T E M A 3

Sistemas de Ficheros

Contenido

TEMA 3

SISTEMAS DE FICHEROS

3.1. Introducción

Todas las aplicaciones necesitan almacenar y recuperar información. Cuando un proceso está en ejecución puede almacenar una cantidad limitada de información dentro de su propio espacio de direcciones en memoria principal. Sin embargo, este espacio está limitado por el espacio de direcciones virtual, que puede ser insuficiente para algunas aplicaciones.

Un segundo problema que se presenta al almacenar la información en el espacio de direcciones de un proceso es que cuando éste termina la información se pierde. Esto es inaceptable para muchas aplicaciones, que pueden requerir que la información permanezca disponible durante largos periodos de tiempo.

Otra limitación es que varios procesos pueden necesitar acceder a una misma información de forma concurrente. Como los espacios de direcciones de los procesos son privados a ellos mismos, un proceso no puede acceder a los datos que se encuentran en el espacio de direcciones de otro. La forma de solucionar este problema es hacer que la información sea independiente de los procesos.

Por tanto, podemos establecer tres requisitos esenciales para almacenar la información durante un tiempo indefinido:

- Debe ser posible almacenar una gran cantidad de información.
- La información debe de persistir tras la terminación de los procesos que la usan.
- Varios procesos deben de ser capaces de acceder a la información de forma concurrente.

La solución a estos problemas consiste en almacenar la información en discos magnéticos u otros dispositivos en unas unidades llamadas **ficheros**, que pueden ser leídas y escritas por los procesos que así lo requieran. La información almacenada en ficheros debe de ser **persistente**, es decir, no debe verse afectada por la creación y finalización de los procesos. La gestión de ficheros es tarea del sistema operativo, y la parte del mismo que realiza dicha gestión se conoce como **sistema de ficheros**.

Desde el punto de vista de los usuarios, el aspecto más importante de un sistema de ficheros es cómo éste se presenta a ellos. Es decir, qué es un fichero, cómo se nombra, qué operaciones se permiten, etc. En definitiva, al usuario le interesa saber qué es lo que puede hacer. Desde el punto de vista de los diseñadores de sistemas, lo interesante es saber cómo está implementado el sistema de ficheros.

3.2. Ficheros

Un fichero es una abstracción de un mecanismo que permite almacenar información en un dispositivo y leerla posteriormente. Podemos definir un fichero como una colección de información que tiene un nombre.

3.2.1. Denominación de Ficheros

Los ficheros tienen asignados un nombre a través del cual los usuarios se refieren a ellos. Sin embargo, las reglas de denominación de ficheros difieren de sistema a sistema. Muchos sistemas operativos dividen el nombre de los ficheros en dos partes separadas por un punto. La primera parte sería el nombre, propiamente dicho. La segunda parte se suele denominar **extensión** y suele aportar información sobre el contenido del fichero. Por ejemplos, los ficheros cuya extensión es el carácter 'c' indican que contienen programas escritos en el lenguaje C.

Las reglas básicas de denominación de ficheros en los sistemas operativos MS-DOS, UNIX y Windows NT son las siguientes:

- **MS-DOS**: el nombre de un fichero se compone de un máximo de ocho caracteres, seguidos, opcionalmente, por un punto y una extensión de tres caracteres como máximo. Las mayúsculas y

- **UNIX**: el nombre de un fichero se compone de un máximo de 256 caracteres. Se distinguen las mayúsculas de las minúsculas. Un fichero puede tener más de una extensión (por ejemplo, *image.tar.Z*)
- **Windows NT**: el nombre de un fichero se compone de un máximo de 256 caracteres. Las mayúsculas y minúsculas no son distinguibles y los ficheros pueden tener más de una extensión.

En muchos casos, las extensiones son meras convenciones y no son vinculantes con el contenido de los ficheros. Por otro lado, muchas aplicaciones requieren que los ficheros que utilizan tengan unas extensiones concretas.

3.2.2. Estructura de un Fichero

Un fichero se puede estructura de varias formas. Una posibilidad es organizar un fichero como una secuencia de bytes. De esta forma, el sistema operativo no conoce el significado del contenido de los ficheros, lo que simplifica la gestión de los mismos. Serán los programas de aplicación los que deberán de conocer la estructura de los ficheros que utilizan. Este enfoque es el empleado por MS-DOS y UNIX.

Un esquema más estructurado es considerar un fichero como una secuencia de registros de longitud fija, cada uno de los cuales presenta una estructura determinada. La idea es que las operaciones de lectura devuelvan un registro y las escritura modifiquen o añadan un registro. El sistema operativo CP/M usa registros de 128 bytes.

La tercera forma es organizar fichero en forma de árbol de registros, que no tienen po
misma longitud. Cada registro tiene un campo clave por el que está ordenado el árbol de forma que las operaciones de búsqueda por clave se realizan rápidamente. Este esquema se emplea en grandes computadores (*mainframes*) orientados al proceso de grandes cantidades de información.

En cualquier caso, todos los sistemas operativos deben soportar al menos una estructura, que es la de los ficheros ejecutables.

3.2.3. Tipos de Ficheros

Muchos sistemas operativos soportan varios tipos de ficheros. En UNIX, por ejemplo, existen ficheros regulares, directorios, ficheros especiales de caracteres y ficheros especiales de bloques. Los ficheros regulares contienen información procedente de los usuarios. Los directorios son ficheros del sistema que mantienen la estructura del sistema de ficheros. Los ficheros especiales de caracteres están relacionados con las operaciones de entrada/salida de dispositivos basados en caracteres, como los terminales y las impresoras. Los ficheros especiales de bloques se usan básicamente para acceder a discos.

Los ficheros regulares u ordinarios son generalmente ficheros ASCII o ficheros binarios. Los ficheros ASCII tienen la ventaja de poder ser examinados directamente por pantalla o impresora y manipulados con editores de textos.

Los ficheros binarios son todos aquellos que no son ASCII. Normalmente presentan algún tipo de estructura interna. Todos los sistemas operativos deben de reconocer la estructura de los ficheros que son ejecutables, como se comentó anteriormente.

3.2.4. Métodos de Acceso

La información que almacenan los ficheros tiene que poder ser accedida de alguna forma. El método de **acceso secuencial**, en que la lectura de los bytes o registros de los ficheros se realiza en orden, empezando por el comienzo del fichero. Programas como editores de texto y compiladores acceden a los ficheros de esta forma. Existe un puntero que indica el siguiente trozo del fichero a leer. Cuando se realiza una lectura, se devuelve el byte o registro que indica el puntero del fichero y éste avanza a la siguiente posición. Cuando se realiza una escritura, se escribe en la posición que indica el puntero del fichero. Si éste apunta al final del mismo, la información que se escribe se añade al final del fichero. El acceso secuencial se basa en modelar los ficheros como si fuesen cintas magnéticas y se puede

	Contraseña requerida para acceder al fichero
Creador	Identificador del creador del fichero
Propietario	Identificador del propietario
Indicador de sólo-lectura	0 si lectura/escritura, 1 si sólo lectura
Indicador de fichero oculto	0 para normal, 1 para no mostrar en los listados
Indicador de fichero de sistema	0 para normal, 1 para fichero de sistema
Indicador ASCII/binario	0 para ASCII, 1 para binario
Indicador de acceso aleatorio	0 para acceso secuencial, 1 para acceso aleatorio
Indicador de fichero temporal	0 para normal, 1 para borrar cuando termine el proceso
Indicadores de bloqueo	0 para no bloqueado, 1 para bloqueado
Longitud de registro	Número de bytes de un registro
Posición de la clave	Desplazamiento de la clave dentro del registro
Longitud de la clave	Número de bytes del campo clave
Fecha de creación	Fecha y hora en la que se creó el fichero
Fecha de último acceso	Fecha y hora en la que realizó el último acceso al fichero
Fecha de última modificación	Fecha y hora en la que se realizó la última modificación al fichero
Tamaño en bytes	Número de bytes del fichero
Tamaño máximo	Tamaño máximo que puede tener el fichero

3.2.6. Operaciones sobre Ficheros

Los ficheros se pueden considerar como tipos abstractos de datos, por lo que para definir qué es un fichero hay que considerar las operaciones que pueden ser realizadas con ellos. Cada sistema operativo ofrece su propio conjunto de operaciones sobre ficheros. Las operaciones más comunes son:

1. **CREATE**: El fichero es creado sin datos.
2. **DELETE**: Cuando el fichero no es necesario se ha de borrar para liberar el espacio que ocupa en disco. Algunos sistemas borran de forma automática aquellos ficheros que no han sido usados en *n* dias.
3. **OPEN**: Antes de usar un fichero, un proceso ha de abrirlo para que el sistema busque y almacene en memoria aquellos atributos y direcciones de disco necesarias para acceder rápidamente al fichero.
4. **CLOSE**: Cuando se acaban los accesos a un fichero, el espacio que ocupa en memoria ha de ser liberado.
5. **READ**: El proceso que lee información de disco debe de especificar la cantidad de información a leer y el lugar donde colocarla (*buffer*).
6. **WRITE**: Los datos se escriben en un fichero a partir de la posicion que indica el puntero de desplazamiento.
1. **APPEND**: Es una forma restringida de WRITE que fuerza a que los nuevos datos se escriban al final del fichero.
2. **SEEK**: Posiciona el puntero de desplazamiento en cualquier posición de un fichero de acceso aleatorio.
3. **GET ATTRIBUTES**: Para algunos procesos es necesario obtener algunos atributos de los ficheros (ej.: utilidad `make` de Unix).
4. **SET ATTRIBUTES**: Algunos atributos pueden ser modificados por el usuario (ej.: cambiar la protección).
5. **RENAME**: Es frecuente que sea necesario cambiar el nombre de un fichero.

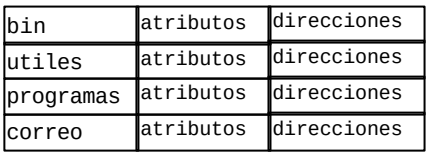


Figura 1.- Atributos y direcciones en la entrada de Directorio.

3.3. Directorios

Los sistemas de ficheros pueden contener grandes volúmenes de información, por lo que los sistemas operativos proporcionan mecanismos para estructurarlos. Esta estructuración se suele realizar a dos niveles.

En primer lugar, el sistema de ficheros se divide en particiones, que se pueden considerar como discos virtuales. Un disco puede contener una o varias particiones, y una partición puede estar repartida

En segundo lugar, cada partición contiene una tabla de contenidos de la información que contiene. Esta tabla se denomina directorio, y su función principal es hacer corresponder un nombre de un fichero con una entrada en la tabla. Cada entrada contiene, como mínimo, el nombre del fichero.

A continuación se pueden almacenar los atributos y direcciones del fichero en disco (Figura 1), o un puntero a otra estructura de datos que contiene dicha información (Figura 2). Esta estructura de datos se suele conocer con el nombre de **nodo índice**.

Cuando se abre un fichero el sistema operativo busca en el directorio correspondiente hasta que encuentra el nombre del fichero, toma la información que necesita y la introduce en una tabla residente en la memoria principal. Las siguientes referencias al fichero utilizarán la información que ya está en memoria.

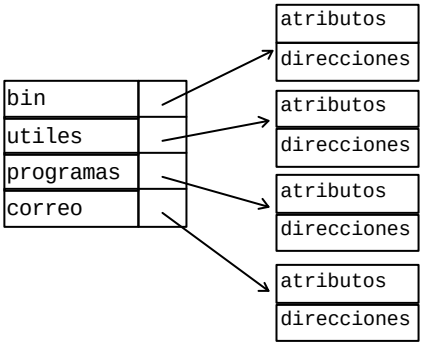


Figura 2.- Atributos y direcciones en una estructura de datos.

3.3.1. Sistemas con Directorios Jerárquicos

El número de directorios puede variar de un sistema a otro. El diseño más simple consiste en tener un , que contiene todos los ficheros. El problema es que si existen varios usuarios se pueden originar fácilmente conflictos si más de uno elige el mismo nombre para dos ficheros diferentes. Incluso si existe un único usuario, a medida que el número de ficheros aumenta, puede ser difícil recordar los nombres de todos los ficheros. Este esquema se ha usado en los sistemas operativos de los primeros computadores personales (por ejemplo, CP/M).

Una mejora del esquema anterior es tener un directorio por usuario, con lo que se tiene una **estructura en dos niveles**. De esta forma se elimina el problema de la duplicidad de nombres, pero no permite a los usuarios agrupar juntos aquellos ficheros que estén relacionados, problema que se acrecienta a medida que crece el número de ficheros. Con esta estructura cada usuario está aislado, lo que es una ventaja cuando los usuarios quieren ser independientes, pero es un inconveniente cuando quieren cooperar en algún trabajo y necesitan acceder a ficheros de otro usuario.

La solución a estos problemas consiste en que el sistema de ficheros tenga una estructura jerárquica o **estructura en árbol**, de forma que cada usuario pueda crear sus propios subdirectorios para organizar sus ficheros. El árbol de directorios tiene un origen o referencia denominado directorio raíz. Todo fichero tiene una ruta de acceso única, siendo ésta el camino desde la raíz hasta un fichero, especificándose los directorios (subdirectorios) que se atraviesan. Este nombre único, cuando se parte del directorio raíz, se llama camino absoluto o **ruta de acceso absoluta**. Normalmente, cada usuario trabaja en un directorio actual o **directorio de trabajo**. Si la referencia a un fichero se hace partiendo del directorio de trabajo se habla de **ruta de acceso relativa**.

3.3.2. Ficheros Compartidos

Cuando dos o más usuarios trabajan en un proyecto conjunto, es frecuente que necesiten compartir ficheros o directorios. Además, como todos los usuarios miembros del proyecto pueden tener los mismos privilegios y responsabilidades, cada uno de ellos quiere tener los ficheros y directorios compartidos en sus propios subdirectorios. Para satisfacer estos requisitos es necesario que los ficheros y directorios comunes sean compartidos, lo que significa que deben de aparecer en el sistema de ficheros en más de un lugar a la vez.

Mediante estructuras en árbol no se pueden compartir ficheros, ya que cuando se comparte un fichero, el árbol de directorios se transforma en un grafo acíclico dirigido. Por otro lado, si se comparten directorios el grafo puede ser cíclico, lo que puede originar algunos problemas.

Los ficheros y subdirectorios compartidos se pueden implementar de varias formas. En sistemas operativos en los que los directorios contienen las direcciones de disco de los ficheros, la única forma de compartir un fichero es realizar una copia del mismo. Esto tiene el inconveniente de cuando uno de los ficheros es modificado, el resto de las copias permanece inalterada. La solución más común consiste en crear una nueva entrada de directorio llamada **enlace** (*hard link*), que no es más que un puntero a un fichero o directorio. Este esquema es el usado por UNIX, y un enlace simplemente apunta al nodo índice del fichero o directorio a compartir. Otra solución consiste en la creación de ficheros especiales que contengan la ruta de acceso del fichero que se quiere compartir. Este tipo de enlace se denomina **enlace simbólico** (*symbolic link*).

El inconveniente del primer tipo de enlaces es que si un usuario A realiza un enlace sobre un fichero de un usuario B, el nodo índice sigue indicando que el propietario del fichero es B. Si B borra el fichero, el nodo índice permanece intacto, por que se llega a una situación en la que el usuario B es el propietario de un fichero que ha borrado. Este problema no se produce con enlaces simbólicos porque únicamente el verdadero propietario del fichero tiene un puntero al nodo índice. El resto tiene rutas de acceso al fichero compartido, no punteros al nodo índice. Cuando el propietario borra el fichero éste es destruido y posteriores referencias al mismo a través de enlaces simbólicos serán rechazadas porque le fichero no existe.

El problema de los enlaces simbólicos es que acceder al fichero que contiene requiere accesos a disco adicionales. Además, cada enlace simbólico gasta un nodo índice. Una ventaja de los enlaces simbólicos es que se pueden usar para compartir ficheros que se encuentran en lugares distintos a su propio sistema de ficheros.

Cuando se usan enlaces, del tipo que sea, los ficheros compartidos tienen dos o más rutas de acceso. Si un programa realiza una búsqueda a partir de un directorio, se puede encontrar con el mismo fichero varias veces. Si el enlace se realiza sobre un directorio, la estructura de directorios se puede convertir en un grafo cíclico. En estos casos, un programa de búsqueda puede entrar en bucle sin fin. Una forma de evitar esta situación es prohibir a los usuarios la creación de enlaces a directorios. Estos sólo pueden ser creados por el administrador del sistema.

3.3.3. Operaciones sobre Directorios

Las operaciones relacionadas con directorios varían considerablemente de unos sistemas a otros. En UNIX son las siguientes:

1. **CREATE**: Crea un directorio vacío, exceptuando los directorios '.' y '..', que son creados automáticamente.
2. **DELETE**: Borra un directorio si está vacío. Un directorio que tiene los directorios '.' y '..', se consiera vacío.
3. **OPENDIR**: Abre un directorio para lectura.
4. **CLOSEDIR**: Cierra un directorio.
5. **READDIR**: Devuelve la siguiente entrada en un directorio abierto previamente.
6. **RENAME**: Los directorios se puede renombar al igual que los ficheros.
7. **LINK**: Los enlaces permiten que un fichero aparezca en más de un directorio.
8. **UNLINK**: Elimina una entrada de directorio. Si el fichero sobre el que elimina el enlace únicamente tiene una entrada en un directorio el fichero se borra del sistema de ficheros.

3.4. Implementación de Sistemas de Ficheros

Mientras que desde el punto de vista de un usuario interesa conocer aspectos como la denominación de los ficheros, qué operaciones se suministran, etc., desde el punto de vista de un diseñador de sistemas interesa conocer la forma de almacenamiento de los ficheros y directorios, cómo se gestiona el espacio en disco, y cómo hacer que el sistema de ficheros sea eficiente y fiable.

3.4.1. Organización de un Sistema de Ficheros

Los discos magnéticos constituyen la base sobre la que se sustentan los sistemas de ficheros. Para mejorar la eficiencia, la transferencia de información entre memoria y los discos se realiza en unidades denominadas bloques. Cada bloque está formado por uno o varios sectores de disco. El tamaño de sector de un disco suele ser de 512 bytes.

El diseño de un sistema de ficheros plantea dos problemas diferentes:

- definir cómo el sistema de ficheros aparece al usuario, y
- diseñar los algoritmos y estructuras de datos necesarias para implementar este sistema de ficheros lógico en los dispositivos físicos de almacenamiento secundario.

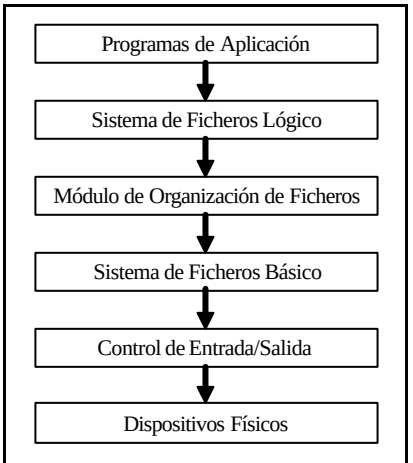


Figura 3.- Estructura en niveles de un sistema de ficheros.

Un sistema de ficheros se puede estructurar en diferentes capas o niveles (Figura 3). El nivel de control de entrada/salida se compone de los manejadores de dispositivo (*device drivers*) y los manejadores de interrupciones (*interrupt handlers*), que son necesarios para transmitir la información entre la memoria y los discos. Los manejadores de dispositivos reciben instrucciones de bajo nivel, del tipo "escribir o leer el bloque nº x", y generan el conjunto de instrucciones dependientes del hardware que son enviadas al controlador de disco.

El sistema de ficheros básico transmite las instrucciones de bajo nivel al manejador de dispositivo adecuado para leer y escribir bloques físicos en disco. Cada bloque físico se identifica por su dirección numérica en el disco, que viene dado por el dispositivo, cilindro, superficie y sector.

El módulo de organización de ficheros tiene conocimiento sobre los ficheros, los bloques lógicos que los componen, y los bloques físicos. Mediante el tipo de esquema de asignación de bloques y la localización del fichero, el módulo de organización de ficheros traslada las direcciones de disco lógicas en direcciones de disco físicas. Cada bloque de disco lógico tiene un número (de 0 a N), que no suele coincidir con la dirección de los bloques físicos, por lo que es necesario un mecanismo de traducción. Este módulo también incluye el gestor de espacio libre, que controla los bloques libres para que puedan ser usados posteriormente.

Por último, el sistema de ficheros lógico proporciona la estructura de directorio que es observada por los programas de usuario. También es responsable de proporcionar seguridad y protección al sistema de ficheros.

Para crear un nuevo fichero, un programa de aplicación realiza una llamada al sistema sobre el sistema de ficheros lógico. Este lee el directorio correspondiente en memoria, le añade una nueva entrada y lo escribe en disco. En este proceso, el sistema de ficheros lógico ha solicitado al módulo de organización de ficheros la dirección física del directorio, que se envía al sistema de ficheros básico y al control de entrada/salida. Cuando el directorio ha sido actualizado, el sistema de ficheros lógico lo puede usar para realizar operaciones de entrada/salida. Para optimizar los accesos, el sistema operativo tiene en memoria una tabla de ficheros abiertos que contiene información sobre todos los ficheros abiertos en un momento dado, como nombre, permisos, atributos, direcciones de disco, etc. La primera referencia a un fichero suele ser una operación de apertura, que hace que se realice un búsqueda en la estructura de directorio y se localice la entrada para el fichero que se quiere abrir. Esta entrada se copia en la tabla de ficheros abiertos y el índice de dicha entrada (denominado descriptor de fichero) se devuelve al programa de aplicación, que lo usa para realizar las operaciones sobre el fichero. De esta forma, todas las operaciones relacionadas con la entrada de directorio del fichero se realizan en la tabla de ficheros abiertos en memoria. Cuando el fichero se cierra, la entrada modificada se copia a disco.

Del mismo modo que los ficheros se abren antes de ser usados, los sistemas de ficheros deben de ser **montados** antes de que los procesos puedan usarlo. Para realizar esta operación, el sistema operativo debe de conocer el nombre del dispositivo y la localización dentro de la estructura de directorio en la cual se debe de "enganchar" el sistema de ficheros. Esta localización se denomina punto de montaje. A continuación, se verifica que el dispositivo contiene un sistema de ficheros válido. Por último, el sistema operativo anota en la estructura de directorio que el sistema de ficheros se ha montado en el punto especificado. Por ejemplo, el sistema operativo de los ordenadores Macintosh, cuando detecta un disco por primera vez, busca un sistema de ficheros en el mismo y lo monta en el escritorio (que equivale al directorio raíz). En un Macintosh, los discos duros se detectan cuando el sistema operativo arranca, mientras que los floppies se detectan cuando se insertan en la unidad de disquete.

3.4.2. Implementación de Ficheros

El aspecto básico de la implementación de ficheros consiste en determinar qué bloques de disco están asociados a cada fichero. El problema que se plantea es cómo asignar el espacio libre en disco a los ficheros de forma que ese espacio sea usado de forma eficiente y los ficheros puedan ser accedidos rápidamente. Hay que tener en cuenta que el tamaño de los ficheros es variable, por lo que habrá que diseñar algún mecanismo de almacenamiento dinámico tanto para los ficheros como para el espacio libre.

Existen tres métodos básicos de asignación: contiguo, enlazado e indexado.

3.4.2.1. Asignación contigua

El esquema de asignación más simple consiste en almacenar cada fichero como un secuencia adyacente de bloques en disco. La asignación contigua de un fichero se define por la dirección del primer bloque y el

Las ventajas de este método es su fácil implementación, ya que únicamente hay que conocer la dirección en disco del primer bloque del fichero, y su eficiencia, ya que la lectura de un fichero se puede realizar en una sola operación. El acceso a un fichero almacenado de forma contigua es sencillo y rápido.

Para acceso secuencial, el sistema de ficheros no tiene más que recordar la dirección del último bloque referenciado. El acceso directo del bloque `i` de un fichero que comienza en el bloque `b` se realiza, simplemente, accediendo al bloque `b+i`.

Un inconveniente de este esquema de asignación es que no se puede implementar a no ser que se conozca el tamaño de los ficheros en su momento de creación. Si esta información no está disponible, el sistema operativo no sabe cuánto espacio en disco debe reservar. Otra desventaja es la fragmentación externa que se produce en el disco, que se origina debido a que entre los ficheros quedan porciones de disco libre que no tienen el tamaño suficiente para ser asignadas a un nuevo fichero. La gravedad de este problema depende del espacio de almacenamiento disponible y del tamaño medio de los ficheros. La compactación de disco es una solución, pero es costosa y de poder hacerse sería en horas en las que el

El sistema operativo Amoeba usa este método de asignación. El problema de conocer el tamaño de los ficheros en tiempo de creación se resuelve haciendo que los ficheros sean inmutables. Esto significa que cuando un fichero se crea no se puede modificar. Una modificación implica borrar el fichero original y crear uno nuevo. Por otro lado, los ficheros se han de leer de una sola vez en memoria, por lo que la eficiencia es muy alta, pero a cambio se exigen unos requisitos mínimos de memoria muy elevados. Por último, la fragmentación de disco se soluciona teniendo discos de gran capacidad de almacenamiento.

3.4.2.2. Lista encadenada

Otro esquema consiste en mantener una lista enlazada con los bloques que pertenecen a un fichero, de forma que una palabra del bloque sería un índice al bloque siguiente. Con este método se elimina el problema de la fragmentación, y, al igual que en la asignación contigua, cada entrada de directorio únicamente contiene la dirección del primer bloque del fichero. Los ficheros pueden crecer de forma dinámica mientras que exista espacio libre en disco y no es necesario compactar los discos. Un inconveniente es que, mientras que el acceso secuencial es fácil de realizar, el acceso aleatorio es muy lento.

Otro inconveniente viene dado por el espacio requerido por los punteros. Si un puntero requiere 4 bytes y los bloques son de 512 bytes, se pierde un 0.78 por ciento de espacio útil por bloque. Una solución a este problema consiste en no asignar bloques individuales, sino conjuntos de bloques denominados *clusters*. De esta forma el porcentaje de espacio perdido por los punteros se reduce. Además, se mejora el rendimiento de los discos porque se lee más información en una misma operación de lectura, y se reduce la lista de bloques libres. El coste de este método es que se produce un incremento de la fragmentación interna (espacio no ocupado en un bloque o cluster).

Otro problema de este método es la fiabilidad, ya que si un bloque se daña, se puede perder el resto del fichero. Peor aún, si se altera el contenido del puntero, se puede acceder a la lista de bloques libres o a bloques de otro fichero como si fuesen bloques propios.

Un problema más sutil viene dado por el hecho de que la cantidad de información que contiene cada bloque no es potencia de dos, ya que el puntero al bloque siguiente estará constituido por varias palabras. Esto puede originar una pérdida de eficiencia porque los programas normalmente leen y escriben en bloques cuyo tamaño es potencia de dos.

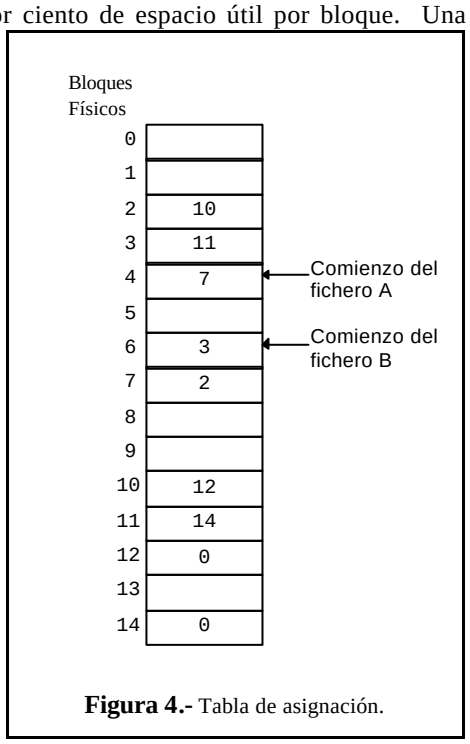


Figura 4.- Tabla de asignación.

3.4.2.3. Lista encadenada mediante tabla de asignación

Las desventajas del esquema de lista encadenada se pueden eliminar si el puntero de cada bloque de disco se almacena en una tabla o índice en memoria. De esta forma, cada bloque únicamente contiene datos. Además, aunque el acceso aleatorio implica seguir una cadena, ésta está toda en memoria, por lo que la búsqueda es mucho más eficiente que en el esquema anterior. De igual modo, es suficiente que cada

index-node o *i-node*), que incluye información sobre los atributos del fichero y las direcciones de los bloques de disco que pertenecen al mismo.

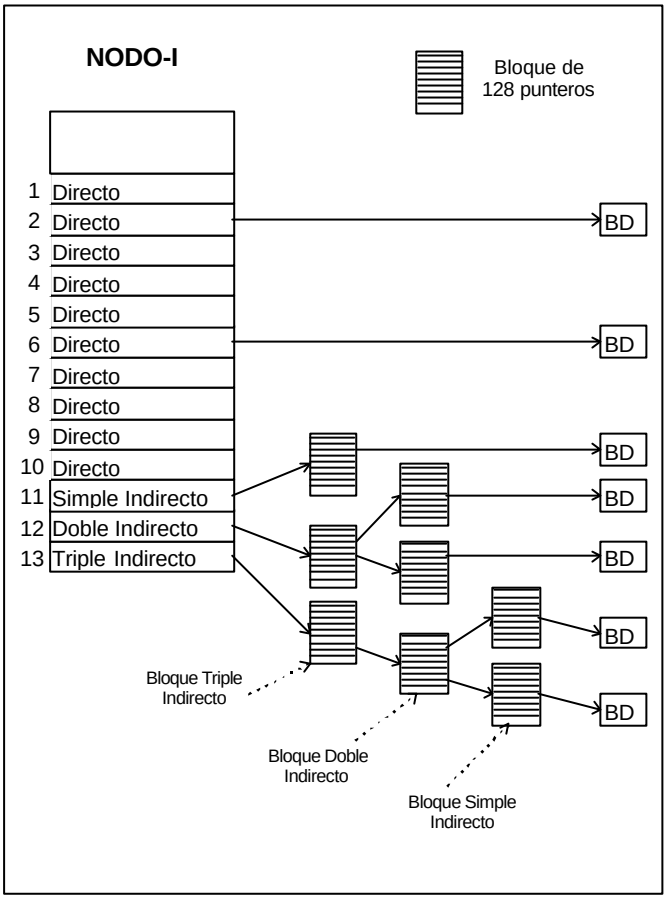


Figura 5.- Nodo índice en UNIX.

Las direcciones de los primeros bloques se almacenan el propio nodo índice, de forma que, para ficheros pequeños, toda la información necesaria para su localización siempre está en memoria cuando son abiertos. Para ficheros de mayor tamaño, una de las direcciones del nodo índice es la dirección de un bloque simple indirecto, que contiene a su vez direcciones de otros bloques del fichero. Si no es suficiente, existen dos direcciones, una para un bloque doble indirecto, que contiene direcciones de bloques simples directos, y otra para un bloque triple indirecto, que contiene direcciones de bloques dobles indirectos.

Una de las ventajas de este método es que permite el acceso directo a cualquier bloque del fichero. Además, los accesos a un fichero, una vez abierto éste, no implican accesos adicionales a disco a no ser que sean necesarios bloques indirectos.

Un inconveniente es que añadir o borrar bloques en medio del fichero implica el tener que reorganizar los punteros en el bloque de índices, lo cual constituye una operación costosa. El sistema operativo UNIX, que usa este esquema, soluciona el problema prohibiendo estas operaciones.

3.4.3. Implementación de Directorios

Para acceder a un fichero, ya sea para lectura o escritura, previamente hay que abrir el fichero. Esta operación implica que el sistema operativo, a partir de la ruta de acceso, debe de localizar la entrada de directorio correspondiente. Esta entrada proporciona la información necesaria para localizar los bloques de disco del fichero. Esta información, dependiendo del sistema, puede ser la dirección en disco del fichero completo (asignación contigua), el número del primer bloque (los dos métodos basados en listas encadenadas) o el número de nodo índice. En cualquier caso, la principal función del sistema de directorios es asociar el nombre en ASCII de un fichero con la información necesaria para localizar los datos. Los atributos de los ficheros se pueden almacenar en la misma entrada de directorio. Si usan nodos índice, otra posibilidad es almacenar los atributos en el nodo índice.

A continuación se exponen brevemente la implementación de directorios en los sistemas operativos CP/M, MS-DOS y UNIX.

3.4.3.1. Directorios en CP/M

En CP/M solamente existe un directorio, por lo que el sistema de ficheros únicamente tiene que buscar el nombre de un fichero en dicho directorio. Si un fichero ocupa más de 16 bloques se le asignan más entradas de directorio.

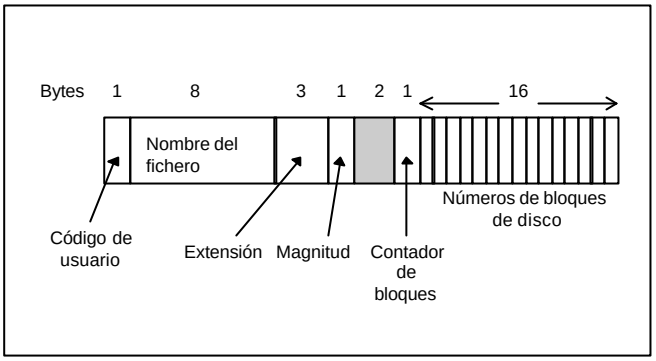


Figura 6.- Entrada de directorio en CP/M.

Los campos de una entrada de directorio en CP/M son los siguientes:

- El código de usario (1 byte) permite conocer el propietario del fichero.
- El nombre y extensión del fichero (8 y 3 bytes, respectivamente).
- Si un fichero ocupa más de 16 bloques, el campo magnitud (1 byte) indica el orden que ocupa esa entrada.
- El contador de bloques (1 byte) indica cuántos bloques están en uso de los 16 posibles.
- Los últimos 16 campos contienen las direcciones de bloques de disco del fichero. Como el último bloque puede no estar lleno, es imposible conocer con exactitud el tamaño en bytes de un fichero.

3.4.3.2. Directorios en MS-DOS

MS-DOS tiene un sistema de ficheros jerárquico. Cada entrada de directorio tiene 32 bytes, y contiene, entre otros datos, el nombre del fichero, la extensión, los atributos y el número del primer bloque en disco

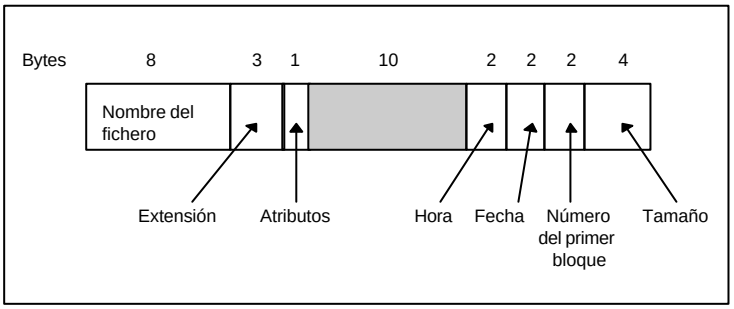


Figura 7.- Entrada de directorio en MS-DOS.

del fichero. Este número es un índice a una tabla de asignación denominada FAT (*File Allocation Table*). Un directorio puede contener otros directorios, lo que origina la estructura jerárquica del sistema de ficheros.

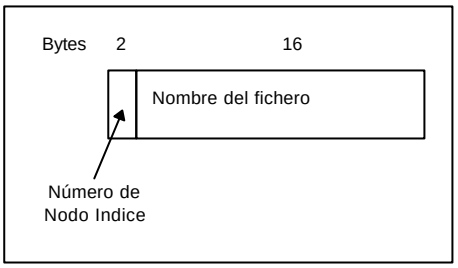
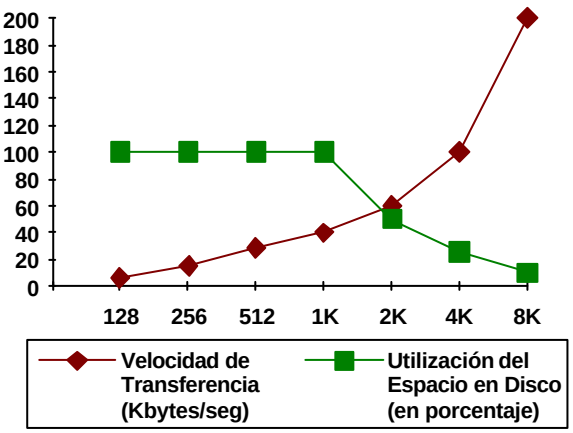


Figura 8.- Entrada de directorio en UNIX.

Por ejemplo, si un disco tiene 32768 bytes por pista, un tiempo de rotación de 16.67 milisegundos y un tiempo medio de búsqueda de 30 milisegundos, el tiempo en milisegundos para leer un bloque de k bytes es la suma del tiempo de búsqueda, el tiempo de latencia o retraso rotacional y el tiempo de transferencia. En este caso:

$$30 + 8.3 + (k / 32768) * 16.67$$

Si se asume que todos los ficheros ocupan 1KB, la siguiente gráfica muestra la velocidad de transferencia y la utilización del espacio del disco:



Si se estudia la gráfica, se observa que a partir de 1K aumenta la velocidad de transferencia pero disminuye

Lo usual es elegir un tamaño de bloque que oscila entre 512 y 2K bytes. Si se elige un tamaño de 1K y el disco tiene sectores de 512 bytes, el sistema de ficheros leerá o escribirá siempre dos sectores consecutivos.

3.4.4.2. Gestión del Espacio Libre

Para determinar los bloques que quedan libres el sistema de ficheros mantiene una lista de bloques libres. Cuando se crea un fichero, los nuevos bloques que necesita se toman de esta lista. De igual modo, cuando se borra un fichero sus bloques pasan a la lista de bloques libres.

La implementación de la lista de bloques libres se suele implementar de varias formas. Una forma consiste en usar un mapa de bits, donde cada bit representa un bloque. Si el bloque está libre, el bit esta a 1, y en caso contrario está a 0. Si un disco tiene n bloques, el mapa de bits tendrá n bits. La principal ventaja de este método es que es relativamente simple y eficiente la búsqueda del primer bloque libre. El principal inconveniente es que si el disco contiene muchos bloques, la mapa de bits puede no caber en memoria, lo cual hace que sea ineficaz. Por ejemplo, si un disco de 200 MB tiene un tamaño de bloque de 1024 bytes, se necesitan 200K para almacenar los bloques libres. En estos casos, se suelen usar clusters de bloques para reducir el tamaño de la lista de bloques libres.

Otro método consiste en enlazar todos los bloques libres del disco, manteniendo el puntero al primer elemento de la lista en un lugar conocido del disco y copiado en memoria. Si se usa un método de asignación por lista enlazada mediante tabla de asignación, la lista de bloques libres se gestiona como un

Por último, se puede usar una lista encadenada de bloques, cada uno de los cuales contiene un determinado número de direcciones de bloques libres. Los bloques de esta cadena puede ser bloques libres, por lo que no se desperdicia espacio útil.

3.5. Fiabilidad

Aunque es imposible el asegurar un sistema de ficheros contra fallos físicos de los discos o la destrucción física del sistema informático, hay que tratar de proteger la información todo lo posible. La técnica más común para asegurar la disponibilidad de la información es hacer copias de seguridad periódicas (*backups*). Estas copias se suelen hacer en cintas y se almacenan en un lugar seguro. Esto a veces no es suficiente porque todas las modificaciones desde la última copia se pueden perder.

Otra técnica es registrar todas las transacciones en un fichero que se encuentra en otro disco. Este método es costoso, pero en caso de fallo todo el trabajo se puede recuperar (si el disco con el fichero de transacciones no se ha destruido).

Como no hay forma de asegurar de forma absoluta la seguridad de los ficheros, el sistema operativo debe de proporcionar suficientes mecanismos para obtener copias de seguridad sin que se degrade el rendimiento del sistema. Sin embargo, en situaciones en las que la integridad de los datos es crítica, no hay otra opción que usar algún mecanismo de copias de seguridad a costa de degradar el rendimiento. En el peor de los casos, los usuarios tendrán que reconstruir todas las transacciones a partir de la última copia.

El método de hacer copias de seguridad periódicas tiene varios inconvenientes:

- El sistema puede tener que se parado durante el proceso de copia.
- El sistema de ficheros puede ser muy grande, por lo que el proceso de copia puede durar varias horas.
- Cuando se produce un fallo, la recuperación de la última copia puede durar mucho tiempo.

Una ventaja de las copias de seguridad es que el sistema de ficheros se puede reorganizar para permitir que los bloques de los ficheros de usuario que estén repartidos por el disco se almacenen de forma consecutiva.

Una alternativa a realizar una copia de todo el sistema de ficheros con relativa frecuencia es hacer una copia incremental (*incremental dumping*). La forma más sencilla de hacer una copia incremental es hacer una copia de seguridad completa de forma periódica, semanal o mensual, y hacer una copia incremental de

forma diaria únicamente de aquellos ficheros que han sido modificados desde la última copia completa. Una mejora es copiar los ficheros que han sido modificados desde la última copia incremental.

3.6. Rendimiento de un Sistema de Ficheros

La lectura de una palabra de memoria cuesta del orden de varias decenas de nanosegundos. Por el contrario, el tiempo de lectura de un bloque de disco es del orden de varias decenas de milisegundos, una diferencia de seis órdenes de magnitud respecto a un acceso a memoria. Como consecuencia de esta diferencia, muchos sistemas de ficheros han sido diseñados para reducir el número de accesos a disco.

La técnica más usada es el uso de zonas de memoria para almacenamiento temporal de bloques de disco (*buffer cache*). En esta memoria *cache* se almacenan bloques que lógicamente están en disco, pero que están en memoria para mejorar el rendimiento. El algoritmo habitual para manejar esta memoria es comprobar si los bloques que se solicitan están en ella, en cuyo caso no es necesario hacer lectura a disco alguna. Si no es así, el bloque es leído de disco y copiado en la memoria.

Cuando la memoria de almacenamiento temporal de bloques está llena y hay que cargar un nuevo bloque, hay que elegir un bloque para ser eliminado de ella y reescrito en disco si ha sido modificado. Sin embargo, el uso de una técnica de reemplazamiento LRU no es deseable porque la probabilidad de fallo en referencias a bloques es pequeña, lo que da lugar a que los bloques estén modificados en memoria mucho tiempo, con el consiguiente aumento del riesgo de inconsistencia de datos ante algún fallo del sistema. Si el bloque es de nodos índice, todo el sistema de ficheros quedará en estado inconsistente.

Es necesario modificar el esquema LRU de forma que tenga en cuenta si:

1. el bloque va a ser necesitado en breve tiempo, o
2. el bloque es esencial para la consistencia del sistema de ficheros.

Los bloques que no se vayan a necesitar se colocan al comienzo de la lista LRU, y los que se estime que se van a usar, como bloques de datos incompletos que están siendo modificados, se colocan al final. Si un bloque es esencial para la integridad del sistema de ficheros, cada modificación del mismo debe de provocar una escritura del bloque en disco, independientemente de la posición que ocupe dentro de la

De cualquier forma, es indeseable mantener en memoria cualquier bloque modificado durante un espacio de tiempo relativamente grande. En UNIX existe una llamada al sistema, `sync`, que copia en disco todos los bloques modificados. Existe un programa que cada treinta segundos hace una llamada `sync`, por lo que como máximo se pierden treinta segundos de trabajo en caso de fallo. En MS-DOS, cada bloque modificado es escrito inmediatamente en disco. Este tipo de memorias *cache* se denominan *write-through caches*. El esquema de UNIX es más eficiente, pero el de MS-DOS más seguro.

Otra técnica para aumentar el rendimiento de un sistema de ficheros es reducir el desplazamiento de la cabeza de lectura/escritura del disco. Una forma de conseguirlo es agrupar en el disco, preferiblemente en el mismo cilindro, aquellos bloques que se van a usar de forma consecutiva. Esto implica que cuando se está escribiendo un nuevo bloque de un fichero, el sistema debe de buscar en la lista de bloques libres uno que esté próximo al resto de bloques del fichero. Si se usa una lista encadenada, de la que habrá una parte en disco, ésta búsqueda será más difícil que si se usa un mapa de bits, que estará en memoria.

3.7. Referencias

- “Modern Operating Systems”. A.S. Tanenbaum. Prentice-Hall. 1992.
- "Operating Systems Concepts". Silberschatz, Peterson. Addison-Wesley, 1994.
- "Sistemas Operativos, Segunda Edición". Deitel, H. M., Addison-Wesley, 1992.
- “Operating Systems, Second Edition”. Stallings, W. Prentice-Hall. 1995.