6 Gestión de ficheros

En este capítulo se estudian los aspectos de la implementación de los sistemas de ficheros, fundamentalmente los relacionados con la gestión de direcciones. Se analizan las alternativas en la ubicación de bloques sobre el dispositivo soporte (disco) desde el punto de vista del rendimiento y la funcionalidad que soportan.

Contenido

6.1 Introducción

De un modo genérico, desde el punto de vista de usuario, se puede decir que un fichero es un conjunto de información con una serie de características:

- La información se almacena de modo **permanente**, a diferencia de otros objetos (variables, procesos).
- Se identifican mediante un **nombre**, representado por una cadena de caracteres que puede tener una parte con significado especial (p. ej., pueden tener *extensión*) o no, y se agrupan en **directorios** con una estructura de organización jerárquica para constituir un **Sistema de Ficheros**. A veces, un fichero se puede **compartir** desde varios directorios (mediante enlaces).
- Se aplican diferentes **operaciones** sobre ficheros (*crear*, *abrir*, *leer*, *escribir*, *posicionar*, *cerrar*, *borrar*, *ejecutar*…).
- En los sistemas multiusuario se protegen mediante permisos o **derechos de acceso**, en cuanto al tipo de operación y el usuario que hace el acceso.
- Varios procesos pueden **acceder concurrentemente** a un fichero, al menos para alguna de las operaciones.
- La información de un fichero se agrupa en **registros lógicos**, de tamaño fijo o variable. Lo normal en el sistema operativo es considerar el carácter como unidad de acceso de las llamadas al sistema. Los registros lógicos habitualmente se acceden de forma **secuencial,** aunque pueden también ser accedidos de modo **directo**, y se **organizan** de manera **consecutiva**, pero también son posibles otras formas de organización, como **al azar** e **indexada**.

Un sistema de ficheros requiere un dispositivo soporte que cumpla determinadas propiedades. Fundamentalmente, el dispositivo debe proporcionar almacenamiento permanente (no volátil). Además, debe permitir un número ilimitado de accesos, tanto de lectura como de escritura, aunque también hay sistemas de sólo lectura, o sistemas para los que es suficiente, o incluso preferible, que se escriban una sola vez. En las últimas décadas, los discos magnéticos se han venido utilizando como soporte habitual para los sistemas de ficheros. Para sistemas de sólo lectura, o de una única escritura, se utilizan sobre todo los discos ópticos (también los hay regrabables). En los últimos años han surgido las tarjetas de memoria tipo *flash*, que se utilizan en dispositivos de pequeño tamaño (teléfonos, PDAs, cámaras, reproductores de audio/video) o como almacenamiento auxiliar. Dada su rápida evolución, es de esperar que ganen terreno para otros usos. Además, para las tarjetas se están adoptando los mismos sistemas de ficheros utilizados para los discos magnéticos, lo

que simplifica su integración. Finalmente, algunos sistemas permiten definir ficheros sobre memoria RAM, lo que tiene interés para ficheros temporales.

El sistema de ficheros utiliza el **bloque** como unidad de acceso y ubicación. Para el sistema de ficheros el dispositivo es un vector de bloques. El bloque proporciona independencia entre las aplicaciones (registros lógicos), y la estructura del dispositivo (cilindros, pistas y sectores).

El modelo UNIX también trata como ficheros los elementos de comunicación como *pipes* y *colas FIFO* (buzones). Su organización es similar a la de un fichero que se accede secuencialmente y se incluyen en el sistema de ficheros como ficheros especiales. Siguiendo esta tendencia, también se integran en el sistema de ficheros los dispositivos de entrada/salida e incluso otros objetos del sistema, como los procesos (por ejemplo en Linux).

La implementación de un sistema de ficheros en disco plantea un buen número de necesidades de gestión a nivel del sistema operativo:

- Gestión de directorios y nombres (caminos, enlaces, etc).
- Protección de accesos para gestión de la seguridad, en sistemas multiusuario.
- Gestión de canales (independencia del dispositivo) y de accesos concurrentes (a ficheros y buzones).
- Traducción de direcciones. Dada una operación de acceso a un registro lógico de un fichero, se requiere encontrar el (los) bloque(s) en el dispositivo que contienen ese registro lógico.
- Gestión de la ubicación de los bloques y el espacio libre en el dispositivo, para proporcionar una utilización eficiente del espacio y tiempos de acceso mínimos.
- Gestión de la cache de bloques en memoria.
- Gestión de errores y mantenimiento de la consistencia del sistema de ficheros.

Además, como se trató en el capítulo de gestión de dispositivos, a nivel del manejador hay que resolver los problemas derivados del acceso al dispositivo, como la planificación eficiente de los accesos y el tratamiento de errores.

6.2 Gestión de directorios y nombres

Algunos sistemas de ficheros del pasado tenían un directorio único, pero los sistemas actuales poseen una estructura jerárquica de ficheros. En principio, esta estructura es

un árbol donde los nodos son directorios y las hojas ficheros, aunque la compartición de ficheros mediante enlaces da lugar a nodos con múltiples referencias. La tendencia, como hace UNIX, es considerar los directorios como un tipo especial de ficheros.

6.2.1 Representación de los directorios

Los directorios se almacenan en el disco. Cada entrada de un directorio permite acceder a la **descripción** (o **atributos**) del fichero y a su **contenido**[1]. La descripción comprende información acerca de la identificación, propiedades y características del fichero (nombre, tipo, propietario, derechos de acceso, fechas de creación y de último acceso y modificación, tamaño, número de enlaces). El contenido son los datos que se almacenan en el fichero en disco, identificados, en general, mediante apuntadores a los bloques (se verá en detalle en la gestión de la ubicación).

Básicamente, se siguen dos enfoques para representar los ficheros:

(a) Especificar en cada entrada de un directorio el nombre del fichero y un apuntador a una estructura que contiene la descripción del fichero y apuntadores al contenido. Un ejemplo es UNIX (Figura 6.1a), que contiene la descripción del fichero en lo que se denomina **i-nodo** o *i-node*.

(b) Especificar en cada entrada toda la descripción del fichero y un apuntador o apuntadores al contenido. Ejemplos son CP/M y MS-DOS[2] (Figura 6.1b).

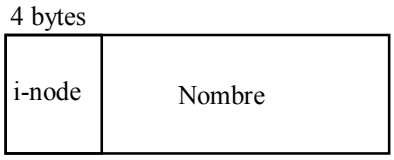

Figura 6.1a. Entrada de directorio en UNIX

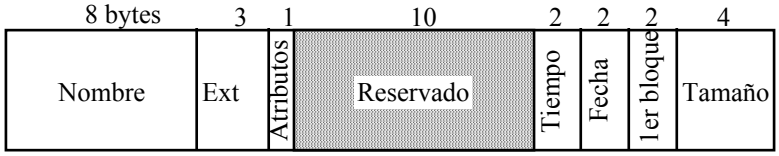

Figura 6.1b. Entrada de directorio en MS-DOS

[1] Para el sistema de ficheros de Windows NT (NTFS), todo, incluso el contenido, son atributos.

[2] El sistema de ficheros de MS-DOS, basado en una FAT y que estudiaremos más adelante, sigue siendo soportado por los sistemas actuales y es el utilizado para las tarjetas flash.

La descripción del fichero se carga en memoria bien cuando se lee el directorio (en el caso de MS-DOS), bien cuando se abre el fichero (en el caso de UNIX, que utiliza una tabla de i-nodos para ello).

6.2.2 Compartición de ficheros desde directorios

En muchos sistemas se puede compartir un fichero por varios directorios[3], mediante **enlaces**, que pueden ser:

(a) **Hardware**. Dentro de la descripción del fichero, un campo indica el número de enlaces al fichero, entradas de directorios que apuntan a él. Las llamadas al sistema de borrar fichero, crear fichero y establecer un nuevo enlace (*link* en UNIX) modifican este campo. Solo se liberan los bloques que ocupa el fichero en disco (descripción y contenido) cuando la cuenta de enlaces llega a cero. Para evitar ciclos en el sistema de ficheros, sistemas como UNIX no permiten establecer enlaces hardware a directorios.

(b) **Software**. Un fichero de un tipo especial contiene el camino al fichero que se quiere compartir. Los comandos y las aplicaciones son quienes interpretan estos ficheros y resuelven su contenido como un nombre absoluto. Ejemplos son los sistemas de ficheros para MS-DOS y Windows basados en FAT, donde estos enlaces se denominan *accesos directos*).

Los enlaces hardware proporcionan transparencia y consistencia en el manejo de los ficheros[4]. La estructura de la representación de los ficheros es determinante para permitir enlaces hardware. Como se observa en la Figura 6.1a, la referencia a través de apuntadores desde los directorios al i-nodo que representa al fichero permite implementar los enlaces hardware manteniendo una cuenta de enlaces como atributo del fichero dentro del inodo[5]. En sistemas FAT no es posible implementar un mecanismo semejante (Figura 6.1b).

6.2.3 Gestión de nombres

El nombre de un fichero se entiende como **relativo** al directorio que lo apunta. El nombre **absoluto** de un fichero incluye como prefijo todo el **camino** en el árbol de directorios desde el **directorio raíz** del Sistema de Ficheros. Para cada proceso se

---

[3] Incluso, en sistemas tipo UNIX, dentro de un mismo directorio es posible compartir un fichero con dos nombres.

[4] En cambio, en los sistemas Windows antiguos (basados en FAT), si se mueve o elimina el fichero enlazado, los accesos directos pierden la referencia.

[5] La estructura del i-nodo se representa en la Figura 6.10.

mantiene una variable del contexto del proceso que indica el directorio de trabajo actual.

El nombre se resuelve en el momento de abrir el fichero. Si el fichero se encuentra en el directorio de trabajo actual, el proceso lo accede directamente desde la entrada del directorio. En caso contrario es necesario recorrer todo el camino especificado en el nombre absoluto, abriendo cada directorio y leyendo su contenido hasta alcanzar el fichero.

6.3 Protección de accesos y seguridad

Se refiere a los criterios y mecanismos de protección que se establecen para el acceso a ficheros. En general, protección se suele enfocar desde un punto de vista genérico que abarca no sólo ficheros sino cualquier elemento hardware (memoria, dispositivos…) o software (procesos, rutinas, ficheros…). En el caso de los ficheros, los derechos de acceso suelen especificarse como parte de la información acerca de la descripción del fichero.

En un modelo de protección general, se denomina **objeto** al elemento del sistema susceptible de ser protegido. Un objeto se identifica por un nombre y sobre él se definen un conjunto de operaciones. Al permiso para realizar una operación sobre un objeto se le denomina **derecho**. El sistema operativo tiene dos alternativas para comprobar los derechos de acceso a un fichero: al abrir el fichero (como hacen los sistemas UNIX), o en cada operación de acceso.

Un **dominio** identifica un conjunto de pares (objeto, derechos) y expresa determinadas condiciones de acceso para los sujetos a los que se aplica. Por ejemplo, un usuario del sistema constituye un dominio, pues se establecen unos derechos de acceso determinados a los ficheros, por ejemplo:

Fich1 [RWX], Fich2 [R], Fich3 [RX]

Conceptualmente, la forma de representar el control de accesos en un sistema es mediante una matriz de protecciones. Cada fila de la matriz representa un dominio y cada columna un objeto. Para un elemento *(i, j)* de la matriz se especifican los derechos del dominio *i* sobre el objeto *j*.

La matriz de protecciones de un sistema tendría un tamaño enorme y sería poco densa, así que se utilizan dos mecanismos alternativos para implementar la protección de accesos, que básicamente consisten en representar únicamente los elementos no nulos de la matriz, por filas o por columnas:

(a) **Lista de Control de Accesos** (ACL). Cada elemento de esta lista recoge los derechos de acceso de un objeto, especificando, para cada dominio que tenga

derechos sobre el objeto, cuáles son éstos. Los derechos de acceso así representados se acceden como atributos de un fichero.

(b) **Credenciales** *(capabilities).* Para cada dominio se especifican los objetos sobre los que tiene derechos, indicando para qué operaciones. Este es un concepto análogo al de la tarjeta de acceso que porta un individuo para acceder a determinadas estancias (por ejemplo, en un recinto universitario). Cada dominio (tipo de tarjeta) se asocia a un determinado colectivo de individuos (alumnos, profesores...) y representa las credenciales que determinan los derechos de acceso a los objetos (a qué estancias, en qué horarios...). Para modificar los derechos de acceso se proporcionan nuevas credenciales (nuevas tarjetas). A una credencial se le puede asociar un tiempo de vida, más allá del cual deja de ser válida. Los procesos del sistema operativo representan las credenciales como parte de su contexto.

Los permisos se comprueban generalmente en las llamadas al sistema de *abrir*, *borrar*, *ejecutar* y establecer un enlace (*link* en UNIX). Sin embargo, algunos sistemas realizan la comprobación también en otras operaciones, como *leer* y *escribir*.

En UNIX, la implementación de la protección de accesos a ficheros es una lista de control de accesos muy simple, ya que sólo se definen tres dominios con derechos de acceso a un fichero: usuario propietario del fichero, grupo propietario, y otros usuarios. Como los derechos de acceso sobre ficheros son de tres tipos (r, w, x), se utilizan 9 bits por fichero para establecer su protección, que se almacenan en el i-nodo del fichero, junto con los identificadores de usuario y grupo propietarios del fichero. Cada proceso lleva asociados también sus identificadores de usuario (*uid*) y grupo (*gid*), que definen sus dominios.

6.4 Gestión de canales. Accesos concurrentes

A nivel de llamadas al sistema no se distingue entre fichero y dispositivo en las operaciones de acceso. El objetivo es proporcionar a las aplicaciones independencia del fichero concreto o dispositivo físico. Esta independencia se implementa proporcionando una **tabla de canales** por proceso.

La llamada al sistema *abrir* crea una nueva entrada en la tabla de canales (**descriptor del fichero**), cuyo índice devuelve para ser usado en sucesivas operaciones (p. ej. *leer* o *escribir*). La llamada al sistema *cerrar* elimina la entrada que se especifica de la tabla de canales. El descriptor del fichero hace referencia a información que se estructura a través de una serie de indirecciones. En general, esta información comprende:

- El nombre del fichero.

- Un apuntador al descriptor del dispositivo que contiene el fichero (introducido en el capítulo de gestión de dispositivos).
- La ubicación del (primer bloque del) fichero en el dispositivo.
- La siguiente posición del fichero a la que se va a acceder.
- El modo de operación (*lectura*, *escritura*, *append*).
- El número de procesos que tienen abierto el fichero, tanto para lectura como para escritura.

Para explicar el modelo de compartición que vamos a introducir consideraremos que la información referenciada por el descriptor del fichero se almacena en una **tabla de ficheros abiertos**. Aunque el modelo UNIX considera un nivel más de indirección, este modelo será suficiente para nuestros propósitos.

La llamada al sistema *abrir* realiza los siguientes pasos:

(1) Comprueba que hay una entrada libre en la tabla de canales.

(2) Busca el fichero en el directorio correspondiente a partir del nombre (absoluto o relativo). Esto implica el acceso a cada directorio que forma parte del camino del fichero, y si el directorio no está cargado en memoria, su transferencia desde el dispositivo.

(3) Comprueba los derechos de acceso.

(4) Comprueba en la tabla de ficheros abiertos si el fichero ya ha sido abierto y su modo de operación. Determina cómo se va a compartir, aspecto que se estudiará a continuación.

(5) Si los pasos anteriores han tenido éxito, se crea el descriptor del fichero, estableciendo los apuntadores a la ubicación en el dispositivo del comienzo del fichero.

El paso (4) depende de la naturaleza del fichero abierto y del tipo del modo de operación. Si se trata de un fichero ordinario abierto en modo *lectura* o *escritura*, se crea para el descriptor del fichero una entrada propia en la tabla de ficheros abiertos. De esta forma, dos procesos que tengan abierto un mismo fichero accederán a él con apuntadores privados (Figura 6.2), proporcionando a cada proceso la sensación de que su acceso al fichero es independiente de otros accesos concurrentes. Algunos sistemas proporcionan sincronización de lectores/escritores en el acceso a ficheros por varios procesos, mientras que otros, como UNIX, no lo hacen, pasando la responsabilidad de una eventual sincronización a las aplicaciones.

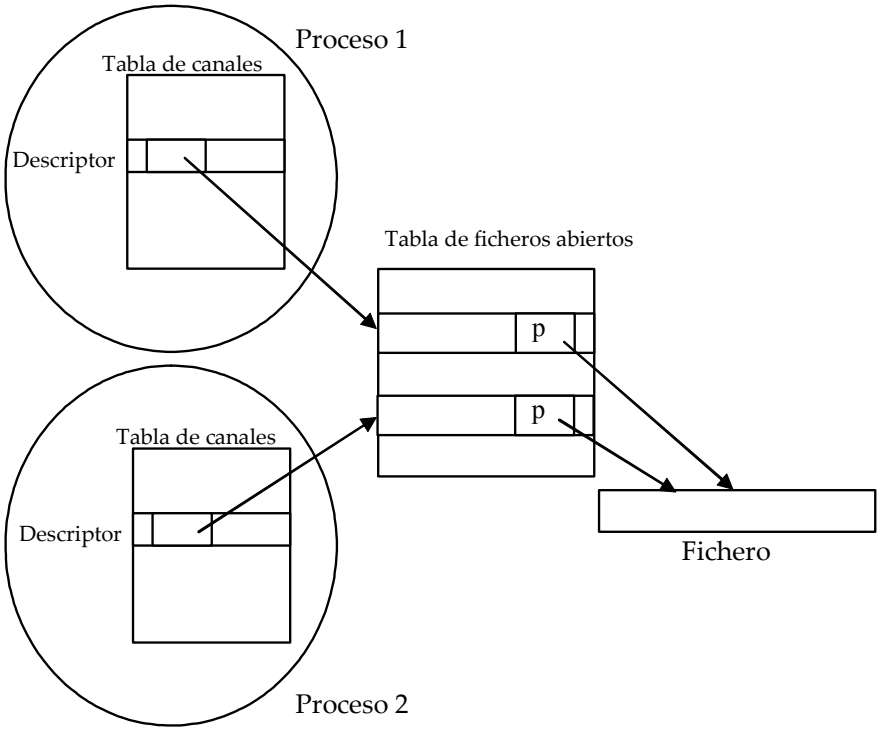

Figura 6.2. Acceso a ficheros mediante apuntador no compartido.

En cambio, cuando el fichero se abre en modo *append*, o cuando se trata de elementos de comunicación (*pipes* y buzones), se comparte una misma entrada de la tabla de ficheros abiertos por todos los procesos que lo mantienen abierto (Figura 6.3). La entrada compartida mantiene el puntero a la posición actual[6]. También hay que almacenar el número de referencias al fichero abierto (para lectura y escritura), para que lo maneje la llamada *cerrar*.

La implementación de la llamada al sistema *cerrar* consiste básicamente en liberar el descriptor del fichero y la entrada en la tabla de canales. Para gestionar la compartición de accesos, cerrar hace que se decremente la cuenta de referencias en la tabla de ficheros abiertos. Cuando se queda sin referencias, la entrada se libera.

Siguiendo con el modelo UNIX, la llamada al sistema de *borrar* ha de tener en cuenta que el fichero puede estar siendo usado por otros procesos. En este caso, marca el fichero como borrado y será la llamada al sistema *cerrar* quien se encargue de eliminarlo del directorio[7].

---

[6] En modo *append*, un puntero al último elemento añadido; en buzones un puntero de lectura y otro de escritura.

[7] Obsérvese que este modo de funcionamiento requiere un nivel de inderección más. De hecho, los sistemas tipo unix introducen una *tabla de i-nodos* en memoria que incluye los contadores de referencias (procesos que tienen abierto el fichero para lectura y para escritura).

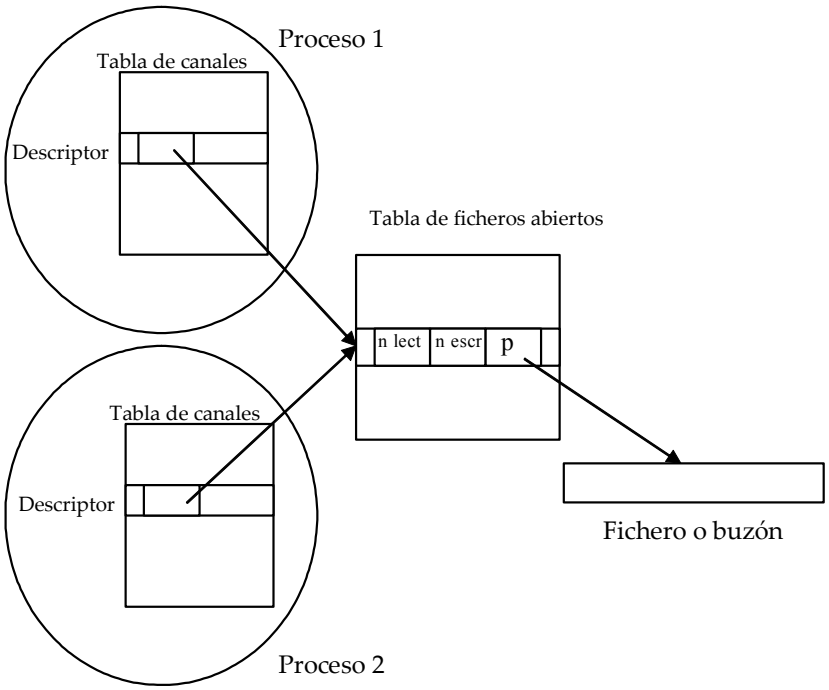

Figura 6.3. Acceso a ficheros mediante apuntador compartido.

6.5 Gestión de direcciones y de la ubicación

Mientras que las aplicaciones acceden a registros lógicos de ficheros, el dispositivo maneja la ubicación de esta información mediante parámetros como número de pista y sector. El proceso de traducción sigue un largo camino que involucra a la aplicación, el sistema de ficheros y el driver del dispositivo. En esta sección partiremos de la visión de los ficheros desde las aplicaciones, introduciremos la representación del espacio de direcciones del sistema de ficheros y el proceso de traducción de direcciones, estudiaremos las implicaciones del tamaño del bloque en el rendimiento, y presentaremos los modos de ubicación de los bloques en el dispositivo y de gestión del espacio libre.

6.5.1 Los ficheros vistos desde las aplicaciones

Desde el punto de vista del usuario y las aplicaciones un fichero se ve como un conjunto de **registros lógicos**. La **organización** del fichero y los **métodos de acceso** a los registros lógicos del mismo se resuelven mediante rutinas de librería específicas de las aplicaciones.

La organización de un fichero puede ser:

- Consecutiva. Los registros lógicos se ordenan secuencialmente. El registro lógico *n* ocupa la *n*-ésima posición lógica en el fichero.

- Al azar. El registro lógico *n* ocupa la posición lógica *f(n)* en el fichero. *n* es una clave para acceder al registro lógico mediante la función de acceso *f*.

- Indexada. Una tabla de índices permite conocer la posición lógica del registro.

Organizaciones más sofisticadas pueden combinar algunos de los tipos básicos.

El método de acceso para acceder a un registro lógico puede ser:

- Secuencial. Para acceder al registro lógico *n* hay que acceder los *n–1* anteriores.

- Directo. Sólo es necesario un acceso lógico para acceder a cualquier registro lógico.

Las aplicaciones particulares, mediante las librerías de E/S, son las encargadas de interpretar la organización del fichero y proporcionar a las llamadas al sistema parámetros de acceso normalizados, habitualmente la posición en bytes relativa al comienzo del fichero y la longitud del registro lógico también en bytes.

6.5.2 Espacios de direcciones en el sistema de ficheros

En lo que respecta al sistema de ficheros, los componentes del espacio de direcciones de las aplicaciones son los ficheros y sus registros lógicos, mientras que el espacio de direcciones del dispositivo depende de las características físicas de éste. Por ejemplo, como vimos en el capítulo precedente, para un disco estos parámetros están determinados por su geometría. El **bloque** es el elemento que permite independizar el espacio del sistema de ficheros con respecto al del dispositivo y constituye la piedra angular en el proceso de traducción de direcciones.

La capacidad de direccionamiento del sistema de ficheros está establecida, en principio, (1) por la longitud de los identificadores de bloques, y (2) por el tamaño de bloque. Por ejemplo, con identificadores de 2 bytes y bloques de 1 Kb pueden direccionarse 64 Mb de información (2^{16} por 1Kb). Estos parámetros delimitan la cantidad de información direccionable por el sistema de ficheros en el dispositivo. Si la capacidad de direccionamiento no cubre todo el dispositivo, se puede recurrir a **particionar** el dispositivo, creando tantos sistemas de ficheros sobre él como particiones, cada una con la capacidad de direccionamiento como límite de tamaño[8].

Ya que el tamaño de los identificadores es un parámetro fijo de un sistema de ficheros, la única forma de alterar el espacio direccionable es variar el tamaño de

[8] Esto resultaba familiar hace unos años en Windows 95, cuando los discos vivieron una época de aceleración en cuanto a incremento de tamaño.

bloque, que suele ser un parámetro configurable. Esto tiene implicaciones sobre el rendimiento, como se estudiará más adelante.

Finalmente, dejando aparte el propio tamaño del dispositivo, existe otro factor restrictivo para el espacio de direcciones: el modo de ubicación del sistema de ficheros, al que dedicaremos un apartado más abajo. Como veremos, el modo de ubicación puede limitar el tamaño máximo de un fichero, del sistema de ficheros, o de ambos.

6.5.3 Traducción de direcciones

Las rutinas de librería de las aplicaciones proporcionan un paso previo en todo el proceso de traducción para el acceso a la información. Las aplicaciones realizan *accesos lógicos* a los registros del fichero. El sistema de ficheros entiende de *accesos físicos* a los bloques del dispositivo. Se requiere un proceso de traducción de direcciones en varios pasos, culminado por la conversión a los parámetros del dispositivo que se estudió en el capítulo precedente. Un acceso lógico (a un registro del fichero) puede implicar ninguno, uno, o más accesos físicos al disco, dependiendo de si el registro lógico al que se quiere acceder (en una operación de lectura, por ejemplo), se encuentra todo o parcialmente cargado en memoria, y del tamaño del registro lógico con respecto al tamaño del bloque. Es incluso posible que una operación de lectura de un registro lógico implique un acceso extra al dispositivo para escribir el bloque antiguo que ocupa el buffer sobre el que hay que leer el nuevo. Como veremos más adelante, en sistemas con cache de bloques en memoria, el encontrar un determinado registro lógico en memoria dependerá también de la política de reemplazo de bloques que se siga.

El proceso de traducción completo se muestra en la Figura 6.4. En dicho esquema se muestra, para cada paso en la traducción, la función del sistema operativo que realiza la operación, qué parámetros utiliza, y de acuerdo a qué criterios realiza la traducción. El resultado de la traducción se utiliza como parámetro para el paso siguiente. Hay que enfatizar las tres fases de la traducción que soporta el sistema operativo:

(1) Las llamadas al sistema de lectura/escritura sobre ficheros obtienen a través de la tabla de canales la rutina de E/S que se encargará de la operación (rutina de acceso a ficheros). A partir de la tabla de canales se obtiene también el nombre del fichero al que se accede y el valor actual del puntero (compartido o privado a cada proceso), con el que se calcula el **bloque relativo** a comienzo de fichero, en función del **tamaño del bloque**.

(2) A partir del fichero y el número de bloque relativo se calcula el número de **bloque absoluto** del dispositivo. Esto se hace en función del **modo de ubicación** del sistema de ficheros, utilizando la información de ubicación del

fichero, que se obtiene a partir de la entrada del directorio correspondiente al fichero (por ejemplo, para MS-DOS, el puntero al primer bloque), o a partir de su i-nodo en sistemas UNIX. En este punto se determina si el bloque accedido está en un buffer en memoria o si es necesario cursar una petición de acceso al manejador del dispositivo.

(3) El bloque es la unidad de información que manejan las rutinas independientes del dispositivo. Como se estudió en la gestión del disco, el manejador del dispositivo traduce el número de bloque absoluto, de acuerdo a las características físicas del dispositivo concreto; en el caso del disco, número de cilindro, pista y sector, que son los parámetros que manejan las primitivas del núcleo del sistema operativo.

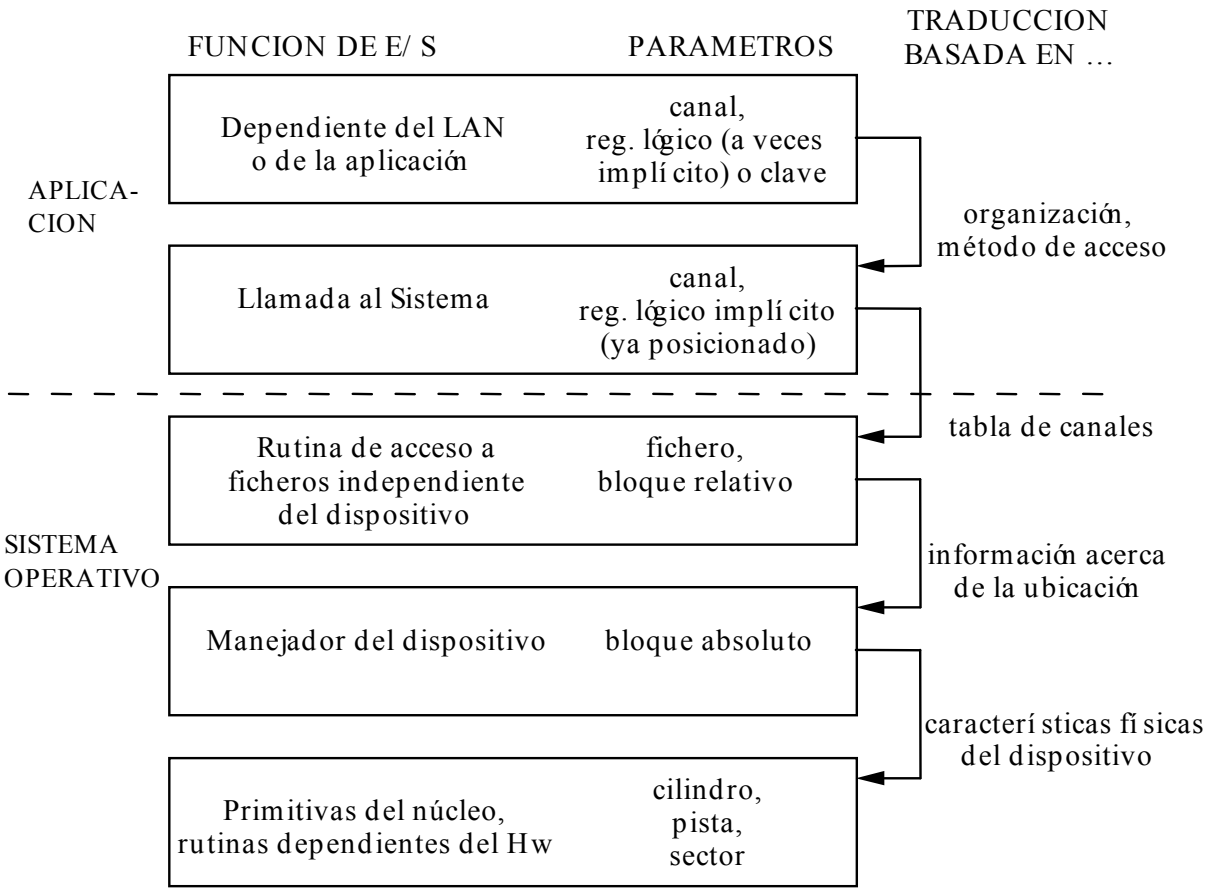

Figura 6.4. Esquema general de la traducción de direcciones

6.5.4 Tamaño del bloque

Como hemos visto, la unidad de información con la que trabaja el sistema operativo para la ubicación de ficheros es el bloque. Consistentemente, el manejador del dispositivo representa el bloque de acuerdo a sus características. Por ejemplo, en el caso del disco, un bloque es un conjunto de sectores lógicamente contiguos.

El tamaño de bloque está definido para el sistema de ficheros (puede ser configurable) y es independiente tanto del tamaño de los registros lógicos como de las características físicas del dispositivo donde se ubica el fichero. Sin embargo, como es obvio por razones de eficiencia y sencillez de implementación, en el caso de los discos será siempre un múltiplo del tamaño de sector.

El tamaño de bloque definido para el sistema de ficheros, aparte de afectar a la capacidad de direccionamiento, determina en gran parte el rendimiento del sistema de ficheros:

- Un tamaño de bloque grande aumenta la capacidad de direccionamiento y favorece la **tasa de transferencia** (cantidad de información transferida por unidad de tiempo, es decir, la productividad o *throughput* del sistema de ficheros), pero reduce la eficiencia espacial de los discos por fragmentación interna.

- Por el contrario, un tamaño de bloque pequeño reduce la fragmentación interna, pero requiere más cantidad de información para control de la ubicación y del espacio libre, disminuye la tasa de transferencia y proporciona menor capacidad de direccionamiento.

El compromiso entre estos factores (Figura 6.5) determina un tamaño adecuado de bloque, que además debe ajustarse a un número entero de sectores.

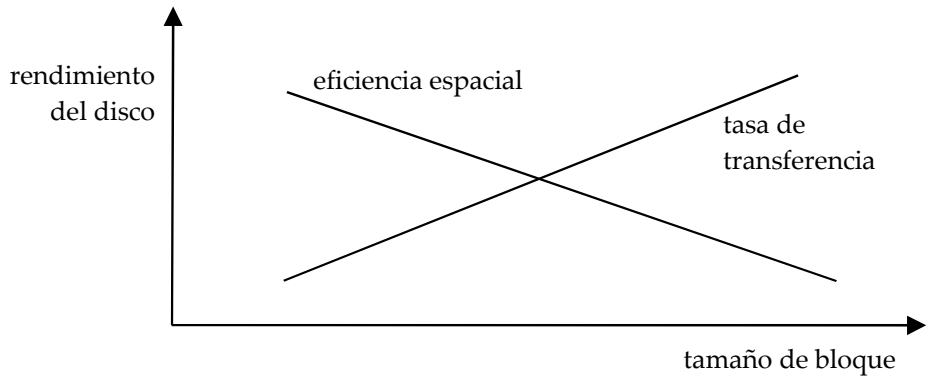

Figura 6.5. Rendimiento del disco en función del tamaño de bloque

6.5.5 Modos de ubicación

El objetivo de la gestión de la ubicación es proporcionar una utilización eficiente del espacio disponible en el dispositivo (disco) y del propio espacio de direcciones del sistema de ficheros.

Dependiendo de cómo se hacen corresponder los bloques relativos de los ficheros con los bloques absolutos del dispositivo, se pueden definir tres mecanismos básicos

de ubicación de sectores. Los analizaremos teniendo en cuenta criterios como la información de control que requieren, el número de accesos necesarios para alcanzar un bloque determinado, y la utilización que hacen del espacio. Por razones históricas, nos tomaremos al disco como dispositivo soporte de referencia.

6.5.5.1 Ubicación contigua

Los bloques del fichero se ubican consecutivamente en sectores contiguos del disco (Figura 6.6). El bloque relativo 1 se ubica en el bloque absoluto *i*, el 2 en el *i+1*, y así sucesivamente.

Se requiere como información de control únicamente el apuntador al primer bloque y el número de bloques del fichero.

El acceso directo a cualquier registro lógico es sencillo y no requiere accesos adicionales.

Requiere una definición inicial del tamaño del fichero, a partir del cual el crecimiento es muy problemático. La inserción de un bloque intermedio obliga a reubicar todos los bloques posteriores o incluso todo el fichero. Por su propia naturaleza, produce fragmentación externa y degradación del espacio de disco.

La ubicación contigua es adecuada para ficheros de sólo lectura, por ejemplo en discos CD-ROM.

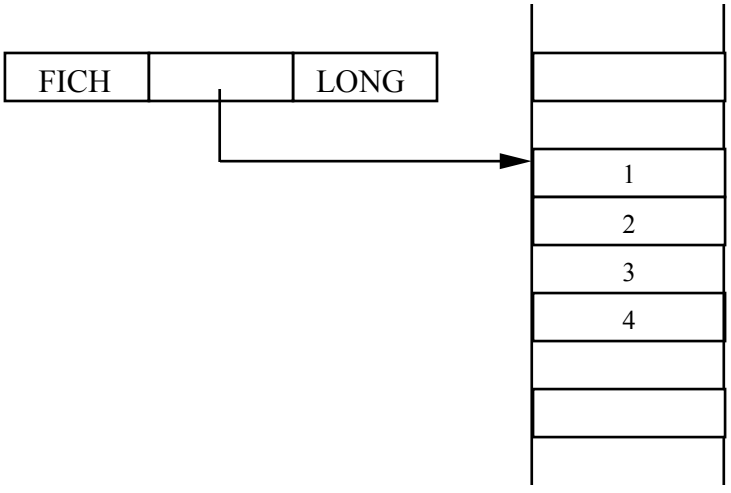

Figura 6.7. Ubicación contigua.

6.5.5.2 Ubicación encadenada

Los bloques del fichero se dispersan por los sectores libres del disco (Figura 6.7). Dentro de cada bloque se incluye un apuntador al siguiente bloque. Desde la información de control del fichero se requiere un apuntador al primer bloque.

No hay fragmentación externa; sólo se pierde espacio de almacenamiento por la información de control. El crecimiento de un fichero es sencillo.

En ubicación encadenada pura, el apuntador al bloque siguiente está incluido dentro del propio bloque en el disco. En este caso, el acceso directo no es posible, ya que para acceder al *n*-ésimo bloque hay que leer los *n–1* anteriores.

Otro problema de la ubicación encadenada es su escasa tolerancia a fallos. Si se daña un sector del disco que contiene un apuntador, se rompe la cadena de apuntadores y no es posible acceder a los bloques siguientes. Se puede aumentar la tolerancia a fallos definiendo la lista como doblemente encadenada, con un apuntador al último bloque en la descripción del fichero.

Una variante que minimiza estos problemas, y es la usada comercialmente, consiste en representar los apuntadores en una lista encadenada aparte, en vez de incluirlos dentro de los bloques de datos. Esta lista se carga en memoria y permite recorrer la información de control sin necesidad de leer los bloques de datos. Este mecanismo (**mapa de bloques**) se implementa en los sistemas operativos MS-DOS y Windows 9x, como se verá más adelante.

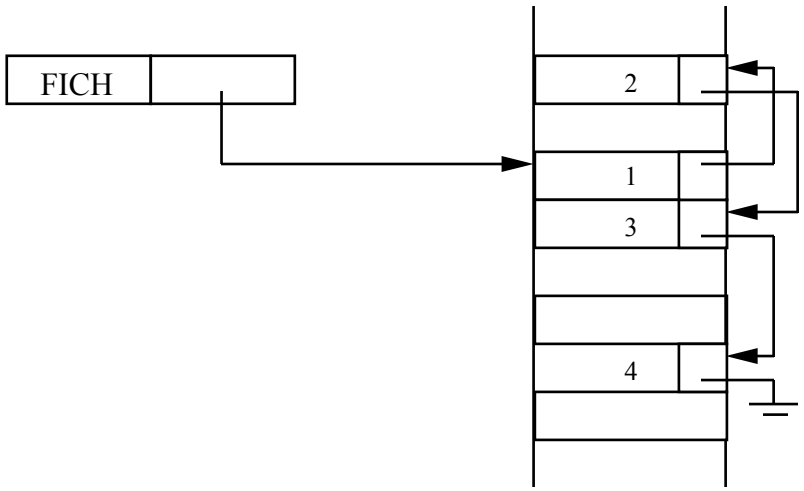

Figura 6.7. Ubicación encadenada.

6.5.5.3 Ubicación indexada

Para tratar de conjugar las ventajas de los dos modos de ubicación anteriores, la información de control se almacena aparte de forma contigua. Así, se tiene uno o más bloques de apuntadores (índices), que pueden cargarse en memoria al abrir el fichero, y los bloques de datos, libres de información de control (Figura 6.8).

En la información de control del fichero se requiere un apuntador al primer bloque de índices y el número de bloques de índices que se utilizan para el fichero.

Cuando se permite más de un bloque de índices, hay que resolver su ubicación, para lo que se puede utilizar cualquiera de los métodos anteriores, es decir, en bloques contiguos o encadenados, o incluso puede haber una jerarquía de bloques de índices, como en UNIX[9].

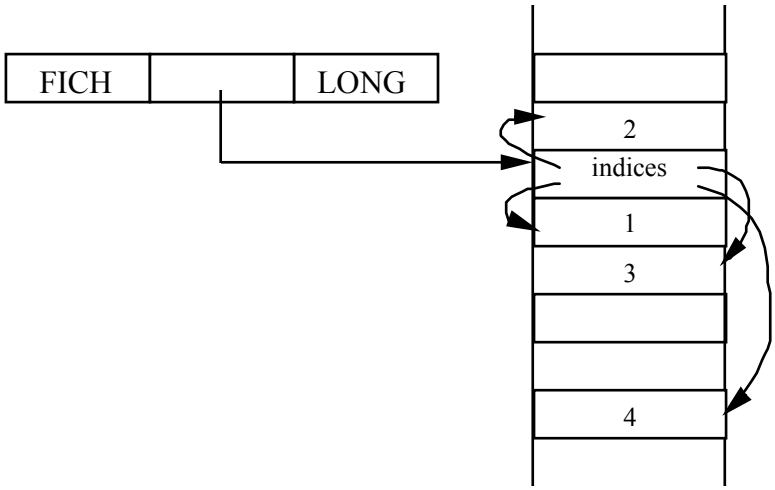

Figura 6.8. Ubicación indexada.

No existe fragmentación externa, pero sí fragmentación interna en la zona de índices. El crecimiento o modificación de un fichero obliga a modificar los bloques de índices y eventualmente a reubicarlos.

Los bloques de índices se almacenan en disco y pueden estar eventualmente cargados en memoria. El acceso directo es posible recorriendo la lista de índices una vez cargada en memoria.

6.5.6 Gestión del espacio libre

Como hemos visto, el espacio del sistema de ficheros se gestiona con particiones de tamaño fijo (bloques), lo que hace del **mapa de bits** un mecanismo adecuado para representar el espacio libre en la mayoría de los modos de ubicación, excepto en el contiguo.

Sin embargo, la elección de los bloques donde ubicar un fichero es un aspecto que puede afectar al rendimiento del dispositivo, sea cual fuere el modo de ubicación. Así, para un sistema de ficheros sobre disco, aún en modos de ubicación no contigua, un criterio suele ser la asignación de bloques consecutivos para un mismo fichero, ya que, debido a que los accesos a los ficheros son mayoritariamente secuenciales, se

[9] La ubicación de los bloques de índices plantea los mismos problemas que la ubicación de los bloques de datos, pero a una escala mucho menor, por lo que la elección del modo de ubicación de los bloques de índices resulta mucho menos crítica.

minimiza el número de posicionamientos de los cabezales y se optimizan los tiempos de espera por rotación. Por este motivo, los sistemas de ficheros comerciales suelen utilizar **listas de bloques libres**, bien como mecanismo único de representación, bien como elemento auxiliar en combinación con mapas de bits. Por los mismos motivos, tiene sentido hablar de fragmentación externa y de degradación del disco cuando los ficheros se almacenan de forma muy dispersa, lo que hace conveniente la **compactación**[10] de los ficheros. Obsérvese que el interés aquí es mantener compactos los bloques de un mismo fichero, no los huecos, ya que el problema a resolver es la pérdida de la eficiencia temporal, no la espacial.

La información sobre el espacio libre se almacena en disco y se carga en memoria.

6.5.7 Ejemplos de ubicación

6.5.7.1 MS-DOS

La ubicación de los ficheros en MS-DOS, el primer sistema operativo de Microsoft, se gestiona mediante un mapa de bloques único por sistema de ficheros, que se denomina **Tabla de Asignación de Ficheros** o **FAT**. Este mecanismo, concebido para discos flexibles, se ha mantenido en los sistemas Windows 95/98, con pocas modificaciones. El directorio raíz y la FAT, por duplicado, se almacenan en los primeros sectores del disco. Cada entrada del directorio de una unidad de disco contiene información acerca de la descripción del fichero y un apuntador al primer bloque del fichero.

La FAT contiene tantas entradas como bloques la unidad. Cada bloque está representado por una entrada de la FAT. El contenido de una entrada de la FAT que representa el bloque relativo *i* de un fichero es un apuntador a otra entrada de la FAT, que representa el bloque *i+1*, de forma que la ubicación de un fichero se representa como una lista encadenada en la FAT. Los bloques libres y los dañados se identifican con códigos especiales. Un ejemplo que ilustra la ubicación de ficheros mediante una FAT se muestra en la Figura 6.9.

Originalmente, las entradas de la FAT eran de 12 bits, pronto ampliadas a 16 bits (FAT16). Hay que tener en cuenta que este mecanismo se diseñó para las unidades de discos flexibles de los PCs, de capacidad muy limitada[11]. Cuando el uso de discos rígidos, de cada vez mayor capacidad, se hizo habitual en los ordenadores personales, aumentar el tamaño de bloque se convirtió en la solución de urgencia. Al

---

[10] *Desfragmentación*, en terminología Windows.

[11] Hacia 1980, Microsoft no sospechaba que los ordenadores personales terminarían incorporando unidades de discos rígidos...

poco de la aparición de Windows 95, los PCs montaban discos con tamaños prohibitivos para la FAT16, fundamentalmente por la fragmentación interna que inducían tamaños de bloque tan grandes[12]. La creación de varias particiones en el disco, cada una con su FAT propia, se convirtió en la única alternativa durante un tiempo[13]. Microsoft tuvo que plantearse ampliar el tamaño de las entradas de la FAT a 32 bits (FAT32)[14], lo que requiere un mayor espacio de memoria para ubicar la FAT, que se infrautiliza[15].Los sistemas más modernos (Windows NT) soportan FAT32. Con Windows NT Microsoft introdujo un sistema de ficheros específico (NTFS), de características muy diferentes a los basados en FAT y que no vamos a estudiar aquí[16]. Con todo, y pese a la práctica desaparición de los discos flexibles, la FAT16 se adoptó como sistema de ficheros para tarjetas tipo flash, repitiéndose la historia: el tamaño habitual de las tarjetas supera ya los 2 Gbytes y su progresión es aún más rápida que la que tuvieron los discos en su día.

6.5.7.2 UNIX

A diferencia de los sistemas FAT, UNIX almacena la información de ubicación independientemente para cada fichero.

Como ya vimos, un directorio de UNIX es un fichero especial que contiene para cada entrada únicamente el nombre del fichero y un número de i-nodo, estructura de datos que se almacena en disco y que contiene la descripción del fichero e información acerca de su ubicación.

Básicamente, la ubicación en UNIX es indexada. Los bloques de información se alcanzan bien a través de un apuntador directo en el i-nodo, bien a través de uno o más niveles de indirección en los bloques de índices, que presentan una estructura en árbol, como muestra la Figura 6.10.

---

[12] La FAT16 admitía bloques de hasta 32 Kbytes, permitiendo direccionar 2 Gbytes.

[13] Pero hay que tener en cuenta que cada partición constituye un sistema de ficheros independiente, y que Windows, al contrario que UNIX, no proporciona independencia entre sistema de ficheros y dispositivo físico. Es decir, a efectos del sistema operativo (y por lo tanto de las aplicaciones), cada partición se ve como un disco.

[14] Obsérvese que la modificación afecta también a la estructura del directorio, por lo que los sistemas FAT16 y FAT32 son incompatibles.

[15] Como, por su tamaño, resulta impensable almacenarla entera en memoria, se hace muy importante que los bloques de un fichero estén ubicados en posiciones contiguas o muy cercanas (localidad espacial), por lo que a veces se requiere compactar el disco.

[16] Básicamente, NTFS se basa en una tabla de registros, cada uno de 1 Kbyte de tamaño. Cada registro puede representar un directorio o un fichero. Si el fichero (o directorio) es pequeño, todo el contenido se almacena en el registro. En caso contrario se utilizan apuntadores a otros registros.

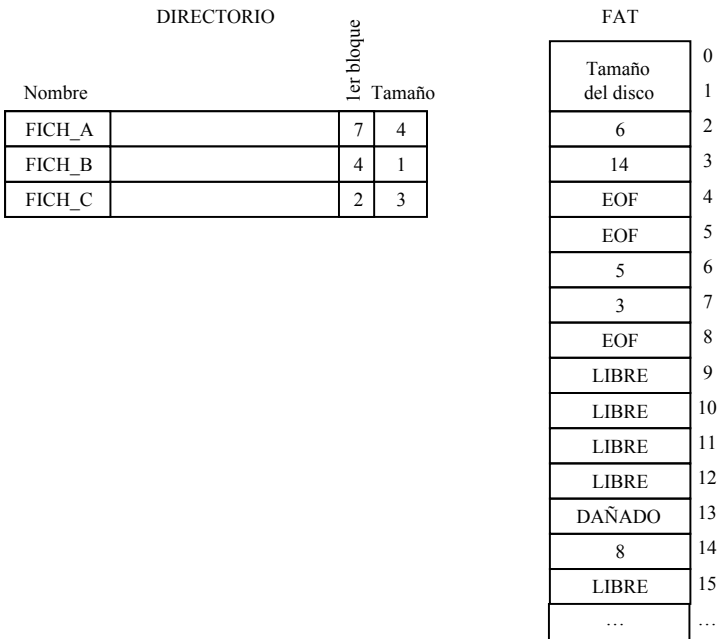

Figura 6.9. Un ejemplo de ubicación mediante FAT

En lo que respecta a información para la ubicación, el i-nodo contiene:

- 10 apuntadores directos a los 10 primeros bloques del fichero
- 1 apuntador a un bloque de apuntadores a bloques de datos (un nivel de indirección)
- 1 apuntador de dos niveles de indirección
- 1 apuntador de tres niveles de indirección

El tamaño de bloque se define en la instalación, y suele ser de 512, 1024, 2048, 4096 bytes o mayor. Un apuntador ocupa 4 bytes[17].

Los ficheros de hasta 10 bloques pueden ser ubicados utilizando exclusivamente apuntadores directos. Ficheros muy grandes tendrán bloques apuntados indirectamente cuyo acceso será más lento. Sin embargo, los accesos a los primeros bloques de un fichero son estadísticamente más frecuentes, por lo que este mecanismo puede ofrecer un buen rendimiento incluso para ficheros grandes.

El hecho de que el mecanismo limite el número de índices para un fichero determina el espacio direccionable del fichero. Por ejemplo para bloques de 1 Kbyte (256 índices por bloque), el tamaño máximo de fichero será del orden de $256^3 \cdot 1$ Kbyte, es decir, 16 Gbyte.

[17] En los primeros UNIX (1969) los apuntadores eran de 16 bits.

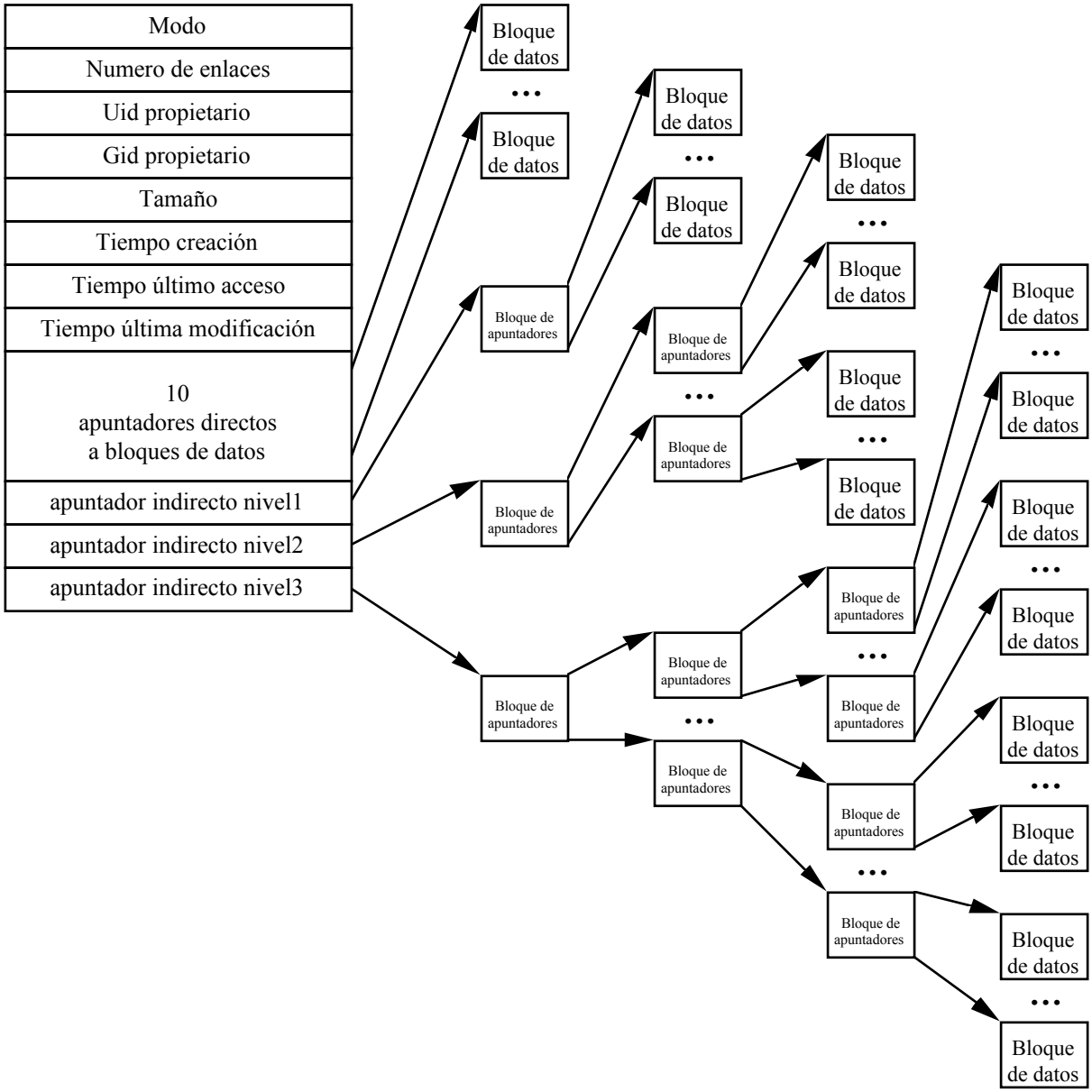

Figura 6.10. Ubicación en UNIX

La información acerca del espacio libre se almacena en una zona específica del disco, denominada *superbloque*, que contiene la tabla de apuntadores a i-nodos y de bloques libres. El superbloque contiene también el *booting* del sistema.

Aunque todos siguen el mismo esquema de ubicación, existen diferentes sistemas de ficheros UNIX que difieren entre sí en detalles acerca de la estructura del i-nodo y en la implementación, como *s5fs* del System V, el *FFS* (*Fast File System*) del 4.2BSD, o los sistemas *ext2* y *ext3* de Linux. Los sistemas actuales incluyen una capa de abstracción en el acceso al sistema de ficheros (*Sistema de Ficheros Virtual*, *VFS*) que proporciona independencia entre el sistema operativo y el sistema de ficheros y permite montar diferentes sistemas de ficheros bajo el mismo directorio común de UNIX. Básicamente, ahora las llamadas al sistema acceden a *v-nodes* o nodos virtuales que

representan cada fichero. Un v-nodo contiene la referencia al i-nodo del sistema de ficheros UNIX correspondiente, o puede hacer referencia a un sistema de representación muy diferente, como por ejemplo a un fichero de una partición FAT, o incluso a un sistema de ficheros remoto[18].

6.6 Cache de bloques en memoria

Como ocurre en otros niveles de la jerarquía de memoria, es posible plantear un almacenamiento de acceso más rápido para un subconjunto de los bloques del disco. El concepto es análogo al de la memoria cache para acelerar el acceso a la memoria principal. En el sistema de ficheros la cache se implementa en memoria principal, en el espacio de direcciones del sistema operativo, como un conjunto de buffers donde se almacena una parte de los bloques de disco referenciados, lo que permite transformar un acceso a disco en un acceso a memoria.

La gestión de este mecanismo reproduce los problemas y las soluciones que aparecen en niveles inferiores de la jerarquía de memoria. Aquí se pueden aplicar algoritmos de reemplazo (de bloques) semejantes a los utilizados en memorias cache y en memoria virtual. Asimismo, es necesario evitar los problemas de inconsistencia, como veremos más adelante.

El mecanismo de memoria cache de bloques se combina con la utilización de buffers de entrada/salida estudiada en el capítulo anterior. La cache de bloques se implementa como un conjunto de buffers de entrada/salida, que se gestionan de acuerdo a una política específica.

6.7 Gestión de errores

El sistema de ficheros está particularmente expuesto a errores en relación con otras partes del sistema operativo. Como se estudió en el capítulo de gestión de dispositivos, los discos, por su naturaleza mecánica, son objeto de frecuentes situaciones de error. Los dispositivos de hoy en día tratan algunas situaciones de error, pero en general es responsabilidad del manejador del disco gestionar la recuperación de una operación ante errores, mediante reintento. Si el error persiste, el manejador terminará informando a la rutina de E/S y eventualmente se propagará hasta la llamada al sistema, que devolverá el código correspondiente.

Los problemas específicos del sistema de ficheros se refieren a posibles situaciones de **inconsistencia** del sistema de ficheros, debido a la cache de bloques. En una

[18] Los sistemas de ficheros en red, como NFS (*Network File System*) se acceden de esta forma. De hecho Sun Microsystems desarrolló originalmente VFS para soportar NFS.

operación de escritura, la modificación de un bloque no se refleja inmediatamente en el disco. Si se produce una finalización anómala del sistema operativo, el sistema de ficheros podría contener puntos de inconsistencia. Nótese que la situación de inconsistencia puede afectar incluso a directorios o a i-nodos.

Los sistemas operativos proporcionan mecanismos para evitar y detectar inconsistencias en el sistema de ficheros. Por ejemplo, UNIX proporciona una llamada al sistema, *sync*, que se llama normalmente cuando se cancela el sistema (*shutdown*) para vaciar los buffers y evitar inconsistencias.

Las técnicas de detección de inconsistencias[19] se basan en recorrer el sistema de ficheros (1) desde la información de los directorios y (2) desde la información de bloques libres, anotando en cada caso el número de veces que se alcanza cada bloque. El sistema de ficheros será consistente si cada bloque ocupado se accede una única vez desde los directorios, y si todos los bloques no ocupados aparecen registrados como libres. Las situaciones de inconsistencia se reflejan como (a) bloques que no se reclaman ni como libres ni como ocupados; (b) bloques que se reclaman simultáneamente como libres y ocupados, y (c) bloques que se reclaman más de una vez como ocupados. El caso (c) es el más grave, ya que no es posible tomar automáticamente una decisión acerca de a qué fichero corresponde el bloque.

6.8 Bibliografía

[SIL08] es quizás el más cercano a nuestro enfoque. Dedica el capítulo 11 a describir la implementación de los sistemas de ficheros. [MIL94] incluye pseudo-algoritmos para describir las operaciones sobre ficheros. Los sistemas de ficheros de UNIX/Linux y Windows pueden encontrarse en los textos generales ([STA05], [TAN08] y [SIL08]) y en los específicos, por ejemplo, [BAC86] (capítulo 4) para la implementación del sistema de ficheros en UNIX y [SOL00] para NTFS (capítulo 12).

6.9 Ejercicios

1 Explicar cómo se implementa una llamada al sistema del tipo *posicionar (canal, registro lógico)*. Indicar a qué estructuras accede y qué información modifica.

2 La capacidad de almacenamiento de los discos ha aumentado enormemente en los últimos años, a un ritmo superior a la velocidad de transferencia. Discutir qué efectos tiene este hecho sobre la configuración del tamaño de bloque.

---

[19] En UNIX la detección de inconsistencias se activa mediante el comando *fsck*.

3 Considerar tres sistemas de ficheros, el primero con ubicación contigua, el segundo encadenada y el tercero indexada. Los tres usan bloques de 1024 bytes e índices de 16 bits. Una aplicación maneja ficheros con registros lógicos de 256 caracteres a los que accede directamente. Cada fichero tiene una entrada al directorio que indica el nombre del fichero, el primer bloque (o el primer bloque índice), la longitud del fichero y la posición del último bloque. Supondremos el directorio cargado en memoria. Para cada uno de estos sistemas:

(a) Explicar cómo se realizara la traducción de bloque relativo a bloque absoluto. (Para ubicación indexada, la entrada al directorio apunta al primer bloque índice, que a su vez apunta a 511 bloques de fichero y contiene un puntero al siguiente bloque índice.)

(b) Si acabamos de acceder al registro lógico 40 del fichero y queremos acceder al registro lógico 15, ¿cuántos bloques debemos leer en el disco?

4 El sistema FAT16 no permite tamaños de bloque mayores de 32 Kbytes. (a) Calcular el tamaño máximo direccionable en el dispositivo de almacenamiento. (b) Con bloques de ese tamaño ¿qué tamaño permite el sistema FAT32? (c) Para un disco de 80 Gbytes, con bloques de 4 Kbytes ¿cuánto espacio ocupa la FAT32?

5 Considerar una instalación con un sistema de ficheros con un tamaño de bloque de 32 Kbytes. De los 10.000 ficheros que almacena, un 40% tiene menos de 1024 bytes de longitud, con un tamaño medio de 200 bytes; otro 40% de los ficheros tiene un tamaño entre 1 Kbytes y 16 Kbytes, con una media de 5 Kbytes, y el 20% restante es mayor de 16 Kbytes, con un tamaño medio de 200 Kbytes. (a) Hacer una estimación de la fragmentación interna total de la instalación. (b) Si reducimos el tamaño de bloque a la mitad, ¿cuál es la nueva estimación de la fragmentación interna?

6 Considerar la siguiente secuencia de código:

```
...
fp= abrir("fich1", LECTURA);      /* Abre fich1 para lectura */
posicionar(fp, 2400);     /* Posiciona el puntero de lectura en el byte 2400 */
leer(fp, &buffer, 1000);          /* Lee 1000 caracteres */
...
```

que se ejecuta sobre un sistema operativo con un mecanismo de ubicación de ficheros FAT 16, con bloques de 1024 bytes, y que tiene el estado que se muestra en la Figura 6.11.

(a) ¿En qué bloque(s) relativo(s) del fichero están los 1000 caracteres leídos?

(b) ¿Y en qué bloque(s) absoluto(s) del sistema de ficheros?

DIRECTORIO

Nombre	...	Long.(bytes)	1er bloque
fich1	…	4000	13
fich2	…	205	11
fich3	...	9200	15
…			…

FAT

10	...
11	EOF
12	7
13	14
14	12
15	LIBRE
16	…

Figura 6.10. Estado del sistema de ficheros para el Ejercicio 6.

(c) ¿Cuántas peticiones tendrá que poner la rutina de E/S al gestor del disco para leer los 1000 caracteres, suponiendo que la FAT está en memoria y que los bloques del disco no están en ningún buffer?

(d) ¿Cuál es la fragmentación interna producida en el disco por fich1 y por fich2?

(e) ¿Qué ocurre con fich3? Interpreta cuál puede ser la causa de su situación.

(f) El disco donde se ubica este sistema de ficheros tiene 40 cilindros de 2 pistas de 8 sectores de 512 bytes. Dibujando un esquema, calcular los parámetros (cilindro, pista sector) de todos los accesos a disco que hará el driver del dispositivo para tratar cada una de las peticiones a las que se refiere el apartado c.

(g) Considerar ahora que estos ficheros los representamos en un sistema de ficheros como el de UNIX y que reducimos el tamaño de bloque a la mitad (un sector). Representar la información del directorio y los inodos (sólo los campos relevantes) correspondientes a fich1 y fich2. Considerar que el contenido de los ficheros permanece en los mismos sectores del disco FAT original.

(h) Calcular el espacio de direccionamiento de este sistema de ficheros UNIX, con apuntadores de 16 bits. (Independientemente del tamaño del disco).

(i) En las mismas condiciones, calcular el tamaño máximo de un fichero en este sistema UNIX.

7 Considerar un sistema de ficheros UNIX típico con inodos que ocupan 128 bytes. Los índices son de 32 bits y el tamaño de bloque se ha configurado a 4 Kbytes.

(a) Calcular el tamaño máximo de disco que este sistema es capaz de direccionar con dicha configuración.

(b) Calcular el tamaño máximo de fichero que admite.

(c) Si una aplicación se posiciona en el byte 50.000 de un fichero *fich.dat* y lee 100 caracteres, (c1) ¿en qué bloque(s) relativo(s) del fichero *fich.dat* están los 100 caracteres leídos; (c2) ¿cuántos bloques hay que leer para que la rutina de E/S obtenga esos 100 caracteres? (considerar que el inodo ya está en memoria).

(d) Suponiendo que el bloque del fichero *fich.dat* obtenido en el apartado anterior está asociado al bloque absoluto 40 de un disco con 800 cilindros de 10 pistas de 200 sectores de 512 bytes, razona en qué sectores de qué pista de que cilindro se ubica dicho bloque absoluto. Nota: todos los parámetros se numeran a partir de 0.

(e) Estamos diseñando un nuevo sistema de ficheros tipo UNIX para aumentar sus prestaciones. Hemos pensado aumentar el tamaño del inodo al doble (256 bytes). Los nuevos 128 bytes los dedicaremos a ubicar apuntadores directos a bloques del fichero. (e1) Calcular, en tanto por ciento, el aumento que conseguimos en el máximo tamaño de fichero representable. (e2) Considerando en general el rendimiento en el acceso al sistema de ficheros, ¿resulta interesante esta modificación?