-nodo, solo dentro del mismo filesystem)
  + Link simbólicos (para filesystem diferentes)

**UNIX – Estructura del Volumen**

* Boot Block: Código para bootear el S.O
* Superblock: Atributos sobre el File System
* I-NODE Table: Tabla que contiene todos los I-NODOS
  + I-Nodo: Estructura de control que contiene la información clave de un archivo
* Data blocks: Bloques de datos de los archivos.

**LINUX – VFS- Virtual File System**

Es una interfaz uniforme para el acceso de los procesos a los archivos. Una capa de abstracción sobre los distintos fileSystems utilizados.

Asume que los archivos son objetos que comparten propiedades más alla del file System que los almacena.

**Objetos:**

* **Superblock object**
  + Representa el file system montado
* Inode object
  + Representa un archivo
* D-entry object
  + Representa una entrada en un directorio
* File object
  + Representa un archivo abierto asociado a un proceso.

**Windows – FAT**

File Allocation Table(mapa de bloques del sistema de archivos) es un sistema de archivos utilizado originalmente por DOS y Windows 9x

¿Por qué aun es soportado por Windows?

* Para compatibilidad con otro SO en sistemas multiboot
* Para permitir upgrades desde versiones anteriores
* Para formato de dispositivos como diskettes

Las distintas versiones de FAT se diferencian por un numero que indica la cantidad de bits que se usan para identificar diferentes bloques o clusters: FAT12, FAT16, FAT32

* La FAT tiene tantas entradas como bloques.
* La FAT, su duplicado y el directorio raíz se almacenan en los primeros sectores de la partición
* Se utiliza un esquema de **ASIGNACIÓN ENCADENADA**
* La única diferencia es que el puntero al próximo bloque esta en la FAT y no en los bloques
* Bloques libres y dañados tienen códigos especiales.

**Windows - NTFS**

Es el filesystem nativo de Windows, usa 64-bit para referenciar clusters. Teóricamente tiene la habilidad de direccionar hasta 16 exabytes.

**¿Por qué usar NTFS en lugar de FAT?**

FAT es simple, más rápido para ciertas operaciones, pero NTFS soporta:

* Tamaños de archivo y discos mayores
* Mejor performance en discos grandes
* Nombres de archivos de hasta 255 caracteres
* Atributos de seguridad.

**Buffer Cache – última clase**

* Buffers en memoria principal para almacenamiento temporario de sectores de disco
* Contienen una copia de algunos sectores de disco.
* ¿Objetivo? **Minimizar la frecuencia de acceso al disco**

**Algunas observaciones:**

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

* Se copia el espacio de direcciones de usuario
* O se trabaja como memoria compartida (no se copia permitiendo acceso a varios procesos)

**Estrategia de reemplazo**

Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo no es referenciado, se trata de una lista de bloques donde el último es el más recientemente usado (LRU, Least Recently Used)

Cuando un bloque se referencia o entra en al cache queda al final de la lista.

No se mueven los bloques en la memoria, se asocian punteros.

Otra alternativa: Least frecuently Used (reemplaza el que tenga menor número de referencias)

**Buffer Cache – Unix System V**

**Objetivo y estructura**

* Minimizar la frecuencia de acceso a disco
* Es una estructura formada por buffers
* El kernel asigna un espacio en la memoria durante la inicialización para esta estructura
* Un buffer tiene dos partes: el header y el lugar donde se almacena el bloque de disco traído a memoria

**El Header**

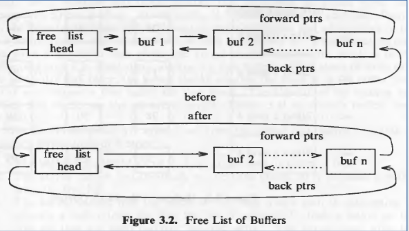
* Identifica por número de dispositivo lógico y número de bloque
* Estado
* Punteros a:
  + 2 punteros para la hash queue
  + 2 punteros para la free list
  + 1 puntero al bloque en memoria

**Estados de los buffers**

* Free o disponible
* Busy o no disponible (en uso por algún proceso)
* Se está escribiendo o leyendo del disco
* Delayed Write(DW): buffers modificados en memoria, pero los cambios no han sido reflejados en el bloque original del disco

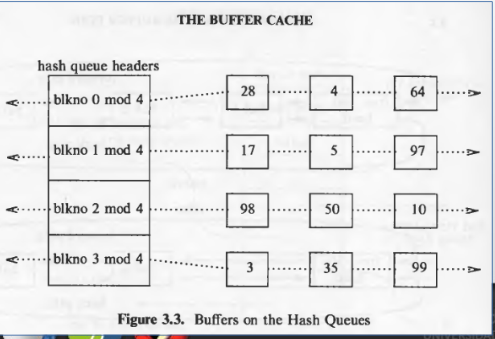
**Free list**

* Organiza los buffers disponibles para ser utilizados para cargar nuevos bloques de disco
* No necesariamente los buffers están vaciós.
* Se ordena según Least Recently Used



**Hash Queues**

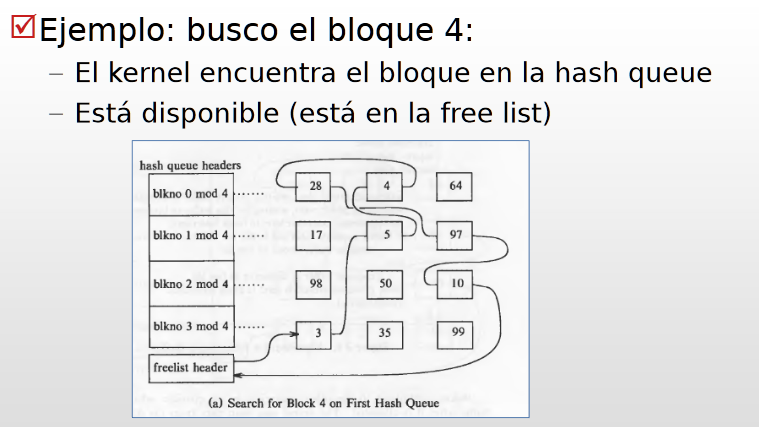
* Son colas para optimizar la búsqueda de un buffer en particular.
* Se organizan según una función de hash usando (dispositivo, #bloque)



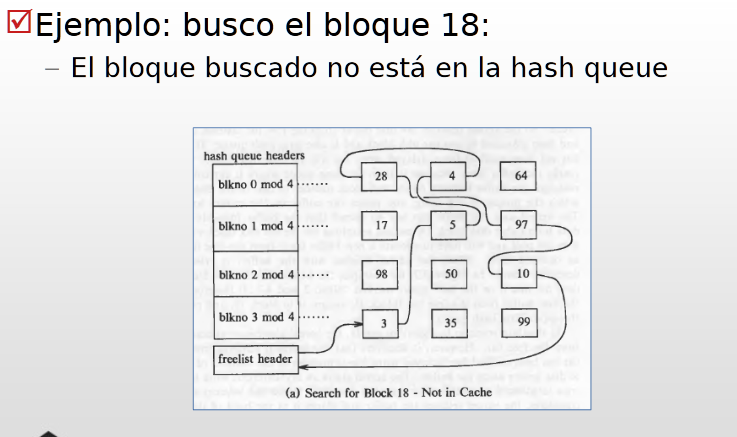
**Funcionamiento del Buffer caché**

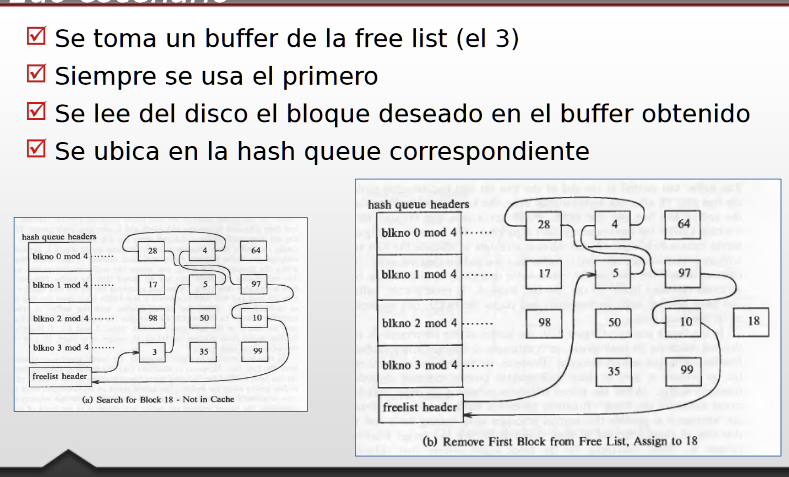
* Cuando un proceso quiere acceder a un archivo, utiliza su inodo para localizar los bloques de datos donde se encuentra éste.
* El requerimiento llega al buffer cache quien evalúa si puede satisfacer el requerimiento o si debe realizar la E/S
* Se pueden dar 5 escenarios:
  1. El kernel encuentra el bloque en la hash queue y el buffer esta 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 esta BUSY.

**PRIMER ESCENARIO**

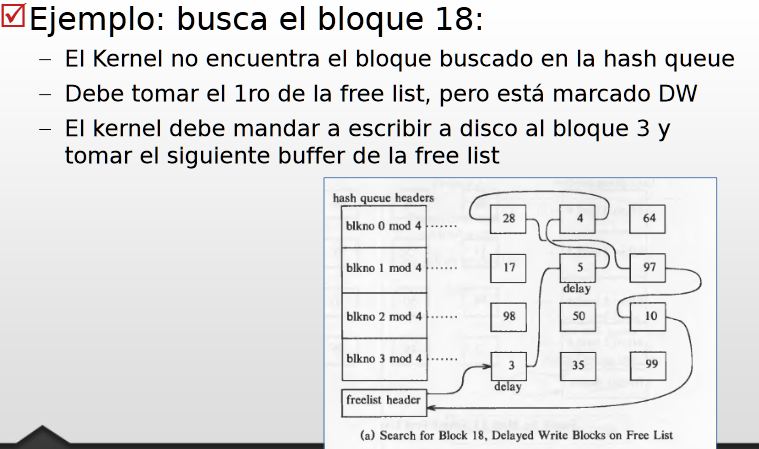


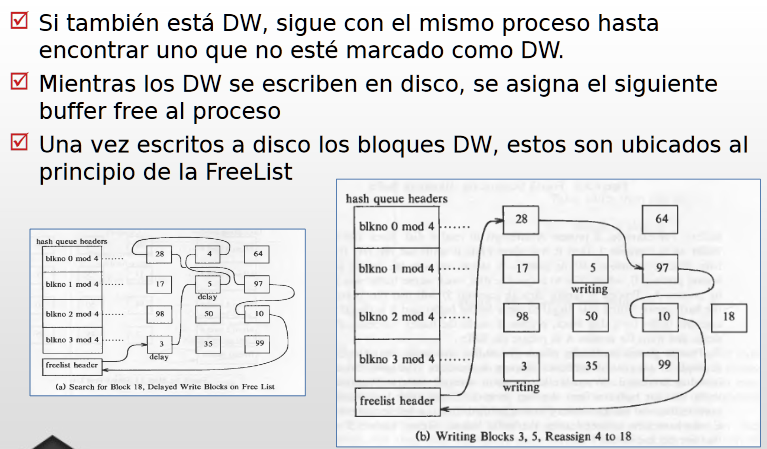
**SEGUNDO ESCENARIO**





**TERCER ESCENARIO**





**Busqueda/Recuperación de un buffer: 4to escenario**

* El kernel no encuentra el bloque en la hash queue y la free list está vacía
* El proceso queda bloqueado en espera a que se “libere” algún buffer
* Cuando el proceso despierta se debe verificar nuevamente que el bloque no esté en la hash queue (algún proceso pudo haberlo pedido mientras este dormía)

**QUINTO ESCENARIO**

