Tema 4. GESTION DE ARCHIVOS

1. Interfaz de los sistemas de archivos
2. Diseño del sistema de archivos
3. Generalidades sobre el sistema de archivos en Unix
4. Implementación de sistemas de archivos en Linux

(El presente archivo contiene los puntos 1 al 3; el 4 se entregará posteriormente)

•**Bibliografía** :

Stallings, W.; *Sistemas Operativos. Aspectos Internos y Principios de Diseño* (5/e), Prentice Hall.
Marquez, Fco.; *Unix programación avanzada*; Ed. Ra-Ma
Aranda, J.; *Sistemas Operativos. Teoría y Problemas*. Ed. Sanz y Torres
Love, R.; *Linux Kernel Development (3/e)*, Addison-Wesley Professional
Mauerer, W.; *Professional Linux Kernel Architecture*, Wiley

*** LECTURAS PARA CUBRIR LOS CONTENIDOS DE LOS PUNTOS 1 y 2:**
Del libro de Stallings, puntos 12.3 hasta el 12.6 ambos inclusive.
*** LECTURAS PARA CUBRIR LOS CONTENIDOS DEL PUNTO 3:**
Del libro de Marquez, páginas 13 a 34 ambas inclusive.

*** PROBLEMAS RESUELTOS**:
Del libro de Aranda se estudiarán:
Cuestiones 5-4, 5-7, 5-10, Problemas 5-3, 5-5, 5-6, 5-7, 5-9 a 5-14

1. Interfaz de los sistemas de archivos.

- **Archivo**:
  - Colección de información relacionada y almacenada en un dispositivo de almacenamiento secundario, el usuario puede hacer uso de esta información sin verse involucrado en cómo está almacenada en el dispositivo de almacenamiento secundario.
- **Estructura interna (lógica) de un archivo, ejemplos de posibilidades:**
  - secuencia de bytes
  - secuencia de registros de longitud fija
  - secuencia de registros de longitud variable
- **Formas de acceso a un archivo:** secuencial, aleatorio, otros

- **Atributos (metadatos)**. En general podrían ser....
  - **Nombre**
  - **Tipo**
  - **Localización**: información sobre su localización en el dispositivo
  - **Tamaño**: tamaño actual del archivo
  - **Protección**: Especificación de qué operaciones pueden hacer sobre el archivo los distintos usuarios
  - **Tiempo, fecha e identificación** del usuario: necesario para protección, seguridad y monitorización

Operaciones sobre archivos

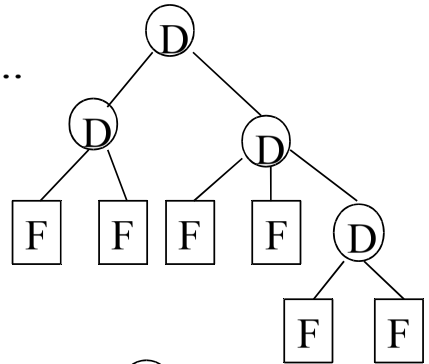
- **Gestión**:
  - Crear
  - Borrar
  - Renombrar
  - Copiar
  - Establecer y obtener atributos

- **Procesamiento**:
  - Abrir y Cerrar
  - Leer
  - Escribir (modificar, insertar, borrar información)

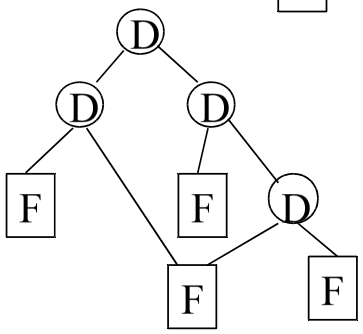
Arbol de archivos y directorios....

En árbol. Se usan los conceptos de...

- Directorio actual (de trabajo)
- Directorio inicial
- Rutas absolutas o relativas
- Lista de búsqueda
- Enlace duro y simbólico



En grafo . Un elemento puede estar incluido en más de un directorio padre



● **Protección.**

• Cada archivo/directorio tiene especificadas qué operaciones pueden realizar sobre él los distintos usuarios.

• El sistema operativo debe garantizar que las operaciones que se realicen sobre un archivo/directorio sean las permitidas.

● Principal solución a la protección: hacer el acceso dependiente del **identificativo del usuario**

● ACL: Access Control List. Cada archivo/directorio tiene especificada una lista formada por pares con el significado de:

usuario, operaciones permitidas

● Posibilidad: agrupación de usuarios:

- propietario
- grupo
- resto

● Propuesta alternativa: Asociar un *password* con el archivo.

Problemas:

- Recordar todos
- Si solo se asocia un *password* → acceso total o ninguno

- **Funciones básicas del Sistema de Archivos....**

- Tener conocimiento de todos los archivos del sistema
- Controlar la compartición y la protección de archivos
- Gestionar el espacio del sistema de archivos
  - Asignación/liberación del espacio en el medio de almacenamiento
- Traducir las direcciones lógicas del archivo en direcciones físicas del disco
  - Los usuarios especifican las partes que quieren leer/escribir en términos de direcciones lógicas relativas al archivo

2. Diseño del sistema de archivos

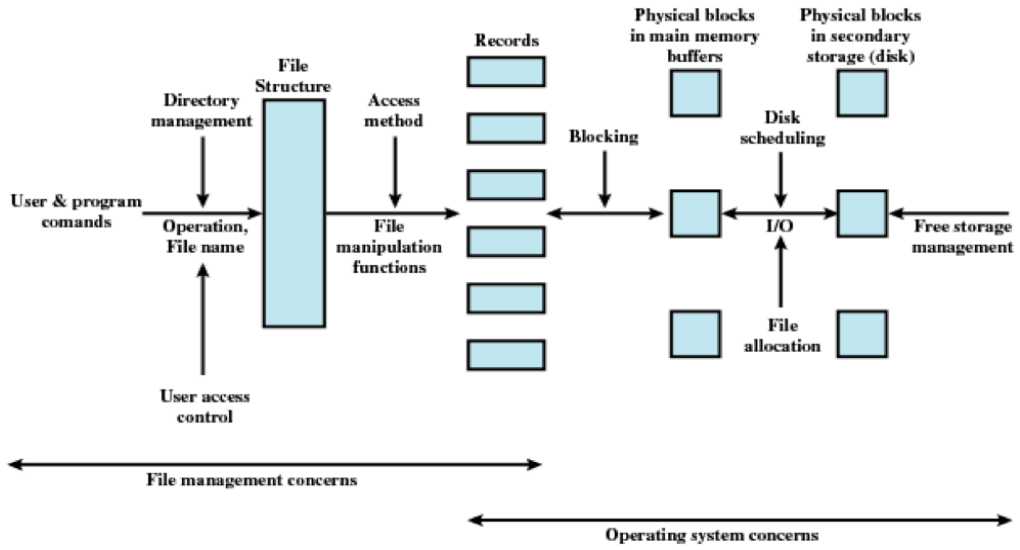


Figure 12.2 Elements of File Management

Figura 12.2 de Stallings

2.1 Métodos de Asignación de espacio: Asignación contigua

- Cada archivo ocupa un conjunto de bloques contiguos

- **Ventajas**

  Sencillo: solo necesita nº de bloque de comienzo y tamaño

  – Buenos tanto el acceso secuencial como el directo

- **Desventajas**

  Fragmentación externa

  Dificil solución al cambio de tamaño del archivo

Asociación lógica a física

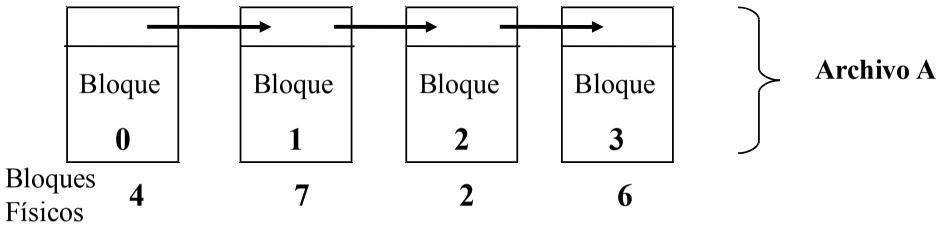
Supongamos que los bloques de disco son de NB bytes:

Dirección lógica DL / NB → C(cociente), R(resto)

- Bloque a acceder = C + dirección de comienzo
- Desplazamiento en el bloque = R

2.2 Métodos de Asignación de espacio: No Contiguo - Enlazado

- Cada archivo es una lista enlazada de bloques de disco. Los bloques pueden estar dispersos en el disco



- **Ventajas**
  - Evita la fragmentación externa
  - El archivo puede crecer dinámicamente
  - Basta almacenar el puntero al primer bloque del archivo

- **Desventajas**
  - El acceso directo no es efectivo, si el secuencial
  - Espacio requerido para los punteros de enlace.
    Solución: agrupaciones de bloques (*clusters*)
  - Seguridad por la pérdida de punteros
    Solución: lista doblemente enlazada

- **Asociación lógica a física** (dirección = 1byte)

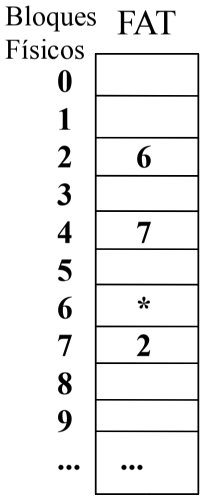
  Dirección lógica DL/(TamañoBloque -1) → C(cociente), R(resto)

  - Bloque a acceder = C-ésimo
  - Desplazamiento en el bloque = R + 1

● Tabla de Asignación de Archivos (***FAT***): variación del método enlazado (*Windows* y *OS/2*)

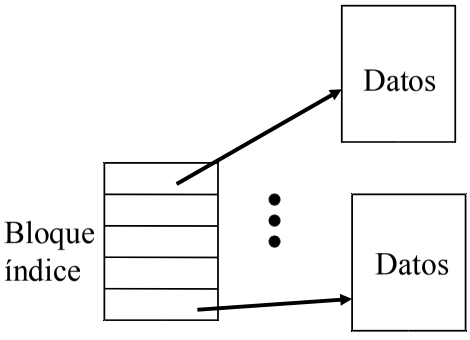
– Reserva una sección del disco al comienzo de la partición para la FAT
– Contiene una entrada por cada bloque del disco y está indexada por número de bloque de disco
– Simple y eficiente siempre que esté en caché
– Para localizar un bloque solo se necesita leer en la FAT → se optimiza el acceso directo
– Problema: pérdida de punteros → doble copia de la FAT

(En el ejemplo, la dirección de comienzo del archivo representado es 4)



2.3 Métodos de Asignación de espacio: No Contiguo - Indexado

● Todos los punteros a los bloques están juntos en una localización concreta: **bloque índice**
● El directorio tiene la localización a este bloque índice y cada archivo tiene asociado su propio bloque índice
● Para leer el i-*ésimo* bloque buscamos el puntero en la i-*ésima* entrada del bloque índice
● **Ventajas**
  –Buen acceso directo
  –No produce fragmentación externa



Desventajas

– Posible desperdicio de espacio en los bloques índices

– Tamaño del bloque índice. Soluciones:

(a) Bloques índices enlazados

(b) Bloques índices multinivel

- Problema: acceso a disco necesario para recuperar la dirección del bloque para cada nivel de indexación

- Solución: mantener algunos bloques índices en memoria principal

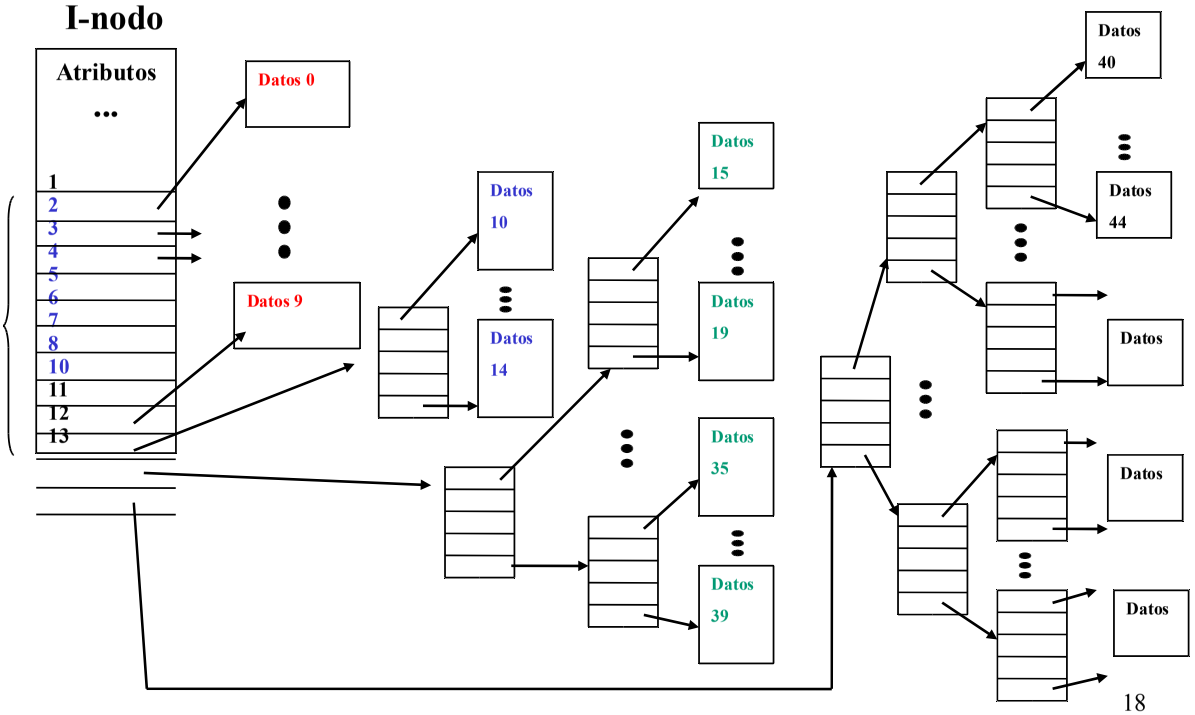
(c) Esquema combinado (Unix)

Variante Unix

Ejemplo en que en 1 bloque caben 5 direcciones debloque

Rojo: direccionamiento directo

Azul: indexación a 1 nivel; Verde: indexación a 2 niveles;



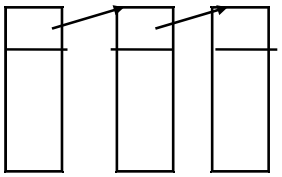
2.4 Gestión de espacio libre

- El sistema mantiene una lista de los bloques que están libres: **lista de espacio libre**
- La **FAT** no necesita ningún método
- La lista de espacio libre tiene diferentes implementaciones:

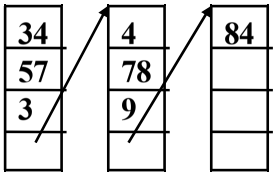
1. **Mapa o Vector de Bits**
   - Cada bloque se representa con un bit (0-Bloque libre; 1-Bloque ocupado)
   - Fácil encontrar un bloque libre o *n* bloques libres consecutivos. Algunas máquinas tienen instrucciones específicas
   - Fácil tener archivos en bloques contiguos
   - Ineficiente si no se mantiene en memoria principal

10010001
11111101
11100000
11111110
00000000
11100011
11100000

2. **Lista enlazada de bloques libres**
   Enlaza todos los bloques libres del disco, guarda un puntero al primer bloque en un lugar concreto



3. **Lista enlazada con agrupación**
   - Cada bloque de la lista almacena *n-1* direcciones de bloques libres
   - Obtener muchas direcciones de bloques libres es rápido



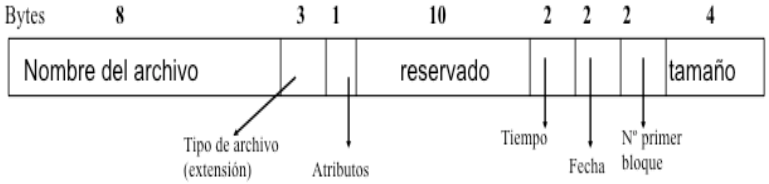
4. **Variante:** cada entrada de la lista contiene una dirección de bloque libre y un contador del nº de bloques libres que

2.5 Implementación de Directorios

- Contenido de una entrada de directorio. **Casos**:

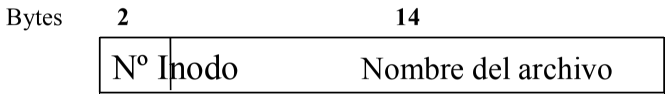
  (a) En MS-DOS:
  Nombre de Archivo + Atributos + Dirección de los bloques de datos

Entrada de directorio de MS-DOS



(b) Nombre de Archivo + Puntero a una estructura de datos que contiene toda la información relativa al archivo

Entrada de directorio de UNIX (s5fs)



Cuando se abre un archivo

- El SO busca en su directorio la entrada correspondiente
- Extrae sus atributos y la localización de sus bloques de datos y los coloca en una tabla en memoria principal
- Cualquier referencia posterior usa la información de dicha tabla

● **Implementación de enlaces**:

1. Enlaces **simbólicos**
   - Se crea una nueva entrada en el directorio, se indica que es de tipo *enlace* y se almacena la ruta o camino de acceso (absoluto o relativo) del archivo al cual se va a enlazar
   - Se puede usar en entornos distribuidos
   - Gran número de accesos a disco

2. Enlaces **absolutos** (o *hard*)
   - Se crea una nueva entrada en el directorio y se copia la dirección de la estructura de datos con la información del archivo
   - **Contador de enlaces**

2.6 Estructura del almacenamiento secundario: Estructura del Disco

● Desde el punto de vista del SO, el disco se puede ver como un array de bloques (B_0, B_1,...,B_{n-1})

● La información se referencia por una dirección formada por **varias partes**:
   - unidad (número de dispositivo),
   - superficie (o cara),
   - pista, y
   - sector

● Existe un esquema de asociación de la dirección de un bloque lógico B_i a dirección física (pista, sector, ...)
   - El área de asignación más pequeña es un bloque (1 o más sectores)
   - Fragmentación interna en los bloques

Planificación de Disco

- El SO puede mejorar el **tiempo medio de servicio** del disco
- Una petición se atiende en **tres fases**:
  1. **Posicionamiento** de la cabeza en la pista o cilindro
  2. **Latencia**: espera a que pase el bloque deseado
  3. **Transferencia** de los datos entre MP y disco

tº_total de servicio = tºposicionamiento + tº_latencia + tº_transferencia

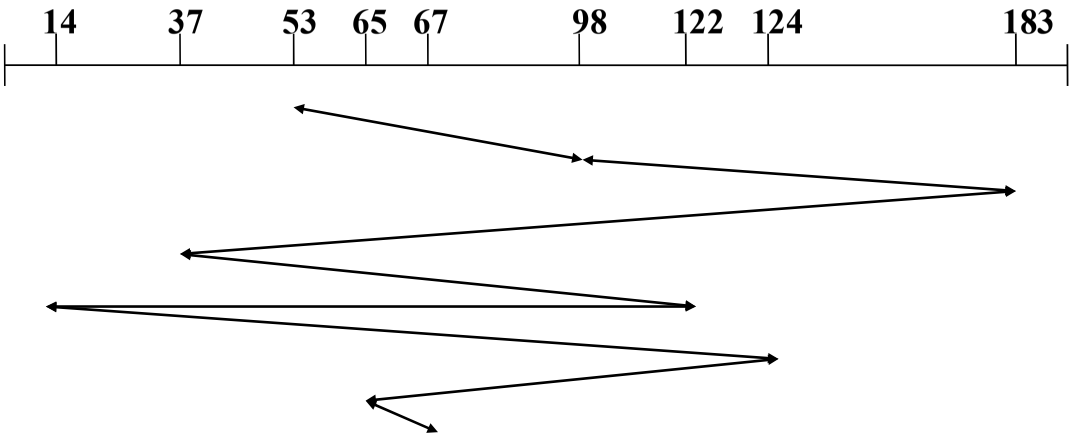
- La planificación intenta minimizar el **tiempo de posicionamiento** (principal factor: distancia de posicionamiento)
- Si el disco está ocupado, las peticiones se encolan

Planificación de Disco (II)

- Información necesaria para una petición:
  - Si la operación es de entrada o de salida
  - Dirección de bloque
  - Dirección de memoria a donde, o desde donde, copiar los datos a transferir
  - Cantidad de información a transferir
- Existen distintos algoritmos de planificación de peticiones
- **Ejemplo**: Cola de peticiones (números de pistas)
  98, 183, 37, 122, 14, 124, 65, 67.
  Inicialmente la cabeza está en la pista 53

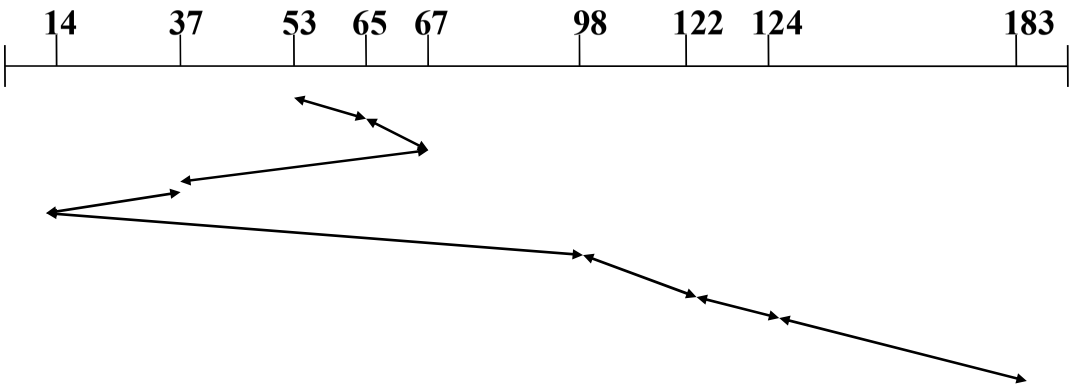
Planificación de Disco (III)

- **FCFS** (*First Come First Served*)



Planificación de Disco (IV)

- **SSTF** (*Shortest Seek Time First*): Primero la petición cuyo tº de posicionamiento sea más corto



Selección de un algoritmo de planificación

- En general, para cualquier algoritmo, el rendimiento depende mucho del **número** y **tipo** de las peticiones
  - si la cola está prácticamente vacía, cualquier algoritmo es válido

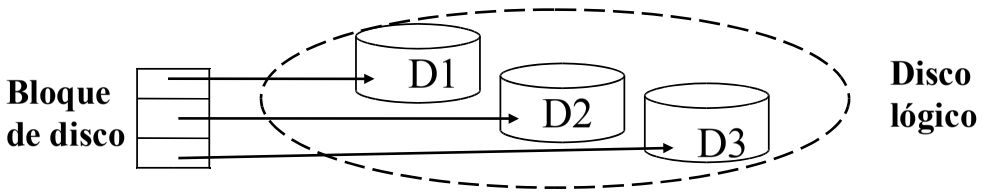
- El servicio de peticiones puede estar muy influenciado por el método de asignación de espacio en disco utilizado

  - con asignación contigua: peticiones que reducen el tiempo de posicionamiento

  - con asignación no contigua (enlazado o indexado): mayor aprovechamiento de disco pero mayor tiempo de posicionamiento

Eficiencia y Seguridad de Disco *(no entra en examen)*

- Técnica de **entremezclamiento** (*Striping*) de disco
  ⇨ Un grupo de discos se trata como una unidad. Cada bloque es ahora en realidad un conjunto de subbloques que se almacenan en discos diferentes
  ⇨Los discos **en paralelo** realizan el posicionamiento y transfieren sus bloques ➔ decrementa el tiempo de transferencia del bloque



(Ver tambien www.acnc.com/raid)

3. Generalidades sobre el sistema de archivos en Unix

- **i-nodo**: representación interna de un archivo
- Un archivo tiene asociado un único i-nodo
- Varios archivos pueden tener asociados el mismo nº de inodo
- Si un proceso.....

  -Crea un archivo ➔ se le asigna un i-nodo

  - Referencia a un archivo por su nombre ➔ se analizan permisos y se lleva el i-nodo a memoria principal

Estructuras de datos en M.P.:

(1) **Tabla de i-nodos**: **Conjunto de copias en memoria de los inodos más recientemente utilizados** (Estructura global)
El núcleo lee el i-nodo en m.p. cuando se opera con él.

(2) **Tabla de archivos**: Una entrada por **cada apertura** (Estructura global) Contenido de cada entrada:
- puntero al i-nodo correspondiente de la tabla de i-nodos
- offset de lectura/escritura
- modo de apertura del archivo
- contador de entradas de las tablas de descriptores de archivos asociados con esta entrada

(3) **Tabla de Descriptores de Archivos (o Tabla de Archivos Abiertos)**:

Estructura local a cada proceso.

Se crea una entrada por operación open

Archivos abiertos por omisión:

Entrada nº 0: Entrada estandar (teclado por omisión)

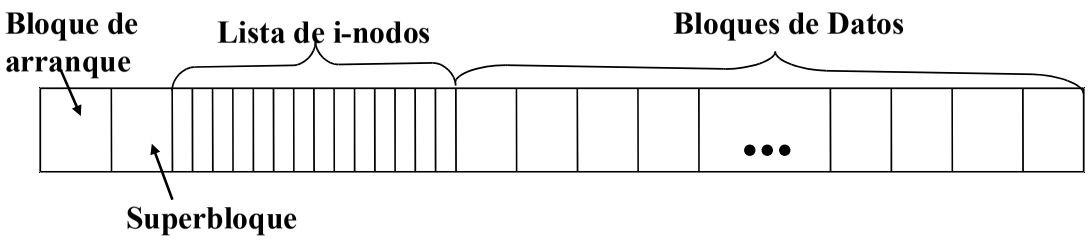
Entrada nº 1: Salida estandar (pantalla por omisión)

Entrada nº 2: Salida estandar de error (pantalla por omisión)

Contenido de la entrada nº i: puntero a la Tabla de archivos donde estarán los datos correspondientes a esta apertura.

Estructura en disco del sistema de archivos

- Un "sistema de archivos" es una secuencia de bloques lógicos con la siguiente estructura (***s5fs***):
  - **Bloque de arranque**: puede contener código de arranque para inicializar el SO
  - **Superbloque**: estado, características, del sistema de archivos
  - **Lista de i-nodos**: su tamaño es constante
    se almacena en el superbloque
    se especifica en la configuración del sistema de archivos
  - **Bloques de datos**



CONTENIDO DEL SUPERBLOQUE

- Tamaño del sistema de archivos

- Número de bloques libres en el sistema de archivos
- Una lista de bloques libres disponibles y un índice al siguiente bloque libre de la lista de bloques libres

- Tamaño de la lista de i-nodos

- Número de i-nodos libres en el sistema
- Una lista de i-nodos libres y un índice al siguiente i-nodo libre de la lista de i-nodos libres

- Además, cuando el superbloque está en memoria tiene:
  - Flag indicando si la lista de bloques libres está bloqueada
  - Flag indicando si la lista de inodos libres está bloqueada
  - Flag indicando que el superbloque ha sido modificado

Contenido de un i-nodo

- **Identificador del propietario** del archivo: UID, GID
- **Tipo de archivo** (regular, directorio, dispositivo, cauce, link). Si es 0 ➔ el i-nodo está libre
- **Permisos de acceso**
- **Tiempos de acceso**: última modificación, último acceso y última vez que se modificó el i-nodo
- **Contador de enlaces**
- Tabla de contenidos para las **direcciones de los datos** en disco del archivo
- **Tamaño**

Datos sobre un i-nodo en la tabla de i-nodos = Contenido de la “copia en MP” de un inodo

● El núcleo mantiene en la tabla de i-nodos, además del contenido del i-nodo, los siguientes campos:

- El **estado** del i-nodo en memoria, indicando si:
  - » el i-nodo está bloqueado
  - » un proceso está esperando a que el i-nodo se desbloquee
  - » la copia en memoria del i-nodo difiere de la copia en disco por cambio en los datos del i-nodo
  - » la copia en memoria del archivo difiere de la copia en disco por cambio en los datos del archivo
  - » el archivo es un punto de montaje

(… continúa….)

(…continuación…)

– El **número de dispositivo lógico** del sistema de archivos que contiene al archivo

– El **número de i-nodo**

– **Punteros a otros i-nodos** en memoria: se usa una estructura hash para enlazar los i-nodos;
la clave de acceso se basa en el número de dispositivo lógico y el número de i-nodo.
Punteros al siguiente/anterior inodo en la lista de i-nodos libres.

– **Contador de referencias**: número de referencias actuales al i-nodo (p.e. cuando varios procesos abren el mismo archivo)

Un i-nodo sólo estará en la lista de i-nodos libres cuando su contador de referencias sea 0.

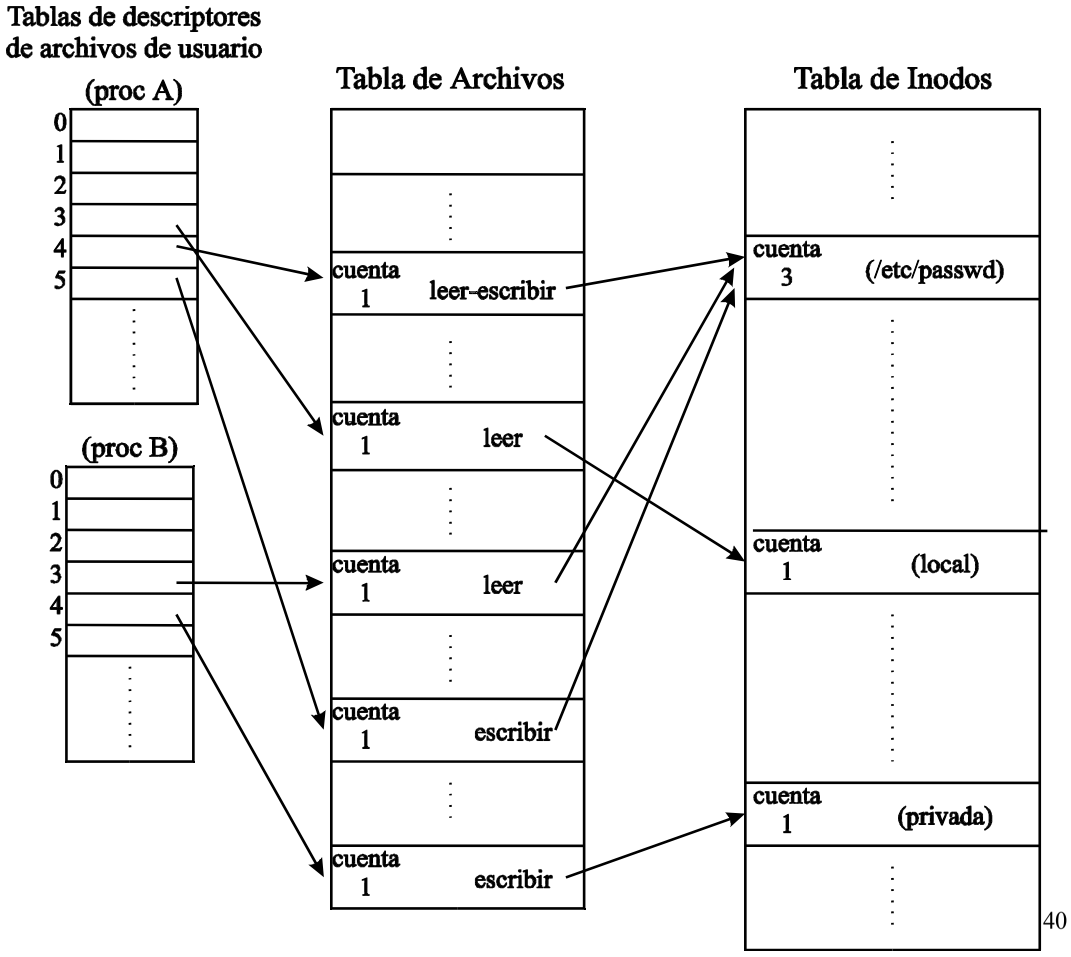
Ejemplo: los procesos A y B realizan estas aperturas:

PROCESO A:

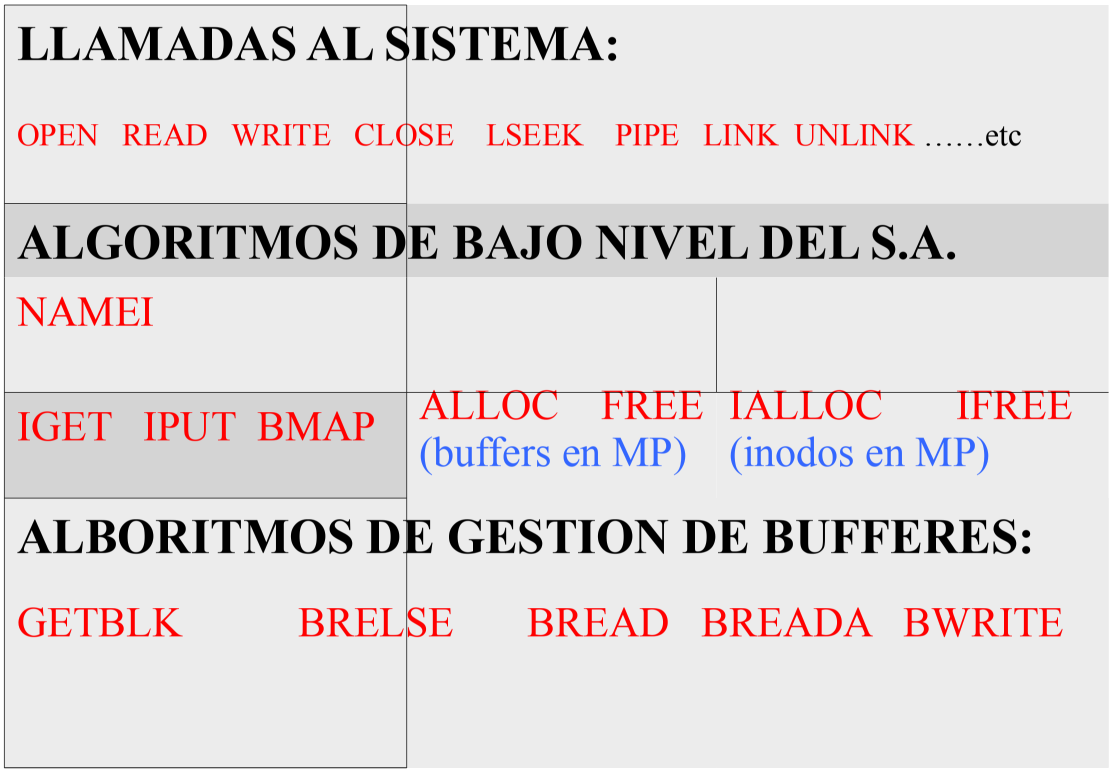
```
…
int fd1, fd2, fd3;
fd1 = open (“local”, O_RDONLY);
fd2 = open (“/etc/passwd”, O_RDWR);
fd3 = open (“/etc/passwd”, O_WRONLY);
```

PROCESO B:

```
…
int fd1, fd2;
fd1 = open (“/etc/passwd”, O_RDONLY);
fd2 = open (“private”, O_RDONLY);
```



Estructura del software del sistema de archivos



Buffer Caché

- Disminuye el nº de accesos a disco
- "Buffer caché": conjunto de bloques de disco que el S.O. mantiene en M.P.
- En lectura, el S.O. Comprueba si el bloque aludido se encuentra en el buffer-cache, evitando así una lectura
- En escritura, el bloque se mantiene también el un buffer para que no sea necesaria otra lectura si es aludido en el futuro
- El tamaño de la buffer caché se puede determinar durante la inicialización del sistema

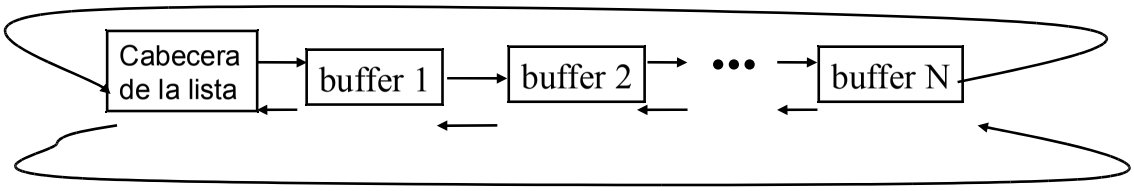
Partes de un buffer

1. **Copia de un bloque lógico de disco**. Un bloque de disco no se puede encontrar en mas de un buffer.
2. **Cabecera del buffer:**
   - Nº de dispositivo , Nº Bloque: Identificación del bloque
   - Puntero a la copia del bloque de disco
   - Estado, combinación de las siguientes condiciones:
     » bloqueado (=ocupado, no libre) (durante la realización de una llamada al sistema el buffer permanece bloqueado)
     » contiene datos válidos
     » **escritura retardada** o postergada (el núcleo debe escribir el contenido del buffer antes de reasignarlo)
     » actualmente se está realizando una E/S sobre el buffer
     » un proceso está esperando a que el buffer se libere
   - Punteros utilizados por los algoritmos de asignación para mantener la estructura del depósito de bufferes.

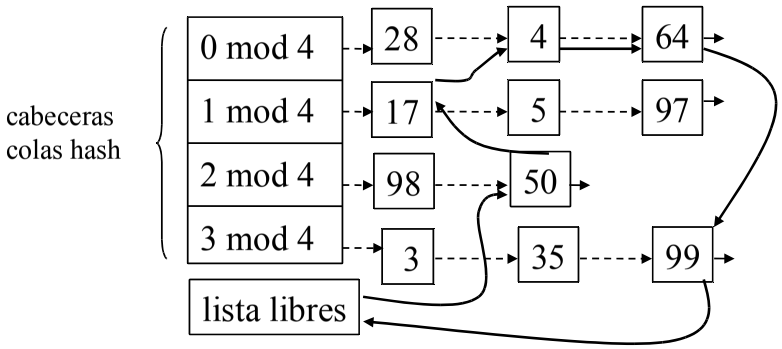
Estructura de la buffer caché

- Se gestiona mediante el algoritmo **LRU**
- **Organización**:
  – lista circular doblemente enlazada de bufferes libres
  – varias colas separadas (listas doblemente enlazadas). Utiliza una función *hash* basada en **nºdispositivo + nºbloque**
- El nº de elementos en una cola hash varía durante la vida del sistema
- Cada buffer está en una única cola hash aunque también puede estar en la lista de bufferes libres
- La posición del buffer dentro de la cola hash no es relevante

Inicialmente, todos los bufferes están libres



Después de un tiempo ...



- son **listas doblemente enlazadas**
- en el ejemplo sólo usamos el nº de bloque por simplicidad

Comentarios finales sobre el buffer-caché

- **Ventajas**
  - –reducción del trafico de disco, incrementando el rendimiento y decrementando el tº de respuesta
  - – los algoritmos aseguran la **integridad del sistema**
  - ➔ serializan el acceso a los datos
- **Desventajas**
  - – vulnerabilidad ante caídas del sistema

Algoritmos a bajo nivel del S.A.

– **iget** = devuelve un i-nodo de la tabla de i-nodos identificado previamente, si no está, lo carga en la tabla

– **iput** = libera un i-nodo de la tabla de i-nodos

– **namei** = convierte un *pathname* a un número de i-nodo

– **alloc** y **free** = asignan y liberan bloques de disco

– **ialloc** e **ifree** = asignan y liberan i-nodos de disco

– **bmap** = traducción de direcciones lógicas a físicas

● Supongamos el nombre de ruta */usr/ast/correo,*
¿cómo se convierte en un nº de i-nodo?