Tema 4 – Gestión de Archivos

- Interfaz de los Sistemas de Archivos
- Aspectos de diseño del Sistema de Archivos
- Implementación de la gestión de archivos en Linux

Sistemas Operativos
Alejandro J. León Salas, 2019
Lenguajes y Sistemas Informáticos

Objetivos

- Comprender el concepto de archivo, directorio y estructura de directorios.
- Conocer las distintas funciones que se realizan sobre un sistema de archivos y cómo se puede estructurar esta funcionalidad.
- Conocer las ventajas e inconvenientes de distintos **métodos de asignación de espacio de 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.

Bibliografía

- [Sta2005] W. Stallings, Sistemas Operativos. Aspectos Internos y Principios de Diseño (5/e), Prentice Hall, 2005.
- [Lov2010] R. Love, *Linux Kernel Development (3/e)*, Addison-Wesley Professional, 2010.
- [Mau2008] W. Mauerer, Professional Linux Kernel Architecture, Wiley, 2008.

Contenidos

- Interfaz de los Sistemas de Archivos
- Aspectos de diseño del Sistema de Archivos
- Implementación de la gestión de archivos en Linux

Interfaz de los Sistemas de Archivos

- Concepto de archivo.
- Estructura de directorios.
- Protección.
- Semánticas de consistencia de datos de archivo.
- Funcionalidad básica de SA.

Concepto de archivo

- Colección de información relacionada y almacenada en un dispositivo de almacenamiento secundario.
- Espacio de direcciones lógicas (relativas) contiguas.
- Estructura interna (lógica)
  - **Secuencia de bytes**: el tipo de archivo determina su estructura, ej. texto se interpreta como caracteres, líneas y páginas.
  - Secuencia de registros de longitud fija.
  - Secuencia de registros de longitud variable.
- Tipos de archivos: regulares, directorios, de dispositivo (bloques y caracteres), FIFO, socket, enlace simbólico.
- Formas de acceso: Secuencial y aleatorio.

Concepto de archivo

Atributos (metadatos) de archivo:

- Nombre. Información para usuario de aplicación.
- Tipo. Los SOs soportan diferentes tipos de archivo (no confundir con diferentes tipos de formatos de archivo).
- Localización. Información sobre su localización en el dispositivo de almacenamiento secundario.
- Tamaño (*size*). Tamaño actual del archivo.
- Protección (*permissions)*. Permite controlar las operaciones permitidas sobre el archivo. (Junto con identificación permite determinar quién puede leer, escribir y ejecutar el archivo).
- Tiempos y fechas e identificación de usuarios y/o grupos. Necesario para protección, seguridad y monitorización.

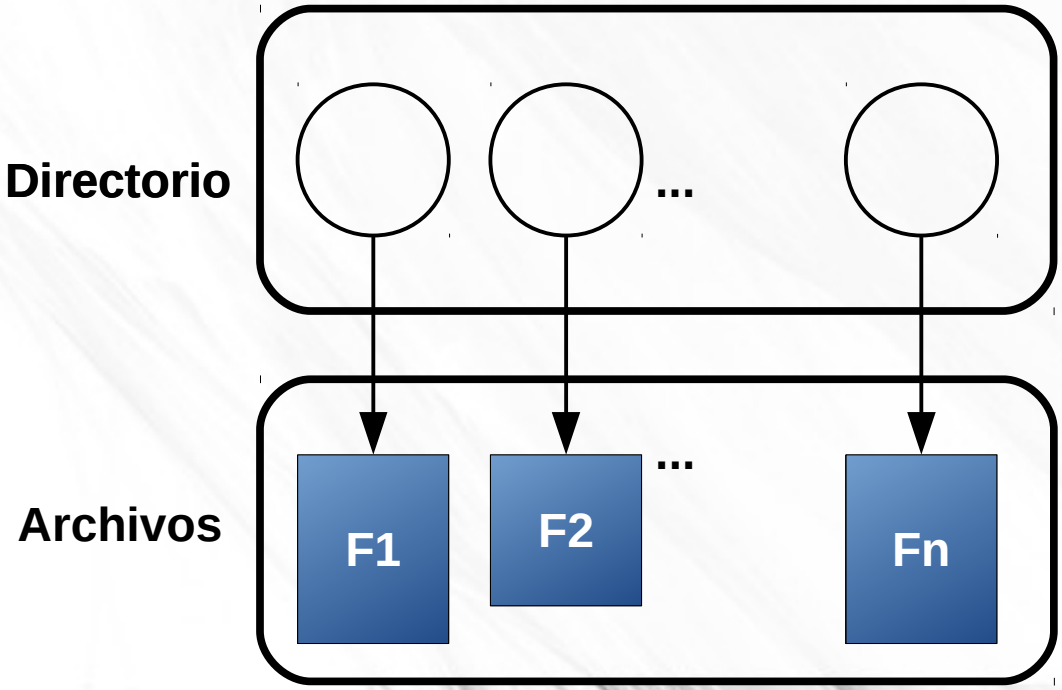
Concepto de archivo

Operaciones sobre archivos

- Gestión:
  - Crear: **`open(),link(),symlink(),mknod(),mkfifo()`**
  - Borrar: **`unlink()`**
  - Renombrar: **`rename()`**
  - Copiar: **`rename()`**
  - Establecer y obtener atributos: **`stat(),chmod(), chown(),open(),read(),write()`**,...
- Procesamiento:
  - Abrir y Cerrar: **`open()`** y **`close()`**.
  - Leer y Escribir (modificar, insertar, borrar información): **`read()`** y **`write()`**.
  - Cambiar el puntero de lectura/escritura: **`lseek()`**.

Estructura de directorios

- Colección de nodos conteniendo información acerca de todos los archivos (Organización de archivos).
- Tanto la estructura de directorios como los archivos residen en el almacenamiento secundario. Los archivos de tipo directorio contienen y mantienen la organización del sistema de archivos.



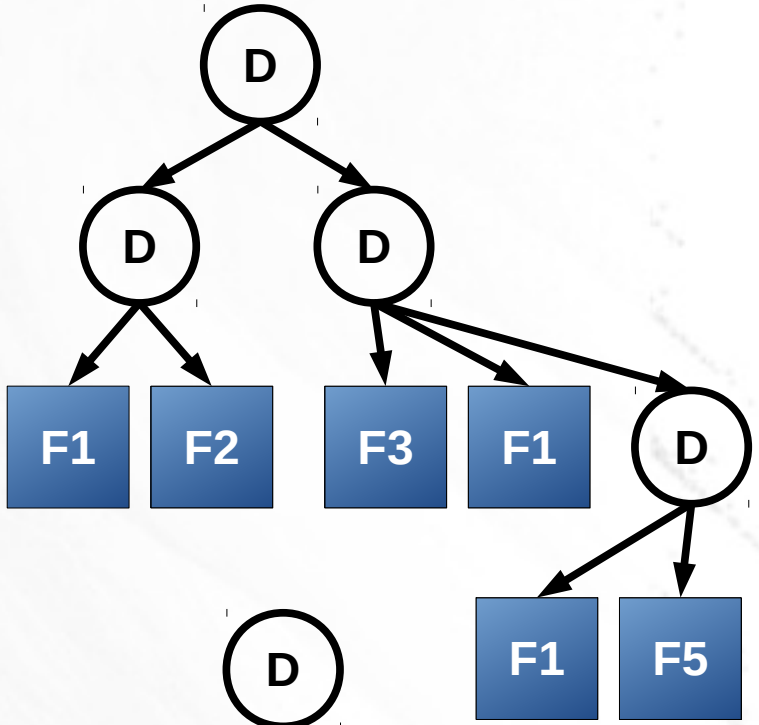
Estructura de directorios

- La organización (lógica) de los directorios debe proporcionar:
- **Eficiencia**. Localización rápida de un archivo en la estructura de directorios.
- **Denominación** adecuada a las necesidades de los usuarios:
  - Dos usuarios pueden tener el mismo nombre para diferentes archivos.
  - El mismo archivo (metadatos) puede tener varios nombres: enlaces *hard* y *soft*.
- **Agrupación**. Permitir agrupar los archivos de forma lógica según sus propiedades o lo que considere el usuario. (Ej. todos los programas en C).

Estructura de directorios: Diseño

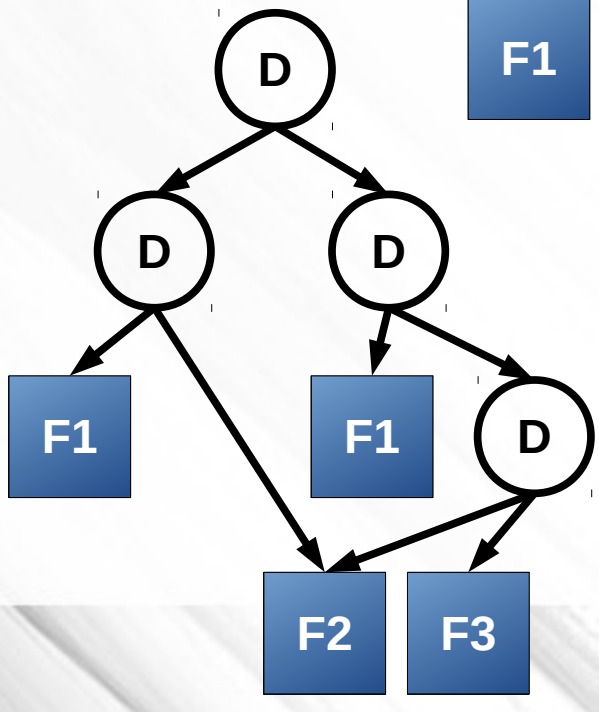
Árbol.

- Necesidad de búsquedas eficientes.
- Posibilita la agrupación de archivos.
- Permitir nombres de camino absoluto y relativo (directorio de trabajo actual).



Grafo.

- Añade compartición de directorios y archivos, por lo que requiere algunas medidas de seguridad (ej. evitar bucles/lazos en el grafo).



Protección

- Consiste en proporcionar un acceso controlado a los archivos: Quién puede acceder y qué operaciones puede hacer.
- Tipos de acceso sobre archivos y directorios:
  - Sobre archivos: leer, escribir, ejecutar, añadir (requiere poder escribir), borrar.
  - Sobre directorios: leer, escribir (crear nuevos archivos/directorios), borrar y cambiar a un directorio.

Protección: Listas de acceso

- Solución para protección: hacer el acceso dependiente del identificativo del usuario, `UID`.
- Las **listas de acceso** de usuarios individuales tiene el problema del tamaño. Solucionado con “clases” de usuarios:
  - Propietario, UID del propietario del archivo.
  - Grupo, GID grupo del archivo.
  - Público, los UID que no pertenezcan a las dos primeras clases.
- Propuesta alternativa: Asociar un *password* con el archivo que presenta varios problemas:
  - Recordar todos
  - Si solo se asocia un password al SA entonces acceso total o ningún acceso.

Semánticas de consistencia

- Especifican cuándo las modificaciones de los datos de un archivo compartido se observan por otros usuarios.
- Semántica de Unix
  - La escritura se observa tras finalizar la operación.
  - Varios niveles de compartición.
- Semánticas de sesión (Sistema de archivos de Andrew)
  - La escritura en un archivo no se observa por el resto tras la operación de escritura, solamente se actualiza al cerrar sesión.
- Archivos inmutables.
  - Cuando un archivo se declara como compartido, no se puede modificar.

Funcionalidad básica del SA

- Tener conocimiento de todos los archivos del sistema de archivos.
- 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 (offset o puntero de R/W) en direcciones físicas del disco usando *Logical Block Addressing* (LBA).
  - Los usuarios especifican las partes que quieren leer/escribir en términos de direcciones lógicas relativas en el archivo.

Contenidos

- Interfaz de los Sistemas de Archivos
- Aspectos de diseño del Sistema de Archivos
- Implementación de la gestión de archivos en Linux

Aspectos de diseño del Sistema de Archivos

- Estructura del sistema de archivos.
- Métodos de asignación de espacio en disco.
- Gestión de espacio libre.
- Implementación de directorios.
- Distribución del sistema de archivos.
- Recuperación de información.

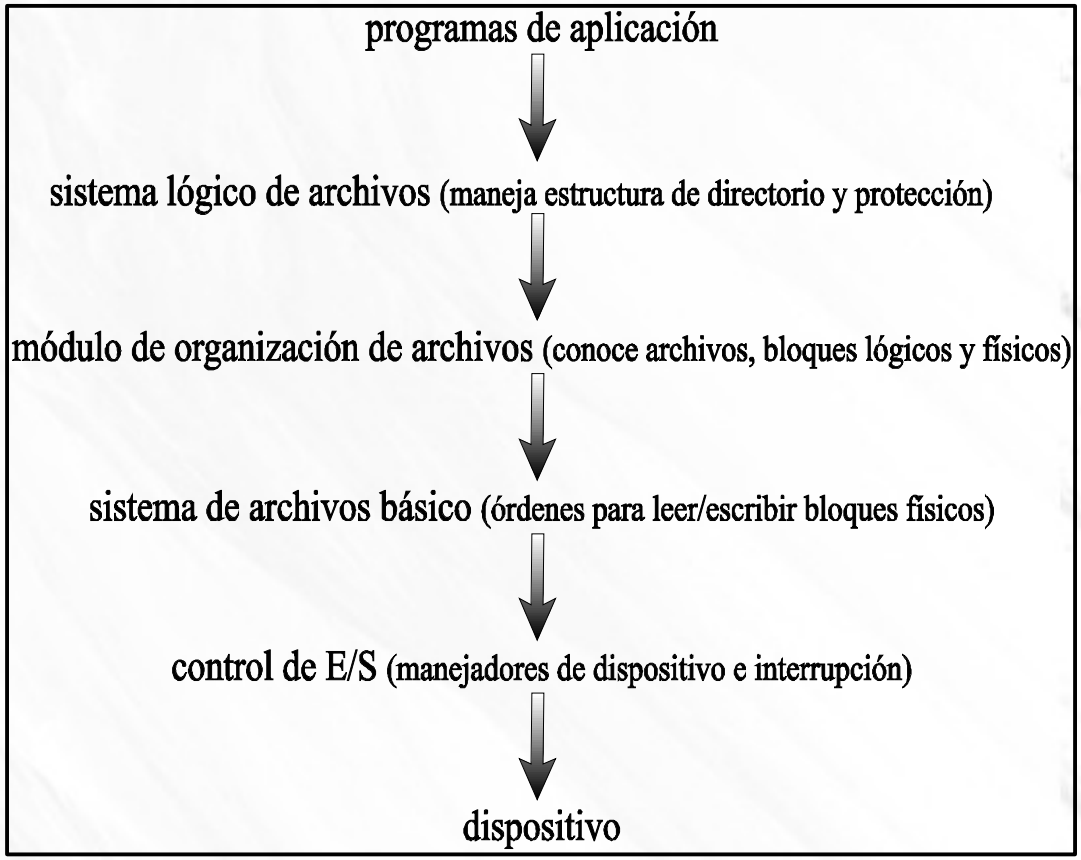
Diseño del sistema de archivos

- Un sistema de archivos posee dos ámbitos de diseño diferentes:
- Definir cómo debe ver el usuario el sistema de archivos, e.d. que abstracciones proporciona:
  - Definir un archivo y sus atributos.
  - Definir las operaciones permitidas sobre un archivo.
  - Definir la estructura de directorios.
- 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.

Diseño del sistema de archivos

- Por eficiencia, el SO mantiene una tabla indexada (por descriptor de archivo) de archivos abiertos.
- Información de sesión de apertura sobre un archivo.
- Bloque de control de archivo: estructura con información de un archivo en uso.

Organización en capas



Métodos de asignación de espacio: Contiguo

- Los datos de un archivo se almacenan en un conjunto de bloques contiguos en disco.

Ventajas:

- Metadatos de localización: (nº de primer bloque asignado, número de bloques asignados).
- Bueno tanto el acceso secuencial como el directo.

Desventajas:

- No se conoce inicialmente el tamaño que podrá requerir el archivo. Si se asigna un tamaño de bloques prefijado podemos tener fragmentación externa.
- La asignación dinámica de espacio a archivos agravará la fragmentación externa con el tiempo.
- Los archivos no pueden crecer, a no ser que se realice compactación del espacio de disco.

Métodos de asignación de espacio: Contiguo

- Correpondencia de dirección lógica en el archivo a dirección física de bloque de disco.

```
Dir_logica/tama_bloque => cociente(C), resto(R)
```

Y el acceso a los datos:

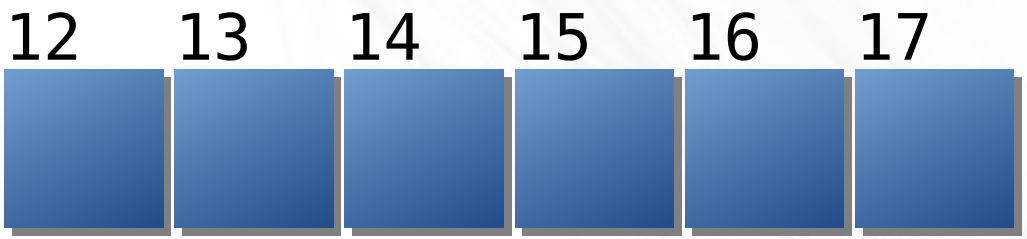
```
Nº de bloque = Nº primer bloque asignado + C

Desplazamiento dentro de bloque = R

Ej.: Metadatos de localización: (12,6)

read(fd,buf,1) y R/W = 1028; 1028/512 => C=2, R=4

E/S de bloque 12+2=14, buf=dir_base_bloque + 4
```



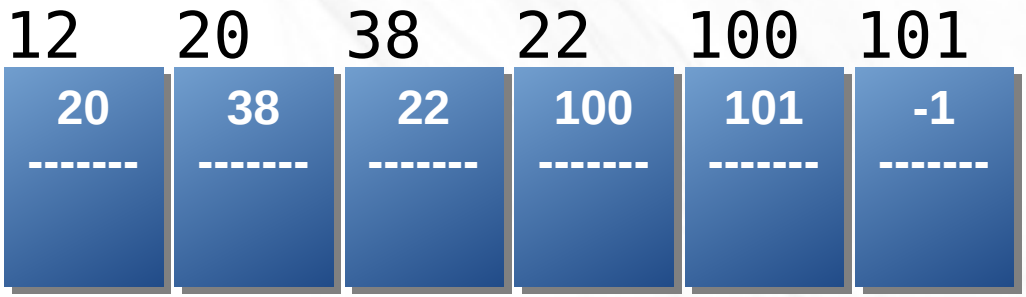
Métodos de asignación de espacio: Enlazado

- Los datos de un archivo se almacenan en bloques de disco que no tienen que ser consecutivos. Lista enlazada de bloques.
- Metadatos de localización: Nº primer bloque.

Ventajas:

- Evita la fragmentación externa.
- Resuelve el crecimiento dinámico del archivo, evitando necesidad de compactación.

```
Localización = 12
```



Métodos de asignación de espacio: Enlazado

Desventajas:

- El acceso directo no es efectivo (el secuencial si).
- Espacio desperdiciado por bloque: nº siguiente bloque, solución: asignar grupos de bloques (*clusters*)
- Problema de seguridad: bloque dañado => pérdida de la lista. Solución: lista doblemente enlazada.
- Correpondencia de dirección lógica en el archivo a dirección física de bloque de disco.

```
Dir_logica/(tama_bloque – tamaño nº_sig_bloque)  =>
cociente(C), resto(R)
```

Y el acceso a los datos:

```
Nº de bloque = C-ésimo bloque de la lista.

Desplazamiento dentro de bloque = tamaño nº_sig_bloque+R
```

Métodos de asignación de espacio: Enlazado

Ej: Metadatos de localización: 12

Espacio para Nº_sig_bloque: 1 Byte

read(fd,buf,1) y R/W = 1028; 1028/(512-1) => C=2, R=6

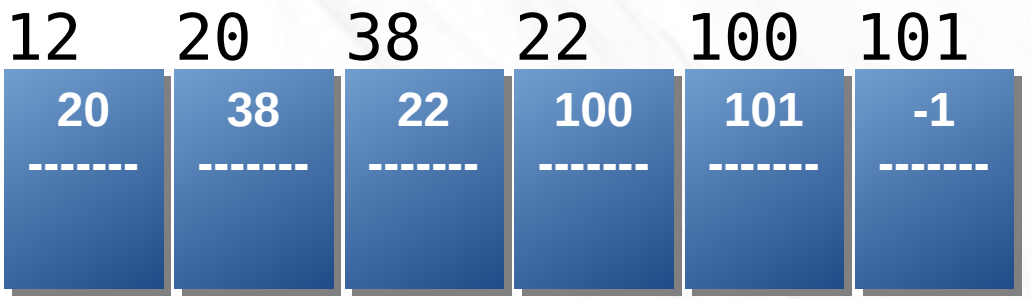
Bloque a acceder: 2-ésimo

Desplazamiento dentro del bloque 2-ésimo: 1 + 6 = 7

E/S del bloque 12 (bloque 0-ésimo)

E/S del bloque 20 (bloque 1-ésimo)

E/S del bloque 38 (bloque 2-ésimo)



Métodos de asignación de espacio: Enlazado, variante FAT

- Variación del método enlazado inicialmente desarrollada en OS/2, DOS y Windows: *File Allocation Table* (FAT).
- Reserva una sección del disco (bloques consecutivos) al comienzo de la partición para almacenar la FAT.
- Contiene una entrada por cada bloque del disco y está indexada por número de bloque de disco.
- Para localizar un bloque solo se necesita leer en la FAT, por lo que se optimiza el acceso directo, siempre que se mantenga en memoria principal.
- Problema: pérdida de enlaces si bloque dañado, solución con dos copias de la FAT en disco.

Métodos de asignación de espacio: Enlazado, variante FAT

- Tabla FAT en memoria principal y en memoria secundaria.

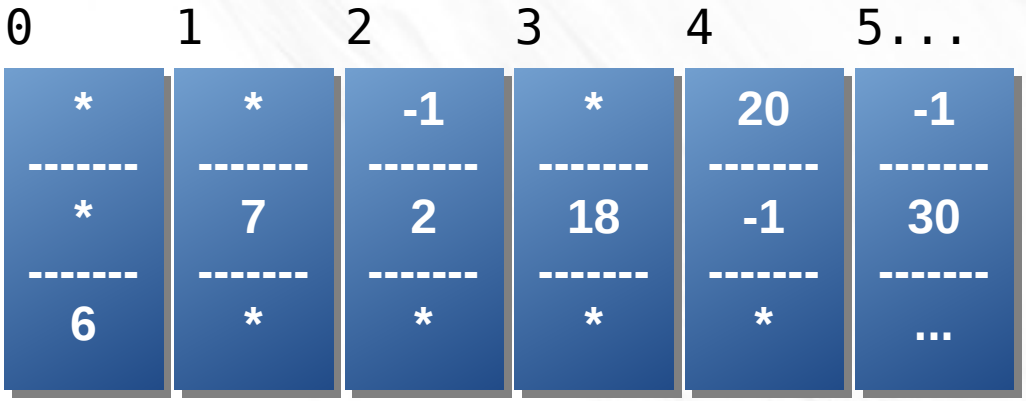
Ej.: Metadatos de localización: 4

read(fd,buf,1) y R/W = 1028; 1028/512 => C=2, R=4

Bloque a acceder: 2-ésimo => seguir punteros en FAT: 0-ésimo=4, 1-ésimo=7, 2-ésimo=2

Desplazamiento dentro del bloque 2-ésimo: 4

E/S del bloque 2, buf=dir_base_bloque + 4



0	*
1	*
2	6
3	*
4	7
5	*
6	-1
7	2
8	*
9	*
10	18
11	*
12	20
13	-1
...	...

Métodos de asignación de espacio: Indexado

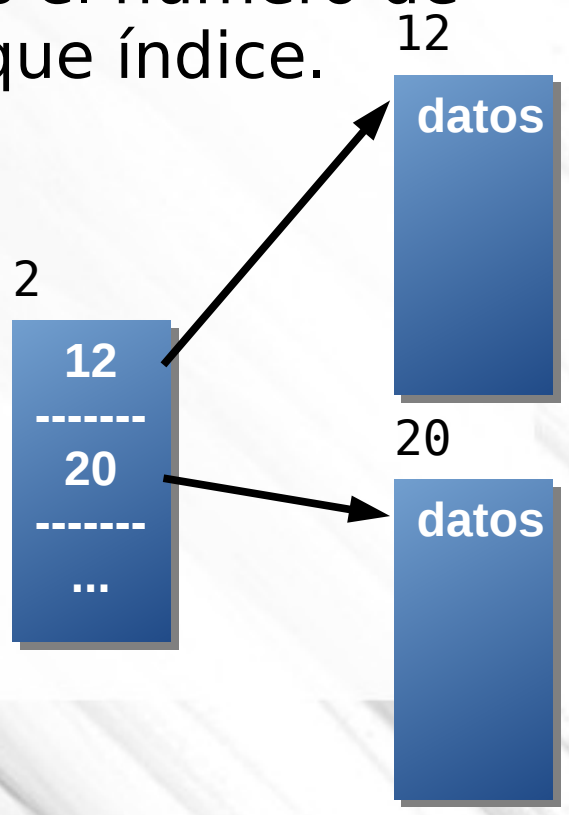
- Los números de bloque asignados a un archivo están juntos en una localización concreta: **bloque índice**.
- El metadato de localización indica el número de bloque índice y cada archivo tiene asociado su propio bloque índice.
- Para leer el i-ésimo bloque buscamos el número de bloque de la i-ésima entrada del bloque índice.

Ventajas:

- Buen acceso directo.
- No produce fragmentación externa.

```
Ej: Metadatos de localización: 2

(bloque índice)
```



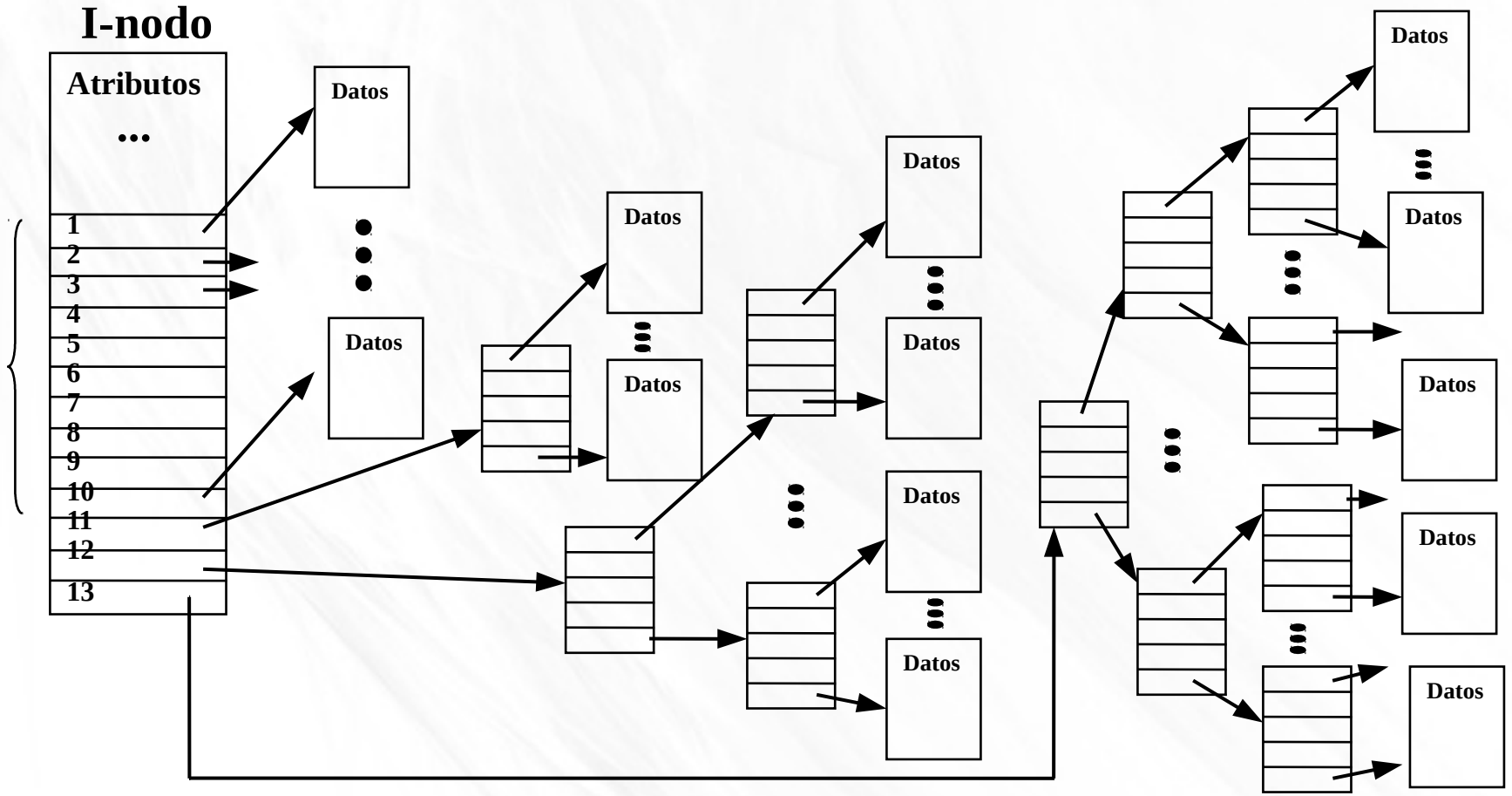
Métodos de asignación de espacio: Indexado

Desventajas:

- Posible desperdicio de espacio en los bloques índices.
- Tamaño del bloque índice. Soluciones:
  - Bloques índice enlazados.
  - Bloques índice multinivel.
    - Problema de los bloques índice multinivel: Acceso a disco necesario para recuperar la dirección del bloque para cada nivel de indexación.
    - Solución: Mantener algunos bloques índices en memoria principal.
  - Esquema combinado (Unix)

Métodos de asignación de espacio: Indexado

- UNIX (s5fs de SVr2)



Gestión de espacio libre

- El sistema mantiene una lista de los bloques que están libres: lista de bloques libres.
- La FAT no necesita ningún método adicional.
- A pesar de su nombre, la lista de bloques libres tiene diferentes implementaciones:

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.
- Fácil tener archivos en bloques contiguos.
- Ineficiente si no se mantiene en memoria principal.

Gestión de espacio libre

Lista enlazada:

- Enlaza todos los bloques libres del disco, y guarda el número del primer bloque libre.
- No derrocha espacio.
- Relativamente ineficiente porque proporciona bloques y el acceso directo no es necesario.

Lista enlazada con agrupación:

- Cada bloque de la lista almacena n-1 direcciones de bloques libres, por tanto obtener muchas bloques libres es rápido.

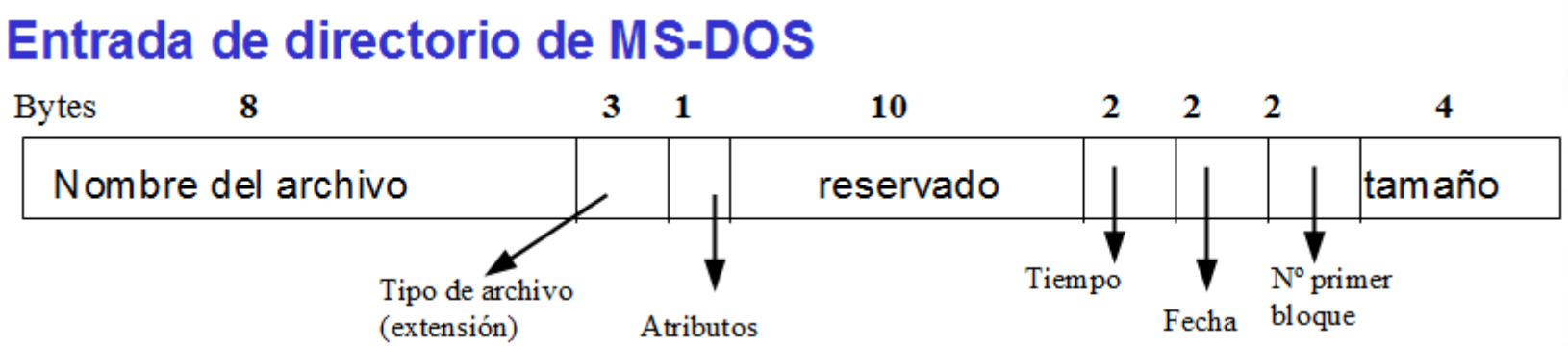
Cuenta:

- A cada entrada de la lista con agrupación se le añade el número de bloques libres consecutivos.

Implementación de directorios

¿Qué puede contener una entrada de directorio?

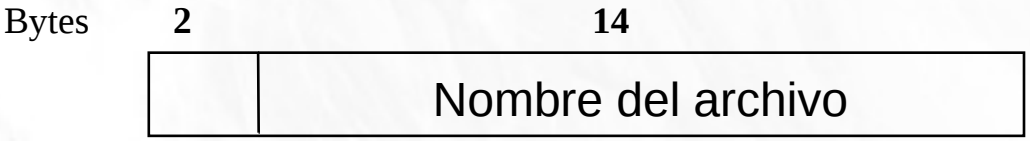
- MS-DOS: Nombre de archivo + Atributos + Nº primer bloque de datos (FAT).



Implementación de directorios

- UNIX: Nombre de archivo + Referencia a los Atributos.

Entrada de directorio de UNIX (s5fs)



- Sesión de trabajo. 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

Implementación de la compartición de archivos (enlaces):

Enlaces simbólicos o blandos (***soft links***):

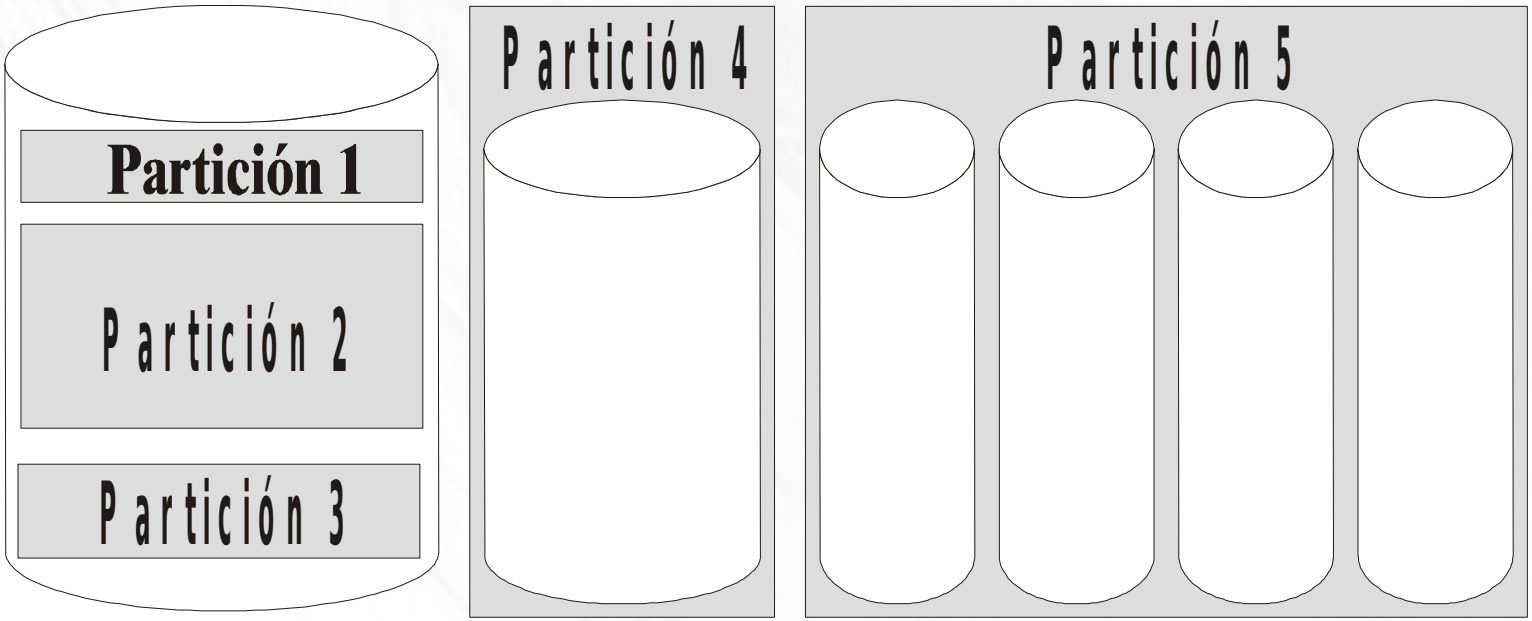
- Se crea un nuevo archivo de tipo enlace que almacena el camino absoluto o relativo del archivo al que enlaza.
- Se puede usar en entornos distribuidos.
- Provoca gran número de accesos a disco.

Enlaces duros (***hard links***)

- Se crea una nueva entrada en el directorio con un nuevo nombre y se copia la referencia a los atributos.
- Problema. Borrar un nombre de archivo no implica borrar los atributos. Solución: Contador de enlaces. Si contador de enlaces == 0 => borrar los atributos.

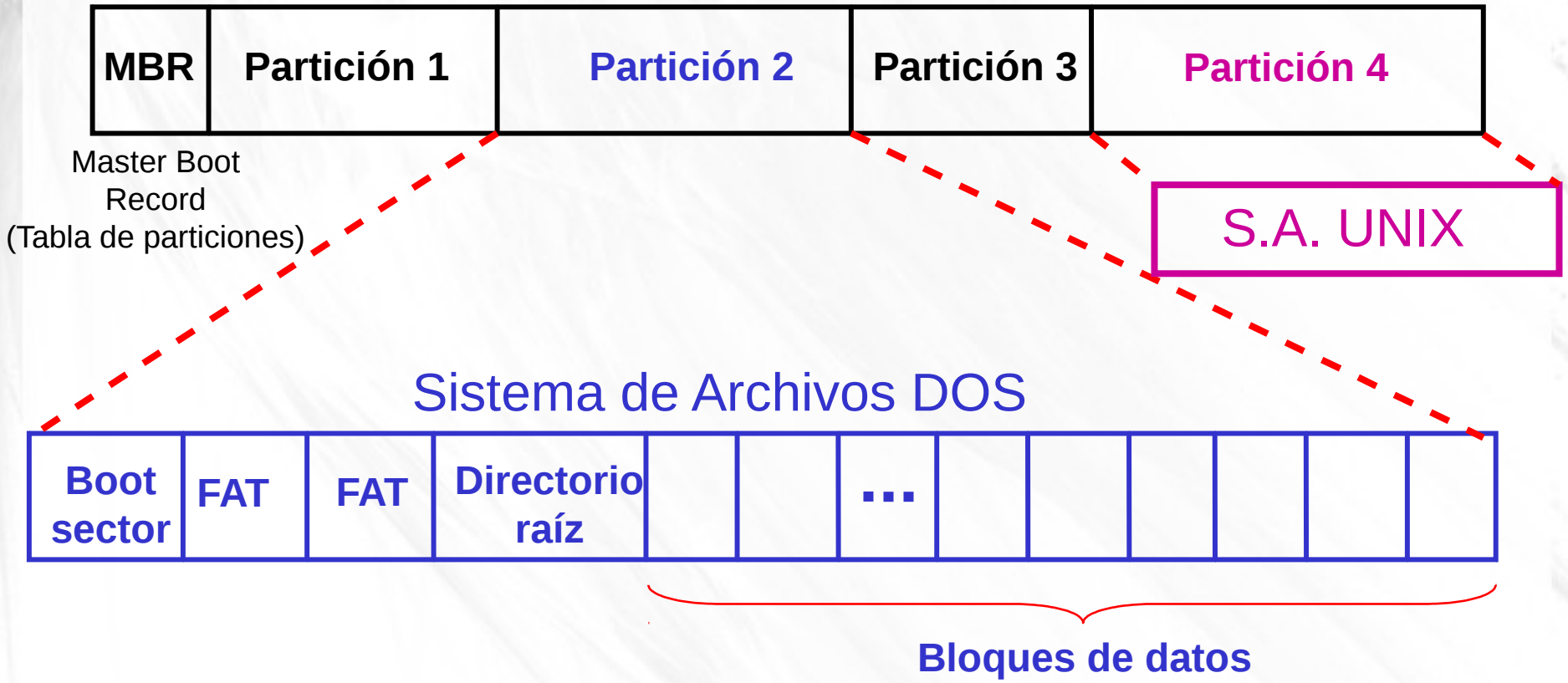
Distribución del sistema de archivos

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



Distribución del sistema de archivos

- Ejemplo de organización en particiones.



Distribución del sistema de archivos

- 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, lista de bloques libres, espacio para atributos de archivo,...).
- Bloque de arranque para inicializar el sistema localizado por bootstrap.
- Métodos necesarios para detectar y manejar bloques dañados.

Tolerancia a fallos y 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.

Distintos enfoques:

- 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.
- 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.

Contenidos

- Interfaz de los Sistemas de Archivos
- Aspectos de diseño del Sistema de Archivos
- Implementación de la gestión de archivos en Linux

Implementación de la gestión de archivos en Linux

- Sistema de archivos (SA).
- inodo (*inode*).
- *Virtual Filesystem* (VFS).
- Sistema de archivos `ext2`.
- Montaje y desmontaje de un SA.

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 (**`vfat`**, **`iso9660`**, **`ntfs`** y muchos más **`mount(8)`**)
- El kernel incluye una capa entre los procesos de usuario (o la biblioteca estándar) y la implementación del SA particular: Sistema de Archivos Virtual (VFS – Virtual File System).
- Esta capa permite proporcionar una interfaz uniforme para la gestión de archivos que abstrae las particularidades de cada SA.

Sistema de archivos

- Abstracción de las particularidades de los SA proporcionada por VFS: conjunto único de llamadas para trabajar con archivos y SAs.

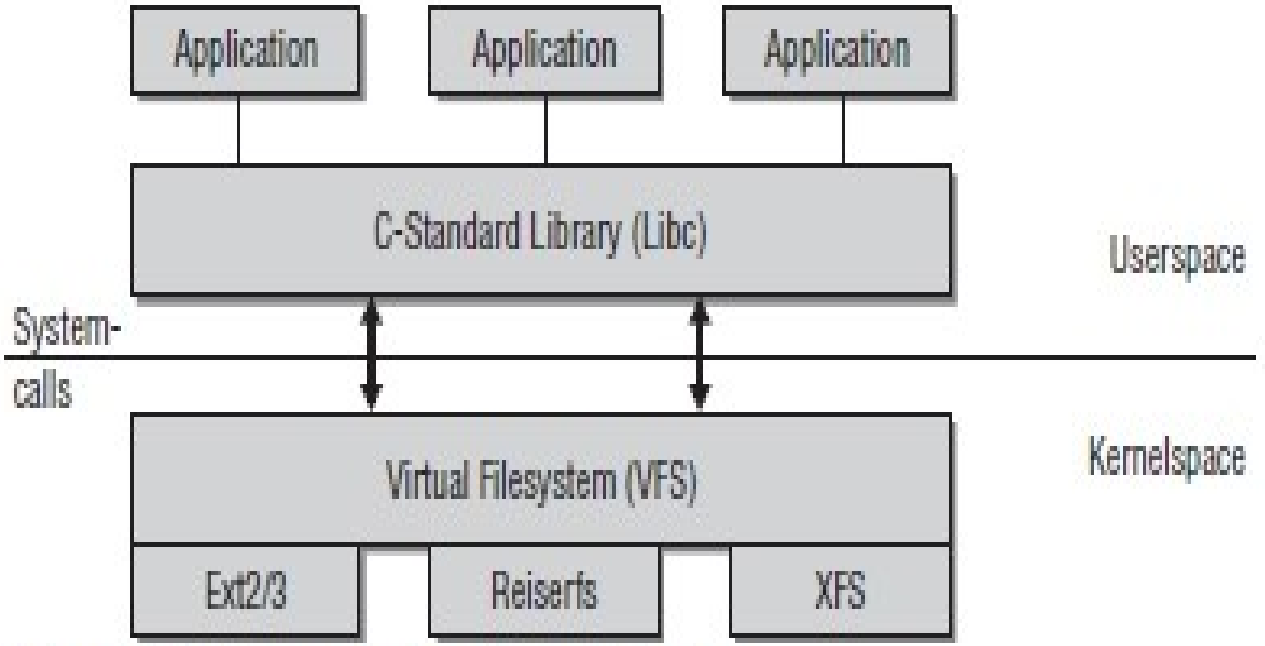


Figure 8-1: VFS layer for filesystem abstraction.

Sistema de archivos

- 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 la llamada al sistema **`write()`**.

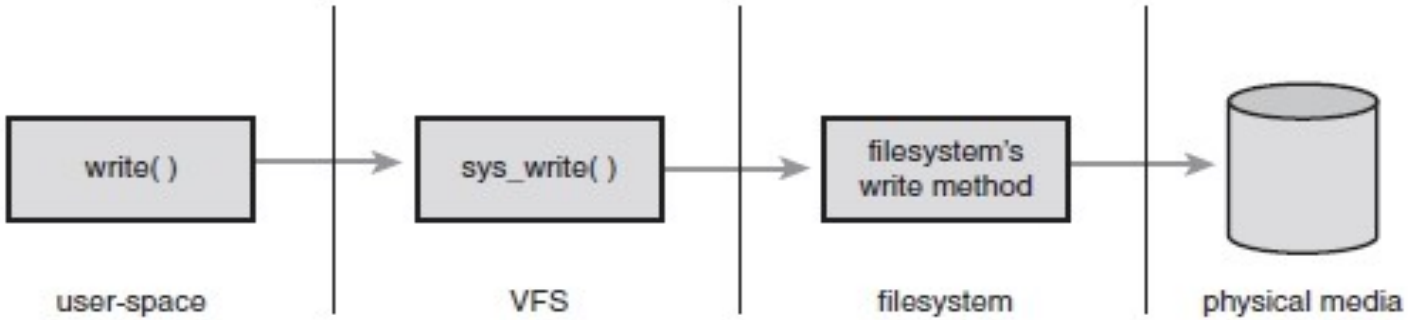


Figure 13.2 The flow of data from user-space issuing a `write()` call, through the VFS's generic system call, into the filesystem's specific write method, and finally arriving at the physical media.

Sistema de archivos: Tipos

- **SA basados en disco** (Disk-based filesystems). Forma clásica de almacenar archivos en medios no volátiles: **`ext2/ext3/ext4,reiserfs,vfat,iso9660`**, etc.
- **SA Virtuales** (Virtual filesystems): generados por el kernel y constituyen una forma simple para permitir la comunicación entre los programas y los usuarios. Por ejemplo el **`/proc`**. No requieren espacio de almacenamiento en ningún dispositivo hardware porque la información está en MP.
- **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 tienen por qué apuntar al mismo archivo.
- Un inodo es la estructura asociada a cada archivo y directorio y contiende sus metadatos.
- Existe otra estructura que mantiene principalmente el puntero de lectura/escritura del archivo y el modo de apertura (información de sesión).

i-nodo

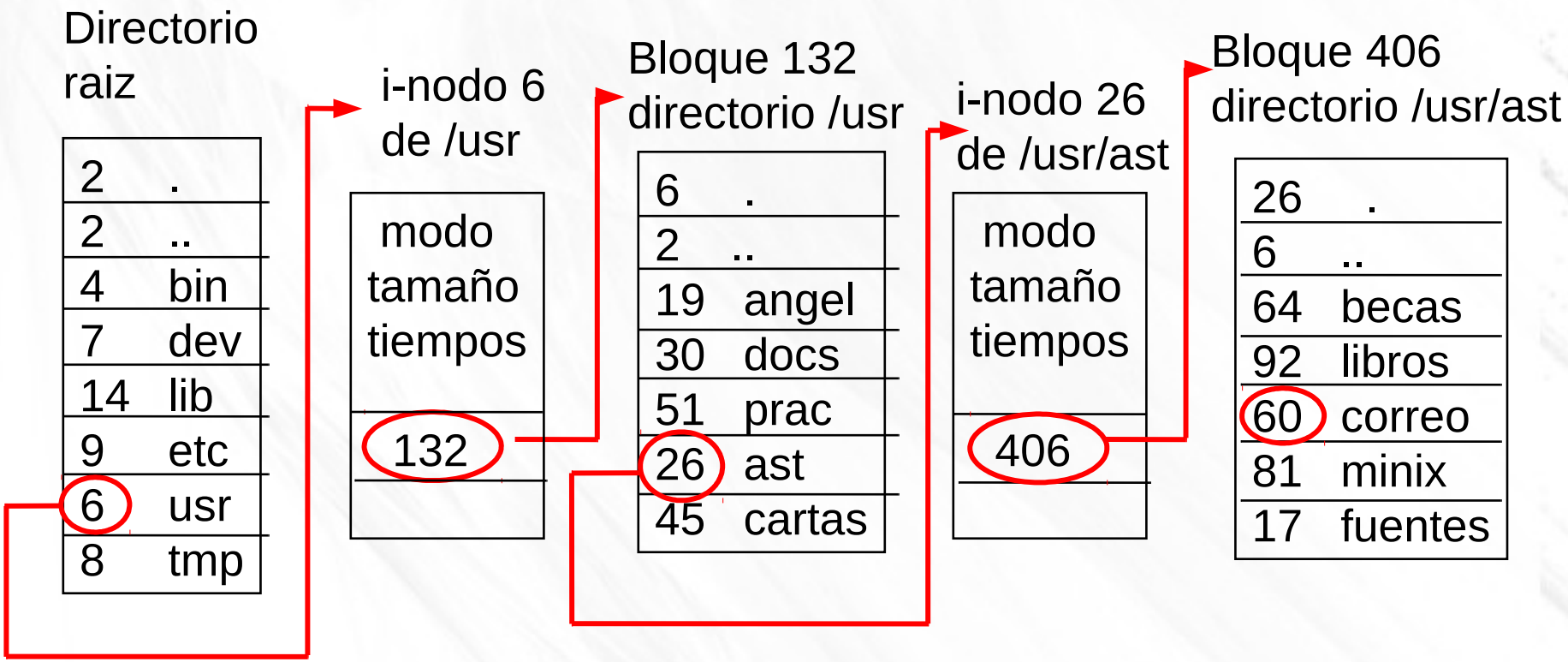
- **i-nodo**: representación interna de un archivo, es decir, metadatos o atributos de un archivo almacenados en una estructura de datos.
- Un archivo tiene asociado un único i-nodo, aunque éste puede tener distintos nombres (enlaces duros) en el espacio de nombres.
- Si un proceso:
  - Crea un archivo, entonces se le asigna una nueva estructura de datos (**i-nodo en disco**).
  - Referencia a un archivo por su nombre, entonces se analizan permisos y se lleva el i-nodo a memoria principal (**i-nodo *in core***) hasta que se cierre.

Campos de un i-nodo

- Identificador del propietario del archivo y del grupo asociado al archivo: **`UID`**, **`GID`**
- Tipo de archivo (regular, directorio, dispositivo, cauce, link). Si este campo es 0, entonces 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 (*hard links*).
- **Campo de localización**: estructura de datos que mantiene los números de bloques de disco que soportan los datos del archivo.
- Tamaño.
- Recordad **`stat()`** y **`struc stat`** de prácticas.

Acceso a un archivo

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



Virtual File System (VFS)

- VFS sigue un diseño orientado a objetos, y consta de dos entidades, archivo 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 del proceso (local). Las anteriores son estructuras de sistema (globales).

Virtual File System (VFS)

Cada uno de estos objetos primarios tiene un vector de operaciones, **`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_inode()`** 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()`**.

Virtual File System (VFS)

- 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 **`files_struct`**.

Virtual File System (VFS)

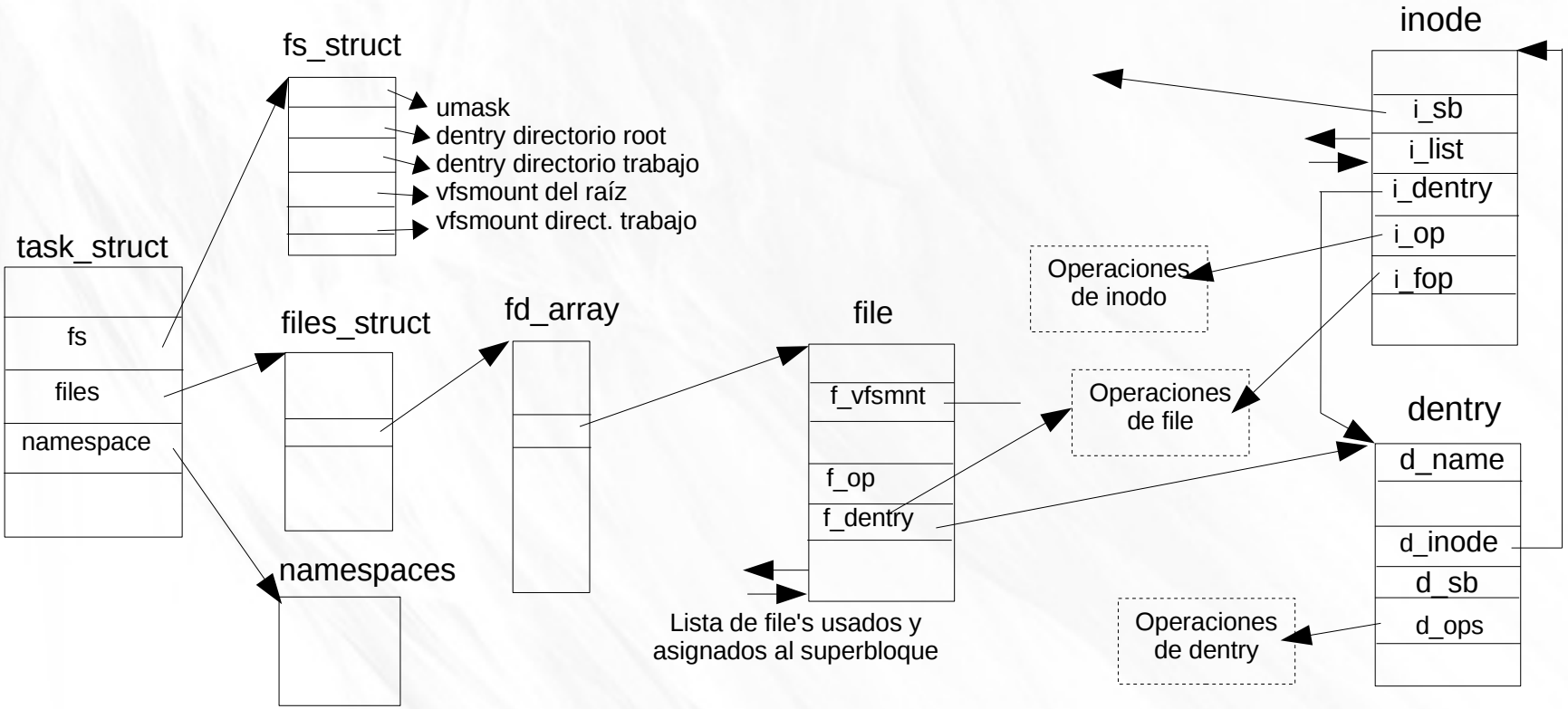


Figura: Estructuras de datos VFS en Linux 2.6.x de José Antonio Gómez

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*).

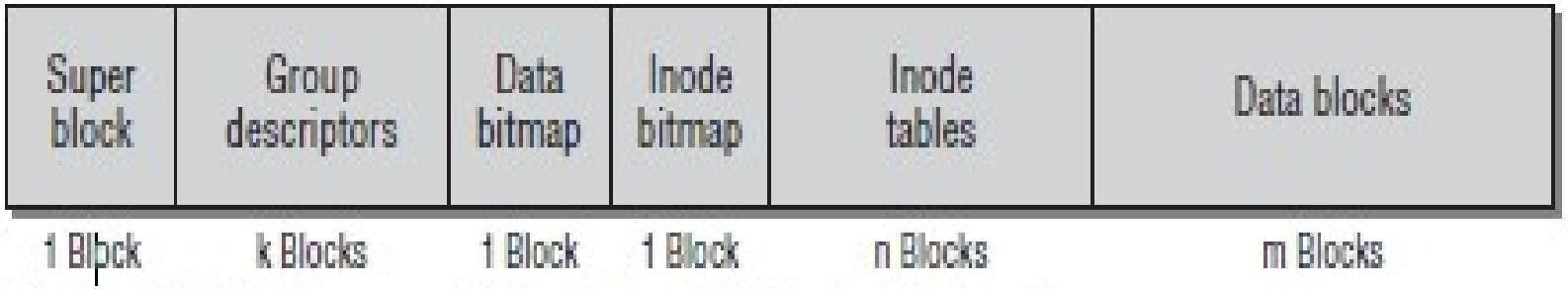


Figure 9-2: Block group of the Second Extended Filesystem.

Sistema de archivos ext2

- 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.

SA ext2: 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.

SA ext2: 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`

SA ext2: Campo de localización

- 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.

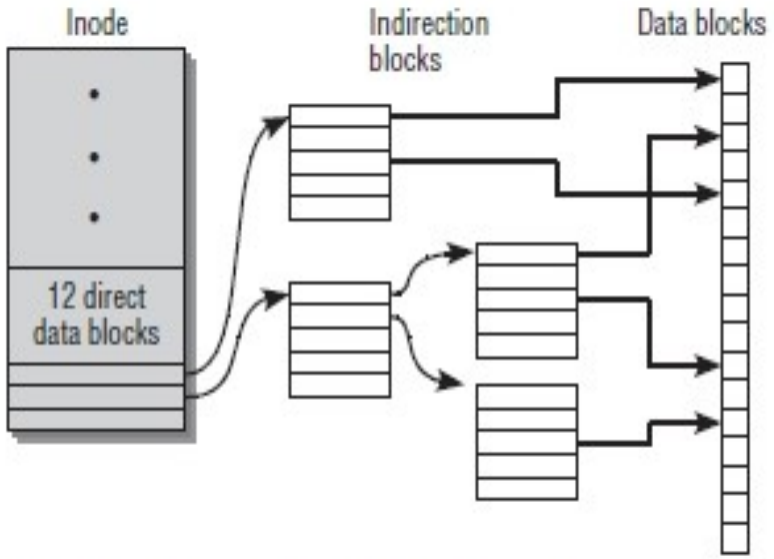


Figure 9-4: Simple and double indirection.

Montaje y desmontaje de SA

- 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
  - Referencia 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

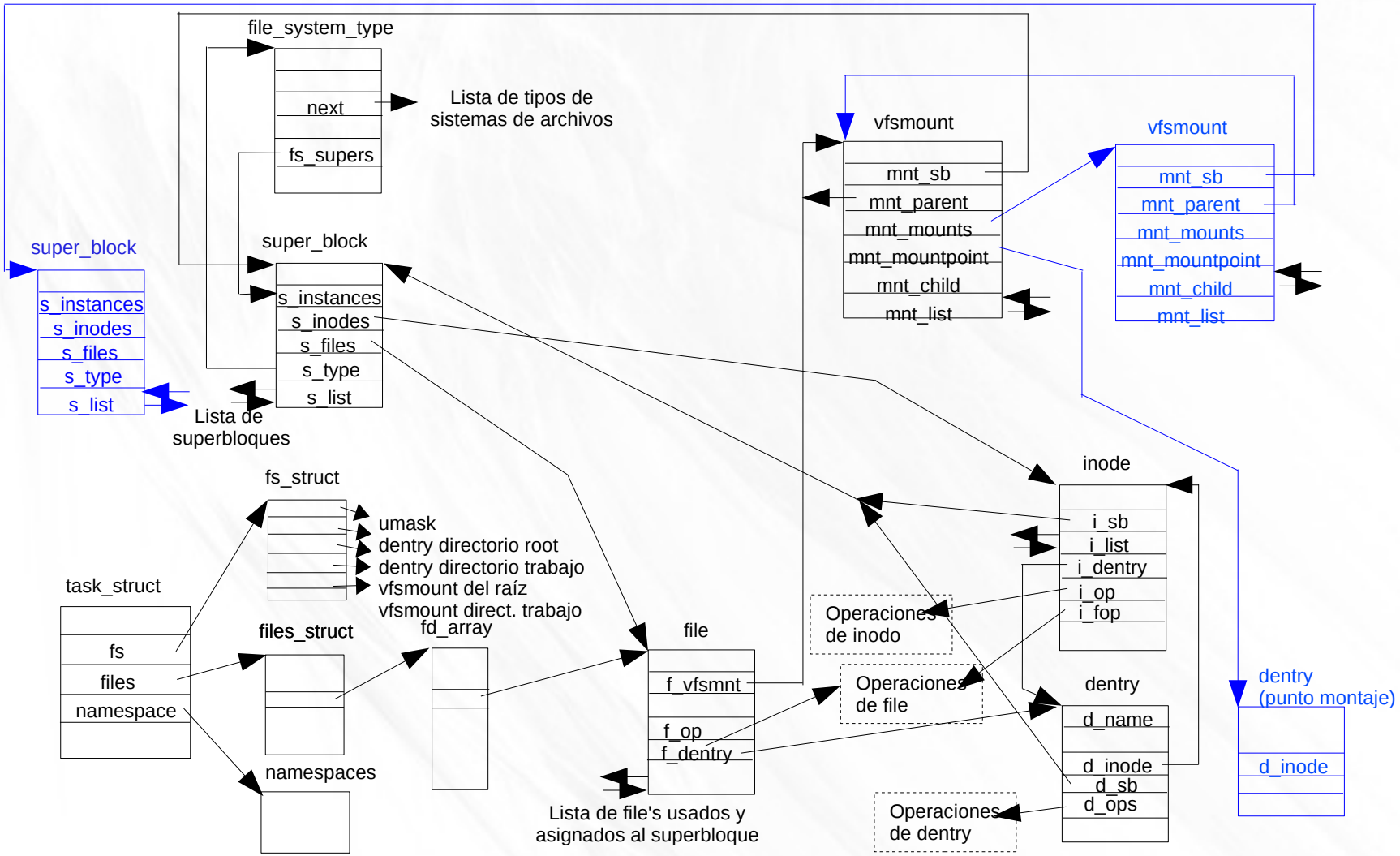


Figura © José Antonio Gómez