Sistemas de Archivos

Introducción

Todas las aplicaciones de la computadora requieren almacenar y recuperar información. Mientras un proceso está en ejecución, puede almacenar una cantidad limitada de información dentro de su propio espacio de direcciones. Sin embargo, la capacidad de almacenamiento está restringida por el tamaño del espacio de direcciones virtuales (lo cual puede ser demasiado pequeño).

Por otro lado, cuando un proceso termina, la información que hay en el espacio de direcciones se pierde, por lo que hay que buscar una forma de guardarla adecuadamente.

Un tercer problema es que frecuentemente es necesario que varios procesos accedan a cierta información simultáneamente. La manera de resolver este problema es hacer que la información en sí sea independiente de cualquier proceso.

Definiciones y Conceptos

- Los **archivos** son unidades lógicas de información creada por los procesos.
- La información que almacena en los archivos debe ser **persistente**, es decir, no debe ser afectada por la creación y terminación de procesos (sólo debe desaparecer cuando el propietario lo elimina de forma explícita).
- Los archivos son administrados por el **Sistema Operativo**. La parte del SO que trata con los archivos se conoce como **sistema de archivos** y es el tema del capítulo.

Consideraciones Generales

1. **Disico Físico**: Dispositivo de almacenamiento permanente. Contiene un conjunto de bloques de tamaño fijo donde cada bloque tiene un identificador.
2. **Disco Lógico**: Abstracción que el SO ve como una secuencia lineal de bloques accesibles aleatoriamente. El driver traduce los números de bloque lógico a bloque físico.

El Disco Físico se divide en particiones físicas contiguas, cada una asociada a un disco lógico.

- Un Sistema de Achivos está contenido por completo en un Disco Lógico.
- Un Disco Lógico puede contener un sólo Sistema de Archivos (o el swap).
- Un Disco Lógico puede estar formado por varios discos físicos.
-

Nomenclatura de Archivos

Las reglas exactas para denominar archivos varían un poco de un sistema a otro. Por ejemplo, algunos sistemas de archivos diferencian mayúsculas de minúsculas (como UNIX) y otros no (como MS-DOS).

Por otro lado, Windows 95 y 98 usan el mismo sistema de archivos que MS-DOS conocido como **FAT-16**. Más tarde otros Windows comenzaron a implementar el sistema de archivos **FAT-32**. A partir de Windows NT admiten ambos sistemas de archivos, aunque en realidad ya son obsoletos.

Estos SO tienen un sistema de archivos nativo NTFS con diferentes propiedades.

Extensiones de Archivos

Muchos SO aceptan nombres de archivos en dos partes, separadas con un punto *main.c*. La parte que va después del punto se conoce como **extensión del archivo** y por lo general indica algo sobre su naturaleza.

En el caso de UNIX las extensiones de archivo son sólo convenciones y no son obligatorias, además, también permite más de una extensión (como *paginaWeb.html.zip*).

En algunos casos las convenciones son muy recomensables de seguir, por ejemplo, algunos compiladores de C leen la extensión del archivo para comprobar si se trata de archivo de código.

Estructuras de Archivos

Los archivos se pueden estructurar en una de varias formas (como se muestra en la figura)

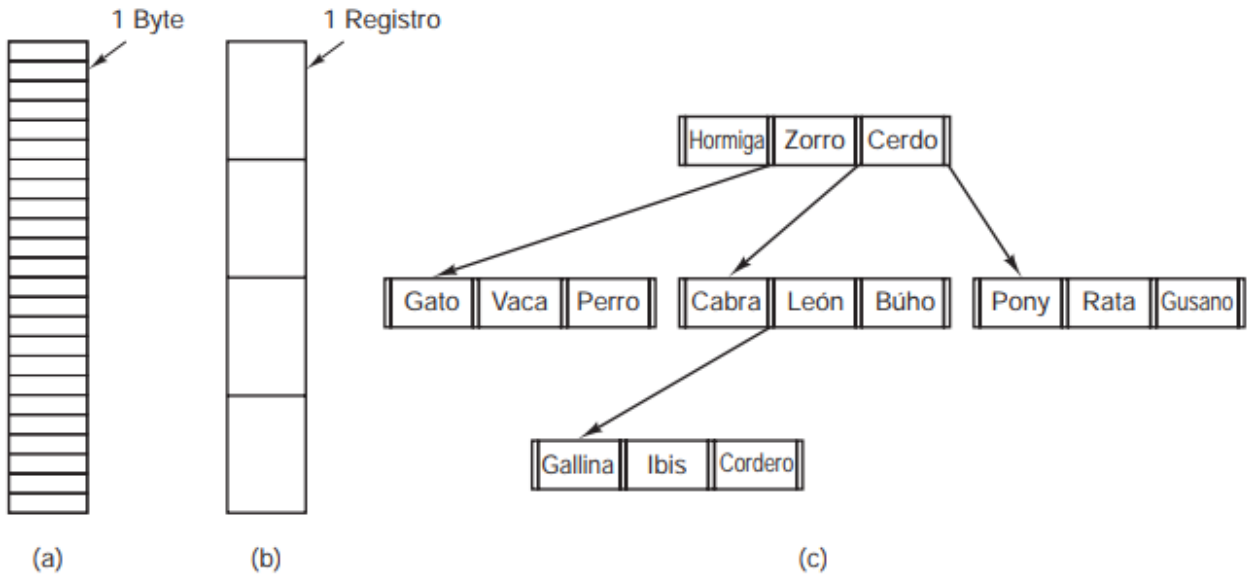


Figura 4-2. Tres tipos de archivos. (a) Secuencia de bytes. (b) Secuencia de registros. (c) Árbol.

Tanto UNIX como Windows utilizan la **secuencia de bytes sin estructura**: el SO no sabe, ni le importa lo que hay en el archivo (todo lo que ve son bytes). Cualquier significado debe ser impuesto por los programas a nivel usuario (lo cual provee la máxima flexibilidad al permitir que los programas de usuario coloquen cualquier cosa que quieran y lo denominen de la manera más conveniente que vean).

En el modelo 'b' de la imagen, el archivo es una **secuencia de registros**: cuando lee o escribe lo hace sobre uno de los registros. Y si necesita agregar nueva información agrega un registro.

En el modelo 'c' de la imagen muestra un **árbol de registros** (no todos necesariamente de la misma longitud). Cada registro contiene un **campo llave** en una posición fija dentro de cada registro, que el árbol utiliza para ordenar el árbol y realizar búsquedas rápidamente. Aquí para obtener un registro que se quiera, es necesario tener la llave.

Tipos de Archivos

1. Los **archivos regulares** son los que contienen información de usuario (nos centraremos en estos).
2. Los **directorios** son sistemas de archivos para mantener la estructura del sistema de archivos.
3. Los **archivos especiales de caracteres** se relacionan coon la E/S.
4. Los **archivos especiales de bloques** se utilizan para modelar discos.

Archivos Regulares

Los Archivos Regulares pueden ser Archivos ASCII o binarios

El contenido de los **Archivos ASCII** son líneas de texto. La ventaja es que si la mayoría de programas usan este tipo de archivos no sólo para expresar texto regular, sino para E/S, es fácil conectar la salida de un programa con la entrada de otro (el pipe de shell poro ejemplo).

El contenido de los **Archivos Binarios** es un contenido incomprensible de caracteres. Por lo general tienen cierta estructura interna conocida para los programas que los utilizan.

Máscara de Archivo

Cada archivo tiene asociada una máscara de 16 bits **Máscara de Modo**. Esta contiene:

1. Tipo de Archivo.
2. Digitos Locales.
3. Permisos Propietario.
4. Permisos Grupo.
5. Permisos de otros.

Acceso a Archivos

Los primeros Sistemas Operativos proporcionaban sólo un tipo de acceso: **acceso secuencial** (útil cuando el medio de almacenamiento era cinta magnética en lugar de disco).

Cuando se empezó a usar discos para almacenar archivos, se hizo posible leer registros en distinto orden. Estos archivos se llaman **archivos de acceso aleatorio**.

1. *read* da la posicion del archivo en la que se va a empezar a leer.
2. *seek* establece la posición actual donde empezar a leer.

Atributos de Archivos

Es necesario asociar información adicional a los archivos para que el SO los pueda gestionar mejor. A estos elementos se le llaman **atributos** o **metadatos**

Atributo	Significado
Protección	Quién puede acceso al archivo y en qué forma
Contraseña	Contraseña necesaria para acceder al archivo
Creador	ID de la persona que creó el archivo
Propietario	El propietario actual
Bandera de sólo lectura	0 para lectura/escritura; 1 para sólo lectura
Bandera oculto	0 para normal; 1 para que no aparezca en los listados
Bandera del sistema	0 para archivos normales; 1 para archivo del sistema
Bandera de archivo	0 si ha sido respaldado; 1 si necesita respaldarse
Bandera ASCII/binario	0 para archivo ASCII; 1 para archivo binario
Bandera de acceso aleatorio	0 para sólo acceso secuencial; 1 para acceso aleatorio
Bandera temporal	0 para normal; 1 para eliminar archivo al salir del proceso
Banderas de bloqueo	0 para desbloqueado; distinto de cero para bloqueado
Longitud de registro	Número de bytes en un registro
Posición de la llave	Desplazamiento de la llave dentro de cada registro
Longitud de la llave	Número de bytes en el campo llave
Hora de creación	Fecha y hora en que se creó el archivo
Hora del último acceso	Fecha y hora en que se accedió al archivo por última vez
Hora de la última modificación	Fecha y hora en que se modificó por última vez el archivo
Tamaño actual	Número de bytes en el archivo
Tamaño máximo	Número de bytes hasta donde puede crecer el archivo

Figura 4-4. Algunos posibles atributos de archivos.

Operaciones de Archivos

1. *Create*: El archivo se crea sin datos. El propósito de la llamada es anunciar la llegada del archivo y establecer algunos de sus atributos.
2. *Delete*: Cuando el archivo ya no se necesita, se tiene que eliminar para liberar el espacio en disco. Siempre con una llamada al sistema.
3. *Open*: El propósito de esta llamada es permitir que el sistema lleve los atributos y la lista de direcciones de disco a memoria principal para tener acceso rápido a llamadas posteriores.
4. *Close*: Cuando ya no es necesario referirse al archivo en memoria principal se debe cerrar el archivo para liberar espacio en la tabla interna.
5. *Read*: Los datos se leen del archivo. Por lo general, los butes provienen de la posición actual.
6. *Write*: Los datos se escriben en el archivo otra vez, por lo general en la posición actual (si es el final del archivo, este aumenta de tamaño y si es en mitad del archivo se sobresctiben).
7. *Append*: Es una forma restringida de *write*. Sólo puede agregar datos al final del archivo.
8. *Seek*: Para los archivos de acceso aleatorio, se necesita un método para especificar de dónde se van a tomar los datos. Esta llamada reposiciona el apuntador del archivo en una posición específica del archivo.
9. *Get Atribbutes*: Obtiene los atributos del archivo, el programa 'make', por ejemplo, necesita obtener los tiempos de modificación para ver si necesita recompilar o ya está actualizado el ejecutable que genera.
10. *Set Atribbutes*: Algunos atributos puede establecer el usuari y se pueden mofificar. La mayoría de banderas que se aprecia en la última imagen caen en esta categoría.
11. *Rename*: El usuaio cambia el nombre sdel archivo.

Por supuesto existen más **llamadas al sistema** pero estas son las más comunes y merecía la pena echarles un vistazo en este documento.

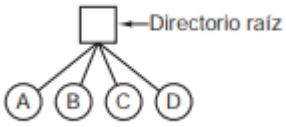
```
#define TAM_BUF     4096
#define MODO_SALIDA 0700

int main(int argc, char *argv[])
{
    int     ent_da;
    int     sal_da;
    int     leer_cuenta;
    int     escribir_cuenta;
    char    buffer[TAM_BUF];

    if (argc != 3)  exit(1);                    // Error de sintaxis

    ent_da = open(argv[1], O_RDONLY);           // Abrimos el archivo
(permisos de lectura)
    if (ent_da < 0) exit(2);
    sal_da = creat(argv[2], MODO_SALIDA);       // Creamos archivo destino
(full permisos al owner)
    if (sal_da < 0) exit(3);

    while(TRUE)
    {
        leer_cuenta = read(ent_da, bufer, TAM_BUFER);       // Leemos de
'ent_da' tantos bytes como 'TAM_BUFER' y guardarlos en 'bufer'
        if (leer_cuenta <= 0)   break;
        escribe_cuenta = write(sal_da, bufer, leer_cuenta); // Escribimos
en 'sal_da' lo que pone en 'bufer' los bytes que retorne 'leer_cuenta'
('read' devuelve el numero de bytes leidos)
        if (escribir_cuenta <= 0)   exit(4);
    }

    close(ent_da);
    close(sal_da);
    if (lee_cuenta == 0)
    {
        exit(0);
    } else
    {
        exit(5);
    }
}
```

Directorios

Para llevar el registro de los archivos, los sistemas de archivos por lo general tienen **directorios** o **carpetas**.

Sistemas de Directorios de Un Nivel

La forma más simple de un sistema de dorectorios es tener un directorio que contenga todos los archivos. Tiene el nombre de **Directorio Raiz** aunque si es el único directorio... el nombre no importa mucho.



Sistemas de Directorios Jerárquicos

Para los computadores del S. XXI con miles de archivos es imposible encontrar algo si todos los archivos estuvieran en un sólo directorio.

Lo que se necesita es una jerarquía (un **árbol de directorios**). Además, si varios usuarios comparten un servidor de archivos común, como se da en el caso de muchas redes de empresas, cada usuario puede tener un directorio raíz privado para su propia jerarquía.

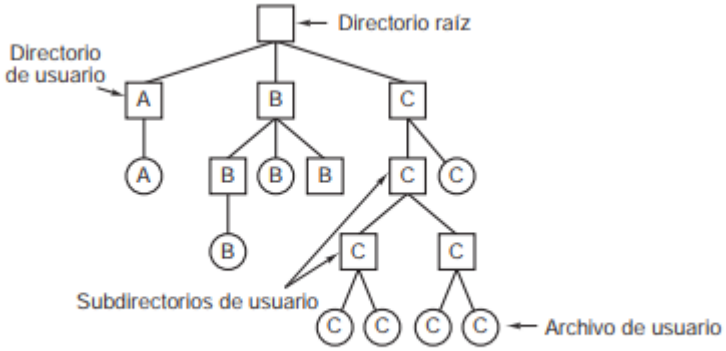


Figura 4-7. Un sistema de directorios jerárquico.

Nombres de Rutas

Cada archivo recibe un **nombre de ruta absoluto** qus consiste en la ruta desde el directorio raíz hasta al archivo (/home/Tanenbaum/Escritorio/mailbox)

```
En UNIX, los componentes de la ruta van separados por /.
En Windows, el separador es \.
En MULTICS era >
```

Cuando utilizamos el primer caracter del nombre del archivo '/' (en el caso de UNIX) nos estamos refiriendo a una ruta absoluta.

Por otro lado, están los **nombres de ruta relativa**. Los cuales se utilizan dentro del concepto de **directorio de trabajo** o **directorio actual**. Esto quiere decir que, cuando el usuario designa un directorio de trabajo (por ejemplo, entrando mediante 'cd' hasta un directorio como por ejemplo, 'Escritorio'), dentro del directorio de trabajo los nombres de las rutas que no empiezen por el separador se toman en forma relativa.

```
cp /home/Tanenbaum/Escritorio/mailbox
/home/Tanenbaum/Escritorio/mailbox.bak
ó estableciendo 'Escritorio' como directorio de trabajo:
cp mailbox  mailbox.bak
```

La mayoría de los SO que proporcionan un sistema de directorios jerárquicos tienen dos entradas especiales en cada directorio: '.' y '..'

- **.** : Se refiere al directorio actual.
- **..** : Se refiere a su padre (excepto en el directorio raíz que se refiere a sí mismo).

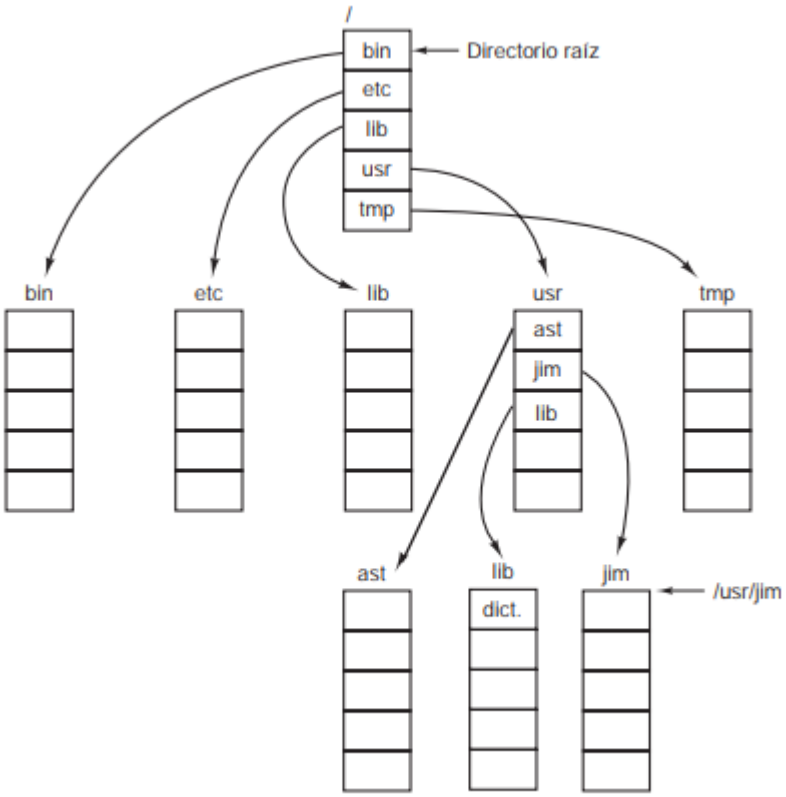


Figura 4-8. Un árbol de directorios de UNIX.

Operaciones de Directorios

1. *Create*: Se crea un directorio. Está vacío, excepto por . y .. que el sistema crea de manera automática (por ejemplo *mkdir*).
2. *Delete*: Se elimina un directorio únicamente si está vacío (. y .. no cuentan).
3. *Opendir*: Abre un directorio, se suele usar con otras operaciones como leer (para leer los archivos que contiene el directorio).
4. *Closedir*: Cuando se ha leído un directorio se debe cerrar para liberar espacio en la tabla interna.
5. *Readdir*: Esta llamada devuelve la siguiente entrada en un directorio abierto. Esta llamada devuelve siempre una entrada en formato estándar (a diferencia que *read* que necesita saber a priori la estructura interna de los directorios).
6. *Rename*: Cambiar el nombre del directorio.
7. *Link*: El ligado es una técnica que permite a un archivo aparecer en más de un directorio (crea un vínculo entre un archivo y una ruta). A este vínculo se le llama **vínculo duro** o **liga dura**.
8. *Unlink*: Se elimina una entrada de directorio (si estuviera únicamente en un sólo directorio se quita completamente del sistema de archivos).

Un **vínculo duro** crera una réplicia exacta del origen y a la vez se vincula a él, de tal forma que un cambio en el original se actualiza de inmediato en su vínculo fuerte y viceversa. (Si eliminamos el arcchivo original, su contenido aún es accesible a través del vínculo (al ser una réplica)).

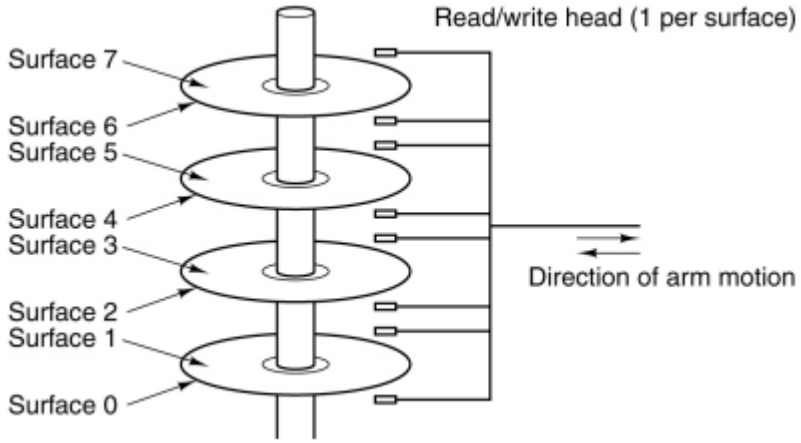
El **vínculo simbólico** es un apuntador que referencia a un archivo en otro lugar. Leen la información almacenada en el destino u origen y se comportan como si fuesen reales. El vínculo simbólico no ocupa más espacio ya que el contenido queda en el archivo original.

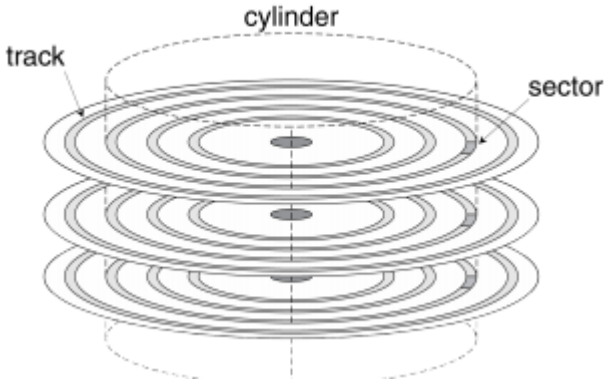
Implementación de Sistema de Archivos

Discos

Los sistemas de Archivos se almacenan en discos. Cada disco tiene varios **platos**. Cada plato está dividido en círculos concéntricos o **tracks**. Verticalmente las tracks forman un **cilindro**.

Las tracks se dividen en sectores, teniendo en cuenta que el número de sectores no es el mismo en todas las tracks.





Distribución del Sistema de Archivos

Los Sistemas de Archivos se almacenan en discos. La mayoría de los discos se pueden dividir en una o más particiones, con Sistemas de Archivos independientes en cada partición.

El sector 0 del disco se conoce como **MBR** (*Master Boot Record*) y se utiliza para arrancar la computadora. El final del MBR contiene la tabla de particiones (que proporciona el inicio y fin de cada partición). En la tabla, una de las particiones se marca como **activa**.

Cuando se arranca la computadora, el BIOS lee y ejecuta el MBR, lo primero que hace el MBR es localizar la partición activa y ejecutar su primer bloque llamado **Bloque de Arranque** que es el encargado para ejecutar el Sistema Operativo.

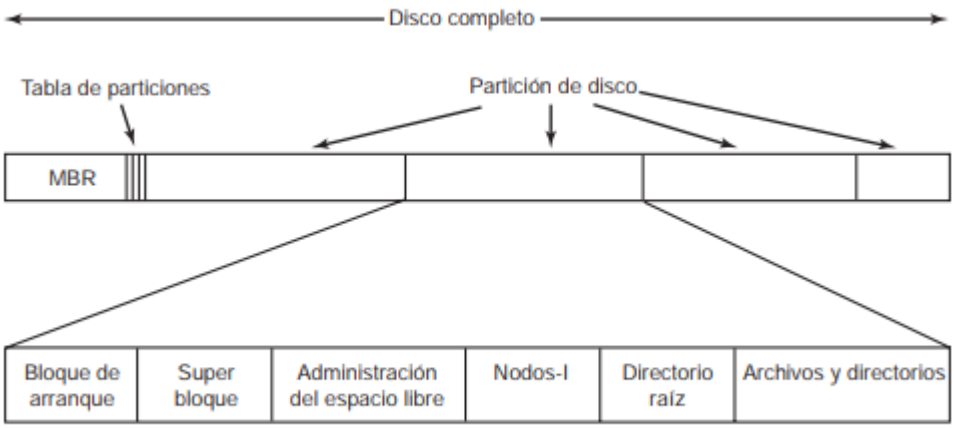


Figura 4-9. Una posible distribución del sistema de archivos.

Para arrancar **cualquier partición de disco**, es necesario entrar en su primer bloque como hemos visto antes para arrancar el SO. El siguiente es el **Superbloque** que contiene todos los parámetros clave acerca del sistema de archivos: la información típica incluye el **número mágico** para identificar el tipo de sistema de archivos.

A continuación puede venir información adicional como el espacio libre, los **i-nodos** (un arreglo de estructuras de datos, uno por archivo que indica todo acerca del archivo)...

Implementación de los Archivos

Probablemente la cuestión más importante al implementar el almacenamiento de archivos es tener un registro acerca de **cuál archivo va el cuál bloque**.

Asignación Contigua

Esquema de asignación más simple. De esta forma, un disco con bloques de 1KB, a un archivo de 50KB se le asignarán 50 bloques consecutivos. Si el archivo fuera de 49,5KB el último bloque no estaría lleno y se desperdiciaría un poco de espacio, ya que el siguiente archivo empezaría en uno nuevo.

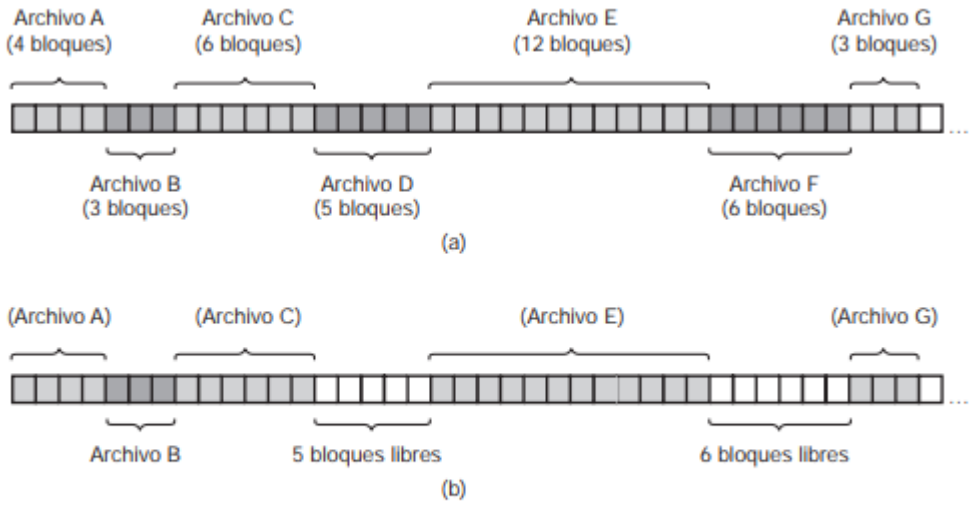


Figura 4-10. (a) Asignación contigua de espacio de disco para siete archivos. (b) El estado del disco después de haber removido los archivos *D* y *F*.

Ventajas:

- Es fácil de implementar: Ubicación = dirección primer bloque + nº bloques.
- El rendimiento de lectura es óptimo ya que están contiguos los bloques y sólo necesita una búsqueda.

Desventajas:

- Fragmentación de disco: compactación o lista de huecos libres para reutilizarlos :
  - Reutilización del espacio complicada (es necesario saber el espacio del archivo a priori).
  - Copiar todos los bloques que están después de los huecos es muy costoso.

Asignación de Lista Enlazada (Ligada)

La primera palabra de cada bloque se utiliza como apuntador al siguiente, el resto del bloque es para los datos.

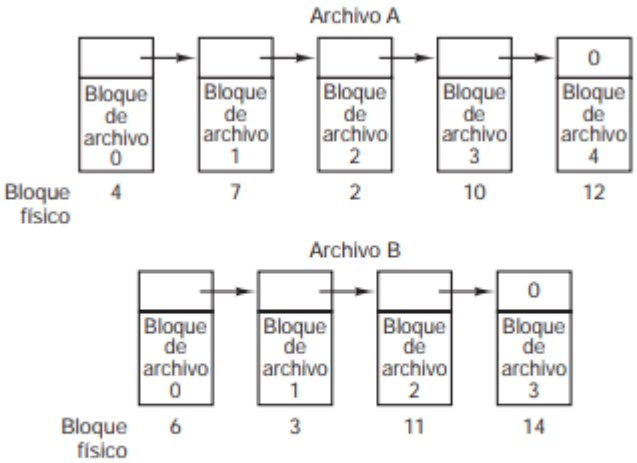


Figura 4-11. Almacenamiento de un archivo como una lista enlazada de bloques de disco.

Ventajas:

- Este método permite utilizar cada bloque del disco y no se pierde espacio debido a la fragmentación (excepto la fragmentación interna del último bloque).

- Localización: dirección del primer bloque.

Desventajas:

- Acceso muy lento: para llegar al bloque n, hay que leer los n-1 anteriores
- Leer un bloque de datos implica acceder a dos bloques de disco (con los primeros bytes de cada bloque ocupamos un apuntador al siguiente y este mismo ocupa espacio).

Asignación de Lista Enlazada Utilizando Tabla en Memoria

Ambas desventajas de la asignación de lista enlazada se pueden eliminar si tomamos la palabra del apuntador de cada bloque de disco y la colocamos en una tabla de memoria. En este caso el final de la cadena se marca con un marcador especial (como en la imagen que es -1).

Esta **Tabla en Memoria Principal** se conoce como **FAT** (File Allocation Table).

Ventajas:

- El bloque completo está disponible para datos.
- Acceso aleatorio más sencillo: la cadena de apuntadores está en Memoria Física en vez de disco.

Desventajas:

- Toda la información en memoria: **poco escalable** y debe estar **presente todo el tiempo**.

I-Nodos

El último método es asociar con cada archivo una **estructura de datos** conocida como **I-Nodo**, que lista los atributos y las direcciones del disco de los bloques del archivo.

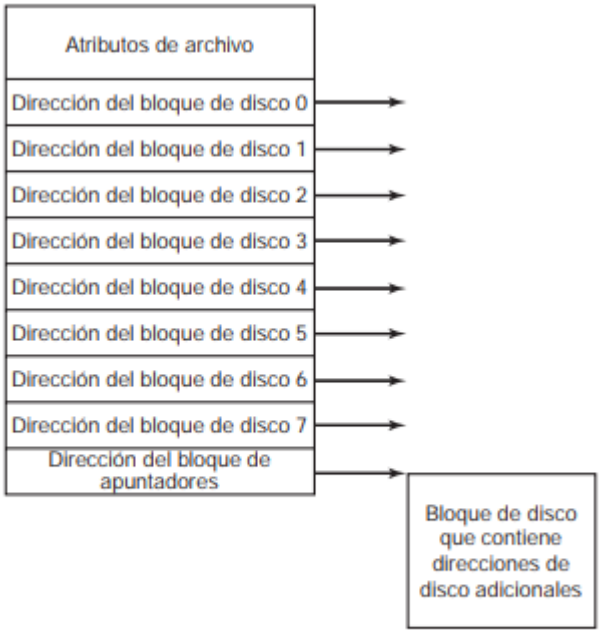


Figura 4-13. Un nodo-i de ejemplo.

Ventajas:

- Sólo necesita estar en memoria si el archivo está abierto.
- Tamaño proporcional al número de archivos abiertos (no depende de disco).
- El espacio que ocupa la tabla es más pequeño que los arreglos anteriores.

Existe un número máximo de apuntadores por lo que en cierto modo limita el tamaño del archivo, la posible solución es que el último apuntador apunte a otro bloque de i-nodos con direcciones de bloque de archivos.

Implementación de Directorios

Cuando se abre un archivo, el SO utiliza el nombre de la ruta para reconocer la entrada del directorio. La función principal del sistema de direcotios es asociarel nombre ASCII del archivo a la información necesaria para localizar datos.

Los sistemas que **no utilizan I-Nodos** pueden hacer como la figura 1 y tener en cada directorio los archivos y para cada archivo sus atributos.

Los sistemas que **sí utilizan I-Nodos** pueden almacenar un número de I-Nodo.

Archivos de Nombre Largo

La alternativa a tener longitud estática de nombres es crear un nuevo campo en la entrada el archivo llamado **longitud de entrada del archivo** seguido de los **atributos** y de todo el nombre del archivo cuya longitud hemos definido en el primer campo. (Como se muestra en el apartado 'a' de la imagen).

Otro método es dejar que las entradas de directorio sean de longitud fija y mantener los nombres de los archivos juntos en el heap al final del directorio. Esto tiene la ventaja de que cuando se borre una entrada, el siguiente archivo a introducir siempre cabrá ahí (al estar en el heap).

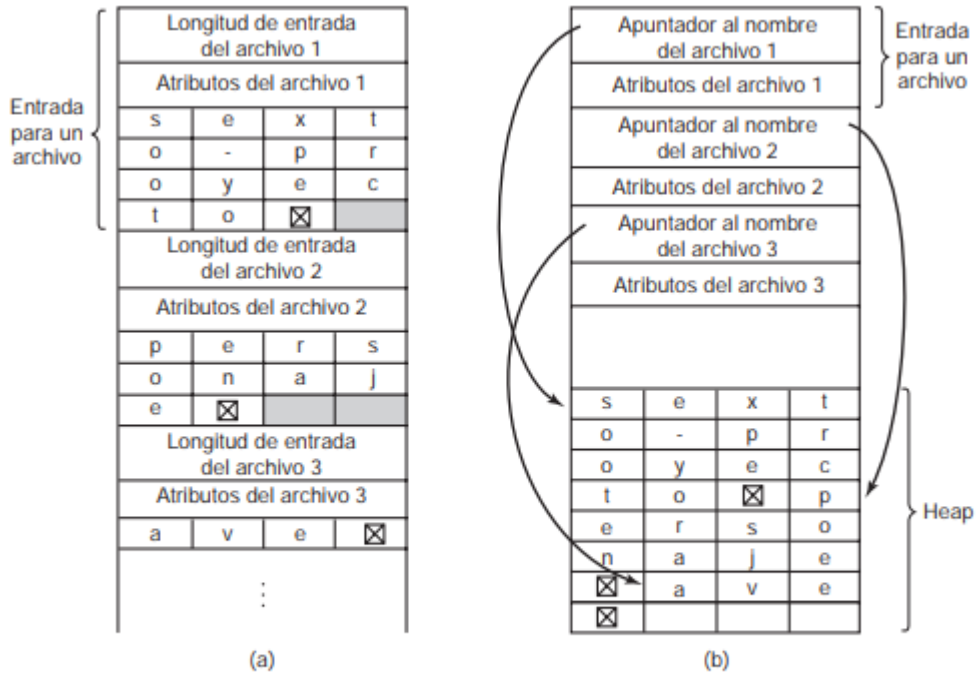


Figura 4-15. Dos maneras de manejar nombres de archivos largos en un directorio. (a) En línea. (b) En un heap.

Archivos Compartidos

A veces, es conveniente que aparezca un archivo compartido de forma simultánea en distintos directorios que pertenezcan a distintos usuarios. Como se ve en la imagen, hay una conexión entre el directorio 'B' y el archivo dentro de 'C' conocido como **vínculo**.

El sistema de archivos en sí ahora es un **Gráfo Acíclico Dirigido** (DAG)

```
No sirve tener una copia de las direcciones de disco en cada directorio ya
que necesitamos tener la misma información en los dos. Es decir, si 'B' o
'C' agregan nuevos bloques, estos deben ser visibles para ambos.
```

- **Nodo-I**: Los directorios apuntan al Nodo-I del archivo que mantiene la información sobre bloques.
- **Vínculo simbólico**: uno de los directorios sólo tiene la ruta del archivo vinculado.

Sistemas de Archivo Estructurado por Registros

Esta técnica consiste diseñar un tipo de sistema de archivos completamente nuevo, llamado **LFS**.

La idea que impulsó el diseño de LFS es que, dado que las **CPU son más rápidas** y las **RAM más grandes**, las cachés de disco también incrementan con rapidez. Por lo tanto, se pueden **satisfacer todas las peticiones directamente del caché** sin necesidad de accesos a disco.

Por otro lado, escribir fragmentos pequeños es muy ineficiente ya que se tiene que hacer en disco y es una ardua tarea que ocupa mucho tiempo. La solución es almacenar todas las **escrituras pendientes en un buffer en memoria**. Estas escrituras se escriben en disco como un **sólo segmento al final del registro** (con Nodos-I, Bloques de Datos y de Directorios).

El problema es que la **búsqueda del Nodo-I es más dificil** ya que están en otros sitios.

Sistema de Archivos por Bitátoca

Un ejemplo de estos Sistemas de Archivos son NTFS y ext3.

Esta técnica mantiene un registro con los pasos que debe seguir el sistema de archivos para una cierta operación antes de hacerla. Lo que proporciona una gran **robustez frente a fallos**. Si falla, se acude al registro para saber los pasos para terminar.

El registro se borra al terminar la operación.

Administración Y Optimización del Sistema de Archivos

División en Bloques

Existen dos alternativas para almacenar un archivo:

- Bytes consecutivos en disco (poco eficiente).
- Dividirlos en bloques no necesariamente contiguos.

El tamaño del bloque si es muy **grande se desperdicia espacio** (los archivos ocupan bloques completos). Si el bloque es muy pequeño, un archivo necesita almacenarse en muchos bloques lo que produce que **el acceso sea muy lento**.

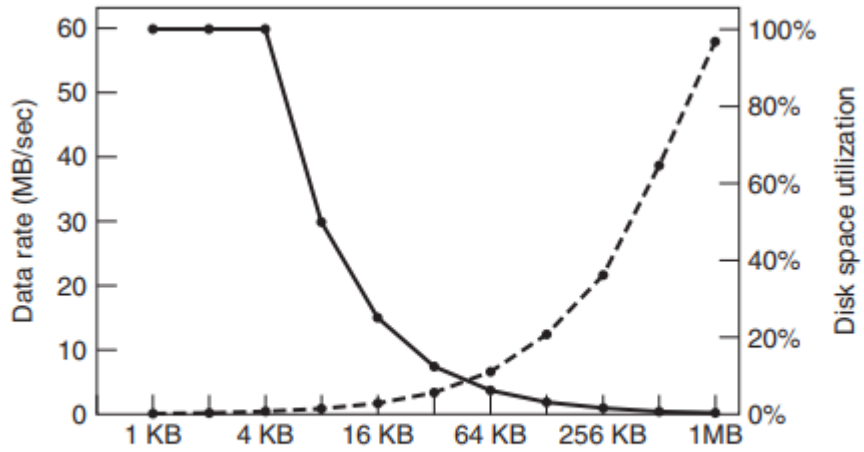


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk-space efficiency. All files are 4 KB.

Registro de Bloques Libres

Una vez que se ha elegido un tamaño de bloque, la siguiente cuestión es cómo llevar registro de los bloques libres. Hay dos métodos:

1. Utilizar una **Lista Enlazada de Bloques de Disco**: con un bloque de 1KB y números de bloque de 32 bits puede contener 256 números en total (255 bloques libres ya que se requiere una ranura de puntero).

2. Utilizar un **Mapa de Bits**: un disco con *n* bloques requiere un mapa con *n* bits. Los bloques libres se representan mediante '1' en el mapa.

```
Disco de 500GB ---\
                  |--> 488 millones de bloques
Bloques de 1KB ---/

    Lista Enlazada (255 numeros de bloque) --> 488.000.000 bloques/255 =
1.900.000 bloques

    Mapa de Bits --> 488.000.000*1bit cada uno = 488.000.000 -->
488Mbits/1KB = 61.000 bloques
```

No es sorprendente que el mapa de bits requiera menos espacio, ya que utiliza 1 bit por bloque, en comparación con 32 bits en el modelo de la lista enlazada.

Sólo si el disco está casi lleno (pocos bloques libres) es cuando el esquema de la lista enlazada requiere menos bloques que el mapa de bits. (Recordemos que estamos teniendo un registro de bloques libres y la lista enlazada si tiene, por ejemplo, un sólo bloque libre, sólo almacenaría 1 mientras que el mapa de bits tendría un registro con absolutamente todos los bloques).

Otras cosas a tener en cuenta...

Si los bloques libres se presentan en series de bloques consecutivos la lista enlazada se simplifica (se lleva cuenta de series en lugar de bloques individuales [número del primer bloque + número de bloques consecutivos])

Cuotas de Disco

Evita que los usuarios ocupen mucho espacio de disco, ya que cada usuario tiene asignada un máximo de archivos y bloques.

Mecanismo de gestión de cuotas:

- Tabla de **Archivos abiertos** en memoria principal:
  - Atributos, direcciones de disco...
  - Puntero a Tabla de Registro de Cuotas.
- Tabla de **Registro de Cuotas** de cada usuario con un archivo abierto:
  - En general información de bloques, archivos, etc.

Tipos de BackUps

El respaldo requiere tiempo y ocupa mucho espacio. Por lo que se plantea la siguiente pregunta: *¿Respaldar el sistema completo o sólo parte de él?*.

Por lo general, se realizan respaldos en cinta para manejar uno de dos problemas potenciales:

1. Recuperarse de un desastre.
2. Recuperarse de una estupidez porque eres subnormal (y lo sabes).

Respondiendo a la pregunta anteriormente planteada, tenemos que tener en cuenta que muchos programas ejecutables, archivos temporales, archivos especiales (E/S)... es innecesario respaldarlos.

En resumen, por lo general es conveniente respaldar sólo directorios específicos y todo lo que contengan, en vez de respaldar todo el sistema de archivos.

Por otro lado, es un desperdicio respaldar archivos que no han cambiado, lo cual nos lleva a pensar en **respaldos incrementales**. La forma más simple es realizar un respaldo completo de forma periódica y realizar un respaldo diario de sólo aquellos archivos que se hayan modificado desde el último respaldo completo.

Finalmente, no merece la pena realizar respaldos de sistemas que estén en continuo cambio (**activo**) ya que puede producir fallos si se respalda el disco mientras se está escribiendo y leyendo en muchos bloques.

- **Respaldo Físico**: del bloque 0 al último bloque. Simple y libre de errores. Respaldo total.
- **Respaldo Lógico**: el más usado. Comienta en uno o varios directorios y se respalda de forma recursiva. Únicamente se respaldan argunos de los archivos del sistema de archivos.

Más Datos sobre el Respaldo Lógico

El algoritmo de respaldo mantiene un mapa de bits indexado por número de I-Nodos con varios bits por I-Nodo. Los bits activarán y borrarán el mapa a medida que el algoritmo realice su trabajo.

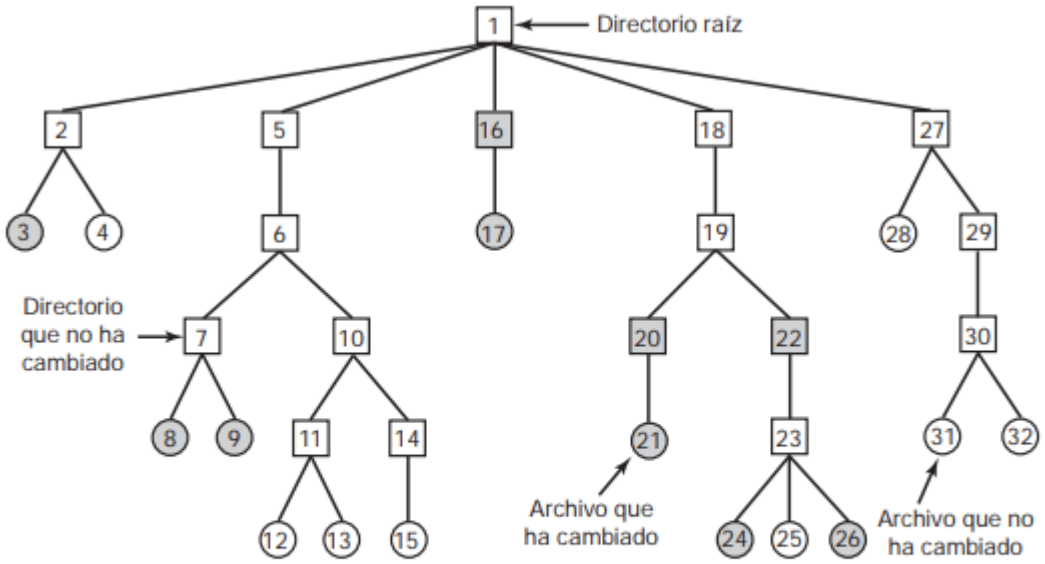


Figura 4-25. Un sistema de archivos que se va a vaciar. Los cuadros son directorios y los círculos son archivos. Los elementos sombreados han sido modificados desde el último vaciado. Cada directorio y archivo está etiquetado con base en su número de nodo-i.

1. Directorio inicial: examina todas las entradas que contiene. Para cada archivo modificado se marca su I-Nodo en el mapa de bits. Cada directorio también se marca independientemente se haya modificado o no.
2. Desmarcar directorios sin archivos modificados.
3. Respaldar Directorios.
4. Respaldar Archivos.

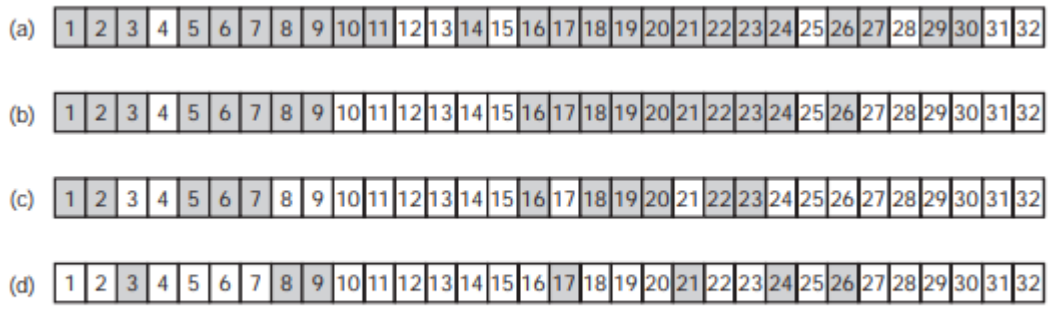


Figura 4-26. Mapas de bits utilizados por el algoritmo de vaciado lógico.

Consistencia en Sistema de Archivos

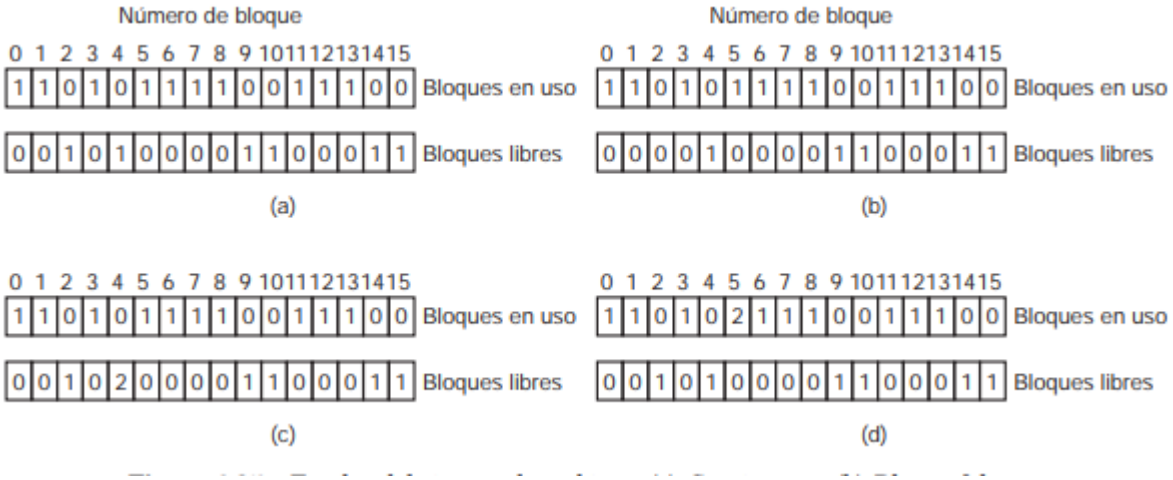
Las modificaciones del Sistema de Archivos no se escriben en disco inmediatamente.

Si el sistema falla antes de escribir todos los bloques se produce lo que se llama **inconsistencia**. Para la verificación de esta se usa *fsck* en UNIX y *scandisk* en Windows, que comprueban cada Sistema de Archivos: comprueban archivos y bloques.

Verificación de Bloques

Existen dos tablas con un contador para cada bloque:

- Tabla 1: Muestra **cuántas veces está presente un bloque en un archivo**.
- Tabla 2: Muestra **cuántas veces está un bloque en la lista de bloques libres**.



La figura superior muestra varios ejemplos.

1. Consistente: un 1 en la 1º tabla o en la 2º.
2. Bloque faltante (bloque 2): se agrega a bloques libres.
3. Duplicados en lista de bloques libres (bloque 4): reconstruir la tabla de bloques libres.
4. Bloque en dos o más archivos (bloque 5): dejar el bloque en uno de los archivos, copiarlo a otro bloque y asignarlo a otro archivo.

Verificación de Sistema de DIrectorios

Utiliza una Tabla de Contadores por archivo. Va descendiendo recursivamente por el árbol de directorios e incrementa el contador un valor por cada I-Nodo que vea en un directorio.

Tenemos ahora mismo un contador con el número total de I-Nodos del sistema. Ahora hay que compararlo con la lista indexada por el número de I-Nodos que indica cuantos directorios contienen cada archivo.

Mejora del Rendimiento

Cache de Bloques

En este contexto, una caché es una colección de bloques que pertenecen lógicamente al disco, pero se mantienen en memoria por cuestiones de rendimiento.

Un algoriitmo común para administrar la caché es verificar todas las peticiones de lectura para ver si el bloque necesario está en la caché. Aunque, como hay muchos bloques en caché y realmente necesitamos un acceso rápido y saber si está un bloque o no en caché para que sea útil se emplea otra técnica.

La forma usual es codificar en hash la dirección de dispositivo y de disco, buscando el resultado en una tabla hash. Todos los bloques con el mismo valor hash se encadenan en una lista enlazada, de manera que se pueda seguir la cadena de colisiones.