

Tema 5

Sistemas de Ficheros

Sistemas Operativos

UITEC Departamento de Ingeniería
y Tecnología de Computadores

Índice

1. **Introducción** (Tanenbaum [C6.Introducción])
2. **Ficheros** (Tanenbaum [C6.1])
3. **Directorios** (Tanenbaum [C6.2])
4. **Implementación del sistema de ficheros** (Tanenbaum [C6.3])
5. **Sistemas de ficheros en Linux** (Tanenbaum [C10.6])
6. **Sistemas de ficheros en Windows 2000** (Tanenbaum [C11.7])

1. Introducción

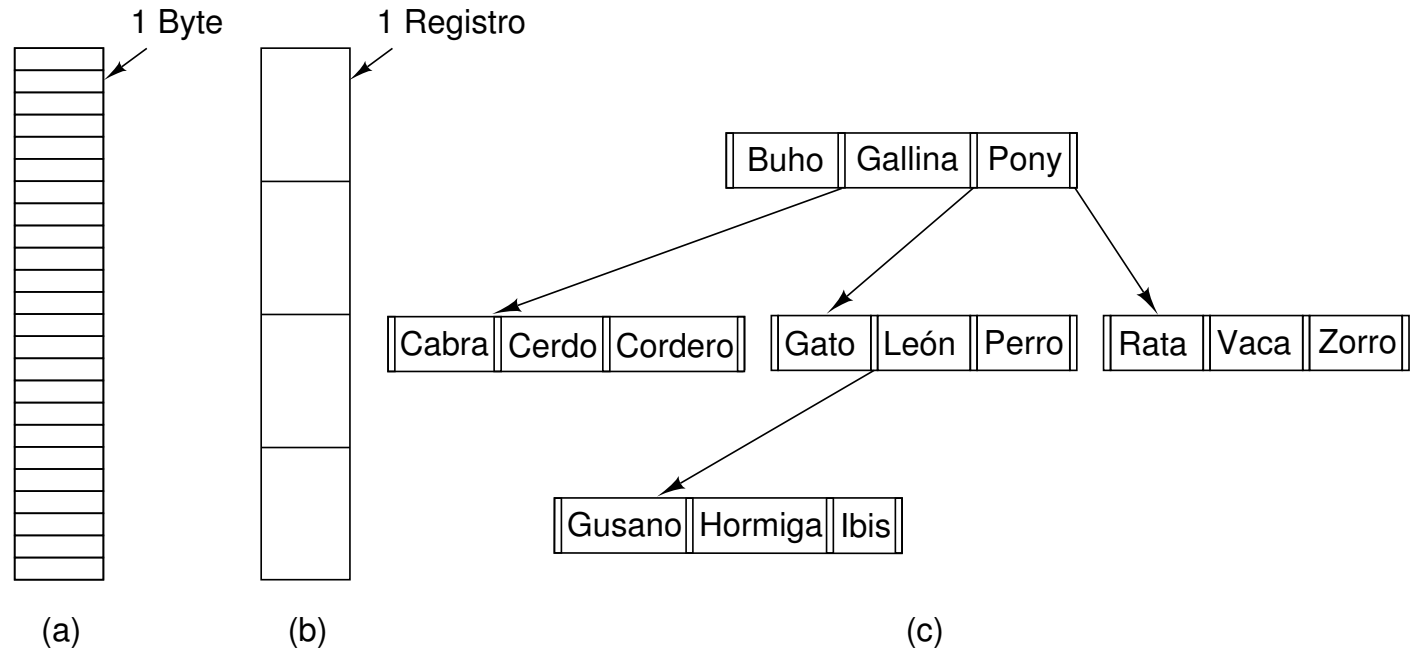
- El **almacenamiento secundario** es necesario para:
 - Almacenar gran cantidad de datos
 - Almacenar datos persistentes (válidos entre sesiones)
 - Compartir datos (si la protección de la memoria no lo permite)
- Los dispositivos de almacenamiento secundario pueden ser muy distintos \Rightarrow El S.O. debe proporcionar una **interfaz sencilla** para acceder a dichos dispositivos
- Solución: **sistema de ficheros**, basado en **ficheros** y **directorios**
- Primero veremos el sistema de ficheros desde el \sphericalangle del usuario y, después, desde el \sphericalangle del S.O. (implemen.)

Índice

- 2. Ficheros (Tanenbaum [C6.1])
 - 2.1. Concepto de fichero
 - 2.2. Estructura de un fichero. Tipos de ficheros
 - 2.3. Acceso a un fichero
 - 2.4. Atributos de un fichero
 - 2.5. Operaciones con ficheros
 - 2.6. Ficheros proyectados en memoria
 - 2.7. Semánticas de compartición Carretero[C8.2.5]

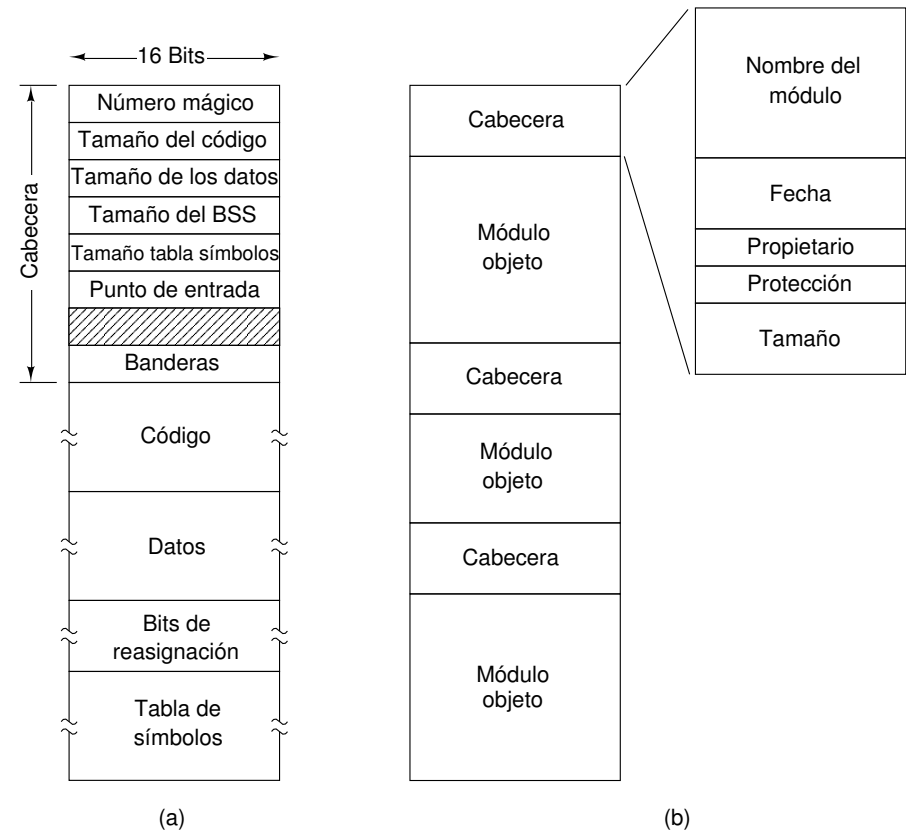
2. Ficheros: concepto y estructura

- Un **fichero** es la unidad lógica de almacenamiento y se identifica mediante un nombre (estructurado o no)
- Son posibles varias **estructuras**:
 - **secuencia de bytes** (la más genérica)
 - **secuencia de registros**
 - **árbol**



2. Tipos de ficheros y accesos

- Tipos de ficheros
 - Ficheros normales o regulares (**ASCII o binarios**)
 - Directorios
 - Ficheros especiales de caracteres
 - Ficheros especiales de bloques
- Acceso:
 - Secuencial
 - Aleatorio



(a) Fichero ejecutable (b) Biblioteca

2.4 Atributos de un fichero

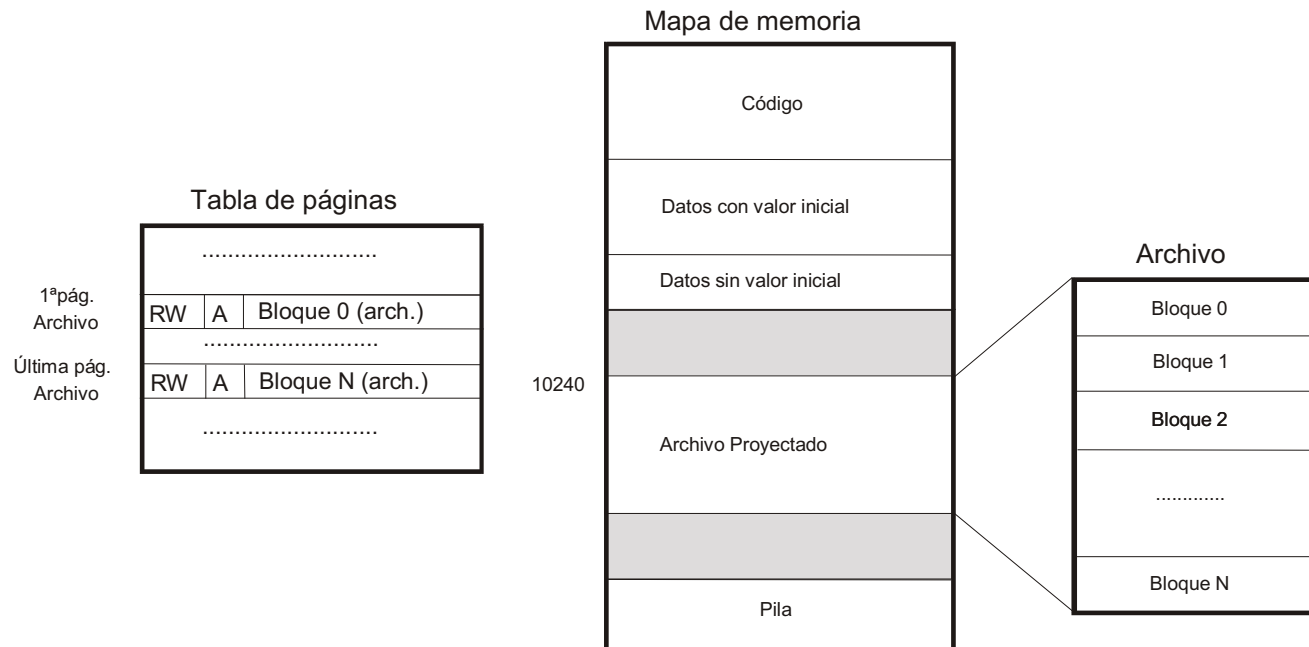
Campo	Significado
Protección	Quién debe tener acceso y de qué forma
Contraseña	Contraseña necesaria para tener acceso al fichero
Creador	Identificador de la persona que creó el fichero
Propietario	Propietario actual
Bandera «sólo lectura»	0 Lectura/escritura, 1 para lectura exclusivamente
Bandera de ocultación	0 normal, 1 para no exhibirse en listas
Bandera de sistema	0 fichero normal, 1 fichero del sistema
Bandera de biblioteca	0 ya se ha respaldado, 1 necesita respaldo
Bandera ASCII/binario	0 fichero en ASCII, 1 fichero en binario
Bandera de acceso aleatorio	0 sólo acceso secuencial, 1 acceso aleatorio
Bandera temporal	0 normal, 1 eliminar al terminar el proceso
Bandera de cerradura	0 no bloqueado, $\neq 0$ bloqueado
Longitud de registro	Número de bytes en un registro
Posición de la clave	Ajuste de la clave dentro de cada registro
Longitud de la clave	Número de bytes en el campo clave
Tiempo de creación	Fecha y hora de creación del fichero
Tiempo del último acceso	Fecha y hora del último acceso al fichero
Tiempo de la última modificación	Fecha y hora de la última modificación del fichero
Tamaño actual	Número de bytes en el fichero
Tamaño máximo	Tamaño máximo al que puede crecer el fichero

2.5 Operaciones con ficheros

- Create: crea un fichero vacío
- Delete: elimina un fichero
- Open: abre un fichero para operar con él
- Close: cierra un fichero abierto
- Read/Write: lee/escribe datos de/en un fichero
- Append: escribe datos al final del fichero
- Seek: especifica el punto de lectura/escritura de datos en un fichero de acceso aleatorio
- Get/Set attributes: obtiene/establece los atributos asociados a un fichero
- Rename: cambia el nombre de un fichero en un directorio
- Truncate: elimina el contenido de un fichero a partir de una posición dada

2.6 Ficheros proyectados en memoria

- Es otra forma de acceder a un fichero (sin operaciones `read`, `write`, etc.)
- Consiste en hacer corresponder una zona del espacio de direcciones de un proceso con un fichero
 - Dos nuevas funciones: `mmap` (crea la correspondencia) y `munmap` (la elimina)
 - Se accede al fichero como se accede a la mem. principal



2.6 Ficheros proyectados en memoria (II)

- La técnica se adapta bien a la memoria virtual:
 - En **paginación**: las páginas de una zona de memoria se corresponden con el fichero
 - Fallo de página \Rightarrow leer página del fichero
 - Reemplazo de página \Rightarrow escribir modificaciones en fichero
 - En **segmentación**: un segmento se corresponde con el fichero
- Problemas de la proyección de ficheros en memoria:
 - ¿Se conoce el tamaño exacto de un fichero proyectado en memoria o su tamaño es múltiplo del tamaño de página?
 - ¿Qué pasa si el fichero a proyectar es mayor que el tamaño máximo de segmento o mayor que el tamaño máximo de la memoria virtual?
 - ¿Qué pasa si un proceso proyecta un fichero y otro proceso accede a él mediante operaciones ordinarias (`read`, `write`,...)?

2.7 Semánticas de compartición

Puede ocurrir que dos o más procesos usen el mismo fichero y uno de ellos escriba en él. ¿Cuándo verán los cambios el resto de procesos?:

- **Semántica Unix:** inmediatamente
 - Problema: si varios procesos escriben sobre la misma porción del fichero, el resultado final se desconoce
 - Solución: Unix ofrece mecanismos para que un proceso pueda bloquear todo el fichero o parte de él, y así poder hacer cambios de forma exclusiva

2.7 Semánticas de compartición (II)

- **Semántica de sesión:** cuando el proceso que modifica el fichero lo cierre o explícitamente indique que se debe actualizar
 - Las modificaciones sólo son visibles para «sesiones» posteriores del fichero (siendo una «sesión» el conjunto de operaciones que se realizan entre una operación «open» y la correspondiente operación «close»)
 - El proceso que modifica el fichero tiene su propia copia del fichero, llamada «versión»
 - Problema: cuando varios procesos modifican un mismo fichero, cada uno tiene su propia versión (posiblemente obsoleta) del fichero. Al final, ¿qué copia es la válida?
 - Solución: mecanismos de sincronización entre procesos proporcionados por el S.O. (por ejemplo, para impedir que 2 o más procesos modifiquen a la vez el fichero o para mezclar el contenido de los mismos)

2.7 Semánticas de compartición (III)

- **Semántica de ficheros inmutables:** nunca
 - Para escribir algo en un fichero es necesario crear uno nuevo y escribir en él los datos
 - El nuevo fichero tendrá un nuevo nombre (salvo que se borre el anterior)
 - Se puede utilizar una técnica de «copia en escritura» para optimizar la implementación

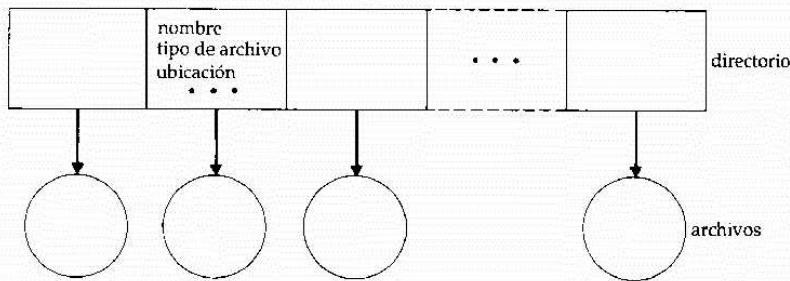
Índice

- 3. Directorios (Tanenbaum [C6.2])
 - 3.1. Sistemas jerárquicos de directorios
 - 3.2. Nombre de la ruta de acceso
 - 3.3. Operaciones con directorios

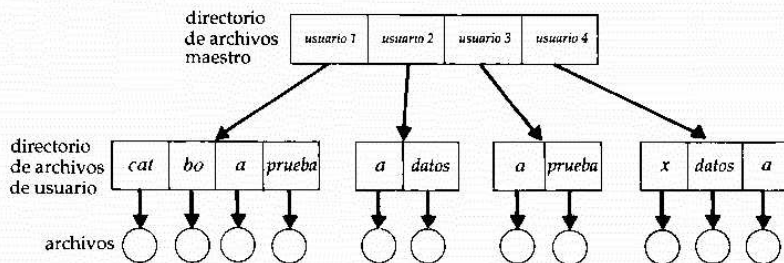
3.1 Directorios

- Suelen ser ficheros que almacenan información sobre otros ficheros (nombre, atributos, etc.)
- Son posibles distintas organizaciones:

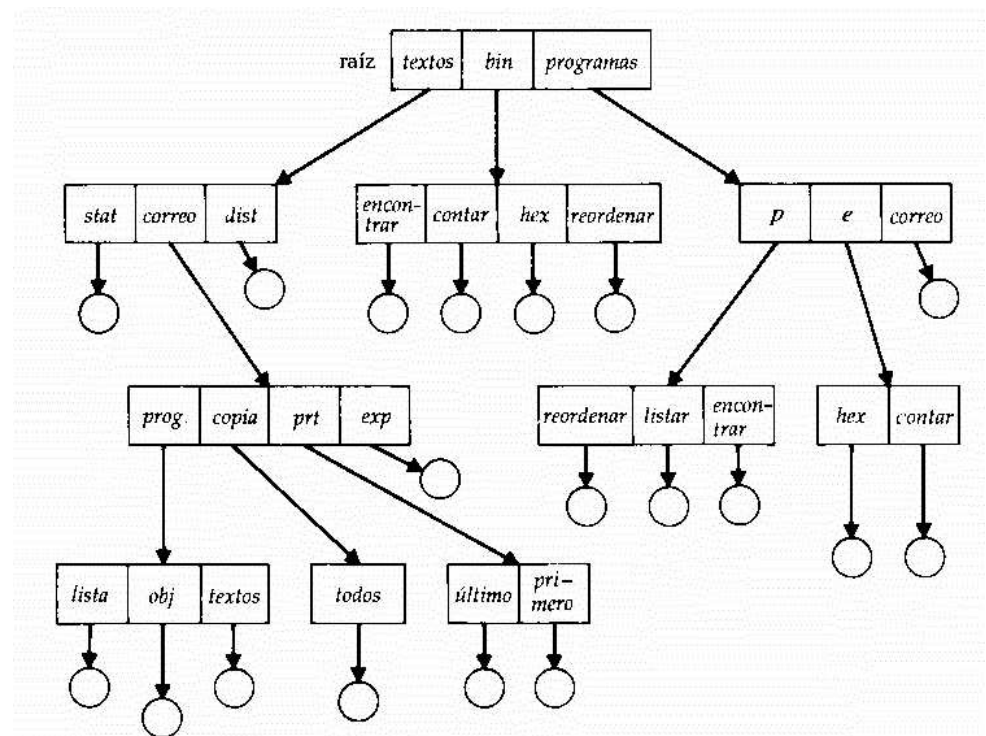
Directorio único



Un directorio por usuario



Sistema jerárquico de directorios



3.2 Nombre de la ruta de acceso

Los SS.OO. con un sistema jerárquico de directorios suelen tener dos formas básicas para indicar la ruta de acceso o nombre de un fichero:

- Ruta absoluta:
 - Especifica el camino desde el **directorio raíz** hasta el fichero
 - El primer carácter de la ruta es el separador («/» en Unix, «\» en Windows). Ejemplo: `/usr/bin/mozilla`
- Ruta relativa:
 - Asociada al concepto de **directorio actual o de trabajo**
 - No empieza por el carácter separador y dependen del directorio actual. Ejemplo: `bin/mozilla` si el directorio actual es `/usr`
- Dos directorios especiales:
 - Directorio «.»: directorio actual
 - Directorio «..»: **directorio padre**

3.3 Operaciones con directorios

- Operaciones posibles:
 - Create: crea un directorio vacío
 - Delete: borra un directorio vacío
 - Opendir: abre un directorio para operar con él
 - Closedir: cierra un directorio abierto
 - Readdir: lee una entrada del directorio
 - Rename: cambia de nombre a un directorio
 - Link: crea un **enlace físico** para un fichero existente
 - Unlink: elimina un enlace físico (o el fichero asociado si se elimina el último enlace)
- Preguntas: ¿por qué se usa `readdir`, y no `read`, si el directorio es un fichero?, ¿por qué no existe `writedir`?

Índice

- 4. Implementación del sistema de ficheros (Tanenbaum [C6.3])
 - 4.1. Implementación de ficheros
 - 4.2. Implementación de directorios
 - 4.3. Ficheros compartidos
 - 4.4. Administración del espacio en disco
 - 4.5. Confiabilidad del sistema de ficheros
 - 4.6. Caché de disco. Jerarquía de almacenamiento
 - 4.7. Sistemas de ficheros modernos

4. Implementación del sistema de ficheros

- En lo que sigue vamos a representar al dispositivo de almacenamiento como un **array lineal de bloques**:
 - Supondremos bloques con un tamaño potencia de 2 (512, 1024, 4096, etc. bytes)
 - Los bloques se numerarán desde 0 hasta $N - 1$, siendo N el tamaño del disp. de almacenamiento, en bloques
- Primero veremos la implementación de los elementos individuales (ficheros, directorios, etc.)
- Al final del tema veremos cómo se combina todo para construir un sistema de ficheros completo (en Linux y en Windows 2000)

4.1 Implementación de ficheros

- Vamos a ver distintas técnicas para llevar un registro de qué bloques pertenecen a un fichero

- **Asignación contigua**

- Todos los bloques de un mismo fichero están contiguos

- Pros y contras

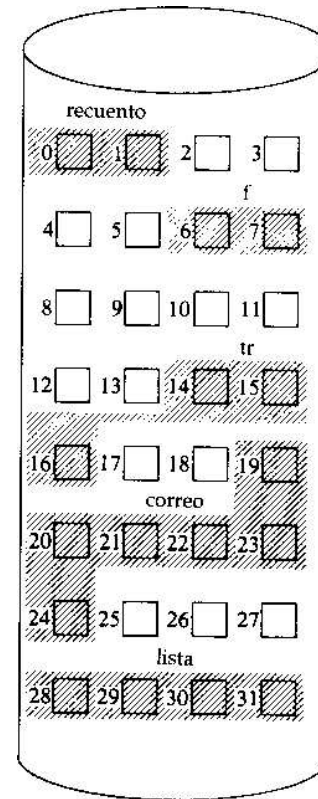
- ✓ Fácil implementación ⇒ Registro: bloque inicial

- ✓ Buen rendimiento

- ✗ Irreal (salvo que se sepa de antemano el tamaño del fichero)

- ✗ Mucha fragmentación externa

- Útil para CD-ROMs y DVDs

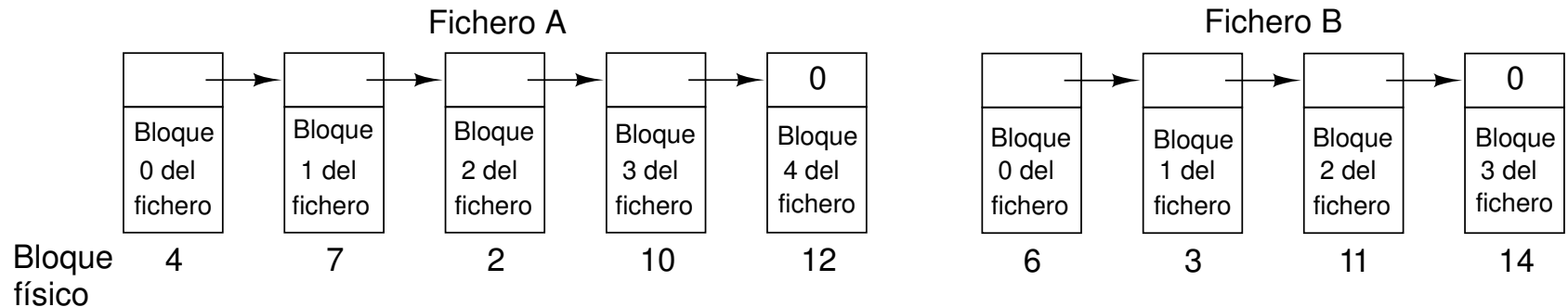


directorio

Archivo	Inicio	Longitud
recuento	0	2
tr	14	3
correo	19	6
lista	28	4
f	6	2

4.1 Implementación de ficheros (II)

● Asignación con lista ligada

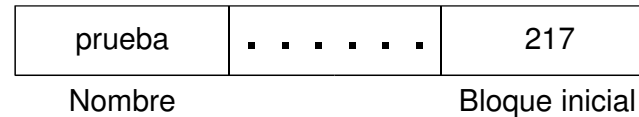


- Cada bloque contiene un puntero (nº de bloque) al bloque siguiente
- Pros y contras
 - ✓ Fácil implementación ⇒ Registro: bloque inicial
 - ✓ Se aprovechan todos los bloques del disco
 - ✗ El acceso aleatorio es lento (hay que recorrer una lista)
 - ✗ El espacio de almacenamiento de un bloque deja de ser potencia de 2

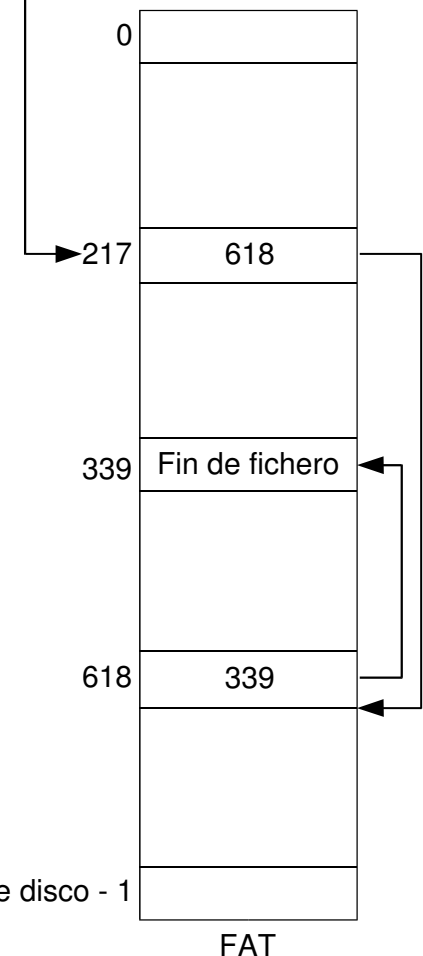
4.1 Implementación de ficheros (III)

● Asignación con lista ligada e índice

Entrada de directorio

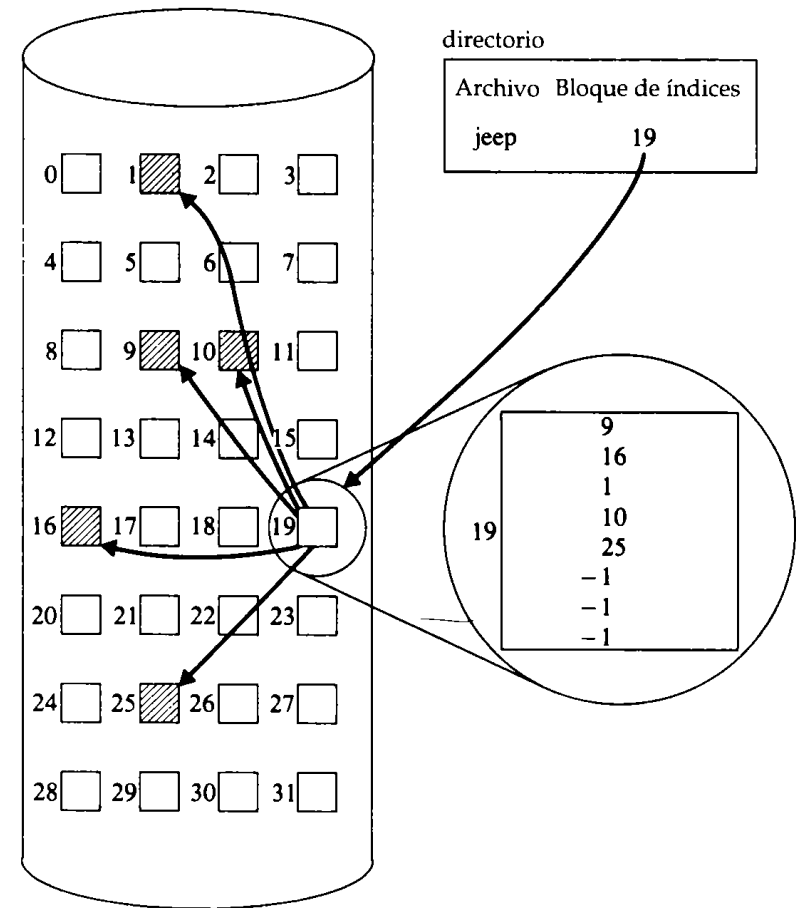


- Misma idea que antes, pero los punteros se almacenan en una estructura aparte (**índice**) que se almacena en disco, se lee cuando se usa el sistema de ficheros y se escribe de nuevo en disco si se modifica
- Desaparecen desventajas anteriores
- Posible problema: tamaño de la tabla
- Ejemplo: FAT de MS-DOS



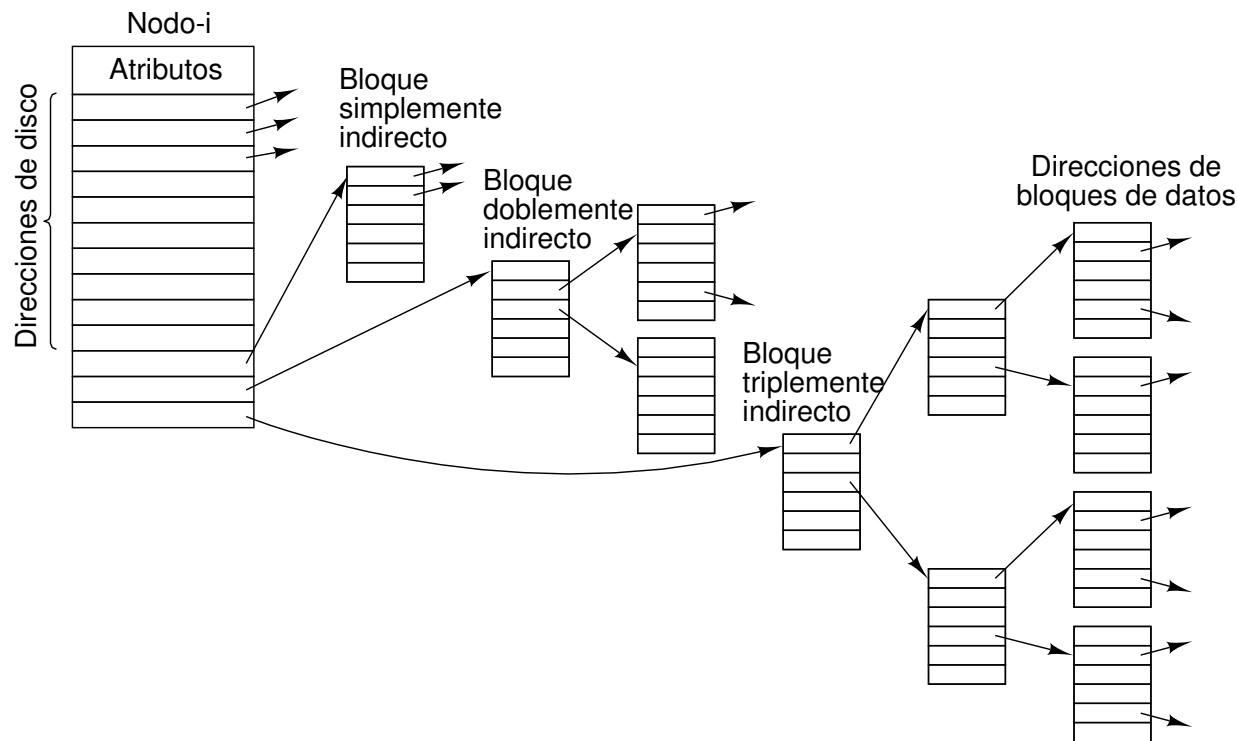
4.1 Implementación de ficheros (IV)

- **Asignación con nodos-i**
 - A cada fichero se le asigna una estructura de datos, llamada **nodo-i**, en donde se almacenan sus atributos y las direcciones de sus bloques
 - Dicha estructura se guarda en disco y se lee cuando se abre el fichero



4.1 Implementación de ficheros (V)

- **Asignación con nodos-i** (continuación...)
 - Para ficheros grandes hay bloques (**bloques indirectos**) que no almacenan datos, sino más direcciones de bloques

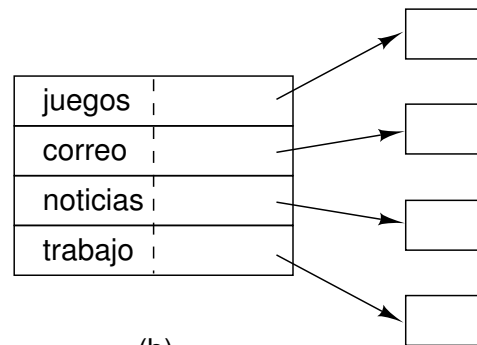


4.2 Implementación de directorios

- **Principal función** de los directorios: asociar un nombre de fichero con la información del propio fichero
- Un aspecto estrechamente relacionado con el anterior es dónde se guardan los **atributos** del fichero. Dos posibilidades:
 - En la propia entrada del directorio (caso (a))
 - En una estructura aparte apuntada por la entrada del directorio (caso (b))

juegos		atributos
correo		atributos
noticias		atributos
trabajo		atributos

(a)



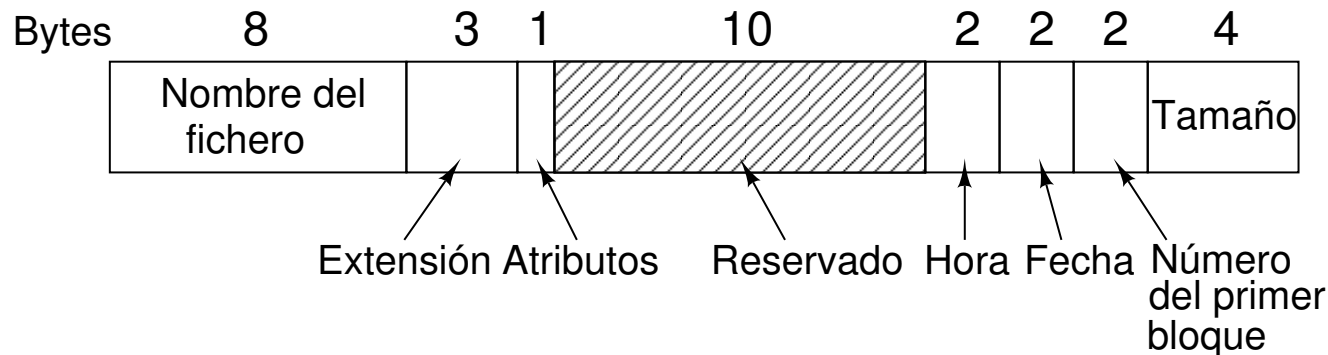
(b)

La estructura de datos contiene los atributos

4.2 Implementación de directorios (II)

● Directorios en MS-DOS

- Los directorios son ficheros que almacenan una lista desordenada de entradas (o registros) de 32 bytes

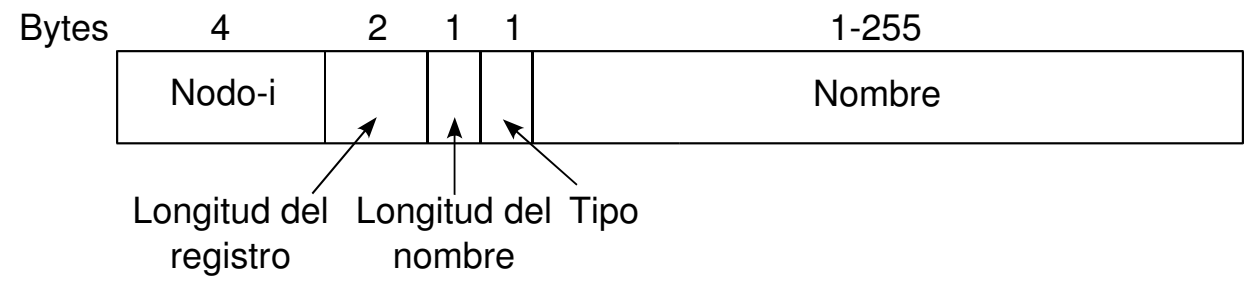


- Un bit de los atributos de la entrada distingue a un directorio de un fichero normal \Rightarrow un directorio puede tener subdirectorios \Rightarrow Árbol de directorios
- El directorio raíz es una excepción, ya que ocupa unos bloques fijos de disco y tiene un tamaño máximo establecido

4.2 Implementación de directorios (III)

● Directorios en Linux (Ext2/Ext3)

- También es posible crear un árbol de directorios
- Todos los directorios (incluido el raíz) son ficheros que almacenan una lista desordenada de entradas de longitud variable



- Las entradas no almacenan atributos de ficheros sino nombres y n^o's de nodos-i asociados. El tipo de fichero es una excepción y se utiliza para acelerar los listados de directorios

4.2 Implementación de directorios (IV)

● Directorios en Windows 2000 (NTFS)

- Al igual que antes, también permite crear un árbol de directorios
- Los directorios pequeños son una lista desordenada de entradas con un formato similar al siguiente:

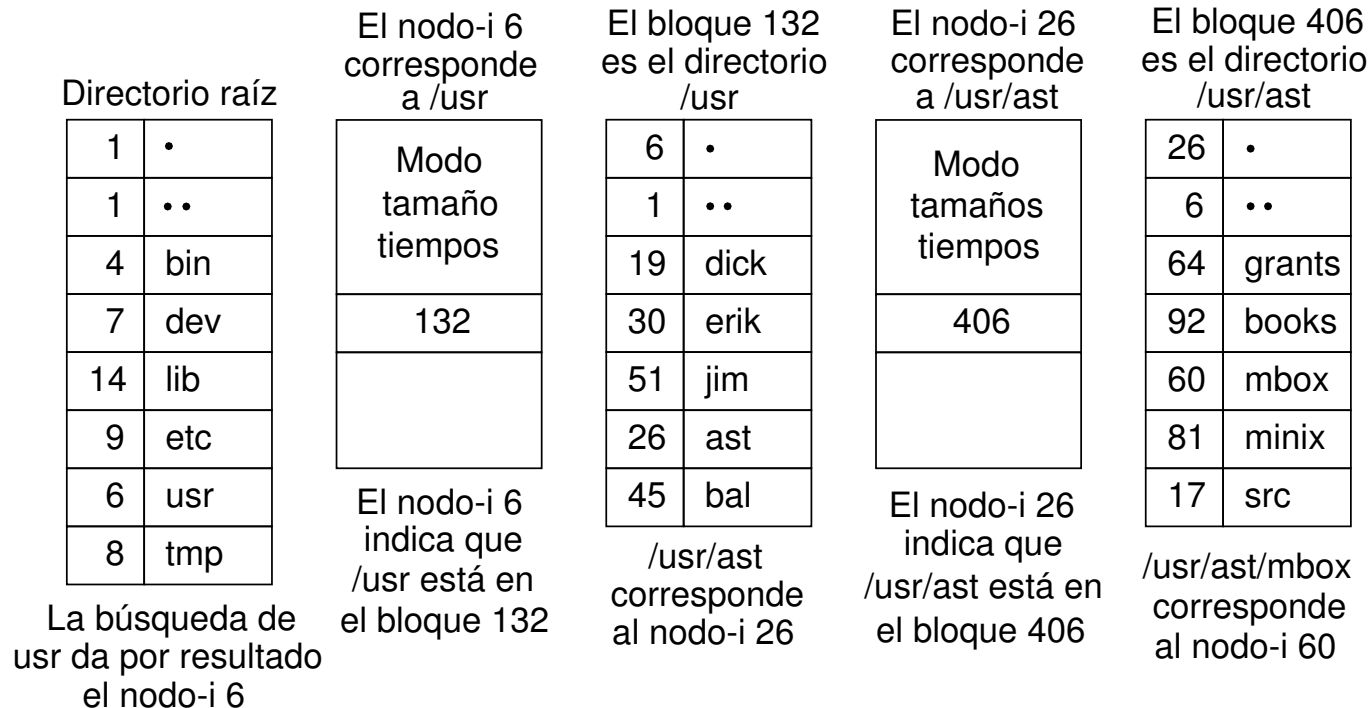
Nº entrada de la MFT	Atributos	Longitud del nombre	Nombre en Unicode
----------------------	-----------	---------------------	-------------------

- Algunos atributos (como el instante de modificación o el tamaño) tienen una copia en la entrada de directorio para optimizar el listado del directorio
- Los directorios grandes se implementan como árboles B+
- El número de entrada en la MFT desempeña un papel similar al de los nodos-i, como ya veremos

4.2 Implementación de directorios (V)

● Ejemplo de resolución de ruta

- Se basa en la implementación de directorios de Unix
- El mecanismo es similar para otros casos



- Se puede usar una «caché» para guardar el resultado de la resolución y así acelerar el proceso

4.3 Ficheros compartidos

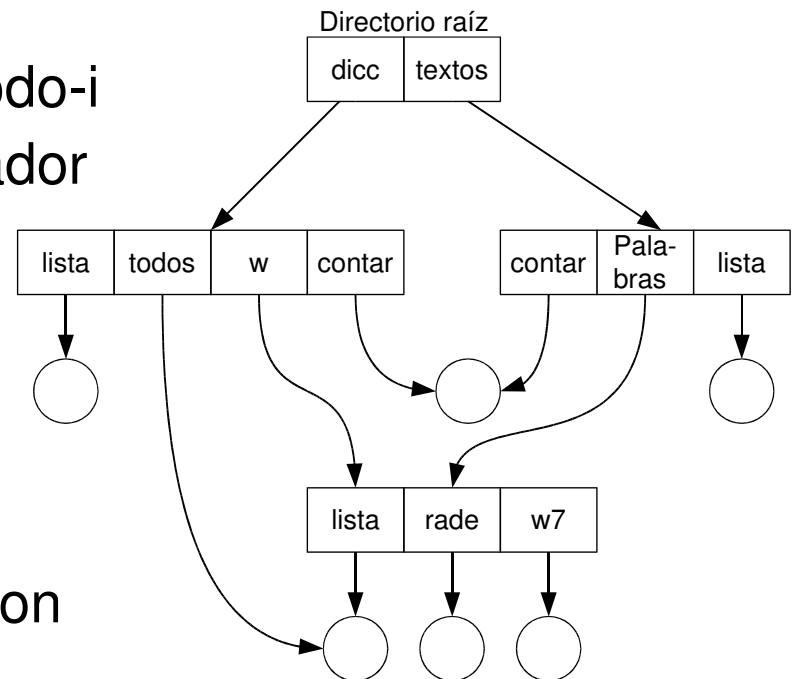
- Son ficheros con más de un nombre, dentro de un mismo directorio o en directorios distintos
- 2 implementaciones básicas

- **Enlaces físicos** (*hard links*)

- Necesitan estructura tipo nodo-i
- También necesitan un contador de enlaces

- **Enlaces simbólicos** (*soft links*)

- Ficheros especiales de tipo «link»
- Pueden tener un alto coste (temporal y espacial) pero son más flexibles

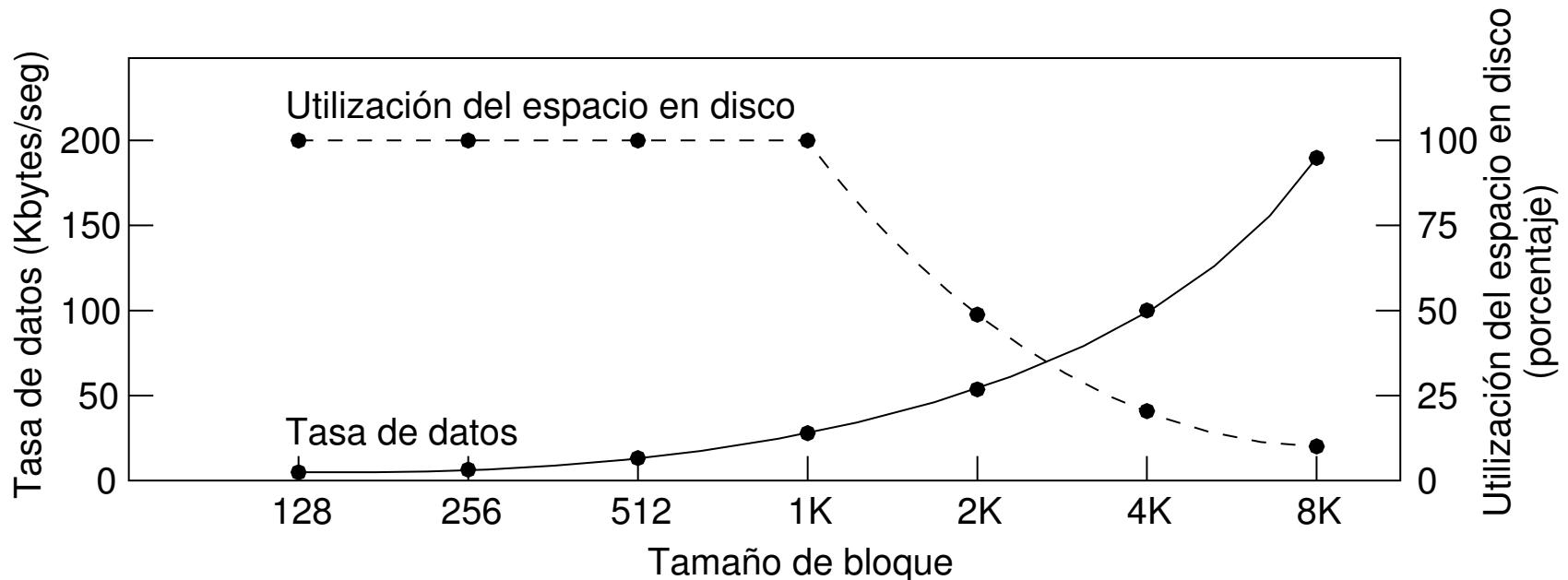


- **Problema:** peligro de duplicidad de datos

4.4 Administración del espacio en disco

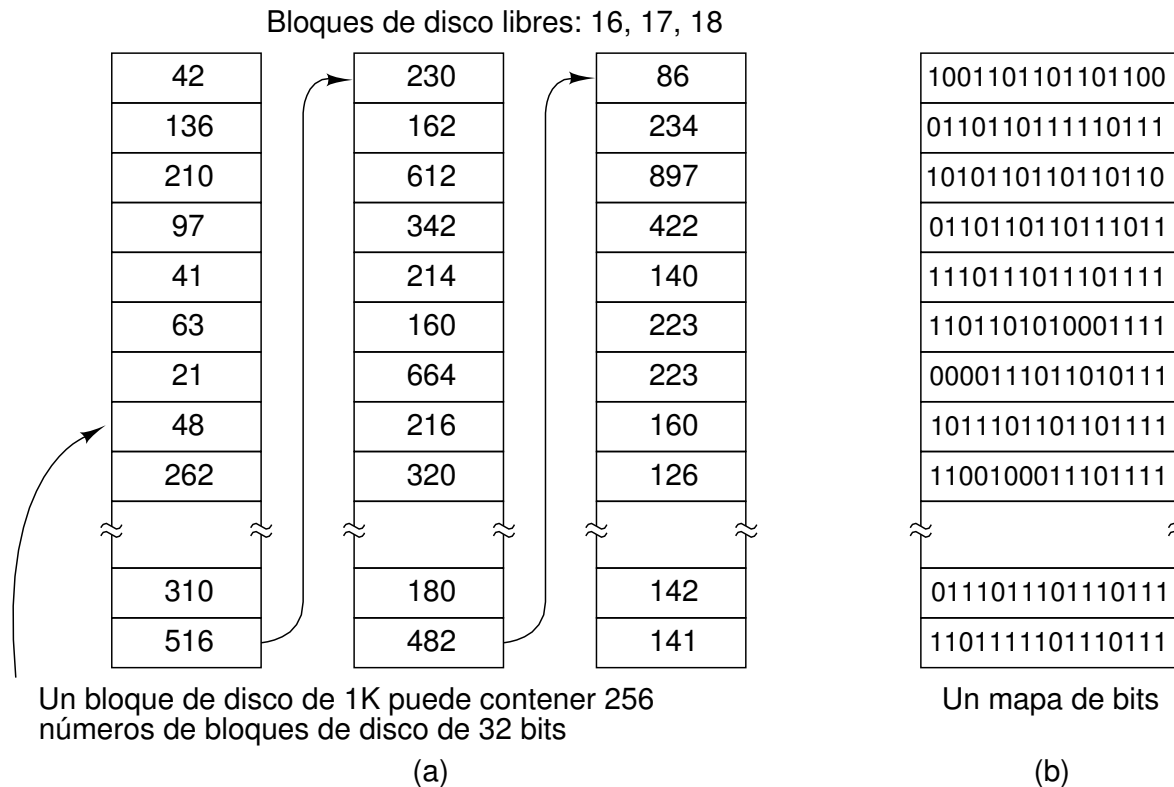
● Tamaño de bloque lógico

- El tamaño del **bloque lógico** (usado por el S.F.) suele ser múltiplo del tamaño del **bloque físico** (usado por el disp. de almacenamiento)
- A la hora de elegir un tamaño de bloque lógico hay que buscar un equilibrio razonable entre:
 - eficiencia en el uso del espacio (desperdiciado, principalmente, por fragmentación interna) y
 - tasa de transferencia



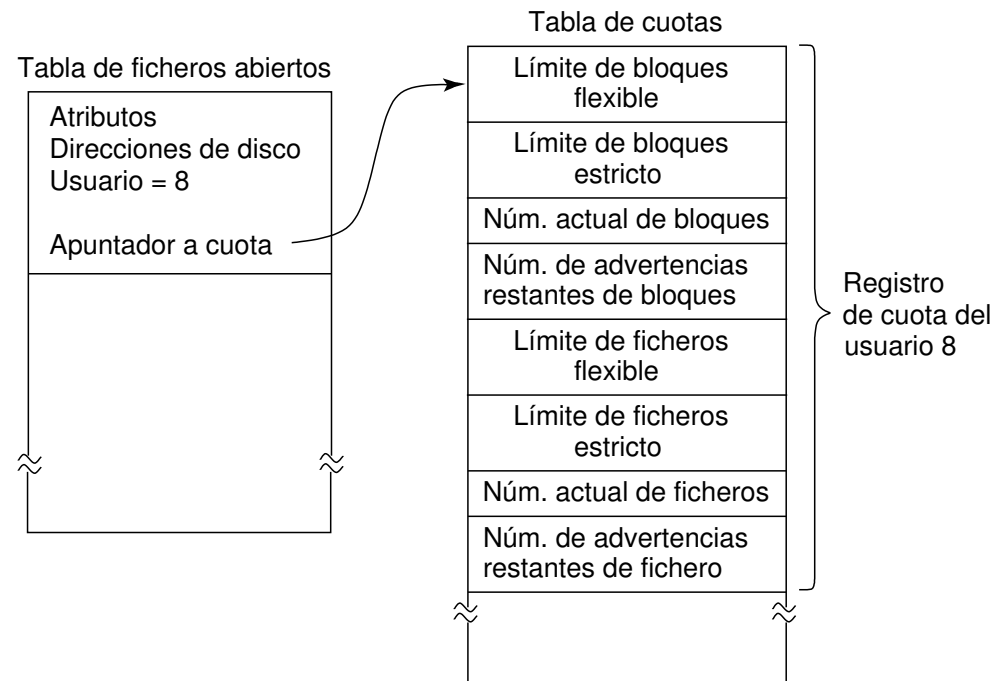
4.4 Administración del espacio en disco (II)

- **Registro de bloques libres.** Se pueden usar algunas técnicas similares a las vistas para la memoria principal:
 - **lista ligada de bloques libres agrupados**
 - **mapa de bits** (FAT en MS-DOS, Ext2/Ext3 en Linux, NTFS en Windows 2000, etc.)



4.4 Administración del espacio en disco (III)

- **Cuotas de disco.** Evitan que un usuario se apodere de todo el espacio de disco. Permiten limitar:
 - el nº de ficheros usados
 - el nº de bloques usados
- El administrador puede establecer cuotas de usuario y de grupo
- La configuración sobre cuotas se guarda en ficheros especiales del propio sistema de ficheros
- Se puede tener un nivel «flexible» y otro «estricto»



4.5 Confiabilidad del sistema de ficheros

- La pérdida de datos es un **desastre importante**
- El S.F. no puede ofrecer protección contra la destrucción física del equipo y los medios, pero sí puede ayudar a proteger la información
- Para mejorar la confiabilidad del sistema de ficheros hay que:
 - Evitar el uso de bloques defectuosos (los que contienen algún sector defectuoso)
 - Evitar la pérdida de datos cuando se estropean bloques sanos o un usuario comete un error
 - Recuperar la consistencia del sistema de ficheros cuando el sistema se cae

4.5 Confiabilidad del sistema de ficheros (II)

- **Manejo de bloques defectuosos.** Tenemos que evitar el uso de bloques defectuosos de disco:
 - **Solución hardware:** el propio disco tiene sectores de reserva y un mapa que asocia, transparentemente, bloques defectuosos a bloques sanos
 - **Solución software:** el S.F. identifica a los bloques defectuosos y evita su uso
 - En Ext2 y NTFS: fichero inaccesible al que pertenecen todos los bloques defectuosos
 - En MS-DOS: se marca el bloque en la FAT como defectuoso

4.5 Confiabilidad del sistema de ficheros (III)

● Copias de seguridad

- Tratan de solucionar dos problemas potenciales:
 - **Recuperarse de un desastre**: un bloque sano, un conjunto de bloques o todo un disco pueden estropearse y perder los datos que contienen
 - **Recuperarse de los errores de los usuarios**: cuando uno de ellos borra «accidentalmente» uno o más ficheros
- Las copias de seguridad pueden serlo bien de todo (total) o bien de lo último que se ha modificado (incremental)
- La **copia de seguridad total**:
 - tarda mucho para dispositivos muy grandes y requiere dispositivos de respaldo de gran capacidad
 - puede enlentecer todo el sistema si la copia se hace durante el funcionamiento de éste (¿es posible?)

4.5 Confiabilidad del sistema de ficheros (III)

- **Copias de seguridad** (continuación...)
 - La **copia de seguridad incremental** necesita apoyo del S.O.:
 - En MS-DOS se activa el atributo A de cada fichero que se modifica. El atributo se desactiva cuando se respalda el fichero
 - En Unix se guarda la fecha de la última modificación para cada fichero que se puede comparar con la fecha de la última copia de seguridad (total o incremental)
 - La restauración se hace con la última copia de seguridad total y las copias incrementales posteriores
 - Los sistemas **RAID** son otra alternativa que veremos en el siguiente tema

4.5 Confiabilidad del sistema de ficheros (IV)

● Consistencia del sistema de ficheros

- Los SS.FF. leen bloques de disco, los modifican en memoria y los escriben en disco después. ¿Qué pasa si el sistema falla cuando se están escribiendo los bloques modificados y hay bloques que se escriben y bloques que no?:
 - El SS.FF. puede quedar en un **estado inconsistente**
 - El problema se agrava si los bloques que no se escriben son de **metadatos** (nodos-i, bloques indirectos, directorios, etc.)
- Una solución: ejecutar un programa cuando se reinicie el sistema para comprobar, al menos, la consistencia de bloques y de ficheros

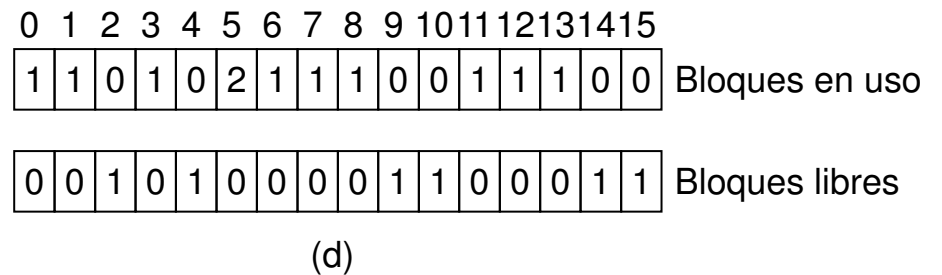
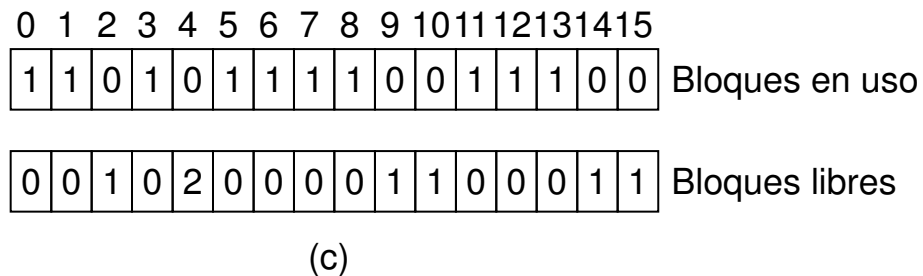
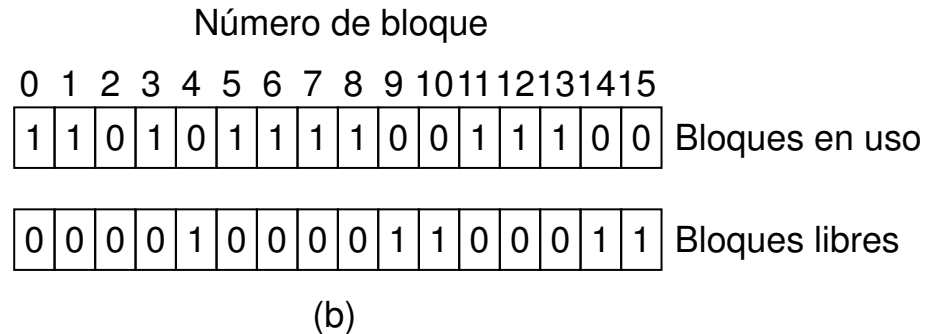
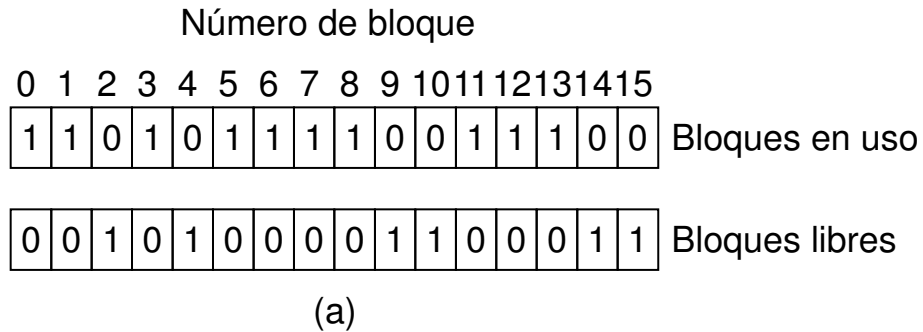
4.5 Confiabilidad del sistema de ficheros (V)

● Consistencia de bloques

- Se tienen 2 contadores inicializados a 0 para cada bloque:
 - Uno cuenta cuántas veces aparece el bloque como ocupado
 - El otro cuenta cuántas veces aparece como libre
- Se recorren todos los nodos-i (en el caso de Unix) para ver qué bloques hay ocupados y se actualiza el primer contador
- Se analiza la lista o mapa de bits de bloques libres y se actualiza el segundo contador
- Casos posibles para cada bloque:
 - 1 en el primer contador o en el segundo \Rightarrow **Bien**
 - 0 en los dos contadores («bloque faltante») \Rightarrow **Añadir a libres**
 - 2 o más veces como libres \Rightarrow **Reconstruir lista enlazada**
 - 2 o más veces en un mismo fichero o en ficheros distintos \Rightarrow ¡Error grave! \Rightarrow **Hacer copias**
 - Ocupado y libre a la vez \Rightarrow ¡Error potencialmente grave! (podría degenerar al caso anterior) \Rightarrow **Quitar de libres**

4.5 Confiabilidad del sistema de ficheros (VI)

● Consistencia de bloques (continuación...)



4.5 Confiabilidad del sistema de ficheros (VII)

● Consistencia de ficheros

- Se comprueba si la información de los directorios es correcta En el caso de Unix:
 - Se tiene un contador de enlaces inicializado a 0 para cada nodo-i
 - Se recorre todo el árbol de directorios y, para cada fichero, se incrementa el contador del nodo-i al que apunta
- Casos posibles para cada nodo-i
 - El n° de enlaces calculado coincide con el del nodo-i \Rightarrow **Bien**
 - El n° de enlaces calculado es menor que el del nodo-i \Rightarrow **Reparar el contador de enlaces del nodo-i y liberar dicho nodo-i si el valor es 0**
 - El n° de enlaces calculado es mayor que el del nodo-i \Rightarrow ¡Error potencialmente grave! (el nodo-i se podría liberar antes de que desaparezcan todos sus enlaces) \Rightarrow **Solución anterior**

4.5 Confiabilidad del sistema de ficheros (VIII)

- **Otras comprobaciones.** En general, es conveniente comprobar todo lo que se pueda como, por ejemplo:
 - Ver si el n° de nodo- i es válido (no es mayor que el total de nodos- i)
 - Comprobar que un nodo- i no almacena números de bloques inválidos
 - Comprobar si los permisos de los ficheros tienen sentido

4.6 Caché de disco. Jerarquía de almacenamiento

● Cachés de disco

- Los dispositivos sobre los que se construyen los SS.FF. suelen ser lentos
- Para mejorar su desempeño, la solución más común es utilizar una porción de la memoria principal para almacenar bloques de disco, conocida como **caché de disco** (o, también, «caché de bloques» o «caché de buffers»)
- Ahora podemos usar como algoritmo de reemplazo un LRU con listas ligadas pero hay bloques que necesitan un tratamiento especial ⇒ **LRU modificado** que tiene en cuenta dos factores independientes:
 1. ¿Es probable que el bloque se vuelva a necesitar pronto? Si no es así, colocar directamente al final de la lista LRU
 2. ¿Es esencial el bloque para la consistencia del SS.FF.? Si es así, escribir en disco lo antes posible

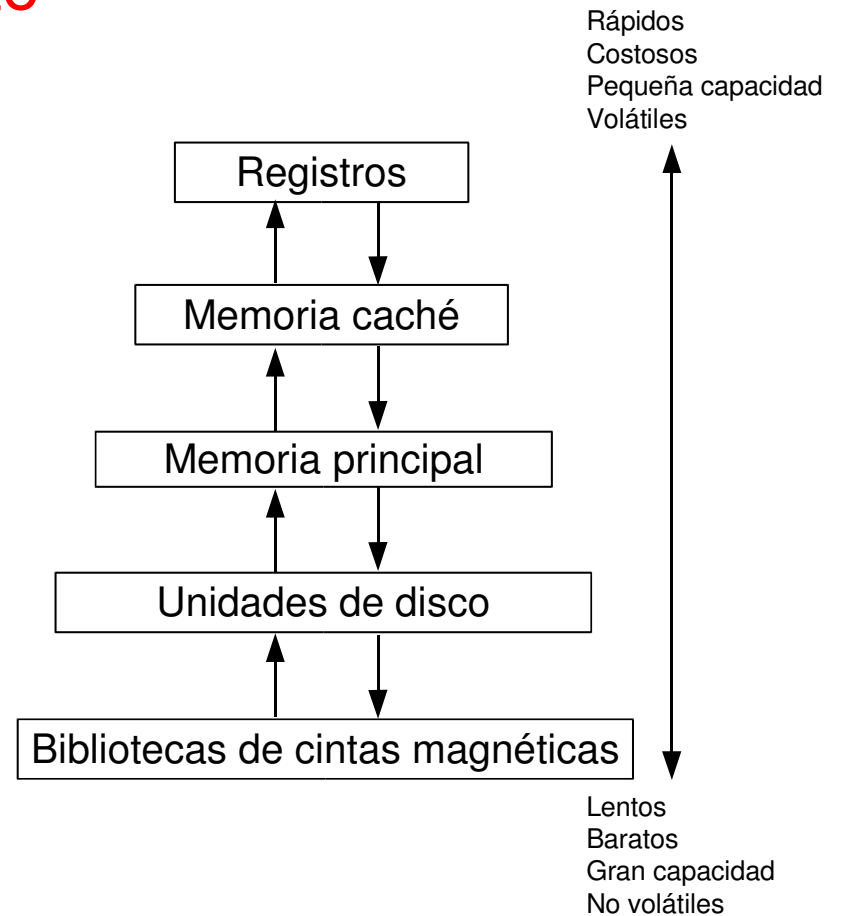
4.6 Caché de disco. Jerarquía de almacenamiento (II)

- **Cachés de disco** (continuación...)
 - En un LRU modificado los bloques se dividen en varias categorías: nodos-i, bloques indirectos, de directorio, parcialmente ocupados, totalmente ocupados, etc.
 - Los «bloques de datos» tampoco pueden permanecer indefinidamente en memoria (los usuarios podrían perder información importante). 2 soluciones son:
 - **Caché de escritura directa** (*write-through*): cualquier modificación se escribe inmediatamente en disco. Es útil para dispositivos pequeños extraíbles (disquetes)
 - **Caché de escritura diferida** (*write-back*): los bloques modificados de caché se escriben periódicamente en disco (por ejemplo, en Linux se escriben cada 30 segundos)

4.6 Caché de disco. Jerarquía de almacenamiento (III)

● Jerarquía de almacenamiento

- Los discos son sólo uno de los muchos sistemas de almacenamiento que hay, pero también existen: registros, memoria principal, cintas, etc.
- Una forma adecuada de organizar todos esos elementos es **jerárquicamente**, en función de la velocidad, coste, tamaño y volatilidad de cada elemento



4.7 Sistemas de ficheros modernos

- Los sistemas de ficheros modernos se enfrentan a dos grandes problemas:
 - La gran capacidad de los dispositivos de almacenamiento
 - Su lentitud
- La gran capacidad de almacenamiento obliga a dichos SS.FF. a utilizar **estructuras de datos escalables** (árboles B+, principalmente) y **transacciones** para poder recuperar rápidamente la consistencia tras una caída
- La lentitud determina, en gran medida, la estructura global del sistema de ficheros y su funcionamiento interno ⇒ se deben **optimizar las lecturas y escrituras** que deben ser secuenciales y en grandes grupos si es posible
- Como ejemplo, veamos el mecanismo de transacciones

4.7 Sistemas de ficheros modernos (II)

- **Recuperación rápida de la consistencia**
 - Para no perder la consistencia del S.F. las modificaciones tienen que ser atómicas:
 1. Las modificaciones se guardan primero en un fichero especial, llamado **registro**, **bitácora** o *journal*
 2. Cuando dicho fichero está a salvo en disco, se escriben en disco las modificaciones del propio sistema de ficheros
 3. Si el sistema cae en el primer punto se pierden modificaciones (pero ninguna queda a medio)
 4. Si cae en el segundo, se rehacen las modificaciones del registro (que deben ser idempotentes)
 - El tiempo de recuperación depende del tamaño del registro, que suele ser muy pequeño (32 MB, 64 MB, etc.)
 - Ejemplos de **sistemas de ficheros transaccionales**: Ext3, ReiserFS, XFS, JFS y NTFS

Índice

- 5. Sistemas de ficheros en Linux (Tanenbaum [C10.6])
 - 5.1. Conceptos fundamentales
 - 5.2. El sistema de ficheros virtual (VFS)
 - 5.3. Montaje de sistemas de ficheros
 - 5.4. Ext2

5.1 Conceptos fundamentales

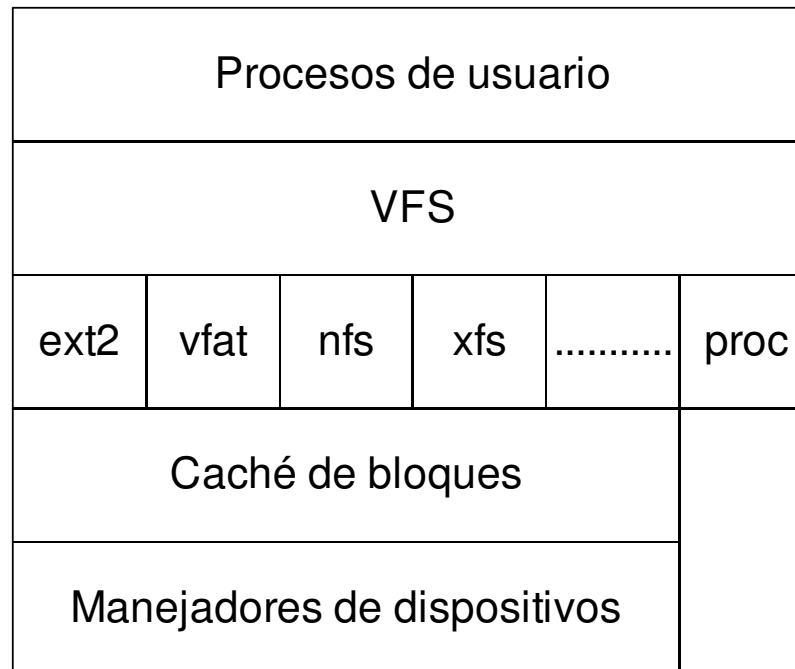
- Los **ficheros**:
 - son una **sucesión de 0 o más bytes** que contienen **información arbitraria** cuyo significado establece el creador
 - Se identifican mediante **nombres arbitrarios** compuestos de cualquier carácter (excepto «\0») y en donde la extensión no tiene ningún significado especial para el S.O.
- Existe un **sistema de ficheros jerárquico** con estructura de grafo acíclico:
 - Los ficheros y directorios se especifican mediante **rutas absolutas y relativas** (éstas últimas, dependientes del **directorio de trabajo actual**)
 - Es posible crear **enlaces físicos** (para cualquier tipo de fichero salvo directorios) y **enlaces simbólicos** (para cualquier tipo de fichero)

5.1 Conceptos fundamentales (II)

- Cuando varios procesos usan un mismo fichero:
 - Se usa la **semántica Unix** de contención ⇒ los cambios son inmediatos
 - No obstante, es posible establecer **bloqueos** (mediante la función `fcntl`) para un rango de bytes de un fichero. Los bloqueos pueden ser:
 - compartidos: permiten establecer otro bloqueo compartido sobre el mismo rango de bytes o sobre un rango de bytes que se solape con otro que ya tenga un bloqueo compartido ⇒ Útil para **operaciones de lectura**
 - exclusivos: los bytes de un rango afectado por un bloqueo exclusivo no pueden pertenecer a ningún otro bloqueo, ni exclusivo ni compartido ⇒ Útil para **operaciones de escritura**
 - La propia operación de bloqueo puede ser bloqueante o no, según se desee

5.2 El sistema de ficheros virtual

- Linux nos permite acceder a varios sistemas de ficheros a la vez de forma transparente
- Para ello implementa un **sistema de ficheros virtual** (*Virtual File System, VFS*) que invoca a funciones específicas de los sistemas de ficheros reales cuando lo necesita



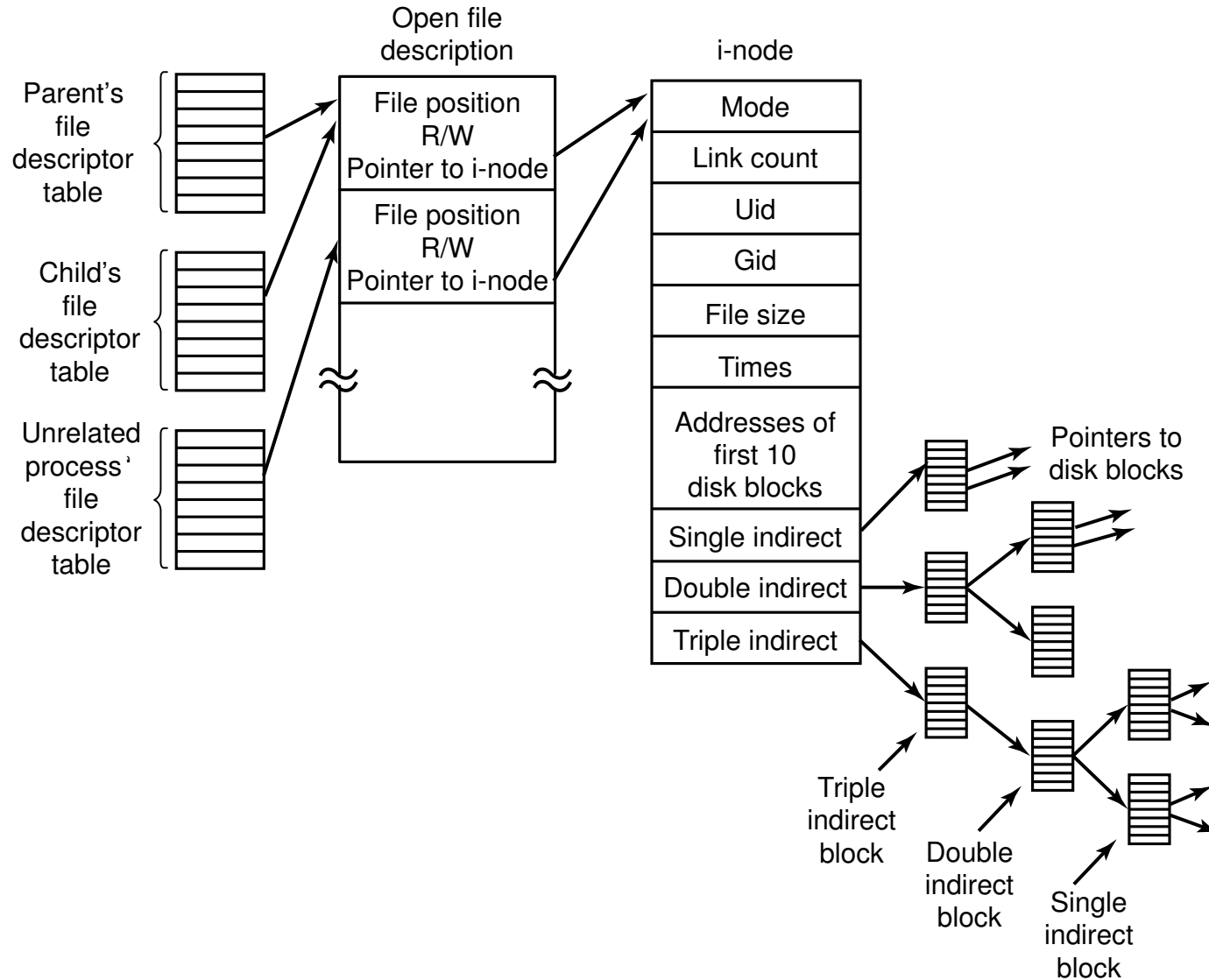
5.2 El sistema de ficheros virtual (II)

- El VFS contiene código genérico que implementa la interfaz de llamadas al sistema relacionadas con los sistemas de ficheros, mientras que los SS.FF. reales contienen código específico de cada sistema de ficheros soportado
- Los SS.FF. reales acceden a los dispositivos de almacenamiento a través de la **caché de buffers** que es una capa de abstracción que aísla a dichos SS.FF. de los detalles del HW
- En la interfaz de llamadas al sistema, las operaciones con ficheros y directorios se realizan mediante **descriptores de fichero**
 - Estos descriptores son enteros que se utilizan como índices en la **tabla de descriptores de fichero** que posee cada proceso
 - Cada entrada de esta tabla posee información suficiente para acceder al nodo-i asociado al fichero deseado

5.2 El sistema de ficheros virtual (III)

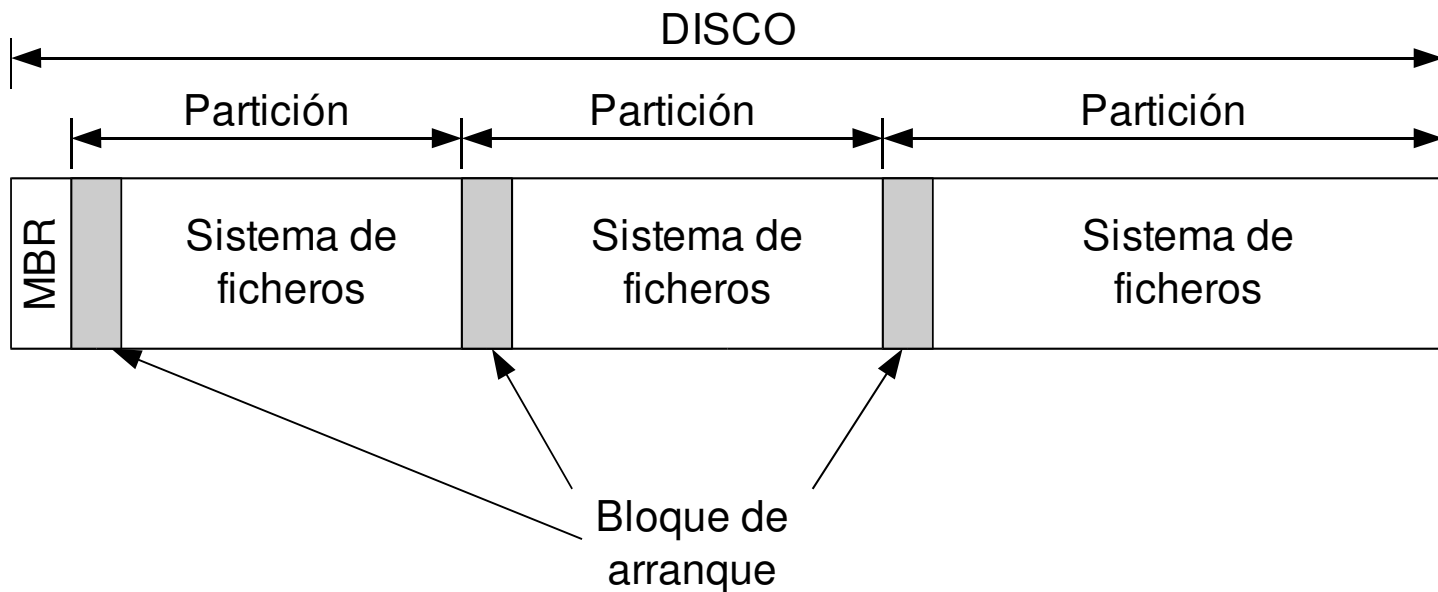
- El VFS mantiene una **tabla de nodos-i virtuales** (una especie de caché) en donde se coloca la información de los nodos-i reales de los distintos ficheros abiertos
- En una llamada al sistema, el núcleo debe poder acceder a un fichero a partir de un descriptor:
 - Solución: que las entradas de las tablas de descriptors de fichero apunten a las entradas de la tabla de nodos-i virtuales
 - **Problema:** ¿dónde se coloca el puntero de lectura/escritura?
 - En el nodo-i virtual \Rightarrow MAL: dos procesos independientes con el mismo fichero abierto comparten el puntero cuando no deben
 - En la entrada de la tabla de descriptors de fichero asociada al descriptor usado por el proceso \Rightarrow MAL: un proceso padre y otro hijo deben poder compartir el puntero
 - **Solución:** una nueva estructura de datos, la **tabla de descripción de ficheros abiertos**, entre la tabla de descriptors de fichero de cada proceso y la tabla de nodos-i del VFS

5.2 El sistema de ficheros virtual (IV)



5.3 Montaje de sistemas de ficheros

- Linux puede utilizar varios dispositivos de almacenamiento a la vez
- Para utilizar un dispositivo lo habitual es crear en él un S.F.
- Algunos dispositivos de almacenamiento se pueden dividir en «dispositivos lógicos» más pequeños. Por ejemplo, un disco duro puede dividirse en **particiones**:



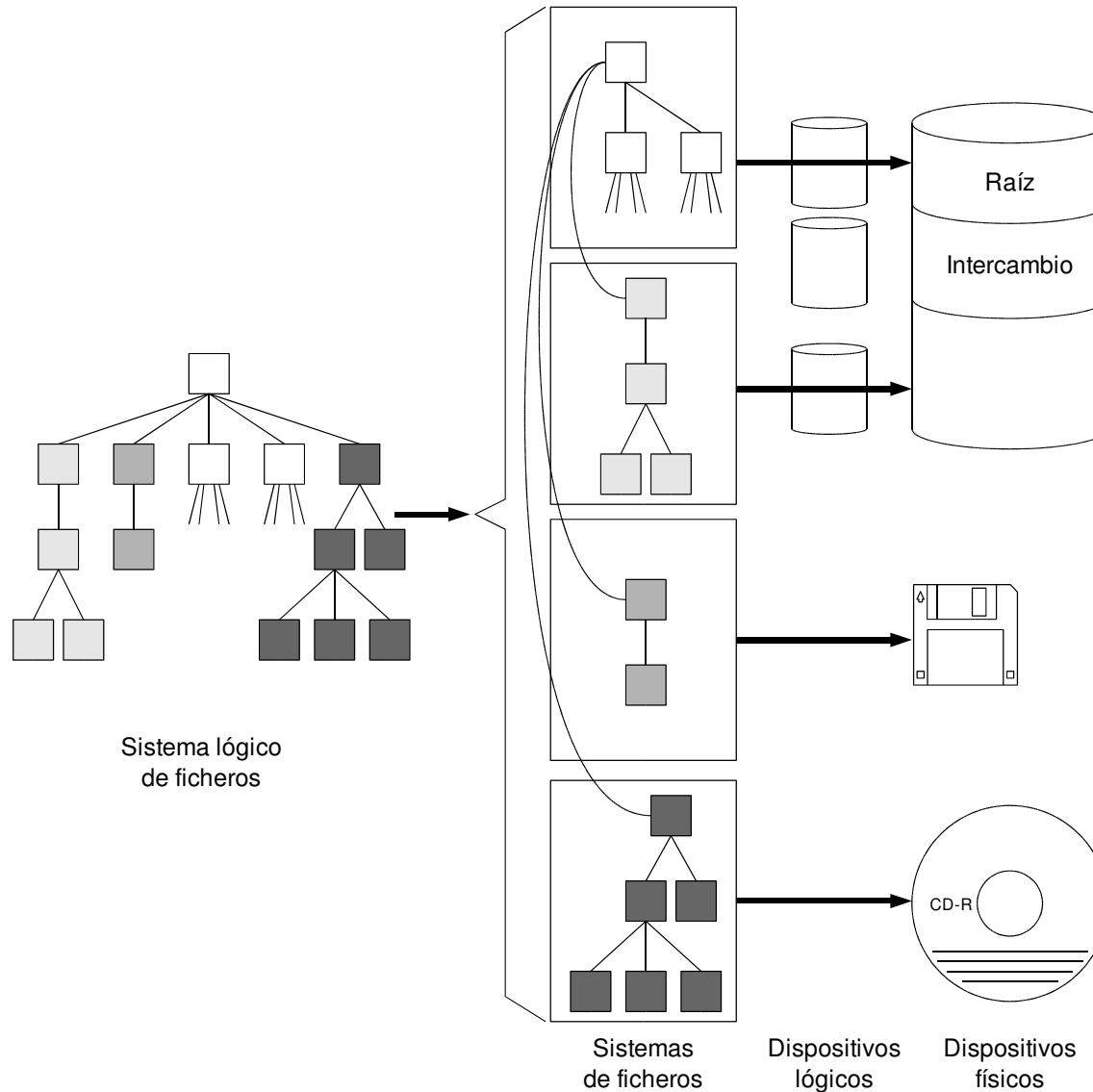
5.3 Montaje de sistemas de ficheros (II)

- Particiones (continuación...)
 - La información sobre particiones se guarda en la **tabla de particiones** del primer sector del disco duro, conocido como **MBR** (*Master Boot Record*)
 - El MBR suele contener también el código inicial de arranque del ordenador
 - Para usar una partición lo habitual es crear en ella un S.F., reservando el primer bloque como «bloque de arranque», que puede contener el código para cargar el S.O. de la partición
 - Cuando un ordenador arranca desde el disco duro, el MBR determina la **partición activa**, carga en memoria el código de su «bloque de arranque» y le cede el control

5.3 Montaje de sistemas de ficheros (III)

- Aunque Linux puede usar varios dispositivos de almacenamiento a la vez, sólo existe una «única jerarquía de directorios» o **sistema lógico de ficheros**
- Para usar un dispositivo, Linux debe «insertar» su sistema de ficheros en el sistema lógico de ficheros. Dicha operación se llama **montaje**
- Un S.F. se monta en un **punto de montaje**, un directorio bajo el cuál se hace visible el S.F. montado, pasando así a formar parte de la jerarquía única de directorios
- Cuando ya no se quiere utilizar un dispositivo de almacenamiento se **desmonta**, haciendo que su sistema de ficheros deje de ser accesible
- El S.F. que se monta durante el arranque del S.O y sobre el que se montan todos los demás SS.FF. se conoce como **sistema de ficheros raíz**

5.3 Montaje de sistemas de ficheros (IV)



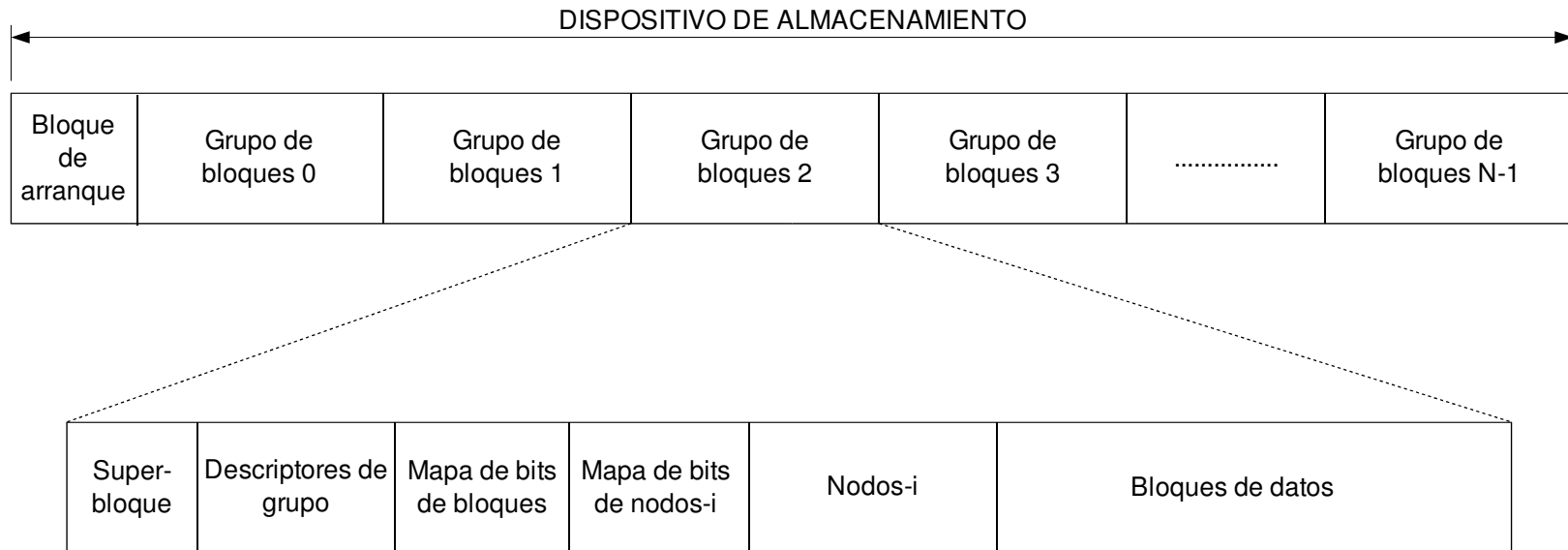
5.3 Montaje de sistemas de ficheros (V)

- **Funcionamiento del montaje:**
 - Cuando se resuelve una ruta (absoluta o relativa) se comprueba si sobre un directorio hay un S.F. montado, es decir, si es un punto de montaje o no (hay información en el nodo-i virtual que indica esto)
 - Si es así se localiza, a través de una **tabla de montajes**, el nodo-i raíz del S.F. montado, y se resuelve el resto de la ruta dentro de ese S.F.
 - Por el contrario, si el siguiente elemento de la ruta a resolver es el directorio «..» y estamos en la raíz de un S.F. montado, se usa la tabla de montajes para acceder al nodo-i del punto de montaje, el cuál se usa para resolver el resto de la ruta
 - Cuando se resuelve una ruta es posible atravesar varios puntos de montaje. El procedimiento que se sigue es el mismo que el descrito

5.4 Ext2

- Ext2 es el sistema de ficheros por defecto de Linux. Sus principales características son:
 - Nombres largos (hasta 255 caracteres)
 - Ficheros grandes (más de 4 TBytes)
 - Direcciones de bloque lógico de 4 bytes
 - Buen rendimiento
- Ext2 divide el dispositivo de almacenamiento en **grupos de bloques**. Cada grupo contiene:
 - Una copia del superbloque
 - Una copia de los descriptores de grupo
 - Un bloque para el mapa de bits de bloques
 - Un bloque para el mapa de bits de nodos-i
 - Varios bloques para nodos-i
 - Bloques de datos hasta completar el grupo

5.4 Ext2 (II)



- El **superbloque** contiene información sobre la estructura general del sistema de ficheros:
 - Tamaño del bloque lógico (en logaritmo en base 2)
 - Total de nodos-i y número de nodos-i libres
 - Total de bloques de datos y número de bloques libres
 - Tamaño de grupo (en bloques)
 - Nodos-i por grupo

5.4 Ext2 (III)

- Cada grupo tiene un **descriptor de grupo** asociado que indica:
 - Posición de cada mapa de bits (de bloques y de nodos-i)
 - Primer bloque de la tabla de nodos-i
 - Contador de nodos-i libres y contador de bloques libres
 - Contador de directorios asociados al grupo
- Ext2 utiliza el último contador de los descriptores de grupo para repartir los directorios de manera uniforme por todo el disco
- Ext2 realiza una **agrupación de ficheros basada en directorios**: a cada nuevo directorio se le asigna un grupo en el que se intentan situar el nodo-i y los bloques de cualquier fichero creado dentro de ese directorio
- Los **nodos-i** tienen un tamaño de 128 bytes y almacenan 15 direcciones: 12 de bloques directos y 3 de bloques indirectos (1 BSI, 1 BDI y 1 BTI). El nodo-i 2 es el del **directorio raíz**

5.4 Ext2 (IV)

- Un sistema de ficheros Ext2 se crea con **mke2fs**: es una aplicación de usuario que recibe como parámetro el fichero especial de bloques que representan al dispositivo de almacenamiento (p.e., /dev/hda1)
- Para recuperar la consistencia tras una caída se usa **e2fsck** que también es una aplicación de usuario
- **Ext3** es un sistema de ficheros transaccional derivado de Ext2:
 - Comparten la misma estructura ⇒ un sistema de fichero Ext2 se puede tratar como Ext3 y viceversa
 - Ext3 añade un registro que permite recuperar rápidamente la consistencia tras una caída
 - También se crea con `mke2fs` y, si se desea, se puede comprobar totalmente con `e2fsck`

Índice

- 6. Sistemas de ficheros en Windows 2000 (Tanenbaum [C11.7])
 - 6.1. Conceptos fundamentales
 - 6.2. NTFS

6.1 Conceptos fundamentales

- Muchos conceptos son similares a los vistos en Linux, como un sistema de ficheros jerárquico, pero otros son específicos, como los *flujos* dentro de los ficheros
- Windows 2000 también puede utilizar varios sistemas de ficheros a la vez: FAT16, FAT32, NTFS y otros para el manejo de CD-ROMs y DVDs. El principal S.F. es **NTFS**
- Los nombres de los ficheros están limitados a 255 caracteres y las rutas completas a 32.767 caracteres
- El API Win32 para manejo de ficheros **no distingue entre mayúsculas y minúsculas** (aunque NTFS sí lo hace)
- Los ficheros se manejan a través de *handles*. Para las aplicaciones de consola hay, por defecto, tres handles abiertos (entrada, salida y salida de error estándares). Esto no ocurre para las aplicaciones gráficas

6.2 NTFS

- NTFS es el principal S.F de Windows NT, 2000, XP y 2003. Sus principales características son:
 - Nombres largos (hasta 255 caracteres) en formato Unicode
 - Ficheros muy grandes (en teoría, hasta 2^{64} bytes)
 - Ficheros multiflujo (un ficheros es uno o más flujos de bytes a los que se puede acceder de manera individual)
 - Direcciones de bloque lógico de 8 bytes y tamaños de bloque lógico entre 512 bytes y 64 Kbytes
 - Compresión transparente de ficheros
 - Cifrado de ficheros
 - Rendimiento adecuado

6.2 NTFS (II)

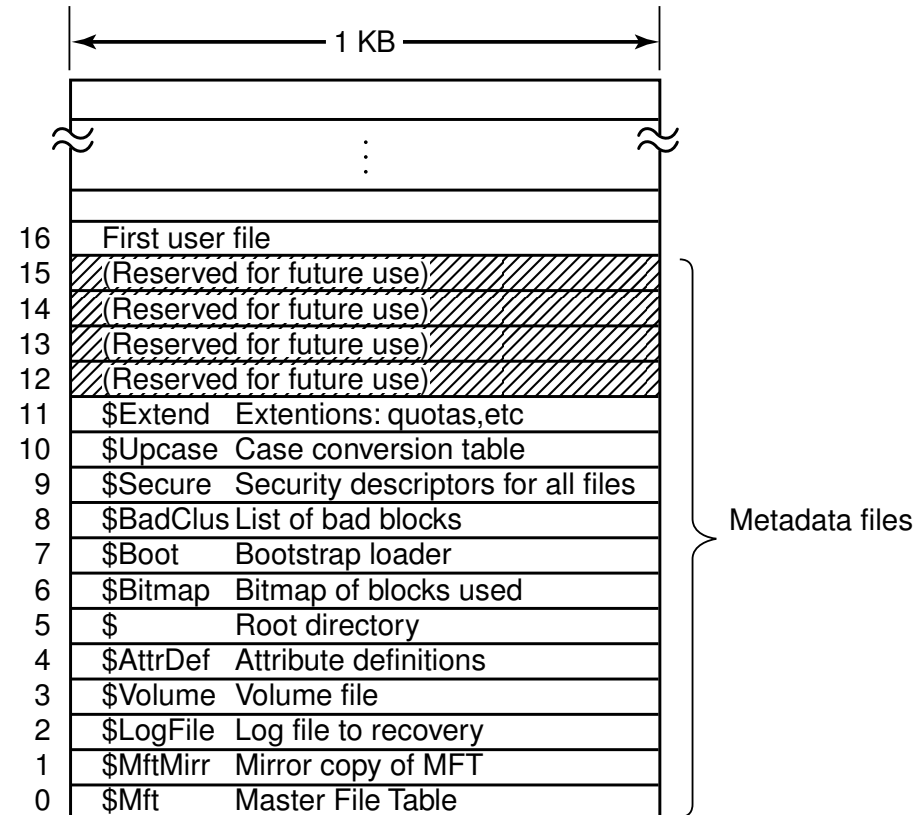
- **Un fichero en NTFS consiste en múltiples atributos**, cada uno de los cuales se representa con un flujo de bytes cuyo tamaño dependen del tipo de atributo (hasta un máximo de 2^{64} bytes)
- Todos los ficheros tienen, al menos, dos atributos: un **nombre de fichero** y un **flujo anónimo** que contiene los datos del fichero
- Además del flujo anónimo, los ficheros pueden tener otros flujos de datos, cada uno de los cuales se identifica con «fichero:nombre del flujo»
- Cada flujo tiene su propio tamaño y puede bloquearse con independencia de los demás flujos

6.2 NTFS (III)

- La principal estructura de datos de un sistema de ficheros NTFS es la **tabla maestra de ficheros** o **MFT** (Master File Table):
 - Es una sucesión lineal de registros de tamaño fijo (1 KB)
 - La MFT es en sí un fichero: puede colocarse en cualquier lugar y puede crecer cuando es necesario (hasta un máximo de 2^{48} registros)
 - La dirección del primer bloque de la MFT se guarda en el bloque de arranque. La información sobre el resto de bloques se guarda en el primer registro (0) de la propia MFT
 - Un mapa de bits controla qué entradas de la MFT está ocupadas y qué entradas están libres

6.2 NTFS (IV)

- Cada registro de la MFT describe un **fichero o directorio**: contiene sus atributos (nombre, marcas de tiempo, flujo anónimo, ...) y la lista de direcciones de disco donde están sus bloques
- Los primeros 16 registros MFT están reservados para **ficheros de metadatos** (tal como indican el signo \$ al principio del nombre)
- El registro 1 es una copia de la primera parte de la MFT (para evitar su pérdida total, lo que sería un desastre)



6.2 NTFS (V)

- Algunos ficheros de metadatos importantes, descritos por los primeros 16 registros, son:
 - **\$LogFile**: fichero de registro para los cambios en los metadatos
 - **\$AttrDef**: definiciones de los atributos contenidos en los registros
 - **\$**: directorio raíz
 - **\$Bitmap**: mapa de bits de bloques usados en el dispositivo de almacenamiento (o volumen)
 - **\$BadClus**: lista de bloques defectuosos del dispositivo de almacenamiento (o volumen)
 - **\$Extend**: a diferencia de los demás, este registro es un directorio que contiene ficheros diversos (cuotas, puntos de reanálisis, etc.)

6.2 NTFS (VI)

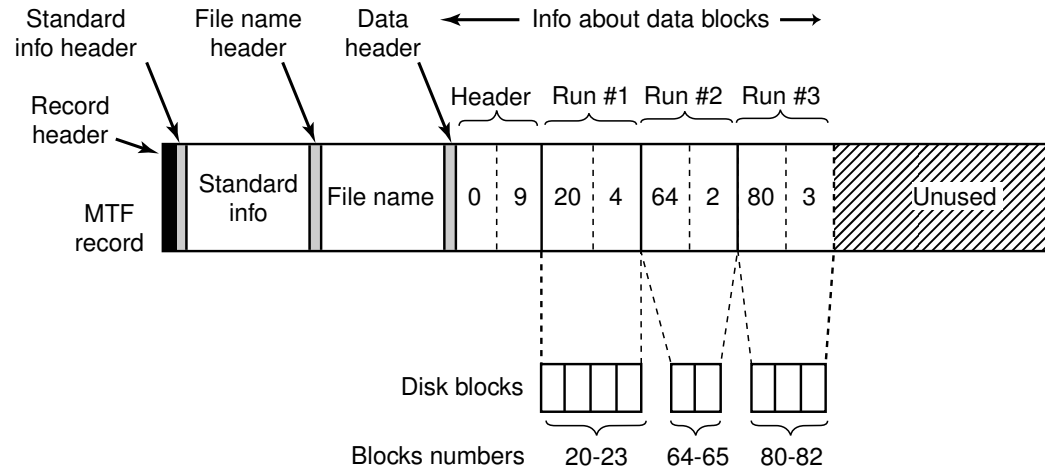
- Cada registro MFT consiste en:
 - un «encabezado de registro»: entre otras cosas, contiene un contador de enlaces físicos y el número real de bytes usados en el registro
 - una secuencia de pares («encabezado de atributo», «valor»):
 - el «encabezado de atributo» identifica el atributo al que representa, y da la longitud y la ubicación del campo «valor» juntos con diversos indicadores y otra información
 - NTFS define 13 atributos posibles, siendo dos de los más importantes el «nombre del fichero» y el atributo «datos»
 - Por lo general, los valores de atributo vienen inmediatamente después del encabezado correspondiente
 - No obstante, si un valor es demasiado grande y no cabe en un registro se puede colocar en un bloque de disco aparte
⇒ atributo no residente

6.2 NTFS (VII)

- De los 13 atributos posibles, el que siempre aparece en último lugar es el **atributo «datos»**. Este atributo se puede repetir varias veces, una por cada «flujo» que posea el fichero:
 - El nombre del flujo va en el encabezado de este atributo (salvo para el flujo por defecto, el «flujo anónimo»)
 - Tras el encabezado viene una lista de direcciones de los bloques de disco que contienen los datos del flujo (o bien, si el flujo es pequeño, el flujo mismo):
 - La lista de direcciones se describe mediante una secuencia de registros, cada uno de los cuales describe una serie de bloques lógicamente contiguos
 - Al igual que en Ext2, los ficheros de NTFS pueden tener «huecos» (que, se supone, están llenos de ceros). En este caso, habrá varios registros para describir las direcciones de disco

6.2 NTFS (VIII)

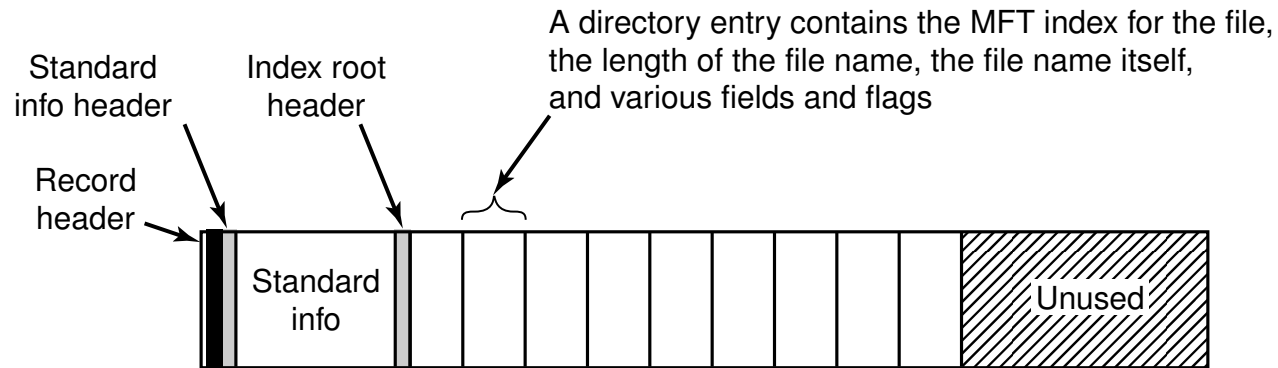
- **Ejemplo de registro MFT** para un fichero de 9 bloques repartidos en 3 series:



- Si el fichero es tan grande o está tan fragmentado que la información de bloques no cabe en su registro MFT (**registro base**), se usan otros registros MFT (**registros de extensión**) para guardar el resto de la información:
 - En este caso, el registro base tiene un atributo con la lista de registros de extensión
 - Si esta lista es muy grande, dicho atributo se puede hacer no residente

6.2 NTFS (IX)

- En NTFS los **directorios pequeños** se implementan como ficheros que contienen una lista desordenada de entradas (al igual que en Ext2):



- Los **directorios grandes**, en cambio, se implementan como **árboles B+**
- El **número de registro MFT** desempeña un papel similar al de número de nodo-i en otros sistemas de ficheros