CSO - Segundo examen de promoción

Murray, Agustín

20 de febrero de 2024

Índice

1. Memoria II

1.1. Introducción

Hasta ahora vimos que usando paginación el espacio de direcciones de un proceso no necesariamente debe estar contiguo en la memoria para poder ejecutarse. Podemos pensar también que no todo el espacio de direcciones del proceso se necesito en todo momento, por ejemplo:

- Rutinas o librerías
- Partes del programa que no vuelven a ejecutarse
- Regiones de memoria alocadas dinámicamente y luego liberadas

Es decir, no hay necesidad de que la totalidad de la imagen del proceso sea cargada en memoria. El SO puede traer a memoria las "piezas" de un proceso a medida que éste las necesita.

1.2. Conjunto Residente

Definiremos como Conjunto Residente (o Working Set) a la porción del espacio de direcciones del proceso que se encuentra en memoria. Con el apoyo del HW:

- Se detecta cuando se necesita una porción del proceso que no esta en su Conjunto Residente
- Se debe cargar en memoria dicha porción para continuar con la ejecución

Ventajas

- Mas procesos pueden ser mantenidos en memoria. Con mas procesos en memoria principal es mas probable que existan mas procesos en estado Ready.
- Un proceso puede ser mas grande que la memoria principal. El usuario no debe preocuparse por el tamaño de sus programas. La limitación la impone el HW y el bus de direcciones.

Requisitos para el funcionamiento de la Memoria Virtual

- El HW debe soportar paginación por demanda (y/o segmentación por demanda)
- Un dispositivo de memoria secundaria (disco) que de apoyo para almacenar la secciones del proceso que no están en memoria principal (área de swap)

1.3. Memoria Virtual con Paginación

- Cada proceso tiene su tabla de paginas
- Cada entrada en la tabla referencia al frame o marco en el que se encuentra la pagina en la memoria principal
- Cada entrada en la tabla de paginas tiene bits de control
  - **Bit V:** Indica si la pagina esta en memoria (lo activa/desactiva el SO, lo consulta el HW)
  - **Bit M:** Indica si la pagina fue modificada. Si se modifico, en algún momento, se deben reflejar los cambios en Memoria Secundaria (lo activa/desactiva el HW, lo consulta el SO)

Entrada en la Tabla de paginas de x86 (32 bits)

- El HW define el formato de la tabla de paginas y el SO se adapta a el
- Una entrada valida tiene:
  - BIT V = 1
  - Page Frame Number (PFN) - Marco de memoria asociado
  - Flags que describen su estado y protección

1.4. Fallo de paginas (Page Fault)

- Ocurre cuando el proceso intenta usar una dirección que esta en una pagina que no se encuentra en la memoria principal. Bit V=0 (también marcado con i = invalido). La pagina no se encuentra en su **conjunto residente**
- El HW detecta la situación y genera un trap al SO
- El SO podrá colocar al proceso en estado de "Blocked"(espera) mientras gestiona que la pagina que se necesite se cargue.
- EL SO busca un marco libre en la memoria y genera una operación de E/S al disco para copiar en dicho marco la pagina del proceso que se necesita utilizar.
- El SO puede asignarle la CPU a otro proceso mientras se completa la E/S (La E/S se realizara y avisara mediante interrupción su finalización)

Cuando la operación de E/S finaliza, se notifica al SO y este:

- Actualiza la tabla de paginas del proceso
  1. Coloca el Bit V en 1 en la pagina en cuestión
  2. Coloca la dirección base del marco donde se coloco la pagina
- El proceso que genero el fallo de pagina vuelve a estado de Ready
- Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que antes genero el fallo de pagina

Resumiendo

- La técnica de paginación intenta alocar la mayor cantidad de paginas necesarias posibles
- Cada vez que hay que alocar una pagina en un marco, se produce un fallo de pagina

¿Que sucede si es necesario alocar una pagina y ya no hay marcos disponibles?

- Se debe seleccionar una pagina víctimas, para lo cual existen diversos algoritmos (FIFO, Óptimo, LRU, entre otros)

¿Cual es el mejor algoritmo?

- El que seleccione como pagina víctima aquella que no vaya a ser referenciada en un futuro próximo

1.5. Performance

- Si los page faults son excesivos, la performance del sistema decae
- Tasa de Page Faults $0 \leq p \leq 1$
  1. Si $p = 0$ no hay page faults
  2. Si $p = 1$, cada acceso a memoria genera un page fault
- Effective Access Time:

$$EAT = (1 - p) \cdot \text{memory access} + p \cdot (\text{page fault overhead} + \text{swap page out} + \text{swap page in} + \text{restart overhead})$$

1.6. Tabla de Paginas

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

Formas de organizarla: Depende del HW subyacente

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

1.7. Tabla de 1 nivel

32 bits Direcciones de 32 bits donde se asignan 20 bits al numero de pagina y 12 bits al desplazamiento dentro de la misma (por ejemplo).

- Cantidad de Page Table Entries (PTEs) máximas que puede tener un proceso = 2^{20}
- El tamaño de cada pagina es de 4KB 2^{12} bits
- El tamaño de cada PTE es de 4 bytes (32 bits), por lo tanto, la cantidad de PTEs que entran en un marco es: $\frac{4KB}{4B} = 2^{10}$ PTEs
- Tamaño de tabla de paginas
  - Cantidad de marcos necesarios para todas las PTEs de la tabla de paginas de un proceso = $\frac{2^{20}}{2^{10}} = 2^{10}$
  - Tamaño tabla de paginas del proceso = $2^{10} \cdot 4bytes = 4$MB por proceso

64 bits Direcciones de 64 bits donde, por ejemplo, se asignan 52 bits al numero de pagina y 12 bits al desplazamiento dentro de la misma.

- Cantidad de PTEs máxima que puede tener un proceso = 2^{52}
- El tamaño de cada pagina es de 4KB
- El tamaño de cada PTE es de 4 bytes. Cantidad de PTEs que entran en un marco: $\frac{4KB}{4B} = 2^{10}$
- Tamaño de la tabla de paginas
  - Cantidad de marcos necesarios para todas las PTEs de la tabla de paginas de un proceso = $\frac{2^{52}}{2^{10}} = 2^{42}$
  - Tamaño tabla de paginas del proceso = $2^{42} \cdot 4bytes = +16000GB$ por proceso (es una banda)

1.8. Tabla de 2 niveles

- El propósito de la tabla de paginas multinivel es dividir la tabla de paginas lineal en múltiples tablas de paginas
- Cada tabla de paginas suele tener el mismo tamaño pero se busca que tengan un menor numero de paginas por tabla. Logrando que cada tabla sea mas pequeña. Logrando que no ocupe demasiada memoria RAM.
- Solo se carga una parcialidad de la tabla de paginas (solo lo que se necesite resolver)
- Existe un esquema de direccionamientos indirectos

1.9. Tabla invertida

- Utilizada en Arquitecturas donde el espacio de direcciones es muy grande
- Hay una entrada por cada marco de pagina en la memoria real. Es la visión invertida a la que veníamos viendo
- Hay una sola tabla para todo el sistema
- El espacio de direcciones de la tabla se refiere al espacio físico de la RAM, en vez del espacio de direcciones virtuales de un proceso
- Usada en PowerPC, UltraSPARC, y IA-64
- El numero de pagina es transformado en un valor de HASH
- El HASH se usa como índice de la tabla invertida para encontrar el marco asociado
- Se define un mecanismo de encadenamiento para solucionar colisiones (cuando el hash da igual para 2 direcciones virtuales)

Solo se mantienen los PTEs de paginas presentes en memoria física

- La tabla invertida es organizada como tabla hash en memoria principal
  - Se busca indexadamente por numero de pagina virtual
  - Si esta presente en tabla, se extrae el marco de pagina y sus protecciones
  - Si no esta presente en tabla, corresponde a un fallo de pagina

Cuadro 1: Tabla de Páginas lineal vs Tabla de Páginas Invertida

Tabla de Páginas lineal	**Tabla de Páginas Invertida**
Una tabla asociada a cada proceso	Una tabla en todo el sistema
Contiene una entrada por cada página que usa el proceso	Contiene una entrada para cada página real (frame) de memoria
Los procesos hacen referencia a las páginas por medio de direcciones virtuales	Se presenta al subsistema una parte de la dirección virtual
Se calcula a través del número de página y el desplazamiento	Se recorre la tabla para buscar una coincidencia
El S.O debe traducir esta referencia a una dirección de memoria física	Si la hay, se genera la dirección física; de lo contrario, hay un fallo de página

1.10. Tamaño de la Pagina

Pequeño

- Menor fragmentación interna
- Mas paginas requeridas por proceso: Tablas de paginas mas grandes
- Mas paginas pueden residir en memoria

Grande

- Mayor fragmentación interna
- La memoria secundaria esta diseñada para transferir grandes bloques de datos mas eficientemente: Mas rápido mover paginas hacia la memoria principal

1.11. Translation Lookaside Buffer

Cada referencia en el espacio virtual puede causar 2 o mas accesos a la memoria física. Uno o mas para obtener la entrada en la tabla de paginas, uno o mas para obtener los datos.
Para **solucionar este problema**, una memoria cache de alta velocidad es usada para almacenar entradas de paginas. Conocida como **TLB** (Translation Lookaside Buffer).

- El TLB contiene las entradas de la tabla de paginas que fueron usadas mas recientemente
- Dada una dirección virtual, el procesador examina la TLB
- Si la entrada de la tabla de paginas se encuentra en la TLB (hit), es obtenido el frame y armada la dirección física. Caso contrario (miss), el numero de paginas usado como índice en la tabla de paginas del proceso.
- Se controla si la pagina esta en la memoria (de no ser así se genera un PF)
- La TLB se actualiza en cada miss
- El cambio de contexto genera la invalidación de las entradas de la TLB

1.12. Asignación de Marcos

Tamaño del Conjunto Residente: Cantidad de paginas de un proceso que se pueden encontrar en memoria.

- **Asignación Dinámica:** El numero de marcos para cada proceso varia
- **Asignación Fija:** Numero fijo de marcos para cada proceso
  - Asignación equitativa: Si tengo 100 frames y 5 proceso, 20 frames para cada proceso.
  - Asignación proporcional: Se asigna acorde al tamaño del proceso.

$$\begin{aligned} s_i &= \text{size of process } p_i \\ S &= \sum s_i \text{ total size of process} \\ m &= \text{ total number of frames} \\ a_i &= \text{ allocation for } p_i = \frac{s_i}{S} \cdot m \end{aligned}$$

1.13. Reemplazo de paginas

Si ocurre un **PF** y todo s los marcos están ocupados, se debe seleccionar una **pagina víctima**. La mayoría de los reemplazos predicen el comportamiento futuro mirando el comportamiento pasado. **Alcance del Reemplazo**

- Reemplazo Global
  - El PF de un proceso puede reemplazar la pagina 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 el
  - Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad
  - Únicamente factible con **asignación dinámica**
- Reemplazo Local
  - El PF de un proceso solo puede reemplazar sus propias paginas, de su working set
  - No cambia la cantidad de frames asignados
  - El SO puede determinar la tasa de PF de cada proceso
  - Un proceso puede tener frames asignados que no usa y no pueden ser usados por otros procesos
  - Factible con **ambos tipos de asignación**

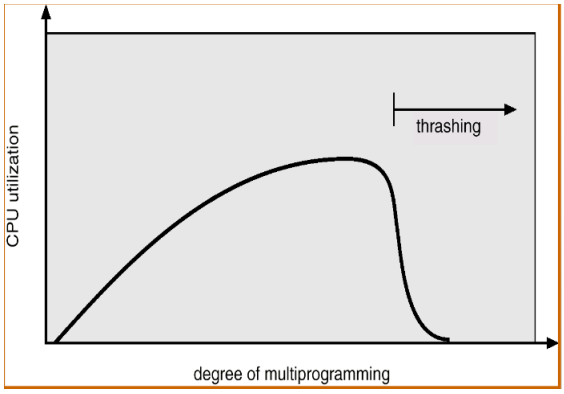
2. Memoria III

2.1. Thrashing (hiper paginación)

Concepto: Decimos que un sistema esta en thrashing cuando pasa mas tiempo paginando que ejecutando procesos. Como consecuencia, hay una baja importante de performance en el sistema.

2.2. Ciclo de thrashing

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



2.3. 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 PF y por lo tanto, mas actividad de paginado.
3. Se decrementa el uso de la CPU.
4. Continua en bucle.

2.4. Control del thrashing

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

2.5. Conclusión sobre thrashing

- Si un proceso cuenta con todos los frames que necesita, no habría thrashing
- Una manera de abordar esta problemática es utilizando la estrategia de Working Set (apoyada en el modelo de localidad)
- Otra estrategia es la del algoritmo PFF (Frecuencia de Fallos de Pagina)

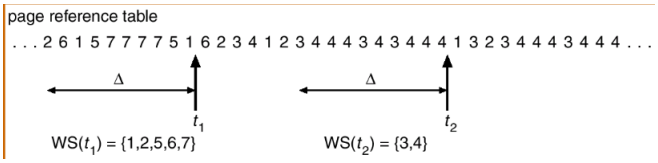
2.6. El modelo de localidad

Se basa en utilizar cercanía de referencias, es decir, agrupar las referencias a datos y programas dentro de un proceso. La localidad de un proceso en un momento dado se da por el conjunto de paginas que tiene en memoria en ese momento. En cortos periodos de tiempo. el proceso necesitara pocas piezas del proceso.

- Un programa se compone de varias localidades.
- Ejemplo: Cada rutina sera una nueva localidad: se referencian sus direcciones (cercanas) cuando se esta ejecutando.
- Para prevenir la hiper actividad, un proceso debe tener en memoria sus paginas mas activas (menos PF).

2.7. El modelo de working set

Se basa en el modelo de localidad. Utiliza una ventana que contiene las referencias de memoria mas recientes. El Δ indica de a cuantas paginas se referencian en memoria para un instante dado. Por ejemplo: $\Delta = 10$



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
- $WSS_i =$ tamaño del working set del proceso p_i
- $\sum WSS_i = D:$ demanda total de frames
- Si $D > m$ habrá thrashing.

2.8. Prevención del thrashing por WS

- SO monitorea cada proceso, dándole tantos frames hasta su WSS_i.
- Si quedan frames, puede iniciar otro proceso.
- Si D crece, excediendo m, se elige un proceso para suspender, reasignadose sus frames.

De esta forma, se mantiene el alto grado de multiprogramación optimizando el uso de la CPU y evitando el thrashing.

Problema del modelo del WS

- Mantener un registro de los WSS_i
- La ventana es móvil

2.9. Prevención del thrashing por PFF

La técnica **PFF (Page Fault Frecuency)**, en lugar de calcular el WS de los procesos, utiliza la tasa de PF para estimar si el proceso tiene un Conjunto Residente que representa adecuadamente al WS.

- PFF alta: Se necesitan mas frames.
- PFF baja: Los procesos tienen frames asignados que le sobran.

Método PFF:

- Se establecen limites superior e inferior de las PFF's deseadas.
- Si se excede la PFF max, se le asignan mas frames al procesos.
- Si la PFF esta por debajo del mínimo, se le quitan frames al proceso para utilizarlos en los que necesitan mas.
- Se puede llegar a suspender un proceso si no hay mas frames libres y todos los procesos están por arriba de PFF max.

2.10. Demonio de paginación

- En Linux: Proceso **"kswapd"**
- En Windows: Proceso **"system"**

Es un proceso creado 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.

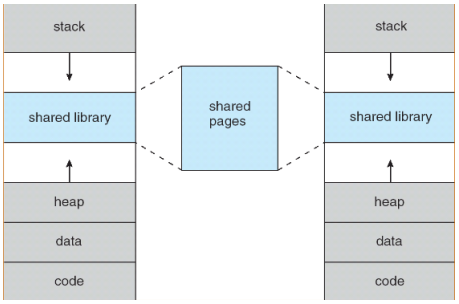
- Poca memoria libre.
- Mucha memoria modificada.

Tareas:

- Limpiar paginas modificadas sincronizandolas con el swap.
  1. Reduce el tiempo de swap posterior ya que las paginas están "limpias".
  2. Reduce el tiempo de transferencia al sincronizar varias paginas contiguas.
- Mantener un cierto numero de marcos libres en el sistema.
- Demorar la liberación de una pagina hasta que haga falta realmente.

2.11. Memoria Compartida

- Gracias al uso de la tabla de paginas varios procesos pueden compartir un marco de memoria; para ello ese marco debe estar asociado a una pagina en la tabla de paginas de cada proceso.
- El numero de pagina asociado al marco puede ser diferente en cada proceso.
- **Código compartido**
  - Los procesos comparten una copia de código (solo lectura).
  - Los datos son privados a cada proceso y se encuentran en paginas no compartidas.



2.12. Mapeo de Archivo en Memoria

Es una 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. El contenido de la pagina que genera el PF es obtenido desde el archivo asociado, no del área de intercambio.

- Cuando el proceso termina o el archivo se libera, las 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 o DLLs.

2.13. Copia en Escritura

La copia en escritura (Copy-on-Write, COW) permite a los procesos compartir inicialmente las misma paginas de memoria.

- Si uno de ellos modifica una pagina compartida la pagina es copiada.
- En fork, permite inicialmente que padre e hijo utilicen las misma paginas sin necesidad de duplicación.

COW permite crear procesos de forma mas eficiente debido a que solo las paginas modificadas son duplicadas.

2.14. Área de Intercambio

- Sobre el **Área utilizada**
  - 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 el disco de su área de intercambio. La lectura se realiza sumando el numero de pagina virtual a la dirección de comienzo del área asignada del proceso.
  2. **No se asigna nada inicialmente**. A cada pagina se le asigna su espacio en disco cuando se va a intercambiar, y el espacio se libera cuando la pagina vuelve a memoria. Problema: se debe llevar la contabilidad en memoria de la localización de las paginas en disco.
- Cuando una pagina no esta en memoria, sino en swap, podemos saber en que parte del área de intercambio se encuentra observando el PTE de dicha pagina. El mismo tendrá el bit V=0 y todos los demás bits sin usar.

2.15. Área de Intercambio - Linux

- Permite definir un numero predefinido de áreas de Swap.
- swap_info es un arreglo que contiene estas estructuras:

```
64 struct swap_info_struct {
65     unsigned int flags;
66     kdev_t swap_device;
67     spinlock_t sdev_lock;
68     struct dentry * swap_file;
69     struct vfsmount *swap_vfsmnt;
70     unsigned short * swap_map;
71     unsigned int lowest_bit;
72     unsigned int highest_bit;
73     unsigned int cluster_next;
74     unsigned int cluster_nr;
75     int prio;
76     int pages;
77     unsigned long max;
78     int next;
79 };
```

- Cada área es dividida en un numero fijo de slots según el tamaño de la pagina.
- Cuando una pagina es llevada a disco, Linux utiliza el PTE para almacenar 2 valores:
  1. El numero de área
  2. El desplazamiento en el área (24 bits, limitando el tamaño máximo del área a 64GB)

3. Mapa de Procesos Linux

3.1. Mapa de Memoria

- Un mapa de memoria es una estructura de datos que indica al SO como esta distribuida la memoria de un proceso, los segmentos que la componen, y los datos almacenados en cada uno de ellos.
- El mapa de memoria de un proceso se divide generalmente en seis segmentos:
  1. Argumentos de la linea de comandos y variables de entorno.
  2. Stack o pila del proceso.
  3. Heap o espacio para almacenar segmentos de memoria dinámica.
  4. Datos no inicializados (BSS)
  5. Datos inicializados.
  6. Segmento de texto (código binario del programa).
- Linux utiliza gestión de memoria basada en segmentación y paginación.
- Las paginas de memoria son de tamaño fijo, por ejemplo, 4KiB. Mientras que la segmentación implica segmentos de tamaño variable.
- Linux divide el mapa de memoria de un proceso en segmentos de diferentes tamaños, cada uno dividido, a su vez, en paginas de tamaño fijo (segmentación paginada).

4. Entrada/Salida

Respecto a los dispositivos de E/S, el SO tiene la responsabilidad de generar comandos para manejarlos, atender las interrupciones que generan y manejar errores que podrían presentarse. A su vez, debe proporcionar una interfaz que haga posible su uso.

Problemas:

- Heterogeneidad de dispositivos.
- Características de los dispositivos.
- Velocidad.
- Nuevos tipos de dispositivos.
- Diferentes formas de realizar E/S.

4.1. Aspectos de los dispositivos de E/S

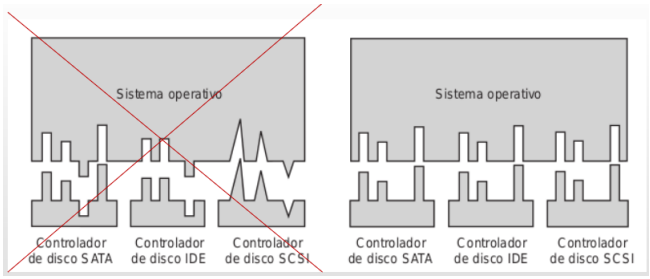
1. Unidad de Transferencia
   - Dispositivos por bloques (discos): Operaciones Read, Write, Seek.
   - Dispositivos por Caracter (keyboards, mouse, serial ports): Operaciones: get, put.
2. Formas de Acceso: Secuencial o Aleatorio.
3. Tipo de acceso
   - Acceso Compartido: Disco Rígido.
   - Acceso Exclusivo: Impresora.

- Read only: CDROM.
- Write only: Pantalla.
- Read/Write: Disco.

4. Velocidad: Muy variable entre dispositivos.

4.2. Metas, Objetivos y Servicios

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



3. Eficiencia:
   - Los dispositivos de I/O pueden resultar extremadamente lentos respecto a la memoria y la CPU.
   - El uso de la multi-programación permite que un proceso espere por la finalización de su I/O mientras que otro proceso se ejecuta.
4. Planificación: Organización de los requerimientos a los dispositivos.
5. Buffering: Almacenamiento de los datos en memoria mientras se transfieren.
   - Soluciona problemas de velocidad entre los dispositivos.
   - Soluciona problema de tamaño y/o forma de los datos entre dispositivos.
6. Caching: Mantener en memoria copia de los datos de reciente acceso para mejorar la performance.
7. Spooling: Administrar la cola de requerimientos de un dispositivo.
   - Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo, por lo tanto debe coordinarse el acceso concurrente al dispositivo.
8. Reserva de Dispositivos: Acceso exclusivo.
9. Manejo de errores:
   - El SO debe administrar errores ocurridos.
   - La mayoría retorna un numero de error o código cuando la I/O falla.

4.3. Formas de realizar I/O

- Bloqueante: El proceso se suspende hasta que el requerimiento de la I/O se completa. Esto es fácil de usar y entender, pero no es suficiente bajo algunas necesidades.
- No bloqueante: El requerimiento de I/O retorna en cuanto es posible. Por ejemplo, una interfaz de usuario que recibe un input de teclado/mouse.

4.4. Diseño - SW capa de usuario

Modo usuario

1. Librerías de funciones:
   - Permiten acceso a SysCalls.
   - Implementan servicios que no dependen del kernel.
2. Procesos de apoyo
   - Demonio de impresión (spooling)

4.5. Diseño - SW del SO independiente del dispositivo

Modo kernel

1. Brinda los principales servicios de E/S antes vistos:
   - Interfaz uniforme
   - Manejo de errores
   - Buffer
   - Asignación de recursos
   - Planificación
2. El kernel mantiene la información de estado de cada dispositivo o componente (archivos abiertos, conexiones de red, etc.)
3. Hay varias estructuras complejas que representan bufares, utilización de memoria, disco, etc.

4.6. Diseño - Controladores

Modo kernel

1. Interfaz entre el SO y el HW.
2. Forman parte del espacio de memoria del kernel (suelen cargarse como módulos).
3. Los fabricantes de HW implementan el driver en función de una API especificada por el SO.
4. Para agregar nuevo HW, no hace falta realizar cambios en el Kernel, solamente indicar el driver correspondiente.
5. Contienen el código dependiente del dispositivo.
6. Manejan un tipo de dispositivo.
7. traducen los requerimientos abstractos en los comandos para el dispositivo:
   - Escribe sobre los registros del controlador.
   - Acceso a la memoria mapeada.
   - Encola requerimientos.
8. Comúnmente las interrupciones generada por los dispositivos son atendidas por funciones provistas por el driver.

4.7. Diseño - Gestor de interrupciones

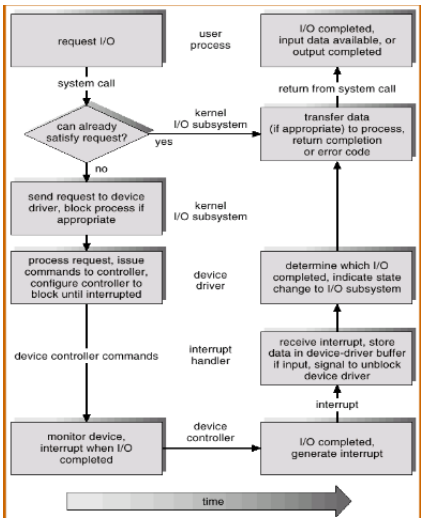
Modo kernel

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

4.8. Ciclo de atención de un requerimiento.

Considerando como ejemplo la lectura sobre un archivo en un disco:

1. Determinar el dispositivo que almacena los datos. Traducir el nombre del archivo en la representación del dispositivo.
2. Traducir el requerimiento abstracto en bloques de disco según el File System.
3. Realizar la lectura física de los datos (bloques) en la memoria.
4. Marcar los datos como disponibles al proceso que realizo el requerimiento para luego desbloquearlo.
5. Retornar el control al proceso.



4.9. Performance

La **I/O** es uno de los factores que mas 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 la copia de datos.

Mejorar la Performance

- Reducir el numero 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:
  1. Transferencias de gran cantidad de datos.
  2. Controladores mas inteligentes.
  3. Polling, si se minimiza la espera activa.
- Utilizar DMA.

5. Arquitectura de Entrada/Salida - Anexo I

5.1. Comunicación: CPU - Controladora

La CPU puede ejecutar comandos o enviar/recibir datos de una controladora de un dispositivo gracias a los registros varios de la misma. Algunos son dedicados a señales de control (busy, ready, etc) y otros a datos. La CPU se comunica con la controladora escribiendo y leyendo en dichos registros.

Comandos de I/O

- La CPU emite direcciones para identificar el dispositivo.
- La CPU emite comandos de control (girar el disco), test(power, error) y dirección de transferencia (read/write).

5.2. Mapeo de E/S y E/S aislada

- Correspondencia en memoria (Memory mapped I/O)
  1. Dispositivos y memoria comparten el espacio de direcciones.
  2. Las transferencias I/O simulan leer/escribir en memoria.
  3. No hay instrucciones espaciales para I/O.
- Isolated I/O
  1. Espacio separado de direcciones.
  2. Se necesitan lineas de I/O. Puertos de E/S.
  3. Se requieren instrucciones especiales.

5.3. Técnicas de I/O

- Programada
  1. La CPU tiene control directo sobre la I/O. Controla el estado, los comandos para leer y escribir y transfiere los datos.
  2. La CPU espera que el componente de I/O complete la operación. Esto hace que se desperdicien ciclos de la CPU.

  En la I/O programada, es necesario hacer **polling** del dispositivo para determinar el estado del mismo (ready, busy, error).
- Manejada por interrupciones
  1. Soluciona el problema de la espera de la CPU.
  2. La CPU no repite el chequeo sobre el dispositivo.
  3. El procesador continua la ejecución de instrucciones.
  4. El componente de I/O envía una interrupción cuando finaliza.
- DMA
  1. Un componente de DMA controla el intercambio de datos entre la memoria principal y el dispositivo.
  2. El procesador es interrumpido luego de que el bloque entero fue transferido.

6. Administración de Archivos - I

6.1. Introducción

Archivo:

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

Punto de vista del usuario:

- Que operaciones se pueden llevar a cabo.
- Como nombrar a un archivo.
- Como asegurar la protección.
- Como compartir archivos.
- 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.

6.2. Sistema de manejo de archivos

Es el conjunto de unidades de SW que proveen los servicios necesarios para la utilización de archivos (crear, borrar, buscar, copiar, leer, escribir, etc.

- Facilita el acceso a los archivos por parte de las aplicaciones.
- Permite la abstracción al programados, en cuanto al acceso de bajo nivel.

Objetivos:

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

6.3. Tipos de Archivos

- Archivos regulares:
  1. Texto plano (src file)
  2. Binarios (Object file, Executable file
- Directorios: Archivos que mantienen la estructura en el FS.

6.4. Atributos de un Archivo

- Nombre
- Identificador
- Tipo
- Localización
- Tamaño
- Protección, seguridad y monitoreo:
  1. Owner, permisos, pswd
  2. Momento en que el usuario lo modifico, creo, accedió por ultima vez.
  3. ACLs (????)

6.5. Directorios

El directorio es, en si mismo, un **archivo** que contiene información acerca de los archivos y directorios que están dentro de el.

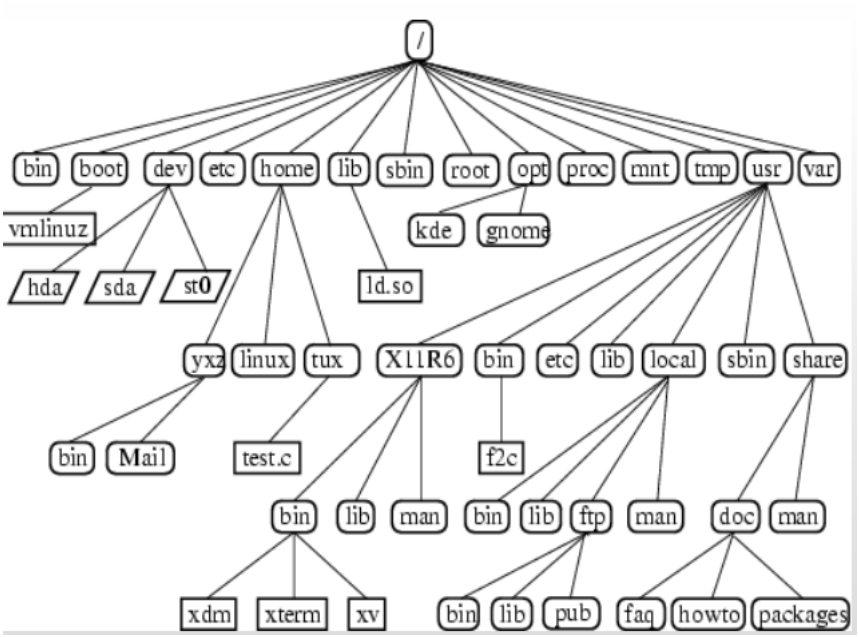
Operaciones en directorios:

- Buscar un archivo
- Crear un archivo
- Borrar un archivo
- Listar el contenido
- Renombrar archivos
- Entre otras

El uso de los directorios ayuda con:

- La eficiencia: Localización a de archivos.
- Uso del mismo nombre de archivo: Diferentes usuarios pueden tener el mismo nombre de archivo.
- Agrupación lógica de archivos por propiedades/funciones.

6.6. 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 fullpathname es único.
- Al directorio actual se lo llama working directory.
- Dentro del working directory, se pueden referenciar los archivos tanto por su PATH absoluto como por su PATH relativo indicando solamente la ruta al archivo desde el directorio actual.

6.7. Protección y derechos de acceso

En un ambiente multiusuario se necesita que varios usuarios puedan compartir archivos. Esto debe ser realizado bajo un esquema de protección, el cual utiliza derechos de acceso y un manejo de accesos simultáneos.

Protección: El propietario/administrador debe ser capaz de controlar que se puede hacer (derechos de acceso) y quien lo debe hacer.

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

- 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 existente del archivo.
- Updating: El usuario puede modificar, borrar y agregar datos. Incluye la creación de archivos, sobrescribirlo y remover datos.
- Changing protection: El usuario puede modificar los derechos de acceso.
- Deletion: El usuario puede borrar el archivo.
- **Owners:**
  1. Tienen todos los derechos.
  2. Pueden dar derechos a otros usuarios. Se determinan clases (usuario especifico, grupos de usuarios, todos).

7. Administración de Archivos - II

7.1. Introducción

El sistema de archivos tiene como **metas principales** brindar espacio en disco a los archivos de usuario y del sistema y, además, mantener un registro del espacio libre, tanto la cantidad como la ubicación dentro del disco.

Conceptos:

- Sector: Unidad de almacenamiento utilizada en los discos rígidos.
- Bloque/Cluster: Conjunto de sectores consecutivos.
- File System: Define la forma en que los datos son almacenados.
- FAT (File Allocation Table): Contiene información sobre en que lugar están alocados los distintos archivos.

7.2. Sobre la asignación

Existen dos criterios para asignar espacio en disco.

Pre-asignación:

- Se necesita saber cuanto espacio va a ocupar el archivo en el momento de su creación.
- Se tiene a definir espacios mucho mas grandes que lo necesario.
- Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo.
- Problemas si el archivo supera el espacio asignado.

Asignación dinámica:

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

7.3. Formas de asignación

Asignación continua:

- Se utiliza un conjunto continuo de bloques.
- Se requiere una pre-asignación.
- La FAT (File Allocation Table) es simple, solo requiere una entrada que incluye el bloque de inicio y la longitud.
- El archivo puede ser leído con una única operación.
- Puede existir fragmentación externa, la misma se trata con **compactación**.
- **Problemas de la técnica:**
    1. Encontrar bloques libres continuos en el disco.
    2. Incremento del tamaño de un archivo.

Asignación encadenada:

- Asignación en base a bloques individuales.
- Cada bloque tiene un puntero al próximo bloque del archivo.
- FAT - Única entrada por archivo: Bloque de inicio y tamaño del archivo (luego se recorre con punteros).
- No hay fragmentación externa.
- Útil para acceso secuencial (no random).
- Los archivos pueden crecer bajo demanda.
- No se requieren bloques contiguos.
- Se pueden consolidar los bloques de un mismo archivo para garantizar la cercanía de los mismos.

Asignación indexada:

- La FAT contiene un puntero al bloque índice.
- El bloque índice no contiene datos propios del archivo, sino que contiene un índice a los bloques que lo componen.
- Es una asignación en base a bloques individuales.
- No se produce fragmentación externa.
- El acceso random a un archivo es eficiente.
- Variante - asignación por **secciones**:
    1. A cada entrada del bloque índice se agrega el campo longitud.
    2. El índice apunta al primer bloque de un conjunto almacenado de manera contigua.
- Variante - **niveles de indirección (i-nodos)**:
    1. Existen bloques directos de datos.
    2. Otros bloques son considerados como bloques índice (apuntan a varios bloques de datos).
    3. Pueden haber varios niveles de indirección.

7.4. Gestión de espacio libre

Resulta útil tener un control sobre cuales son los bloques de disco que están disponibles. Para esto, existen tres alternativas:

- Tablas de bits.
- Bloques libres encadenados.
- Indexación.

Tabla de bits:

- Se utiliza una tabla (vector) con 1 bit por cada bloque de disco.
- Cada entrada contiene 0 si el bloque esta libre o 1 si esta en uso.
- Ventaja: Fácil encontrar un bloque o grupo de bloques libres.
- Desventaja: Tamaño de vector en memoria considerable.
  tamaño vector $= \frac{\text{tamaño disco}}{\text{tamaño bloque en sistema archivo}}$

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 ya que hay que realizar varias operaciones de E/S para obtener un grupo libre.
- Problemas con la perdida de un enlace.
- Difícil encontrar bloques libres consecutivos.

Indexación:

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

8. Administración de Archivos - III

8.1. UNIX - Manejo de archivos

Tipos de archivos:

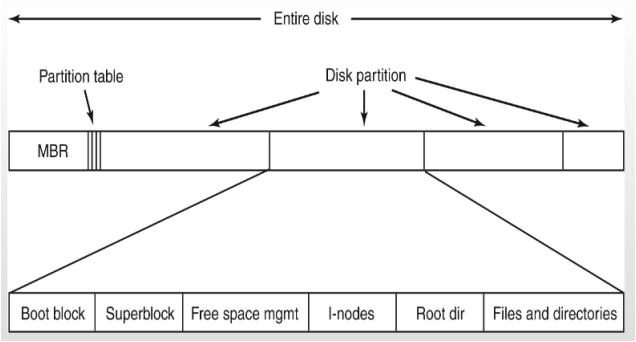
- Archivo común
- Directorio
- Archivos especiales (dispositivos /dev/sda)
- Named pipes (comunicación entre procesos)
- Links (comparten el i-nodo, solo dentro del mismo FS)
- Links simbólicos (tiene i-nodo propio, para FS diferentes)

8.2. UNIX - Estructura del volumen

Cada disco físico puede ser dividido en uno o mas 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 FS (Bloques/Clusters libres)
- I-NODO Table: Tabla que contiene todos los I-NODOS
- Data blocks: Bloques de datos de los archivos.

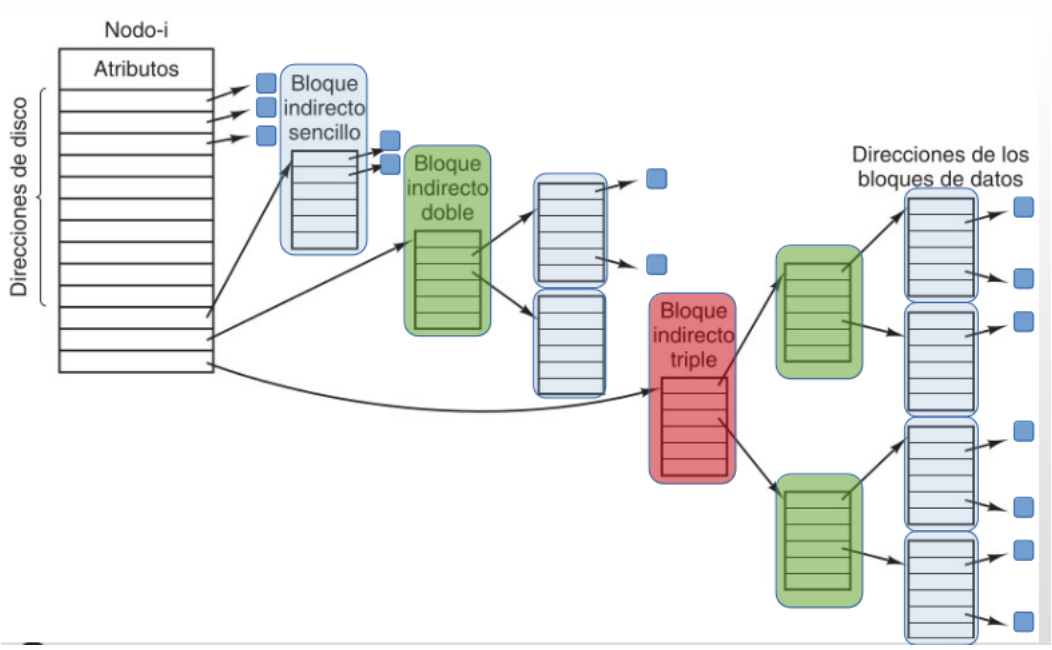
Visto gráficamente:



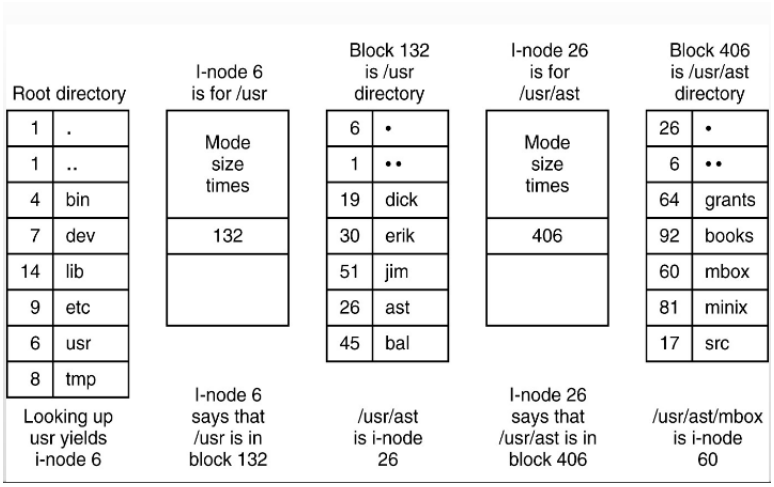
8.3. UNIX - I NODO

La siguiente tabla muestra la información que almacena un i-nodo:

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



Ejemplo de búsqueda de i-nodo: Como se encuentra el i-nodo del archivo /usr/ast/mbox:



8.4. Windows - File systems soportados

- CD-ROM File System (CDFS): CD.
- Universal Disk Format (UDF): DVD, Blu-Ray.
- File Allocation Table (FAT):
  1. FAT12: MS-DOS v3.3 a 4.0.
  2. FAT16: MS-DOS 6.22, nombres cortos de archivo.
  3. FAT32: MS-DOS 7.10, nombres largos pero no soportados en MS-DOS.
- New Technology File System (NTFS)

8.5. Windows - FAT

FAT es un sistema de archivos utilizado originalmente por DOS y Windows 9x.
Windows aun utiliza FAT ya que el mismo tiene compatibilidad con otro SO en sistemas multiboot. Además, permite upgrades desde versiones anteriores y sirve como formato de dispositivos como diskettes.
Las distintas versiones de FAT se diferencian por la cantidad de bits que se usan para identificar diferentes bloques o clusters (12, 16, 32).
Funcionamiento:

- Se utiliza un mapa de bloques del sistema de archivos, llamado FAT.
- 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.
- Los bloques libres y dañados tienen códigos especiales.

FAT12: En sistemas FAT12, al utilizarse 12 bits para la identificación del sector, la misma se limita a $2^{12} = 4096$ sectores.

- Windows utiliza tamaños de sector desde los 512 bytes hasta 8 KB, lo que limita a un tamaño total de volume de 32 MB: $2^{12} \cdot 8KB$
- Windows utiliza FAT12 como sistema de archivos de los disketts de 3,5 y 12 pulgadas que pueden almacenar hasta 1,44 MB de datos.

FAT16: Al utilizar 16 bits para identificar cada sector puede tener hasta 2^{16} sectores en un volumen.

- En windows el tamaño de sector en FAT16 varia desde los 512 bytes hasta los 64 KB, lo que limita a un tamaño máximo de volume de 4 GB.

FAT32: Fue el FS mas reciente de la linea. Utiliza 32 bits para la identificación de sectores, pero reserva los 4 bits superiores, con lo cual efectivamente solo se utilizan 28 bits para la identificación.

- El tamaño de sector en FAT 32 puede ser de hasta 32 KB, con lo cual tiene una capacidad teórica de direccionar volúmenes de hasta 8 TB
- El modo de identificación y acceso de los sectores lo hace mas eficiente que FAT16. Con tamaño de sector de 512 bytes, puede direccionar volúmenes de hasta 128 GB.

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4MB	50 MB	
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8KB		512 MB	2 TB
16KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

8.6. Windows - NTFS

- NTFS es el filesystem nativo de Windows desde Windows NT
- NTFS usa 64-bit para referenciar sectores (Teóricamente permite tener volúmenes de hasta 16 exabytes).

Ventajas de NTFS contra FAT:

- Soporta tamaños de archivo y de discos mucho mayores.
- Mejora la performance en discos grandes.
- Soporta nombres de archivos de hasta 255 caracteres.
- Atributos de seguridad.
- Transaccional.

9. Buffer Cache

9.1. 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:
  1. Header: contiene información del bloque, numero del bloque, estado, relación con otros buffers, etc.
  2. El buffer en si: lugar donde se almacena el bloque de disco traído a memoria.

9.2. Buffer Cache en el Kernel

- El modulo de buffer cache es independiente del sistema de archivos y de los dispositivos de HW.
- Es un servicio del SO.

9.3. Estados de los buffers

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

9.4. 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 sigue en estado DW).
- Se ordena según algoritmo LRU.
- Sigue el mismo esquema de la hash queue pero contiene los headers de los buffers de aquellos proceso que ya han terminado.
- El header de un buffer siempre esta en la hash queue.
- Si el proceso que lo referenciaba termino, va a estar en la hash queue y en la free list.

9.5. 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, numero de bloque).
- Al numero de bloque se le aplica una función de hash que permite agrupar los buffers cuyo resultado dio igual para hacer que las búsquedas sean mas eficientes.
- Se busca que la función de hash provea alta dispersión para lograr que las colas de bloques no sean tan extensas.
- Para agrupar lo bloques se utilizan los punteros almacenados en el header.

9.6. Funcionamiento del buffer cache

- Cuando un proceso quiere acceder a un archivo, utiliza su inodo para localizar los bloques de datos donde se encuentra este.
- El requerimiento llega al buffer cache quien evalúa si puede satisfacer el mismo o si debe realiza la E/S.
- Se pueden dar 5 escenarios:
    1. El kernel encuentra el bloque en la hash queue y el buffer esta libre (en la free list).
    2. El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
    3. Ídem 2, pero el bloque libre esta marcado como DW.
    4. El kernel no encuentra el bloque en la hash queue y la free list esta vacía.
    5. El kernel encuentra el bloque en la hash queue pero esta BUSY.

En las diapositivas hay **ejemplos gráficos de cada escenario**.