TEMA 4: SISTEMAS DE FICHEROS

INTRODUCCIÓN

Los sistemas de computación actuales necesitan algún tipo de almacenamiento secundario (discos) para guardar información que complemente al almacenamiento primario (RAM). Se usa para almacenar gran cantidad de información, preservar la información para que esta no desaparezca y permitir que cierta información sea fácilmente compartida y accedida.

El sistema operativo tiene que crear abstracciones que faciliten a los usuarios y a los programadores el uso de este tipo de almacenamiento. La solución es almacenar la información en unidades llamadas ficheros, los cuales se organizan y agrupan en directorios. La estructura de datos a través de la cual los ficheros y directorios se almacenan en el almacenamiento secundario se llama sistema de ficheros.

FICHEROS

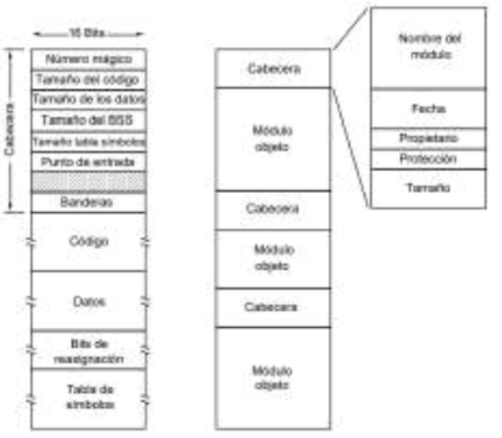
Un **fichero** es la menor unidad lógica de almacenamiento. En Windows, Linux, Mac OS X, etc… , un fichero es una secuencia de bytes cuyo significado está definido por el programa o programas que acceden al mismo. Con esta estructura de fichero se consigue máxima flexibilidad, ya que los programas pueden colocar y organizar información como deseen según una determinada estructura de datos. Cada fichero tiene un nombre que lo identifica. Dependiendo del diseño realizado en el sistema operativo y en el propio sistema de ficheros.

TIPOS DE FICHEROS

Ficheros regulares: contienen información del usuario. Los ***ficheros de texto*** constan de líneas de texto que se pueden ver e imprimir tal cual son y pueden modificarse con un editor de texto. Los ***ficheros binarios*** suelen aparecer como una lista incomprensible de símbolos al ojo humano.

Directorios: son ficheros gestionados por el propio sistema operativo para poder organizar y registrar los ficheros existentes en el sistema de ficheros.

Ficheros especiales de caracteres: están relacionados con la E/S y se utilizan para referenciar y acceder a dispositivos serie de E/S.



Ejemplo Fichero Binario

Ficheros especiales de bloques: son iguales que los anteriores, pero para referenciar y acceder a discos y otros dispositivos de almacenamiento secundario.

Muchas veces el nombre de un fichero contiene una extensión que indica de qué tipo de fichero se trata (texto o binario); hay programas, como un explorador de ficheros, para los que la extensión puede ser importante de cara a determinar qué acción realizar con un fichero. Las extensiones también ayudan a los usuarios a saber, con un simple vistazo, qué tipo de información contiene cada fichero.

ACCESO A UN FICHERO

En el acceso secuencial todos los bytes de un fichero deben leerse en orden, uno detrás de otro, sin posibilidad de saltarse algunos o leerlos en otro orden. En los ficheros de acceso aleatorio, es posible leer los bytes en un orden cualquiera, existiendo llamadas al sistema que permiten el posicionamiento (como lseek).

ATRIBUTOS DE UN FICHERO

Un fichero tiene un nombre y contiene ciertos datos. Además, los sistemas operativos añaden fecha y hora de creación, tamaño actual, tamaño máximo; y a estos elementos adicionales se les llama atributos. Los distintos atributos de un fichero son:

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 de solo lectura	0 Lectura/escritura, 1 para lectura exclusivamente
Bandera de ocultación	0 normal, 1 para no exhibirse en listas
Bandera del sistema	0 fichero normal, 1 fichero de sistema
Bandera de biblioteca	0 ya se ha respaldado, 1 necesita respaldo
Bandera texto/binario	0 fichero de texto, 1 fichero binario
Bandera de acceso aleatorio	0 solo acceso secuencial, 1 acceso aleatorio
Bandera temporal	0 normal, 1 eliminar al salir del proceso
Banderas de cerradura	0 no bloqueado, ≠ 0 bloqueado
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 SOBRE FICHEROS

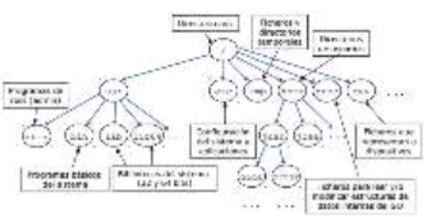
- ***Create***: crear un fichero.
- ***Delete***: borrar un fichero, liberando espacio en disco.
- ***Open***: abre un fichero para su uso.
- ***Close***: cierra un fichero abierto
- ***Read***: lee de un fichero la cantidad de bytes indicada como parámetros.
- ***Write***: escribe en un fichero la información indicada.
- ***Append***: es una forma restringida de write. Solo se puede añadir datos al final del fichero.
- ***Seek***: para los ficheros de acceso aleatorio, nos permite indicar una posición a partir de la cual leer o escribir.
- ***Get attributes***: obtiene los atributos asociados a un fichero.
- ***Set attributes***: nos permite cambiar un atributo con el valor indicado.
- ***Rename***: nos permite cambiar el nombre.
- ***Truncate***: elimina el contenido de un fichero a partir de una posición dada

DIRECTORIOS

Para organizar y llevar un registro de los ficheros, los sistemas operativos utilizan, directorios. Los directorios son ficheros que contienen información sobre otros ficheros; principalmente, contienen el nombre de esos otros ficheros, aunque también pueden incluir información sobre sus atributos. Estos ficheros son gestionados por el propio sistema operativo.

SISTEMAS JERÁRQUICOS DE DIRECTORIOS

Los sistemas operativos actuales utilizan un árbol de directorios para organizar los ficheros. En esta estructura, cada directorio, además de ficheros, puede contener, otros directorios.



NOMBRE DE LA RUTA DE ACCESO

Cada fichero tiene una ***ruta de acceso absoluta***, la cual consta de la ruta de acceso desde el directorio raíz hasta el fichero. Siempre comienzan por " / ".

Cada fichero tiene una ***ruta de acceso relativa*** que se utiliza junto con el concepto de directorio de trabajo o directorio actual. Se construye indicando los directorios que hay que recorrer para llegar desde el directorio actual hasta el fichero o directorio deseado. Además, podemos representar el director actual por ". " y al directorio padre por " .. "

OPERACIONES CON DIRECTORIOS

Se hacen a través de llamadas al sistema para transparencia y portabilidad.

- ***Create***: crea un directorio vacío. Cuando se refiere a vacío en realidad tiene " . " y " .. ".
- ***Delete***: elimina un directorio vacío.
- ***Opendir***: abre un directorio para ser recorrido.
- ***Closedir***: cierra un directorio abierto.
- ***Readdir***: devuelve la siguiente entrada de un directorio abierto.
- ***Rename***: cambia de nombre.
- ***Link***: permite que un mismo fichero aparezca a la vez con nombres diferentes en un mismo directorio, o en varios directorios con el mismo nombre o nombre distintos.
- ***Unlink***: elimina una entrada del directorio.

write:

Si los ficheros tienen atributos, los directorios también los tendrán, aunque, para algunos atributos, el significado podría cambiar ligeramente.

IMPLEMENTACIÓN DEL SISTEMA DE FICHEROS

IMP

Lecturas y escrituras tamaño mín bloque no menos que eso.

En conclusión, el sistema operativo ve los sistemas de ficheros como arrays lineales de bloques y tiene N bloques, desde 0 hasta $N-1$, todos del mismo tamaño. Tanto el tamaño como el número total de bloques dependen del dispositivo y su capacidad. Cada bloque tiene también una posición dentro del array que lo llamaremos dirección de disco del bloque.

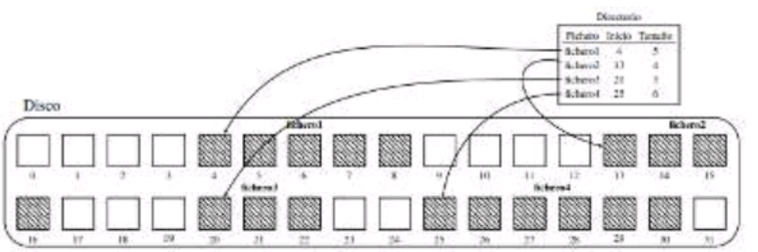
IMPLEMENTACIÓN DE FICHEROS

Todo fichero tiene asociado un conjunto de bloques de disco donde guarda sus datos. Cuando hablamos de implementación de ficheros estamos hablando de una forma de llevar un registro de esos bloques. Puesto que hay varias formas de llevar ese registro, existen varias implementaciones.

ASIGNACIÓN ADYACENTE O CONTIGUA

Es el esquema más sencillo. En esta implementación, cada fichero se almacena como un conjunto de bloques adyacentes en disco. Esta implementación posee dos ventajas. Una es que es de fácil implementación, ya que el registro de la localización se reduce a recordar un solo número. Y la otra ventaja es que esta implementación ofrece un rendimiento excelente, ya que, al estar todos los bloques de un fichero juntos en disco, el recorrido de ese fichero supondrá muy pocos movimientos. Además, es posible leer o escribir varios bloques consecutivos del fichero en una única operación de disco

Sin embargo, presenta algunos inconvenientes. El primero es que no es realizable de manera eficiente en un sistema de ficheros de propósito general si no se sabe qué tamaño máximo va a tener un fichero, no se sabe qué espacio reservarle. Si se reserva un espacio demasiado pequeño puede ocurrir que, si un fichero quiere crecer, no pueda. Otro problema de esta implementación es el de la fragmentación externa, puede ocurrir que el espacio libre esté repartido en huecos pequeños, de tal manera que no se pueda crear un nuevo fichero.

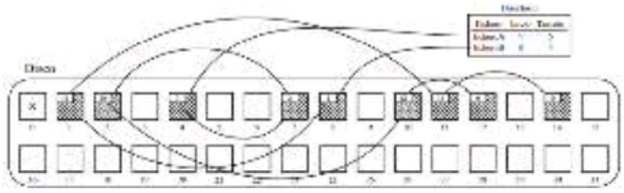


Esta implementación de ficheros es útil en la creación de DVDs o CD ROMs.

ASIGNACIÓN MEDIANTE LISTA LIGADA

En esta implementación cada fichero se mantiene como una lista ligada de bloques en disco. Al principio cada bloque se guarda la dirección de disco del siguiente bloque (un puntero).

Las ventajas de esta implementación son dos. No hay fragmentación externa, ya que se pueden utilizar todos los bloques de disco. Por otro lado, es suficiente que la entrada de directorio correspondiente al fichero guarde solo la dirección en disco del primer bloque. A partir de él se puede acceder a todos los demás.



Sin embargo, aunque la lectura secuencial es directa, el acceso aleatorio a un fichero es lento, ya que, tendríamos que leer todos sus bloques. Y el tamaño del espacio para almacenamiento de datos en un bloque ya no es potencia de dos, puesto que el apuntador ocupa algunos bytes.

IMP

Ya no es potencia de $2 \cdot (2^n)$!

ASIGNACIÓN MEDIANTE LISTA LIGADA E ÍNDICE

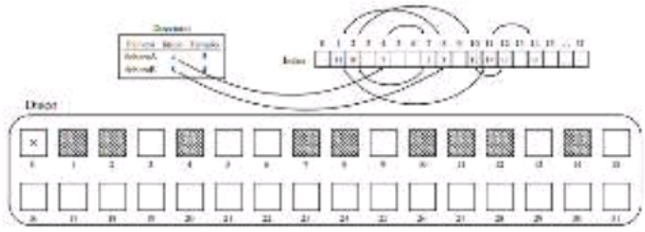
Una forma de eliminar los inconvenientes de la asignación mediante listas ligadas es almacenar los punteros en una tabla o índice en memoria, en esa tabla o índice hay una entrada por cada bloque de disco. La dirección de disco del bloque X de un fichero es D_{X+1} de su bloque $X+1$, se guarda en la entrada D_X de la tabla. La entrada corresponde al último bloque del fichero, almacena un 0, que no se considera un número de bloque válido, por lo que se utiliza para indicar el final de la lista. Con esta técnica ahora todo el bloque está disponible para datos y el acceso aleatorio es mucho más rápido, puesto que, aunque hay que seguir la cadena de punteros, este recorrido se hace en memoria.

Punteros en una tabla aparte.

La principal desventaja de esta técnica es que toda la tabla debe estar en memoria, lo que puede ser un problema si la tabla es grande. Otro aspecto a tener en cuenta es que la tabla se tiene que guardar en disco para que esté disponible cada vez que se arranca el sistema.

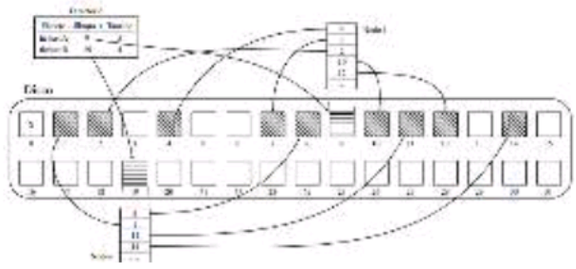
Si hacemos que ciertos números de bloques sean inválidos, entonces podríamos usar la propia tabla y esos valores para indicar qué bloques están libres.

El sistema de ficheros FAT utiliza este método (el sistema de ficheros que usaba MS-DOS y que usa Windows).



NODOS-I

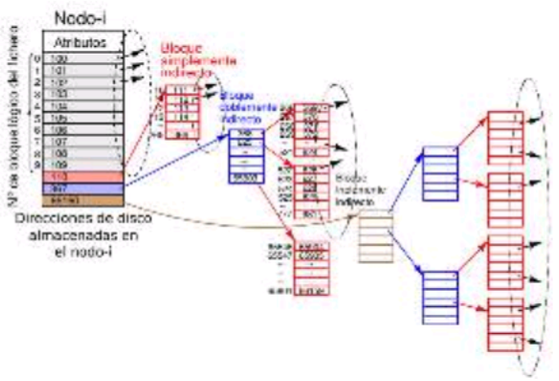
A cada fichero se le asocia una pequeña tabla llamada nodo-i (nodo índice), la cual contiene las direcciones en disco de los bloques del fichero en orden. En la imagen el nido-i de cada fichero se almacena en un bloque de disco; aunque, esta opción es posible no se suele utilizar, lo habitual es utilizar nodos-i pequeños, capaces de almacenar unas pocas direcciones de disco, y guardar varios nodos-i juntos en un mismo bloque de disco.



El problema surge cuando se usa todo un bloque como nodo-i, ¿qué pasa con los ficheros que grandes que tienen más bloques que direcciones caben? La solución es utilizar bloques indirectos, utilizar el nodo-i para almacenar las direcciones de disco de los primeros bloques del fichero correspondiente.

Para ficheros pequeños, las direcciones de todos sus bloques se almacenarán en el nodo-i. Para ficheros un poco más grandes, una de las direcciones en el nodo-i será la dirección de un bloque en el disco, llamado **bloque simplemente indirecto (BSI)**, que contendrá las direcciones en disco de bloques de datos adicionales del fichero. Si lo anterior no es suficiente, otra dirección en el nodo-i será la dirección de un **bloque doblemente indirecto (BDI)**, cuyo contenido serán las direcciones de más bloques simplemente indirectos. Si aún no es suficiente, se puede utilizar un **bloque triplemente indirecto (BTI)**, que almacena bloques doblemente indirectos.

Las ventajas de este método es que se puede calcular que un fichero puede tener hasta 16 843 018 bloques de datos de 1 KiB cada uno, lo que representa un tamaño do poco mas de 16 GiB. Aun así, acceder a cualquier posición del fichero supone, como mucho leer 4 bloques de disco (en el peor de los casos el triplemente indirecto, después el doblemente indirecto, uno simplemente indirecto y por ultimo el bloque de datos). Con el método de la lista ligada con índice necesitaríamos, solo para el fichero, un índice con un tamaño superior a 64 MiB para poder guardar todas las direcciones de bloques de datos. Este índice tendría que estar completamente en memoria.



En los sistemas de ficheros que usan nodos-i de este tipo, todos los nodos-i se guardan juntos en bloques consecutivos de disco a los que se llama tabla de nodos-i. Cada nodo-i recibe un número que es el que se guarda en la entrada de directorio correspondiente a su fichero. Sabiendo el número de nodo-i al que queremos acceder y cuántos nodos-i caben en un bloque, podemos saber qué bloque de la tabla de nodos-i tenemos que leer, y la posición de nuestro nodo-i dentro de ese bloque.

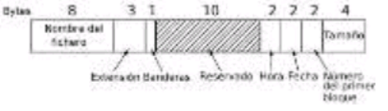
Un nodo-i se lee cuando se abre su fichero correspondiente y solo tiene que estar en memoria mientras el fichero esté abierto. Por lo tanto, la memoria necesaria para almacenar nodos-i es independiente del tamaño del sistema de ficheros y solo depende del número de ficheros abiertos en ese momento.

IMPLEMENTACIÓN DE DIRECTORIOS

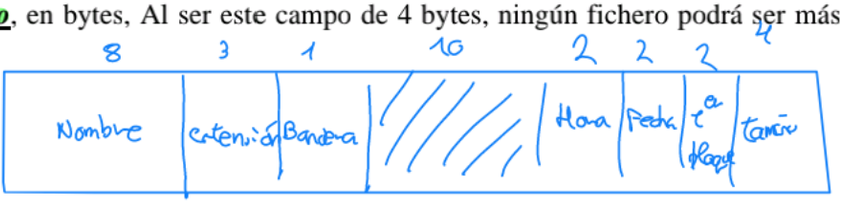
La principal función de un directorio es la de asociar el nombre de un fichero con la información necesaria para localizar los datos de dicho fichero (atributos, direcciones de bloques). Toda esta información se almacena en una tabla, dentro de la memoria principal.

DIRECTORIOS MS-DOS

En **MS-DOS**, los directorios son ficheros que almacenan una lista desordenada de entradas de 32 bytes, una por fichero. Cada una de estas entradas se divide en varios campos:



- ***Nombre y extensión del fichero***: los primeros 11 bytes.
- ***Atributos***: un byte donde varios bits se utilizan como banderas.
- Un campo de 10 bytes reservado para ***usos futuros***.
- ***Hora de la última modificación del fichero***. Observa que este campo tiene 16 bits; 5 bits para la hora, 6 bits para los minutos y 5 bits para guardar los segundos pares.
- ***Fecha de la última modificación del fichero***. Este campo tiene un tamaño de 16 bits; 7 bit para almacenar el año (el valor 0 al año 1980), 4 bits que almacena el mes y los últimos 5 bits almacenan el día.
- ***Número del primer bloque***, es decir, dirección de disco donde comienza.
- ***Tamaño del fichero***, en bytes, Al ser este campo de 4 bytes, ningún fichero podrá ser más grande de 4 GiB.



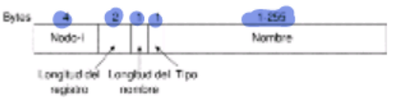
El directorio raíz es una excepción, ya que ocupa unos bloques fijos en disco en lugar de implementarse como un fichero, lo que hace que tenga un tamaño máximo preestablecido.

MS-DOS permite crear un árbol de directorios de tamaño arbitrario (un directorio puede tener subdirectorios). Y un bit del campo de atributos permite distinguir a los ficheros normales de los ficheros que son directorios. Esta estructura es utilizada, por los sistemas de ficheros FAT32 y vFAT.

DIRECTORIOS EN UNIX

En Unix, los directorios son ficheros gestionados por el propio sistema de ficheros. También es posible crear un árbol de directorios, ya que uno de los bits de los atributos almacenados en los nodos-i permite distinguir a un fichero normal de un directorio. En Unix, el directorio raíz no tiene ningún tratamiento especial y es un fichero como cualquier otro.

En las primeras versiones de UNIX, cada entrada tenía un tamaño fijo de 16 bytes, dos de los cuales se utilizaban para guardar el número de nodo-i y los otros 14 bytes almacenaban el nombre. Posteriormente, se han creado sistemas de ficheros para Unix que soportan nombres de hasta 255 caracteres, las entradas ya no tienen un tamaño fijo y se estructuran en varios campos:



- Un ***número de nodo-i***, que ocupa los primeros 4 bytes.
- La ***longitud del registro***, que indica el tamaño de toda la entrada en bytes. Es habitual que esta longitud sea múltiplo de 4.
- La ***longitud del nombre*** un único byte.
- El ***tipo de la entrada*** (directorio, fichero regular o cualquier otra cosa). Este campo es una copia del que hay almacenado en el nodo-i y se utiliza para acelerar.
- El ***nombre del fichero***.

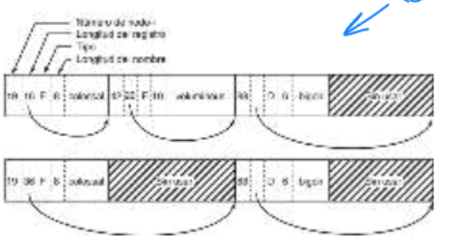
Una entrada fundamentalmente contiene el nombre de un fichero y su número de nodo-i. Toda la información relativa al tipo, tamaño, tiempos, propiedad y bloques en disco del fichero está contenida en su nodo-i.

Necesitamos el campo con la longitud del registro, porque, nos permite tener registros más grandes. Hay varias razones para querer poder hacer esto.

Si renombramos una entrada y usamos un nombre más corto, los bytes que sobren en el campo del nombre seguirán formando parte del registro y no se generará un hueco que no se podrá aprovechar para otro registro por ser excesivamente pequeño. Además, si el tamaño del registro es múltiplo de 4 será frecuente que en el campo del nombre siempre haya algún byte libre. Esto nos permitiría renombrar una entrada a un nombre con uno, dos y hasta tres caracteres más. Es decir, suele ser posible renombrar entradas sin tener que cambiar el tamaño del registro y, por tanto, sin tener que borrar, crear y copiar registros.

Otra razón es la gestión del espacio libre cuando se crea un directorio, el sistema de ficheros buscará un registro con suficiente espacio libre para crear un nuevo registro, dividiendo el registro existente en dos: uno para la entrada que ya existe y otro para la nueva entrada que, además, se quedará con el resto del espacio disponible. Si no hay ningún registro con espacio libre suficiente, entonces se añade un nuevo bloque de datos al directorio.

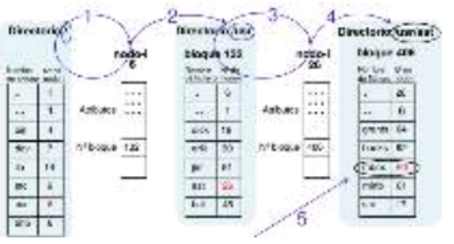
Un aspecto a tener en cuenta es que un registro solo puede estar en un bloque, es decir, un registro no puede atravesar la frontera entre bloques. El campo con la longitud del registro también facilita el borrado de entradas. Basta con añadir el espacio que queda libre al registro que hay justo antes. Si no hay ningún registro previo, entonces el registro se marca como libre utilizando un número de nodo-i inválido (habitualmente, 0).



RESOLUCIÓN DE RUTAS

Ejemplo: /usr/ast/mbox

1. Buscamos en el bloque de datos del directorio raíz la entrada correspondiente a /usr. Una vez localizada la entrada, de ella obtenemos su número de nodo-i.
2. Del nodo-i obtenemos la dirección del bloque de datos del directorio.
3. Ahora, en el bloque de datos de /usr, buscamos la entrada ast, esta entrada nos dice su nodo-i y leemos el nodo-i de disco.
4. Del nodo-i obtenemos la dirección del bloque de datos del directorio /usr/ast.
5. Finalmente, en el bloque de datos de /usr/ast, buscamos la entrada mbox, la cual nos dice cuál es el nodo-i del fichero.



Puesto que las resoluciones de rutas son algo muy frecuente, para acelerar su procesamiento el sistema operativo suele mantener en memoria información de las últimas resoluciones realizadas (*dentry cache* o *caché de entradas de directorio*).

Los nombres relativos se buscan de la misma manera, pero comenzando en el directorio de trabajo. Los directorios «.» y «..», se buscan como cualquier otro nombre, ya que existen entradas reales.

FICHEROS COMPARTIDOS

Cuando un mismo fichero puede aparecer en varios directorios, con el mismo o distinto nombre, o en un mismo directorio con nombres distintos, se dice que es un fichero compartido. La conexión entre un directorio y un fichero compartido se llama enlace.

Los ficheros compartidos se pueden implementar básicamente de dos formas. ***Enlaces físicos*** (*hard link*) los datos relativos a un fichero/atributos se guarden en una estructura de datos y las entradas de los directorios contengan apuntadores a esa estructura. En Unix, varias entradas de un mismo directorio o de directorios distintos guarden un mismo número de nodo-i. Es necesario que exista un contador de enlaces en el nodo-i, si se elimina un enlace borrando un fichero y hay más enlaces, el nodo-i no se debe liberar.

Enlaces simbólicos (*soft links*) es un nuevo tipo de fichero cuyo contenido sea la ruta de acceso del fichero al que se enlaza. Estos ficheros especiales se distinguen del resto de ficheros mediante un bit de bandera. Cuando el fichero original se borra, el uso posterior del fichero a través de un enlace simbólico simplemente fallará, ya que el enlace apuntará a un fichero con el mismo nombre, el enlace simbólico volverá a estar operativo.

El problema es su elevado coste, tanto temporal, como de espacio (hace falta un nodo-i para cada enlace). Sin embargo, son más flexibles que los enlaces físicos. Un problema de todos los enlaces es el riesgo de duplicar datos.

ADMINISTRACIÓN DEL ESPACIO EN DISCO

TAMAÑO DE BLOQUE LÓGICO

Los discos son dispositivos de bloques en los que la unidad mínima de lectura y escritura es el bloque. Estos bloques físicos (sectores), suelen tener tamaños de 512, 1024, 2048 y 4096 bytes. Para facilitar el uso de los discos, los propios dispositivos o el sistema operativo, hacen que estos se vean como un array lineal de bloques físicos.

Es común que el sistema operativo agrupe los sectores para formar bloques lógicos. Si el bloque lógico es grande un fichero constara de pocos bloques, pero puede aparecer una fragmentación interna, lo que puede dar lugar a un gran desperdicio de espacio. Si el bloque lógico es pequeño, se producirá poca fragmentación interna, pero cada fichero ocupará muchos bloques.

El tamaño de bloque lógico también afecta al rendimiento, pero duplicar el tamaño no duplica el tiempo de E/S solo lo aumenta un poco. Por lo tanto, a la hora de elegir un tamaño de bloque lógico, hay que buscar un equilibrio razonable entre la eficiencia en el uso del espacio en disco y la tasa de transferencia en las operaciones de disco.

REGISTRO DE BLOQUES LIBRES

En la lista ligada de bloques, cada bloque de la lista contiene tantos números de bloques libres como pueda, además de un puntero al siguiente bloque. Los bloques de la lista son bloques libres podrán ser usados en el momento en el que dejen de contener información útil.

En el ***mapa de bits***, un disco con *N* bloques necesita un mapa de bits con *N* bits. Los bloques libres se representan con el valor 0 en el mapa y los bloques asignados con el valor 1. El mapa de bits debe guardarse también en bloque de disco, estos bloques nunca se convertirán en bloques libres, pues el mapa de bits siempre tiene el mismo tamaño al depender del tamaño del disco y no del número de bloques libres.

De las dos opciones, la más utilizada en la práctica es el mapa de bits. Linux o Windows usan esta técnica. Un motivo es que el mapa de bits suele ocupar bastante menos espacio. Solo si el disco está casi lleno, el mapa de bits ocupará más que la lista ligada. Otro motivo es que el mapa de bits permite buscar de forma sencilla grupos de bloques libres consecutivos en disco. Basta con buscar una secuencia de bits a 0 lo suficientemente larga. Esto es importante de cara al rendimiento, ya que es más eficiente leer o escribir un fichero que tiene todos sus bloques consecutivos. Para conseguir lo mismo con la lista ligada, los bloques tendrían que estar ordenados por dirección de disco, lo cual puede ser bastante costoso.

CACHE DE DISCO

El acceso a un disco SSD es bastante más lento que el acceso a memoria principal. Para ello, existe la cache de disco, que trata de reducir los accesos a disco necesarios. Esta caché es implementada por el sistema operativo utilizando una parte de la memoria principal y contiene bloques que, pertenecen al disco, pero que se mantienen temporalmente en la memoria principal por razones de rendimiento.

El funcionamiento es similar al de cualquier otra caché. Cuando se lee un bloque, se comprueba si está o no ya en memoria principal. En caso afirmativo, se satisface la solicitud sin acceder al disco; si no, se lee el bloque del disco, se coloca en caché y después se procesa la solicitud de lectura. En este segundo caso la caché puede estar totalmente ocupada, por lo que habrá que eliminar algún bloque (escribiéndolo en disco si ha sido modificado) para hacer hueco.

A la hora de decidir qué bloque expulsar de una caché llena, podemos usar un **algoritmo LRU** con listas ligadas, ya que las referencias a caché son mucho menos frecuentes, pero no es recomendable si queremos mantener la consistencia del sistema de ficheros ante los posibles fallos del sistema.

Si el bloque acaba de ser modificado, se encontrará en la cabeza de la lista LRU, por lo que podrá transcurrir mucho tiempo antes de que sea escrito en disco. Cuanto más tiempo pase sin que un bloque modificado se escriba en disco, mayor será la probabilidad de que se produzcan inconsistencias por caídas del sistema. Además, hay bloques, como los bloques doblemente indirectos, que rara vez tienen dos referencias en un intervalo corto de tiempo, por lo que puede interesar colocarlos en la cola de la lista para que sean los primeros en salir de caché. Las dos consideraciones anteriores dan lugar a un LRU modificado que tiene en cuenta que los bloques recién usados que probablemente no se vuelvan a utilizar pronto pasarán directamente al final de la lista LRU para que sus buffers se reutilicen con rapidez. Si un bloque es esencial para la consistencia del sistema de ficheros y ha sido modificado, debe escribirse en disco lo antes posible, sin importar en qué lugar de la lista LRU se encuentre, se escribe en disco sin esperar a que sea expulsado.

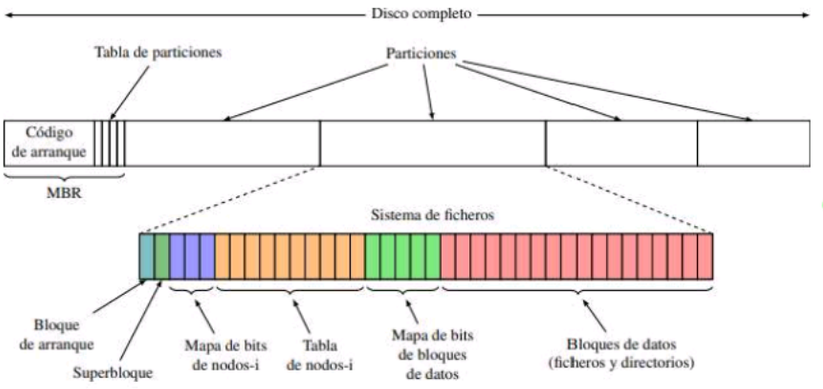
Tampoco es recomendable tener mucho tiempo los bloques de datos en caché, por el riesgo de perder información. En Unix, cualquier bloque de datos modificado se escribe en disco a los 30 segundos o antes. En cambio, los bloques de metadatos se escriben en disco a los 5 segundos o antes.

DISCOS Y SISTEMAS DE FICHEROS

Lo habitual es que un disco contenga varios sistemas de ficheros.

PARTICIONES

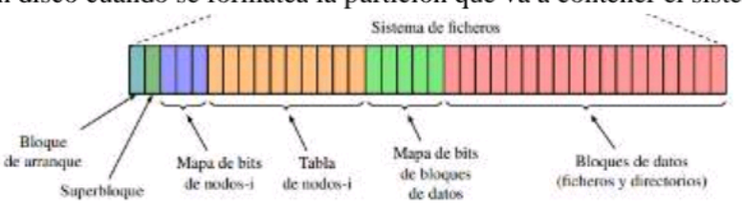
Para facilitar el uso de los discos, los sistemas operativos permiten crear particiones. Una partición es una porción de bloques consecutivos de un disco. Las particiones son manejadas por el sistema operativo que las representa también como un array lineal de bloques. Este array tendrá un bloque 0 y un bloque $N-1$, donde ahora N dependerá del tamaño de la partición en bloques lógicos.



El número y tamaño de las particiones existentes en un disco, junto con los bloques de inicio y fin de cada una, se ha guardado tradicionalmente en la tabla de particiones, la cual, se almacena en el bloque 0 del disco (**MBR**). El MBR también suele contener una pequeña porción de código de arranque que la BIOS carga en memoria y ejecuta para terminar arrancando un sistema operativo.

ESTRUCTURA DE UN SISTEMA DE FICHEROS

El sistema de ficheros usa zonas o grupos de bloques consecutivos de disco para almacenar distintas estructuras de datos. Estas estructuras de datos, su organización en disco y su contenido inicial se crean y escriben en disco cuando se formatea la partición que va a contener el sistema de ficheros.



El bloque de arranque ocupa el bloque 0 de la partición. Este bloque generalmente no se usa, pero puede contener código de arranque del sistema operativo que se encuentra en la partición. Ya que generalmente este bloque no es usado por el sistema de ficheros, se puede usar su dirección (0) como valor nulo o no válido.

El bloque 1 es el ***superbloque***. Contiene información crítica relativa a la organización del sistema de ficheros: número total de bloques lógicos, número total de nodos-i, tamaño de los mapas de bits de bloques y nodos-i, etc. La destrucción del superbloque provocaría que el sistema de ficheros quedara ilegible.

El ***mapa de bits de bloque***, cada bit indica si el bloque correspondiente está libre u ocupado. Su tamaño depende del tamaño de la zona dedicada a ficheros y directorios. Si esta zona tiene B bloques total, de tamaño T_B bytes cada uno, entonces el mapa de bits ocupa $\left\lceil \frac{B}{8 \cdot T_B} \right\rceil$ bloques.

El ***mapa de bits de nodos-i*** se utiliza para saber qué nodos-i hay libres y cuáles están ocupados. Su tamaño depende del total de nodos-i. Si tenemos I nodos-i y un tamaño de bloque de T_B bytes, entonces este mapa necesita $\left\lceil \frac{I}{8 \cdot T_B} \right\rceil$ bloques.

La ***tabla de nodos-i*** contiene los nodos-i usados para ficheros. El tamaño de la tabla de nodos-i depende del tamaño de nodo-i y del total de nodos-i en el sistema. Si tenemos I nodos-i de tamaño T_I bytes cada uno, y bloques de tamaño T_B bytes, esta tabla ocupa $\left\lceil \frac{I \cdot T_I}{T_B} \right\rceil$ bloques.

El ***resto de los bloques*** se usa para almacenar en ellos los bloques de datos de los ficheros regulares, los directorios y los bloques indirectos.