Tema 3 - Memoria

Al fondo están las preguntas de repaso.
Lo marcado en gris es lo que no va, pero sirve de repaso…

Los programas y datos deben estar en el almacenamiento principal para poderlos ejecutar y que sean referenciados directamente.

Por otro lado, el SO debe ser eficiente en el uso de memoria, alojando un gran número de procesos. Además debe llevar un registro de las partes de la memoria que se usen y de las que no, asignar espacio a los procesos que lo pidan y liberarlo cuando ya no lo requieran, brindar seguridad entre los procesos para que no pisen entre ellos, brindar posibilidad de acceso compartido, y lo más importante: que el programador se abstraiga de donde quedan ubicados los programas.

Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones

En cuanto a la seguridad, el chequeo de permisos para acceder a posiciones de memoria debe hacerse durante la ejecución, esto es asi debido a que no es posible anticipar todas las referencias a memoria que un proceso puede realizar.

Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos. El tamaño depende de la Arquitectura del Procesador de:
32 bits: 0 .. 2^{32} - 1
64 bits: 0 .. 2^{64} - 1

Dirección lógica: referencia a una localidad de memoria en el “Espacio de direcciones del proceso”
Dirección fisica: es la referencia a un localidad en memoria fisica (ram).

Para pasar de una a otra se necesita algún tipo de conversión (realizada por hardware, la MMU), ya que si, por ejemplo, el cpu trabaja con direcciones lógicas, para ir a la memoria se debe transformar a fisica.

Una forma facil es usando un registro base (dirección del comienzo del espacio de direcciones en ram) y un registro límite (el final del espacio de direcciones en ram). Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria.
Problemas de esto: es necesario almacenar el espacio de direcciones de forma continua en RAM. Solución? paginación.

Un proceso nunca va a trabajar con direcciones fisicas.

MMU: dispositivo de hardware ubicado en el procesador que mapea direcciones virtuales a fisicas. Para reprogramar el mmu solo se puede hacer en kernel mode.

Mecanismos de asignación de memoria

Particiones Fijas:
La memoria se divide en particiones o regiones de tamaño Fijo (pueden ser todas del mismo tamaño o no). Alojan un proceso cada una. Cada proceso se coloca de acuerdo a algún criterio (First Fit, Best Fit, Worst Fit, Next Fit) en alguna partición.

Particiones dinámicas:
Las particiones varían en tamaño y en número. Alojan un proceso cada una . Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso.

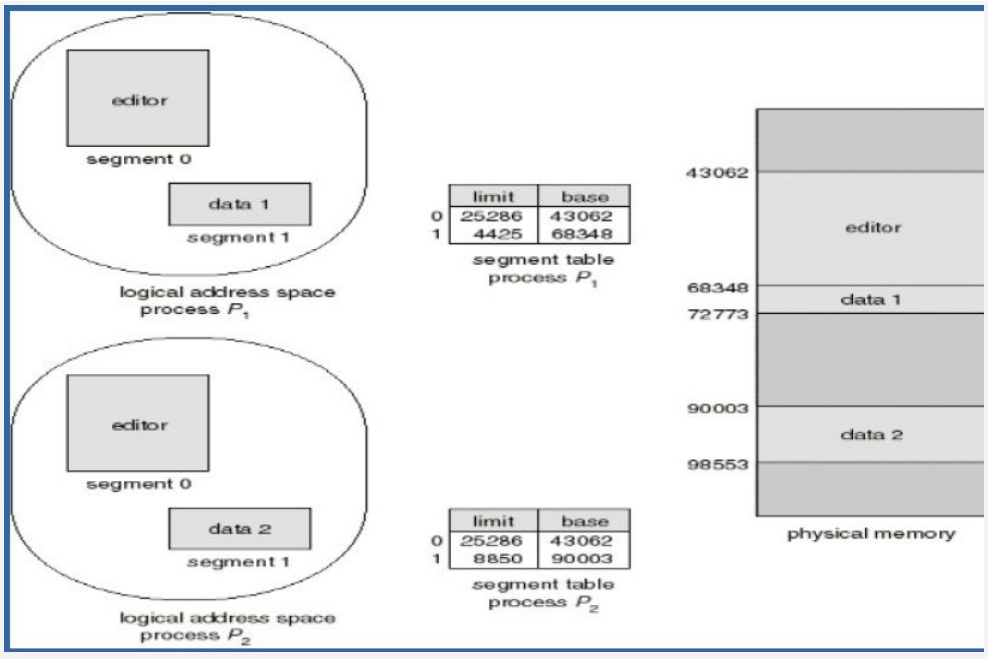
Fragmentación interna: solo en particiones fijas. Porción de partición sin uso.
Fragmentación externa: solo en particiones dinámicas. Huecos que quedan en memoria a medida que finalizan procesos y no pueden ser reutilizados para alocar otro proceso. Se puede compactar pero es costoso.

Paginación

Marcos: memoria física dividida lógicamente en pequeños trozos de igual tamaño .

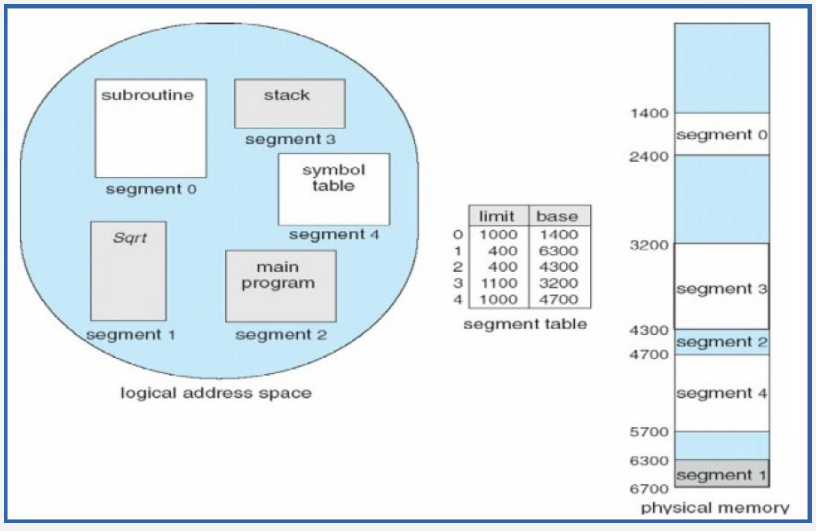
Páginas: memoria Lógica (espacio de direcciones) dividida en trozos de igual tamaño que los marcos.
El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada contiene (entre otras) el Marco en la que se coloca cada página.
La dirección lógica se interpreta como un número de página y un desplazamiento dentro de la misma.

La paginación nos permite proteger nuestros procesos y a su vez compartirlos:



Segmentación

Esquema que se asemeja a la “visión del usuario”. El programa se divide en partes/ secciones
Un programa es una colección de segmentos. Un segmento es una unidad lógica como: Programa Principal, Procedimientos y Funciones, variables locales y globales, stack, etc. Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas). Puede causar Fragmentación.



Las direcciones Lógicas consisten en 2 partes: selector de Segmento y desplazamiento dentro del segmento.

Tabla de Segmentos: permite mapear la dirección lógica en física. Cada entrada contiene:

Base: Dirección física de comienzo del segmento.
Limit: Longitud del Segmento.

Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos.
Segment-table length register (STLR): cantidad de segmentos de un programa

Segmentación paginada

La paginación es transparente al programador. Elimina Fragmentación externa.
La segmentación, por otro lado, es visible al programador. Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección
Segmentación Paginada: Cada segmento es dividido en páginas de tamaño fijo.

No todo el espacio de direcciones del proceso se necesita en todo momento (algunas librerías o rutinas por ejemplo). Por eso se puede traer a memoria las “piezas” que se van necesitando. A este se llama **“Conjunto Residente”** o working set. Con esta tecnología, se pueden mantener cargado los procesos de manera más eficiente.
Para el funcionamiento de esto se necesita un HW que soporte paginación (o segmentación) por demanda, un disco de memoria secundaria para swap y un SO que organice.

Conjunto residente: Porciones del espacio de direcciones de un proceso que están en RAM. Con el HW se detecta si una porción del proceso no está en su conjunto residente, por lo que se debe cargar dicha porción.
La ventaja es que, como se cargan porciones, en RAM va a haber más multiprogramación.

Para el manejo de Memoria Virtual, el HW debe dar apoyo para almacenar en SWAP las secciones del proceso que no están en RAM y poder manejar el movimiento de páginas/segments de SWAP a RAM y viceversa.

Cada proceso tiene su tabla de páginas, en la que cada entrada de la tabla se referencia al frame donde se encuentra la página. Para esto se cuenta con bits de control:
Bit V(validez):Indica si está en memoria la página. 1 si esta 0 si no esta.
Bit M (modificado):Indica si la página se modificó. Si se modificó, los cambios se deben reflejar en memoria secundaria.

***Page Faul*t**: Se genera cuando se pide una página que no está en su conjunto residente (Bit V en 0). El HW detecta esto y genera un TRAP(excepción), a lo que el SO pone el proceso en Bloqueado(espera) mientras se gestiona la carga de la página correspondiente.
El SO busca un Frame Libre, genera una operación de E/S al disco para copiar en ese frame la página que se necesita. Mientras esa E/S se completa, se asigna a la CPU otro proceso.
Cuando la E/S termina, envía una interrupción, actualiza la tabla de páginas poniendo el bit V en 1, y la dirección Base del frame donde se encuentra la página en RAM. El proceso se encola en la cola de Ready to Run. Cuando vuelva a ejecutarse, arranca de donde quedó.

Con respecto a la performance, (por la constante ejecución de operaciones de E/S).

La tasa de PF se evalúa entre 0 y 1. 0 <= p <= 1. Si p = 0 no hay PF. Si p = 1 hubo PF en cada solicitud de página.
El tiempo de acceso efectivo (EAT) se evalúa como

(1-p) x memory_access + p x (page_fault_overhead + swap_page_out + swap_page_in + restart_overhead).

Formas de organizar la Tabla de Páginas:

Nivel 1: Tabla única
Nivel 2: Tabla Multinivel
Tabla invertida: Hashing.

La forma de organizarla va a depender del HW

Con respecto a la tabla de 1 nivel, cuanto más chico es el tamaño de direcciones, más chica es la tabla de páginas. Ejemplo diapo 18
Cuanto más grande es el tamaño de una dirección, más grande es el tamaño de la tabla de páginas. Ejemplo diapo 19

Tabla de página Multinivel: Busca dividir la tabla de páginas en múltiples tablas, soliendo ser cada una del mismo tamaño y que almacenen una menor cantidad de páginas por tabla. Se usa cuando el proceso es muy grande, por ende, la tabla de páginas queda grande también

La desventaja es que, si hay muchos niveles de página, se baja la performance para ubicar el frame de una página.

Tabla de página Invertida: De la dirección virtual (PID,Num pagina virtual y offset), con el PID y NPV aplicando una función de HASH se consigue la entrada en la Hash Table donde se encuentra el número de página Física, que sumado al offset se accede a la física y en el #Frame va a estar ese número de página física.

Sin embargo, sigue el problema que de que se necesitan 2 o más accesos para obtener el frame. Por tanto, se usa la TLB, una memoria caché que se basa en el principio de localidad, donde están las entradas de la tabla de página que fueron usadas recientemente.
Entonces, cuando se da una dirección virtual, se examina la **TLB**. Si la entrada de la tabla de página derivada de la dirección virtual está en la TLB (un hit), se obtiene el frame y se arma la dirección física.
Si no es encontrada (miss), el número de página se usa como índice en la tabla de páginas. Se controla si la página está en memoria, sino PF.
Se actualiza la TLB para incluir la entrada de una nueva página (si se genera PF, se reserva un nuevo frame, se descarga de swap la página en cuestión y se copia en el frame, y por tanto, actualizar la tlb con la nueva página y también actualiza la tabla de páginas con la nueva página en cuestión).
El cambio de contexto genera la invalidación de las entradas de la TLB.

Thrashing (hiperpaginación): Decimos que un sistema está en thrashing cuando pasa más tiempo paginando que ejecutando procesos (Esto se da porque hay mucha multiprogramación en RAM y, por ende, más tendencia a fallos de página. Por ende se pierde performance por atender cantidad de PF).
Como consecuencia, hay una baja importante de performance en el sistema.

Ciclo de Thrashing:
El kernel monitorea el uso de CPU. Si hay baja utilización se incrementa la multiprogramación (el long term empieza a elegir procesos para meter). Con el aumento de la multiprogramación, un proceso empieza a necesitar más frames. Pero como empieza a haber menos frames porque son usados por otros procesos, empiezan los PF. Si hay PF, hay que seleccionar una víctima. Si el algoritmo de reemplazo es global, se le saca un frame a otro proceso (en funcion del algoritmo Sea FIFO,LRU, etc).
Como hay tanto PF, y por ende mucho acceso a disco (swappeando páginas de disco a ram), baja el uso de CPU.

Para controlar el thrashing, se puede limitar usando reemplazo local, ya que no se roban frames de otros procesos, sino del mismo proceso.

Otra manera de abordar es con la estrategia de Working Set apoyada en el modelo de localidad (las referencias de datos dentro de un proceso tienden a agruparse).
Ventana working set (delta): las referencias de memoria más recientes.
Delta del Working Set: Si es muy chico, no se cubre la localidad (la localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento). Si es muy grande, se pueden tomar varias localidades

Prevención de thrashing por PFF: En lugar de calcular el WorkingSet de los procesos, usa la tasa de fallos de página para estimar si el proceso tiene un WS que representa adecuadamente al WS:
PFF: frecuencia de fallo de página
Si es muy grande: Se necesitan más frames
Si es muy chica: Frames asignados de más
Establecer tasa de PF aceptable
Si la tasa actual es baja, el proceso pierde frames
Si la tasa actual es alta, el proceso gana frames

Establecer límites de las PFF deseadas: Si se excede del máximo, le doy un frame más. SI está por debajo, le saco frames. Si no hay más frames, se suspende el proceso. SUs frames se reasginan a procesos de alta PFF

Demonio de paginación linux: Proceso creado por el SO durante el arranque que apoya a la administración de la memoria. Se ejecuta cuando el sistema tiene una baja utilización o algún parámetro de la memoria lo indica (poca memoria libre o mucha memoria modificada).
Tareas que hace:

- 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 páginas libres en el sistema.
- Demorar la liberación de una página hasta que haga falta realmente

Memoria Compartida: Por el uso de la tabla de páginas, los procesos pueden compartir un marco de memoria, por lo que para ellos ese marco debe estar asociado a una página en la tabla de páginas de cada proceso. El num 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 Archivo en Memoria: 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 genera un PF. EL contenido de la página que genera el pf es obtenido desde el archivo asociado, NO DE SWAP
Cuando el proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo correspondiente
Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el Sistema de Archivos
Es utilizado comúnmente para asociar librerías compartidas o DLLs

Tema 4 - E/S

Generalidades: se desea poder manejar los dispositivos de E/S de manera uniforme y estandarizada, pudiendo ocultar los detalles en rutinas de bajo nivel para que los procesos vean dispositivos en términos de operaciones comunes como read,write,close, etc. (Básicamente generar una capa de abstracción)

Eficacia: Los dispositivos de E/S puede ser más lentos que la CPU o la RAM. La multiprogramación permite que un proceso espere porque termine una E/S mientras otro proceso se ejecuta

Aspectos de los dispositivos:

- Unidad de Transferencia: Hay distintos tipos de transferencia
    - Dispositivos que transfieren por bloques como el disco con operaciones como read,write, seek
    - Dispositivos que transfieren carácter por carácter como el mouse, teclado
- Formas de Acceso:
    - Secuencial o Aleatorio

- Tipo de Acceso:
    - Compartido (disco rígido), Exclusivo (Impresora)
    - Read Only: CDROM

- Write Only: Pantalla
- Read/Write: Disco

Servicios

- Planificación: Se basa en cómo organizar los requerimientos que llegan a los dispositivos
    - **Buffering**: Almacena los datos en memoria mientras se transfieren (usan buffers en RAM). Soluciona el problema de velocidad entre dispositivos y problemas de tamaño y forma de los datos entre los dispositivos
    - **Caching**: Mantener en memoria una copia de los datos de reciente acceso para mejorar la performance.
    - **Spooling**: Administrar la cola de requerimientos de los dispositivos. Algunos dispositivos de acceso exclusivo (como la impresora) no pueden atender distintos requerimientos al mismo tiempo. El spooling permite el acceso concurrente al dispositivo.
    - **Reserva de Dispositivos** (se corresponde con acceso exclusivo)
    - **Manejo de Errores**: El SO debe administrar los errores que ocurran como lecturas de disco o errores de escritura, retornando número de error o un código cuando la E/S falla. También lleva un log de errores (el log es una bitácora con los errores)
    - Formas de Realizar la E/S:
        - Bloqueante: El proceso se suspende hasta que el requerimiento de E/S se completa
            - Fácil de usar y entender
            - No es suficiente bajo algunas necesidades
        - No bloqueante: El requerimiento de E/S retorna en cuanto es posible
            - **Ej:** Interfaz de usuario que recibe input desde el teclado/mouse y se muestra en pantalla.
            - **Ej**: Aplicación de video que lee frames desde un archivo mientras va mostrándolo en pantalla.

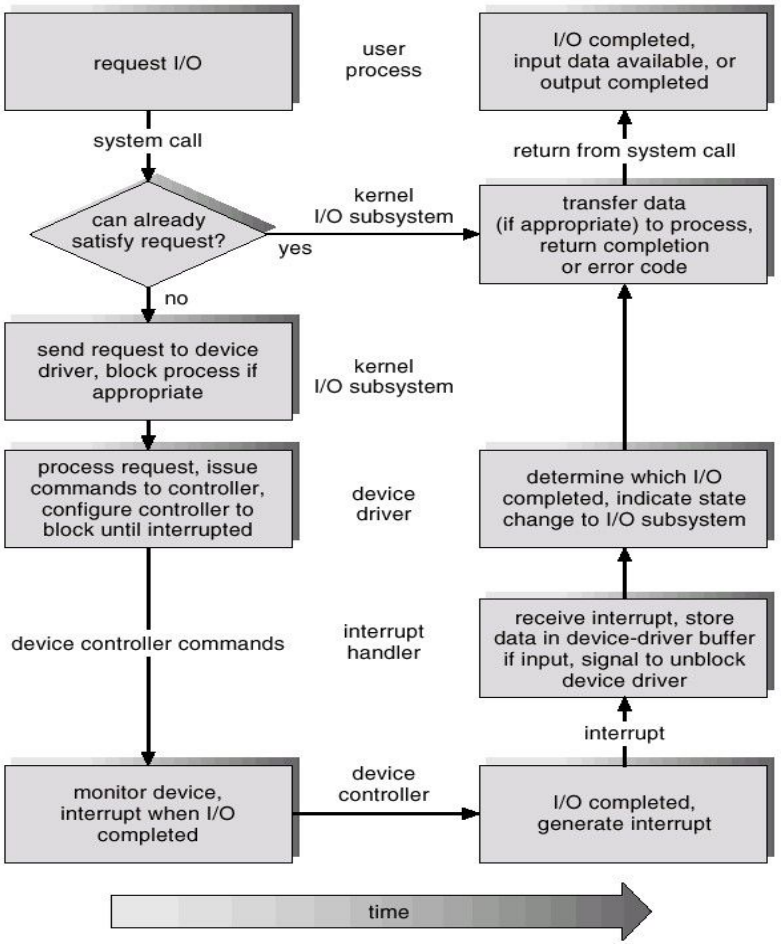
Estructuras de Datos: El kernel mantiene la información de cada dispositivo o componente (punteros a archivos abiertos, conexiones de red, etc) que se llama PCB (process control block).

Desde el Requerimiento de I/O hasta el Hardware: Consideremos la lectura sobre un archivo en un disco

-Determinar el dispositivo que almacena los datos

-Traducir el nombre del archivo en la representación del dispositivo.

-Traducir requerimiento abstracto en bloques de disco (Filesystem)
-Realizar la lectura física de los datos (bloques) en la memoria
-Marcar los datos como disponibles al proceso que realizó el requerimiento
-Desbloquearlo
-Retornar el control al proceso



Drivers: contienen el código dependiente del dispositivo. Manejan un tipo dispositivo. Traducen los requerimientos abstractos en los comandos para el dispositivo (Escribe sobre los registros del controlador, acceso a memoria mapeada, encola requerimientos). Comúnmente las interrupciones de los dispositivos están asociadas a una función del driver
Son una interfaz entre el sistema operativo y el dispositivo. Forman parte del espacio de memoria del kernel (Se cargan como módulos). Los fabricantes de Hardware implementan el driver como una API especificada por el sistema operativo (Osea, el Sistema Operativo pide un Read() y la API del dispositivo ejecuta el read para que lo entienda él mismo)

En linux se distingue 3 tipos de dispositivos: Carácter, Bloque(DMA), Red. Los drivers tiene que, al menos, tener las operaciones init_module para instalarlo y cleanup_module para desinstalarlo.

Las operaciones básicas que debe incluir la E/S:

- Open: Abre el dispositivo
- Release: Cerrar el dispositivo
- Read: Leer bytes del dispositivo
- Write: Escribir bytes en el dispositivo
- ioctl: Orden de control sobre el dispositivo (Input Output control será?)

- Hay otras operaciones menos comunes como: llseek, flush, poll, mmap.. etc (diapo 23)

Por convención, los nombres de las operaciones comienzan con el nombre del dispositivo, por ejemplo /dev/ptr.

Performance:

La E/S es uno de los factores que más afectan a la performance del sistema, ya sea porque:

- Usa mucha CPU para ejecutar los drivers y el código de la E/S
- Provoca context switches por interrupciones y bloqueos
- Utiliza el bus de memoria en copia de datos.

Para mejorar la performance:

- Reducir el número de context switches
- Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación
- Reducir la frecuencia de interrupciones usando polling, controladores más inteligentes y transferencias de gran cantidad de datos
- Utilizar DMA

Anexo clase 4:

Comunicación entre CPU-Controladora:

- Para que la CPU pueda ejecutar comandos o enviar/recibir datos de una controladora de un dispositivo, la controladora tiene uno o más registros (Registros para señales de control y para datos)

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

Comandos de E/S:
- CPU emite direcciones para identificar el dispositivo
- CPU emite comandos
    - Control: Qué hacer
    - Test: Controlar el estado
    - Read/Write: Transferir info desde o hacia el dispositivo

Mapeo de E/S y E/S aislada:
- **Mapeo de E/S**: Los dispositivos y la memoria comparten el espacio de direcciones
    - E/S es como escribir/leer en memoria
    - No hay instrucciones especiales para E/S. (Se usan las mismas de lectura y escritura tanto para memoria como para la E/S)
- **E/S aislada**: Las instrucciones de E/S tiene un espacio de direcciones distinto a las de memoria. Hay instrucciones especiales. Se necesitan líneas de E/S y puertos de E/S

E/S programada: La CPU tiene control directo sobre la E/S, controlando el estado,comandos para leer/escribir, transfiere los datos. La CPU espera a que el componente de E/S termine la operación, perdiendo así ciclos de CPU

Polling: En la E/S es necesario hacer polling del dispositivo para determinar el estado del mismo: Si está listo, ocupado o hubo error. Ciclo de Busy-Wait para realizar la E/S, pudiendo ser muy costoso si la espera es larga (Polling is the process where the computer or controlling device waits for an external device to check for its readiness or state, often with low-level hardware)

E/S manejada por interrupciones: Soluciona el problema de la espera de la CPU. La CPU no se queda chequeando si termina o no el dispositivo, siguiendo así ejecutando instrucciones. Cuando la E/S termina, manda una interrupción a la CPU.

DMA: controla el intercambio de datos directo entre RAM y el dispositivo. La CPU es interrumpida luego de que el bloque entero fue transferido

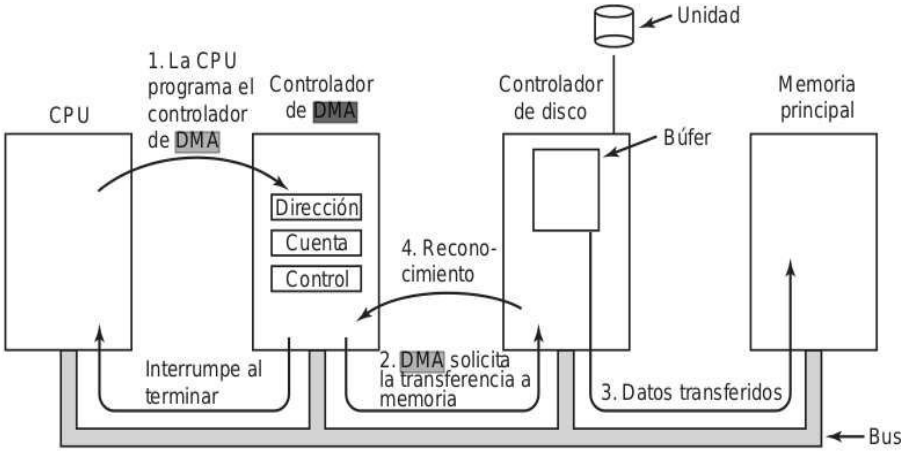


Figura 5-4. Operación de una transferencia de DMA.

Tema 5 - FileSystem

Guardamos archivos para 3 propósitos generales: almacenar grandes cantidades de datos, tener almacenamiento a largo plazo y permitir a distintos procesos acceder al mismo conjunto de información.

Un archivo es una entidad abstracta con nombre. Esta posee un espacio lógico continuo y direccionable. Es el encargado de proveer a los programas datos de entrada y guardar los de la salida. Hasta también guardar el mismo programa en sí.

El usuario puede interactuar con el archivo asignándole un nombre y decidir sobre su seguridad y acceso compartido, es decir, que y quienes puede interactuar con el archivo, pero no puede asignarle un espacio físico dentro del SO.
Por otro lado, el SO debe implementar el funcionamiento de archivos y directorios y mantener eficientemente el espacio libre y ocupado del disco.
Ambos pueden crear, borrar, buscar, copiar, leer, escribir, entre otros

Existen archivos de texto plano, llamados “source files” o archivos binarios que son o “Object files” o “Executable files”.

Un archivo posee ciertos datos propios como: nombre, identificador, tipo, localización, tamaño, tipo de protección o seguridad, última de vez de modificación, creación, acceso, etc.

Un File System o sistema de archivos es el encargado de brindarle una abstracción al programador en el momento de acceder a ciertos archivos mediante las aplicaciones. Entre sus funciones principales se encuentran:

- Gestionar los datos
- Cumplir con las solicitudes del usuario
- Eliminar (o al menos minimizar) la posibilidad de pérdida de datos
- Dar soporte de las E/S a distintos dispositivos
- Brindar interfaces a las E/S para el tratamiento de archivos.
- Proveer la posibilidad de compartir archivos (ambientes multiusuario)

El esqueleto principal de un FileSystem son los directorios. Estos son en sí mismos, archivos. Contienen la información acerca de los archivos y otros directorios dentro de él. Entre las operaciones que estos permiten, se encuentran: buscar, crear o borrar un archivo, listar el contenido, renombrar un archivo, etc.

Cómo diferentes archivos pueden tener el mismo nombre, los directorios pueden solventar este problema ubicándolos en diferentes lugares. Otra gran función es poder agrupar lógicamente archivos por funcionalidades. Ej: programas de java, juegos, librerías, etc.

Los directorios se ordenan en forma de raiz, y cada directorio conforma un elemento de esta raíz o **PATH**.
Tenemos el path relativo que puede ser : `\Summer2018.pdf`
Mientras que el absoluto (o fullpathname) sería:
`C:\Documents\Newsletters\Summer2018.pdf` usuario propietario
Nótese cómo el relativo puede llegar a repetirse pero nunca el absoluto.

En cuanto a la seguridad tanto de archivos cómo directorios, el usuario puede decidir sobre los derechos de acceso. Es decir, decide quien o quienes pueden ejecutar, leer, agregar datos, borrar datos, cambiar los derechos y hasta eliminar ese archivo.

Objetivos del FileSystem: brindar espacio a disco a los archivos del usuario y sistema. Además también debe mantener registros de cuánto espacio libre se tiene, y la cantidad y ubicación de los archivos. De esto se ocupa la FAT (file allocation table).
El Disco, está dividido en Bloques o cluster, y a su vez estos están divididos en sectores.

Conceptos:

Sector: Unidad de almacenamiento utilizada en los Discos Rigidos.

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 almacenados los distintos archivos.

Pre-Asignacion

- Se necesita saber cuanto espacio va a ocupar un archivo en el momento de su creación. Es usual definir espacios mucho mas grandes que los necesarios
- Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo.

Asignacion dinámica

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

Formas de Asignacion

Continua:

Se utiliza un conjunto continuo de bloques.

Para esta forma se tiene que conocer el tamaño del archivo durante su creación.

FAT es simple, posee una entrada que tiene un bloque de inicio y longitud.

El archivo puede ser leído con una única operación.

Puede existir fragmentación externa.

- Compactación.

Que problemas tiene esta técnica?

Encontrar bloques libres continuos en el disco.

El incremento del tamaño de un archivo.

ú Encadenada:

· Asignación en base a bloques individuales. Cada bloque tiene un puntero a su próximo bloque.

· FAT . Única entrada por archivo, tamaño y bloque de inicio.

· No hay fragmentación externa.

· Útil para acceso secuencial.

· Los archivos pueden crecer bajo demanda, no se requiere bloques contiguos.

ú Indexada

· Asignación en base a bloques individuales.

· No se produce fragmentación externa.

· El acceso “random” a un archivo es eficiente.

- FAT. Única entrada con la dirección del bloque de índices (i-nodo)
- Tiene algunas variantes:
    - Asignación por secciones
    - Niveles de indirección.

- **Gestion del espacio libre**
    - Control sobre cuales bloques de disco están disponibles.
    - Alternativas:
        - Tabla de bits
        - Bloques libres encadenados
        - Indexacion.
    - **Tabla de bits:**
        - Vector con un bit por cada bloque de disco
        - Cada entrada:
            - **0 = libre. 1 = ocupado.**
            - Ventaja -> Fácil encontrar un bloque o grupo de bloques libres.
            - Desventaja -> Tamaño de vector en memoria.
    - **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 libre.
        - Problemas con la perdida de un enlace.
        - Difícil de encontrar bloques libres consecutivos.
    - **Indexación:**
        - Variante de “bloques libres encadenados”
        - El primer bloque libre tiene las direcciones de N bloques libres.
    - **Recuento:**
        - Variante de la Indexación.
        - Esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o liberados a la vez (en especial con asignación contigua).
        - En lugar de tener N direcciones libres (índice) se tiene:
            - La dirección del primer bloque libre
            - Los N bloques libres contiguos que le siguen.
- **Estructura del Volumen:**
    - **Bloque de arranque:** Contiene el código requerido para arrancar el sistema.

§ **Superbloque:** Contiene atributos e información sobre el sistema de ficheros, como el tamaño de partición y de la tabla de i-nodos. Bloques/clusters libres.
§ **Tabla de i-nodos:** La colección de nodos-i para cada fichero.
ú **I-nodo** -> estructura de control que contiene la información clave de un archivo.
§ **Bloques de datos:** El espacio de almacenamiento disponible para los ficheros de datos y subdirectorios.

- **Windows – File Systems soportados**
  - **CD-ROM FS (CDFS) -> CD**
  - **Universal Disk Format (UDF) -> DVD, Blu-Ray**
  - **File Allocation Table (FAT)**
    § **FAT12 -> MS-DOS, floppy**
    § **FAT16 -> MS-DOS, nombres cortos de archivo.**
    § **FAT32 -> MS-DOS, nombres largos de archivo, pero no soportados en MS-DOS**
  - **New Technology File System (NTFS)**

  - **FAT**
    § Utilizado originalmente por DOS y Windows 9x
    § ¿Por qué Windows lo sigue soportando?
    ú Por compatibilidad con otro SO en sistemas multiboot
    ú Para permitir upgrades desde versiones anteriores.
    ú Para formato de dispositivos como diskettes
    § El numero de las versiones de FAT indica la cantidad de bits que usan para identificar diferentes bloques o clusters.

    § 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 está en la FAT y no en los bloques
    § Bloques libres y dañados tienen códigos especiales.

  - **FAT12**
    § Hay 2^12 sectores (4096) en un volumen.
    § Tamaño total de volumen para Windows de 32 MB -> 2^12 * 8KB.

        - § FAT12 como sistema de archivos de los disketts.
    - **FAT16**
        - § Hay 2^16 sectores (65536) en un volumen.
        - § En Windows el tamaño de sector en FAT16 varia entre los 512 bytes hasta los 64KB, tamaño máximo de 4GB.
        - § El tamaño del sector depende del volumen al formatearlo.
    - **FAT32**
        - § Utiliza 32 bits para la identificación de sectores, reserva 4 bits, por ende se utilizan 28 efectivamente.
        - § Capacidad de 8 TB
        - § Es mas eficiente que FAT16 , puede direccionar volúmenes de hasta 128GB.
    - **NTFS**
        - § Usa 64 bits para referenciar sectores.
        - § Permite volúmenes de hasta 16 Exabytes.
    - **¿Por qué NTFS en vez de FAT?**
        - § FAT es simple, mas rápido para ciertas operaciones, pero NTFS soporta:
            - ú Tamaños de archivos y de discos mas grandes.
            - ú Mejora performance en discos grandes
            - ú Nombres de archivos de hasta 255 caracteres.
            - ú Atributos de seguridad
            - ú Transaccional.

FALTAN COSAS

Tema 6 - Buffer Caché

Buffers en memoria principal para almacenar temporalmente bloques de disco, con el objetivo de disminuir la frecuencia de acceso al disco.

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
- Se trabaja como memoria compartida → Permite acceso a varios procesos
    - Este 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)
    - Cuando un bloque se referencia o entra en la caché, queda al final de la lista
- No se mueven los bloques, se asocian los punteros
- Otra alternativa, LFU en que se reemplaza el que tiene menos referencias.

Buffer Cache Unix System V

Objetivo y estructura:

- Minimizar la frecuencia de acceso al disco.
- Es una estructura formada por buffers.
- El kernel asigna un espacio en la memoria durante la inicialización para la estructura
- Un buffer tiene 2 partes:
    - **Header**: Con información del bloque, número de bloque, estado, relación con otros buffers, etc
    - **El buffer en sí**: El lugar donde se almacena el bloque de disco traído a memoria

Header: Identifica el número de dispositivo y bloque, estado y punteros

- Punteros a:
    - 2 punteros para la cola hash
    - 2 punteros para la free list
    - 1 puntero al bloque en memoria

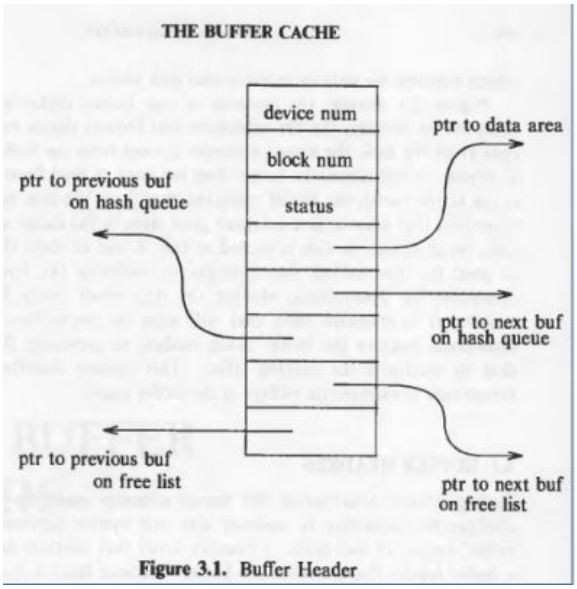


Figure 3.1. Buffer Header

Estado de los buffers: **Disponible**, **No disponible** (por estar en uso de parte de algún proceso), se está **escribiendo o leyendo** del disco, **Delayed Write** (buffers modificados en memoria pero los cambios no han sido reflejados en el bloque original en disco. *[Parecido a la idea de parcialmente cometida de bitácora])*

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 delayed write)
Se ordena según LRU. Sigue el mismo esquema de la Hash Queue pero contiene los headers de los buffers de aquellos procesos que ya han terminado.
El header de un buffer está siempre en la Hash Queue
Si el proceso que lo referenciaba terminó, va a estar en la Hash Queue y en la Free List

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 la tupla (dispositivo, número de 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 eficiente.
Para agrupar los bloques se utilizan los punteros que anteriormente habíamos visto que se almacenaban en el header

Funcionamiento del Buffer Caché: Cuando un proceso quiere acceder a un archivo utiliza su inodo para localizar los bloques de datos donde se encuentra éste.
El requerimiento llega al buffer 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.
2) El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
3) Idem 2, pero el bloque libre esta marcado como DW.
4) El kernel no encuentra el bloque en la hash queue y la free list está vacía.
5) El kernel encuentra el bloque en la hash queue pero está BUSY.

Ejemplos de los escenarios desde la diapo 16. (recomendable ver)

Respuestas de las preguntas de repaso.

Memoria

1. En paginación, si disminuyo la cantidad de bits del desplazamiento de una dirección de memoria, los frames serán más chicos. V

2. La cantidad máxima de páginas en memoria depende sólo del tamaño del proceso. F

3. Analice tamaños de página y page fault.

Tamaño de página muy pequeño	Tamaño de página muy grande
-menos fragmentación interna -más páginas requeridas por procesos -la tabla de páginas es más grande -más páginas pueden residir en memoria -menos probabilidad de fallo de página	-más fragmentación interna -la memoria secundaria está diseñada para transferir grandes bloques de datos por lo tanto es más eficiente -menos probabilidad de fallo de página

4. Relación de tamaño de página, de proceso, de tabla de páginas según la arquitectura de la dirección.
direccion: pagina--desplazamiento

- 2BITS-8BITS
    - Tenemos pocas paginas osea la tabla de páginas va a tener 2^2 entradas pero cada página es grande y tiene 2^8 direcciones.
- 5BITS-6BITS
    - Tenemos 2^5 entradas a la tabla de páginas (tabla grande) y 2^6 direcciones en cada página(página más chica que el caso anterior).

5. La tabla invertida proporciona acceso directo al marco buscado. V

6. Qué consecuencias puede tener la hiperpaginación.
Que el procesador esté más tiempo atendiendo PF que realmente ejecutando a los procesos, esto baja la *performance* del sistema.

7. En la técnica del Conjunto Trabajo ¿Qué pasa si el delta elegido es muy chico? ¿Y si es muy grande?
Si el Delta es muy chico, no se abarca correctamente toda la localidad, por lo que necesitaría más páginas.
Si el Delta es muy grande, se estarían tomando más localidades de las que necesita.

8. Diferencia entre reemplazo de páginas global y local.
En la local, si un proceso tiene PF, tiene que reemplazar una página de su conjunto residente por la página necesitada.

En cambio, si es global, podrá reemplazar una página de otro proceso que no esté utilizando/bloqueada para la que necesita.

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

10. Diferencia entre asignación equitativa y proporcional.
- Reparto equitativo: se asigna la misma cantidad de marcos a cada proceso → m div p
- Reparto proporcional: se asignan marco en base a la necesidad que tiene cada proceso → V p . m / V t

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

Se genera cuando se pide una página que no está en su conjunto residente (Bit V en 0). El HW detecta esto y genera un TRAP(excepción), a lo que el SO pone el proceso en Bloqueado(espera) mientras se gestiona la carga de la página correspondiente.

El SO busca un Frame Libre, genera una operación de E/S al disco para copiar en ese frame la página que se necesita. Mientras esa E/S se completa, se asigna a la CPU otro proceso.

Cuando la E/S termina, envía una interrupción, actualiza la tabla de páginas poniendo el bit V en 1, y la dirección Base del frame donde se encuentra la página en RAM. El proceso se encola en la cola de Ready to Run. Cuando vuelva a ejecutarse, arranca de donde quedó.

No estaba en la TLB, por ende, se agrega para futuras referencias.

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

La óptima sería que no esté ni modificada ni referenciada.

Archivos

13. En qué momento se hace el chequeo sobre si el usuario puede acceder a un archivo: en el open? en cada read? en cada write?

En el open.

14. En Unix System V: ¿puede modificarse el i-nodo del archivo sin modificar el contenido del archivo en sí?

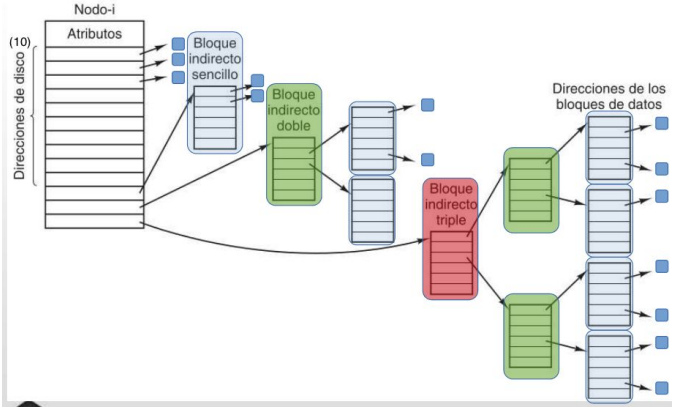
Si, se puede modificar muchas cosas (adjunto foto) sin modificar el contenido en sí.

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

15. En Unix System V: ¿puede modificarse el contenido del archivo sin modificar su i-nodo?

No, de hacerlo se moficiarian un par de cosas de la foto de arriba.

16. En Unix System V se verán beneficiados en performance los archivos cuyo contenido pueda ser referenciado por las 10 primeras direcciones de bloque que están en su i-nodo. V



17. En Unix System V, el acceso random a un archivo puede realizarse accediendo directamente al bloque que necesito, sin leer los precedentes. V

"La asignación indexada da soporte tanto a acceso secuencial como directo a los ficheros y por tanto es la forma más popular de asignación a ficheros"

18. En Unix System V, Puede asignarse un bloque a un archivo sin acceder

previamente al superblock? F

El superbloque de un sistema de archivos contiene la información que describe a dicho sistema. Su función principal consiste en indicar al FS el tamaño de las distintas partes del propio sistema de archivos. Los campos de un superbloque se muestran en la siguiente figura. Hay que tener presente que cuando cargamos en memoria el superbloque vamos a tener una serie de campos más que no se almacenan en el disco.

En memoria existe una tabla de estructuras superbloque donde en cada entrada se colocarán los datos del superbloque de cada sistema de ficheros montando en el sistema.

Presente en el disco y en la memoria	ino_t s_ninodes;	Números de nodos i en el sistema de ficheros
	zone1_t s_nzones;	Tamaño del sistema de ficheros en zonas (v1). Incluidos los bitmaps
	short s_imap_blocks;	Número de bloques del mapa de bits de los nodos-i
	short s_zmap_blocks;	Número de bloques del mapa de bits de las zonas
	zone1_t s_firstdatazone;	Número de zona en el que comienza la zona de datos
	short s_log_zone_size;	Log2 bloques por zona. Para calcular mediante bit-shifting los bloques de una zona
	off_t s_max_size;	Tamaño máximo de un archivo
	short s_magic;	Número mágico identificativo del tipo de sistema de ficheros y versión
	short s_pad;	Relleno
	zone_t s_zones;	Número de zonas (v2)
Presente en memoria y no en el disco	struct inode *s_isup;	Apuntador al nodo-i del directorio raíz del sistema de archivos
	struct inode *s_imount;	Apuntador al nodo-i del directorio en el que se ha montado el sistema de archivos
	unsigned s_inodes_per_block;	Nodos-i por bloque
	dev_t s_dev;	Número del dispositivo
	int s_rd_only;	Flag de lectura solamente
	int s_native;	Big-endian flag
	int s_version;	Versión del sistema de ficheros
	int s_ndzones;	Número de entradas de zonas directas por nodo-i
	int s_nindirs;	Número de zonas indirectas por bloque indirecto

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

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

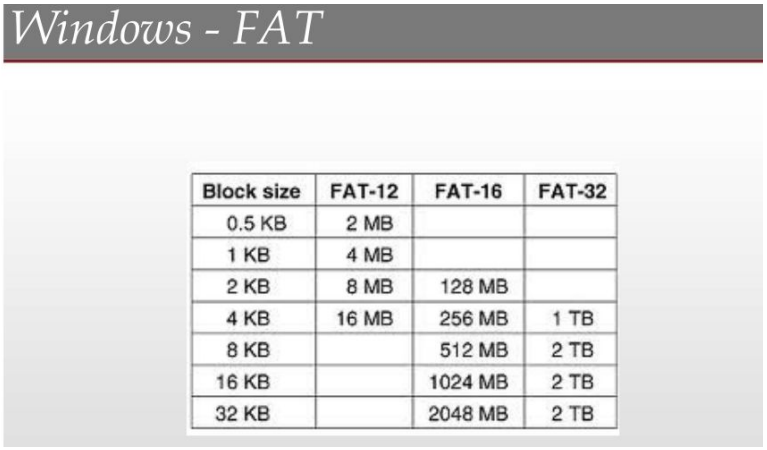
ESTRUCTURA DEL VOLUMEN

Un sistema de ficheros UNIX reside en un único disco lógico o partición de disco y se compone de los siguientes elementos:

- **Bloque de arranque.** Contiene el código requerido para arrancar el sistema operativo.
- **Superbloque.** Contiene atributos e información sobre el sistema de ficheros, tal como el tamaño de la partición y el tamaño de la tabla de nodos-i.
- **Tabla de nodos-i.** La colección de nodos-i para cada fichero.
- **Bloques de datos.** El espacio de almacenamiento disponible para los ficheros de datos y subdirectorios.

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

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



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

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

https://www.techopedia.com/definition/10143/zero-filling

24. La estructura del filesystem define el tamaño máximo del archivo. V
25. La estructura del filesystem define la longitud máxima del nombre. V

Buffer Cache

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

27. El inodo de un archivo que se está usando debe estar en algún buffer del buffer cache. V

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

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

30. Un proceso esperando por un buffer delayed write y ocupado, deberá esperar la escritura a disco antes de que se le asigne a él. (importa el bloque en el buffer). Falso por respuesta 26.

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

32. La free list sirve para buscar por cualquier buffer. F. Solo los vacios.

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

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

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

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

Al final de la free list.

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

Queda en espera el proceso.

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

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

40. Que conviene mas? mas cantidad de hash queues con pocos elementos, o menos cantidad de hash queues con mas elementos.

Más cantidad de hash queues con menos elementos, con esta opc, tendrias q moverte menos por la lista de la hash.