CAPÍTULO VI: SISTEMA DE FICHEROS

ÍNDICE

Bibliografía

[MORERA.95; Capítulo 9] Morera Pascual J., Pérez Campanero J. A., *Teoría y diseño de los sistemas operativos*. Anaya Multimedia, 1995.

[TANENB03] Tanenbaum A. S., *Sistemas operativos modernos*. Prentice-Hall, 2003.

[STALLI01; Capítulo 12] Stallings, William. *Sistemas operativos*. Prentice-Hall 2001.

[CARRET01; Capítulo 8] Jesús Carretero, Félix García, Pedro De Miguel, Fernando Pérez. *Sistemas operativos*. McGraw-Hill, 2001.

[MILEN94; Capítulo 7] M.Milenkovic, *Sistemas Operativos. Conceptos y Diseño*. *Segunda Edición.* McGraw-Hill, 1994

[DEITEL93; Capítulos 12,13] H. M. Deitel. *Introducción a los Sistemas Operativos*. Addison-Wesley Iberoamericana. 1993.

1. Introducción

¿Qué podemos hacer si un proceso necesita gran cantidad de espacio?

Continuando con lo que vimos en el tema de memoria lo más sencillo es reservarle posiciones del espacio virtual.

Problemas:

- ¿Qué pasa si el proceso requiere más espacio del que existe? (i. e. aplicaciones de base de datos).
- El almacenamiento en memoria es volátil.

Para solucionar estos problemas optamos por guardar los datos en un almacenamiento secundario.

Este almacenamiento se realiza a través del concepto de **fichero** o **archivo**.

La parte del sistema operativo que se encarga de gestionar este almacenamiento es el **sistema o servidor de ficheros**.

Este módulo es, sin duda, el más perceptible por parte del usuario (es habitual que en las aplicaciones que usa tenga que leer o escribir ficheros).

En los apartados siguientes veremos la visión que el usuario y el diseñador tienen del sistema de ficheros.

1.1. Visión del usuario

De cara al usuario el sistema de ficheros es:

- Un conjunto de **ficheros** y
- un conjunto de **directorios**.

En los puntos siguientes analizaremos cada uno de estos dos conceptos.

1.1.1. Ficheros

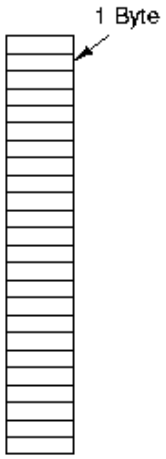
Los **ficheros** son los elementos centrales del sistema. Cualquier usuario genera y usa información a través de las aplicaciones que ejecuta el sistema.

Un **fichero** es una unidad de almacenamiento lógico no volátil que agrupa un conjunto de información relacionada entre sí bajo un mismo nombre.

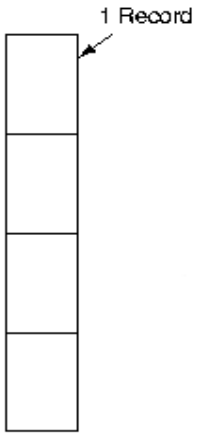
¿Qué características debe de cumplir el nombre que se le asocie a un fichero? Las reglas dependerán del sistema operativo (longitud limitada, distinción entre mayúsculas y minúsculas, etc.).

Internamente, ¿cómo puede estar **estructurada la información** contenida en un directorio? Desde el punto de vista lógico esta puede ser:

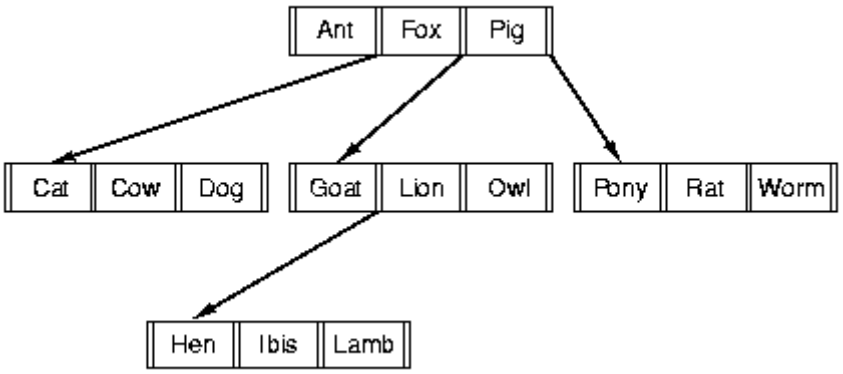
- **En bytes**: serie de bytes no estructurada que proporciona una máxima flexibilidad (UNIX, Windows).



- **En registros**: se comportan como una secuencia de registros de longitud fija y cierta estructura interna. La operaciones de lectura/escritura se hacen con registros (CPM).



- **Jerárquica**: consta de un árbol de registros de distinta longitud. Cada registro tiene un campo que se usa como **clave**. El árbol se ordena mediante este campo para mejorar la búsqueda.



Internamente el SO puede dar soporte a cada una de estas estructuras o soló soportar una estructura lo suficientemente flexible (la estructura secuencial) para permitir desarrollar sobre ella cualquier otra estructura.

Una vez que hay información almacenada en el fichero, ¿Cómo podemos acceder a ella? Hay dos maneras básicas (no todos los SO las proporcionan):

- **Acceso secuencial** (cintas): los bytes se leen en orden empezando siempre desde el comienzo.

- **Acceso directo o aleatorio** (dispositivos de acceso directo): el archivo se considera como un conjunto de registros. Se puede acceder a ellos de forma desordenada.

La mayoría de los S.O disponen de **tipos de ficheros** distintos:

- **Regulares** o de usuario (siguen las estructuras vistas anteriormente), contienen datos de usuario. Suelen contener código ASCII o binario.

- **Directorios**.

- **Especiales de caracteres**, dispositivos de entrada/salida como ratones, impresoras, ...)

- **Especiales de bloques**, discos.

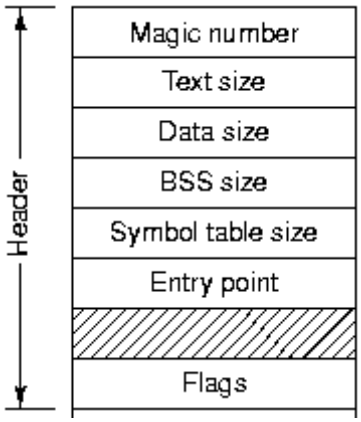
- **Tuberías**, ficheros que establecen un canal de comunicación entre procesos.

- **Enlaces**.

¿Cómo podemos saber qué tipo de información contiene un fichero regular?

- Se puede indicar como parte del propio nombre del fichero usando, usamos la extensión. Muchos sistemas operativos utilizan nombres de archivo con dos partes separadas por un punto. La parte posterior al punto es la **extensión** del fichero (.pas, .c, .bas, ...).

- Se indica como parte del contenido del fichero. Este empieza con una **cabecera** en donde se indica lo que contiene el fichero (La orden *file* de *Linux* usa esta información). Esta cabecera empieza con un número mágico:



¿Qué **operaciones** puede realizar un usuario sobre los ficheros?

- Crear
- Borrar
- Abrir
- Cerrar
- Leer
- Renombrar
- Escribir
- Añadir
- Buscar (ficheros con acceso aleatorio)
- Obtener atributos
- Establecer atributos

El **servidor de ficheros** es la parte del sistema operativo que se ocupa de facilitar el manejo de los dispositivos periféricos, ofreciendo una visión lógica simplificada de los mismos en forma de archivos.

Las funciones que debe cumplir el **sistema de ficheros** son:

- **Ocultar aspectos específicos** de los dispositivos de almacenamiento (**independencia del dispositivo**).
- Posibilitar la **operación** sobre los ficheros (copiar, crear, eliminar, renombrar, listar contenido, cambiar atributos o permisos).
- Posibilitar la **compartición** de los ficheros de forma controlada.
- Proporcionar una **estructura flexible** a los ficheros.
- Proporcionar **utilidades de respaldo y recuperación** de la información.
- Proporcionar mecanismos de **cifrado y descifrado** de la información.

1.1.2. Directorios

Los **directorios** son ficheros especiales que contienen información sobre otros ficheros y que posibilitan al usuario organizar sus ficheros creando una entrada por cada fichero.

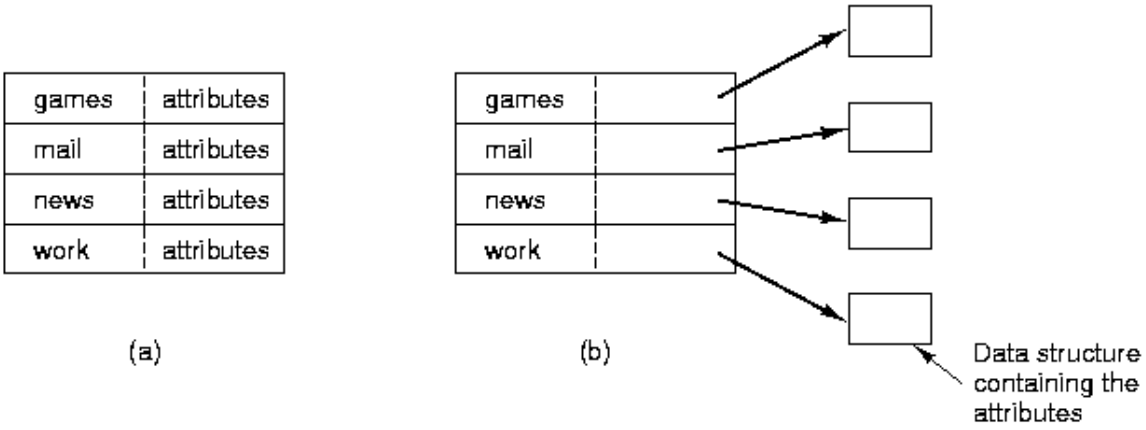
¿Qué información contiene el directorio? Contiene varias entradas, una por archivo contenido, en donde se almacena:

- Tipo de fichero
- Tamaño
- Propietario
- Permisos
- Fecha y hora
- Estadísticas
- **Bloques de disco usados para almacenar el fichero**.

¿Cómo podemos almacenar esta información en el directorio?

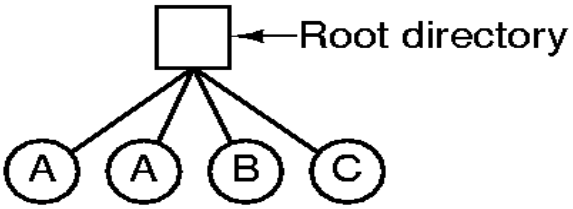
a) En cada entrada del directorio almacenamos el nombre del archivo y una serie de atributos.

b) En cada entrada del directorio almacenamos el nombre del archivo y un puntero a una estructura de datos donde están contenidos los atributos:



El número de directorios anidados varía de un sistema a otro:

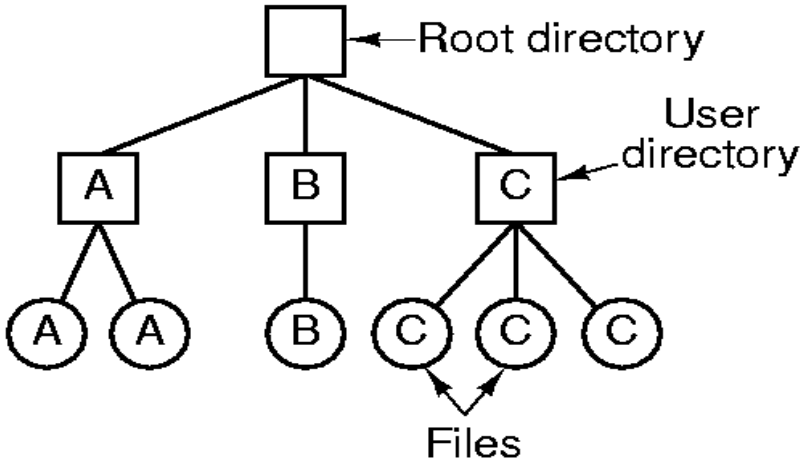
- **Organización a un nivel.** En su forma más sencilla el sistema sólo tiene un directorio, que incluye todos los ficheros de todos los usuarios.



Problemas:

- Protección
- Confusiones
- Conflictos
- Mismos nombres

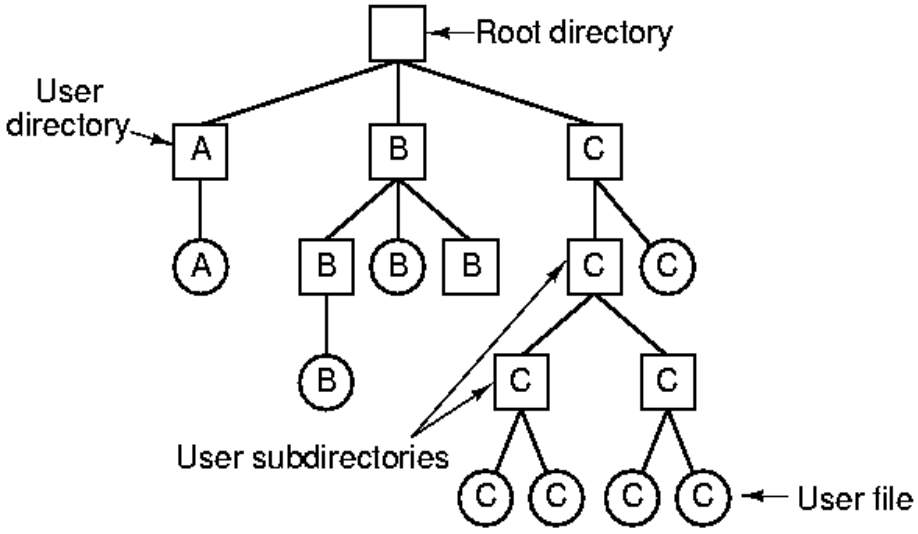
- **Organización a dos niveles.** Un directorio por usuario.



Problemas:

  - Organización para cada usuario

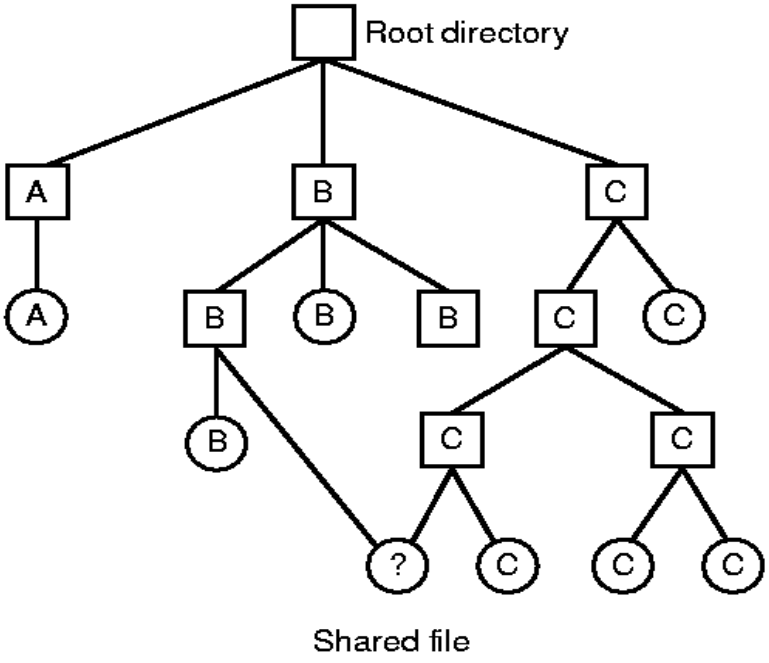
- **Organización en árbol**



Problema:

  - Compartir ficheros entre usuarios

- **Grafos acíclicos.** Permiten la realización de enlaces y enlaces simbólicos.



- **Grafos generales**.

Otra cuestión es: ¿Un único árbol o más de uno?

Existen sistemas con un único árbol de directorios, que deben dejar la posibilidad de usar más de un dispositivo (mount), y otros que disponen de su propia estructura por dispositivo. (A:, B: ...)

El directorio será el encargado de guardar la información para poder localizar en un determinado dispositivo de almacenamiento los datos de un fichero a partir de su nombre. Por ello en algunos sistemas los directorios son llamados también **tablas de traducción**.

¿Dónde guardamos la información de los directorios? También en un dispositivo de almacenamiento, en el mismo dispositivo en el que se encuentran los ficheros que localiza y organiza.

¿Que **operaciones** podemos realizar sobre los directorios?

- Crear
- Borrar
- Abrir
- Cerrar
- Leer
- Renombrar
- Enlazar
- Desenlazar

1.2. Visión del diseñador

La preocupación del diseñador ha de ser cómo se implementa el sistema de ficheros.

Las tareas que deberán tener en cuenta son:

- Elección del tamaño adecuado de bloque lógico (debería ser múltiplo del espacio físico).
- Organización del disco (políticas de acceso, ¿por qué orden se atienden las peticiones de lectura y escritura que llegan?).
- Gestión del espacio libre (cuándo se incrementa el espacio ocupado, qué bloques de disco se le asocian).
- Elección de las estructuras de almacenamiento.
- Gestión del espacio ocupado (debemos evitar conceder a un fichero espacio ocupado por otro).

2. Gestión del disco

Los archivos y directorios se almacenan en dispositivos de almacenamiento secundario.

En estos dispositivos debemos distinguir entre:

- **Sector**: Unidad de acceso. Unidad mínima en la que dividimos el disco.
- **Bloque**: Unidad de asignación de transferencia. Nº de sectores que se transfieren cada vez que traemos o llevamos información al disco. Es la unidad de asignación de transferencia, que será la mínima cantidad de información que se lee o se escribe en cada instante. Vendrá definida por el SO:

  $$Bloque = K \cdot Sector$$

  donde el valor de k se deberá indicar cuando se cree el sistema de ficheros.

- **Registro**: Unidad lógica de trabajo del usuario. Registro lógico de almacenamiento de información del usuario. Es muy buena idea que el registros sea múltiplo del tamaño del bloque. A nivel programación sería así:

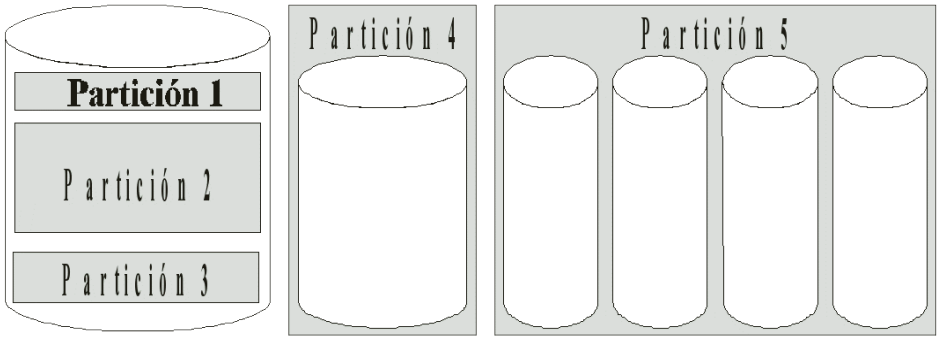
```
Datos = Record

Nombre: Array [1..100] of char

...

...

End;
```

2.1. Estructura lógica del disco

El sistema de ficheros permite organizar la información dentro de los dispositivos de almacenamiento en un formato inteligible para el sistema operativo.

Estos dispositivos vienen vacíos por los que antes de instalar el sistema de ficheros es necesario dividirlos **lógicamente** en particiones o volúmenes.

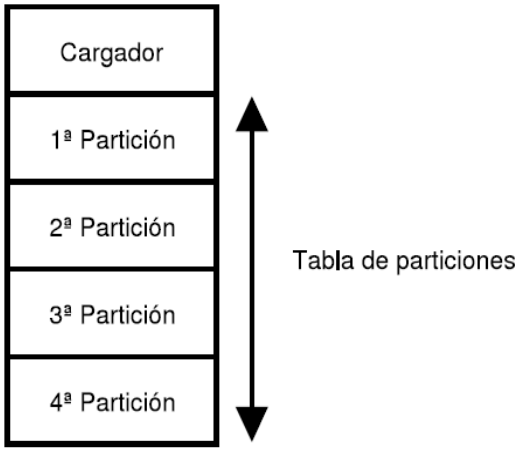
Una partición es una porción de un disco a la que se la dota de una identidad propia y que puede ser manipulada por el sistema operativo como una entidad lógica independiente:



Al finalizar está operación tendremos en el disco tres elementos:

- Sector de arranque o MBR (Master Boot Record).
- Espacio particionado.
- Espacio sin particionar.

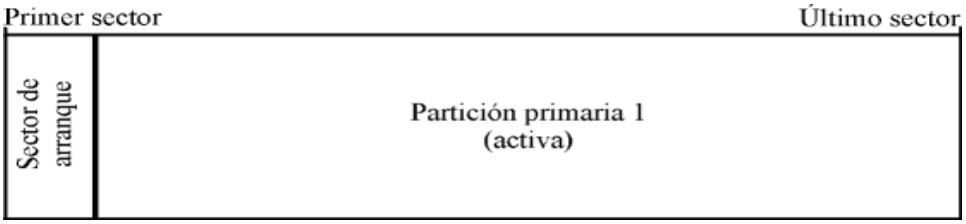
EL **MBR** es el primer sector de todo disco duro (cilindro 0, cabeza 0, sector 1). En el se almacena la tabla de particiones y un pequeño programa cargador denominado **MBC** o (Master Boot Code):



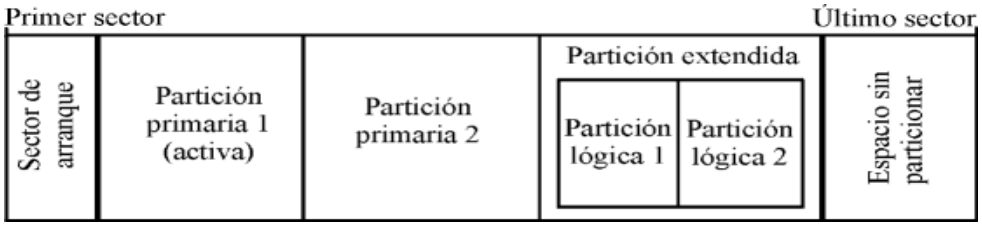
El **espacio particionado** es el espacio del disco que ha sido asignado a alguna partición.

El **espacio no particionado**, es espacio no accesible del disco ya que todavía no ha sido asignado a ninguna partición.

Ejemplo 1: un disco duro con una partición:



Ejemplo 2: un disco duro con 2 particiones primarias, 2 lógicas y espacio sin particionar:



Ejemplo 3: Información de particiones devuelta por fdisk:

```
Disk /dev/hda: 255 heads, 63 sectors, 1467 cylinders

Units = cylinders of 16065 * 512 bytes

Device    Boot Start End Blocks    Id System

/dev/hda1 *    26    254  1839411  b  Win95 FAT32

/dev/hda2      1     22   176683+  84 OS/2 hidden C: drive

/dev/hda3      255   257  24097+   83 Linux

/dev/hda4      258   1467 9719325  f  Win95 Ext'd (LBA)

/dev/hda5      258   1026 6176961  83 Linux

/dev/hda6      1027  1067 329301   82 Linux swap

/dev/hda7      1068  1467 3212968+ b  Win95 FAT32

Partition table entries are not in disk order
```

¿Qué pasos realizan un ordenador previos a la carga del sistema operativo?

1. La BIOS y la CPU hacen diversos test, power-on self test (POST).
2. La BIOS busca el dispositivo de arranque que normalmente es el primer disco.
3. De este dispositivo carga en memoria el MBR y le pasa el control al MBC.
4. El MBC busca en la tabla de particiones la marcada como activa (BOOT=*).
5. Carga en memoria el sector de arranque de la partición activa.

6. Transfiera el control al código contenido en dicho sector.

3. Diseño del sistema de ficheros

Una vez creadas las particiones, el sistema operativo debe crear las estructuras de los sistemas de archivos dentro de esas particiones (format, mkfs).

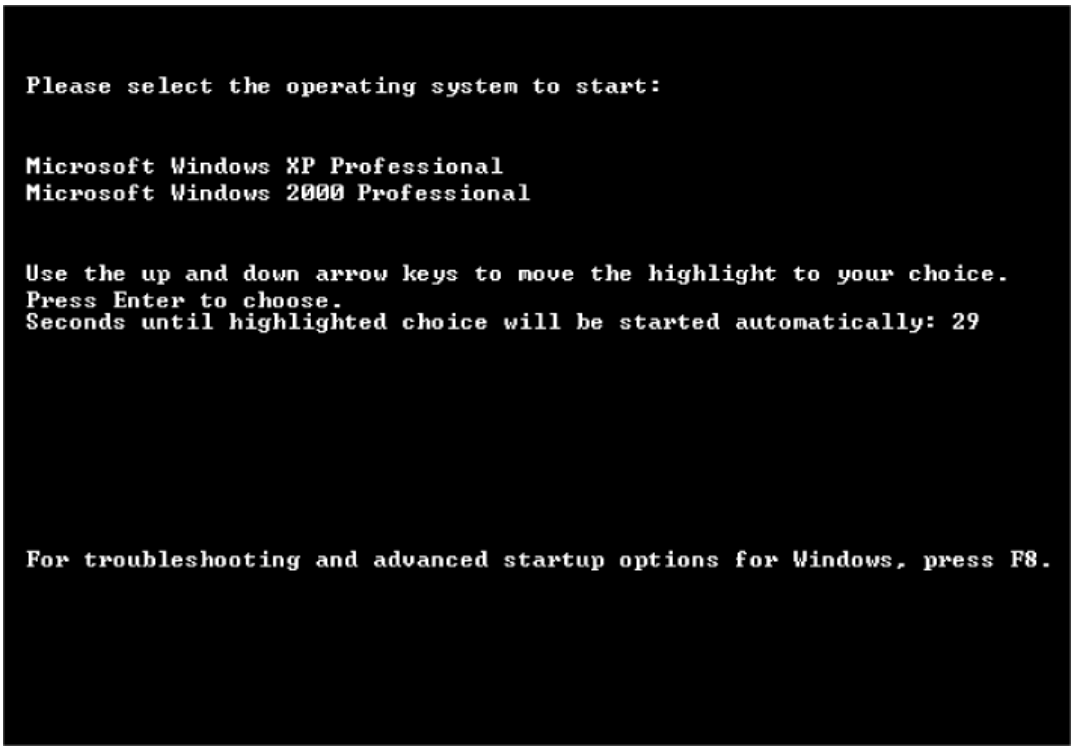
El resultado final de esta operación depende del sistema operativo. A continuación se muestra la estructura de los sistemas de archivos de los sistemas operativos más importantes:

BOOT	1º COPIA DE LA FAT	2º COPIA DE LA FAT	DIRECTORIO RAÍZ	DATOS Y DIRECTORIOS	**MS-DOS**

BOOT	Superbloque	Mapas de bits	i-nodos	DATOS Y DIRECTORIOS	**UNIX**

BOOT	MFT	DATOS Y DIRECTORIOS	**WINDOWS-NT**

En el sector de arranque o **boot** está el código que cargará el sistema operativo en memoria. En NTFS se carga el ntldr.sys que carga el menú de inicio si tenemos varias copias del sistema operativo:



A lo largo de este tema veremos el resto de secciones pero por ahora nos centraremos en la **sección de datos y directorios**.

¿Cómo se almacena la información en estas particiones?

- **Almacenamiento secuencial contiguo de bloques**, se nos plantea un problema al modificar el tamaño de los ficheros (traslado completo del archivo en busca de un hueco)

- **Almacenamiento secuencial no contiguo de bloques**, el inconveniente al que se enfrentaría el SO sería conocer qué bloques utilizan cada uno de los archivos; para esto se definirán distintas estructuras en función del S.O. (UNIX, DOS, etc.).

¿Cuándo se crean estos bloques? Cuando se formatean las particiones. El tamaño de un sistema de ficheros se establece en bloques.

El bloque es la mínima unidad de información que maneja el SO.

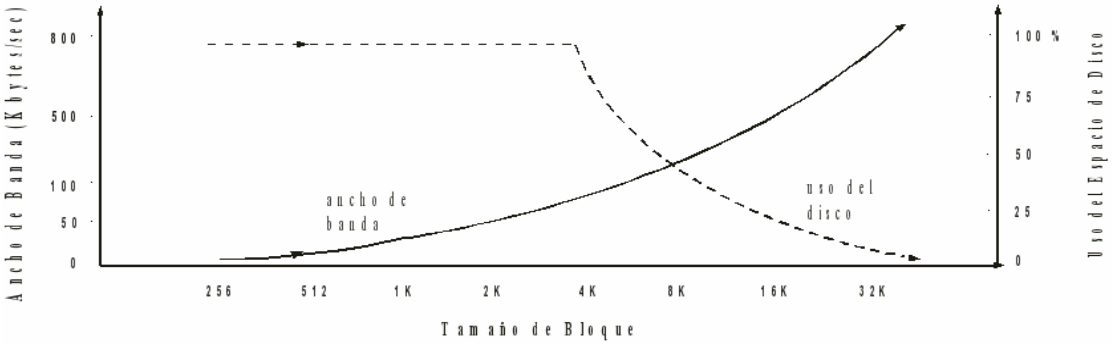
Es muy importante establecer un tamaño de bloque adecuado, de ello dependerá la eficacia de las lecturas y el aprovechamiento que se haga del disco.

Debemos encontrar el equilibrio entre las siguientes cuestiones de implementación:

- Tamaño de los ficheros almacenados.

- Porcentaje de disco usado / desperdiciado (fichero - bloque).

- Velocidad de transferencia.



La curva cambia con la tecnología de los discos y el tamaño medio de archivos.

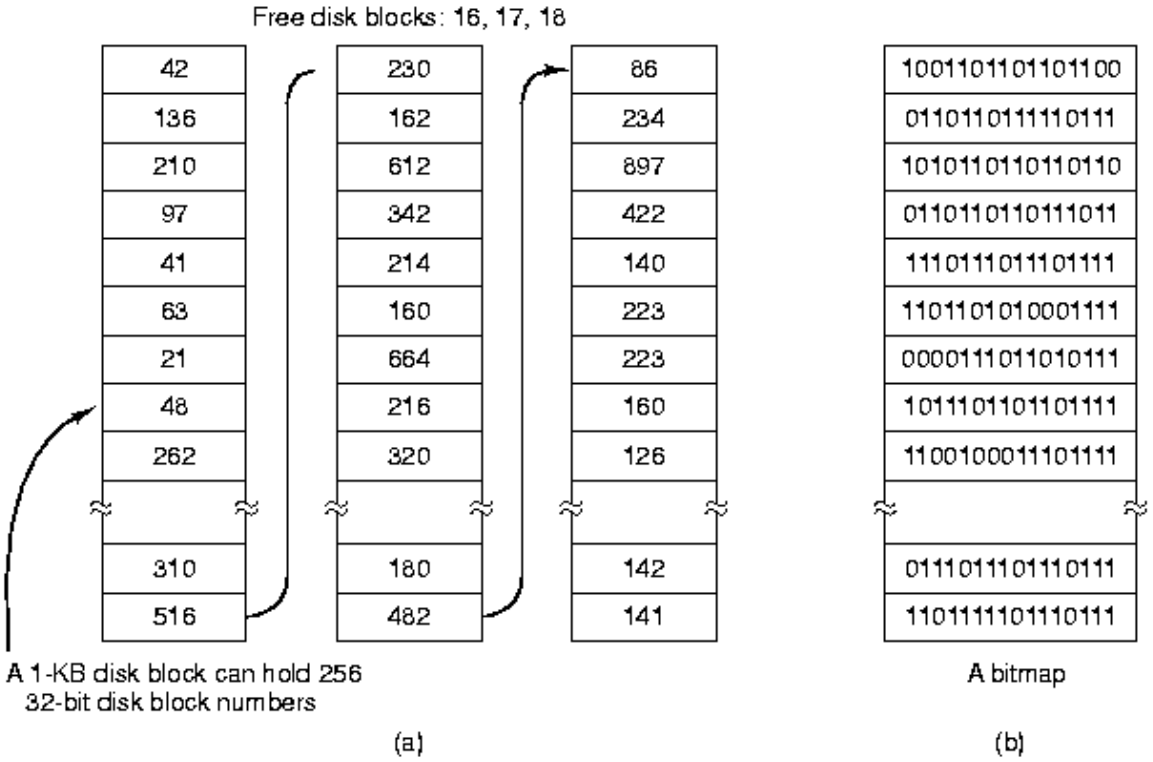
3.1. Gestión del espacio libre

Es necesario gestionar los bloques libres que nos quedan en los dispositivos de almacenamiento.

¿Cómo podemos implementar esta gestión?

- **Mapa de Bits**: En una tabla de bits definimos si un bloque está ocupado o no lo está. Necesitamos 1 bit por cada bloque del disco. (Por ejemplo 1= ocupado y 0=libre).

- **Lista Enlazadas** (MS-DOS): El SO conoce la dirección de un bloque en el que están almacenados punteros a bloques libres. Si se necesitan más bloques, la última posición apuntará al siguiente bloque con apuntadores a bloques libres.



Sólo si el disco está casi completo la lista enlazada requiere menos bloques que el mapa de bits.

Si se puede guardar en memoria todo el mapa de bits es preferible este método.

En otro caso es preferible la lista enlazada (porque un bloque de la lista enlazada tiene siempre bloques libres, y un solo bloque del mapa de bits puede que no tenga ninguno libre).

Una posible optimización consiste en reunir los bloques libres en **agrupaciones** o clúster y tener un mapa de bits (Unix, Linux, NT ....) o una lista enlazada a estas agrupaciones.

En el caso de usar agrupaciones, la mínima unidad que se puede asignar es una agrupación.

3.2. Almacenamiento de ficheros

Si un fichero está compuesto de una sucesión de bloques el sistema operativo debe llevar el control de los bloques de cada archivo.

Hay distintas aproximaciones que intentan solucionar este problema respondiendo a las preguntas:

- Cuando se crea un nuevo archivo, **¿se asigna de una sola vez el máximo espacio que necesite?** Es muy difícil saber el tamaño máximo de un fichero, se suelen hacer sobreestimaciones que provocan un derroche de espacio.

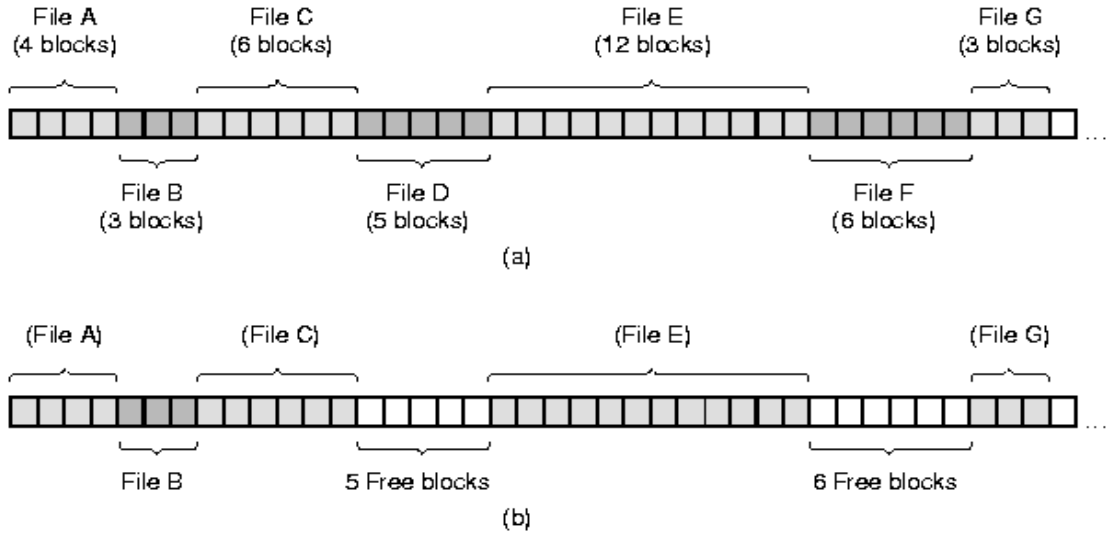
- El espacio se asigna a un archivo en forma de una o más unidades contiguas, que se llaman agrupaciones. El tamaño de una agrupación puede variar desde un único bloque a un archivo entero. **¿Qué tamaño de sección debería usarse para asignar archivos?.**

- **¿Qué tipo de estructura de datos se usará para guardar constancia de las agrupaciones asignadas a un archivo?**.

3.2.1. Almacenamiento en bloques consecutivos

También se le denomina de asignación previa.

Se le asigna un número consecutivo de bloques:



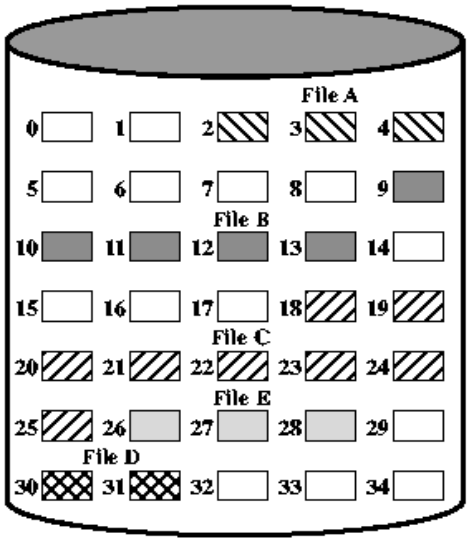
Ventajas:

- Sencillo de implementar
- Rendimiento de E/S muy alto

Desventajas:

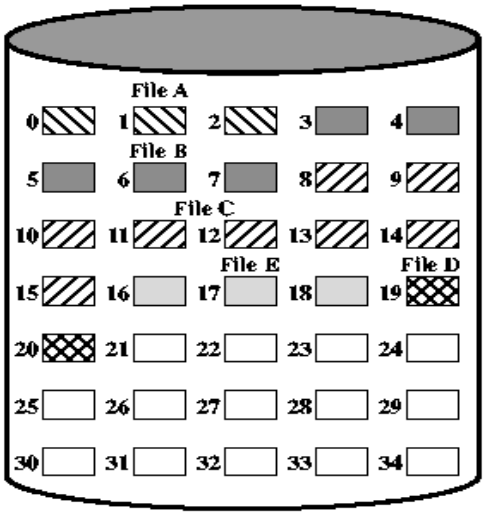
- No conocemos el tamaño máximo del fichero –> reubicación
- Fragmentación externa –> Compactación.

Ejemplo: Problema de la compactación, estado del disco antes y después de hacer la compactación:



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3



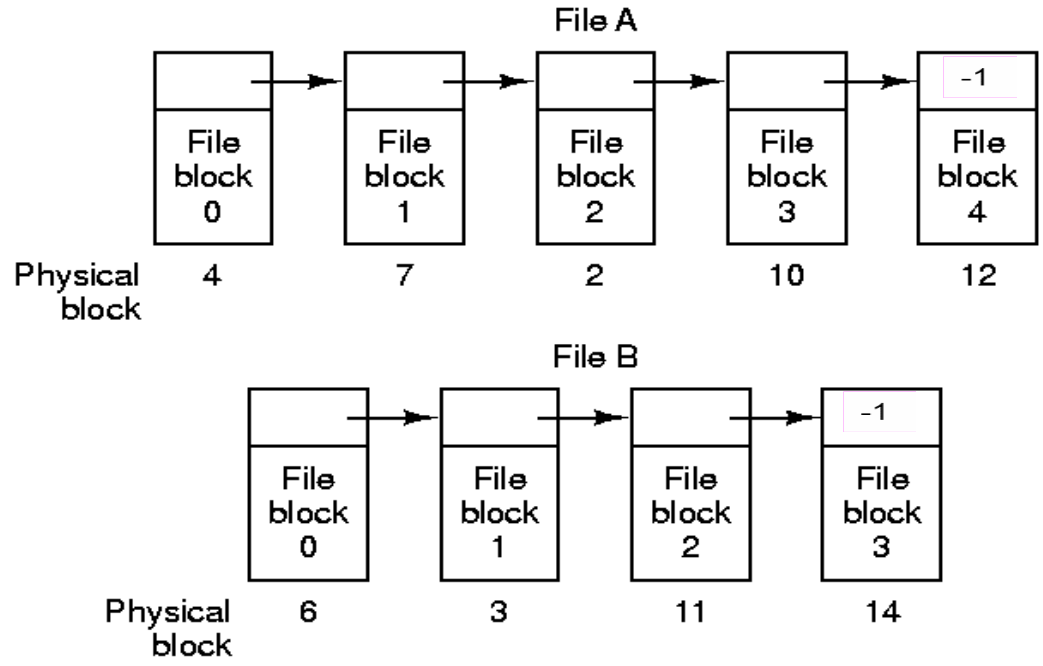
File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

3.2.2. Almacenamiento mediante listas

Utiliza la última parte del bloque como un apuntador al siguiente bloque, el archivo estará formado por una serie de bloques dispersos (se evita la fragmentación externa).

Se trabaja con lista enlazadas:



En el descriptor del fichero sólo es necesario almacenar el primer bloque.

Problemas:

- Es más complicado calcular el tamaño real del fichero.
- El tamaño del bloque ya no es potencia de 2.
- El acceso directo es muy costoso de implementar (se recorre secuencialmente la lista).
- Poco fiable, ¿qué pasa si perdemos un bloque?

Solución al problema de eficiente: cargar los punteros a ficheros en memoria.

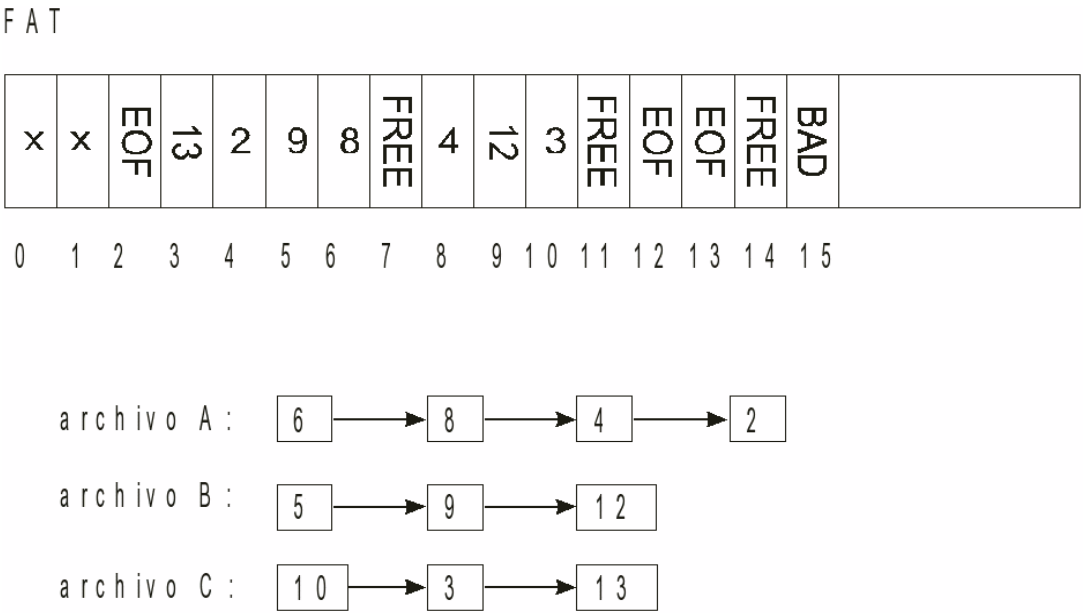
3.2.3. Índice enlazado

Las desventajas de la lista enlazada se pueden eliminar si se quitan los apuntadores de los bloques del archivo y se almacenan en un índice enlazado gestionado por el servidor de archivos.

Cuando se crea el SF se almacena en una parte especial del mismo una tabla que contiene una entrada por cada bloque de disco y que está indexada por número de bloque.

Cada vez que se crea un archivo se incluye en su descriptor (que estará contenido en el directorio donde se localiza el fichero) el índice de la tabla que apunta al primer bloque del archivo.

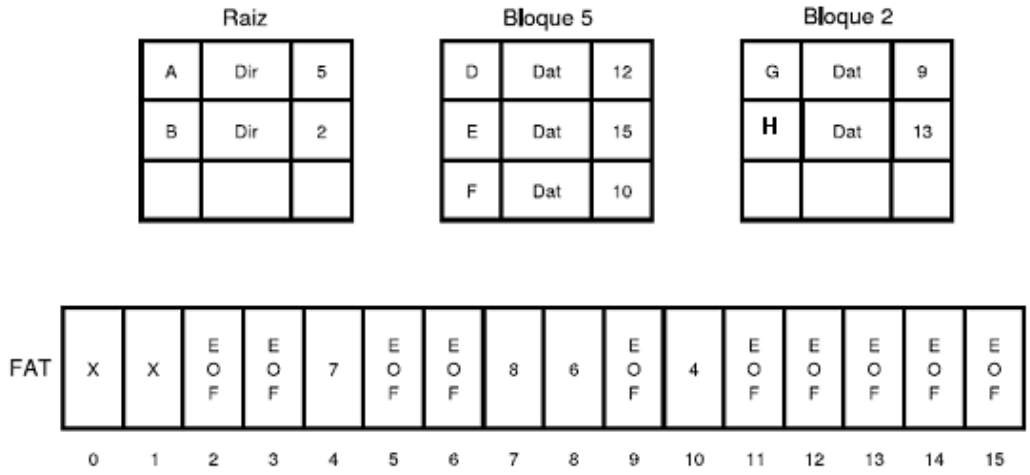
A medida que se asignan nuevos bloques al archivo se apunta a ellos desde la última entrada de la tabla asociada al archivo:



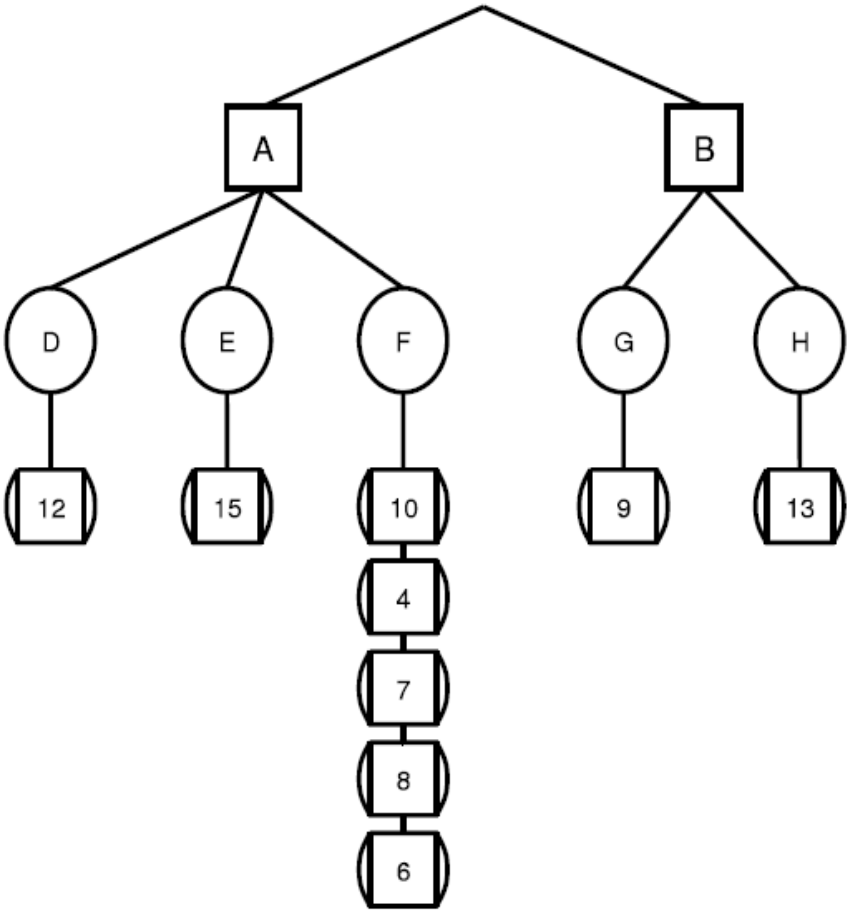
Estas tablas se guardar en el disco (existe una copia) pero para reducir en lo posible el tiempo de búsqueda en esta se intenta mantener en memoria.

Un ejemplo de este mecanismo de correspondencia es la FAT (File Allocation Table) implementada por MSDOS.

Ejemplo: Dada la siguiente FAT y entradas de directorio:



¿Cuál es el árbol de directorios asociado? Incluya en el los bloques que ocupa cada fichero.



La FAT ha ido evolucionando sobre todo debido al incremento en la capacidad de los dispositivos de almacenamiento.

Algunas de estas evoluciones son:

- **FAT 12**: Apuntadores de 12 bits, con lo que 2^{12}= 4096 posibles bloques, permitiendo direccionar hasta 4 Megas como máximo. En MSDOS se usa un tamaño de bloque de 1K, la tabla FAT para los disco de 320K ocupa:

$$Tam\,FAT = 320\text{x}12/8 = 480\,bytes$$

y 360K formateados con este sistema es:

$$Tam\,FAT = 360\text{x}12/8 = 540\,bytes$$

Cuando los discos duros superan los 4Mb necesitamos volver a cambiar la FAT (no tenemos bits suficientes para redireccionar más bloques).

- **FAT 16**: Apuntadores de 16 bits, con lo que 2^{16}= 65536 posibles bloques, permitiendo direccionar hasta 64 Megas. En un disco con esta capacidad tendremos la FAT que ocupará:

$$Tam\,FAT = 65536\text{x}16/8 = 128\,Kbytes$$

Problemas:

- Ocupa bastante espacio de memoria, habrá que descargarla a disco, al menos parcialmente, con lo que pierde su eficacia (se puede aminorar el problema usando agrupaciones o cluster).

- ¿Cómo podemos redireccionar discos más grandes? Aumentando el tamaño del bloque (hasta un máximo de 32 kb –> 2Gb).

Ejemplo: Tamaño máximo de las particiones en sistemas FAT

Tamaño bloque	FAT-12	FAT-16	FAT-32
0,5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	4 TB
32 KB		2048 MB	8 TB

3.2.4. Índice enlazado multinivel

Las desventajas anteriores se pueden solucionar usando un índice multinivel.

El nodo-i es una estructura de datos de almacenamiento en disco de UNIX / LINUX que contiene la información de un fichero.

Inicialmente un nodo-i almacenaba la siguiente información:

- ID de nodo-i.
- ID del dispositivo que contiene el fichero
- Tipo de fichero.
- Número de enlaces.
- UID del propietario.
- GID del grupo.
- Tamaño.
- Fecha de creación.
- Fecha de la última modificación y/o acceso.
- Protección y/o permisos.
- Otras informaciones.

- Apuntador Indirecto Simple: Apunta a un bloque que contiene apuntadores a bloques de datos.

- Apuntador Indirecto Doble: Apunta a un bloque que contiene apuntadores que apuntan a bloques que contienen apuntadores a bloques de datos.

- Apuntador Indirecto Triple: Se añade un nivel más de apuntadores que en el caso anterior.

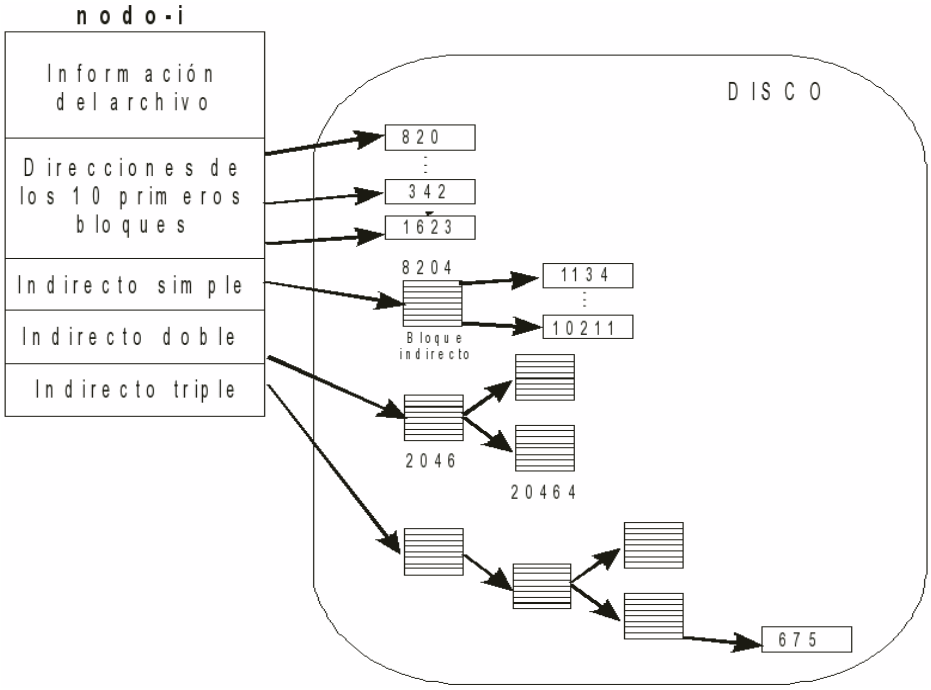
Ahora el archivo tiene sus bloques índice que incluyen apuntadores a los bloques de disco del archivo.

Este esquema presenta un uso ineficiente de espacio para archivos pequeños ya que necesitamos crear un i-nodo y un bloque de índice.

Para solucionar esto se optó por permitir que en el propio nodo-i se almacenaran punteros a las direcciones de los bloques. Así pues al nodo-i se le incluyó la siguiente información:

- 10 Apuntadores Directos: Apuntan a bloques de datos.

Esta solución es la que se usa los los sistemas UNIX:



Los distintos nodos-i se almacenan en forma de lista en disco en una zona especial del disco entre el superbloque y los bloques de datos.

Ventajas:

- Los apuntadores indirectos no se utilizan si no son necesarios.

- La mayoría de los ficheros suelen tener menos de 10 K., por lo que no necesitamos apuntadores dobles.

- El número máximo para cualquier referencia a disco son 3 (sin contar el I-NODE y el dato).

Ejemplo: Cuál será el tamaño máximo de un fichero en un SF que tiene un tamaño de bloque de 1 k, usa un nodo-i con 10 apuntadores directos, 1 simple, 1 doble y otro triple y las direcciones a disco son de 32 bits.

- Por los 10 apuntadores libres: 10 bloques * 1 kb por bloque = 10 Kb
- En cada bloque de 1k se pueden almacenar 1024/4=256 direcciones de bloques.
- Por el apuntador simple: 256 bloques de 1 Kb cada uno = 256 Kb
- Por apuntador doble: 256^2 bloques de 1 Kb = 64 Mb
- Por apuntador triple: 256^3 bloques = 16 Gb
- Total = 10 kb + 256 Kb + 64 Mb + 16 Gb = 16.06 Gb.

3.3. Estructura del directorio

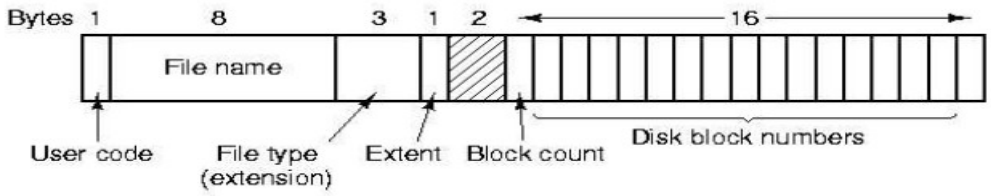
Cuando se abre un fichero usamos la ruta proporcionada por el usuario para acceder a los bloques de datos, de forma que más adelante se puede leer o escribir el fichero.

Esto de las rutas nos lleva a cómo se organiza la información en el disco.

Dependiendo del sistema de ficheros que estemos utilizando, cada entrada contenida en los directorios tendrá diferentes estructuras:

3.3.1. CP/M

CP/M: Estructura muy simple, sólo tiene un directorio, que es el único que existe. Al encontrar la entrada en el directorio ya tenemos también los números de bloques ya que todo se almacena ahí:



Durante la búsqueda sólo se miran las entradas válidas para el usuario.

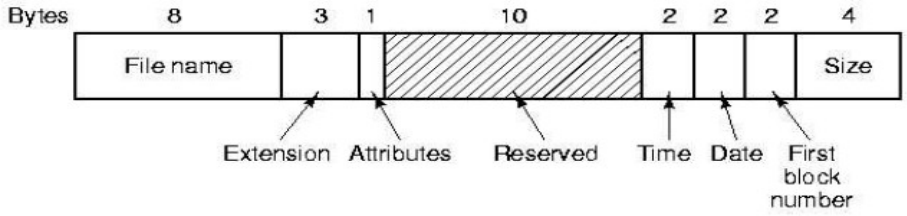
En esta estructura, para poder almacenar más de 16 bloques de fichero utilizaremos otra entrada del directorio, con el mismo nombre , código de usuario, tipo, pero con un grado diferente. Con el conteo de bloque defino un numero que es la cantidad de bloques de datos ocupados por el archivo:

A	txt	1	R	16	7	14	23	67	28	34	60	36	12	16	83	92	47	24	28	44
A	txt	2	R	2	5	8	?	?	?	?	?	?	?	?	?	?	?	?	?	?

No podemos conocer el tamaño en bytes, puesto que el último bloque no tiene porqué estar lleno. Así pues llevamos el tamaño en bloques.

3.3.2. MS-DOS

MS-DOS: Estructura de directorios múltiple, cada unidad tiene su propio árbol de directorios. Las entradas de directorio seguirán el siguiente esquema:



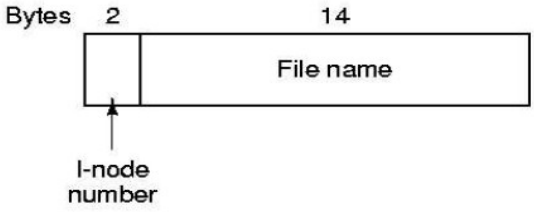
Aparte de dos entrada fijas que hacen referencia al mismo directorio ('.') y al padre ('..') existirán tantas entradas como ficheros contenga el directorio.

Un dato que debe tener en cuenta es que en MS-DOS el directorio raíz tiene un número de entradas fijo.

Las entradas que empiezan con ? indican ficheros borrados.

3.3.3. Linux/UNIX

LINUX / UNIX: Estructura de directorios única. Los dispositivos se asocian con un directorio para acceder a ellos. Las entradas de directorio de UNIX tendrá la siguiente información:



Como vimos anteriormente toda la información del archivo se guarda en el nodo i por lo que en UNIX un directorio es un caso especial de archivo que contiene registros con dos campos:

1. Uno de 2 bytes para almacenar el número de nodo i que guarda la información del archivo.

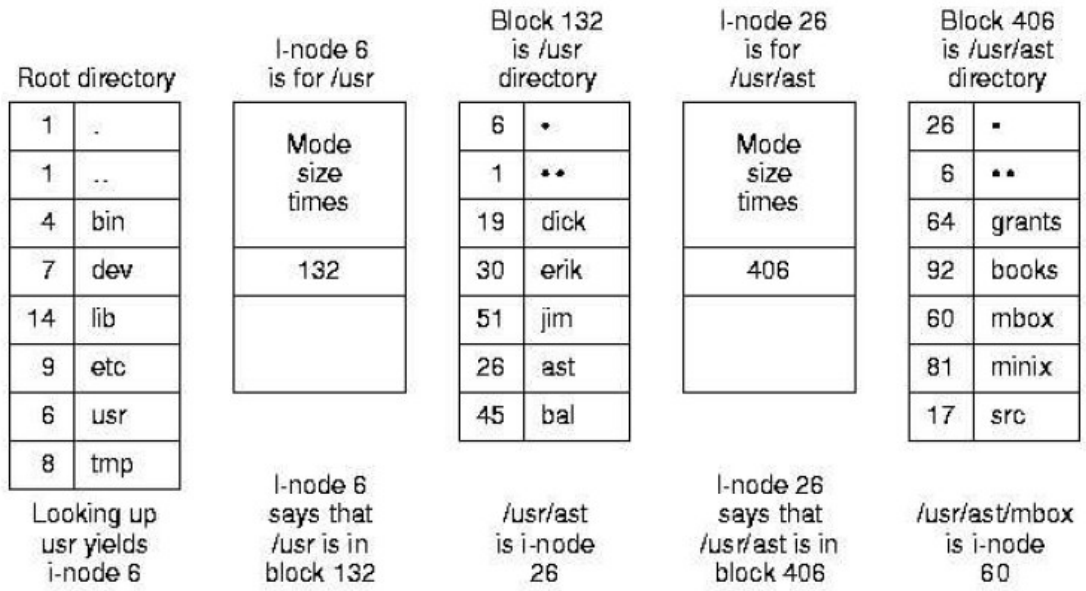
2. 14 bytes para el nombre del archivo.

Al igual que ocurre en MS-DOS. Cada directorio tendrá dos entradas de directorio que serán comunes a todos:

- '.' : Entrada de directorio que apunta al mismo directorio.

- '..' : Entrada de directorio que apunta al directorio anterior (que contiene a este).

Ejemplo: Pasos para buscar el fichero /usr/ast/mbox, la posición del i-nodo raíz es fija:



3.4. Compartición de archivos

Llamamos enlace o *link* a la capacidad de asociar un mismo archivo a dos o más directorios padres. De esta forma, una misma información puede ser accedida desde distintos puntos del árbol de directorios.

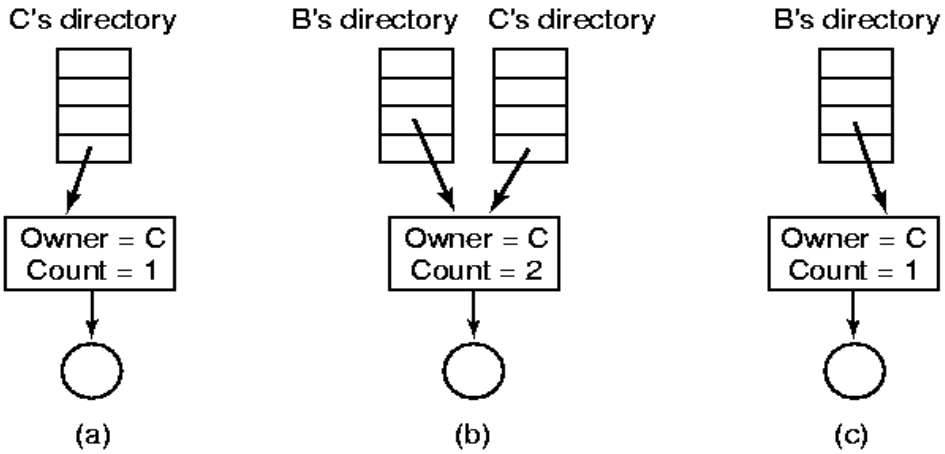
La estructura de árbol vista hasta ahora pasa a ser un grafo acíclico (sin ciclos).

En Windows nos podemos encontrar con los *accesos directos*, que serán una estructura parecida a la utilizada en otros sistemas operativos UNIX, concretamente un acceso directo es el equivalente a lo que vamos a denominar *enlace blando*.

En UNIX nos podemos encontrar con dos tipos de enlaces:

- **Enlace Duro** (Hard Link): En la entrada de directorio se tiene un apuntador a una estructura que guarda los datos relativos al fichero (apuntador a un i-nodo).

  Cuando se pretende realizar un enlace duro a un fichero existente, se incluye en directorio donde se va crear el enlace duro, el nombre del enlace a crear y como i-nodo el i-nodo del fichero al que se va a enlazar. De esta forma, dicho i-nodo es apuntado desde dos directorios diferentes. Para reflejar este hecho, en el i-nodo, el campo *número de enlaces* (el campo *count* de la figura) contendrá el número de veces que ese i-nodo es apuntado. Si el campo *número de enlaces* de un i-nodo es 2, esto quiere decir que a dicho i-nodo se puede acceder desde dos directorios diferentes.



Esto plantea un problema al realizar el borrado. Cuando se borra un fichero, se deberá eliminar la entrada del directorio, liberar el i-node que ocupa y marcar los bloques de datos de dicho fichero como libres. El problema es que si al i-nodo se accede desde más de una ubicación, hacer eso implicará que el resto de ubicaciones apunten a un i-nodo erróneo. Para corregir esto, al borrar un fichero se elimina su entrada del directorio y se decrementa en 1 el número de enlaces del i-nodo, y sólo si el resultado de esa operación es 0, se procede con el resto de las acciones. Si el resultado no es 0, el i-nodo y los bloque de datos permanecen intactos, puesto que indicará que aún se puede acceder a dicha información desde otra ubicación.

El único problema que plantea el borrado de un fichero que tiene enlaces duros, es que si el propietario de un fichero lo elimina, y dicho fichero tiene enlaces, al no eliminarse el i-nodo, el propietario del fichero sigue siendo el que lo creo, a pesar de haberlo borrado, y a efectos estadísticos sigue figurando como un fichero suyo.

- **Enlace Blando** (Soft Link): Se crea un nuevo fichero con la ruta al fichero al cual se enlaza. El nuevo fichero (enlace blando) es de un tipo especial (link) y su contenido es la ruta al fichero al cual apunta. Cuando se intenta acceder al fichero de tipo *link*, se detecta que es un enlace blando y el acceso se realiza a la ubicación indicada por dicho fichero.

En este caso, si el fichero original es borrado, el enlace queda roto, con lo cual los futuros intentos de accesos provocarán un error.

Ejemplo de creación de enlaces blandos en UNIX:

```
/~> ls -la

drwxr-xr-x    2 ifviana  48 2004-03-23 13:07 .

drwxr-xr-x   63 ifviana  3576 2004-03-23 13:07 ..

/~> touch prueba ; ln -s prueba p_sl

~/> ls -la

drwxr-xr-x    2 ifviana 104  2004-03-23 13:09 .

drwxr-xr-x   63 ifviana 3576 2004-03-23 13:07 ..

-rw-r--r--    1 ifviana 0    2004-03-23 13:09 prueba
```

```
lrwxrwxrwx    1 ifviana 6    2004-03-23 13:09 p_sl -> prueba
~/> rm prueba
drwxr-xr-x    2 ifviana 104  2004-03-23 13:09 .
drwxr-xr-x   63 ifviana 3576 2004-03-23 13:07 ..
lrwxrwxrwx    1 ifviana 6    2004-03-23 13:09 p_sl -> prueba
~/> cat p_sl
cat: p_sl: No existe el fichero o el directorio
```

Ejemplo de creación de enlaces duros en UNIX:

```
/~> ls -la
drwxr-xr-x    2 ifviana   48 2004-03-23 13:07 .
drwxr-xr-x   63 ifviana   3576 2004-03-23 13:07 ..
/~> touch prueba ; ln prueba p_hl
~/> ls -la
drwxr-xr-x    2 ifviana 104  2004-03-23 13:09 .
drwxr-xr-x   63 ifviana 3576 2004-03-23 13:07 ..
-rw-r--r--    2 ifviana 0    2004-03-23 13:09 prueba
-rwxrwxrwx    2 ifviana 0    2004-03-23 13:09 p_hl
~/> rm prueba
drwxr-xr-x    2 ifviana 104  2004-03-23 13:09 .
drwxr-xr-x   63 ifviana 3576 2004-03-23 13:07 ..
-rwxrwxrwx    1 ifviana 0    2004-03-23 13:09 p_hl
~/> cat p_hl
~/>
```

Comparativa entre ambos métodos:

	Hard Link	Soft Link
Ventajas	Ahorra espacio Más eficiente	Sin problemas al borrar
Inconvenientes	No se puede borrar si no tienes permisos	Más accesos a disco Ocupa más espacio en Disco

4. Protección de archivos

En el momento en que se almacena información dentro de un sistema informático, dicha información debe ser protegida:

- contra daños físicos (fiabilidad) y
- contra el acceso indebido (protección).

4.1. Fiabilidad

La fiabilidad se consigue realizando copias de seguridad o backups (lógicos y/o físicos) de los ficheros del sistema. Muchos de los sistemas operativos actuales copian automáticamente los ficheros de los discos en dispositivos de almacenamiento específicos, para así disponer de una copia en caso de que el sistema resultara destruido accidentalmente. Esta copia automática se realizará a intervalos regulares de tiempo (diariamente, semanalmente o mensualmente).

Los sistemas de ficheros pueden resultar dañados por diversas causas:

- problemas de hardware,
- errores de lectura/escritura,
- fluctuaciones o cortes en el suministro eléctrico,
- suciedad en las cabezas lectoras,
- temperatura excesiva en el sistema,
- errores en el software del sistema de ficheros.

4.2. Protección

Puede conseguirse de diferentes maneras:

- en sistemas monousuario, se podría conseguir retirando físicamente los disquetes y guardándolos bajo llave;
- en sistemas multiusuario, son necesarios otros mecanismos. Dos extremos:

No permitimos compartir información (acceder a ficheros de otros) -> prohibir completamente el acceso.

Permitir acceso libre, sin ninguna protección.

Es necesario una solución intermedia, los mecanismos de protección facilitan la labor de permitir un acceso controlado (permiten o deniegan acciones sobre objetos).

Operaciones que pueden controlarse:

- Lectura: Leer un fichero.
- Escritura: Escribir o reescribir en un fichero.
- Ejecución: Cargar un fichero en memoria y ejecutarlo.

- Adición: Escribir nueva información al final de un fichero.

- Borrado: Borrar el fichero y liberar su espacio para una posible reutilización del mismo.

Otras operaciones como renombrado, copiado o edición también se pueden controlar ya que para realizarse necesitan realizar alguna de las operación anteriores. Por ejemplo, el copiar un fichero se realizaría como una secuencia de operaciones de lectura, entonces un usuario con permiso de lectura también podría hacer que se copie el fichero, que se imprima, etc.

Las operaciones con directorios que deben protegerse son diferentes, ya que queremos controlar la creación y borrado de ficheros dentro de uno de ellos. Además, posiblemente desearemos controlar que un usuario pueda saber si un fichero está dentro de un directorio, ya que a veces, conocer la existencia y el nombre de un fichero puede ser significativo. Entonces, el listado con el contenido de un fichero también debe ser una operación protegida.

Mecanismos de protección:

- **Espacio de nombres**: dependen de la imposibilidad de los usuarios para acceder a un fichero que no se puede nombrar. Si un usuario no puede nombrar un fichero, entonces no puede operar sobre él (no existe un mecanismo para obtener los nombres de los ficheros de otros usuarios).

- **Contraseñas**: asociar una contraseña a cada fichero.

- **Control de acceso**: el acceso dependa de la identidad del usuario. Posiblemente usuarios distintos necesitarán tipos de acceso diferentes a un fichero o directorio. Puede asociarse una lista de acceso a cada fichero y a cada directorio, especificando el nombre del usuario y los tipos de acceso permitidos a cada uno de ellos. Cuando un usuario solicita un fichero determinado, el sistema operativo verifica su lista de acceso y si figura en la lista para el acceso solicitado, se le permite la operación. En otro caso, se produce un intento de violación de la protección y se deniega el acceso al usuario.

Algunos sistemas como las bases de datos, y “recientemente” los Sistemas Unix y Linux (comando `setfacl`), usan el esquema de control de acceso (GRANT).

Los sistemas operativos suelen usar una versión **condensada** de este tipo de protección de ficheros.

Condensada indica que intentan reducir la listas de acceso que sería necesario almacenar en caso de usar la idea de control de acceso original.

No podemos indicar individualmente los permisos que tiene cada usuario.

- Sólo nos permite controlar la protección a un fichero por grupo, en UNIX podemos das permisos al propietario, al grupo y al resto de usuarios. La lista de acceso asociada a un fichero se reduce a 9 bits.

- Podemos asociar la protección bien con el fichero mismo o bien con el camino utilizado para especificar el fichero.

5. Casos de estudio

5.1. F.A.T. 16

Fue inventada en 1977 por Bill Gates & Marc McDonald

Hereda características de CP/M.

Originalmente muy buena porque permitía a la FAT estar en memoria

Las restricciones de los nombres son heredadas de CP/M.

Comienza con FAT12, que puede direccionar 2^{12} = 4096 bloques de disco, es decir con sectores de 512 bytes hasta 2 Mbytes.

La FAT16 puede direccionar 2^{16} = 65535 bloques de disco, es decir con sectores de 512 bytes hasta 32 Mbytes.

Los discos mayores se pueden manejar agrupando en clusters (de 8 o 16 Kbytes). Para discos de 1 Gbyte necesitamos clusters de 32 Kbytes.

Hay dos copias de la FAT. La segunda sólo se emplea si la 1ª no se puede leer (error físico del disco)

El directorio / tiene un tamaño máximo. (En un disquete, un fichero ocupa como mínimo 512 bytes, y caben 224 archivos en el directorio /).

Las entradas de los directorios no están ordenas.

El directorio / y la FAT están al comienzo del disco (borde exterior).

La estructura de una partición FAT es la siguiente:

BOOT	1º COPIA DE LA FAT	2º COPIA DE LA FAT	DIRECTORIO RAÍZ	DATOS Y DIRECTORIOS

5.2. F.A.T. 32

Con la aparición del sistema operativo Windows 95, se produjo una actualización del sistema de ficheros FAT, como un intento de mejorar su rendimiento. Esta actualización conllevó un cambio de nombre, pasando a llamarse FAT32. Las características principales son:

- Aprovechamiento más eficiente del espacio de disco.
- FAT32 es un sistema de ficheros más robusto y flexible.
- Minimiza el efecto de la fragmentación de archivos.
- El principal cambio de una FAT a otra radica en la ampliación del tamaño de las entradas, que pasa de 16 a 32 bits.
- Con una entrada de FAT de 32 bits se permitirían discos de casi 4 Tbytes.
- FAT32 usa clústers más pequeños, en concreto de 4 Kb, por lo que la eficiencia aumenta y el espacio desperdiciado disminuye.

- FAT32 tiene la habilidad de recolocar el directorio raíz y utilizar la copia de seguridad de la FAT de forma efectiva.

- El boot record ha sido expandido para incluir una copia de seguridad de los datos críticos del sistema.

- El directorio raíz en un volumen FAT32 es ahora una simple cadena de clusters, que puede ser localizada en cualquier lugar de la unidad. Por esta razón, la limitación en FAT16 sobre el número de entradas del directorio raíz ahora ha desaparecido.

- La copia espejo de la FAT puede ser deshabilitada, permitiendo que otra copia de una FAT sea considerada como activa.

5.3. NTFS

Los sistemas de archivos FAT son muy sencillos y no proporcionan ninguna de las características de los sistemas de archivos modernos, tales como journaling, cuotas, seguridad, etc. Su única ventaja es la velocidad en volúmenes pequeños. A día de hoy los sistemas de archivos FAT se emplean solamente en memorias USB extraibles y disquetes.

Microsoft con la introducción del sistema operativo Windows NT, precursor de las versiones actuales de Windows, introdujo un nuevo sistema de archivos denominado NTFS. Pocos años después se convirtió en el sistema de archivos para discos duros estándar en todas las versiones de Windows. Uno de los problemas actuales de este sistema de archivos es que Microsoft, su creador, no ha publicado una especificación completa de dicho sistema de archivos, sino sólo ciertas partes del mismo

Es el sistema de archivos de Windows ideado para aprovechar al máximo discos grandes y procesadores rápidos.

Significa New Technology File System.

Proporciona una combinación de fiabilidad y compatibilidad que no se encuentran en FAT.

Desde el punto de vista del sistema de ficheros, todo en un volumen NTFS es un fichero o parte de él.

La estructura de un volumen NTFS básicamente es la siguiente:

BOOT	MFT (TABLA MAESTRA DE FICHEROS)	AREA DE DATOS

Donde la MFT es la estructura clave del sistema de archivos.

5.3.1. Boot

El boot puede ocupar varios sectores. (hasta 16 sectores de longitud).

Contiene la disposición del volumen y la estructura del sistema de ficheros

Hay un duplicado del boot en el centro lógico del disco.

Contiene la localización de la MFT.

Contiene el código de arranque.

5.3.2. Área de Datos

Es la zona donde se almacenarán los datos de los archivos y los directorios.

5.3.3. MFT

La (MFT), es una lista de todos los contenidos de este volumen NTFS organizada como un conjunto de filas en una estructura de base de datos relacional. Contiene al menos una entrada por cada fichero existente en el volumen, incluyendo la propia MFT.

Suele ocupar el 12,5% del espacio disponible en el volumen.

Es la zona donde se guarda la información sobre los ficheros y directorios almacenados en el volumen.

Está organizada en forma de tabla, donde cada entrada contiene información sobre un fichero o directorio.

Dado que la MFT almacena información sobre sí misma, las 16 primeras entradas están reservadas para información 'especial' (metainformación).

La tabla siguiente detalla los 16 primeros registros de la MFT.

Fichero	Nombre	Registro de la MFT	Descripción del fichero
Master file table	$Mft	0	Autodescribe a la propia MFT
Master file table 2	$MftMirr	1	Es un duplicado de los primeros registros de la MFT. Esto garantiza el acceso a la MFT en caso de fallo en el sector.
Log file	$LogFile	2	Contiene información sobre las transacciones. Se usa para recuperaciones rápidas tras fallos del sistema. El tamaño dependerá del tamaño del volumen.
Volume	$Volume	3	Contiene información sobre el volumen, como la etiqueta del volumen o la versión.
Attribute definitions	$AttrDef	4	Son los nombres de los atributos, su número y su descripción.
Root file name index	$	5	Es el directorio raíz.
Cluster bitmap	$Bitmap	6	Mapa de bits de los clusters del volumen.
Boot sector	$Boot	7	Incluye información para montar el volumen y el código de arranque.
Bad cluster file	$BadClus	8	Es la lista de clusters defectuosos del volumen.
Security file	$Secure	9	Contiene el descriptor de seguridad para todos los archivos dentro de un volumen.
Upcase table	$Upcase	10	Convierte los caracteres minúsculas para adaptarlos a los caracteres mayúsculas Unicode.
NTFS extension file	$Extend	11	Usado para extensiones opcionales, tales como cuotas, identificadores de objetos y datos sobre puntos de recuperación.
		12–15	Reservado para futuros usos.

A partir de la entrada 17ª, están colocados el resto de archivos y directorios.

La MFT reserva una cantidad de espacio para cada registro del fichero. Los atributos del fichero son escritos en ese espacio dentro de la MFT. Los ficheros y directorios pequeños (habitualmente 1500 bytes o menos) pueden ser ubicados completamente dentro de su entrada de la MFT.

Cada entrada de la MFT contiene la siguiente información:

Cabecera	Información Estándar	Nombre del fichero	Descriptor de seguridad	Datos

La cabecera es el número de entrada en la MFT,

La información estándar son las propiedades fundamentales del fichero, las fechas, permisos y resto de información sobre el fichero.

El descriptor de seguridad contiene información sobre los permisos del fichero (Listas de control de acceso)

Si los datos del fichero caben dentro de la entrada correspondiente de la MFT se colocan en el área de Datos.

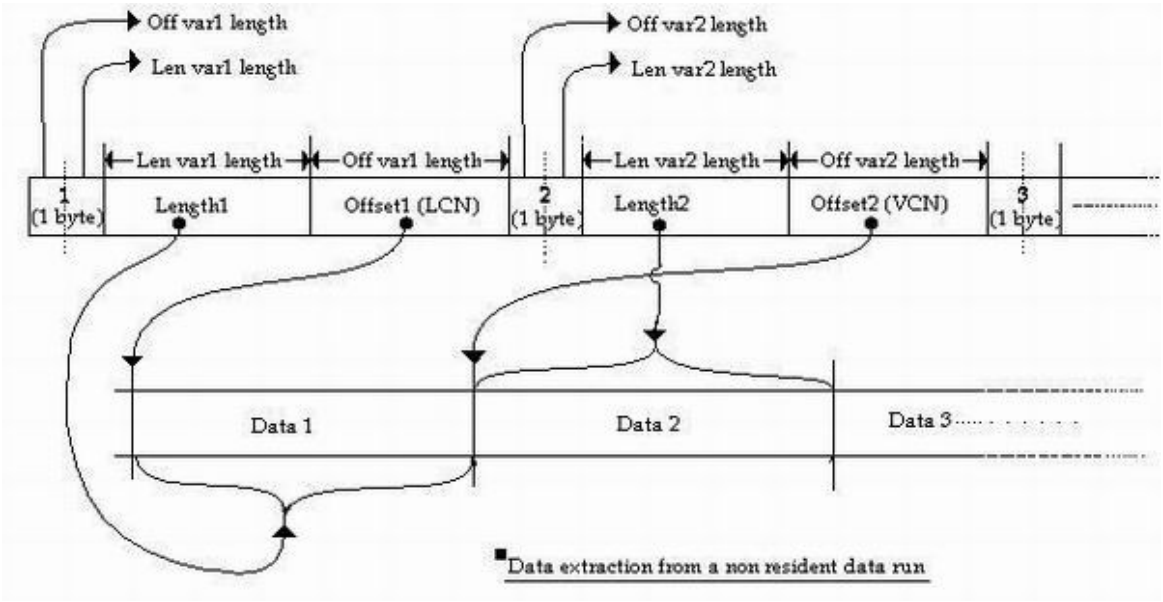
Si el fichero es más grande, y sus datos no caben en la MFT comienzan una serie de expansiones que sacan la información de la MFT de la siguiente forma:

- Se hacen los datos no residentes. Para ello la entrada de datos de la MFT no contiene los datos del fichero, sino punteros a extensiones. Una extensión es un conjunto de bloques (clústers) consecutivos, que contienen los datos del fichero.
  Cada puntero a una extensión contiene la longitud de la misma y un apuntador al 1º bloque de la extensión. Como son contiguas no es necesario leer uno para saber cual es el siguiente. Con esto se reduce la fragmentación.

- Si es fichero es más grande (o las extensiones están muy fragmentadas) y no hay suficiente espacio para almacenar todos los punteros, se colocan punteros a otros registros de la MFT que tendrán apuntadores a las extensiones. En este caso, el primer registro para el fichero (el registro base del fichero) contiene la localización del resto de los registros.

Estas operaciones se repiten tantas veces como sea necesario.

La siguiente figura muestra como se almacena la información sobre las extensiones.



5.3.4. Directorios en NTFS

Al contrario que en la FAT, los atributos y características de los ficheros se guardan en los propios ficheros, y no en los directorios.

Los directorios sólo guardan información del directorio.

Los directorios son considerados como ficheros y tienen sus propias entradas en la MFT.

La entrada del directorio es similar a la del fichero:

Cabecera	Información Estándar	Nombre del directorio	Descriptor de seguridad	Indices

En lugar de almacenar datos, los directorios almacenan los índices (nº de cabecera) de los ficheros que contienen.

Si el número de entradas de índice es lo suficientemente pequeño, éstos se colocan dentro del registro de la MFT.

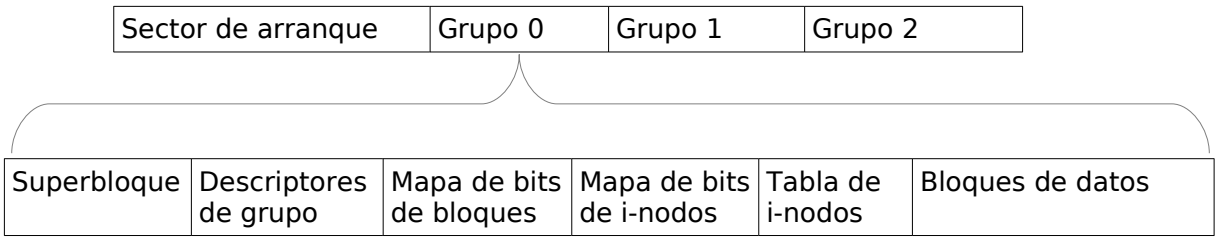
Si el número de entradas fuese mayor se actúa de la misma forma que en los ficheros (apuntando a extensiones).

Para mejorar la búsqueda en directorios grandes, NTFS almacena la información de las entradas del directorio en forma de árbol-B, que localiza la información mucho más rápidamente. Se pierde tiempo en la inserción de una nueva entrada, pero se gana mucho al recuperarla.

5.4. EXT2 y EXT3

Se trata de los sistemas de archivos más empleados en Linux. Ext3 es básicamente un sistema de archivos Ext2 al que se le ha añadido un registro de transacciones. Nos referiremos a ellos de forma genérica como Ext.

La organización de un volumen Ext se muestra en la figura.



Los primeros 1024 bytes del volumen están reservados. Pueden contener el código de arranque del volumen cuando el sistema operativo arranca desde el volumen.

El resto del volumen se divide en bloques de igual tamaño. El concepto de bloque es análogo al concepto de cluster de los sistemas de archivos FAT y NTFS.

Los bloques se organizan en grupos, conteniendo todos ellos el mismo número de bloques, excepto el último que puede ser más pequeño.

El superbloque tiene un tamaño de 1024 bytes (aunque muchos de estos bytes no los usa) y contiene los parámetros del sistema de archivos. Todos los grupos tienen una copia del superbloque al comienzo del grupo, aunque en la práctica sólo se utiliza el superbloque del primer grupo, salvo que éste esté dañado. El superbloque contiene información como el tamaño del bloque, el número de bloques del sistema, nombre del volumen, fecha y hora en la que se montó por última vez el volumen, etc.

La tabla de descriptores de grupo es una tabla de tamaño 1 bloque situada después del superbloque y comenzando en un bloque nuevo. Por lo tanto, puede haber sectores que no se usan entre el superbloque y la tabla de descriptores. La tabla de descriptores contiene entradas de 32 bytes que describen el grupo del que forma parte. Cada una de estas entradas proporciona:

- Número de bloque de comienzo del mapa de bits de bloques,
- Número de bloque de comienzo del mapa de bits de i-nodos
- Número de bloque de comienzo de la tabla de i-nodos
- Número de bloques que aún no han sido asignados en el grupo.
- Número de i-nodos que aún no han sido asignados en el grupo.
- Número de directorios en el bloque.

La contabilidad de los bloques que han sido asignados dentro de cada grupo se lleva a cabo con el mapa de bits de bloques. El mapa de bits de bloques comienza justo después de la tabla de descriptores del grupo y tiene un tamaño de 1 bloque.

El mapa de bits de i-nodos tiene un tamaño de 1 bloque y funciona de forma análoga al mapa de bits de bloques. Cada entrada de la tabla de i-nodos tiene asociado un bit que se activa cuando la entrada es usada o se trata de una entrada reservada.

La tabla de i-nodos comienza en el bloque indicado en el descriptor del grupo y tiene un tamaño especificado en el superbloque. Cada una de sus entradas es un i-nodo que hace referencia a un archivo o directorio. Las primeras entradas de la tabla están reservadas. Por ejemplo, la entrada número 2 contiene el i-nodo del directorio raíz.

Finalmente, comentar que Ext3 para llevar a cabo el registro de transacciones emplea un archivo cuyo i-nodo es típicamente el número 8, aunque se puede especificar en el superbloque

5.5. Sistema de ficheros con journaling

Para mejorar el rendimiento de las operaciones de E/S, los datos del disco son temporalmente almacenados en la memoria RAM a través del page-cache y buffer-cache (caso de linux).

Los problemas surgen si hay un corte de suministro eléctrico antes que los datos modificados en la memoria (dirty buffers) sean grabados nuevamente al disco, se producen inconsistencias

El fsck (file system check) fue la herramienta que resolvía dichas inconsistencias, pero el fsck tiene que analizar la partición completa y verificar las interdependencias entre i-nodos, bloques de datos y contenidos de directorios.

Si la partición es muy grande el tiempo que se necesita para hacer la recuperación es extremadamente alto.

Un sistema con journaling es un sistema de ficheros tolerante a fallos en el cual la integridad de los datos está asegurada porque las modificaciones de la meta-información de los ficheros es primero grabada en un registro cronológico (log o journal) antes que los bloques originales sean modificados.

En el caso de un fallo del sistema el módulo de recuperación analizará el registro y sólo repetirá las operaciones incompletas en aquellos ficheros inconsistentes, es decir que la operación registrada no se haya llevado a cabo finalmente.

Algunos sistemas de ficheros que soportan journaling:

En Linux

- ReiserFS (de Namesys).
- XFS (de Silicon Graphics).
- JFS (de IBM).
- Ext3 (de Stephen Tweedie).

En MACOS:

- Apple's HPS
- En Solaris:
- UFS

En Windows:

- NTFS

Desventajas de los sistemas con journaling:

- Se realizan más operaciones de entrada/salida.
- Como consecuencia de la desventaja anterior se produce más fragmentación.