

Tema 4: Gestión de archivos

4.1. Interfaz de los sistemas de archivos

4.2. Diseño software del sistema de archivos

4.3. Implementación de la gestión archivos
en Linux

Objetivos

- ✓ Comprender el concepto de archivo, directorio y estructura de directorios
- ✓ Saber las distintas funciones que se realizan sobre un sistema de archivos y cómo se pueden estructurar
- ✓ Conocer las ventajas e inconvenientes de distintos métodos de asignación de espacio del almacenamiento secundario y de gestión de espacio libre
- ✓ Abordar distintos aspectos de implementación y valorar sus ventajas e inconvenientes
- ✓ Comprender la importancia de la gestión del almacenamiento secundario y cómo se puede optimizar su funcionamiento

4.1. Interfaz de los sistemas de archivo: Concepto de archivo

- Colección de información relacionada y almacenada en un dispositivo de almacenamiento secundario
- Espacio de direcciones lógicas contiguas
- **Estructura interna** (lógica)
 - secuencia de bytes: el tipo del archivo determina su estructura (texto → caracteres, líneas y páginas, código fuente → secuencia de subrutinas y funciones)
 - secuencia de registros de longitud fija
 - secuencia de registros de longitud variable
- **Tipos** de archivos: regulares, directorios, de dispositivo
- **Formas de acceso**: secuencial, aleatorio, otros

Concepto de archivo (y II)

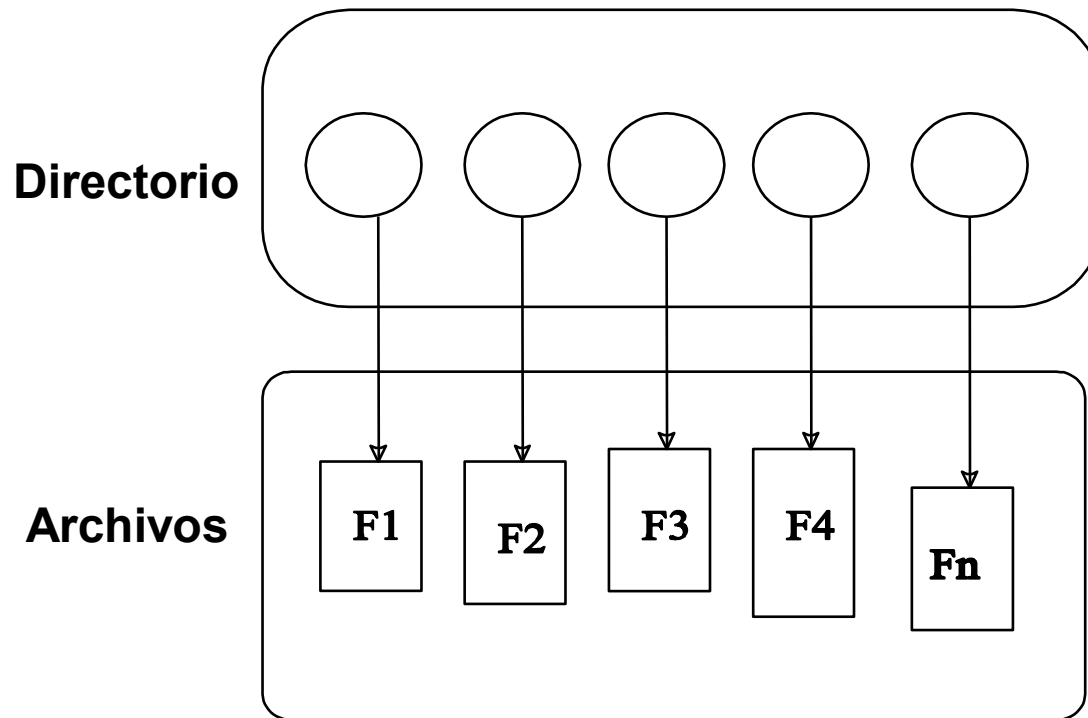
- Atributos (metadatos)
 - **Nombre**: única información en formato legible
 - **Tipo**: cuando el sistema soporte diferentes tipos
 - **Localización**: información sobre su localización en el dispositivo
 - **Tamaño**: tamaño actual del archivo
 - **Protección**: controla quién puede leer, escribir y ejecutar
 - **Tiempo, fecha e identificación del usuario**: necesario para protección, seguridad y monitorización

Concepto de archivo (y III)

- Operaciones sobre archivos
 - **Gestión:**
 - Crear
 - Borrar
 - Renombrar
 - Copiar
 - Establecer y obtener atributos
 - **Procesamiento:**
 - Abrir y Cerrar
 - Leer
 - Escribir (modificar, insertar, borrar información)

Estructura de Directorios

- Colección de nodos conteniendo información acerca de todos los archivos → **Organización**



- Tanto la estructura de directorios como los archivos residen en el almacenamiento secundario

Estructura de Directorios (y II)

La organización (lógica) de los directorios debe proporcionar:

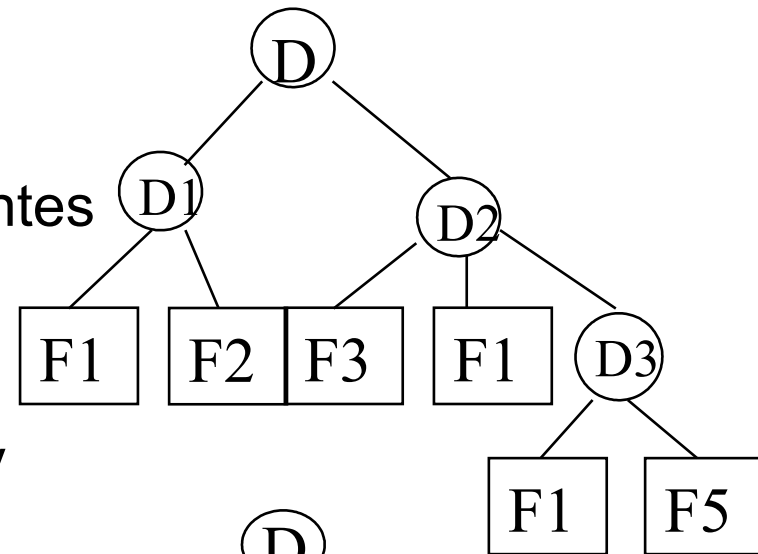
- **Eficiencia**: localización rápida de un archivo
- **Denominación**: adecuada a los usuarios
 - Dos usuarios pueden tener el mismo nombre para diferentes archivos
 - El mismo archivo puede tener varios nombres
- **Agrupación**: agrupar los archivos de forma lógica según sus propiedades (Ej. todos los programas en C)

Estructura de Directorios (y IV)

Casos:

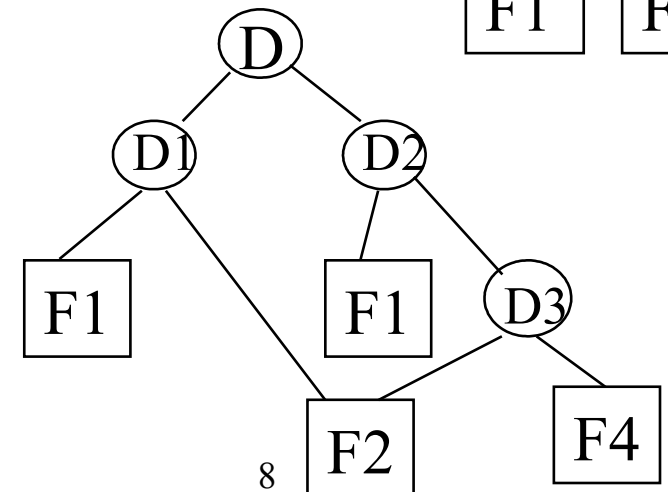
- **En árbol**

- Necesidad de búsquedas eficientes
- Posibilidad de agrupación
- Directorio actual (de trabajo)
- Nombres de camino absolutos y relativos



- **En grafo**

- Compartición de subdirectorios y archivos
- Más flexibles y complejos



Protección

- Básicamente consiste en proporcionar un acceso controlado a los archivos
 - lo que puede hacerse
 - por quién
- Tipos de acceso
 - Leer
 - Escribir
 - Ejecutar
 - Añadir
 - Borrar
 - Listar

Protección: Listas y Grupos de acceso

- Principal solución a la protección: hacer el acceso dependiente del **identificativo del usuario**
- Las **listas de acceso** de usuarios individuales tiene el problema de la longitud
- Solución con clases de usuario
 - propietario
 - grupo
 - público
- Propuesta alternativa: Asociar un *password* con el archivo. Problemas:
 - Recordar todos
 - Si solo se asocia un *password* → acceso total o ninguno

Semánticas de consistencia

- Especifican cuándo las modificaciones de datos por un usuario se observan por otros usuarios

- Ejemplos:

1. **Semántica de Unix**

- La escritura en un archivo es directamente observable
- Existe un modo para que los usuarios compartan el puntero actual de posicionamiento en un archivo

2. **Semánticas de sesión** (Sistema de archivos de Andrew)

- La escritura en un archivo no es directamente observable
- Cuando un archivo se cierra, sus cambios sólo se observan en sesiones posteriores

3. **Archivos inmutables**

- Cuando un archivo se declara como compartido, no se puede modificar

Funciones básicas del Sistema de Archivos

- Tener conocimiento de todos los archivos del sistema
- Controlar la compartición y forzar la protección de archivos
- Gestionar el espacio del sistema de archivos
 - Asignación/liberación del espacio en disco
- 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

4.2. Diseño software del sistema de archivos

- Un sistema de archivos posee dos **problemas de diseño** diferentes:

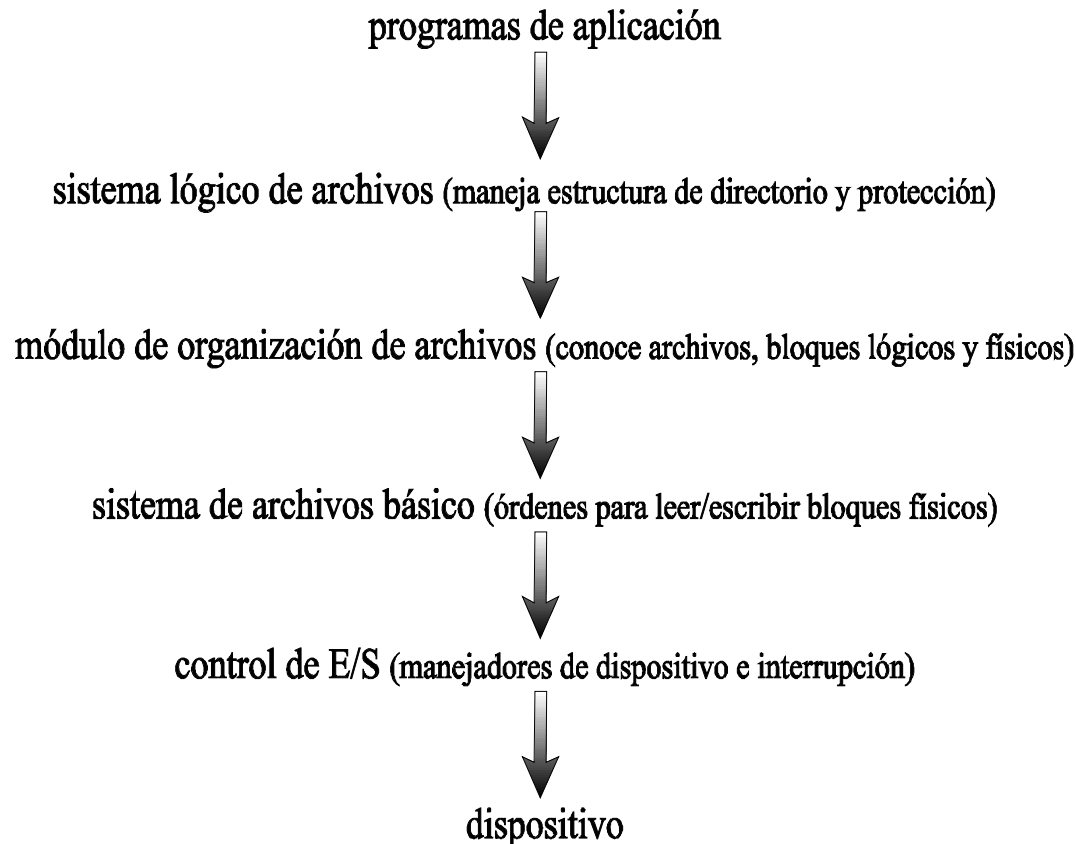
1. Definir cómo debe ver el usuario el sistema de archivos

- definir un archivo y sus atributos
- definir las operaciones permitidas sobre un archivo
- definir la estructura de directorios

2. Definir los algoritmos y estructuras de datos que deben crearse para establecer la correspondencia entre el sistema de archivos lógico y los dispositivos físicos donde se almacenan

Estructura del Sistema de Archivos

- Organización en niveles (capas)



- Por eficiencia, el SO mantiene una tabla indexada (por **descriptor de archivo**) de archivos abiertos

- **Bloque de control de archivo:** estructura con información de un archivo en uso

Métodos de Asignación de espacio: Contiguo

- Cada archivo ocupa un conjunto de bloques contiguos en disco
- **Ventajas**
 - Sencillo: solo necesita la localización de comienzo (nº de bloque) y la longitud
 - Buenos tanto el acceso secuencial como el directo
- **Desventajas**
 - No se conoce inicialmente el tamaño
 - Derroche de espacio (problema de la asignación dinámica → fragmentación externa)
 - Los archivos no pueden crecer, a no ser que se realice **compactación** → ineficiente

Contiguo (y II)

- **Asociación lógica a física**

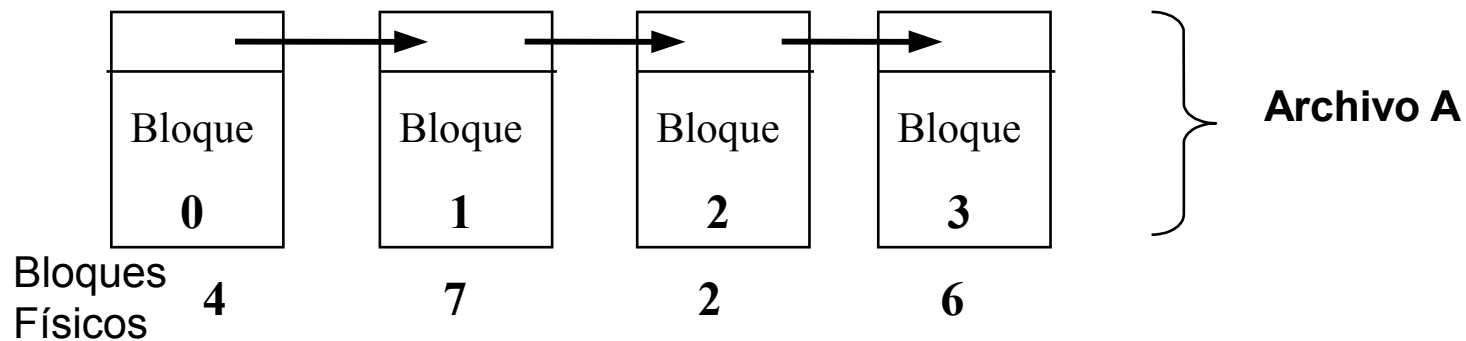
Supongamos que los bloques de disco son de 512 bytes:

Dirección lógica (DL)/512 \rightarrow C(cociente), R(resto)

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

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 cuando hay bloques de disco libres → no es necesario compactar
- Basta almacenar el puntero al primer bloque del archivo

No Contiguo - Enlazado (II)

- **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 (*overhead*)

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

Dirección lógica (DL)/511 \rightarrow C(cociente), R(resto)

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

No Contiguo - Enlazado (y III)

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

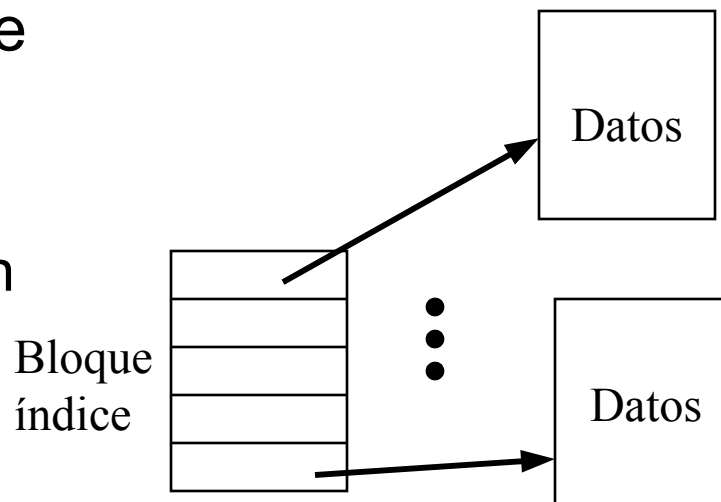
- Reserva una sección del disco al comienzo de la partición para la FAT
- Contiene una entrada por cada bloque del disco y está indexada por número de bloque de disco
- Simple y eficiente siempre que esté en caché
- Para localizar un bloque solo se necesita leer en la FAT → se optimiza el acceso directo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

Bloques FAT	
Físicos	
0	
1	
2	6
3	
4	7
5	
6	*
7	2
8	
9	
...	...

← A

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



No Contiguo - Indexado (II)

- **Desventajas**

- Posible desperdicio de espacio en los bloques índices
- Tamaño del bloque índice. Soluciones:

- (a) Bloques índices enlazados

- (b) Bloques índices multinivel

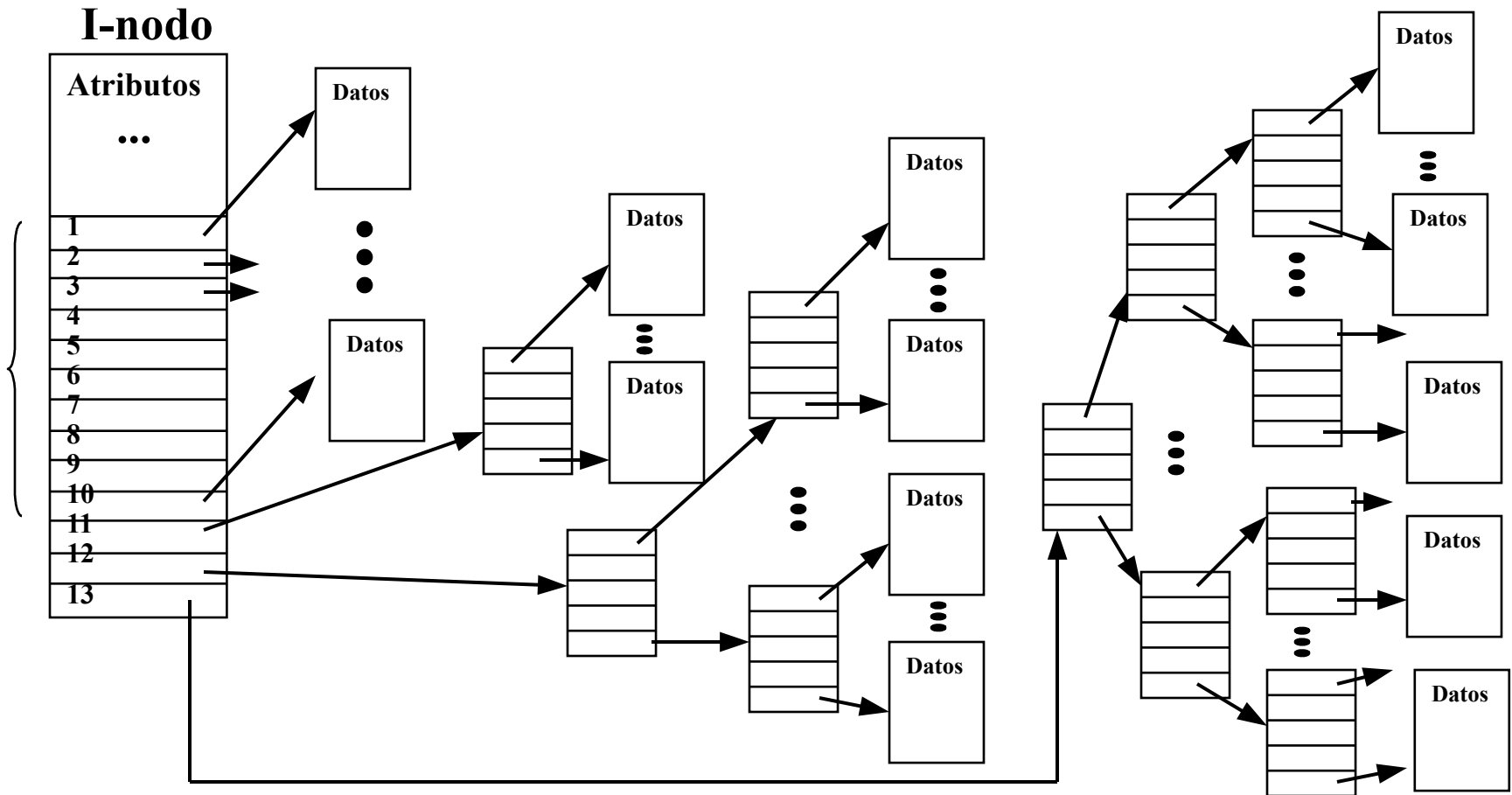
- **Problema**: acceso a disco necesario para recuperar la dirección del bloque para cada nivel de indexación

- **Solución**: mantener algunos bloques índices en memoria principal

- (c) Esquema combinado (Unix)

No Contiguo - Indexado (y III)

- Unix (s5fs)



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
- A pesar de su nombre, la lista de espacio libre tiene diferentes implementaciones:

1. Mapa o Vector de Bits

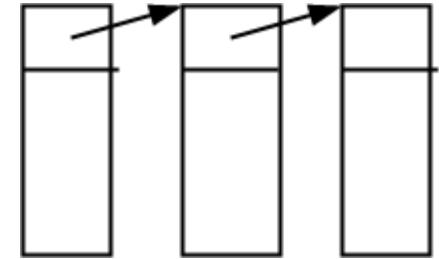
10010001
11111101
11100000
11111110
00000000
11100011
11100000

- 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

Gestión de espacio libre (y II)

2. Lista enlazada

- Enlaza todos los bloques libres del disco, guarda un puntero al primer bloque en un lugar concreto

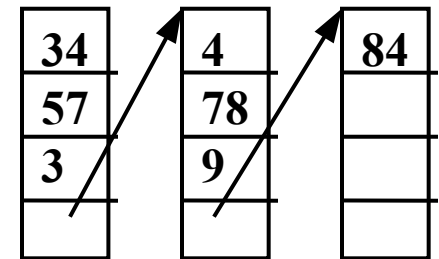


- No derrocha espacio

- Relativamente ineficiente → No es normal atravesar bloques vacíos

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. Cuenta

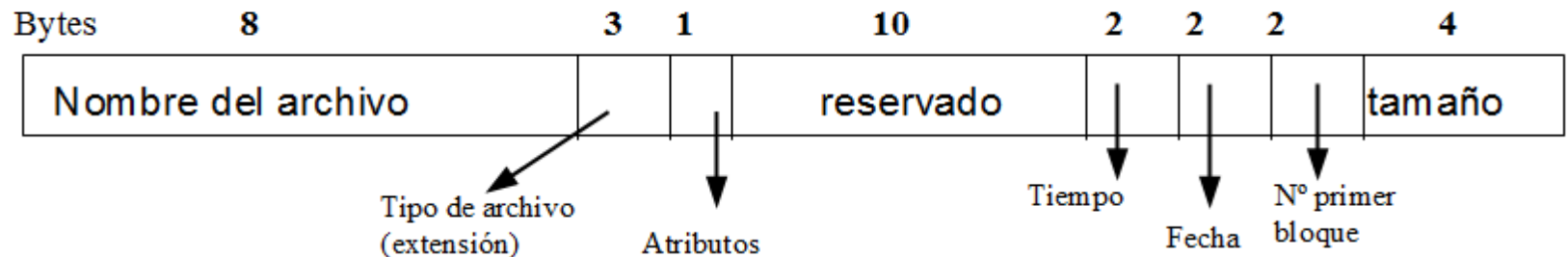
- Cada entrada de la lista: una dirección de bloque libre y un contador del nº de bloques libres que le sigue

Implementación de Directorios

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

(a) Nombre de Archivo + Atributos + Dirección de los bloques de datos (DOS)

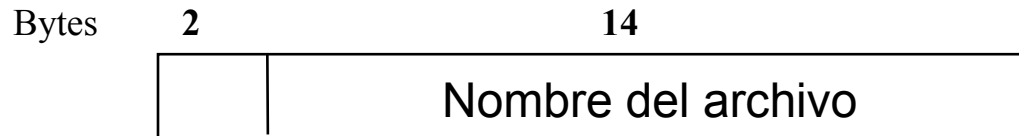
Entrada de directorio de MS-DOS



Implementación de Directorios (y II)

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

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 Directorios (y III)

- Implementación de archivos compartidos (o enlace):

1. Enlaces **simbólicos**

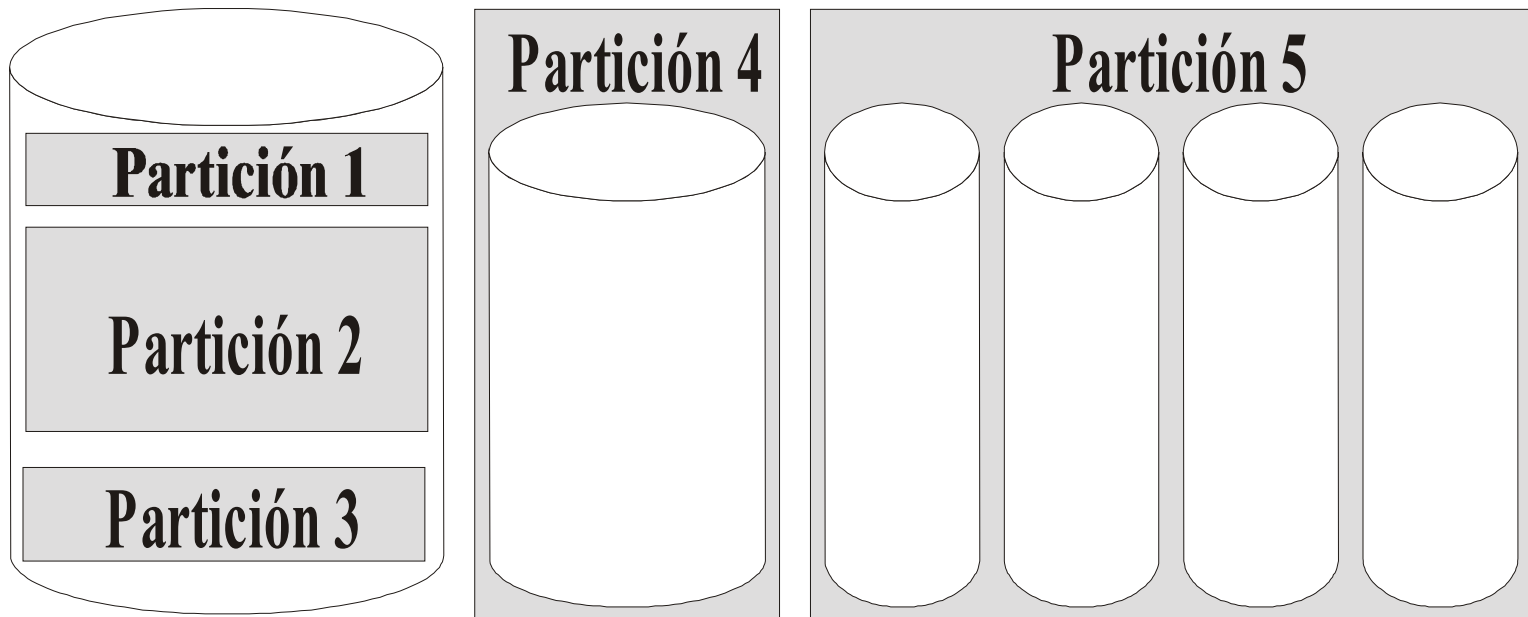
- Se crea una nueva entrada en el directorio, se indica que es de tipo *enlace* y se almacena el camino de acceso absoluto o relativo del archivo al cual se va a enlazar
- Se puede usar en entornos distribuidos
- Gran número de accesos a disco

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
- Problema al borrar los enlaces: solución → **Contador de enlaces**

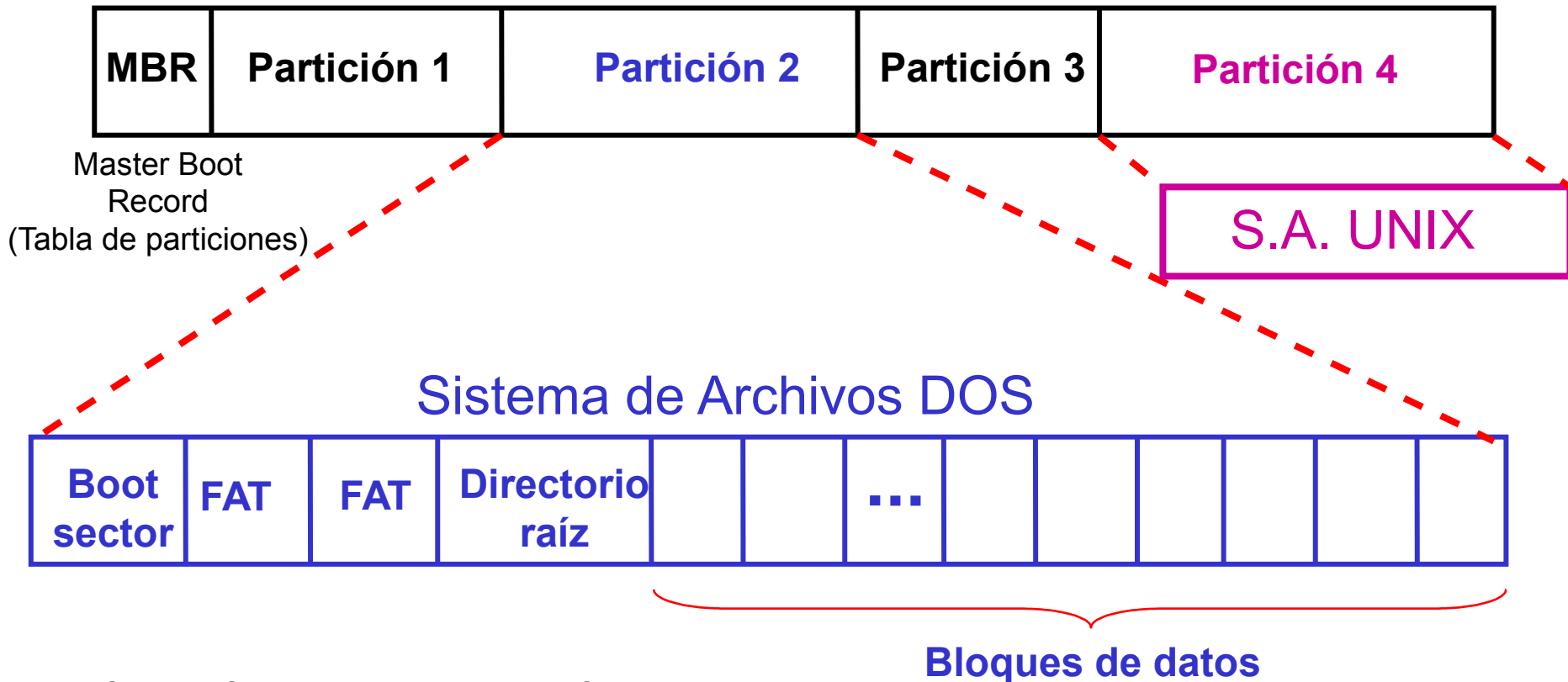
Distribución del sistema de archivos

- Los sistemas de archivos se almacenan en discos que pueden dividirse en una o más particiones.
- **Partición** del disco



Distribución del sistema de archivos (y II)

- Ejemplo de organización en particiones



Distribución del sistema de archivos (y III)

- Formateo del disco:
 - **Físico**: pone los sectores (cabecera y código de corrección de errores) por pista
 - **Lógico**: escribe la información que el SO necesita para conocer y mantener los contenidos del disco (un directorio inicial vacío, FAT, lista de espacio libre, ...)
- Bloque de arranque para **inicializar** el sistema localizado por *bootstrap*
- Métodos necesarios para detectar y manejar bloques dañados

Recuperación

- Como los archivos y directorios se mantienen tanto en MP como en disco, el sistema debe asegurar que un fallo no genere pérdida o inconsistencia de datos

- Distintas formas:

1. **Comprobador de consistencia:**

- Compara los datos de la estructura de directorios con los bloques de datos en disco y trata cualquier inconsistencia
- Más fácil en listas enlazadas que con bloques índices

2. Usar programas del sistema para realizar **copias de seguridad** (*backup*) de los datos de disco a otros dispositivos y de recuperación de los archivos perdidos

4.3. Implementación de la gestión de archivos en Linux

- **i-nodo**: representación interna de un archivo
- Un archivo tiene asociado un único i-nodo, aunque éste puede tener distintos nombres (**enlaces**)
- 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 hasta que se cierre

Sistema de Archivos

- SO implementa al menos un sistema de archivos (SA) estándar o nativo
- En Linux: ext2, ext3 y ext4
- Necesidad de dar soporte a otros SA distintos (FAT, s5fs, etc.): el kernel incluye una capa entre los procesos de usuario (o la biblioteca estándar) y la implementación del SA --> **Sistema de Archivos Virtual (VFS – Virtual File System)**.

Sistema de Archivos (y II)

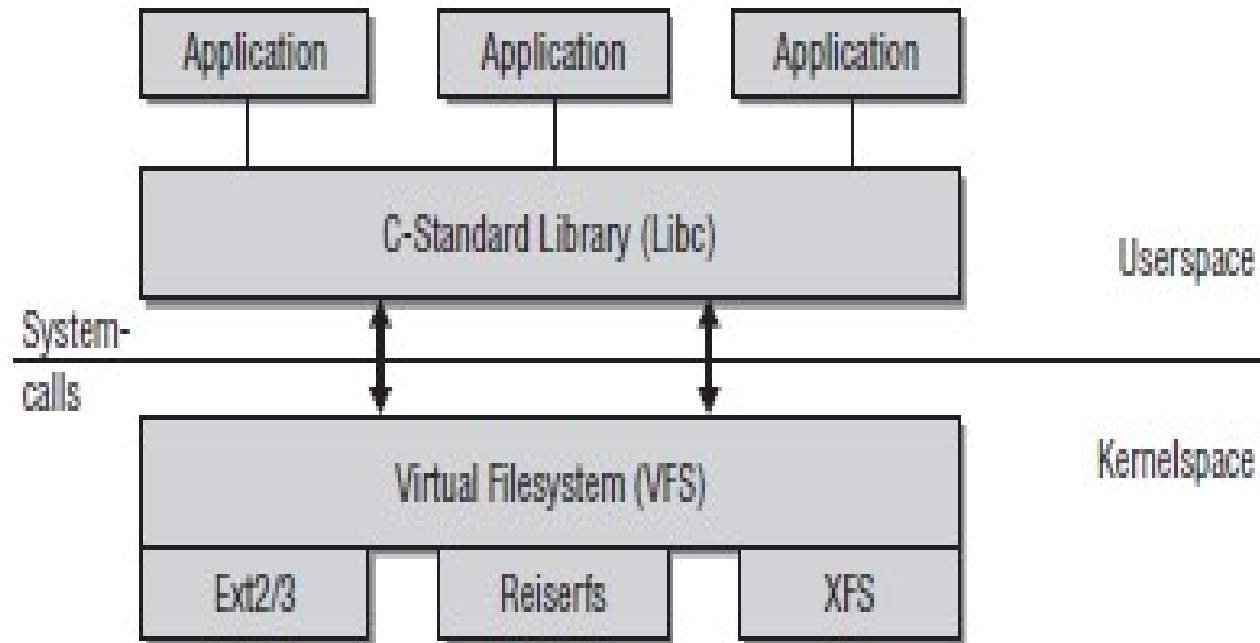


Figure 8-1: VFS layer for filesystem abstraction.

Sistema de Archivos (y III)

- Linux abstrae el acceso a los archivos y a los SA mediante una interfaz virtual que lo hace posible
- Flujo de operaciones/datos por las distintas partes del sistema en una llamada al sistema de escritura (`write`)

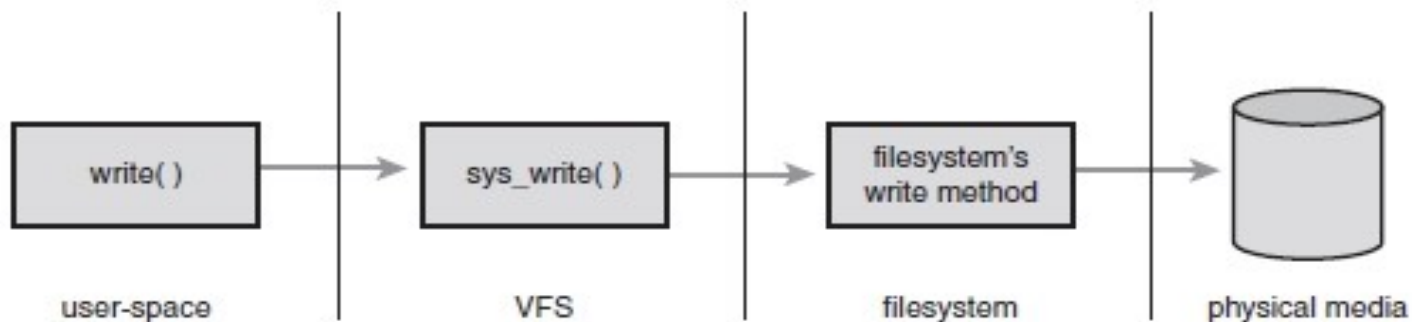


Figure 13.2 The flow of data from user-space issuing a `write()` call, through the VFS's generic system call, into the filesystem's specific write method, and finally arriving at the physical media.

Tipos de Sistemas de Archivos

Tres clases generales de SA:

1. **SA basados en disco** (Disk-based filesystems): forma clásica de almacenar archivos en medios no volátiles: Ext2/3, Reiserfs, FAT, e iso9660.
2. **SA Virtuales** (Virtual filesystems): generados por el kernel y constituyen una forma simple para permitir la comunicación entre los programas y los usuarios. Ejemplo es el SA proc. No requieren espacio de almacenamiento en ningún dispositivo hardware (la información está en MP).
3. **SA de Red** (Network filesystems): permiten acceder a los datos a través de la red.

Modelo de archivo común

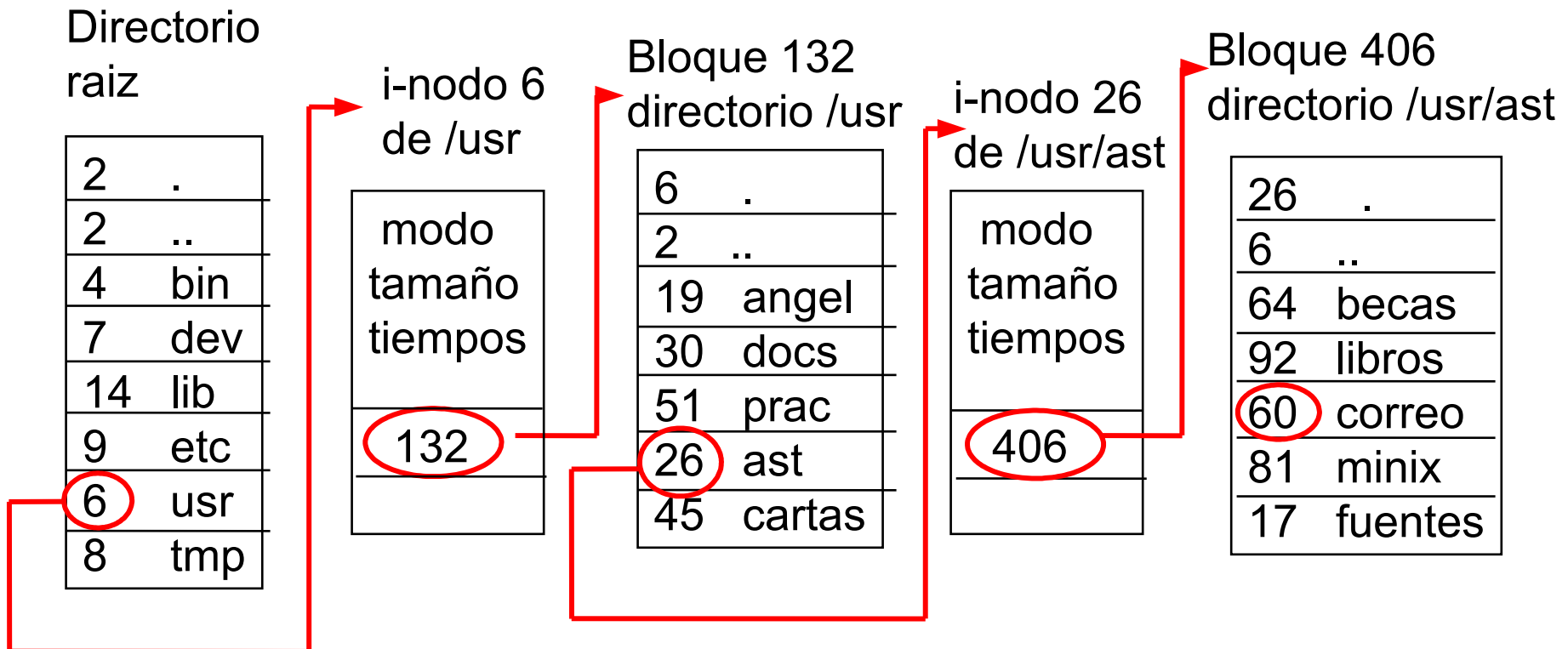
- Para un programa de usuario, un archivo se identifica por un **descriptor de archivo** (nº entero usado como índice en la **tabla de descriptores** que identifica el archivo en las operaciones relacionadas con él).
- El descriptor lo asigna el kernel cuando se abre el archivo y es válido sólo dentro de un proceso.
- Dos procesos diferentes pueden usar el mismo descriptor pero no apuntan al mismo archivo.
- Un inodo es la estructura asociada a cada archivo y directorio y contiene sus metadatos.

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**

Acceso a un archivo

- Ejemplo de qué hace el kernel para acceder a `/usr/ast/correo`



Estructura VFS

- VFS es orientado a objetos, consta de dos componentes, archivos y SA, que necesita gestionar y abstraer.
- Se representa a un archivo y a un SA con una familia de estructuras de datos hechas en C. Existen 4 tipos de objetos primarios del VFS:
 - **objeto superblock**: representa a un SA montado
 - **objeto inode**: representa a un archivo (cualquier tipo)
 - **objeto dentry**: representa a una entrada de un directorio
 - **objeto file**: representa a un archivo abierto y es una estructura por proceso. Las anteriores son de sistema.

Estructura VFS (y II)

Cada uno de estos objetos primarios tiene un vector de “operations”. Estas funciones describen los métodos que el kernel invoca sobre los objetos primarios.

- objeto `super_operations`: métodos que el kernel puede invocar sobre un SA concreto, ej: `write_inodo()` y `sync_fs()`.
- objeto `inode_operations`: métodos que el kernel puede invocar sobre un archivo concreto, ej: `create()` y `link()`.
- objeto `dentry_operations`: métodos que el kernel puede invocar sobre una entrada de directorio, ej: `d_compare()` y `d_delete()`.
- objeto `file_operations`: métodos que un proceso puede invocar sobre un archivo abierto como `read()` y `write()`.

Estructura VFS (y III)

- Cada SA registrado está representado por una estructura `file_system_type`. Este objeto describe el SA y sus capacidades.
- Cada punto de montaje está representado por la estructura `vfsmount` que contiene información acerca del punto de montaje, tal como sus localizaciones y flags de montaje.
- Finalmente, existen dos estructuras por proceso que describen el SA y los archivos asociados con un proceso: `fs_struct` y `file_struct`.

Estructuras de datos VFS

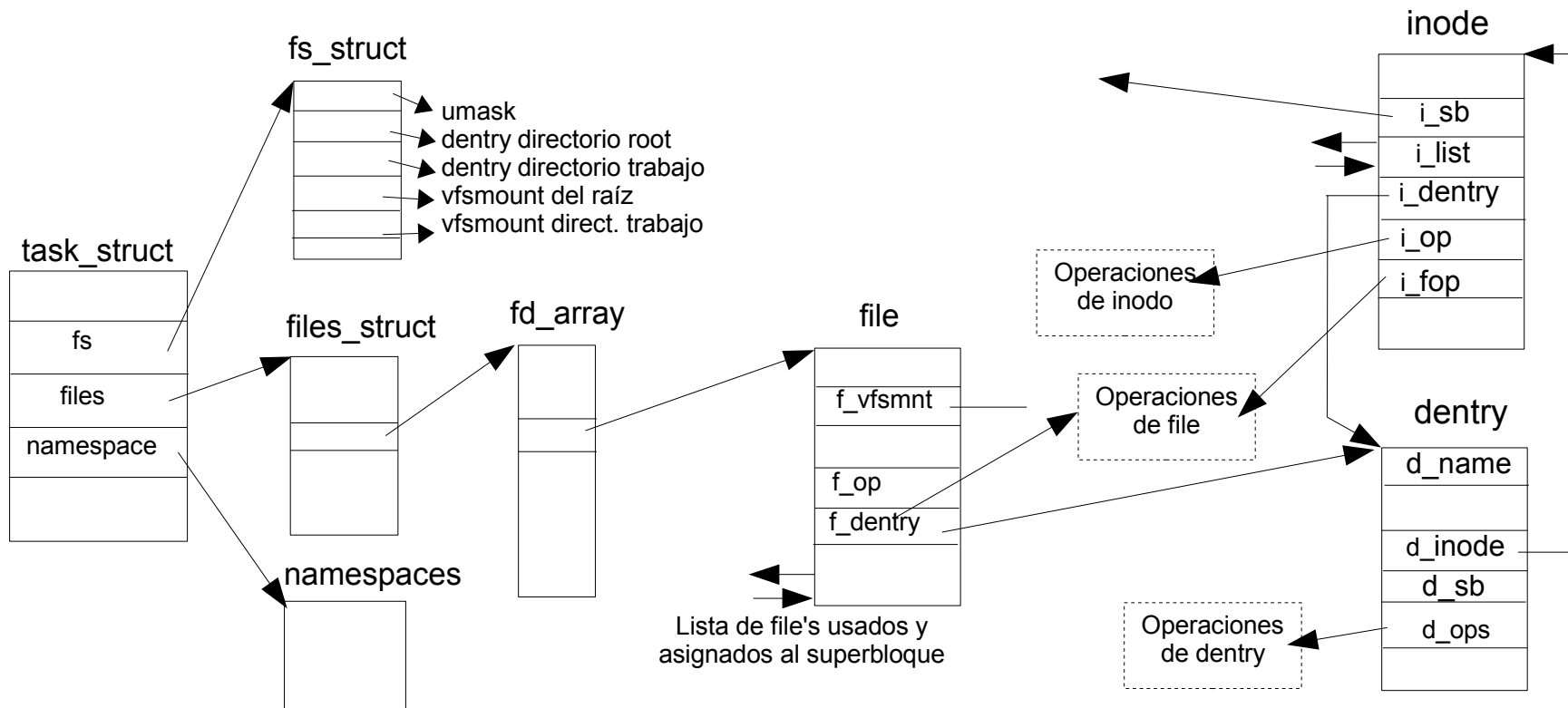


Figura: Estructuras de datos VFS en Linux 2.6.x de José Antonio Gómez

Sistema de archivos Ext2

- Divide el disco duro en un conjunto de bloques de igual tamaño donde se almacenan los datos de los archivos y de administración.
- El elemento central de Ext2 es el “grupo de bloques” (block group).

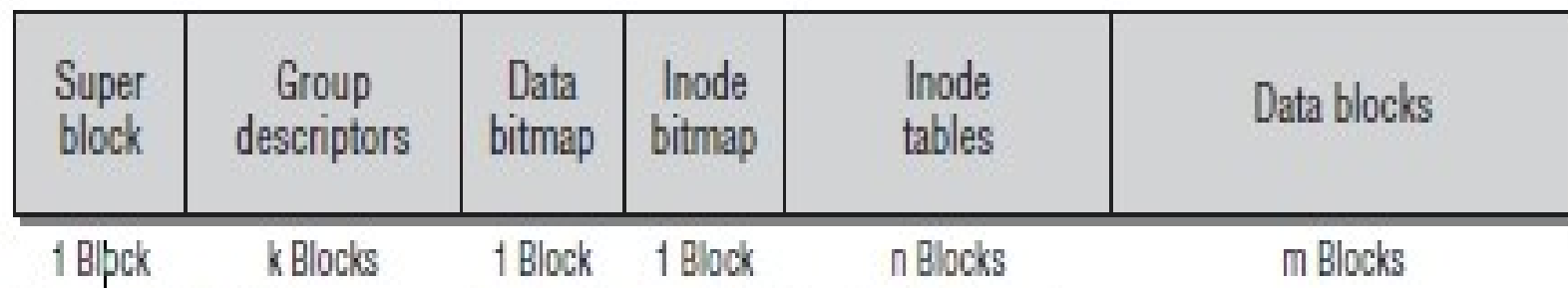


Figure 9-2: Block group of the Second Extended Filesystem.

Sistema de archivos Ext2 (y II)

- Cada SA consta de un gran número de grupos de bloques secuenciales
- Boot sector (Boot block): zona del disco duro cuyo contenido se carga automáticamente por la BIOS y se ejecuta cuando el sistema arranca
- Cuando se usa un SA (se monta), el superbloque, sus datos, se almacenan en MP.



Figure 9-3: Boot sector and block groups on a hard disk.

Descripción de un Grupo de Bloques

- **Superbloque (Superblock)**: estructura central para almacenar meta-información del SA.
- **Descriptores de grupo (Group descriptors)**: contienen información que refleja el estado de los grupos de bloques individuales del SA., por ejemplo, el número de bloques libres e inodos libres.
- **Mapa de bits de bloques de datos y de inodos (Data bitmap, Inode bitmap)**: contienen un bit por bloque de datos y por inodo respectivamente para indicar si están libres o no.
- **Tabla de inodos (Inode tables)**: contiene todos los inodos del grupo de bloques. Cada inodo mantiene los metadatos asociados con un archivo o directorio del SA.

Estructuras en el dispositivo

- Estructura para el superbloque:

struct ext2_super_block

- Estructura para el descriptor de grupo:

struct ext2_group_desc

- Estructura para el inodo:

struct ext2_inode

- Estructura para directorios y archivos:

struct ext2_dir_entry_2

Información sobre los bloques de un archivo

- Linux usa un método de asignación de bloques no contiguo y cada bloque de un SA se identifica por un número.
- En el inodo se almacenan 12 direcciones directas a BD, un primer nivel de indexación, un segundo nivel de indexación y un tercer nivel de indexación (si es necesario)

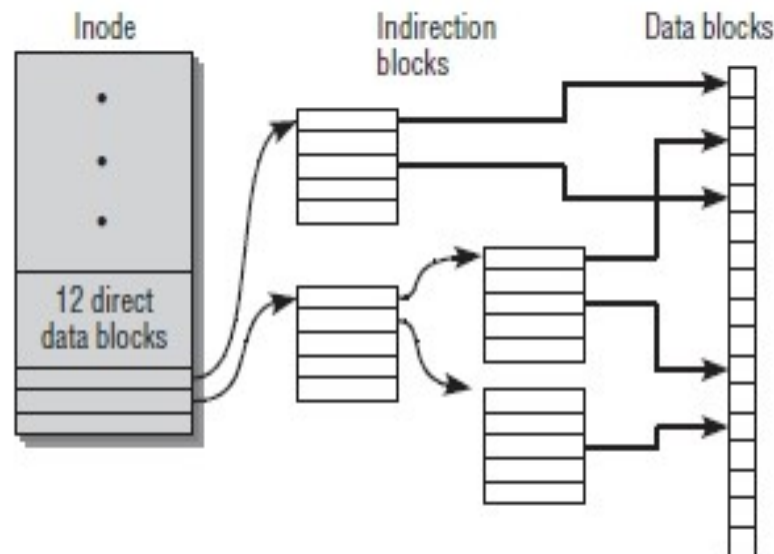


Figure 9-4: Simple and double indirection.

Montaje y desmontaje de un sistema de archivos

- La llamada al sistema **mount** conecta un sistema de archivos al sistema de archivos existente y la llamada **umount** lo desconecta

```
mount (<camino_especial>, <camino_directorio>, <opciones>)
```

- El núcleo tiene una tabla de montaje con una entrada por cada sistema de archivos montado:
 - número de dispositivo que identifica el SA montado
 - puntero a un buffer que contiene una copia del superbloque
 - puntero al i-nodo raíz del SA montado
 - puntero al i-nodo del directorio punto de montaje

Relación entre estructuras de datos

