



UT4: Gestión de la Información Sistemas de Ficheros

Sistemas Informáticos

Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web

Universidad Católica San Antonio de Murcia - Tlf: (+34) 968 27 88 00
info@ucam.edu - www.ucam.edu

Índice

- ◆ Introducción
- ◆ Ficheros
- ◆ Directorios
- ◆ Implementación del sistema de ficheros
- ◆ Sistemas de ficheros en Linux
- ◆ Sistemas de ficheros en Windows 2000



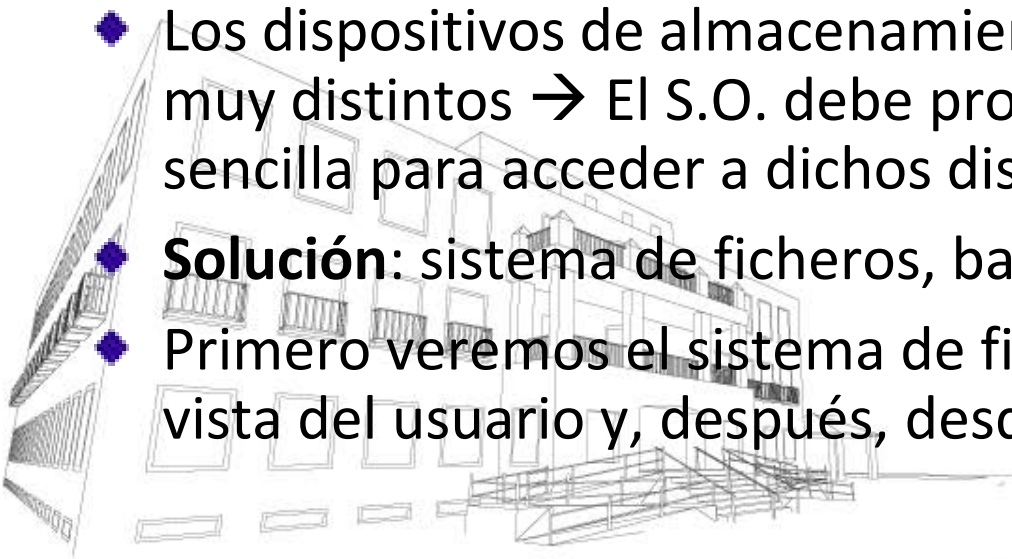
Bibliografía

- ◆ Tanenbaum, A.S. Sistemas operativos modernos. 3ª Edición. Prentice-Hall. ISBN 978-6074420463



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 → 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 punto de vista del usuario y, después, desde el punto de vista del S.O.



Índice

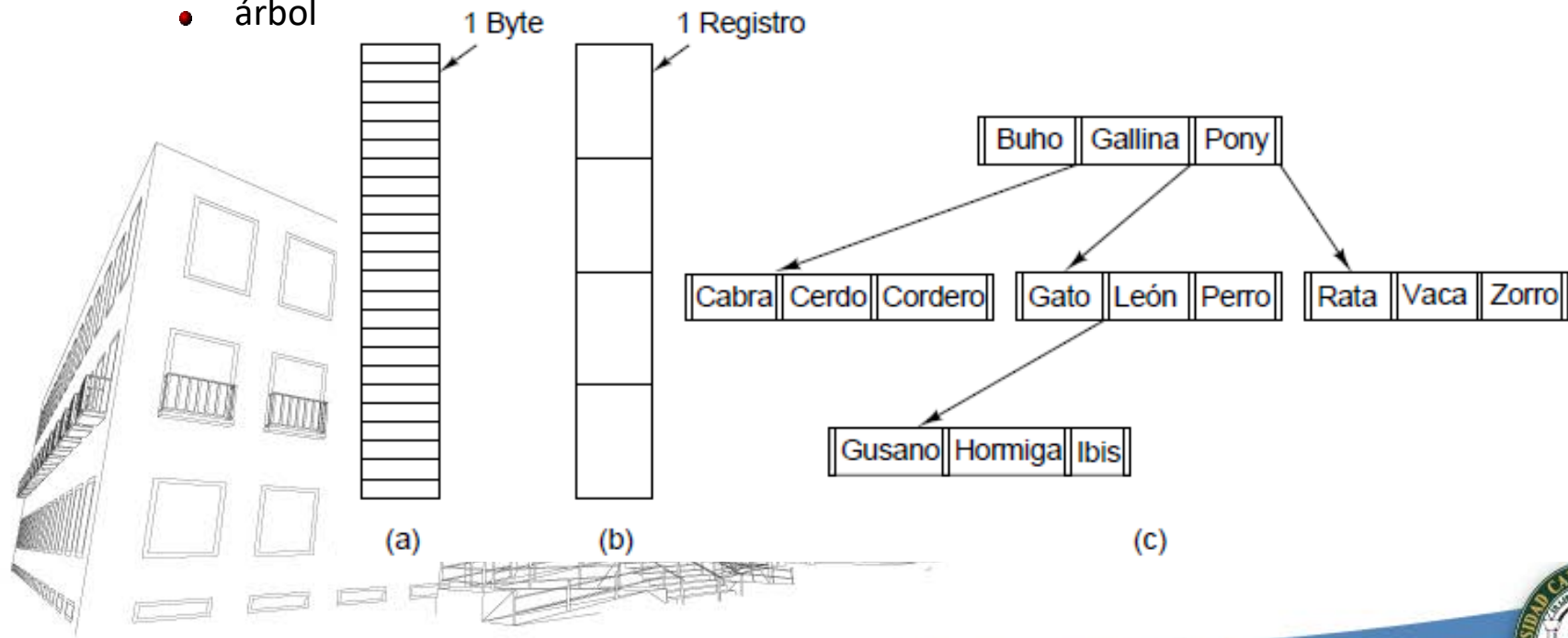
◆ Ficheros (Tanenbaum [C6.1])

- Concepto de fichero
- Estructura de un fichero. Tipos de ficheros
- Acceso a un fichero
- Atributos de un fichero
- Operaciones con ficheros
- Ficheros proyectados en memoria



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



Tipos de ficheros y accesos

Tipos de ficheros

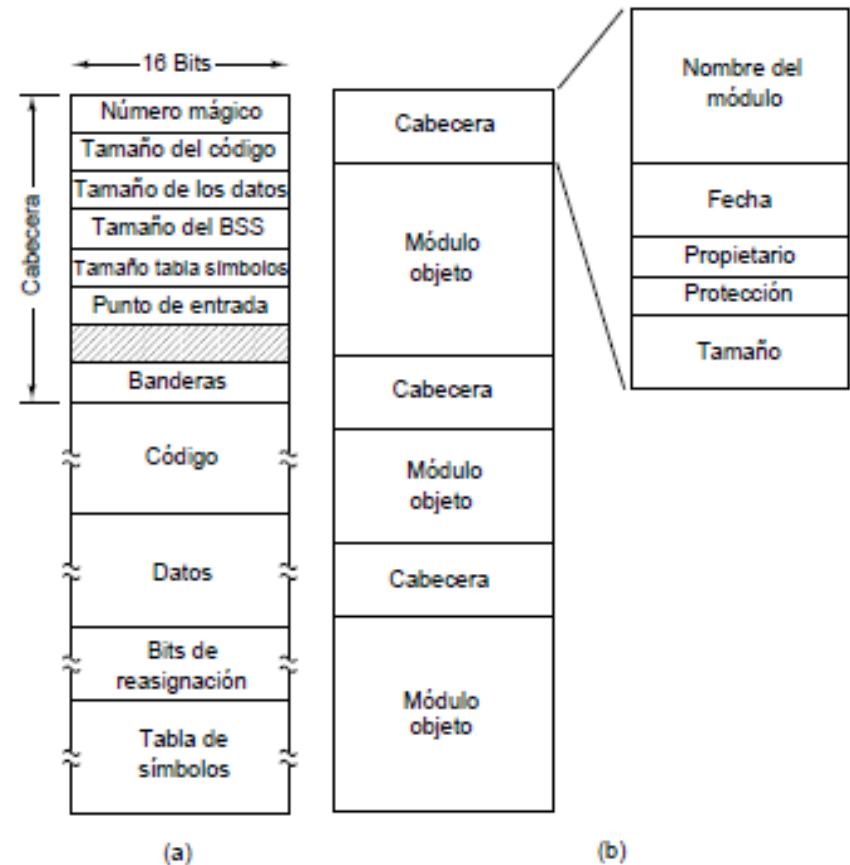
- Ficheros normales o regulares (ASCII o binarios)
- Directorios (mantener estructura del SSFF)
- Ficheros especiales de caracteres (I/O)
- Ficheros especiales de bloques (modelar discos)

Fichero ejecutable (programa)

- Formato específico interpretable por el S.O.

Acceso:

- Secuencial: todos los bytes y registros del fichero se leen en orden sin saltos.
- Aleatorio: e.g., ficheros almacenados en disco (fragmentación)



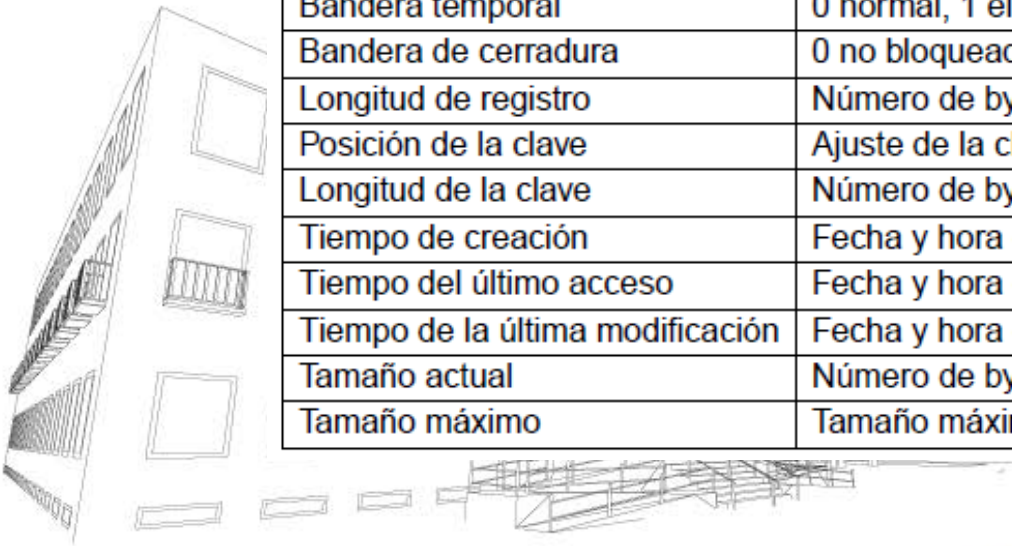
(a) Fichero ejecutable (b) Biblioteca

Atributos de un fichero

Campo

Significado

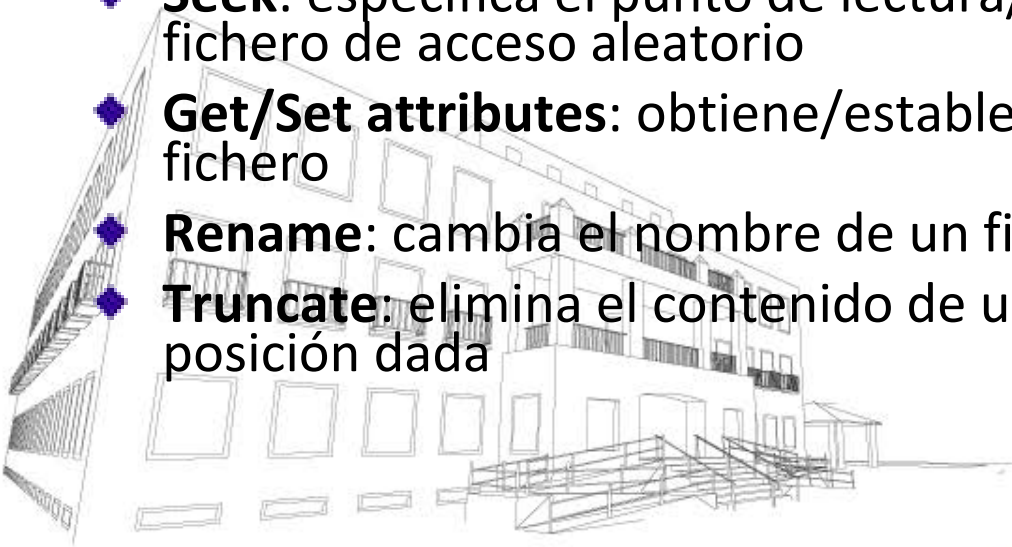
Protección	Quién debe tener acceso y de qué forma
Contraseña	Contraseña necesaria para tener acceso al fichero
Creador	Identificador de la persona que creó el fichero
Propietario	Propietario actual
Bandera «sólo lectura»	0 Lectura/escritura, 1 para lectura exclusivamente
Bandera de ocultación	0 normal, 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



Operaciones con ficheros

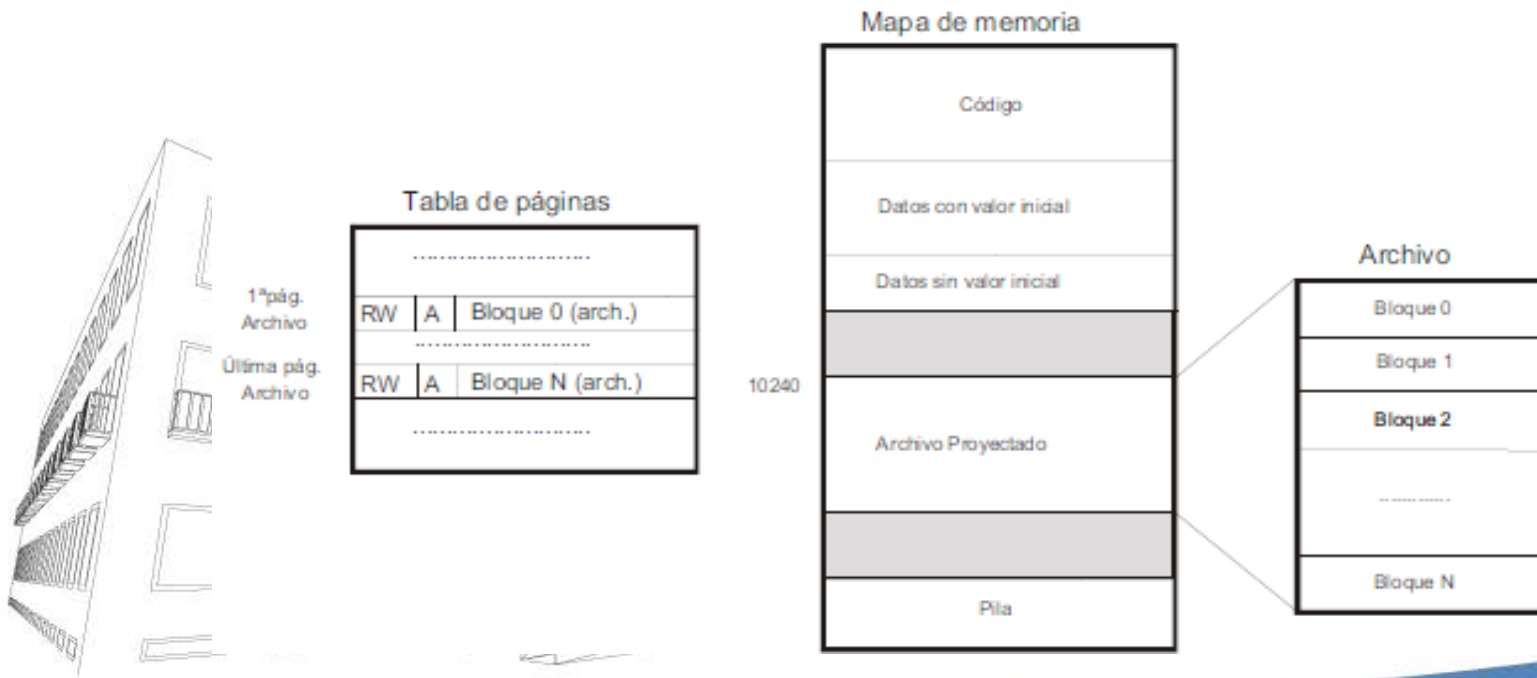
Llamadas al Sistema (syscalls)

- ◆ **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



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 (**instrucciones load/store**)



Índice

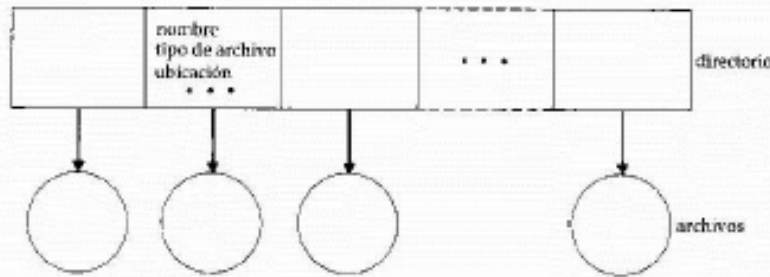
- ◆ Directorios (Tanenbaum [C6.2])
 - Sistemas jerárquicos de directorios
 - Nombre de la ruta de acceso
 - Operaciones con directorios



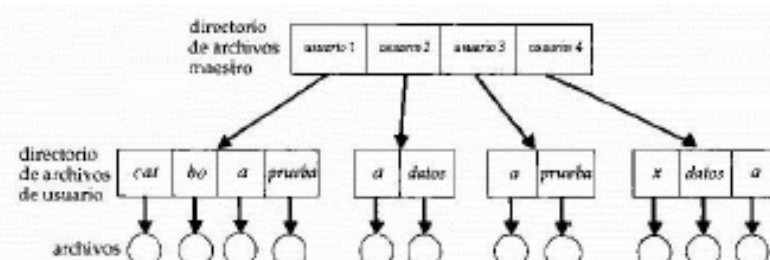
Directorios (o Carpetas)

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

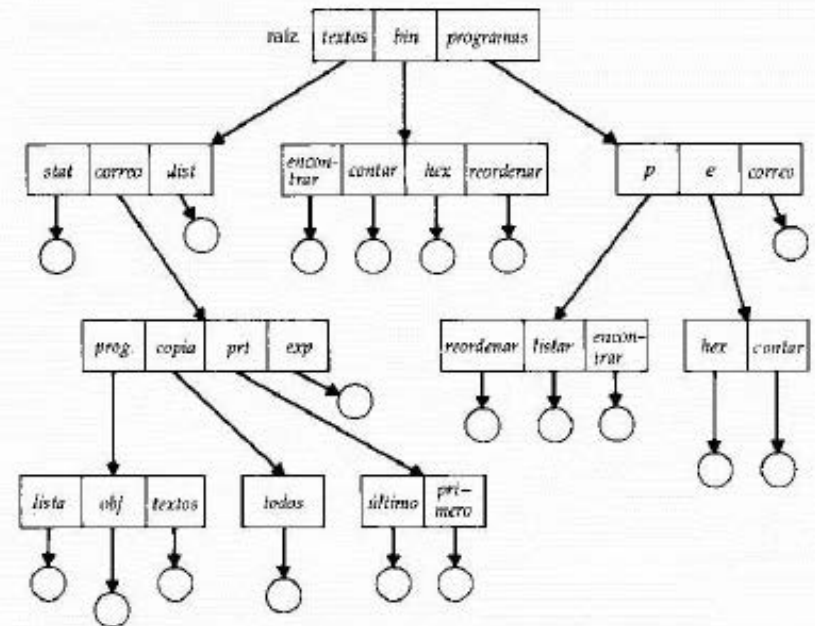
Directorio único



Un directorio por usuario



Sistema jerárquico de directorios



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*

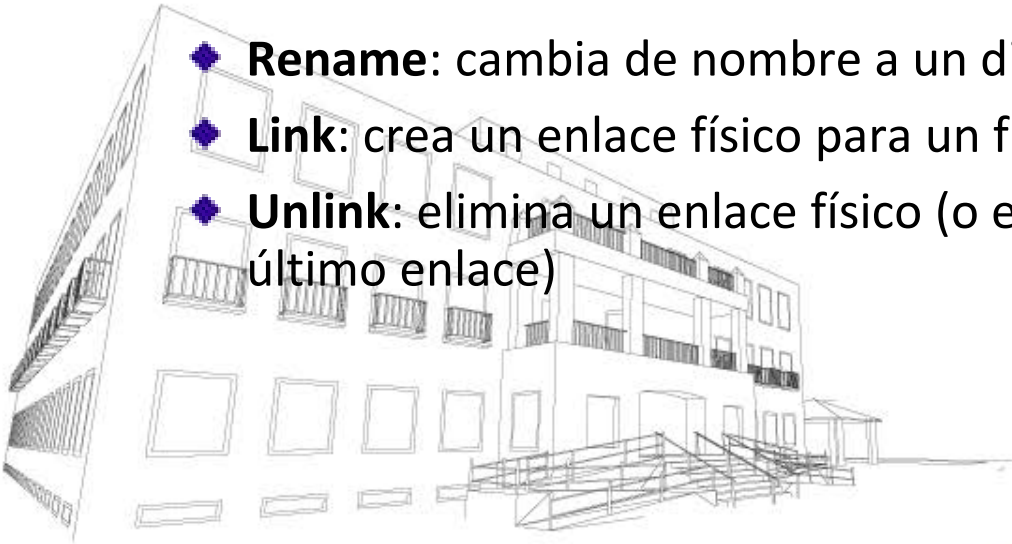
◆ Dos directorios especiales:

- Directorio «.»: directorio actual
- Directorio «..»: directorio padre

Operaciones con directorios

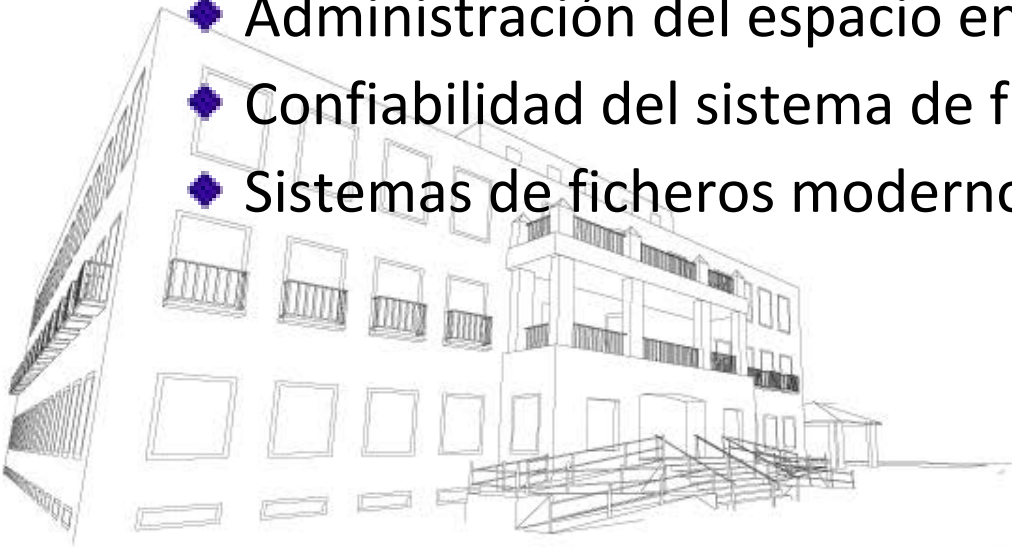
Llamadas al sistema (syscalls)

- ◆ 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)



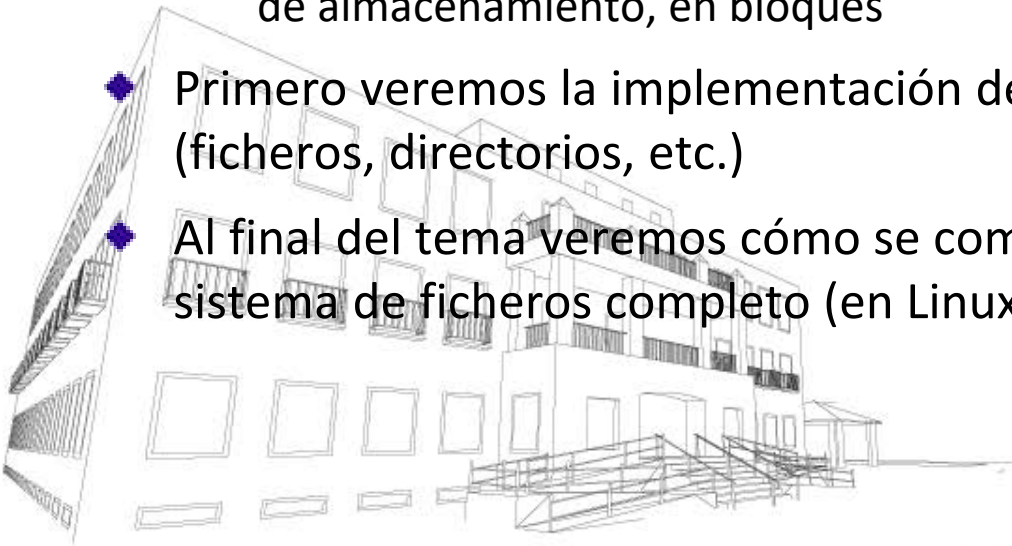
Índice

- ◆ Implementación del sistema de ficheros (Tanenbaum [C6.3])
 - ◆ Implementación de ficheros
 - ◆ Implementación de directorios
 - ◆ Ficheros compartidos
 - ◆ Administración del espacio en disco
 - ◆ Confiabilidad del sistema de ficheros
 - ◆ Sistemas de ficheros modernos



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)



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

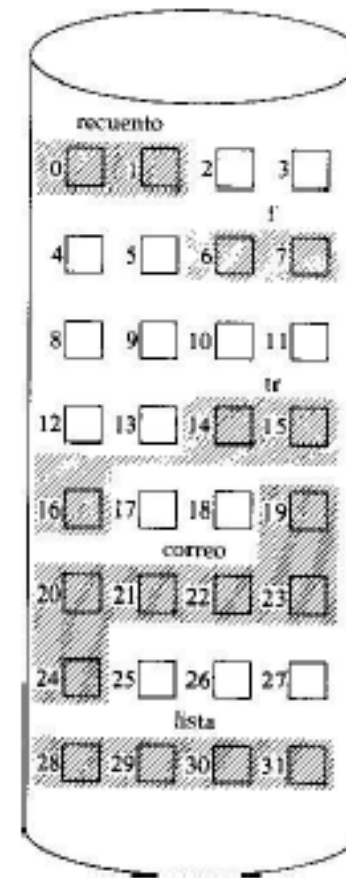
Fácil implementación → Registro: bloque inicial

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

Implementación de ficheros (II)

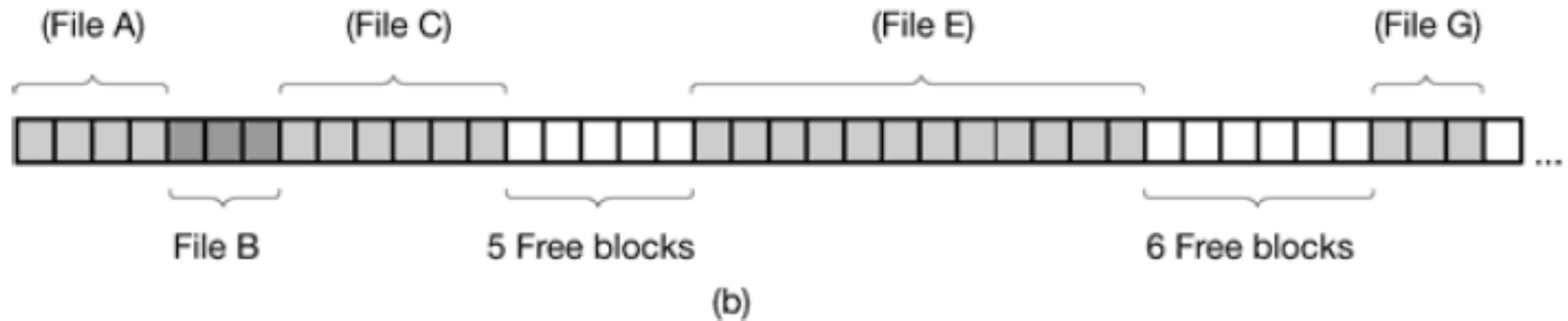
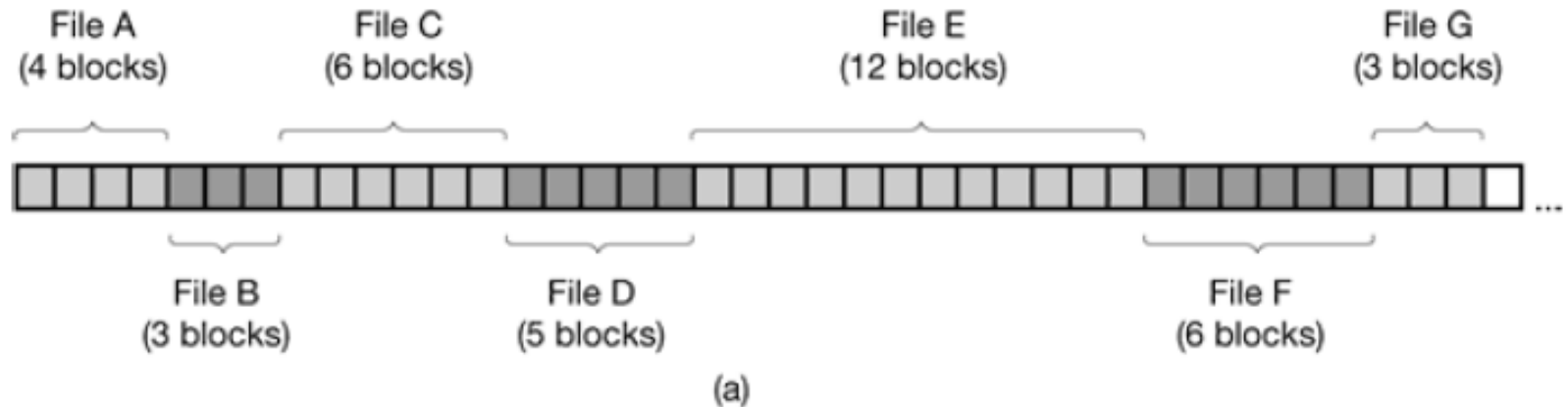
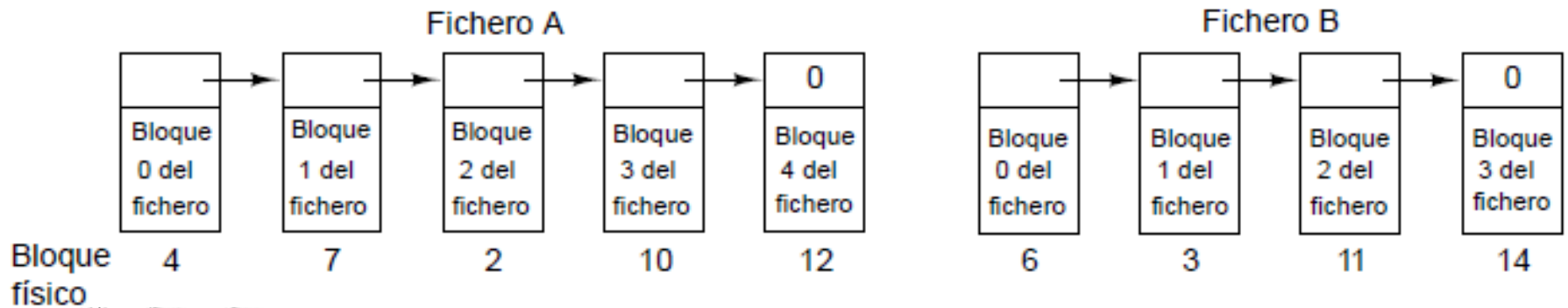


Figure 6-12. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Implementación de ficheros (II)

◆ Asignación con lista enlazada



- 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 (el puntero necesita unos pocos bytes)

Implementación de ficheros (III)

◆ Asignación con lista enlazada e índice

- Misma idea que antes, pero todos los punteros se almacenan en una estructura aparte (índice) que se almacena en disco (tamaño de bloque no se modifica), 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

Physical
block

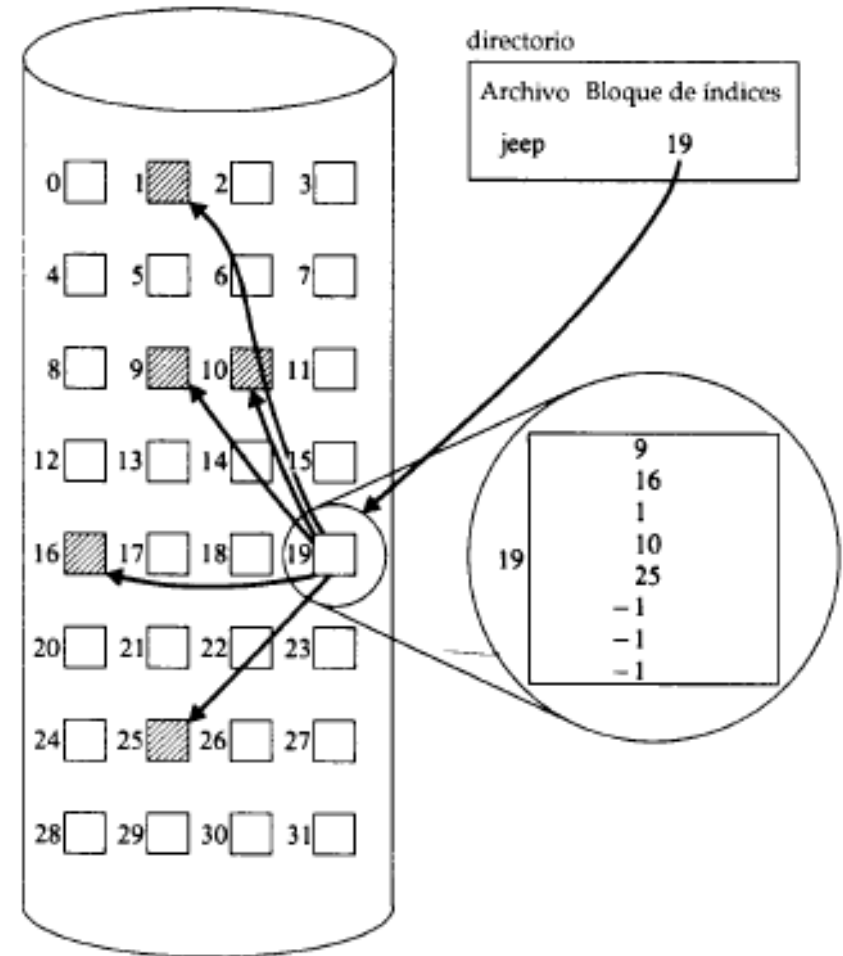
0		
1		
2	10	
3	11	
4	7	← File A starts here
5		
6	3	← File B starts here
7	2	
8		
9		
10	12	
11	14	
12	-1	
13		
14	-1	
15		← Unused block



Implementación de ficheros (IV)

◆ Asignación con nodos-i

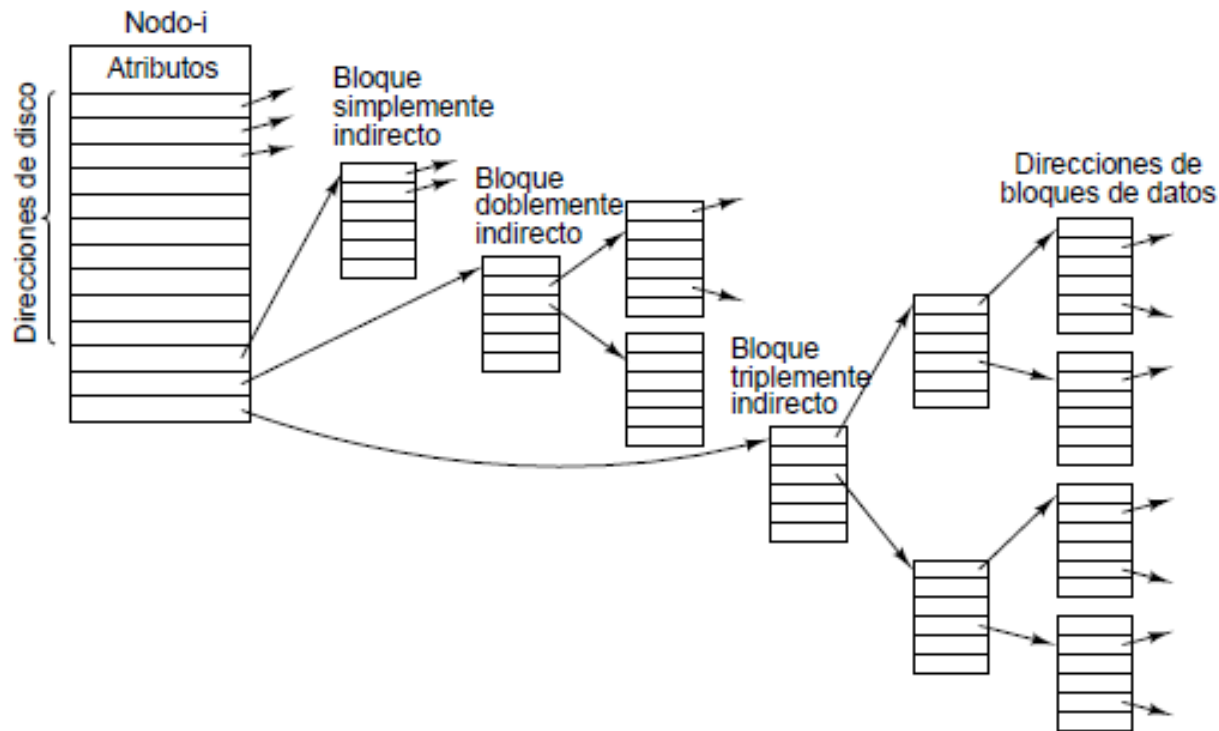
- A cada fichero se le asigna una estructura de datos, llamada **nodo-i**, en donde se almacenan sus atributos y las direcciones de sus bloques
- Dicha estructura se guarda en disco y se lee cuando se abre el fichero



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



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)

juegos	
correo	
noticias	
trabajo	

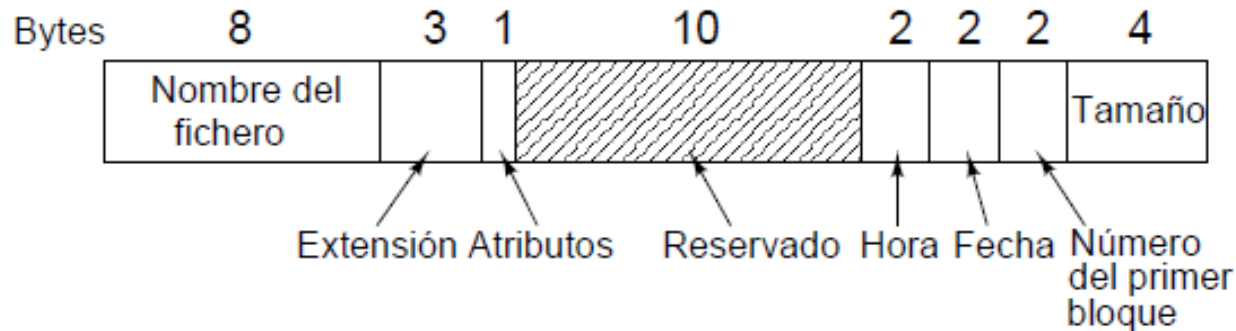
(b)

La estructura de datos
contiene los
atributos

Implementación de directorios(II)

Directorios en MS-DOS

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



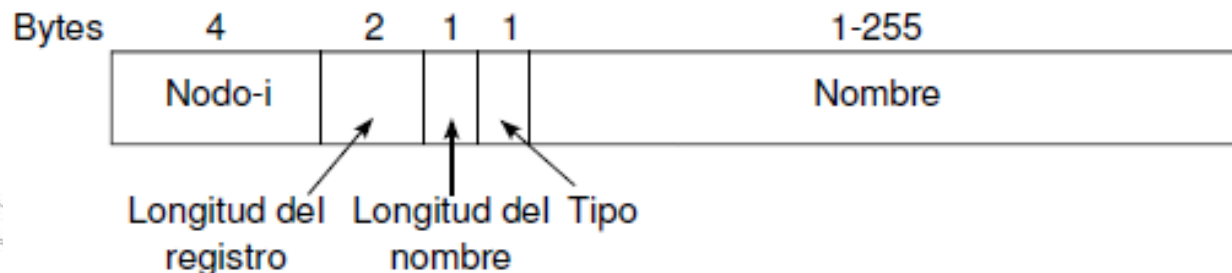
- Un bit de los atributos de la entrada distingue a un directorio de un fichero normal → un directorio puede tener subdirectorios → Árbol de directorios
- El directorio raíz es una excepción, ya que ocupa unos bloques fijos de disco y tiene un tamaño máximo establecido



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 nº's de nodos-i asociados. El tipo de fichero es una excepción y se utiliza para acelerar los listados de directorios

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:

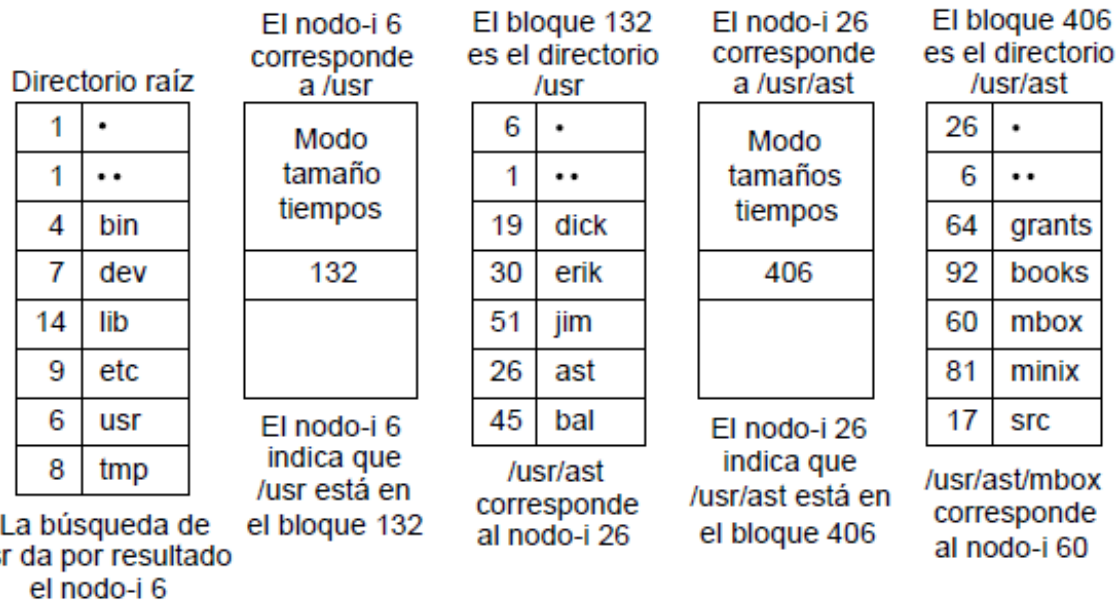
Nº entrada de la MFT	Atributos	Longitud del nombre	Nombre en Unicode
-------------------------	-----------	------------------------	-------------------

- 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

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

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
- Contador de enlaces en el nodo-i

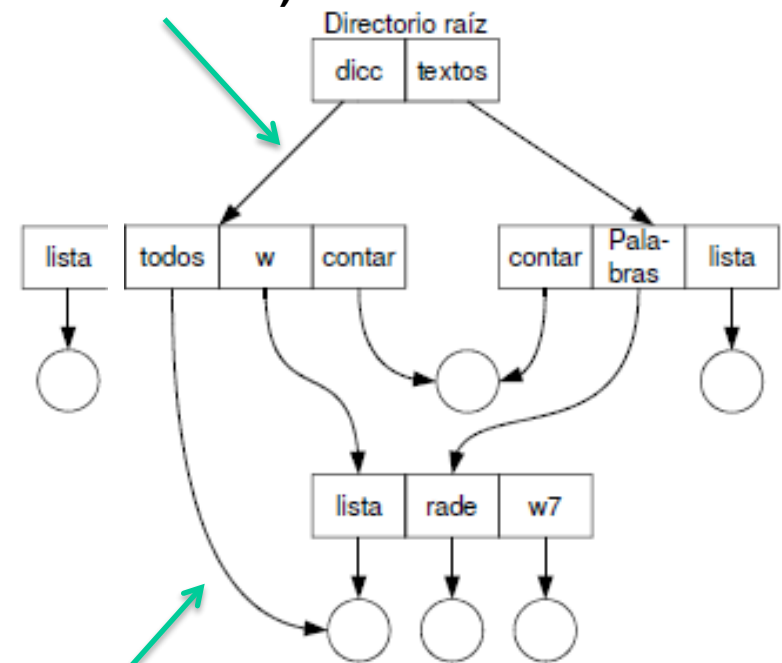
- Enlaces simbólicos (*soft links*)

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

- ◆ Problema: peligro de duplicidad de datos

**Enlace simbólico
(nombre a nombre)**

○ Nodo-i

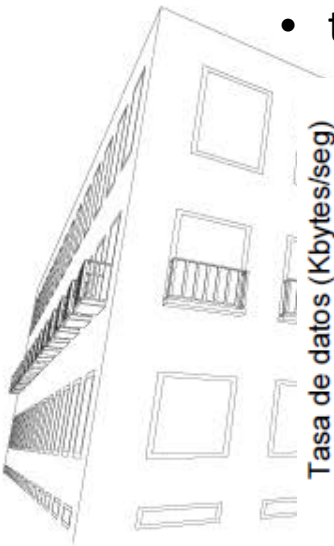
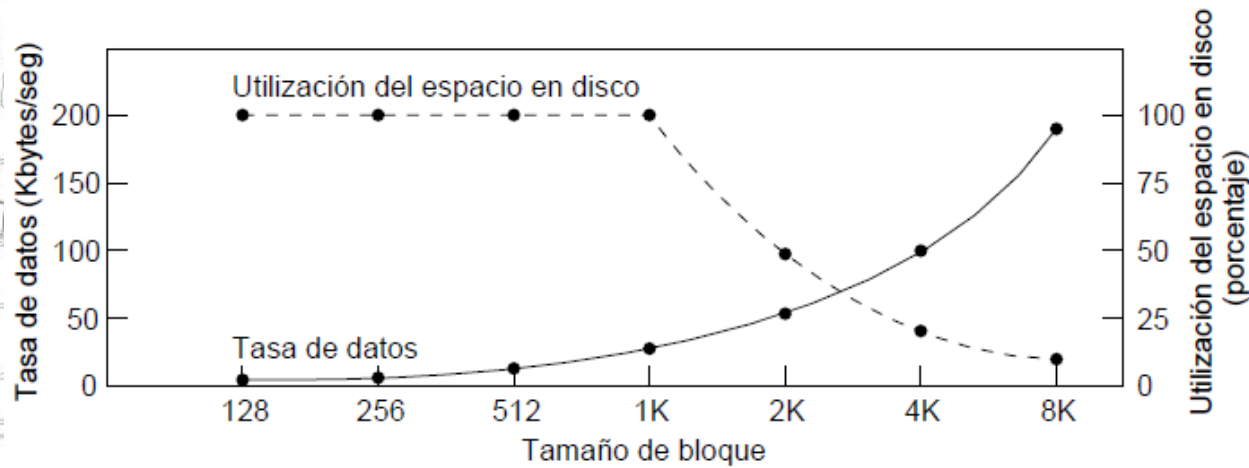


Enlace físico (nombre a nodo-i)

Administración del espacio en disco

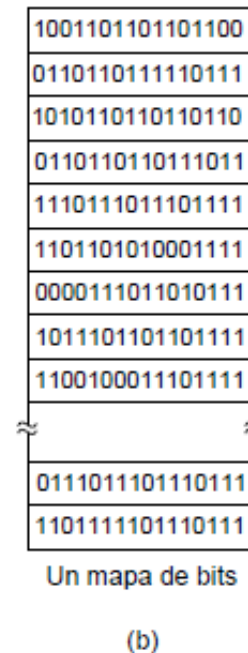
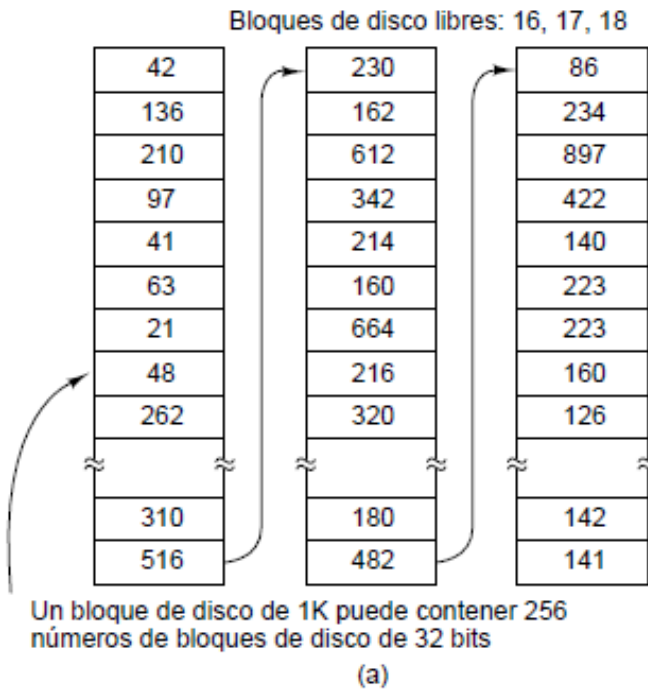
◆ Tamaño de bloque lógico

- 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



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.)



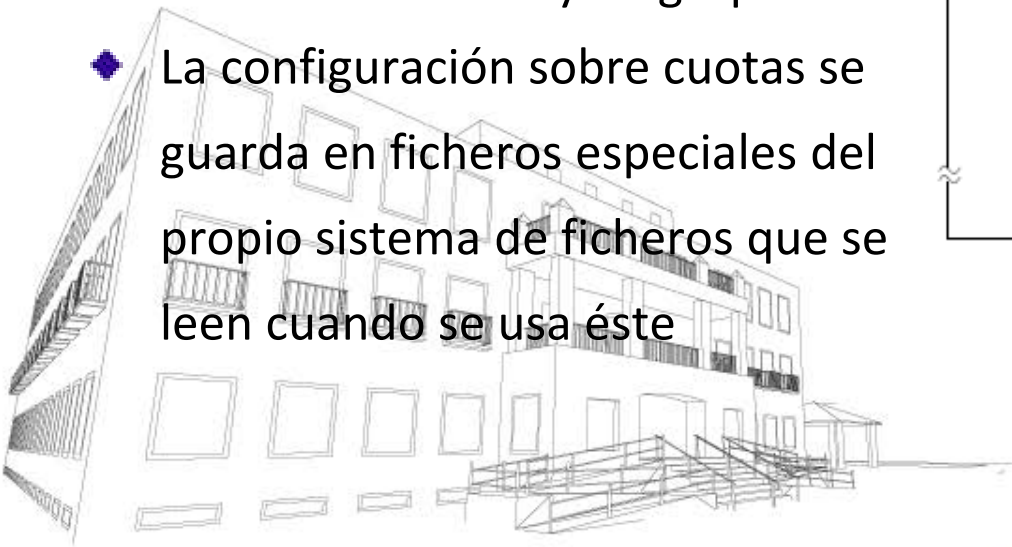
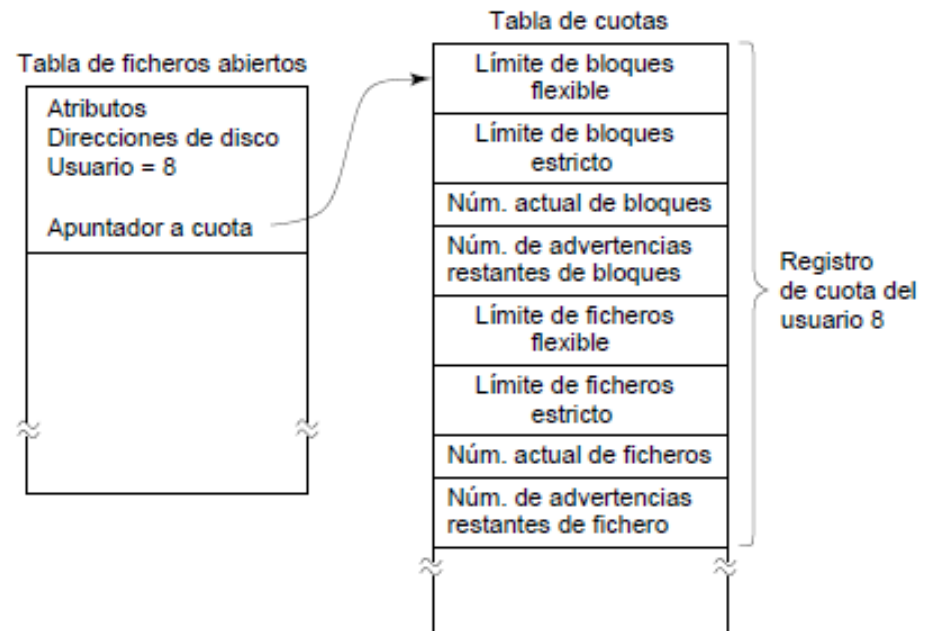
Administración del espacio en disco (III)

- ◆ 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

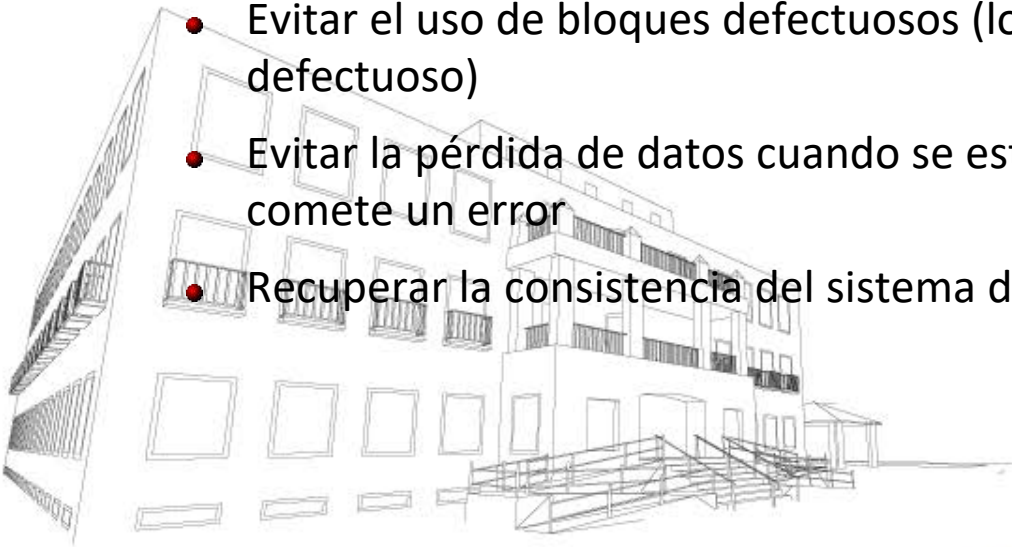
- ◆ El administrador puede establecer cuotas de usuario y de grupo

- ◆ La configuración sobre cuotas se guarda en ficheros especiales del propio sistema de ficheros que se leen cuando se usa éste



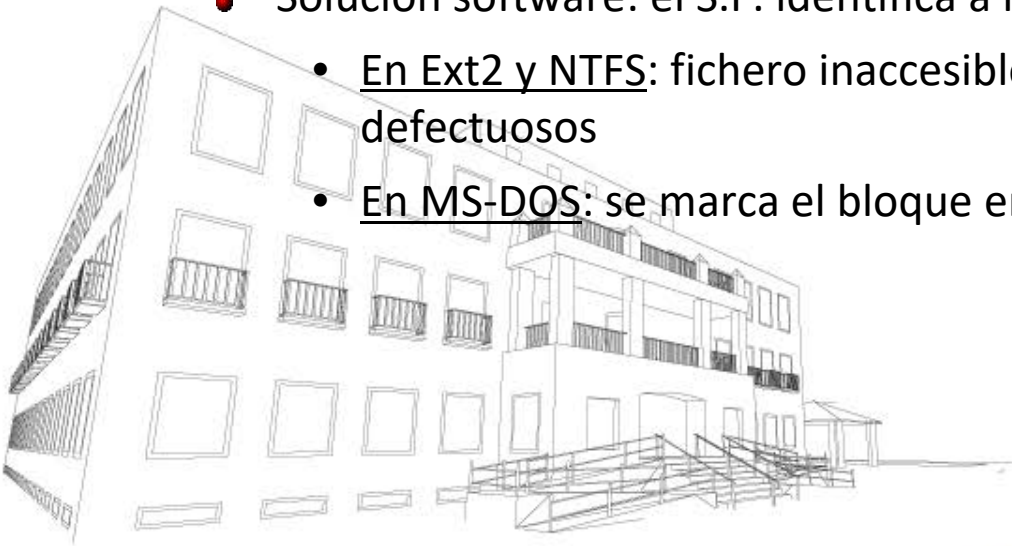
Confiabilidad del sistema de ficheros

- ◆ 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



Confiabilidad del sistema de ficheros (II)

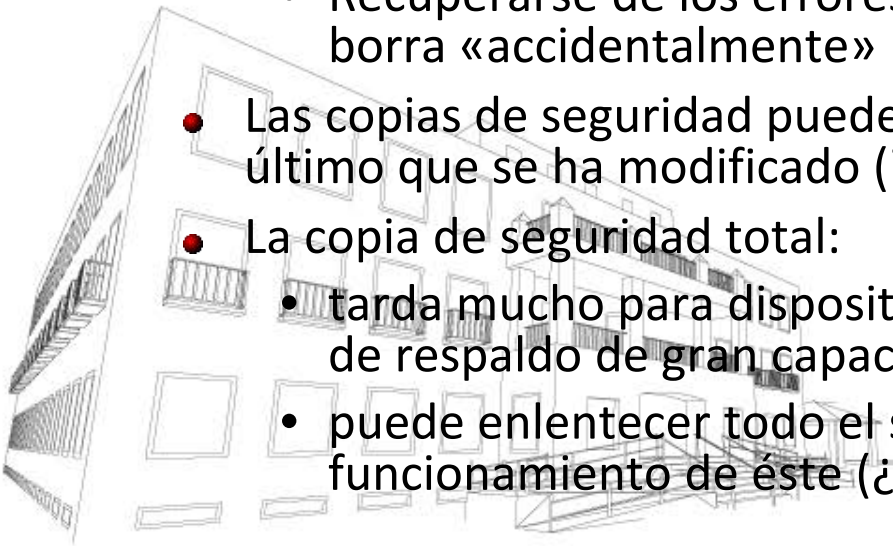
- ◆ Manejo de bloques defectuosos. Tenemos que evitar el uso de bloques defectuosos de disco:
 - Solución hardware: el propio disco tiene sectores de reserva y un mapa que asocia, transparentemente, bloques defectuosos a bloques sanos
 - Solución software: el S.F. identifica a los bloques defectuosos 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



Confiabilidad del sistema de ficheros (III)

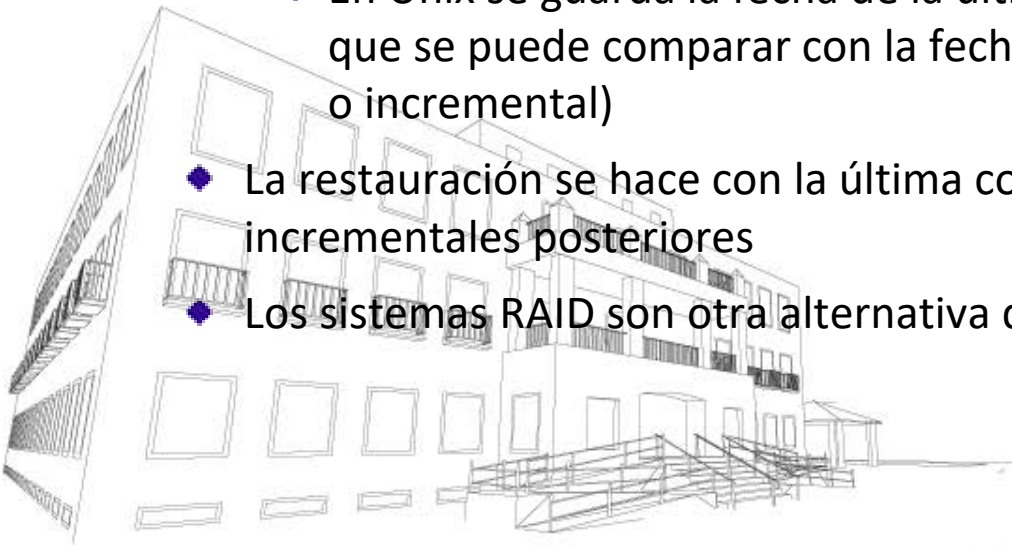
◆ Copias de seguridad

- 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?)



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



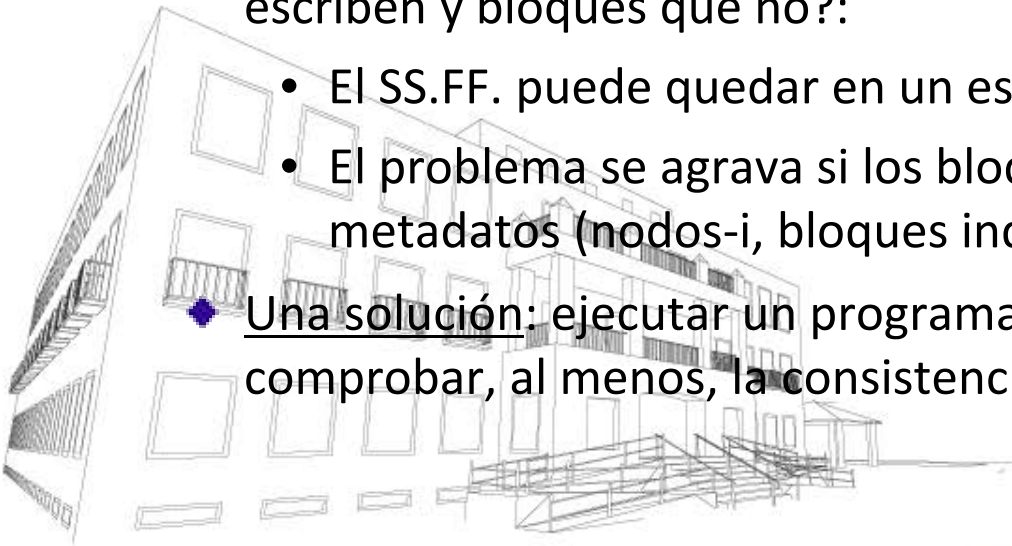
Confiabilidad del sistema de ficheros (IV)

◆ 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

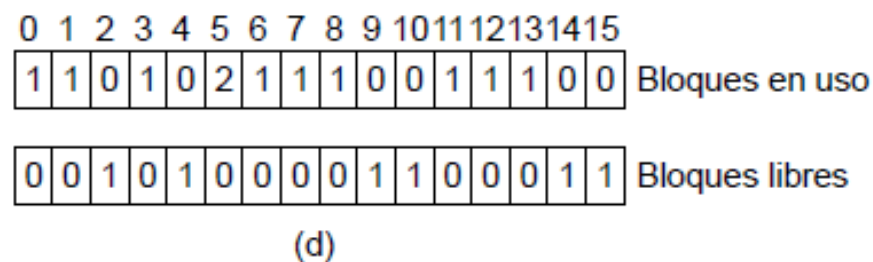
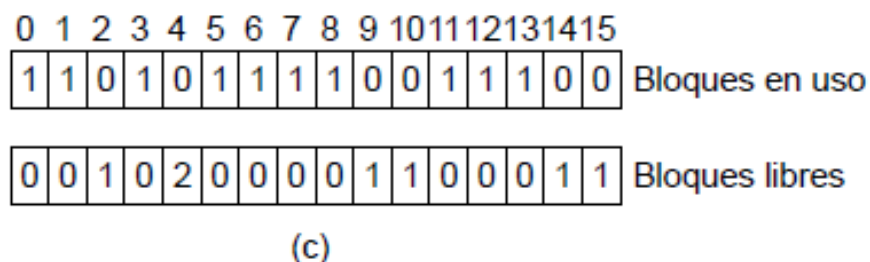
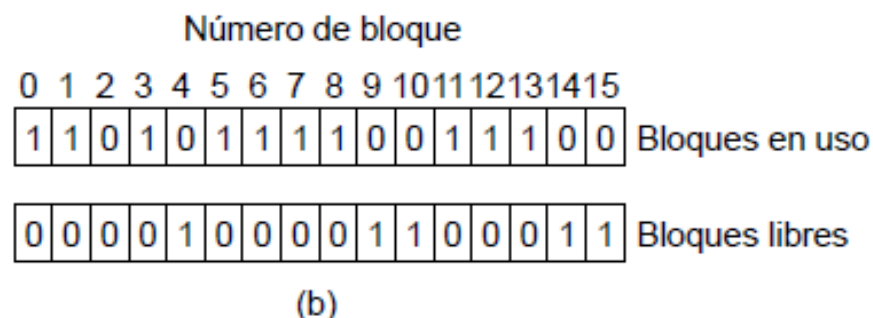
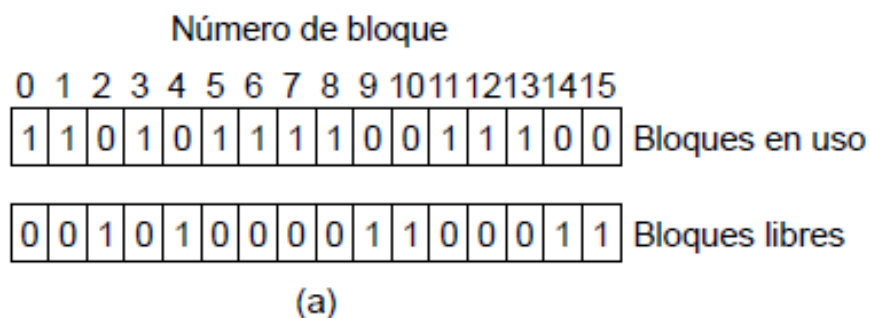


Confiabilidad del sistema de ficheros (V)

- ◆ 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 → **Bien**
 - ◆ 0 en los dos contadores («bloque faltante») → **Añadir a libres**
 - ◆ 2 o más veces como libres → **Reconstruir lista enlazada**
 - ◆ 2 o más veces en un mismo fichero o en ficheros distintos → ¡Error grave!
→ **Hacer copias**
 - ◆ Ocupado y libre a la vez → ¡Error potencialmente grave! (podría degenerar al caso anterior) → **Quitar de libres**

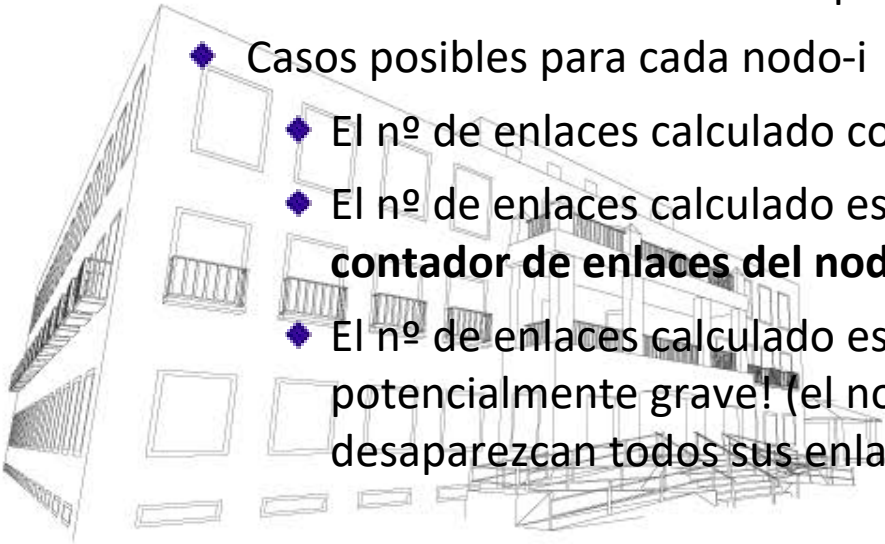
Confiabilidad del sistema de ficheros (VI)

◆ Consistencia de bloques (continuación. . .)



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 nº de enlaces calculado coincide con el del nodo-i → **Bien**
 - ◆ El nº de enlaces calculado es menor que el del nodo-i → **Reparar el contador de enlaces del nodo-i y liberar dicho nodo-i si el valor es 0**
 - ◆ El nº de enlaces calculado es mayor que el del nodo-i → ¡Error potencialmente grave! (el nodo-i se podría liberar antes de que desaparezcan todos sus enlaces) → **Solución anterior**



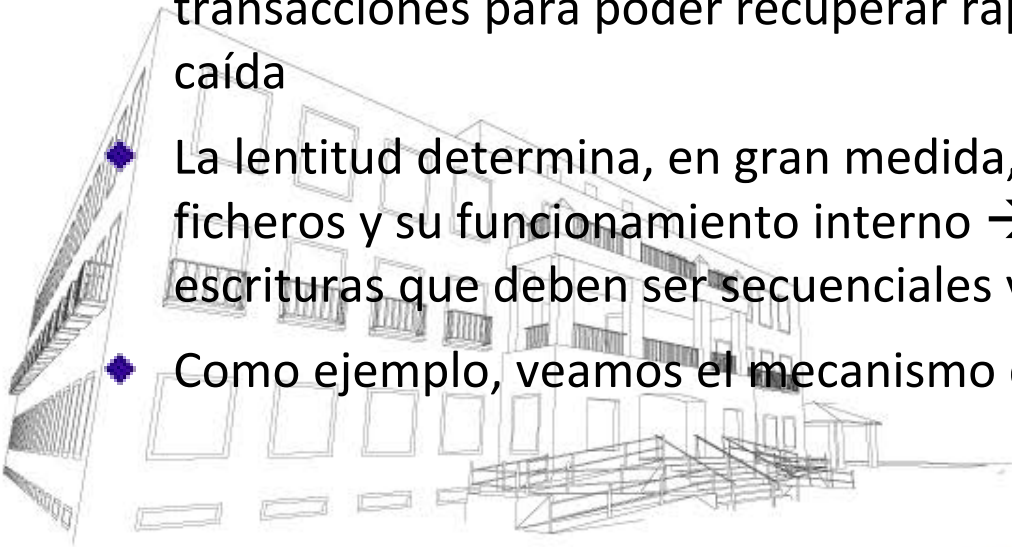
Confiabilidad del sistema de ficheros (VIII)

- ◆ Otras comprobaciones. En general, es conveniente comprobar todo lo que se pueda como, por ejemplo:
 - ◆ Ver si el nº 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



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 lectura y escrituras que deben ser secuenciales y en grandes grupos si es posible
- ◆ Como ejemplo, veamos el mecanismo de transacciones



Sistemas de ficheros modernos (II)

- ◆ Recuperación rápida de la consistencia
 - ◆ Para no perder la consistencia del S.F. las modificaciones tienen que ser atómicas:
 - ◆ Las modificaciones se guardan primero en un fichero especial, llamado registro, bitácora o *journal*
 - ◆ Cuando dicho fichero está a salvo en disco, se escriben en disco las modificaciones del propio sistema de ficheros
 - ◆ Si el sistema cae en el primer punto se pierden modificaciones (pero ninguna queda a medio)
 - ◆ 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

