Memoria virtual

Puede que no todo el espacio de direcciones del proceso se necesite en todo momento. Por lo que no hay necesidad de cargar toda la imagen del proceso en memoria

Conjunto residente

El SO puede traer a memoria las piezas del proceso a medida que se necesitan

Se denomina “Conjunto Residente” a la porción del espacio de direcciones del proceso que se encuentra en memoria (También se le llama “Working Set”)

Con el apoyo del hardware

- Se detecta cuándo 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 mantenido en memoria
    - Sólo son cargadas algunas secciones de cada proceso
    - Con más procesos en memoria principal es 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 los programas
    - La limitación la impone el hardware y el bus de direcciones

Qué se necesita para memoria virtual

- El hardware debe soportar paginación por demanda y/o segmentación por demanda
- Un dispositivo de memoria secundaria que dé el apoyo para almacenar las secciones del proceso que no están en memoria principal (área de intercambio o SWAP)
- El SO debe ser capaz de manejar el movimiento de las páginas o segmentos entre la memoria principal y la secundaria

Memoria virtual con paginación

- Cada proceso tiene su tabla de páginas y cada entrada referencia al marco en el que se encuentra la página en memoria principal
- El formato de la tabla de páginas lo define el HW y el SO se adapta a él
- Cada entrada en la tabla de páginas tiene bits de control
    - Bit V:
        - Indica si la página está en memoria
        - Lo activa o desactiva el SO, lo consulta el HW
    - Bit M:
        - Indica si la página fue modificada.
        - Si se modificó, en algún momento, se deben reflejar los cambios en memoria secundaria
        - Lo activa el HW, lo consulta y desactiva el SO
- Una entrada válida contiene
    - Bit V en 1
    - Page Frame Number (Marco de memoria asociado)
    - Flags que describen su estado y protección

Page fault

Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en la memoria principal. O sea, esa entrada en la tabla de páginas tiene el bit V = 0. O también está marcado con una i de inválido.

Cuando sucede esto, quiere decir que la página no se encuentra en su conjunto residente

El HW detecta esta situación y genera un trap al SO. El cual podrá colocar al proceso en estado de Espera o Bloqueado mientras gestiona la carga de la página

Cuando esto sucede, se busca un marco libre en la memoria y se genera una operación de E/S al disco para copiar en dicho frame la página del proceso que se necesita utilizar

El SO puede asignarle la CPU a otro proceso mientras se completa la E/S

Cuando la operación de E/S finalice, se notifica al SO por una interrupción y éste

- Actualiza la tabla de páginas del proceso
    - Colocar Bit V en 1 en la página en cuestión
    - Colocar la dirección base del marco donde se colocó la página
- El proceso que generó el PF vuelve a estado de Ready
- Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que generó el PF

Reemplazo de páginas - Selección de páginas victimas

Cuando hay que alocar una página y ya no hay marcos disponibles, se debe seleccionar una página víctima para ser reemplazada

Hay varios algoritmos

- FIFO
    - Se tratan los marcos como cola circular
    - Simple de implementar
    - La página más antigua en memoria es reemplazada
- FIFO 2da Chance
    - Un avance del FIFO tradicional que beneficia a las páginas más referenciadas
    - Se utiliza un bit más para marcar aquellas páginas que fueron referenciadas y se reencolan si ese bit está en 1 y se seleccionan como víctimas
- Optimo
    - Se selecciona como página víctima aquella que no vaya a ser referenciada en un futuro próximo
    - Imposible de hacer
    - Teórico
- LRU (Least Recently Used)
    - Se selecciona como víctima aquella página que fue menos utilizada recientemente
    - Favorece a las páginas más recientemente accedidas
    - Requiere soporte del hardware para mantener timestamps de acceso a las páginas
- NRU (Non Recently Used)
    - Utiliza bits R y M
    - Favorece a las páginas que fueron usadas recientemente
    - Según cómo se implemente, puede que los bits R se limpien (poner en 0) en cada ciclo de reloj, o cada cierto tiempo (esto lo hace el kernel)
- Etc.

Alcance del reemplazo

Reemplazo Global

- El PF de un proceso puede reemplazar la página de cualquier proceso
- El SO no controla la tasa de PF 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 marcos de un proceso de menor prioridad

Reemplazo Local

- El PF de un proceso sólo puede reemplazar sus propias páginas (de su conjunto residente)
- No cambia la cantidad de frames asignados

- El SO puede determinar cuál es la tasa de PF de cada proceso
- Un proceso puede tener marcos asignados que no usa y no pueden ser usados por otro proceso

Asignación y Alcance

Table 8.5 Resident Set Management

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

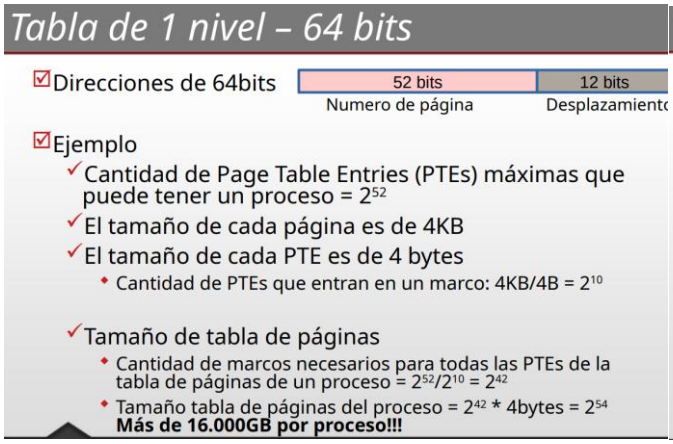
Tabla de páginas

- Cada proceso tiene su tabla de páginas
- El tamaño de la tabla de páginas depende del espacio de direcciones del proceso
- Puede alcanzar un gran tamaño

Organización de la tabla de páginas

La forma de organizarla depende del HW y hay varios métodos

Tabla única lineal (1 nivel)



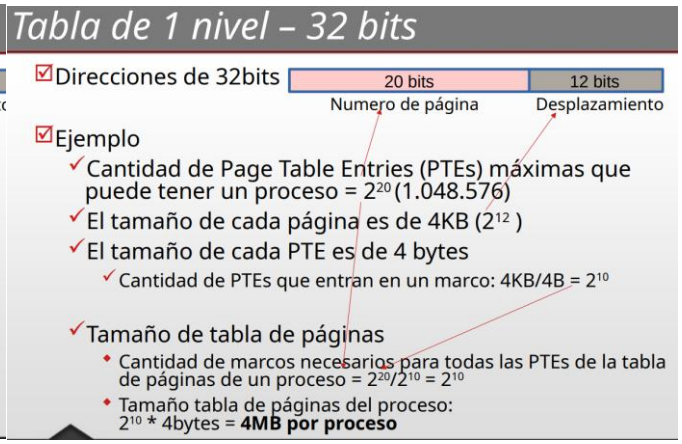
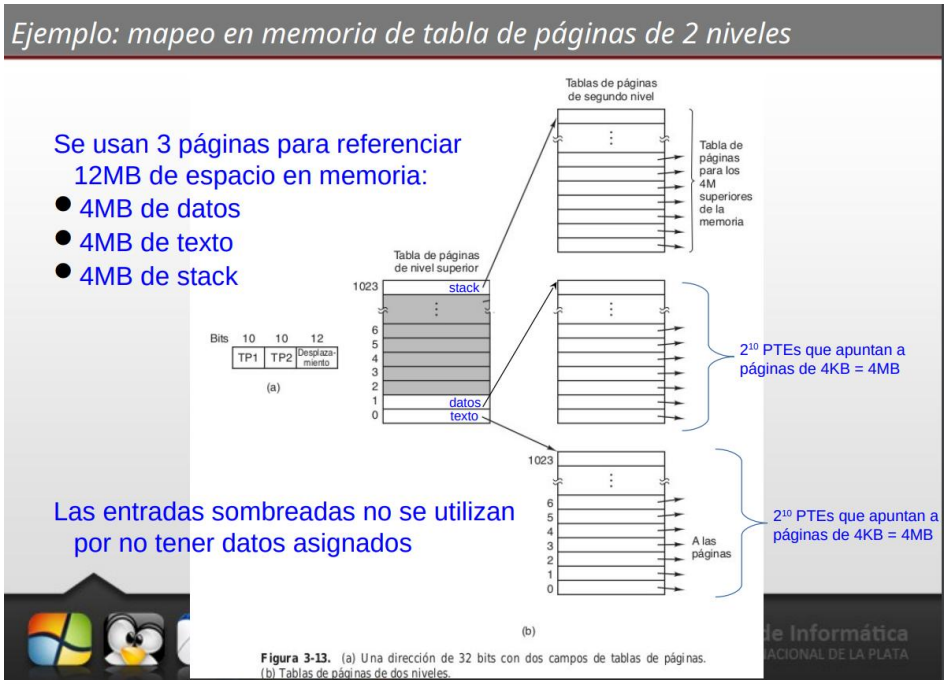


Tabla de 2 niveles o más (multinivel)

- 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 no ocupe demasiada memoria principal
- Sólo se carga una parcialidad de la tabla de páginas (lo que se necesite resolver)
- Existe un esquema de direccionamientos indirectos
- Implican más accesos a memoria



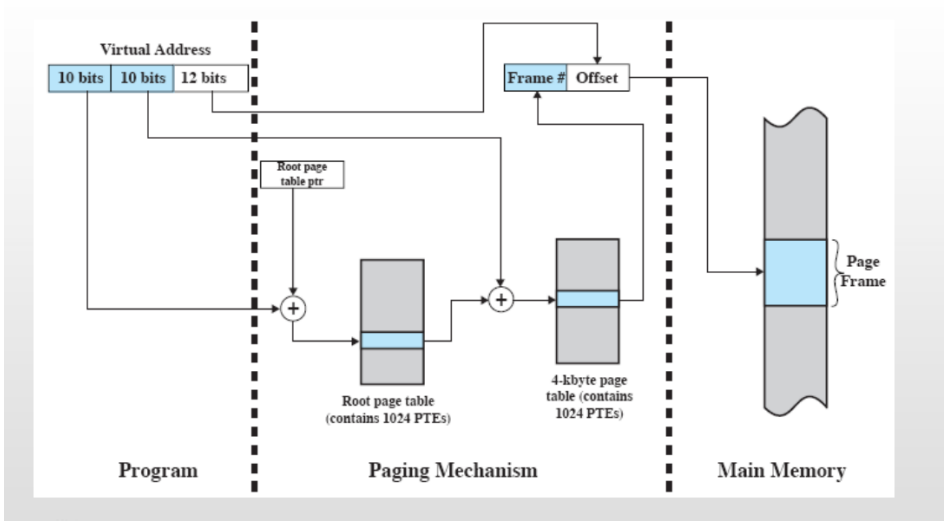


Tabla invertida (Hashing)

- Utilizada en arquitecturas donde el espacio de direcciones es muy grande
    - La tabla de páginas ocuparía muchos niveles y la traducción sería costosa
- Hay una entrada por cada marco de página en la memoria real
- Hay una sola tabla para todo el sistema
- El espacio de direcciones de la tabla se refiere al espacio físico en memoria principal, no a direcciones virtuales
- El número de página es transformado en un valor 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 (cuando el hash da igual para 2 direcciones virtuales)
- Sólo se mantienen las entradas de la tabla de página que tengan páginas presentes en memoria física
    - La tabla invertida es organizada como tabla hash en memoria principal
        - Se busca indexadamente por número de página virtual
        - Si está presente en la tabla, se extrae el marco de página y sus protecciones
        - Si no está presente es un PF

Tamaño de la página

- Pequeño
    - Menor fragmentación interna

    - Más páginas requeridas por proceso
    - Tablas de páginas más grandes
    - Más páginas pueden residir en memoria
- Grande
    - Mayor fragmentación interna
    - La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente por lo que es más rápido mover páginas hacia la memoria principal
    - Tabla de páginas más chica
    - Implica que se puede tener información que potencialmente el proceso no necesite

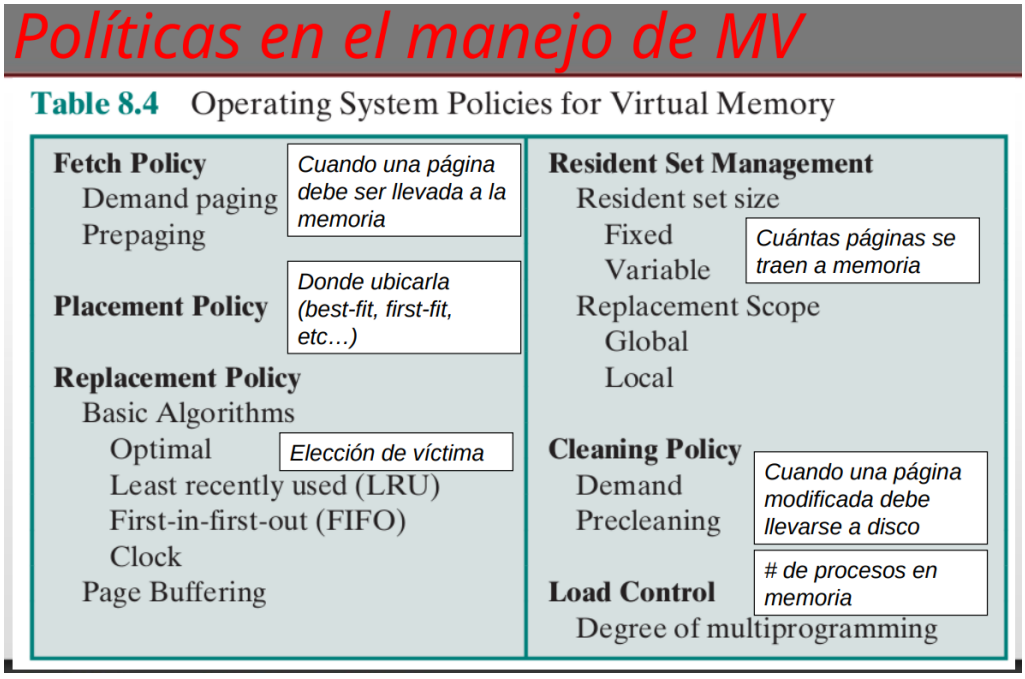
Translation Lookaside Buffer (TLB)

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

Para solucionar este problema se tiene la TLB

- Caché que almacena entradas de páginas.
- Es asociativo (las palabras pueden estar en cualquier ranura de la caché)
- Es LRU
- Reduce accesos a memoria
- 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 proceso
- Se controla si la página está en memoria
    - Si no está, se genera un PF
    - La TLB es actualizada para incluir la nueva entrada
- El cambio de contexto genera la invalidación de las entradas de la TLB (vaciar la caché)

Políticas en el manejo de memoria virtual



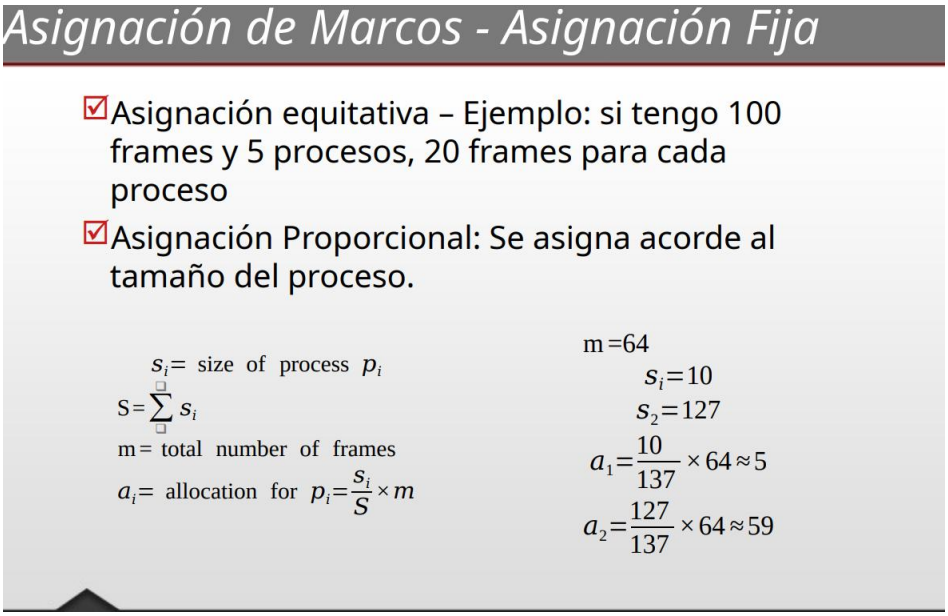
Asignación de marcos

La cantidad de páginas de un proceso que se pueden encontrar en memoria está dada por el tamaño del conjunto residente

Hay varias técnicas de asignación

Asignación fija

- Número fijo de marcos para cada proceso
- Puede ser equitativo o proporcional (acorde al tamaño del proceso)



Asignación dinámica

- Número variable de marcos para cada proceso
- Se dan marcos a demanda

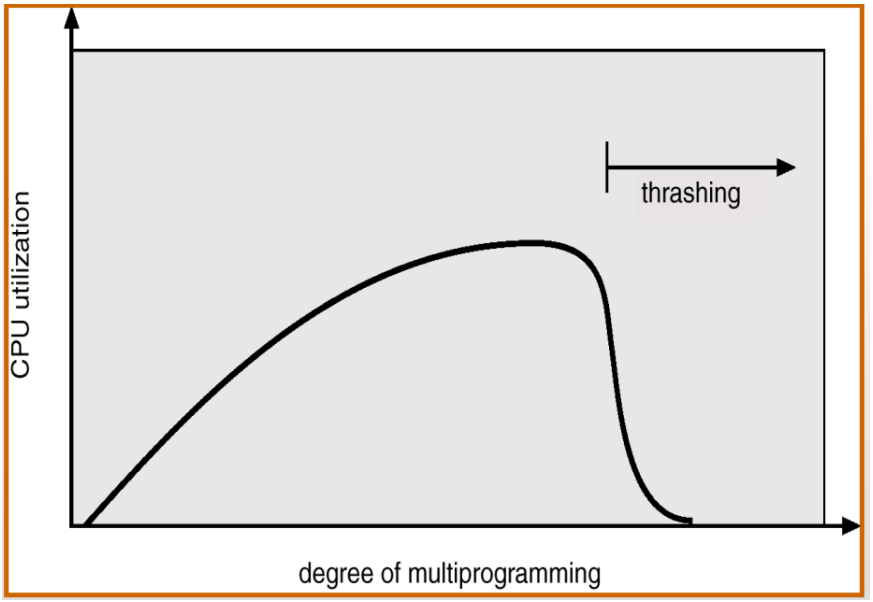
Trashing / Hiperpaginación

Situación que ocurre cuando el sistema pasa más tiempo paginando que ejecutando procesos

Como consecuencia hay una baja importante del rendimiento del sistema

Ciclo del trashing

1. El kernel monitorea el uso de la CPU
2. Si hay baja utilización, aumenta el grado de multiprogramación
3. Si el algoritmo de reemplazo es global, pueden sacarse marcos a otros procesos
4. Un proceso necesita más marcos. Comienzan los PF y robo de marcos a otros procesos
5. Por swapping de páginas y encolamiento en dispositivos, baja el uso de la CPU
6. Vuelve a 1.



El planificador de CPU y el trashing

1. Cuando se decrementa el uso de la CPU, el Long Term Scheduler aumenta el grado de multiprogramación
2. El nuevo proceso inicia nuevos PF y por lo tanto más actividad de paginado
3. Se decrementa el uso de la CPU
4. Vuelve a 1

Control del trashing

- Se puede limitar el trashing usando algoritmos de reemplazo local
- De esta forma, si un proceso entra en trashing, no roba frames a otros procesos
- Si bien perjudica el rendimiento del sistema, es controlable

Conclusiones

- Se soluciona bajando el grado de multiprogramación
- No habría trashing si todos los marcos del proceso están cargados (el proceso cuenta con todos los frames que necesita)
- Una manera de abordar esta problemática es utilizar la estrategia de Working Set que se apoya en el modelo de localidad
- Otra estrategia es el algoritmo PFF (Frecuencia de Fallos de Página)

Principio de cercanía / Modelo de localidad

- La localidad de un proceso se da por el conjunto de páginas cargadas en un momento determinado
- Las referencias a datos y programa dentro de un proceso tienden a agruparse
- Se deben cargas las páginas más activas
- Un programa se compone de varias localidades
    - Por ejemplo, cada rutina será una nueva localidad y se referencian sus direcciones cercanas cuando se está ejecutando
- Para prevenir la hiperpaginación, un proceso debe tener en memoria sus páginas más activas (menos page faults)

Modelo de working set

- Se basa en el modelo de localidad
- La “Ventana del working set”(Δ) contiene las referencias de memoria más recientes y se representa con el símbolo Δ. Define la ventana de tiempo a través de la cual se observa el proceso.
- El “Working Set” es el conjunto de las Δ páginas más recientemente referenciadas. Es el conjunto de páginas del proceso a las que se ha referenciado en las últimas Δ unidades de tiempo.

Selección de Δ

- Chico: No cubrirá la localidad
- Grande: Puede tomar varias localidades

Medida del working set

- ☑ m = cantidad frames disponibles
- ☑ WSS_i= tamaño del working set del proceso p_i.
- ☑ WSS_i=D;
- ☑ D= demanda total de frames.
- ☑ Si D>m, habrá ***thrashing***.

Prevencion del trashing

- El SO monitorea cada proceso dándole tantos marcos hasta su WSSi
- Si quedan marcos libres puede iniciar otro proceso
- Si la demanda crece, excediendo la cantidad de marcos libres, se elige un proceso para suspender, reasignando sus marcos
- Así se mantiene alto el grado de multiprogramación optimizando el uso de la CPU

Problemas del modelo de working set

- Mantener un registro de los WSSi es costoso
- La ventana es móvil/dinámica lo que dificulta su gestión

Prevención del trashing por Page Fault Frecuency (PFF)

En lugar de calcular el WS de los procesos, se utiliza la tasa de fallos de página para estimar si el proceso tiene un conjunto residente que representa adecuadamente al WS

PFF Alta -> Se necesitan más marcos

PFF Baja -> Los procesos tienen marcos asignados que les sobran

- Se establecen límites inferiores y superiores para la tasa de fallos de página (una tasa “aceptable”)
    - Si se supera el límite superior, se incrementa la cantidad de marcos asignados al proceso dado que se están generando muchos PF y probablemente necesite más marcos
    - Si se supera el límite inferior, se puede decrementar la cantidad de marcos asignados al proceso para ser utilizados en los procesos que los necesiten más
    - Se puede llegar a suspender un proceso si no hay más marcos disponibles y todos los procesos están por encima del límite superior

Demonio de paginación

- Proceso creado por el SO durante el arranque que apoya a la administración de memoria
- Se ejecuta cuándo el sistema tiene una baja utilización o algún parámetro de la memoria lo indica
    - Poca memoria libre
    - Mucha memoria modificada
- En Linux el proceso es llamado “Kswapd”
- En Windows el proceso es llamado “System”

Tareas del demonio de paginación

- Limpiar páginas modificadas sincronizándolas con el swap

    - Reducir el tiempo de swap posterior ya que las páginas están “limpias”
    - Reducir el tiempo de transferencia al sincronizar varias páginas contiguas
- Mantener un cierto número de marcos libres en el sistema
- Demorar la liberación de una página hasta que haga falta realmente

Memoria compartida

Gracias al uso de la tabla de páginas, varios procesos pueden compartir un marco de memoria. Para ello, ese marco debe estar asociado a una página en la tabla de páginas de cada proceso

El número de página asociado al marco puede ser diferente en cada proceso

Código compartido

- Los procesos comparten una copia de código (sólo lectura)
- Los datos son privados a cada proceso y se encuentran en páginas no compartidas

Mapeo de archivos 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 PF
- Cualquier modificación a la dirección de memoria es una modificación indirecta al archivo
- El contenido de la página que genera el PF es obtenido desde el archivo asociado, no del área de intercambio
- Cuando el proceso termina o el archivo de libera, las páginas modificadas son escritas en el archivo correspondiente
- Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el sistema de archivos
- Es utilizado comúnmente para asociar librerías compartidas

Copia en Escritura (COW / Copy-On-Write)

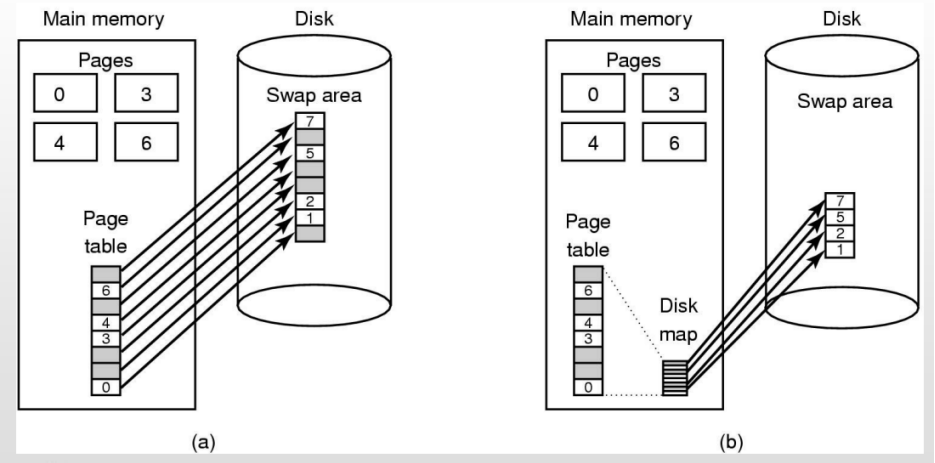
- Permite a los procesos compartir inicialmente las mismas páginas de memoria
    - Si uno de ellos modifica una página compartida, la página es copiada
    - En fork, permite inicialmente que padre e hijo utilicen las mismas páginas sin necesidad de duplicación
- Permite crear procesos de forma más eficiente debido a que sólo las páginas modificadas son duplicadas

Área de intercambio (swap)

- Área dedicada, separada del sistema de archivos (Linux)
- Un archivo dentro del sistema de archivos (Windows)

Técnicas para la administración

1.
    - 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
2.
    - 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 págin 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, sinó en swap, para saber en qué parte del área de intercambio está se utiliza la entrada de dicha página en la tabla. La cual va a tener el Bit V en 0 y todos los demás bits sin usar

- Linux permite definir un número predefinido de áreas de swap
- Swap_info es un arreglo que contiene estas estructuras
- Cada área es dividida en un número fijo de ranuras según el tamaño de la página
- Cuando una página es llevada a disco, Linux utiliza el PTE (Page Table Entry) para almacenar 2 valores:
    - El número de área
    - El desplazamiento en el área (24 bits, lo que limita el tamaño máximo del área a 64GB)

Sistema operativo

Responsabilidades

- Controlar dispositivos de E/S
    - Generar comandos
    - Manejar interrupciones
    - Manejar errores
- Proporcionar una interfaz de utilización

Problemas

- Heterogeneidad de dispositivos
- Características de los dispositivos
- Velocidades diferentes
- Formatos diferentes
- Manejan diferentes cantidades de datos
- Nuevos tipos de dispositivos
- Diferentes formas de realizar E/S
- La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM

Aspectos de los dispositivos de I/O

Unidad de transferencia

- Dispositivos por bloques (discos): Read, Write, Seek
- Dispositivos por carácter: Get, Put

Formas de acceso

- Secuencial
- Aleatorio

Tipo de acceso

- Acceso compartido: Disco rígido

- Acceso Exclusivo: Impresora

Tipo de acceso (modo)

- Read only: CDROM
- Write Only: Monitores
- Read/Write: Disco

Velocidad

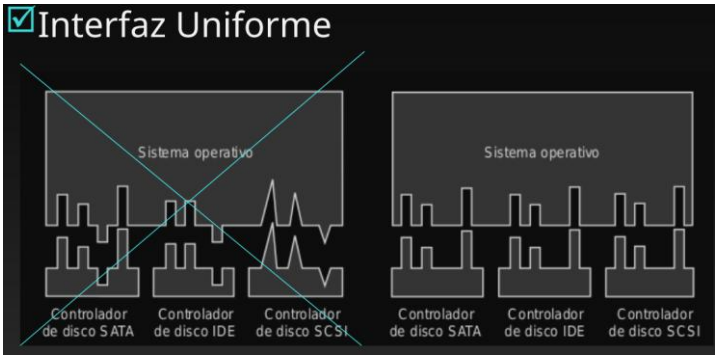
Dispositivo	Velocidad de transferencia de datos
Teclado	10 bytes/seg
Ratón	100 bytes/seg
Módem de 56K	7 KB/seg
Escáner	400 KB/seg
Cámara de video digital	3.5 MB/seg
802.11g inalámbrico	6.75 MB/seg
CD-ROM de 52X	7.8 MB/seg
Fast Ethernet	12.5 MB/seg
Tarjeta Compact Flash	40 MB/seg
FireWire (IEEE 1394)	50 MB/seg
USB 2.0	60 MB/seg
Red SONET OC-12	78 MB/seg
Disco SCSI Ultra 2	80 MB/seg
Gigabit Ethernet	125 MB/seg
Unidad de disco SATA	300 MB/seg
Cinta de Ultrium	320 MB/seg
Bus PCI	528 MB/seg

Variedad en los dispositivos de I/O

- Legible para el usuario
  - Comunicación con el usuario
  - Pantallas, Teclados
- Legible para la máquina
  - Comunicación con componentes electrónicos
  - Sensores, Discos
- Comunicación con dispositivos remotos
  - Modems

Metas, objetivos y servicios

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

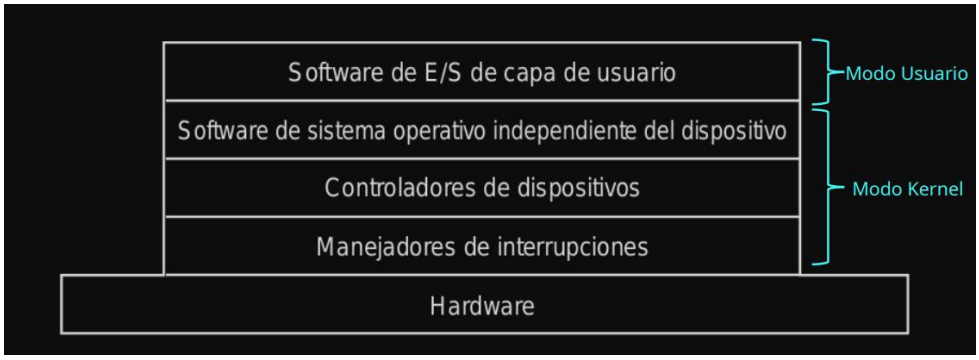


- Eficiencia
  - Los dispositivos de IO pueden resultar extremadamente lentos respecto a la memoria y CPU
  - El uso de la multiprogramación permite que un proceso espere por la finalización de su I/O mientras otro proceso se ejecuta
- Planificación
  - Organización de los requerimientos a los dispositivos
- 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 copias de los datos de reciente acceso para mejorar el rendimiento
- Spooling
  - Administrar la cola de requerimientos de un dispositivo
  - Algunos dispositivos de acceso exclusivo no pueden atender distintos requerimientos al mismo tiempo
  - Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo
  - Consiste en tener una cola de trabajos enviados a un dispositivo
- Reserva de dispositivos
  - Acceso exclusivo
- Manejo de errores
  - El S.O debe administrar errores ocurridos
  - La mayoría retorna un número de error o código cuando la operación I/O falla
  - Registro de errores

Formas de realizar IO

- Bloqueante:
  - El proceso se suspende hasta que el requerimiento de I/O se completa
  - Fácil de usar y entender
  - No es suficiente bajo algunas necesidades
- No bloqueante:
  - El requerimiento de I/O retorna en cuanto es posible
  - Ejemplos:
    - Interfaz de usuario que recibe input desde teclado y se muestra en pantalla
    - Aplicación de video que lee fotogramas de un archivo mientras va mostrandolos en pantalla

Diseño



Software de capa de Usuario

- Librería de funciones
  - Permiten acceso a SysCalls
  - Implementar servicios que no dependen del kernel
- Procesos de apoyo
  - Demonio de impresión (spooling)

Software independiente SO

- Brinda los principales servicios de I/O antes vistos
  - Interfaz uniforme
  - Manejo de errores
  - Buffer
  - Asignación de recursos
  - Planificación
- Kernel mantiene información de estado de cada dispositivo o componente
  - Archivos abiertos
  - Conexiones de red
  - Etc.
- Hay varias estructuras complejas que representan buffers, utilización de memoria, disco, etc.

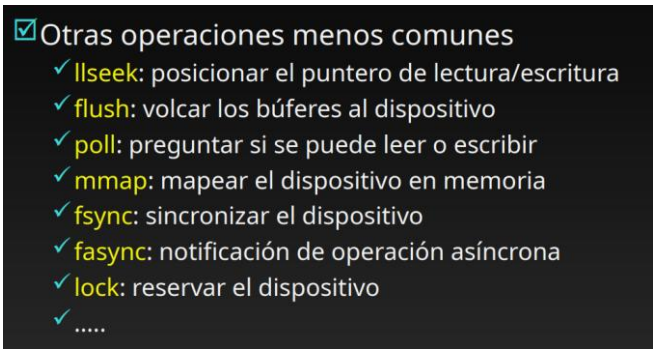
Controladores / Drivers

- 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 generadas por los dispositivos son atendidas por funciones provistas por el driver
- Interfaz entre el SO y el HW
- Forman parte del espacio de memoria del kernel
  - En general 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

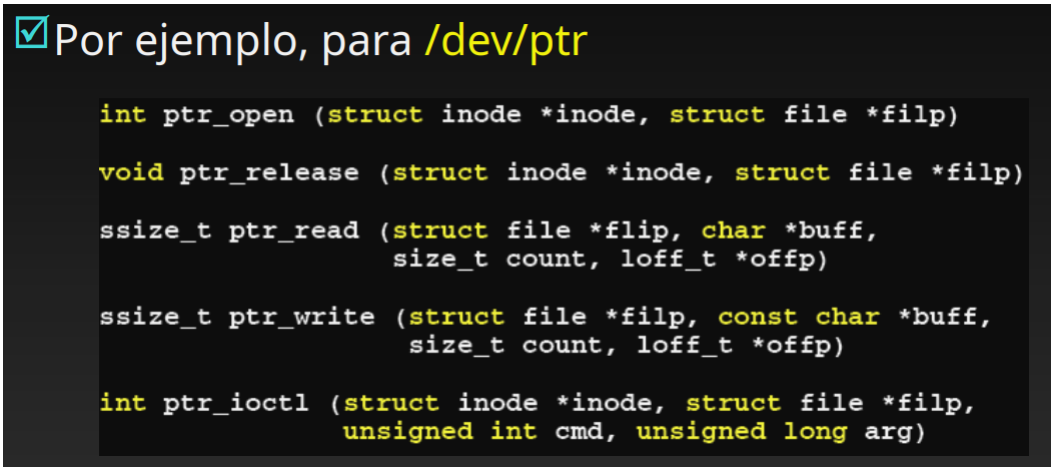
Driver – Ejemplo en Linux

- Linux distingue 3 tipos de dispositivos
  - Carácter: I/O programada o por interrupciones
  - Bloque: DMA
  - Red: Puertos de comunicaciones

- Los drivers se implementan como módulos que se cargan dinámicamente
- Debe tener al menos estas operaciones
    - Init_module: Para instalarlo
    - Cleanup_module: Para desinstalarlo
- Otras operaciones que debe contener para I/O
    - Open: Abrir el dispositivo
    - Release: Liberar el dispositivo (cerrar)
    - Read: Leer bytes del dispositivo
    - Write: Escribir bytes en el dispositivo
    - IOCTL: Orden de control sobre el dispositivo



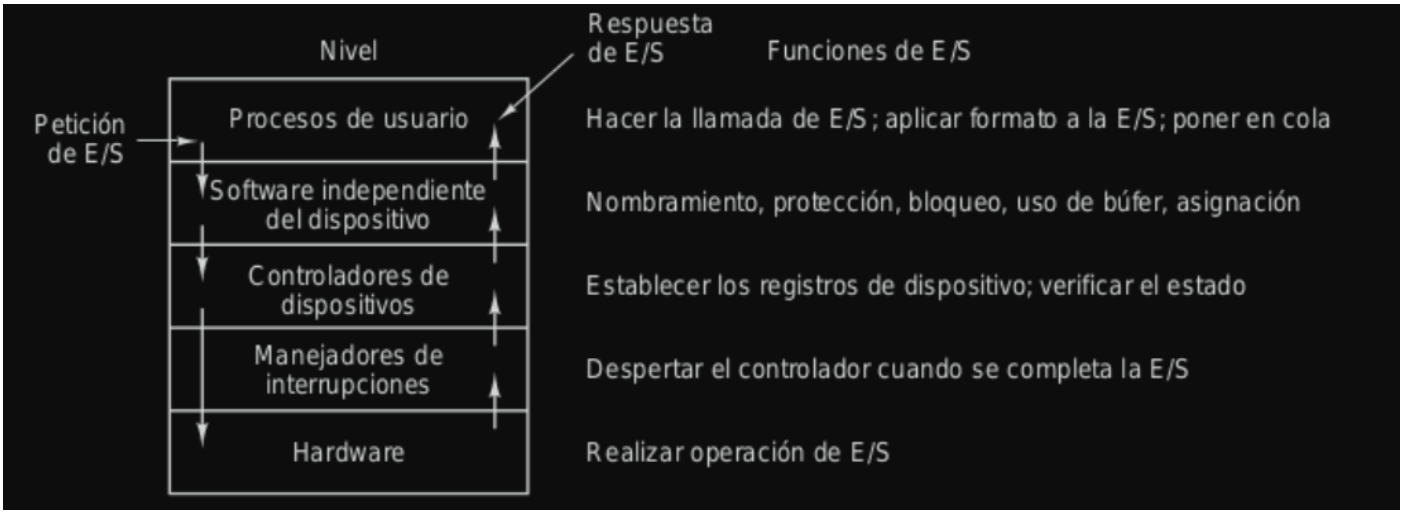
Por convención, los nombres de las operaciones comienzan con el nombre del dispositivo



Gestor de interrupciones

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

Ciclo de atención de un requerimiento

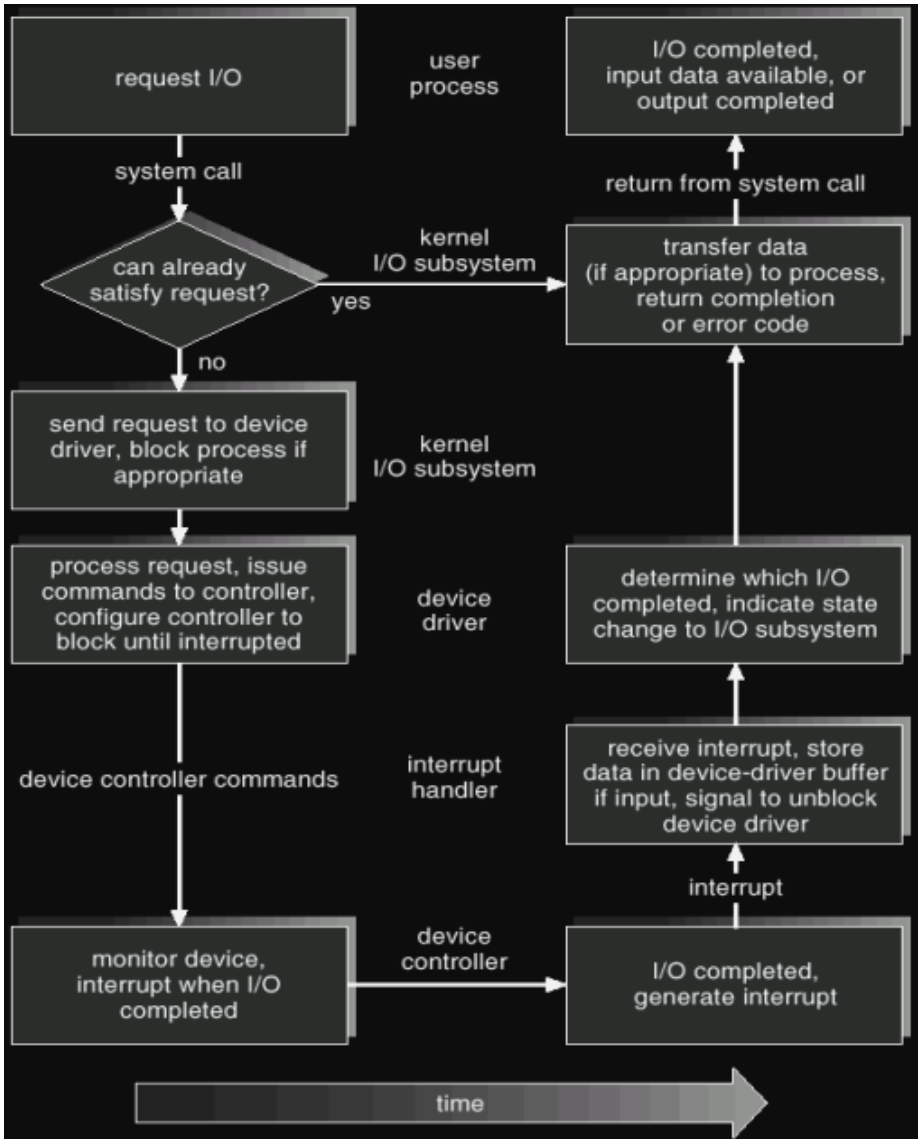


Desde el requerimiento de I/O hasta el HW

Considerando la lectura sobre un archivo en un disco

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

Ciclo de vida de un requerimiento



Rendimiento / Performance

- I/O es uno de los factores que más afectan a la performance del sistema
    - Utiliza mucho la CPU para ejecutar los drivers y el código del subsistema de I/O
    - Provoca context switches ante las interrupciones y bloqueos de los procesos
    - Utiliza el bus de memoria en copia de datos
        - Aplicaciones (espacio usuario) – Kernel
        - Kernel (memoria física) – Controlador

Mejorar el rendimiento

- 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
    - Controladores más inteligentes
    - Polling, si se minimiza la espera activa
- Utilizar DMA

Hardware y software involucrado

- Buses
- Controladores

- Dispositivos
- Puertos de E/S
- Drivers
- Comunicación con controlador del dispositivo
    - I/O Programada
    - Interrupciones
    - DMA

Comunicación CPU-Controladora

- Para que la CPU ejecute comandos o envíe/reciba datos de una controladora de un dispositivo, la controladora tiene uno o más registros
    - Registros para señales de control
    - Registros para datos
- La CPU se comunica con la controladora escribiendo y leyendo en dichos registros

Comandos de I/O

- CPU emite direcciones para identificar el dispositivo
- CPU emite comandos
    - Control
        - Qué hacer
        - Ej: Girar el disco
    - Test
        - Controlar el estado
        - Controlar errores
    - Read/Write
        - Transferir información desde/hacia el dispositivo

Mapeo de E/S y E/S aislada

- Correspondencia en memoria – Memory Mapped I/O
    - Dispositivos y memoria comparten el espacio de direcciones
    - I/O es como escribir/leer en la memoria
    - No hay instrucciones especiales para I/O
- Aislada – Isolated I/O
    - Espacio separado de direcciones
    - Uso de puertos de I/O
    - Se necesitan líneas de I/O
    - Instrucciones especiales
        - Conjunto limitado

Técnicas de I/O

Programada

- La CPU tiene control directo sobre la I/O
    - Controla el estado
    - Comandos para leer y escribir
    - Transfiere los datos
- CPU espera que el componente de I/O complete la operación
- Se desperdician ciclos de CPU

Polling

- En la I/O programada es necesario hacer polling del dispositivo para determinar el estado del mismo
    - Listo para recibir comandos
    - Ocupado

    - Error
- Ciclo de “Busy-Wait” para realizar la I/O
- Puede ser muy costoso si la espera es muy larga

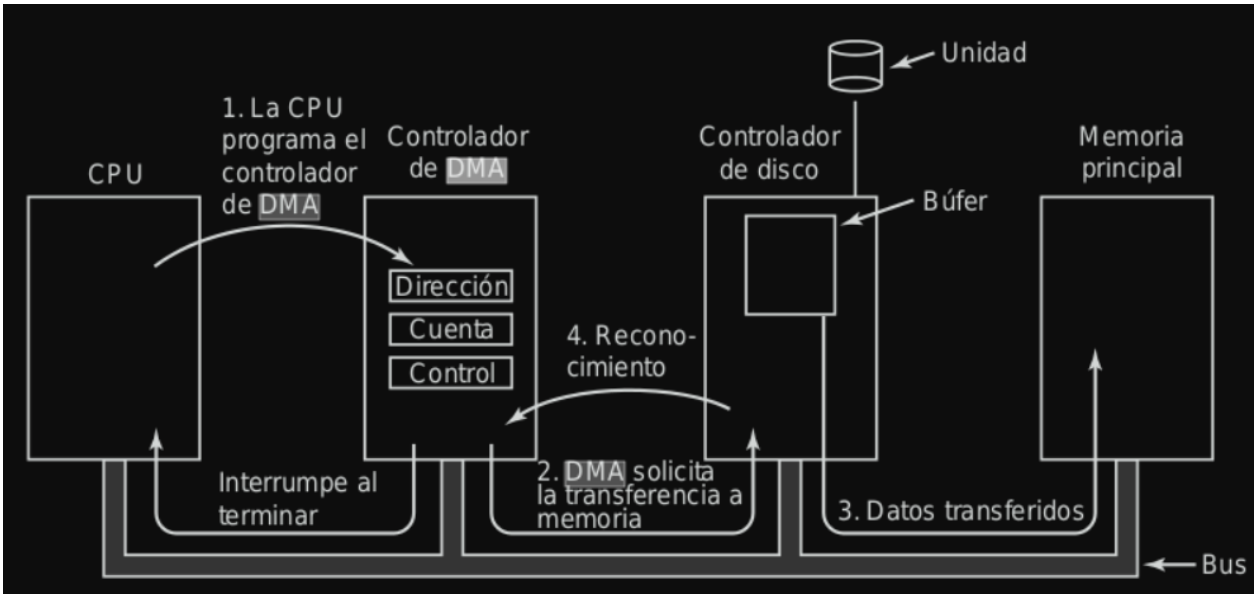
Manejada por interrupciones

- Soluciona el problema de la espera de la CPU
- La CPU no repite el chequeo sobre el dispositivo
- El procesador continúa la ejecución de instrucciones
- El componente de I/O envía una interrupción cuando termina

DMA – Direct Memory Access

- Un componente de DMA controla el intercambio de datos entre la memoria principal y el dispositivo
- El procesador es interrumpido luego de que el bloque entero fue transferido

Pasos para una transferencia DMA



Archivos

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

Punto de vista del Usuario

- Qué operaciones se pueden llevar a cabo
- Cómo nombrar a un archivo
- Cómo asegurar la protección
- Cómo compartir un archivo
- No tratar con aspectos físicos

Punto de vista del diseño

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

Sistema de manejo de archivos

- Conjunto de unidades de software que proveen los servicios necesarios para la utilización de archivos (creación, borrado, búsqueda, lectura, escritura, copiado, etc)
- Facilita el acceso a los archivos por parte de las aplicaciones
- Permite la abstracción al programador, en cuanto al acceso de bajo nivel (el programador no desarrolla el software de administración de archivos)

Objetivos del SO en cuanto a archivos

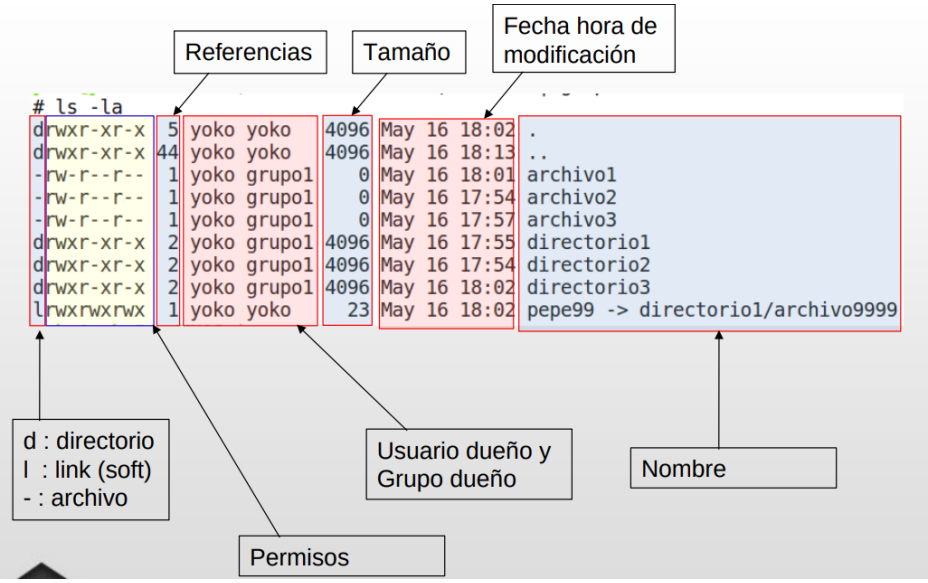
- Cumplir con la gestión de datos
- Cumplir con las solicitudes del usuario
- Minimizar o eliminar la posibilidad de perder o destruir datos
    - Garantizar la integridad de los datos
- Dar soporte de E/S a distintos dispositivos
- Brindar un conjunto de interfaces de E/S para tratamiento de archivos

Tipos de archivos

- Regulares
    - Texto plano
        - Source files
    - Binarios
        - Object Files
        - Ejecutables
- Directorios
    - Archivos que mantienen la estructura en el filesystem

Atributos de un archivo

- Nombre
- Identificador
- Tipo
- Localización
- Tamaño
- Protección, Seguridad y Monitoreo
    - Dueño, Permisos, Contraseña
    - Momento en el que fue creado, modificado y accedido
    - ACLs (Listas de control de acceso)



Directorios

- Contiene información acerca de archivos y directorios dentro de él
- Es un archivo
- Interviene en la resolución entre el nombre y el archivo
- Operaciones en directorios
    - Búsqueda de archivos
    - Creación de archivos
    - Borrado de archivos
    - Listado
    - Etc.
- Su uso ayuda con
    - 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

Estructura de directorios

- Los archivos pueden ubicarse siguiendo un PATH desde el directorio raíz y sus sucesivas referencias
- Distintos archivos pueden tener el mismo nombre, pero el full pathname es único
- El directorio actual es llamado “directorio de trabajo”
- Dentro de este directorio se pueden referenciar a los archivos tanto por su path absoluto como por su path relativo

Identificación de archivos

- Path Absoluto
    - El nombre incluye todo el camino del archivo
- Path Relativo
    - El nombre se calcula relativamente al directorio en el que se esté

Compartir archivos

Debe ser realizado bajo un esquema de protección

- Derechos de acceso
- Manejo de accesos simultáneos

Protección

- El dueño del archivo y el administrador debe ser capaz de controlar
    - Qué se puede hacer
        - Derechos de acceso
    - Quién lo puede hacer

Derechos de acceso

- Ejecución / Execute
    - El usuario puede ejecutar el archivo
- Lectura / Read
    - El usuario puede leer el archivo
- Añadir / Append
    - El usuario puede agregar datos pero no modificar o borrar el contenido del archivo
- Actualización / Update
    - El usuario puede modificar, borrar y agregar datos
    - Incluye la creación y sobreescritura de archivos
- Cambiar protección / Change protection

    - El usuario puede modificar los derechos de acceso
- Borrado / Deletion
    - El usuario puede borrar el archivo

Owner (propietario)

- Tiene todos los derechos sobre el archivo
- Puede dar derechos a otros usuarios. Se determinan clases
    - Usuario específico
    - Grupos de usuario
    - Todos

Permisos en Linux

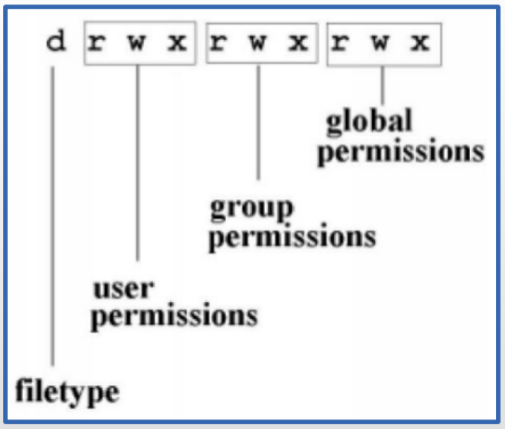
☑ Derechos de acceso son definidos independientemente para:
- ✓ (u) user – Owner (creator) of a file
- ✓ (g) group – Group
- ✓ (o) other – all other users of the UNIX system

☑ **Derechos de Acceso:**
- ✓ **(r) Read access right; List right for directory**
- ✓ **(w) Write access right; Includes delete/append rights**
- ✓ **(x) Execute access right; Traverse right for directories**

Los permisos que se pueden dar o quitar son:

- r - de lectura
- w - de escritura
- x - de ejecución

```
$ ls -l
drwxrwxr-x 4 www     www     ..
-rw-rw-r-- 1 www     www     x_windows.tex
lrwxrwxrwx 1 lee     lee     img -> ../linux/img/
-rw-rw-r-- 1 lee     lee     test.log
```



Metas del Sistema de Archivos

- Brindar espacio en disco a los archivos de usuario y del sistema
- Mantener registro del espacio libre. Cantidad y ubicación del mismo dentro del disco

Conceptos

- Sector
    - Unidad de almacenamiento utilizada en discos rígidos
    - Unidad física más pequeña de almacenamiento
- Bloque/Cluster
    - Conjuntos de sectores consecutivos
    - Es la unidad mínima de asignación de espacio para un archivo
    - Un cluster grande da más fragmentación interna si se trabaja con archivos pequeños
- File Sytem
    - Define la forma en que los datos son almacenados
- File Allocation Table (FAT)
    - Contiene información sobre en qué lugar están alocados los distintos archivos

Preasignación

- 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
- Si un archivo supera el espacio asignado hay que ver de reasignar el nuevo espacio
- Esta técnica suele usar la forma de asignación continua, pero puede usar otras
- Puede tener fragmentación interna

Asignación dinámica

- El espacio se solicita a medida que se necesita
- Los bloques de datos pueden quedar de manera no contigua
- Puede tener fragmentación externa

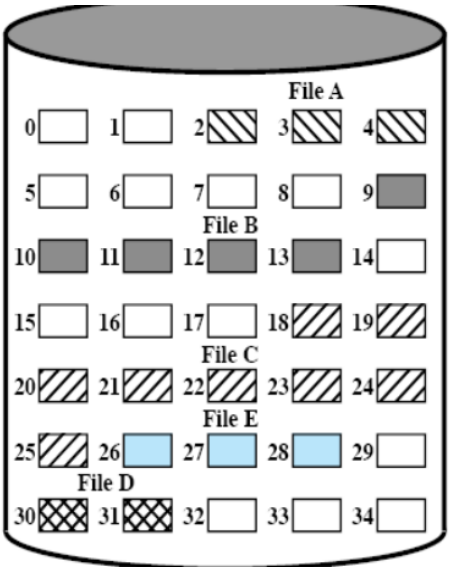
Formas de asignación

Continua

- Se utilizan conjuntos continuos de bloques
- Se requiere una preasignación
  - Se debe conocer el tamaño del archivo durante su creación
- FAT Simple
  - Sólo una entrada que incluye bloque de inicio y longitud
- El archivo puede ser leído con una única operación
- Puede existir fragmentación externa
  - Posibilidad de compactación o “desfragmentación”

Problemas de la asignación continua

- Encontrar bloques libres continuos en el disco
- Incremento del tamaño de un archivo



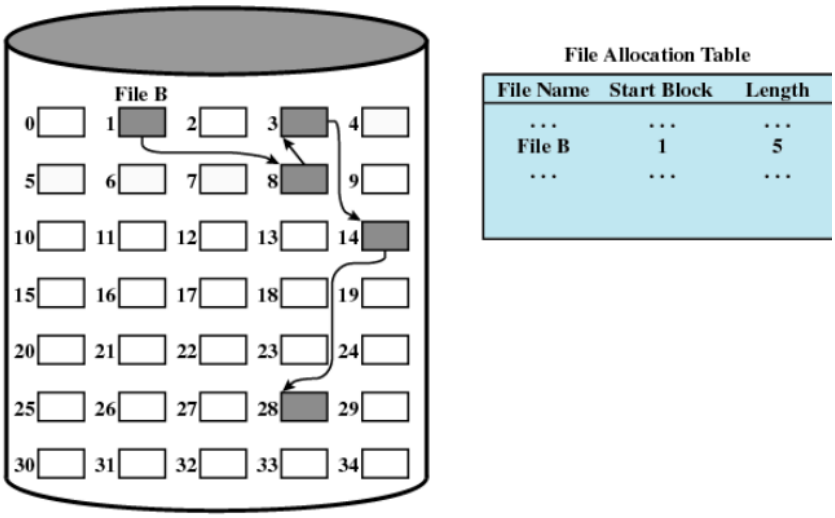
File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Encadenada

- Asignación en base a bloques individuales
- Cada bloque tiene un puntero al próximo bloque del archivo
- FAT simple
  - Única entrada por archivo que contiene el bloque de inicio y tamaño del archivo
- No hay fragmentación externa
- Útil para acceso secuencial
- Los archivos pueden crecer bajo demanda

- No se requieren bloques contiguos
- Se pueden consolidar los bloques de un mismo archivo para garantizar cercanía de los bloques de un mismo archivo

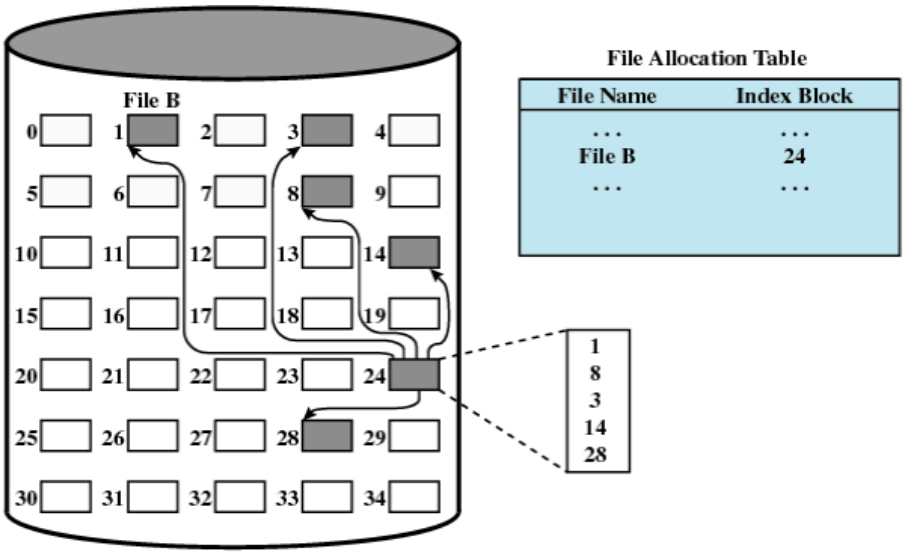


Indexada

- La FAT
  - Contiene un puntero al bloque índice
  - El bloque índice no contiene datos propios del archivo, sinó que contiene un índice a los bloques que lo componen
  - Única entrada con la dirección del bloque de índices (inode)
  - Simple
- Asignación en base a bloques individuales
- No se produce fragmentación externa
- El acceso aleatorio a un archivo es eficiente

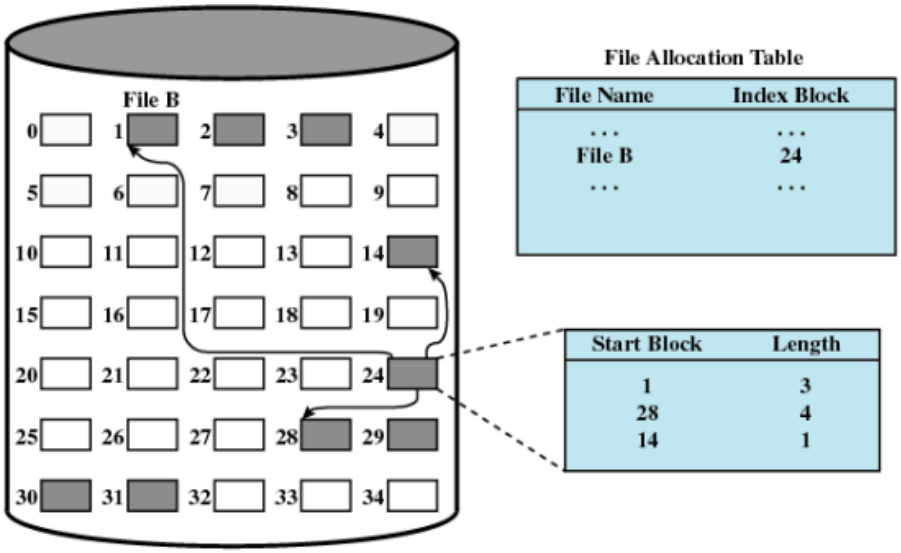
Limitaciones de la asignación indexada

- Más espacio ocupado por tener que usar un bloque índice
- La cantidad de índices es limitada y eso limita el tamaño de los archivos



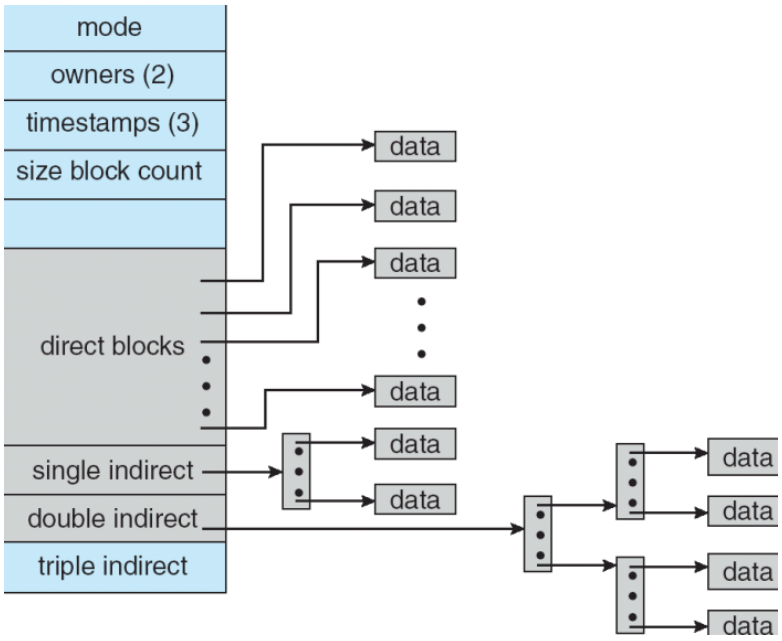
Indexada: Variante “Asignación por secciones”

- A cada entrada del bloque índice se agrega el campo de longitud
- El índice apunta al primer bloque de un conjunto almacenado de manera contigua



Indexada: Variante "Niveles de indirección"

- Existen bloques directos de datos
- Otros bloques son considerados como bloques índices (apuntan a varios bloques de datos)
- Puede haber varios niveles de indirección



Ejemplo de cálculo con asignación indexada

Cada I-NODO contiene 9 direcciones a los bloques de datos, organizadas de la siguiente manera:

- 7 de direccionamiento directo.
- 1 de direccionamiento indirecto simple
- 1 de direccionamiento indirecto doble

Si cada bloque es de 1KB y cada dirección usada para referenciar un bloque es de 32 bits:

- ✓ ¿Cuántas referencias (direcciones) a bloque pueden contener un bloque de disco?

  1 KB / 32 bits = 256 direcciones

- ✓ ¿Cuál sería el tamaño máximo de un archivo?

  $(7 + 256 + 256^2) * 1\ KB = 65799\ KB = 64{,}25\ MB$

Gestión del espacio libre

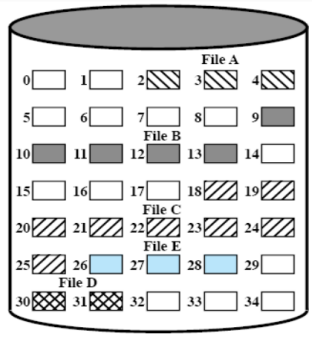
- Control sobre cuáles de los bloques de disco están disponibles
- Alternativas
  - Tabla de bits
  - Bloques libres encadenados
  - Indexación
  - Recuento

Tabla de bits

- Tantas entradas como clusters haya
- Tabla con 1 bit por cada bloque de disco
- Cada entrada
  - 0 -> Bloque libre
  - 1 -> Bloque en uso
- Ventaja
  - Fácil encontrar un bloque o grupo de bloques libres
- Desventaja
  - Tamaño del vector en memoria

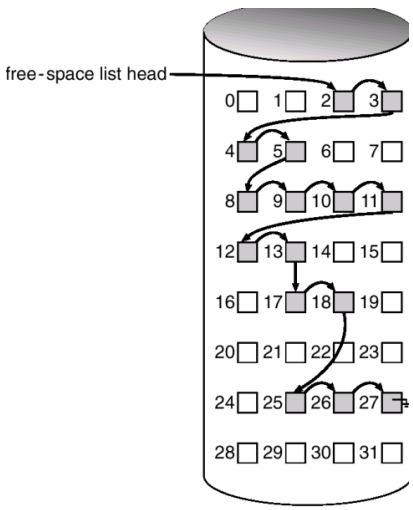
☑ Ejemplo

00111
00001
11110
00011
11111
11110
11000



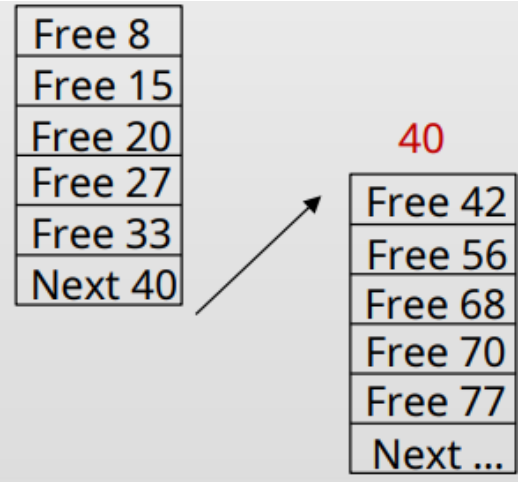
Bloques encadenados

- Se tiene un puntero al primer bloque libre
- Cada bloque libre tiene un puntero al siguiente bloque libre
- Ineficiente para la búsqueda de bloques libres
  - Hay que realizar varias operaciones de E/S para obtener un grupo de bloques libres
- Problemas con la pérdida de un enlace
- Difícil encontrar bloques libres consecutivos



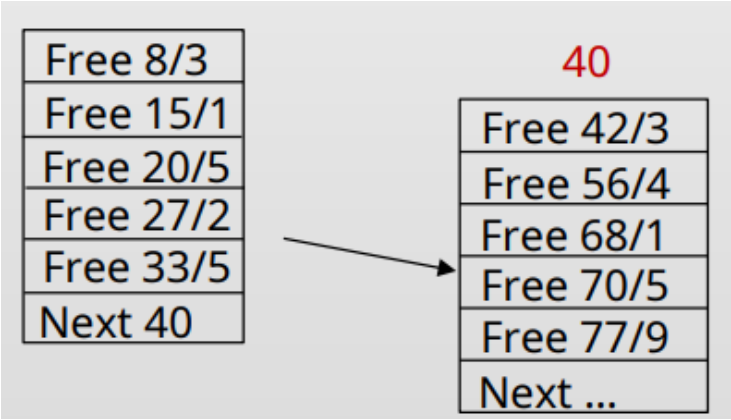
Indexación / Agrupamiento

- 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



Recuento

- Variante de “indexación”
- Esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o liberados a la vez (en especial con asignación continua)
- 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
- Busca achicar estructuras
- Es más compleja de mantener



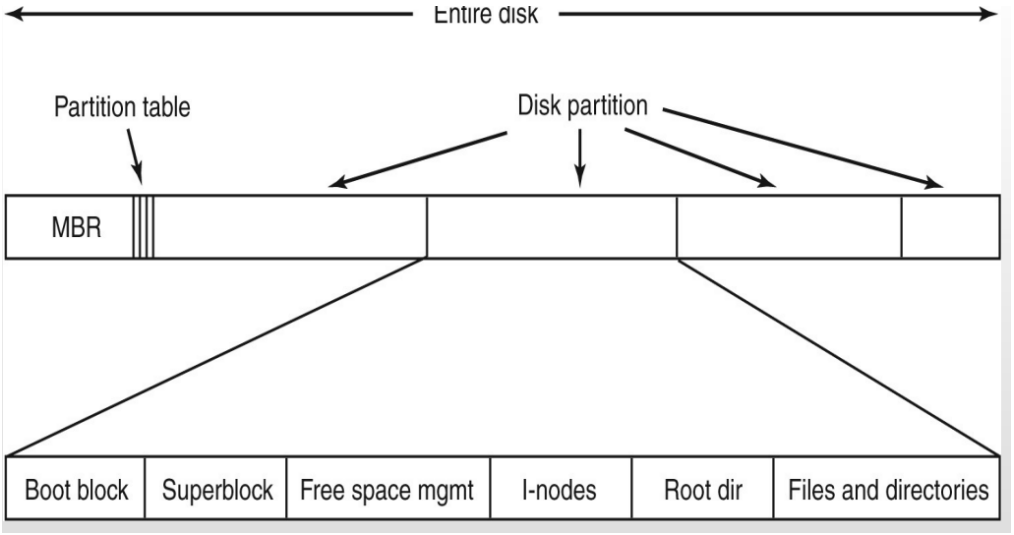
Manejo de archivos en UNIX

Tipos de archivos

- Comunes
- Directorios
- Especiales (dispositivos /dev/sda)
- Named pipes (comunicación entre procesos)
- Links (comparten el inode sólo dentro del mismo filesystem)
- Links simbólicos (tienen inodo propio, para diferentes filesystems)

Estructura del volumen

- Cada disco físico puede dividirse en uno o más volúmenes
- Cada volumen o partición contiene un sistema de archivos
- Cada sistema de archivos contiene
    - Boot Block: Código para bootear el SO
    - Superblock: Atributos sobre el filesystem
        - Bloques/Clusters libres
    - Inode Table: Tabla que contiene todos los Inodos
        - Inodo: Estructura de control que contiene la información clave de un archivo
    - Data blocks: Bloques de datos de los archivos



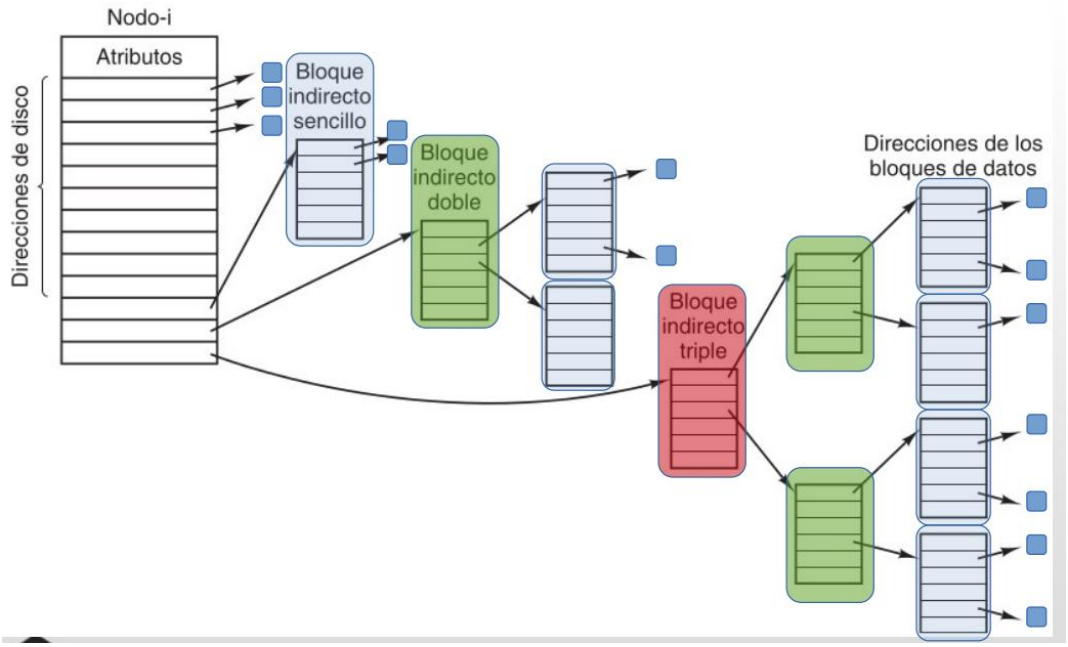
Información del inodo

- Un inodo es una estructura de datos del filesystem que posee información sobre cada archivo, directorio u objeto que se almacene en el sistema de archivos
- Notar que el nombre del archivo no se almacena en esta estructura
    - Los nombres de archivos se almacenan en los directorios

Información que se guarda en un inodo

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

En Linux, los bloques son indexados con hasta 3 niveles de indexación

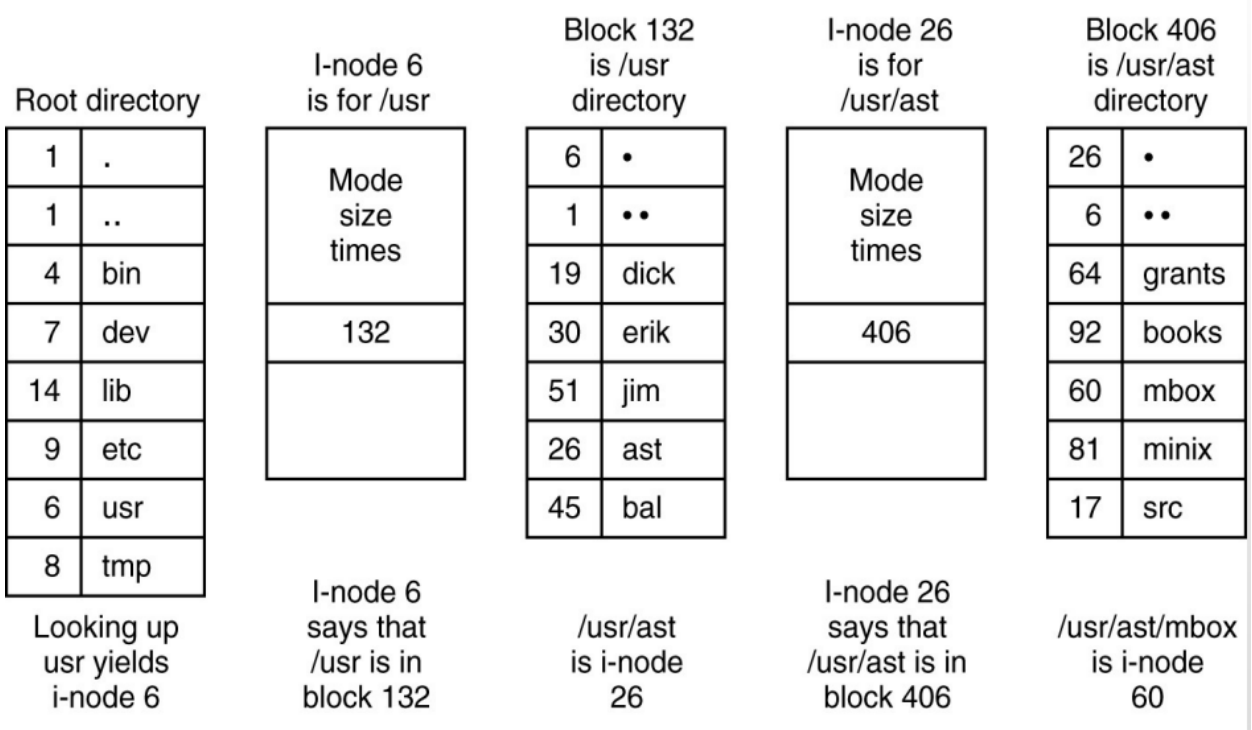


Directorios

- Los nombres de los archivos se almacenan en los directorios
- El directorio es una tabla de tuplas del tipo “N° de inodo, nombre de archivo”

.	1
..	1
unix	117
etc	4
home	18
pro	36
dev	93

Ejemplo de búsqueda de un archivo

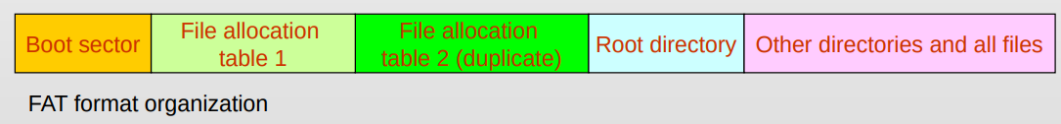


File Systems soportados en Windows

- CDRom File System (CDFS) -> CD
- Universal Disk Format (UDF) -> DVD, Blu-Ray
- File Allocation Table (FAT)
    - Fat12 -> Floppy
    - Fat16 -> MsDos, Nombres cortos de archivo
    - Fat32 -> MsDos, Nombres largos, pero no soportados en MsDos
- New Technology File System (NTFS)

File Allocation Table (FAT)

- Las distintas versiones de FAT se diferencian por un número que indica la cantidad de bits que se usan para identificar diferentes bloques o clusters
- Windows aun soporta FAT por
    - Compatibilidad con otros SO
    - Permitir actualizaciones de versiones viejas
    - Formato de dispositivos como diskettes
- Se utiliza un mapa de bloques del sistema de archivos llamado FAT
- La FAT tiene tantas entradas como bloques/clusters
- Tiene un duplicado para redundancia y recuperación ante errores
- La FAT, su duplicado y el directorio raíz se almacenan en los primeros sectores de la partición



FAT format organization

- Se utiliza un esquema de asignación encadenada
- Los atributos de los directorios se encuentran en cada bloque
- No hay permisos
- El puntero al próximo bloque está en la FAT y no en los bloques
- Los bloques libres y dañados tienen códigos especiales para su detección
- Los directorios mantienen información sobre el nombre de archivo, el primer bloque y el tamaño

DIRECTORIO

Nombre		1er bloque	Tamaño
FICH_A		7	4
FICH_B		4	1
FICH_C		2	3

FAT

	Tamaño del disco	0
		1
	6	2
X	14	3
	EOF	4
	EOF	5
	5	6
X	3	7
X	EOF	8
	LIBRE	9
	LIBRE	10
	LIBRE	11
	LIBRE	12
	DAÑADO	13
X	8	14
	LIBRE	15
	...	...

FAT12

- Se usan 12 bits de identificación de sector (se limita a 2^{12} sectores)
- Windows utiliza tamaños de sector desde los 512 bytes hasta los 8KB (16 bloques consecutivos), lo que limita a un tamaño total de volumen de 32Mb
- Windows utiliza FAT12 como sistema de archivos de los diskettes

FAT16

- Cada sector puede tener hasta 2^{16} sectores en un volumen
- El tamaño de sector varía desde los 512 bytes hasta los 64KB (128 sectores consecutivos), lo que limita a un tamaño máximo de volumen de 4GB
- El tamaño de sector dependía del tamaño del volumen al formatearlo

FAT32

- Utiliza 32 bits para identificar los sectores, pero se reservan los 4 bits superiores, con lo cual se utilizan 28 bits reales para la identificación
- El tamaño de sector puede de hasta 32KB, con lo cual tiene una capacidad teórica de direcciones volúmenes de hasta 8TB
- El modo de identificación y acceso de los sectores lo hace más eficiente que FAT16. Con tamaño de sector de 512 bytes puede direccionar volúmenes de hasta 128GB

NTFS

- Filesystem nativo de Windows
- Se utilizan 64 bits para referenciar sectores
    - Permite tener volúmenes de hasta 16 Exabytes
- Se prefiere NTFS en lugar de FAT porque
    - Tamaños de archivo y de disco mayores
    - Mejora performance en discos grandes
    - Nombres de archivos de hasta 255 caracteres
    - Atributos de seguridad
    - Transaccional
- FAT es más simple y más rápido para ciertas operaciones que NTFS
- Usa Arboles B
- No tiene límite del tamaño del archivo (aunque está limitado por la estructura del sistema de archivos)

Caché de disco / Disk Cache

- Buffers en memoria principal para almacenamiento temporario de bloques de disco
- El objetivo es minimizar la frecuencia de acceso al disco

Observaciones

- Cuando un proceso quiere acceder a un bloque de la caché hay dos alternativas
    - Se copia el bloque al espacio de direcciones del usuario
        - No permitiría compartir el bloque entre procesos
    - Se trabaja como memoria compartida
        - Permite acceso a varios procesos al mismo bloque
        - Esta área de memoria debe ser limitada, con lo cual debe existir un algoritmo de reemplazo

Estrategia de reemplazo

- Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo que no es referenciado
- Es una lista de bloques, donde el último es el más recientemente usado (LRU, Least Recently Used)
- Cuando un bloque se referencia o entra en la caché, queda al final de la lista
- No se mueven los bloques en la memoria
    - Se asocian punteros

- Otra alternativa es el algoritmo Least Frequently Used. Donde se reemplaza el que tenga menor número de referencias

Objetivo y estructura

- Minimizar la frecuencia de acceso al disco
- Es una estructura formada por buffers en memoria principal
- El kernel asigna un espacio en la memoria principal durante la inicialización para esta estructura
- El módulo de buffer-cache es independiente del sistema de archivos y los dispositivos de hardware
- Es un servicio del SO
- Un Buffer-Caché tiene dos partes
    - Headers
        - Contienen información para modelar la ubicación del bloque en RAM, número del bloque, estado, relaciones entre headers, etc.
    - El buffer en sí
        - El lugar en RAM (referenciado por el header) donde se almacena realmente el bloque del dispositivo llevado a memoria

El Header

- Identifica el N° de dispositivo y N° de bloque
- Estado
- Punteros a
    - 2 punteros para la hash queue
    - 2 punteros para la free list
    - 1 puntero al bloque en memoria
- Se usan 2 punteros porque son listas doblemente enlazadas
- Se pone al buffer al final de la free list cuando deja de estar ocupado
- Se pone al buffer al inicio de la free list si su estado era DW y se mandó a escribir a disco
- Un proceso puede tener más de un header

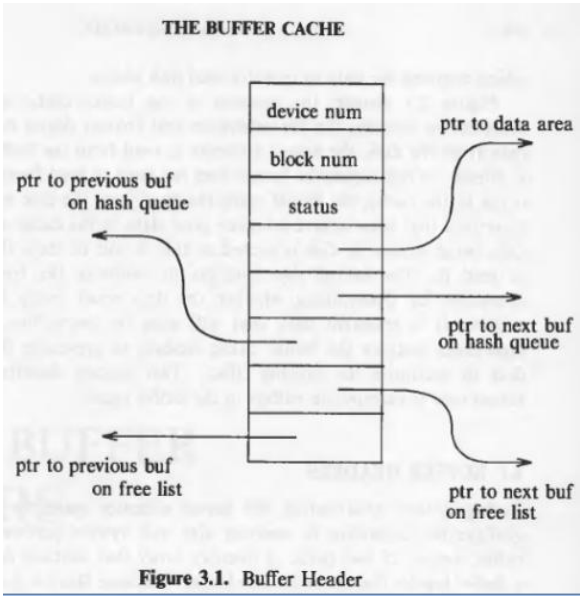


Figure 3.1. Buffer Header

Estados de los buffers

- Free / disponible
- Busy / no disponible (en uso por algún proceso)
- Se está escribiendo / 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 vacíos
  - El proceso puede haber terminado, liberado el bloque, pero siguen en estado “Delayed Write”
- Se ordena según LRU (Least Recently Used)
- Sigue el mismo esquema de la hash queue pero contiene los headers de los buffers de aquellos procesos que ya han terminado
- Si el proceso que referenciaba a un buffer terminó, el buffer va a estar en la hash queue y en la free list

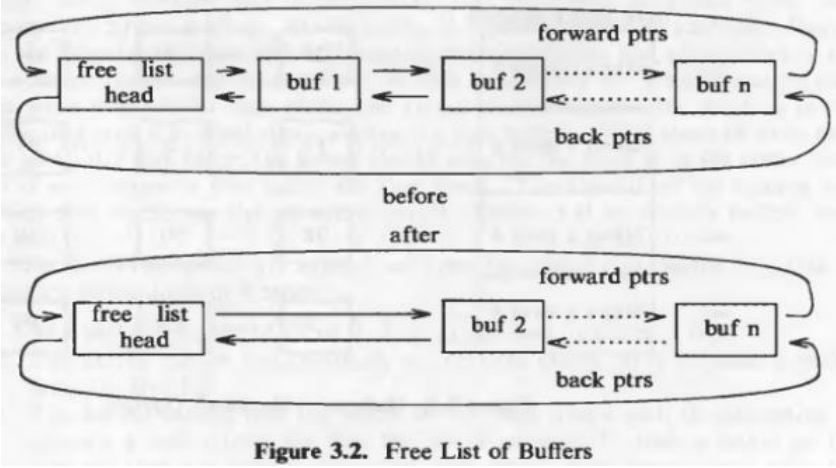


Figure 3.2. Free List of Buffers

Hash Queues

- Son colas para optimizar la búsqueda de un buffer en particular
- Los headers de los buffers se organizan según una función de hash usando “dispositivo, #bloque”
- Al número de bloque (dispositivo / bloque) se le aplica una función de hash que permite agrupar los buffers cuyo resultado dio igual para hacer que las búsquedas sean más eficientes
- Se busca que la función de hash provea alta dispersión para lograr que las colas de bloques no sean tan extensas
- Para agrupar los bloques se utilizan los punteros que se almacenan en el header
- El header de un buffer siempre está en la hash queue

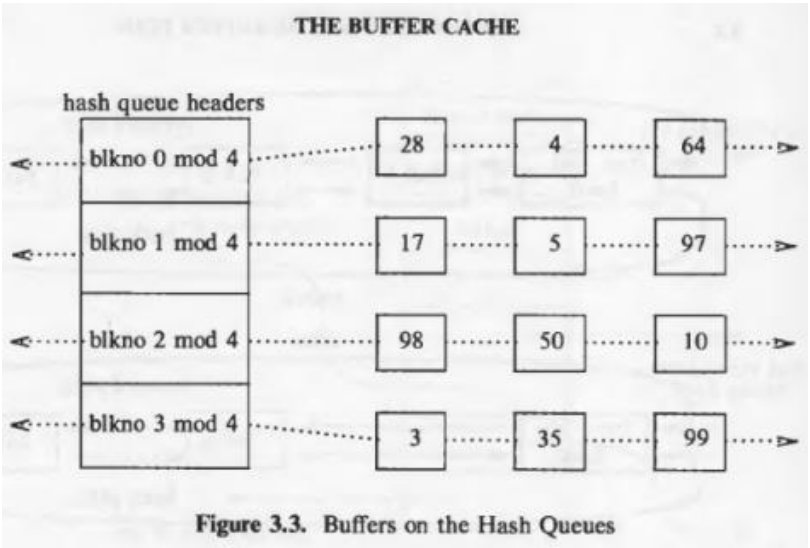


Figure 3.3. Buffers on the Hash Queues

Funcionamiento del buffer caché

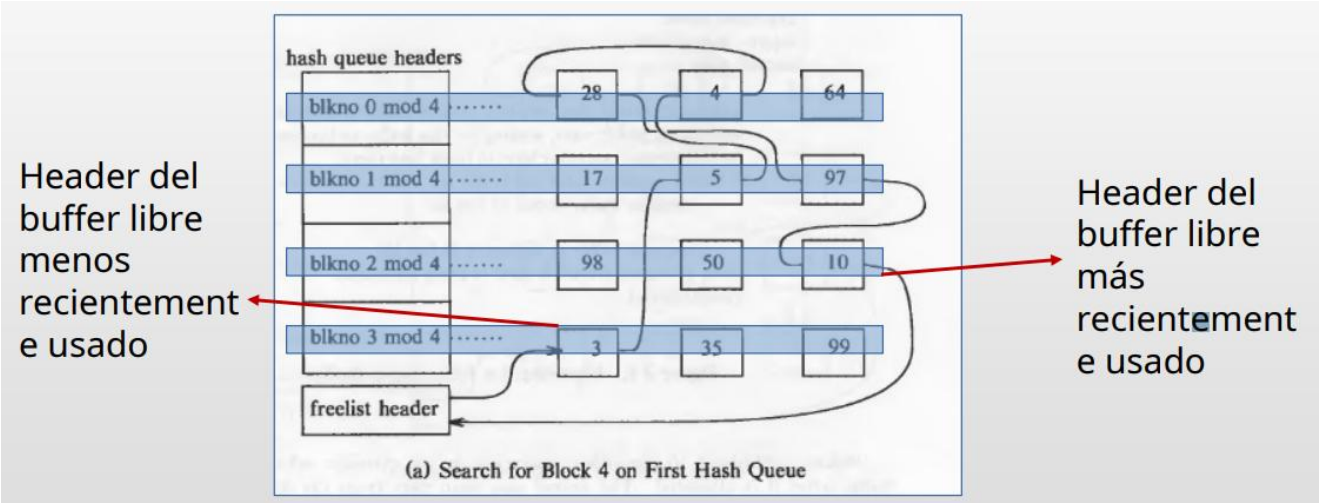
- Cuando un proceso quiere acceder a un archivo, se utiliza su inodo para localizar los bloques de datos donde se encuentra éste

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

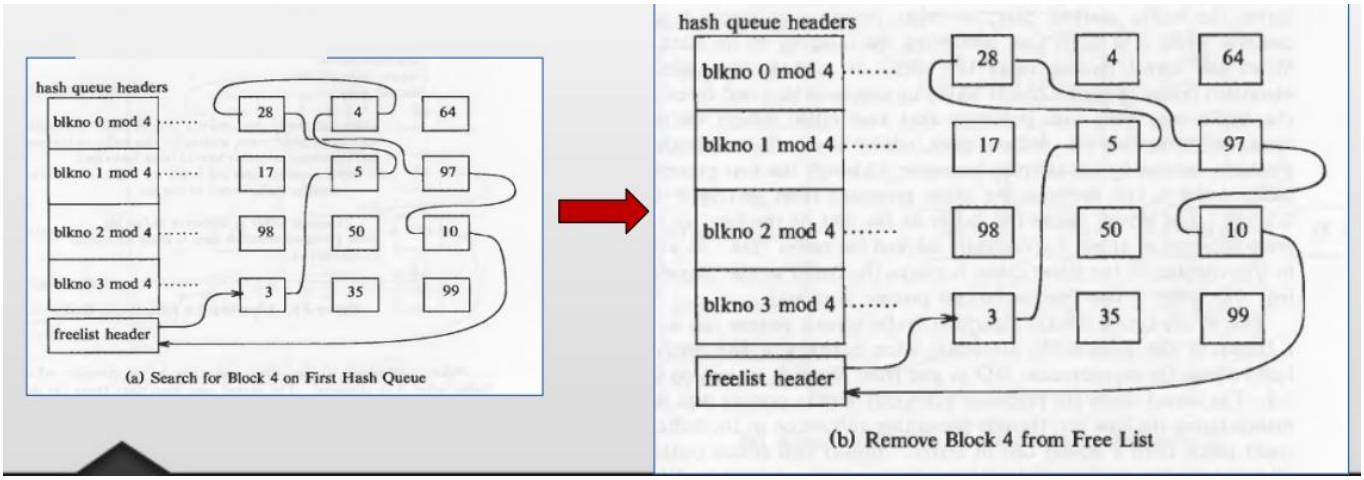
1er escenario: El kernel encuentra el bloque en la hash queue y el buffer está libre

Ejemplo usando el bloque 4

- El kernel encuentra el bloque en la hash queue
- Está disponible



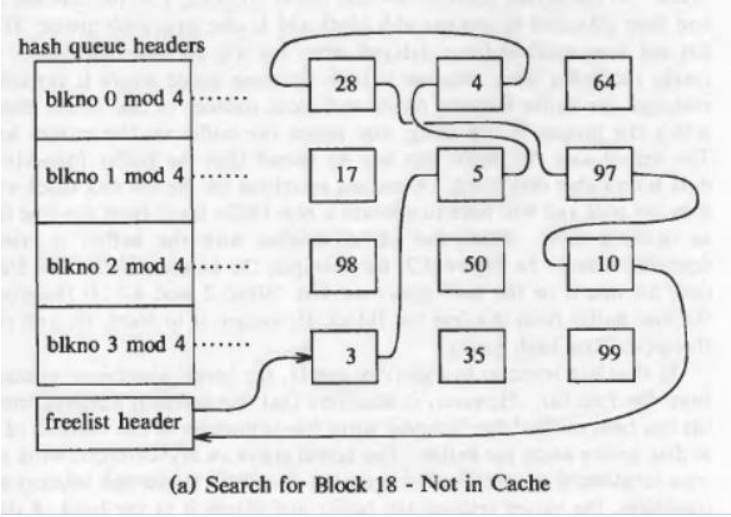
- Se remueve el buffer 4 de la free list
- Pasa el buffer 4 a estado BUSY
- El proceso usa el bloque 4
- Se deben reacomodar los punteros de la free list



2do escenario: El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre

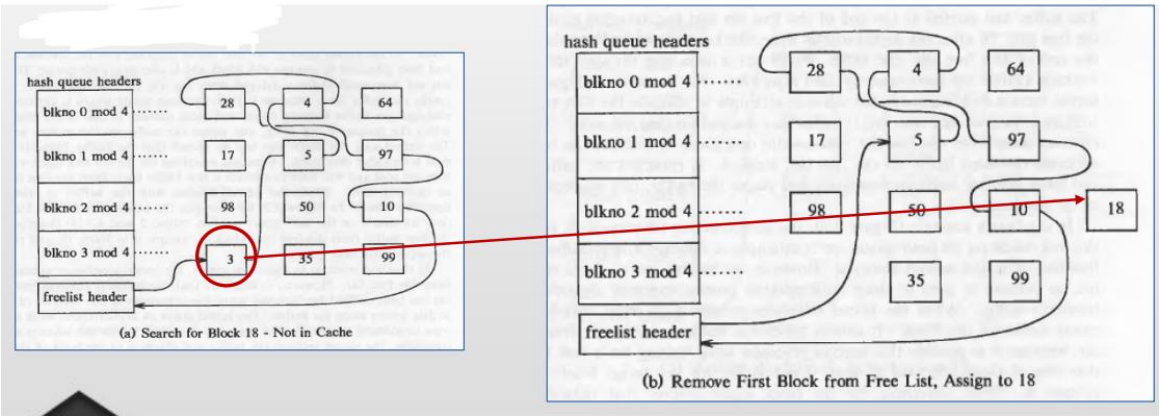
Ejemplo usando el bloque 18

- El bloque buscado no está en la hash queue
- Se debe buscar un buffer libre



(a) Search for Block 18 - Not in Cache

- Se toma un buffer de la free list (el 3)
- Siempre se usa el primer buffer
- Se lee del disco el bloque deseado en el buffer obtenido
- Se ubica el header en la hash queue correspondiente (sólo se cambian punteros, no ubicaciones de memoria)



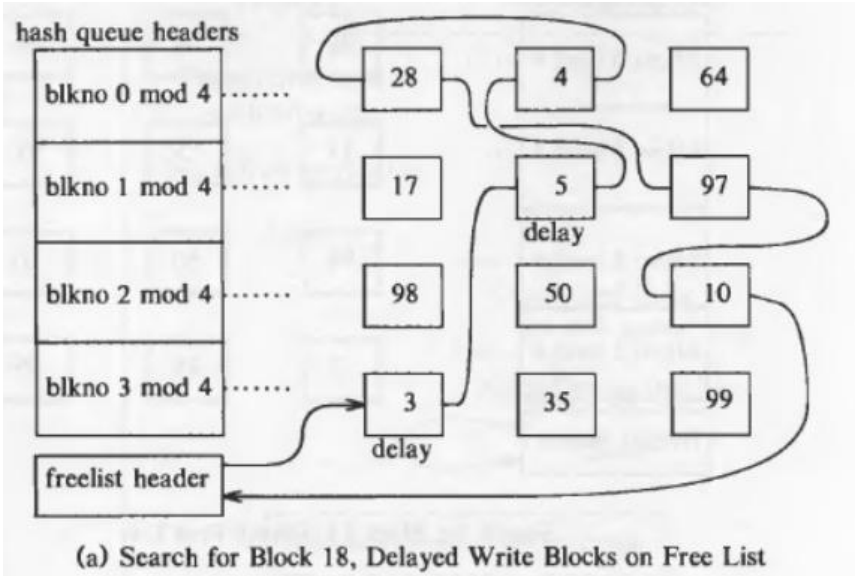
(a) Search for Block 18 - Not in Cache

(b) Remove First Block from Free List, Assign to 18

3er escenario: El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre, pero el buffer libre está marcado como DW

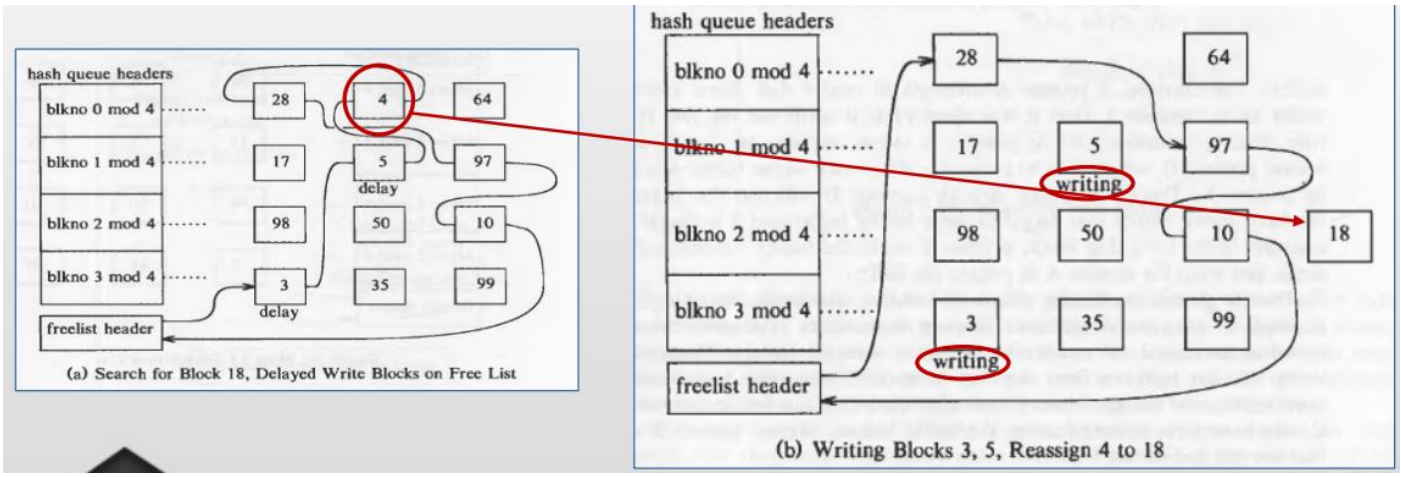
Ejemplo usando el bloque 18

- El kernel no encuentra el bloque buscado en la hash queue
- Debe tomar el 1ro de la free list, pero está marcado DW
- El kernel debe mandar a escribir a disco el bloque 3 y tomar el siguiente buffer de la free list



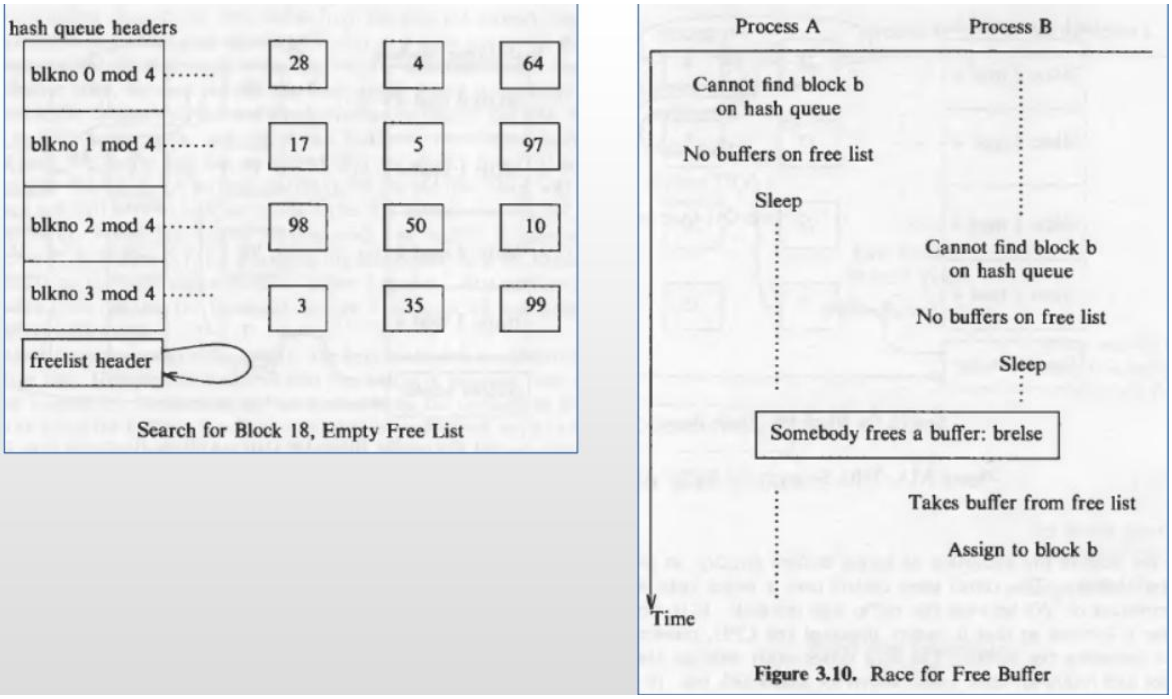
(a) Search for Block 18, Delayed Write Blocks on Free List

- Si el siguiente buffer también está como DW, sigue con el mismo proceso hasta encontrar uno que no esté marcado como DW
- Mientras los DW se escriben en disco, se asigna el siguiente buffer libre al proceso
- Una vez escritos a disco los bloques DW, estos son ubicados al principio de la free list



4to escenario: El kernel no encuentra el bloque en la hash queue y la free list está vacía

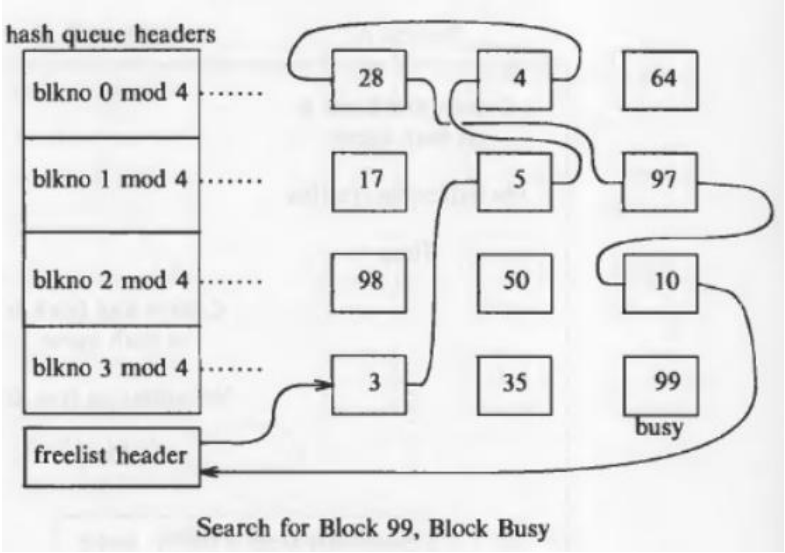
- El proceso queda bloqueado en espera a que se libere algún buffer
- Cuando el proceso despierta se debe verificar nuevamente que el bloque no esté en la hash queue (algún proceso pudo haberlo pedido mientras éste dormía)



5to escenario: El kernel encuentra el bloque en la hash queue pero el buffer está busy

Ejemplo usando el bloque 99

- El kernel busca un bloque y el buffer que lo contiene está marcado como busy
- El proceso se bloquea a la espera de que el buffer se desbloquee



Search for Block 99, Block Busy

- Eventualmente el proceso que tenía el buffer 99 lo libera
  - Se despiertan todos los procesos en espera de algún buffer
  - El proceso que buscaba el buffer 99 debe buscarlo nuevamente en la hashqueue y en la freelist

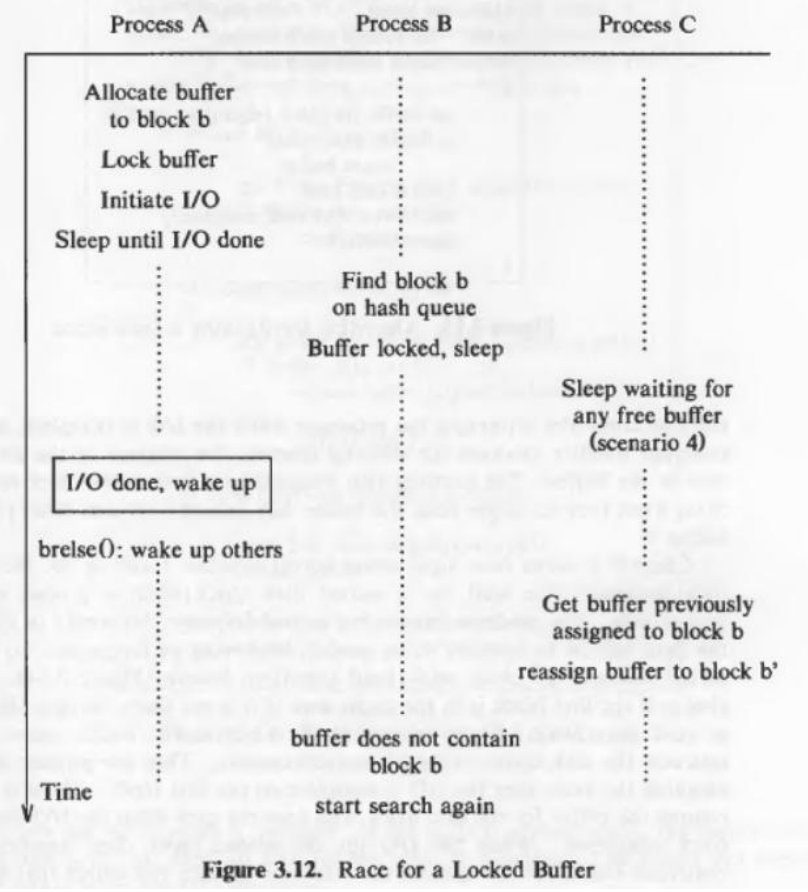


Figure 3.12. Race for a Locked Buffer

Algoritmo de asignación

```
algorithm getblk
input:   file system number
         block number
output: locked buffer that can now be used for block
{
        while (buffer not found)
        {
                if (block in hash queue)
                {
                        if (buffer busy)        /* scenario 5 */
                        {
                                sleep (event buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        mark buffer busy;       /* scenario 1 */
                        remove buffer from free list;
                        return buffer;
                }
                else        /* block not on hash queue */
                {
                        if (there are no buffers on free list)        /* scenario 4 */
                        {
                                sleep (event any buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        remove buffer from free list;
                        if (buffer marked for delayed write) {      /* scenario 3 */
                                asynchronous write buffer to disk;
                                continue;       /* back to while loop */
                        }
                        /* scenario 2 -- found a free buffer */
                        remove buffer from old hash queue;
                        put buffer onto new hash queue;
                        return buffer;
                }
        }
}
```