Tema 3

Gestión de dispositivos

La gestión de los dispositivos de E/S es una de las funciones principales del sistema operativo, debido a que proporciona un medio para que éstos sean manipulados uniformemente mediante un conjunto de órdenes de alto nivel. En este capítulo veremos los principales aspectos del diseño de esta interfaz entre los dispositivos de E/S y los usuarios, prestando especial atención al rendimiento de las operaciones de E/S.

3.1. Introducción

El sistema de E/S es el elemento del sistema operativo que permite el intercambio de información entre éste y el exterior. En su diseño se persiguen dos objetivos fundamentales, **rendimiento** y **uniformidad**.

Debido a que la mayor parte de los dispositivos de E/S son muy lentos en comparación con la memoria principal y el procesador, pueden constituir un cuello de botella. Por este motivo, uno de los objetivos principales del sistema de E/S es obtener un buen rendimiento de los dispositivos.

Dada la diversidad de dispositivos de E/S existentes, un objetivo importante es realizar un tratamiento uniforme de ellos, tanto desde el punto de vista del usuario como de la gestión por parte del sistema operativo. Este objetivo es difícil de alcanzar

en la práctica debido a las diferentes características que presentan los dispositivos.

3.2. Dispositivos de E/S

Los dispositivos de E/S son el medio de comunicación entre el procesador y el exterior, ya que permiten transferir información en ambos sentidos. Existe una gran variedad de dispositivos de E/S. Así, tenemos dispositivos de almacenamiento (discos, cintas), de transmisión (*modems*, tarjetas de red), e interfaces con los usuarios (pantallas, teclados, ratones), cada uno con sus propias características. Sin embargo, éstos se pueden clasificar según dos criterios. Por un lado, si tenemos en cuenta el número de bytes transferidos en una operación de E/S individual podemos distinguir:

Dispositivos de bloques Almacenan la información en bloques de tamaño fijo, cada uno con su propia dirección. Su característica principal es la posibilidad de leer o escribir bloques de forma independiente. Ejemplos de éstos son los discos, CD-ROM, etc.

Dispositivos de caracteres Envían o reciben un flujo de caracteres sin estructura. Al contrario que los dispositivos de bloques, no son direccionables por lo que no se puede acceder directamente a un determinado carácter. Ejemplos de este tipo de dispositivos son las impresoras, terminales, cintas, etc.

También podemos clasificarlos en función de su propósito; en este caso, podemos distinguir:

Dispositivos de almacenamiento Se utilizan para almacenar información de forma permanente. Como ejemplo podemos citar los discos, cintas, etc.

Dispositivos de comunicación Se emplean para la comunicación entre los usuarios y el sistema de computación o entre diferentes sistemas. Ejemplos de estos son los *modems*, terminales, impresoras, etc.

Aunque la primera clasificación no es perfecta, es la más empleada. En ella, algunos dispositivos como los relojes, no están contemplados. Sin embargo, este modelo de clasificación es bastante general y se puede utilizar como base para el software del sistema operativo.

3.2.1. Controladoras de dispositivos

Los dispositivos de E/S, por lo general, constan de un componente mecánico y uno electrónico. A menudo es posible separar estos dos componentes para ofrecer un diseño más modular y general. El primero es el dispositivo propiamente dicho, y el segundo se denomina **controladora**.

Es necesario establecer esta distinción debido a que las controladoras actúan como intermediarias entre el sistema y los dispositivos de E/S. Su propósito es superar las incompatibilidades de velocidad, señalización de niveles entre el procesador y periféricos, traducir las órdenes de E/S genéricas emitidas por la CPU a controles específicos del dispositivo, codificación, etc. Es decir, la controladora se entiende con la CPU a alto nivel, mientras que la comunicación que establece con el periférico es de muy bajo nivel, adaptándose a las características de éste.

En la figura 3.1 se representa un diagrama general de una controladora de E/S, donde podemos observar que se divide en tres capas funcionales:

- Interfaz con el bus.
- Controladora de dispositivo genérico.
- Interfaz del dispositivo.

Las dos capas que actúan de interfaz son de gran importancia para los diseñadores de hardware. La capa intermedia —la controladora de dispositivo genérico— es la más importante para los diseñadores de software. Ésta hace que cada dispositivo tenga la apariencia de un conjunto de registros dedicados, que se conocen como **puerto de E/S**. Éstos disponen normalmente de tres tipos de registros:

1. Registros de datos.
2. Registros de estado.
3. Registros de órdenes.

Los registros de datos forman un *buffer* hardware, almacenando temporalmente los datos hasta que la CPU o el dispositivo pueda recibirlos. El sistema operativo efectúa las operaciones de E/S escribiendo las órdenes oportunas en los registros de órdenes. El dispositivo indica su estado a la CPU mediante los registros de estado. Éstos contienen información sobre el dispositivo (preparado u ocupado), la memoria intermedia (vacía o llena) e indicaciones de error.

Para poder efectuar las operaciones de E/S la CPU ha de poder direccionar los registros. Tradicionalmente, se incorporan una serie de instrucciones específicas para las operaciones de E/S, en las cuales se referencia cada registro mediante un identificador único. De esta forma los registros de la controladora forman un espacio de direcciones diferente al de la memoria principal. En este caso, se dice que la E/S está **mapeada en E/S** o es una **E/S aislada**, y el conjunto de instrucciones de E/S recibe el nombre de **instrucciones de E/S directa**. Ejemplos de este tipo de instrucciones son las siguientes:

input *dirección_ dispositivo*

output *dirección_ dispositivo*

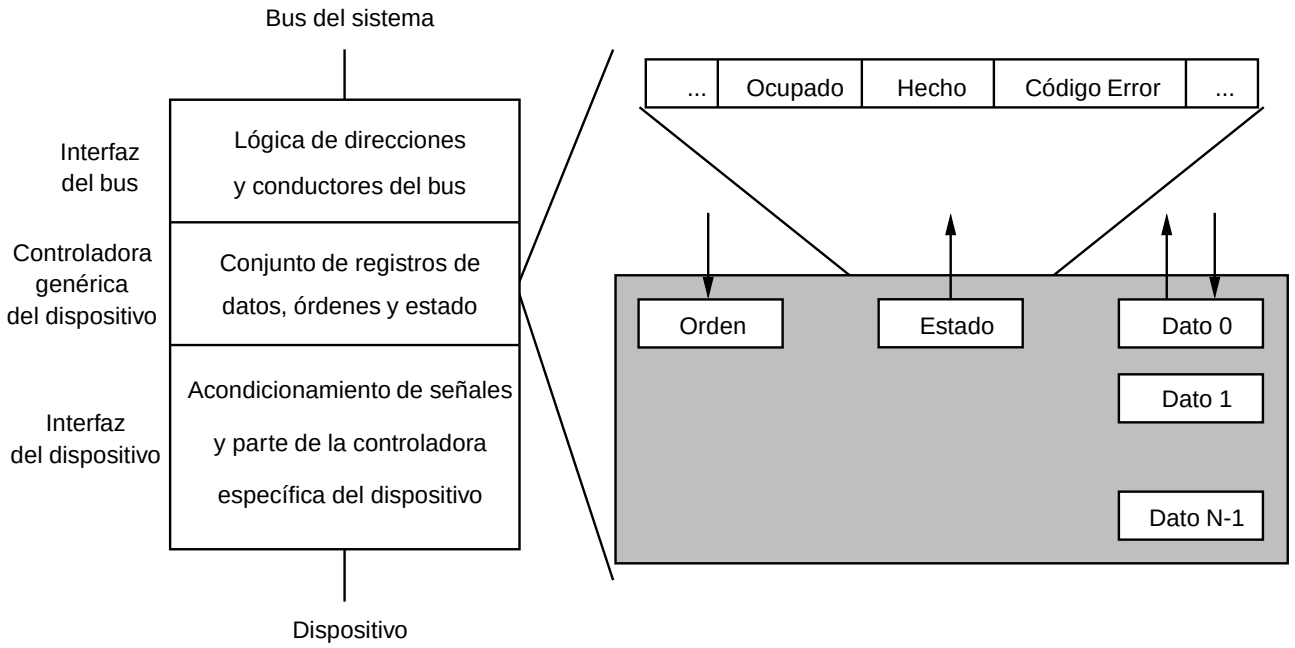

Figura 3.1: Estructura de una controladora de E/S

copy_in *registro_CPU*, *dirección_dispositivo*, *registro_controladora*

copy_out *registro_CPU*, *dirección_dispositivo*, *registro_controladora*

test *registro_CPU*, *dirección_dispositivo*

Otra alternativa es asociar a los registros una dirección de memoria, en lugar de un identificador específico. De esta forma, las direcciones de los registros forman parte del espacio de direcciones. Este esquema recibe el nombre de **E/S mapeada en memoria**. Así, por ejemplo, a un dispositivo se le podría asignar el espacio de direcciones desde `0xFFFF0120` hasta `0xFFFF012F` para referenciar sus registros. Una instrucción de E/S mapeada en memoria podría ser la siguiente:

```
store R3, 0xFFFF0124
```

La E/S mapeada en memoria reduce el número de instrucciones de la CPU ya que las propias instrucciones de acceso a la memoria pueden ser empleadas para realizar las operaciones con los registros. Sin embargo, la E/S aislada necesita incorporar instrucciones específicas para manipularlos. La figura 3.2 muestra las dos alternativas para referenciar los registros de la controladora.

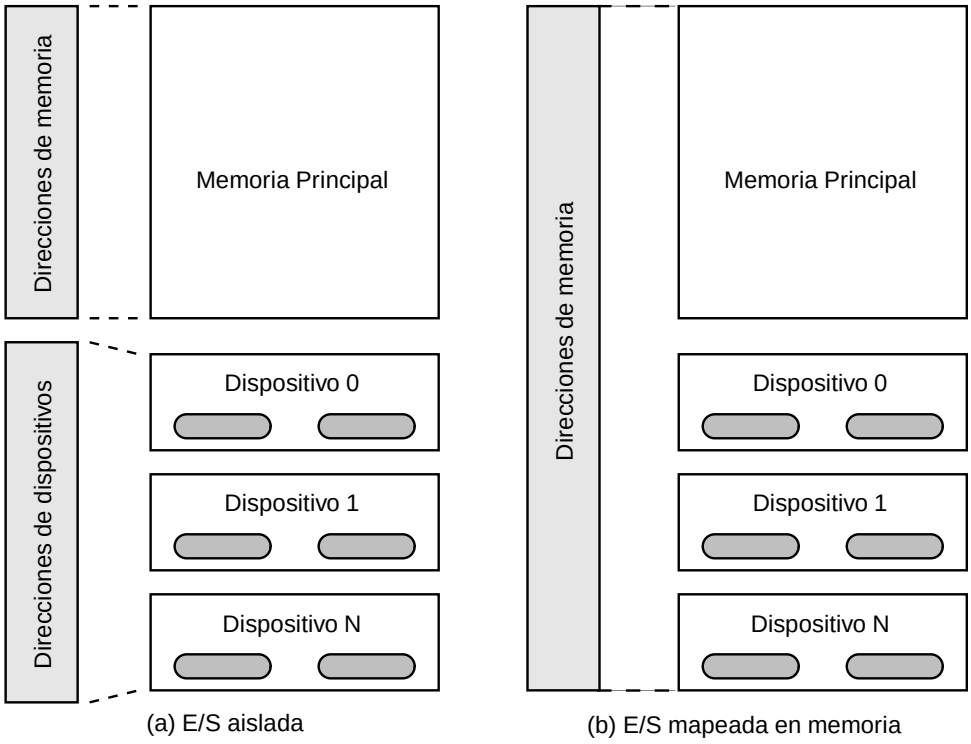

Figura 3.2: Direccionamiento de los registros de las controladoras

3.3. Organización del sistema de E/S

Para lograr los objetivos de uniformidad y buen rendimiento, el sistema de E/S se estructura en capas, de forma que los niveles inferiores ocultan las particularidades del hardware a los procesos y a los niveles superiores del software de E/S.

Una posible organización es la que aparece en la figura 3.3 que muestra los distintos componentes hardware y software que intervienen en las operaciones de E/S. Situándonos en el nivel más bajo, cada dispositivo utiliza una controladora para conectarse a los buses de datos y direcciones del ordenador. La capa de software que está en contacto directo con las controladoras de dispositivos, está formada por un conjunto de manejadores de dispositivos que se encargan de tratar las particularidades de éstos. Por encima de ésta se implementa otra que presenta una interfaz uniforme de todos los dispositivos a los usuarios y programadores del sistema. Esta capa se denomina **subsistema de E/S** e implementa una interfaz de programación de aplicaciones abstracta (API).

Es decir, el subsistema de E/S se encarga de conseguir el objetivo de la uniformidad proporcionando las funciones disponibles en los dispositivos mediante la API, mientras que los manejadores de dispositivos implementan estas funciones. El objetivo de obtener un buen rendimiento es compartido por ambas capas.

Un problema que plantea la división del sistema de E/S en dos capas es que los manejadores de dispositivos, además de ser específicos para el dispositivo, deben respetar la API del sistema operativo. Como cada sistema operativo tiene su propia interfaz, los fabricantes de dispositivos tienen que proporcionar un manejador para cada sistema operativo.

Un aspecto importante en el diseño de un manejador es que necesita ejecutar instrucciones privilegiadas, ya que debe leer y escribir información en el espacio de los procesos de usuario y manipular ciertas estructuras del sistema operativo, como la tabla de estado de los dispositivos. En sistemas antiguos, el manejador se incorporaba en el núcleo. Por lo que añadir un nuevo dispositivo implicaba modificar el código fuente para introducir su manejador y recompilarlo. Esto era aceptable cuando el propio vendedor de hardware lo instalaba.

En los sistemas modernos se simplifica esta instalación utilizando manejadores de dispositivos reconfigurables, que permiten añadir un nuevo manejador al sistema sin necesidad de volver a compilar. En estos casos, las funciones de los manejadores de dispositivos se enlazan de forma dinámica al código del núcleo. La figura 3.4 muestra las estructuras necesarias en este proceso. El sistema operativo emplea una **tabla de referencias indirectas** para acceder a cada función del manejador del dispositivo. De esta forma, el sistema crea una API independiente del dispositivo. Cuando un proceso realiza una petición de un servicio para una operación de E/S, el núcleo la pasa al manejador del dispositivo correspondiente a través de la tabla de referencias indirectas. Cuando el manejador se instala, se actualiza la información de dicha tabla para que el sistema pueda efectuar las llamadas correspondientes.

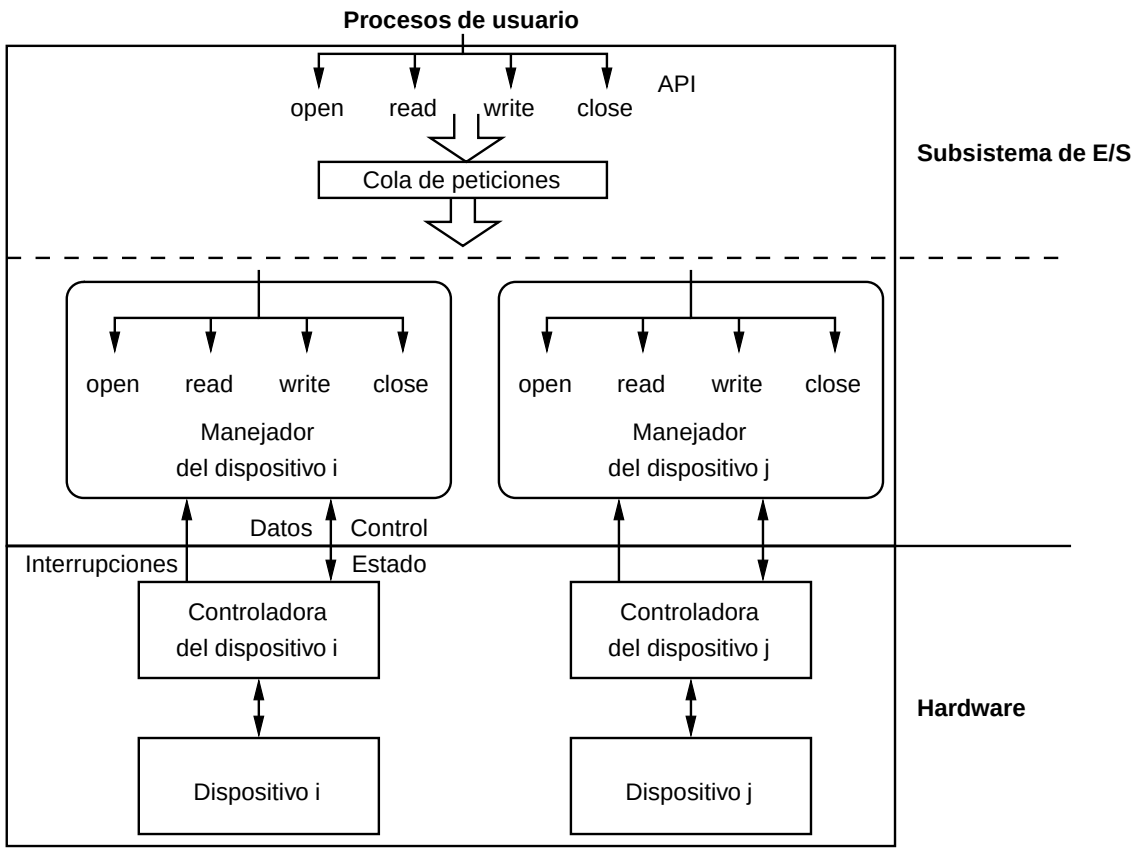

Figura 3.3: Estructura del sistema de E/S

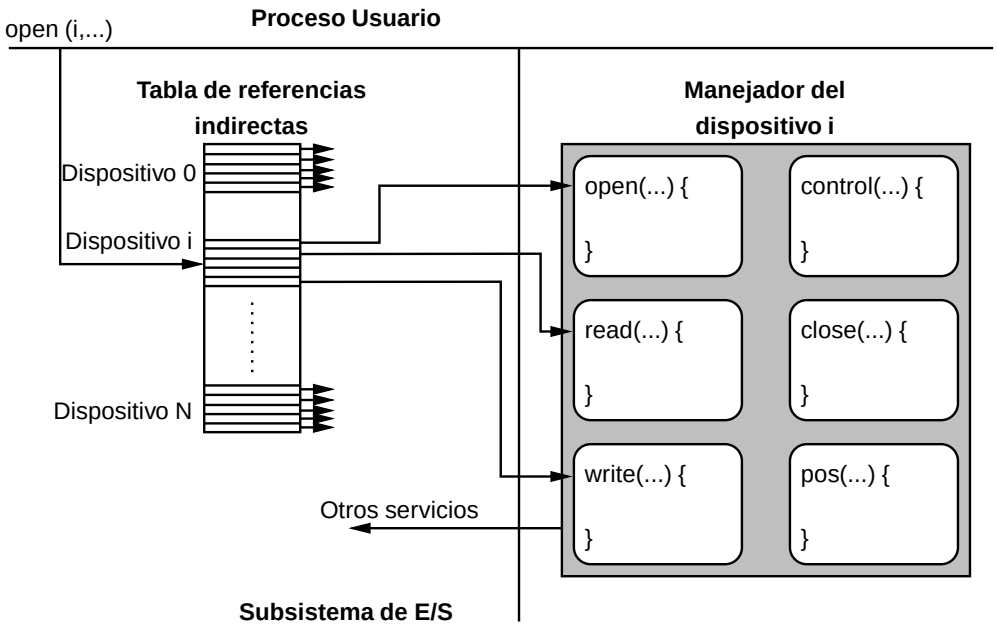

Figura 3.4: Manejadores de dispositivos reconfigurables

3.4. Modos de realizar las operaciones de E/S

Los manejadores de dispositivos, para realizar las operaciones de E/S, tienen que hacer uso del puerto de E/S. Dependiendo de las características del dispositivo, esto se puede llevar a cabo de tres formas diferentes:

- **E/S controlada por programa**.
- **E/S controlada por interrupciones**.
- **Acceso directo a memoria**.

3.4.1. E/S controlada por programa

Esta técnica también se conoce como **escrutación**. En ella, el procesador tiene un control completo sobre la operación de E/S, encargándose de transferir la información entre la memoria principal y los registros de la controladora.

La controladora no realiza ninguna acción para alertar al procesador, sino que es responsabilidad del sistema comprobar periódicamente el estado de la controladora hasta que el dispositivo haya completado la operación.

Para una operación de entrada los pasos a realizar son los siguientes (ver figura 3.5):

1. El proceso de usuario realiza una operación de lectura; a través del servicio correspondiente da la orden al manejador del dispositivo.

2. El manejador del dispositivo comprueba el registro de estado de la controladora para determinar si el dispositivo está libre. Si está ocupado, espera hasta que termine.

3. El manejador almacena una orden de entrada en el registro de órdenes de la controladora y, de esta forma, inicia el dispositivo.

4. El manejador comprueba continuamente el registro de estado de la controladora, esperando a que el dispositivo finalice su operación. Esta comprobación se denomina **escrutinio**.

5. El manejador copia el contenido de los registros de datos de la controladora en el espacio de memoria del proceso de usuario.

6. Se devuelve el control al proceso de usuario.

En el caso de realizar una operación de escritura, una vez determinado que el dispositivo está libre se copian los datos del espacio de usuario a los registros de la controladora y se inicia la operación.

Podemos ver que cuando se utiliza esta técnica el código del manejador del dispositivo incluye instrucciones para que el procesador tenga un control total sobre la operación de E/S. Entre éstas se incluyen la comprobación del estado de los dispositivos, el envío de órdenes de lectura o escritura y la transferencia de los datos.

3.4.2. E/S controlada por interrupciones

El problema que plantea la técnica anterior es que el procesador consume ciclos esperando la finalización de la operación de E/S, interrogando repetidamente el estado del dispositivo. Es decir, se encuentra en una espera activa. Como consecuencia, el rendimiento del sistema en conjunto se degrada considerablemente.

Con objeto de eliminar la necesidad de interrogar continuamente al dispositivo, una alternativa es el empleo de las interrupciones. En este caso, cuando el procesador da una orden de E/S a la controladora puede continuar con otro trabajo, ya que ésta le notificará mediante una interrupción del fin de la operación de E/S. El procesador, al igual que en la técnica anterior, ejecutará la transferencia de datos.

Con este esquema se necesitan otros elementos para la gestión del dispositivo, una **tabla de estado de los dispositivos**, que contiene una entrada por cada uno, y el manejador de interrupciones, encargado de determinar el tipo de interrupción y efectuar las acciones oportunas.

Los pasos a realizar para una operación de entrada son los siguientes (ver figura 3.6):

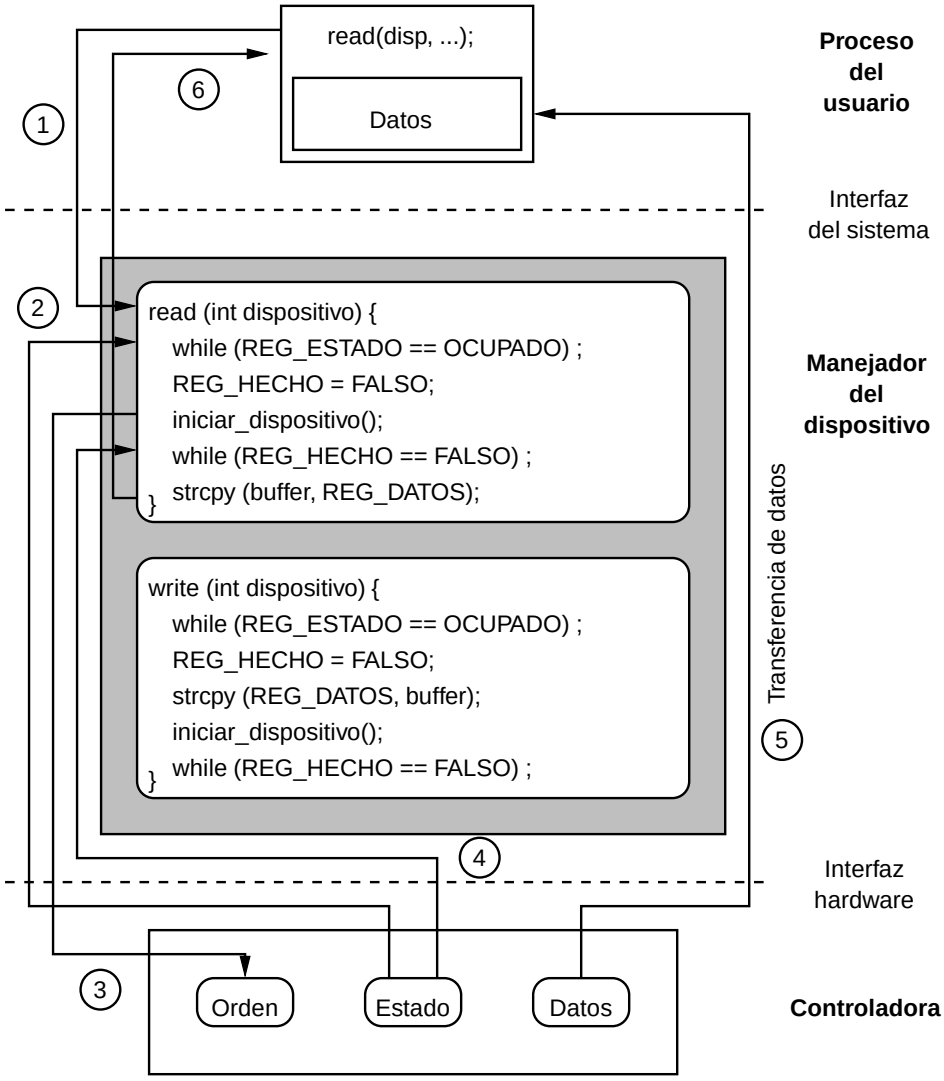

Figura 3.5: E/S controlada por programa

1. El proceso de usuario pide realizar una operación de lectura a través del servicio correspondiente.

2. El manejador del dispositivo comprueba el registro de estado de la controladora para determinar si el dispositivo está libre. Si está ocupado, espera a que termine.

3. El manejador almacena una orden de entrada en el registro de órdenes de la controladora y, de esta forma, inicia el dispositivo.

4. El manejador almacena la información de la operación iniciada en la tabla de estado de los dispositivos. En la entrada correspondiente se guarda la dirección de retorno de la llamada original del proceso, así como parámetros especiales para la operación de E/S. En este momento, el proceso pasa a estado bloqueado

y se llama al planificador a corto plazo para continuar con la ejecución de otro proceso.

5. Cuando el dispositivo termina la operación emite una interrupción, que provoca que el sistema operativo tome el control y se lo pase al manejador de interrupciones.

6. El manejador de interrupciones determina qué dispositivo causó la interrupción y le pasa el control a su manejador.

7. El manejador del dispositivo recupera de la tabla de estado de los dispositivos la información de la operación de E/S.

8. El manejador del dispositivo copia el contenido de los registros de datos de la controladora en el espacio del usuario.

9. El manejador del dispositivo devuelve el control al planificador para que desbloquee al proceso de usuario.

En este esquema, el manejador del dispositivo dispone de dos elementos. El primero son las rutinas que se encargan de traducir la orden de E/S a la controladora (pasos 3 y 4). El otro es la rutina a ejecutar cuando se produce la interrupción de finalización de la operación de E/S (pasos 7, 8 y 9).

Con este esquema el procesador no espera la terminación de la operación de E/S, pero sigue siendo el responsable de la transferencia de todos los datos entre el procesador y el dispositivo. Para entender mejor cómo funciona esto, veamos cómo se realiza una operación de lectura. En primer lugar, veamos lo que ocurre desde el punto de vista de la controladora. Para la entrada, el módulo de E/S recibe una orden `READ` del procesador. La controladora procede a leer los datos del periférico adecuado. Una vez que los datos están en el registro de datos de la controladora, ésta le envía una interrupción al procesador. La controladora espera hasta que los datos son pedidos por el procesador. Cuando se hace la petición, la controladora coloca los datos en el bus de datos y está listo para otra operación de E/S.

Desde el punto de vista del procesador, la acción es la siguiente. El procesador emite una orden `READ`. Guarda el contexto del programa actual, pasándolo a estado bloqueado, y empieza a ejecutar otro, es decir, realiza un cambio de proceso. Al final de cada ciclo de instrucción comprueba las interrupciones. Cuando se produce la interrupción que proviene de la controladora, guarda el contexto del proceso actual y empieza a ejecutar el manejador de interrupciones, que llama al manejador del dispositivo; el procesador lee la palabra de datos de la controladora de E/S y la almacena en memoria. En ese momento, el proceso bloqueado que esperaba la finalización de la operación de E/S pasaría a estado listo, para poder continuar su ejecución. Recupera el contexto del programa que emitió la orden de E/S (o de cualquier otro) y reanuda la ejecución.

Esta técnica de E/S permite aumentar el rendimiento del sistema, ya que permite ejecutar un proceso mientras otro está esperando una operación de E/S. Por tanto,

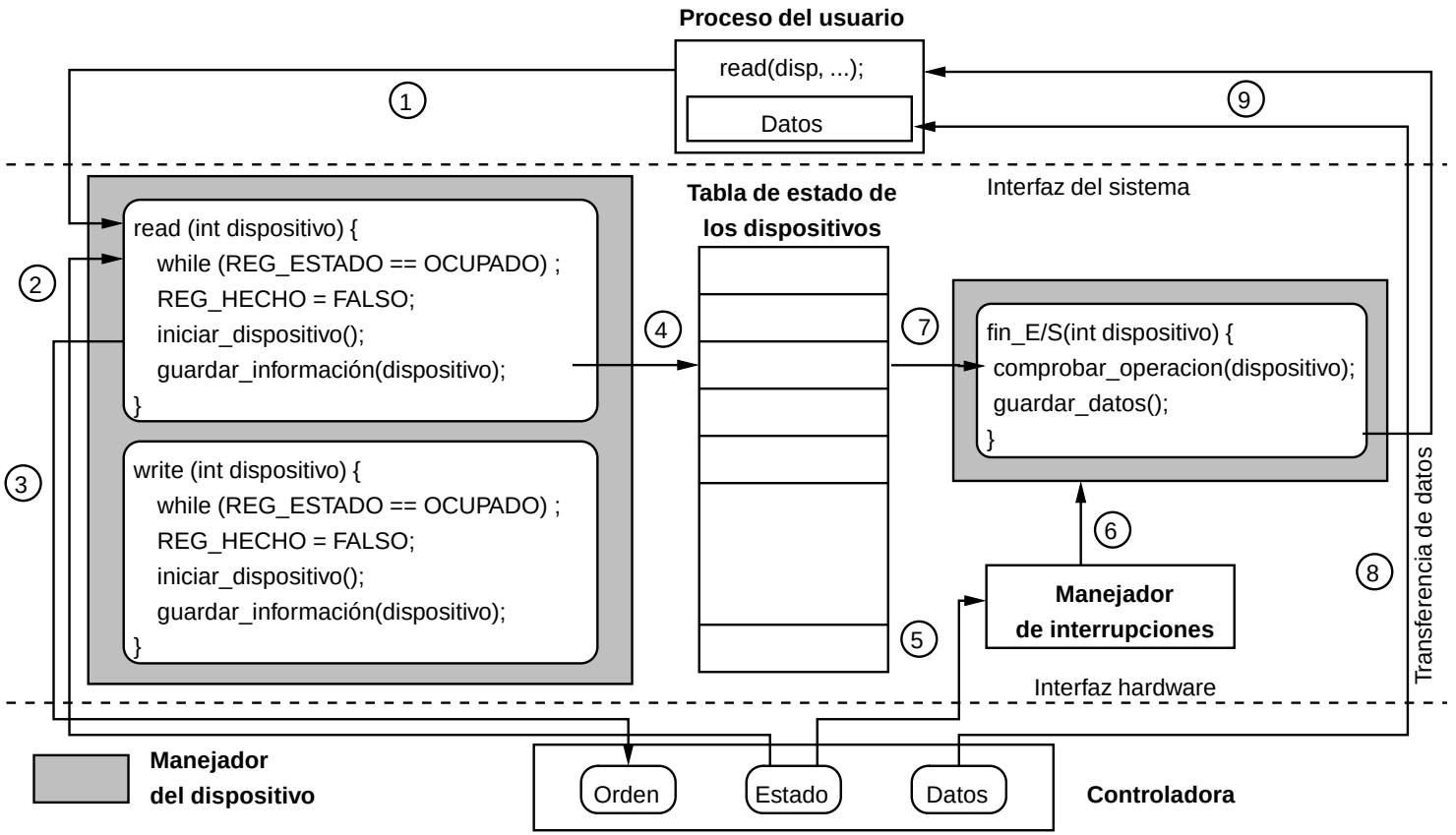

Figura 3.6: E/S controlada por interrupciones

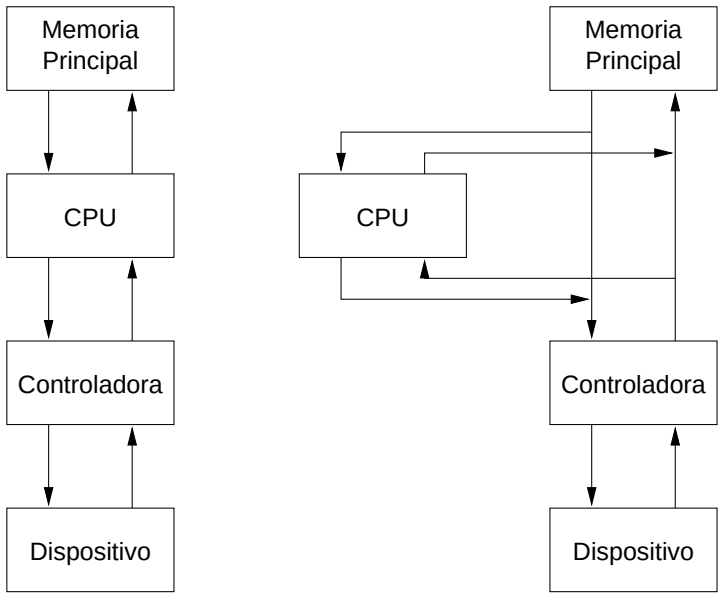

Figura 3.7: Modos de realizar las operaciones de E/S

la E/S controlada por interrupciones es más eficiente que la controlada por programa porque elimina las esperas innecesarias. Sin embargo, sigue consumiendo una gran cantidad de tiempo del procesador, ya que éste se encarga de transferir cada palabra de datos que va de la memoria al dispositivo o viceversa. Esto hace que el rendimiento del sistema disminuya cuando se tiene que transferir un gran volumen de datos.

3.4.3. Acceso directo a memoria

Una técnica más eficiente que las anteriores es el acceso directo a memoria o DMA[1]. Ésta permite que el dispositivo transfiera datos directamente a o desde la memoria, descargando al procesador de esta tarea, por lo que es ideal cuando se transfieren grandes volúmenes de datos.

La figura 3.7 muestra las diferencias entre las técnicas de E/S tradicionales (programada y por interrupciones) y el acceso directo a memoria.

Para poder utilizar esta técnica son necesarias controladoras especiales capaces de realizar las operaciones de E/S directamente en memoria. Éstas incorporan dos registros adicionales para almacenar la dirección del bloque de memoria implicado y el número de bytes a transferir en la operación.

Cuando se utiliza una controladora DMA, los pasos que se realizan para una operación de E/S son los siguientes:

[1] *Direct Memory Access.*

1. El proceso de usuario solicita realizar una operación de E/S a través del servicio correspondiente.

2. El manejador del dispositivo envía la orden a la controladora de DMA con la siguiente información:

   - Tipo de operación (lectura o escritura).
   - Dirección del dispositivo de E/S implicado.
   - Dirección de memoria donde hay que empezar a leer o escribir.
   - Número de bytes a transferir.

3. El proceso que solicita la operación de E/S pasa a estado bloqueado, y el planificador a corto plazo escoge otro proceso listo para continuar con su ejecución. En este caso, se delega la realización de la operación de E/S a la controladora DMA.

4. Dependiendo del tipo de operación, los datos a transferir se copiarán desde la memoria (salida) o el dispositivo (entrada) al *buffer* de la controladora.

5. A continuación la controladora transferirá los datos desde su *buffer* a su destino (dispositivo o memoria).

6. Cuando la transferencia se completa, la controladora produce una interrupción.

7. El procesador comprueba que la operación ha finalizado correctamente, y el proceso bloqueado pasa a estado listo para que continúe con su ejecución.

Podríamos pensar que la controladora podría realizar la transferencia de la información en un solo paso, en lugar de los pasos 4 y 5, sin necesidad del *buffer*. Sin embargo, la transferencia de cada palabra de información requiere que la controladora tome el control del bus, que es compartido con el resto de componentes del sistema. Dado que el dispositivo envía la información a una velocidad constante, es posible encontrarnos con una palabra para transferir a memoria y no disponer del control del bus en ese momento. Esto originaría que la controladora necesitase almacenar la información mientras espera el bus. Si éste estuviera muy ocupado, puede ser necesario almacenar bastante información. Por este motivo, la controladora, en lugar de disponer de un registro de datos, dispone de un *buffer*, con objeto de almacenar un bloque de información, y agilizar las transferencias a y desde la memoria, evitando la pérdida de datos.

3.5. Optimización de las operaciones de E/S

Dado que uno de los objetivos del sistema de E/S es la eficiencia en la realización de las operaciones de E/S, en este apartado se van a estudiar técnicas que permiten mejorar el rendimiento.

3.5.1. Técnicas de *buffering*

Un *buffer* es un área de memoria que almacena datos temporalmente. Su objetivo principal es suavizar las diferencias de velocidad entre los dispositivos y el procesador.

La técnica de *buffering*, vista en Sistemas Operativos I, se puede implementar tanto en el sistema de E/S como por hardware. De hecho, los registros de datos de las controladoras suelen estar duplicados para actuar como un *buffer*.

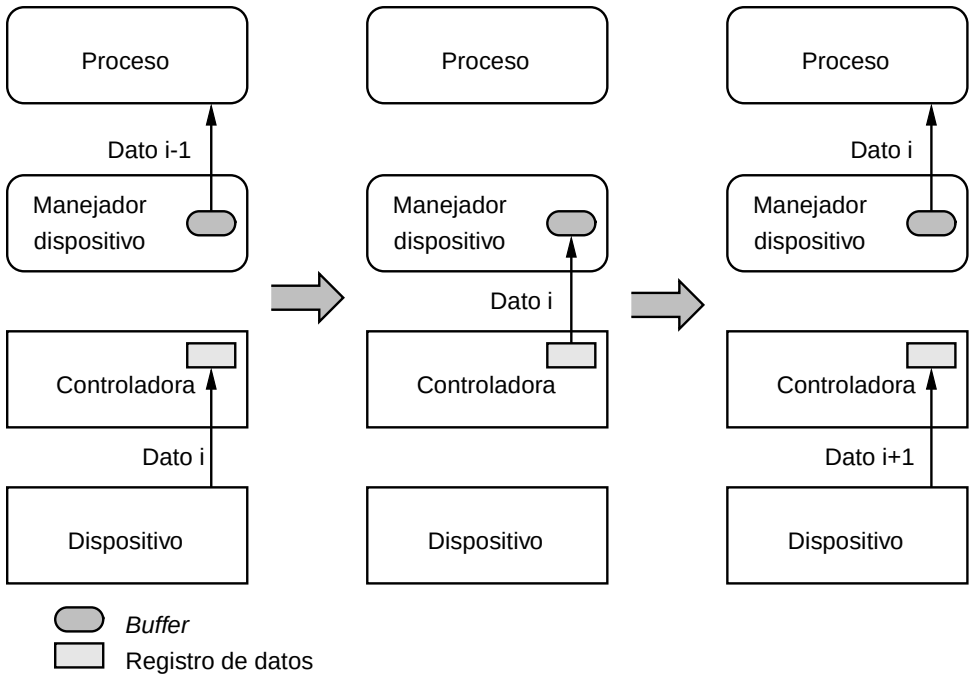

Figura 3.8: Operación de entrada con *buffer*

Esta técnica puede ser aplicada tanto a dispositivos de bloques como de caracteres. En ambos casos, el *buffer* debe tener el tamaño adecuado para almacenar un bloque o carácter, según el dispositivo.

El *buffering* permite reducir la cantidad de tiempo que el proceso espera para leer o escribir una palabra, como muestra la figura 3.8. En ella, mientras el dispositivo realiza la lectura del dato $i+1$, el procesador está trabajando con el dato i. En este caso, se consigue con la incorporación de un *buffer* en el manejador del dispositivo.

Se puede realizar una comparación de rendimiento entre un sistema que incorpora un *buffer* frente a otro que no lo posee. Sea T el tiempo para realizar la lectura de un dato, y C el tiempo de procesarlo.

Si no se emplea *buffer*, el tiempo de ejecución por dato es $T + C$. Sin embargo, con el empleo del *buffer* el tiempo viene dado por $max(T, C) + M$, siendo M el tiempo necesario para mover los datos desde el *buffer* del manejador al espacio de usuario, que suele ser menor que los anteriores.

La técnica del *buffering* se puede mejorar empleando dos *buffers*. De este modo, mientras un proceso está transfiriendo datos desde uno de ellos, la controladora puede estar empleando el otro. Esta técnica se conoce como **doble *buffer*** o **intercambio de *buffers***.

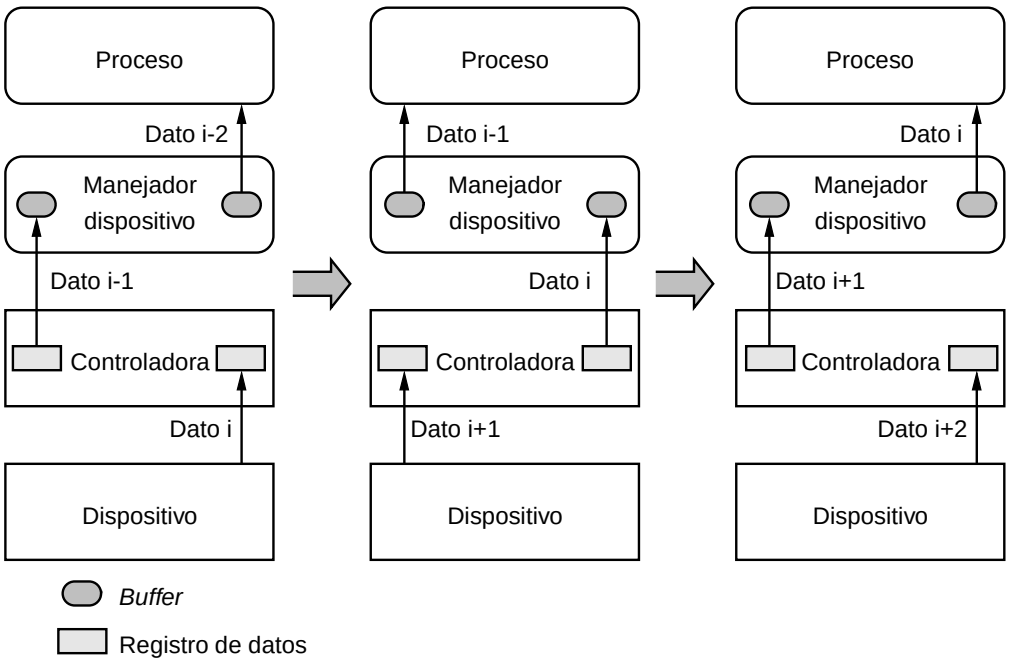

Figura 3.9: Doble *buffer*

Es posible mejorar esta técnica aumentando el número de *buffers* de 2 a N, recibiendo el nombre de ***buffer* circular**. La utilización del mismo se convierte en el problema de sincronización que se establece entre un proceso productor y otro consumidor. De esta forma, el proceso productor (la controladora en operaciones de lectura y el procesador en escritura) escribe en el *buffer* i, mientras que el proceso consumidor (controladora en escritura y procesador en lectura) lee del *buffer* j. La figura 3.10 muestra esta técnica. En ella, los *buffers* j hasta $n-1$ y desde 0 hasta $i-1$ están llenos, y pueden ser leídos por el consumidor, mientras el productor puede llenar los *buffers* i a $j-1$.

3.5.2. Caché de disco

Una caché de disco es una ampliación del concepto de *buffer* explicado anteriormente. La caché contiene datos de sectores del disco que, debido a que se encuentran en memoria principal, su acceso es más rápido que al dato original. El funcionamiento es el siguiente, cuando un proceso solicita una petición de E/S, se comprueba si el dato se encuentra en la caché, en cuyo caso no es necesario realizarla. En caso contrario, se inicia la operación almacenando el sector leído en la caché para futuras referencias.

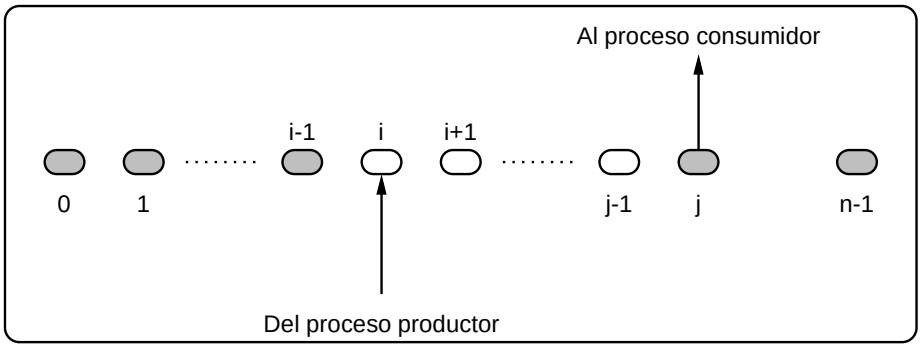

Figura 3.10: *Buffer* circular

La caché y el *buffer*, aunque son parecidos, presentan distintas funciones. Mientras el *buffer* mantiene datos de la operación en curso, la caché mantiene datos de las últimas operaciones realizadas.

En el diseño de esta caché es necesario tener en cuenta dos aspectos, el trasiego de información entre la caché y el proceso de usuario, y la política de sustitución. Respecto al primero, cuando se realiza una petición de E/S y los datos se encuentran en la caché, éstos deben ser entregados al proceso. En este caso, podemos copiar los datos de la caché al espacio de usuario, o emplear la zona de la caché como una memoria compartida, devolviendo al proceso un puntero a ella. Esta última posibilidad evita el trasiego de información dentro de la memoria, además de permitir compartir la información entre varios procesos.

El otro aspecto viene motivado por el tamaño limitado de la caché de disco. Esto origina que cuando se realiza una operación de entrada y la caché esté llena, se deberá sustituir un dato antiguo por el nuevo. Este problema es idéntico al de sustitución de páginas, y se pueden emplear los mismos algoritmos.

El algoritmo más utilizado es el menos recientemente usado (LRU), que sustituye el bloque que lleva más tiempo sin referenciarse en la caché. En este caso, la caché consiste en una pila de bloques, donde el más reciente se encuentra en la cima de la pila. Cada vez que un bloque de la caché se referencia, se sitúa en cima. Cuando se necesita sustituir un bloque de la caché, se escoge el situado en la parte inferior de la pila.

Otro algoritmo es el más frecuentemente usado (LFU), donde se sustituye el bloque que ha sido más veces referenciado. Para ello, cada bloque lleva un contador del número de referencias realizadas. Cuando un bloque se introduce por primera vez en la caché se le asigna un contador con valor 1. Cuando se necesite sustituir un bloque, se escoge aquél que posee un contador menor.

El problema del algoritmo LFU es que, debido a la localidad de referencias, en un breve espacio de tiempo se realizan continuas referencias al mismo bloque, lo que hace incrementar considerablemente el valor del contador. Una vez pasado ese intervalo, no se vuelve a referenciar, aunque su contador indique lo contrario. En

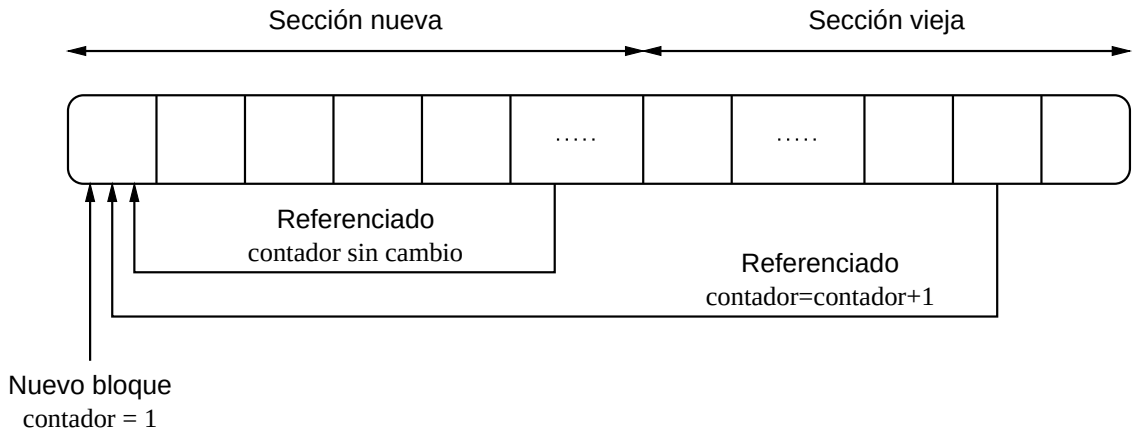

Figura 3.11: Algoritmo de sustitución basado en frecuencia

este caso, el efecto de localidad hace que el algoritmo LFU tenga un rendimiento pobre.

Para solucionar esta técnica se propone una modificación denominada **sustitución basada en frecuencia**. Para comprenderlo mejor, consideraremos una versión simplificada. En ella, los bloques se organizan lógicamente en una pila, al igual que en el algoritmo LRU. Una parte de la cima de la pila se considera como sección nueva. Cuando un bloque de la caché se referencia se sitúa al principio de la pila. Si el bloque estaba en la sección nueva, su contador no se incrementa; en caso contrario, se incrementa en uno. Cuando se necesita escoger a un nuevo proceso se escoge el bloque con el contador más pequeño de los que no estén situados en la sección nueva. La figura 3.11 muestra este esquema.

El refinamiento a este algoritmo consiste en dividir la pila en tres zonas: nueva, vieja y central. Los contadores no se incrementan si se encuentran en la zona nueva. Sin embargo, sólo los bloques de la zona vieja son los que pueden ser sustituidos. Este algoritmo se comporta mejor que LRU y LFU.

En algunos sistemas, como LINUX, la caché de disco se emplea también como *buffer*, es decir, las operaciones de E/S se realizan sobre un *buffer*, que formará parte de la caché de disco del propio sistema. Ésta recibe el nombre de **caché de *buffers***.

3.5.3. Planificación de discos

Uno de los dispositivos de uso más frecuente es el disco, de forma que el rendimiento del sistema se va a ver afectado por su velocidad de acceso. Aunque en los últimos años se han producido grandes avances en la velocidad de acceso a éstos, las diferencias con los procesadores y la memoria ha ido en aumento. Por este motivo, se ha prestado una gran atención a la forma de realizar las operaciones con estos dispositivos para mejorar su rendimiento.

3.5.3.1. Parámetros de rendimiento del disco

Todos los discos o CD-ROM se organizan en uno o más **platos**, cada uno con una o dos **superficies**. Cada superficie se divide lógicamente en varios **sectores** definidos como una porción angular del círculo del disco, similar a las porciones de una tarta. Cada superficie del disco se organiza en varias **pistas** concéntricas que pasan por cada sector. El conjunto de pistas en las diferentes superficies se denomina **cilindro**. La figura 3.12 muestra los distintos elementos de un disco. Cuando se realiza una petición de E/S, el sistema debe determinar en qué disco, superficie, pista y sector se encuentra, para poder posicionar la cabeza de lectura/escritura y realizar la transferencia.

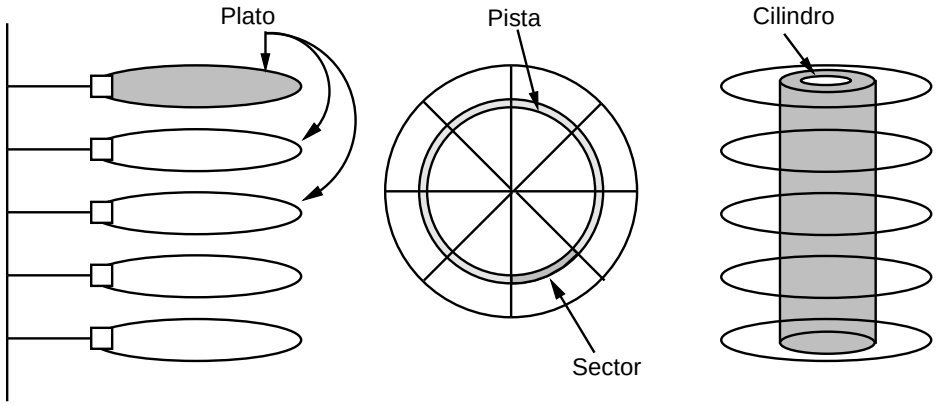

Figura 3.12: Componentes de un disco

Cuando se recibe una petición de E/S sobre un disco, el sistema debe posicionar la cabeza de lectura/escritura en la pista correspondiente. El tiempo que se tarda en realizarlo se conoce como **tiempo de búsqueda**. Éste se puede aproximar mediante la siguiente fórmula:

$$T_s = m \times n + s$$

donde T_s es el tiempo de búsqueda estimado, m es el tiempo que tarda en pasar de una pista a otra, que depende de la unidad de disco, n es el número de pistas que va a recorrer, y s es el tiempo de arranque.

Una vez que la pista está seleccionada, la controladora del disco espera hasta que el principio del sector se sitúa debajo de la cabeza. El tiempo que tarda en producirse esta operación se conoce como **demora rotacional** o **tiempo de latencia**. Éste depende de la velocidad de rotación del dispositivo, que varía en función del tipo de disco. Así, un disquete gira a 360 rpm, es decir, una revolución cada 167 milisegundos, lo que supone un retardo medio de 83 milisegundos. Las unidades de discos duros tienen una velocidad de rotación de 7200 rpm o superior, lo que supone una latencia media de 4 milisegundos. La suma del tiempo de búsqueda y de la demora rotacional

es el **tiempo de acceso**, es decir, el tiempo que tarda en situarse la cabeza en la posición adecuada donde tiene que leer o escribir.

Una vez que la cabeza está posicionada, la operación de lectura o escritura se realiza a medida que el sector se mueve bajo ésta. Al tiempo que se tarda en hacer la transferencia se denomina **tiempo de transferencia**, que viene dado por:

$$T = \frac{b}{r \times N}$$

donde T es el tiempo de transferencia, b el número de bytes a transferir, N el número de bytes por pistas, y r es la velocidad de rotación en revoluciones por segundo.

Por tanto, el **tiempo de acceso total** se puede expresar como la suma de los tres tiempos anteriores, que comprende buscar la pista, después el sector y, finalmente, transferir la información. Este tiempo se expresa como sigue:

$$T_a = T_s + \frac{1}{2} \times r + \frac{b}{r \times N}$$

De estos tres componentes, el tiempo de búsqueda es el que tiene más peso en el total. Dado que en un sistema de tiempo compartido es habitual que haya varias peticiones pendientes de E/S al disco, éstas se podrían organizar de tal forma que se minimice el tiempo medio de acceso total. Esto es lo que se llama **planificación del disco**. Las características que debe tener un buen algoritmo de planificación son las siguientes:

- Maximizar la cantidad de peticiones servidas por unidad de tiempo.
- Minimizar el tiempo medio de retorno, es decir, el tiempo promedio de espera más el tiempo promedio de servicio.
- Minimizar la varianza, es decir, la desviación de una petición particular respecto al promedio.

A continuación veremos algunos de los algoritmos que se pueden utilizar.

3.5.3.2. Algoritmo FIFO

La forma más simple de servir las peticiones consiste en respetar su orden de llegada (FIFO). Éste es un método justo de servirlas, pero puede conducir a tiempos de espera muy largos. FIFO exhibe un patrón de búsqueda aleatorio, en el cual peticiones sucesivas pueden ocasionar búsquedas costosas, que impliquen ir de los cilindros más internos a los más externos o viceversa. La figura 3.13 muestra un ejemplo de aplicación de este algoritmo a un conjunto de peticiones.

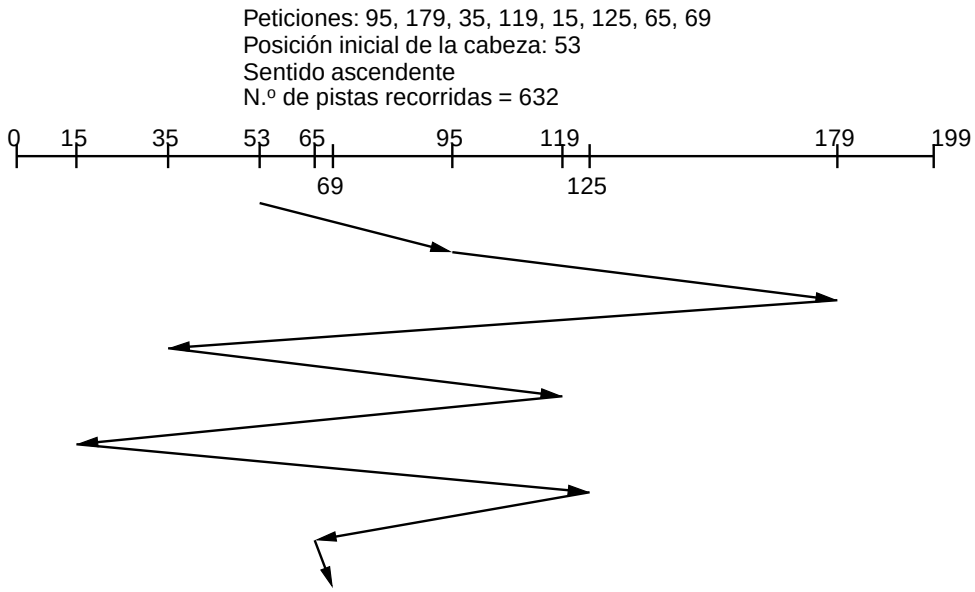

Figura 3.13: Algoritmo FIFO

3.5.3.3. Algoritmo SSTF

El algoritmo SSTF[2] selecciona la petición que requiere el menor tiempo de búsqueda, es decir, la que necesita recorrer un menor número de pistas desde la posición actual de la cabeza del disco. Por supuesto, esto no garantiza que el tiempo de búsqueda promedio para varios movimientos del brazo del disco vaya a ser mínimo. Sin embargo, suele dar mejores resultados que la política FIFO.

En la figura 3.14 se muestra el comportamiento de este algoritmo para la misma lista de peticiones anterior. En este caso, el número de pistas recorridas es 234, frente a las 632 del FIFO. Aunque el algoritmo SSTF proporciona mejores resultados que FIFO no es la estrategia óptima, ya que minimizar el tiempo de búsqueda de la siguiente petición no garantiza obtener el menor tiempo de búsqueda medio. Así, en el ejemplo anterior, si movemos la cabeza de la pista 53 a la 35, aunque ésta no sea la más cercana, y después a la 15, antes de dar servicio a 65,69,95,119,125 y 179, el movimiento total sería de 202 pistas.

Un problema que puede presentar el algoritmo SSTF es la inanición de aquellas solicitudes que se encuentran muy alejadas de la posición actual de la cabeza del disco. Si ésta se encuentra en un extremo y llega una solicitud del otro, puede ser que no se atienda si continuamente llegan peticiones del extremo donde nos encontramos. Esto hace que SSTF no sea aceptable en sistemas interactivos, ya que la varianza de los tiempos de retorno es alta.

SSTF necesita disponer de una regla de arbitraje, similar a la de los algoritmos de planificación de la CPU, para decidir en los casos de igualdad.

[2] *Shortest-Seek-Time-First.*

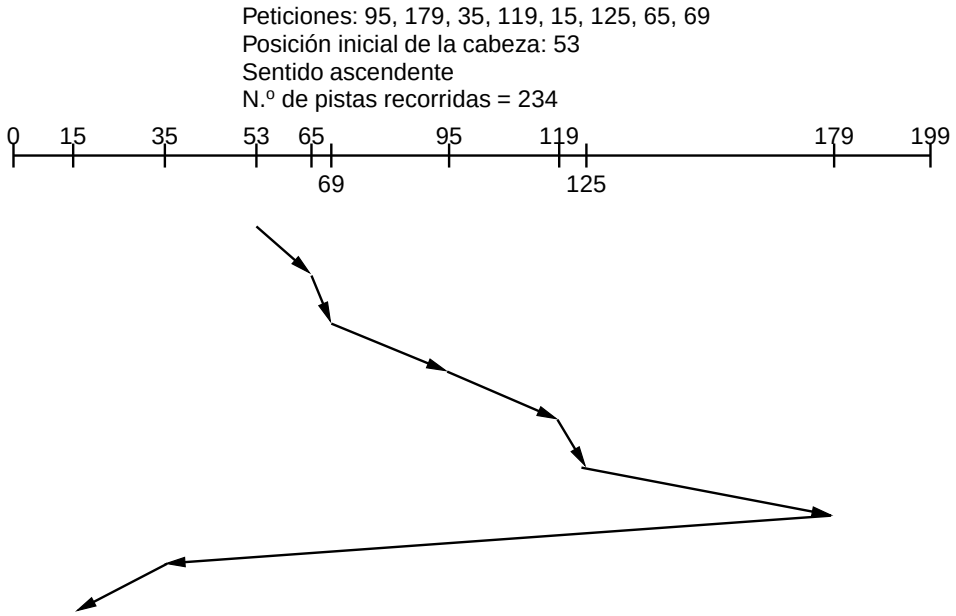

Figura 3.14: Algoritmo SSTF

3.5.3.4. Algoritmo SCAN

El algoritmo SSTF puede dejar discriminadas ciertas peticiones que implican un movimiento grande de la cabeza del disco. Para vencer este problema, Denning propuso el algoritmo SCAN.

En éste, la cabeza se mueve de un extremo del disco al otro, de forma que cuando llega a uno de ellos se invierte el sentido de su movimiento. Durante su recorrido va atendiendo todas las peticiones que encuentra. Así, peticiones recién llegadas pueden ser atendidas inmediatamente, mientras que las que afectan a posiciones por las que ha pasado recientemente tendrán que esperar hasta que la cabeza cambie de sentido.

Este algoritmo requiere conocer tanto la posición actual de la cabeza como el sentido del movimiento, para lo que emplea un bit de sentido.

En nuestro ejemplo, si la cabeza se está moviendo hacia la pista 199, su movimiento sería el que se muestra en la figura 3.15.

Este algoritmo elimina la posibilidad de inanición del SSTF, ofreciendo un rendimiento similar. Sin embargo, favorece a las pistas interiores debido a que en una vuelta del disco pasa dos veces por ellas, mientras que los extremos sólo los visita una vez. No obstante presenta una varianza mucho más baja que el algoritmo SSTF.

3.5.3.5. Algoritmo LOOK

En la práctica, SCAN suele implementarse sin que sea necesario llegar hasta los extremos para invertir el sentido de la cabeza del disco. Esta variante se denomina

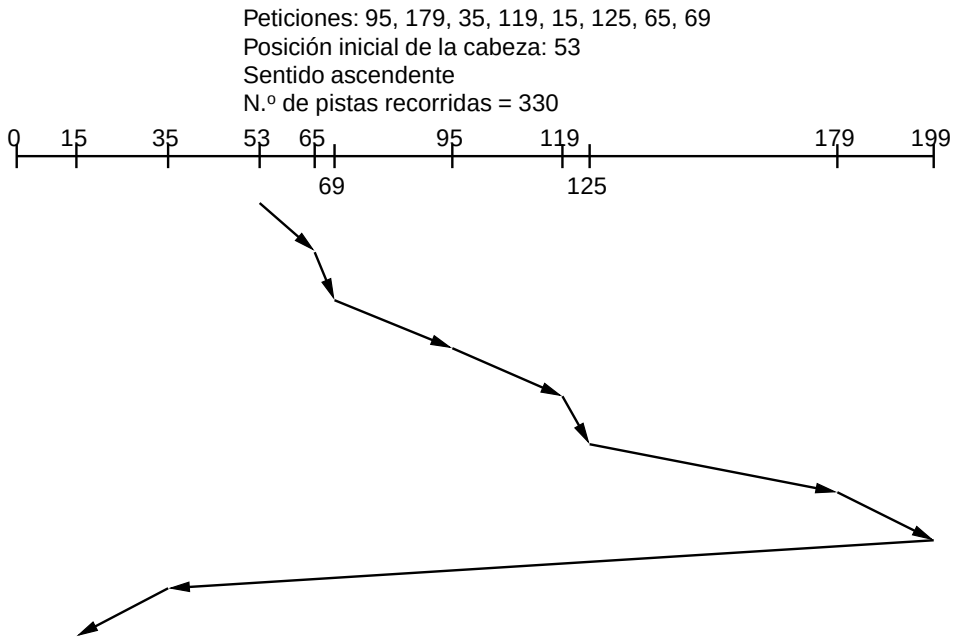

Figura 3.15: Algoritmo SCAN

algoritmo LOOK o del ascensor. La figura 3.16 muestra el comportamiento de este algoritmo en nuestro ejemplo.

3.5.3.6. Algoritmos C-SCAN y C-LOOK

Una modificación interesante de la estrategia SCAN es C-SCAN, también denominado SCAN circular o exploración circular. En C-SCAN, la cabeza se mueve de un extremo a otro sirviendo las peticiones que encuentra en su camino. Cuando llega al extremo salta al otro sin servir peticiones, volviendo a comenzar su recorrido. Este algoritmo considera al disco como si la última pista y la primera fueran adyacentes. Así, C-SCAN elimina completamente la discriminación de los extremos que presenta SCAN, presentando una varianza menor que éste.

Estudios realizados han mostrado que el algoritmo SCAN produce buenos resultados cuando la carga del sistema es baja, sin embargo, cuando la carga es alta C-SCAN se comporta mejor.

La figura 3.17 muestra el orden en el que se atenderían las distintas solicitudes de nuestro ejemplo con el algoritmo C-SCAN. En ella, la flecha con trazo discontinuo representa el salto de la cabeza de un extremo a otro sin servir peticiones.

Del mismo modo, existe un algoritmo C-LOOK que consiste en modificar la estrategia LOOK, tratando las pistas del disco como una lista circular, igual que en C-SCAN. La figura 3.18 muestra el comportamiento de este algoritmo en nuestro ejemplo.

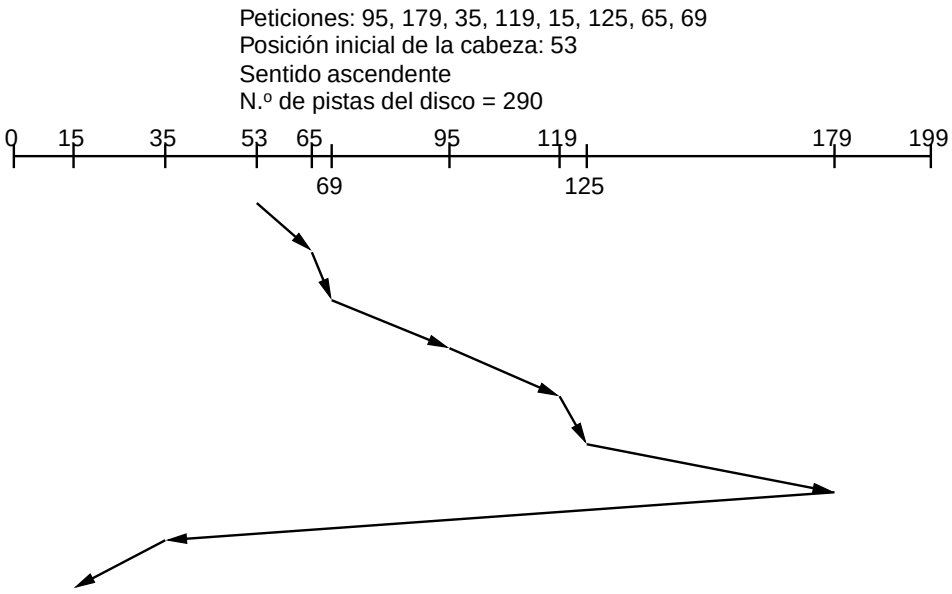

Figura 3.16: Algoritmo LOOK

Estas dos políticas suponen la existencia de una orden especial del dispositivo que mueve la cabeza de un extremo a otro en un breve espacio de tiempo, en relación al movimiento sobre cualquier otra pista.

3.5.3.7. Algoritmos N-SCAN y FSCAN

El algoritmo N-SCAN es una modificación interesante de SCAN. En ella la cola de peticiones se divide en varias de longitud N. Cada una se procesa independientemente usando SCAN. Mientras se está procesando una de ellas, las nuevas peticiones que van llegando se ponen en otra distinta. Si al terminar de procesar una cola hay menos de N peticiones disponibles, todas ellas se procesan a continuación.

El rendimiento de este algoritmo depende del valor de N. Así, para valores grandes, su rendimiento se aproxima al de SCAN; si $N = 1$, el algoritmo se convierte en un FIFO.

El algoritmo de planificación FSCAN utiliza dos colas. Cuando comienza una exploración, todas las peticiones están en una de ellas, estando la otra vacía. El algoritmo que emplea para atender las peticiones es SCAN. Todas las peticiones nuevas que llegan durante su recorrido se introducen en la cola que estaba inicialmente vacía.

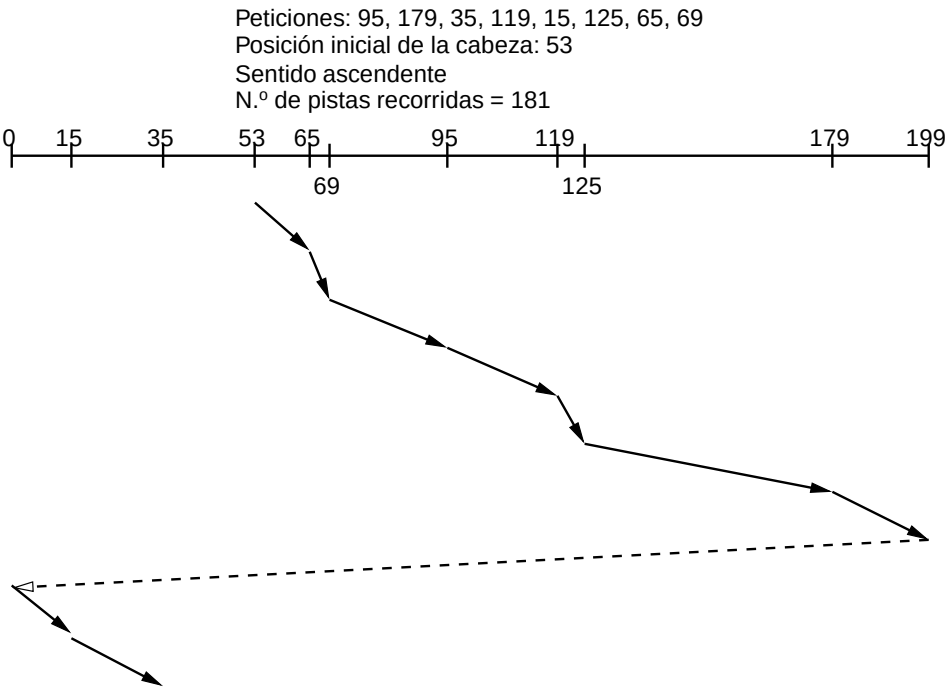

Figura 3.17: Algoritmo C-SCAN

3.6. E/S en LINUX

Una característica que diferencia a LINUX de otros sistemas operativos es el tratamiento de los dispositivos de E/S, puesto que éstos se tratan como ficheros, que se denominan **ficheros especiales**. Éstos se almacenan en el directorio `/dev`, de esta forma, al tener integrados los dispositivos de E/S en el sistema de ficheros, se logra una mayor flexibilidad de acceso.

Cada dispositivo se identifica por una clase (bloques o caracteres), un **número mayor** y un **número menor**. La clase permite acceder a la tabla de configuración de los dispositivos, que apunta a los manejadores de éstos. El número mayor indica el tipo de dispositivo, que se usa como índice en la tabla anterior para acceder al manejador correspondiente. El número menor se utiliza para identificar un dispositivo físico entre varios del mismo tipo.

3.6.1. Técnicas de optimización

Para los dispositivos de bloques, LINUX dispone de una caché de *buffers* situada entre los manejadores y el sistema de ficheros. Ésta es una tabla en el núcleo que contiene los bloques de uso reciente, y se utiliza tanto en las operaciones de escritura como en las de lectura. El algoritmo de sustitución que emplea es el LRU. Periódicamente se realizan escrituras de los bloques que han sido modificados para minimizar las potenciales inconsistencias en los sistemas de ficheros.

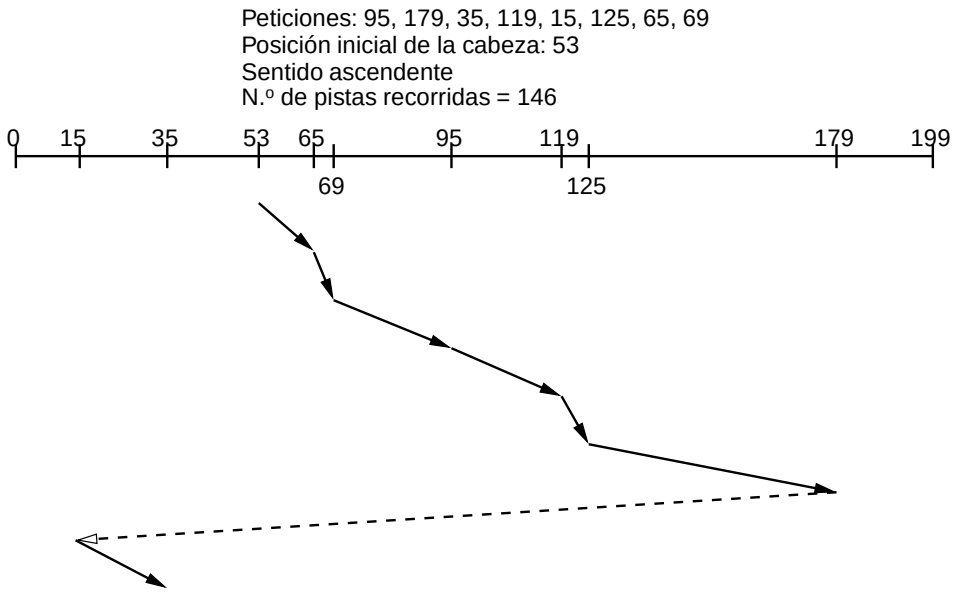

Figura 3.18: Algoritmo C-LOOK

El tamaño de la caché de *buffers* puede ser un parámetro importante para la eficiencia de un sistema, ya que, si es suficientemente grande, el porcentaje de éxitos en caché puede ser grande y el número de transferencias de E/S bajo.

Otro aspecto de optimización a tener en cuenta es la planificación de peticiones al disco. LINUX emplea el algoritmo LOOK.

Los dispositivos de caracteres no usan la caché de *buffers* porque trabajan con flujos de caracteres para pasar la información entre el dispositivo y la memoria. Algunos de ellos utilizan estructuras de datos especiales llamadas **listas-C**[3], que son *buffers* circulares.

Cada lista-C está formada por un contador y una lista de bloques de tamaño variable. Al llegar los caracteres de las terminales y otros dispositivos de caracteres, éstos se almacenan en la lista. Permiten la manipulación de datos carácter a carácter o en grupos. Es útil para la transmisión de pequeños volúmenes de datos típicos de dispositivos lentos tales como terminales.

3.6.2. Llamadas al sistema

La E/S en un sistema LINUX se maneja principalmente usando las llamadas al sistema que aparecen en la tabla 3.1.

[3] *C-list: character list.*

Llamada	Descripción
open	Solicitar un dispositivo.
close	Liberar un dispositivo.
read	Leer de un dispositivo.
write	Escribir en un dispositivo.
lseek	Posicionarse en un dispositivo.
ioctl	Establecer opciones al dispositivo.

Cuadro 3.1: Llamadas al sistema relacionadas con los dispositivos

3.7. Resumen

El sistema de E/S es el componente del sistema operativo encargado de la realización de operaciones de E/S. Sus objetivos principales son el rendimiento y la uniformidad.

Los dispositivos de E/S permiten el intercambio de información entre el sistema y el mundo exterior. Éstos se conectan al sistema a través de las controladoras, que actúan como intermediarias. El objetivo de éstas es superar las incompatibilidades de velocidad, señalización, traducción de órdenes de E/S genéricas a controles específicos del dispositivo, etc.

El sistema operativo, a través de los manejadores de dispositivos, efectúa las operaciones de E/S escribiendo las órdenes oportunas en los registros de la controladora. Dado que existe una gran variedad de dispositivos, para conseguir la uniformidad a la hora de realizar una operación de E/S, se implementa una interfaz común para todos ellos, la API. De este modo, añadir un nuevo dispositivo consiste en agregar su manejador respetando la interfaz existente con los usuarios.

Para realizar las operaciones de E/S existen tres técnicas, la E/S controlada por programa, la E/S controlada por interrupciones y el acceso directo a memoria. En la primera, la CPU es la encargada de realizar las transferencias entre la memoria y los registros de datos de la controladora. En concreto, es el manejador del dispositivo el que copia los datos del área del espacio de usuario a los registros de la controladora y viceversa. El problema es la existencia de una espera activa mientras se realiza una operación de E/S, debido a la necesidad de interrogar de forma repetida el estado del dispositivo.

Una alternativa a este enfoque es la E/S por interrupciones. En este caso, la controladora notificará al sistema operativo de forma automática el fin de la operación de E/S mediante una interrupción. De este modo, el procesador tras emitir una orden de E/S a la controladora puede continuar con la ejecución de otro proceso. Al finalizar la operación de E/S, la controladora producirá una interrupción para realizar la transferencia de datos entre ella y el sistema.

Por último, el acceso directo a memoria permite la realización de operaciones de E/S directamente con la memoria, sin necesidad de interrumpir al procesador para realizar la transferencia. Esta técnica es más eficiente que las anteriores, sobre todo

en grandes volúmenes de datos.

Un aspecto importante del sistema de E/S es el rendimiento de las operaciones, debido a la diferencia en velocidad entre el procesador y los dispositivos. Así, para los dispositivos de caracteres se puede aplicar una técnica de *buffering* para simultanear la realización de una operación de E/S con una de cálculo. En el caso de dispositivos de bloques se dispone de la caché de disco, que consiste en tener en memoria los últimos bloques transferidos. De este modo, cuando se necesite un bloque del dispositivo, es posible que se encuentre en la caché, por lo que no hay que realizar transferencia alguna.

Para los discos, podemos realizar una planificación de las peticiones con objeto de minimizar el tiempo medio de acceso. Existen diversos algoritmos de planificación como FIFO, SSTF, LOOK, etc., aunque diferentes estudios realizados indican que la mejor política podría operar en dos etapas. Cuando la carga es baja, la política SCAN es la mejor. Para una carga media a alta, C-SCAN conduce a los mejores resultados.

Tema 4

Sistemas de ficheros

El sistema operativo, a través del sistema de ficheros, proporciona al usuario un mecanismo para almacenar información de forma permanente. En este capítulo abordaremos la implementación de un sistema de ficheros, determinando cómo es la interfaz de éste con los usuarios mediante el estudio de los ficheros y directorios. Posteriormente veremos las distintas estrategias para la gestión del espacio libre, asignación de espacio a los ficheros y los métodos existentes para mejorar el rendimiento.

4.1. Introducción

El objetivo principal de un sistema es ejecutar programas. Éstos junto con los datos a los que acceden deben encontrarse en memoria principal durante la ejecución. En una situación ideal, nos gustaría que todos los programas y datos residieran permanentemente en la memoria principal. Esto no es posible por dos razones, la memoria es pequeña para contener todos los datos de forma permanente y, además, es un dispositivo de almacenamiento volátil, por lo que pierde su contenido al apagar el sistema.

No sólo existen los problemas de necesidad de almacenamiento, sino que también puede presentarse la necesidad de que varios procesos accedan a distintas partes de

esa información al mismo tiempo. Si esta información está almacenada dentro del espacio de direcciones de un proceso, sólo éste podría acceder a ella. La forma de resolver este problema es hacer que la información sea independiente de cualquier proceso.

Tenemos, por tanto, tres requisitos esenciales para el almacenamiento de información a largo plazo:

1. Almacenar una gran cantidad de información.
2. La información debe permanecer después de la terminación del proceso que la está usando.
3. Múltiples procesos deben ser capaces de acceder a la información concurrentemente.

La solución a estos problemas consiste en almacenar la información en medios externos (discos, cintas, CD-ROM, etc.), en unidades llamadas **ficheros**. Un fichero es una colección de datos con un nombre que suele residir en un dispositivo de almacenamiento secundario.

Los ficheros son manejados por el sistema operativo mediante el denominado **sistema de ficheros**. Éste es muy importante dentro del sistema operativo, ya que es la parte más visible por el usuario.

No se puede, ni se debe hacer una separación radical entre el capítulo anterior sobre gestión de dispositivos y el que estamos estudiando. Aunque todas las operaciones de E/S que se realizan en un sistema no están relacionadas con los ficheros, si lo están una gran parte de ellas, por lo que es interesante ver el punto de contacto entre ambos capítulos. En la figura 4.1 se muestra cómo se relacionan estos componentes del sistema. Podemos ver cómo las operaciones en las que están implicados ficheros, tienen que ir a través del sistema de ficheros antes de acceder al dispositivo mediante su manejador.

4.2. Funciones del sistema de ficheros

Para comprender mejor las funciones que realiza el sistema de ficheros, veamos qué operaciones se realizan cuando un usuario da una orden sobre un fichero.

Los usuarios y los programas de aplicación interaccionan con el sistema de ficheros por medio de órdenes para la creación, borrado y la realización de operaciones sobre ficheros. Antes de realizar cualquier operación, el sistema de ficheros debe identificar y localizar el fichero seleccionado. Esto requiere el uso de alguna estructura que describa la localización de todos los ficheros más sus atributos. Ésta se suele denominar **directorio**. Además, la mayor parte de los sistemas de tiempo compartido tienen un sistema de control de acceso de los usuarios a los ficheros, así sólo pueden acceder a un fichero los usuarios autorizados y de una determinada forma.

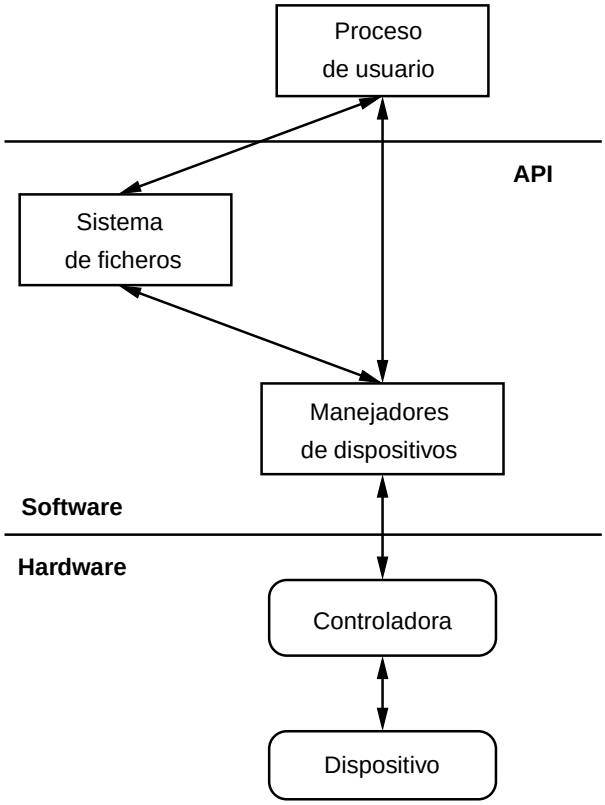

Figura 4.1: Relación entre el sistema de ficheros y el de E/S

El usuario o aplicación ve el fichero con una estructura de registros. De esta forma, las operaciones básicas que se pueden realizar sobre un fichero son llevadas a cabo sobre los registros individuales. Así, para traducir órdenes de usuario en órdenes específicas de manipulación de ficheros debe emplearse el método de acceso adecuado a la estructura del fichero.

Mientras que los usuarios y las aplicaciones están interesados en los registros, la E/S se hace por bloques. Así, para la realización de una operación de E/S sobre un fichero sus registros deben ser agrupados en bloques. Para el manejo de los dispositivos de bloques el sistema debe disponer de técnicas para el manejo del espacio libre y para la asignación de bloques a los ficheros.

En la tabla 4.1 podemos ver las funciones del sistema de ficheros, donde se separan las que forman parte de la visión del usuario, el **sistema de ficheros lógico**, y las del sistema operativo, **el sistema de ficheros físico**. Por tanto, a la hora de diseñar el sistema de ficheros nos encontramos con dos problemas. El primero de ellos es definir cómo aparecerá éste ante el usuario, es decir, cómo se define un fichero, sus atributos, operaciones permitidas, y la estructura de los directorios. El otro son los algoritmos y estructuras de datos que deben crearse para establecer una correspondencia entre el sistema de ficheros lógico y el físico. En este capítulo estudiaremos ambas visiones del sistema de ficheros, su interfaz y su diseño, respectivamente.

Sistema de ficheros lógico	
Gestión de directorios	Creación de ficheros
	Borrado de ficheros
	Localización de ficheros
Gestión de ficheros	Método de acceso a la información
Control de acceso	Operaciones permitidas al usuario
Sistema de ficheros físico	
Agrupamiento de registros en bloques	Fijo
	Longitud variable extendido
	Longitud variable no extendido
Asignación de espacio del disco	Contigua
	Encadenada
	Indexada
Gestión del espacio libre	Lista enlazada
	Mapa de bits
Fiabilidad	Consistencia del sistema
	Copias de seguridad

Cuadro 4.1: Funciones del sistema de ficheros

De forma gráfica, esas funciones pueden visualizarse en la figura 4.2, la cuál sugiere una división entre lo que podría ser considerado como la visión por parte del usuario, y por otro la del sistema operativo. Por tanto, a la hora de diseñar el sistema de ficheros nos encontramos con dos problemas.

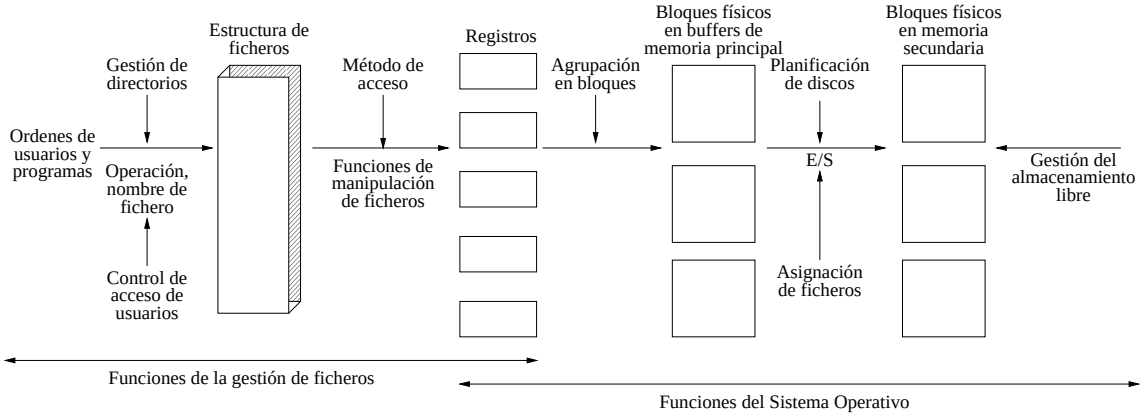

Figura 4.2: Funciones del sistema de ficheros

4.3. Interfaz del sistema de ficheros

Desde el punto de vista del usuario los aspectos a tener en cuenta en un sistema de ficheros son la estructura del fichero, la forma en que éstos se nombran y protegen, las operaciones que se permiten hacer con ellos, etc.

4.3.1. Ficheros

Los ficheros son un mecanismo de abstracción que proporcionan una forma de almacenar información en dispositivos de almacenamiento secundario. Éstos permiten ocultar al usuario los detalles de cómo y dónde está almacenada la información, así como la forma en que funcionan los dispositivos de almacenamiento.

Todos los sistemas operativos asocian cierta información a cada fichero, por ejemplo, su nombre, tipo, la fecha y la hora en que el fichero fue creado, su tamaño, su localización, etc. A estos datos se les llaman **atributos del fichero**. La lista de atributos puede variar considerablemente de un sistema a otro, así como los valores que éstos pueden tomar.

Los ficheros no suelen ser elementos estáticos, por el contrario, suelen sufrir numerosas modificaciones y manipulaciones durante su vida. Los sistemas operativos proporcionan servicios para realizar operaciones sobre los ficheros, que son independientes del dispositivo físico donde están almacenados. Las operaciones básicas que se pueden realizar son:

- Creación.
- Escritura.
- Lectura.
- Búsqueda.
- Borrado.

Estas operaciones son las mínimas que se deben proporcionar, y a partir de ellas, combinándolas, se pueden obtener otros tipos de operaciones comunes.

Muchos sistemas operativos soportan varios tipos de ficheros. LINUX, por ejemplo, distingue entre ficheros regulares, directorios y ficheros especiales de dispositivos. Los ficheros regulares son los que contienen información del usuario. Los directorios mantienen información sobre la estructura del sistema de ficheros, y los ficheros especiales están relacionados con la E/S.

El sistema operativo MS-DOS, que admite extensiones en los nombres de ficheros, utiliza éstas para distinguir diferentes tipos de ficheros y las operaciones que se pueden realizar con éstos. Por ejemplo, sólo un fichero con una extensión `.com`, `.exe` o `.bat` puede ser ejecutado.

4.3.1.1. Estructura de los ficheros

Al hablar de estructura de los ficheros debemos hacer una distinción entre **estructura lógica** u **organización del fichero**, es decir, la visión externa que se tiene de éste, y la **estructura física** o **visión interna del fichero**.

Un fichero es una colección de **registros**, que a su vez, suelen componerse de uno o varios **campos**. Por este motivo, un elemento fundamental en el diseño de un sistema de ficheros es la forma en la que los registros están organizados o estructurados. La organización de ficheros hace referencia a la organización lógica de los registros, es decir, la forma en que se guardan y ordenan los registros de un fichero. Por tanto, la colección de todos los registros que componen el fichero y la percepción que tiene el usuario de la forma en que están organizados constituye la estructura lógica del fichero, que coincide con la visión que tiene el usuario de éste.

La estructura física de un fichero es la colección de bloques físicos que lo componen y la forma en que éstos están almacenados en memoria secundaria.

Hay sistemas operativos que soportan múltiples estructuras lógicas, como por ejemplo VMS, que define tres. Sin embargo, otros sistemas consideran una sola estructura, como por ejemplo LINUX, que considera un fichero como una secuencia de bytes. Ambos enfoques tienen sus ventajas y sus desventajas. En el primer caso, el sistema operativo nos va a prestar más servicios, pero también va a ser mucho más complicado. En el segundo, vamos a tener un sistema más simple y flexible, aunque nos preste menos ayuda en algunos casos.

Existen distintos tipos de organizaciones de ficheros:

- Pila.
- Secuencial.
- Secuencial-indexada.
- Indexada.
- Directa o de dispersión.

Estrechamente relacionada con la organización, está la forma de acceder a la información que hay en los ficheros. Esto es lo que se conoce como **método de acceso**. Algunos sistemas sólo proporcionan un único método, mientras que otros proporcionan varios. Estos métodos pueden ser:

- Acceso secuencial.
- Acceso secuencial-indexado.
- Acceso directo o indexado.

En un sistema multiusuario además de tener en cuenta los aspectos anteriores, hemos de pensar que existe la necesidad de permitir a los usuarios compartir ficheros. Entonces aparecen dos cuestiones, derechos de acceso y acceso simultáneo. La primera hace referencia a las operaciones que pueden hacerse sobre los ficheros y la segunda a la la posibilidad de que varios usuarios deseen actualizar el mismo fichero, haciendo necesario incorporar mecanismos que aseguren el acceso exclusivo al fichero.

4.3.2. Directorios

Todo sistema de ficheros necesita de una estructura que permita organizar de alguna forma el conjunto de ficheros. Los directorios son los que realizan esta función. Éstos son ficheros que contienen información sobre otros ficheros y se diferencian de ellos en el modo de acceso.

La principal tarea de un directorio es la de asociar los nombres de los ficheros con su ubicación física en el almacenamiento secundario. Para ello, los directorios se estructuran en entradas, una por fichero. Cada entrada incluirá el nombre del fichero y atributos de éste, como su ubicación física.

Sin embargo, la información que contienen los directorios acerca de los ficheros varía de un sistema a otro. En algunos contienen todos los atributos de los ficheros, y en otros simplemente un puntero a una estructura de control del fichero.

Dependiendo de la información que se almacene en cada entrada del directorio, su tamaño será mayor o menor. Los directorios normalmente se almacenan en memoria secundaria debido a que pueden resultar muy grandes. No obstante, se suele cargar una parte en memoria principal para mejorar la velocidad de acceso.

Las operaciones que se pueden realizar sobre un directorio son:

- Ver los atributos de un fichero.
- Establecer los atributos de un fichero.
- Renombrar, crear, borrar, copiar y mover ficheros.
- Crear y deshacer enlaces.
- Listar el contenido del directorio.

Los directorios se pueden organizar de distintas formas, a continuación se estudiarán las más comunes.

4.3.2.1. Directorio con estructura de árbol

Esta estructura permite que un directorio contenga tanto ficheros como otros directorios (figura 4.3). Esto permite a los usuarios crear sus propios directorios y organizar sus ficheros. El sistema de ficheros de MS-DOS, por ejemplo, tiene estructura de árbol. El árbol tiene un **directorio raíz**, a partir del cuál se organizan todos los ficheros y directorios del sistema. Cada fichero en el sistema tiene un camino único, que es su ubicación dentro del árbol. Este camino puede ser **absoluto** o **relativo**, según venga expresado a partir del directorio raíz o a partir del directorio donde se encuentra situado el usuario, denominado **directorio de trabajo**, respectivamente.

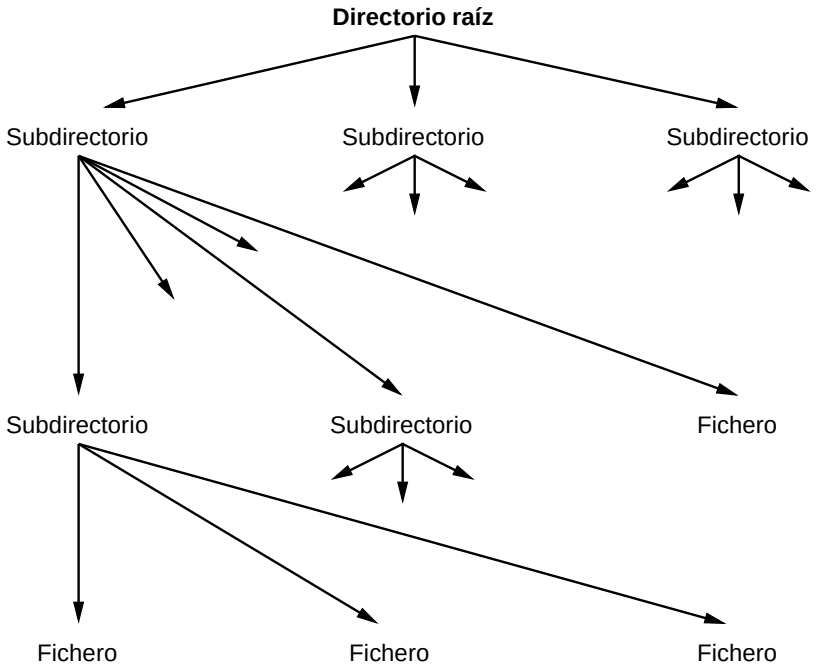

Figura 4.3: Directorio con estructura de árbol

4.3.2.2. Directorio con estructura de grafo acíclico

Una estructura de árbol no permite la compartición de ficheros o directorios. Un grafo acíclico permite que un directorio tenga ficheros compartidos, ya que un mismo fichero puede aparecer en dos directorios diferentes, como se muestra en la figura 4.4.

Los ficheros compartidos pueden ser implementados de varias formas. Una forma común, usada por LINUX, es crear una nueva entrada de directorio llamada **enlace duro**, que duplica la entrada del fichero.

Una estructura de grafo acíclico es más flexible que una estructura de árbol, pero también, más compleja. Pueden aparecer varios problemas debido a que un fichero puede tener múltiples caminos. Así, cuando examinamos el sistema de ficheros completo (para encontrar un fichero, tomar datos estadísticos sobre los mismos, o hacer una copia de seguridad) este problema es significativo, puesto que no se debe recorrer las estructuras compartidas más de una vez.

Otro problema está relacionado con el borrado. ¿Cuándo se debe borrar la estructura del fichero? Sólo cuando se hayan eliminado todas las referencias, lo que implica llevar un contador de éstas.

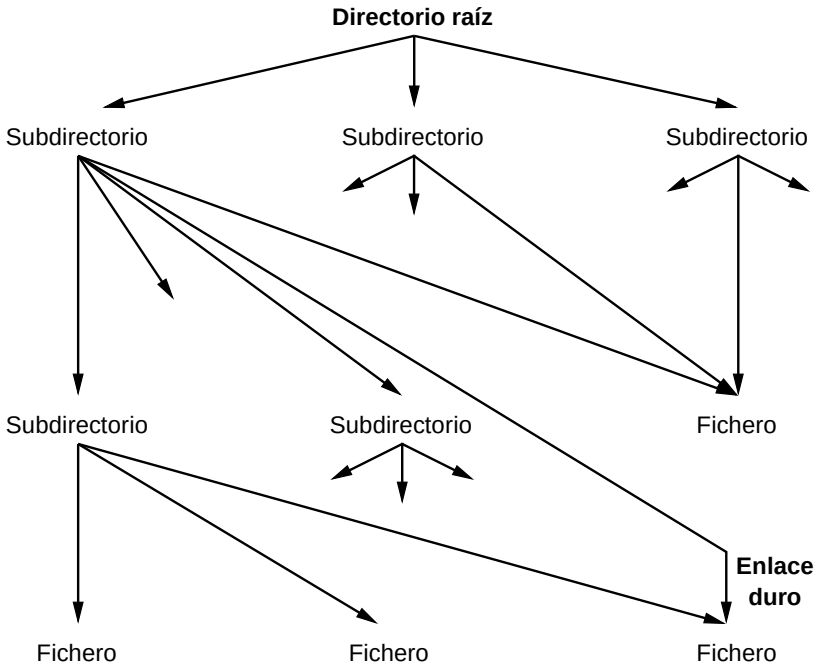

Figura 4.4: Directorio con estructura de grafo acíclico

4.3.2.3. Directorio con estructura de grafo general

Es el caso más general de estructura de directorio. Se da cuando en el grafo acíclico se permite que un subdirectorio esté catalogado en más de un directorio. Esto se puede conseguir en los sistemas LINUX con los llamados **enlaces simbólicos**. Éste no es más que un puntero a otro fichero o subdirectorio. Cuando se hace referencia a un enlace simbólico, se busca en el directorio su entrada que está marcada como un enlace y proporciona el camino para localizar el fichero real. Los problemas que teníamos en el caso de un grafo acíclico se acentúan más.

4.4. Diseño del sistema de ficheros

En este apartado vamos a estudiar los sistemas de ficheros desde el punto de vista del diseñador del sistema operativo, cómo manejar el espacio en disco, cómo almacenar los ficheros y cómo hacer que todo funcione de forma eficiente y fiable.

4.4.1. Agrupamiento de registros

Los registros son las unidades lógicas de acceso a un fichero, mientras que los bloques son las unidades de E/S del disco. Como normalmente el tamaño del bloque y de los registros no coinciden es necesario realizar el agrupamiento de registros.

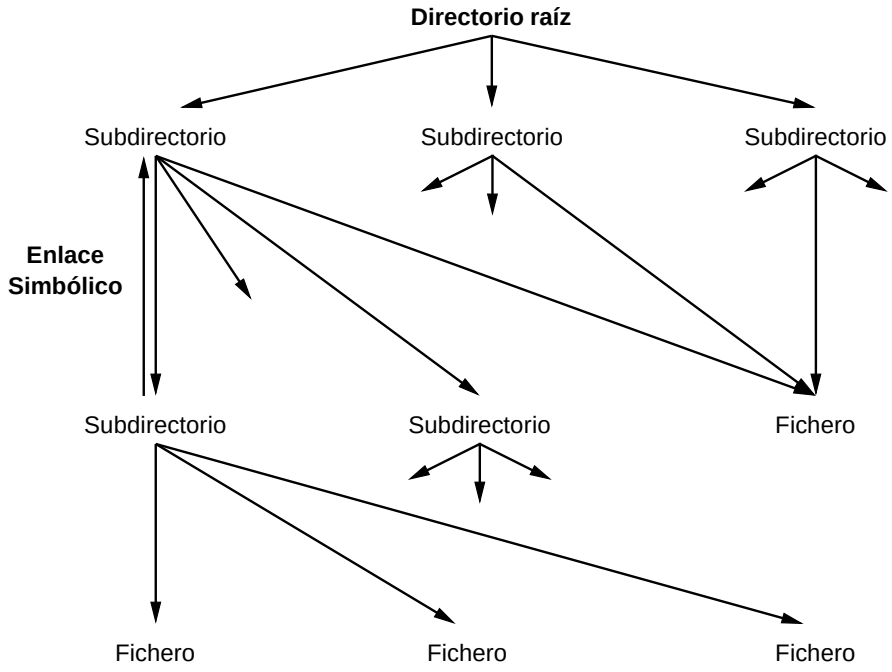

Figura 4.5: Directorio con estructura de grafo general

Antes de estudiar éste haremos algunas consideraciones acerca del tamaño de los bloques.

Hay varios aspectos a considerar. Primero, ¿deberían ser los bloques de longitud variable o fija? En la mayoría de los sistemas los bloques son de longitud fija. Esto simplifica la E/S, la asignación de memoria intermedia en memoria principal, y la organización de los bloques en el almacenamiento secundario.

Otro aspecto a considerar es el tamaño relativo de un bloque con relación al tamaño promedio de los registros. La situación es la siguiente, cuanto más grande sea un bloque, más registros se pasan en una operación de E/S. Si un fichero está siendo procesado o se está realizando una búsqueda de forma secuencial, esto sería una ventaja porque el número de operaciones de E/S se ve reducido al usar bloques más grandes, acelerándose así el proceso. Por otro lado, si se va a acceder de forma aleatoria a los registros, entonces los bloques más grandes dan lugar a una transferencia innecesaria de registros que no se van a usar. Además, hemos de tener en cuenta que el bloque es la unidad mínima de asignación en la memoria secundaria, por lo que ficheros pequeños pueden desperdiciar espacio en el bloque. Estos dos factores se ven reflejados en la gráfica de la figura 4.6.

El tamaño del bloque puede colaborar con el hardware de la memoria virtual. En un entorno de memoria virtual paginada es deseable que la unidad básica de transferencia sea la página. De este modo, conviene formar bloques que equivalgan a una página o bien a un conjunto de ellas, para que en cada operación de E/S dispongamos de un conjunto de páginas.

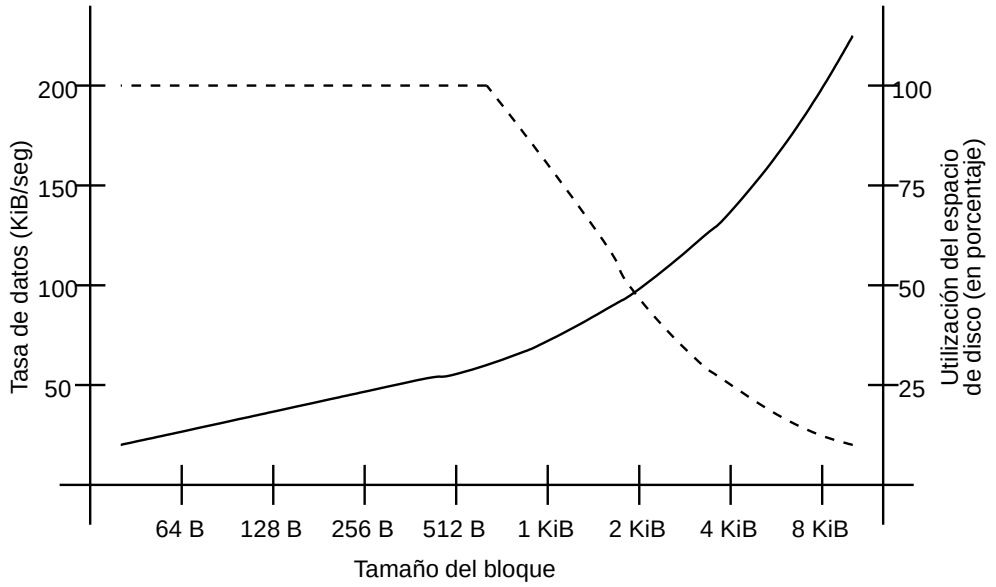

Figura 4.6: Efecto del tamaño de bloque

Se suelen utilizar los métodos siguientes de agrupación de registros:

Agrupamiento fijo Se usan registros de longitud fija, y en cada bloque se almacena un número determinado y entero de registros, conocido como **factor de bloque**. Puede existir espacio no usado al final de cada bloque, es decir, fragmentación interna. Es el método más común para ficheros secuenciales con registros de longitud fija. El tamaño del registro está limitado al tamaño del bloque.

Agrupamiento de longitud variable extendido Se usan registros de longitud variable que son empaquetados en bloques sin dejar espacio libre. Así, algunos registros deben expandirse a dos o más bloques; en este caso, un puntero indica donde reside el resto del registro. Constituye un almacenamiento eficaz, pero es de difícil implementación, pues se pueden requerir dos operaciones de E/S para leer un registro, y además se hace difícil la actualización de los ficheros. Se conoce también como **agrupamiento de longitud variable por tramos**.

Agrupamiento de longitud variable no extendido También denominado **agrupamiento de longitud variable sin tramos**. Se usan registros de longitud variable, pero no se emplea la expansión. Hay espacio desperdiciado en la mayoría de los bloques debido a la incapacidad para usar el resto de un bloque si el registro siguiente es mayor que el espacio sobrante no usado. También se limita el tamaño máximo de un registro al tamaño de un bloque.

La figura 4.7 ilustra estos métodos, considerando que un fichero está almacenado en bloques secuenciales en un disco. El efecto no cambiaría si se utilizara cualquier otro método de asignación del espacio del disco.

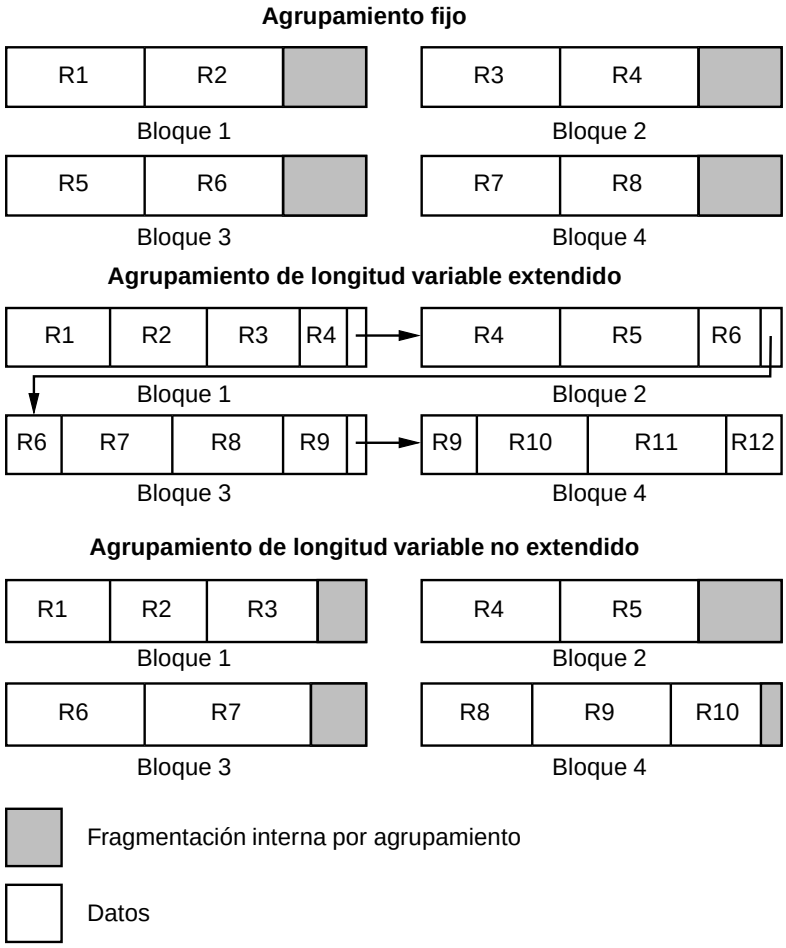

Figura 4.7: Métodos de agrupamiento de registros

4.4.2. Métodos de asignación de espacio a los ficheros

Se suelen utilizar tres métodos, **asignación contigua**, **enlazada** e **indexada**. Cada uno tiene sus ventajas y sus inconvenientes, por lo que algunos sistemas operativos proporcionan más de un método, aunque la mayoría sólo proporciona un método de asignación para todos los ficheros.

4.4.2.1. Asignación contigua

En este caso se asigna al fichero un conjunto contiguo de bloques, en el momento de su creación, suficiente para almacenar su contenido actual y hacer frente a su posible crecimiento en un futuro. Por tanto es una estrategia de asignación previa. En este caso, la entrada del directorio necesita almacenar el bloque de comienzo, el tamaño máximo del fichero y su tamaño actual. Esto se puede ver en la figura 4.8.

Este tipo de asignación permite acceso secuencial y directo a los ficheros. Para un acceso secuencial el sistema de ficheros recuerda la dirección de disco del último

bloque referenciado y, cuando es necesario, lee el bloque siguiente. Para el acceso directo al bloque x de un fichero que comienza en el bloque i, podemos acceder inmediatamente al bloque $i + x - 1$.

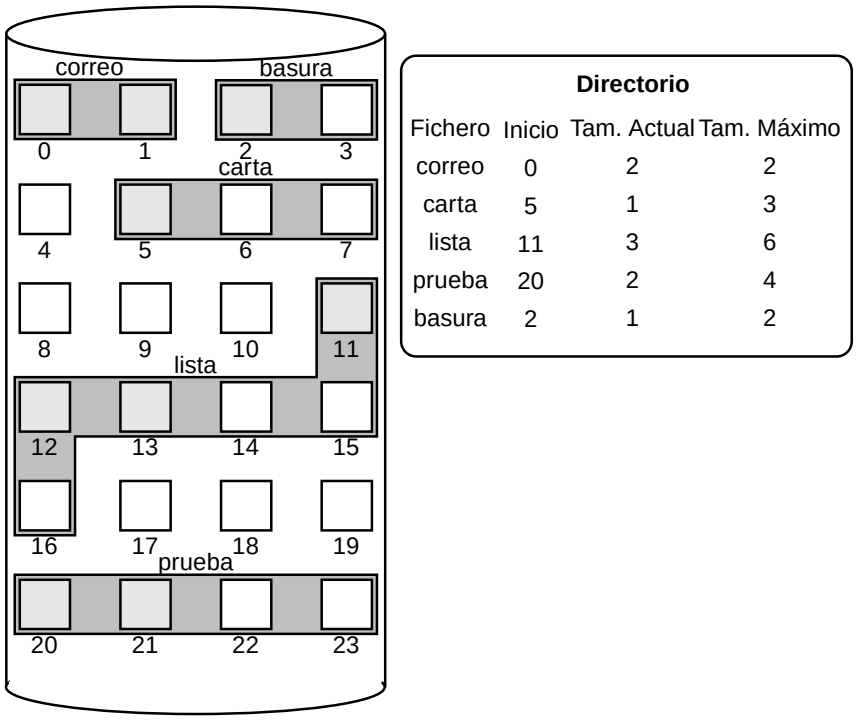

Figura 4.8: Asignación contigua

La asignación contigua presenta algunos problemas:

- Puede aparecer fragmentación externa, haciendo difícil encontrar bloques contiguos libres de suficiente longitud. Para reducirla es necesario la compactación del disco de forma periódica, pero esto es costoso en tiempo, obligando a detener el sistema mientras se realiza.

- Además, es necesario realizar una política de asignación del espacio libre como primer ajuste, mejor ajuste, peor ajuste, etc.

- Es necesario declarar el tamaño máximo de un fichero en el momento de su creación (preasignación). Esto origina que inicialmente se produzca fragmentación interna, debido a que se solicita el tamaño máximo que tendrá el fichero, pudiendo ocuparlo o no.

- Dificultad en la realización de operaciones de borrado de registros, creación de nuevos registros, etc.

4.4.2.2. Asignación enlazada

Otro método de asignación es la enlazada o encadenada (figura 4.9). En este caso, los bloques se asignan individualmente y cada uno de ellos contiene, además de una zona de datos, un puntero al siguiente bloque de la cadena. Cada entrada del directorio contendrá el primer y el último bloque del fichero. Se realiza una asignación dinámica de acuerdo a las necesidades del fichero y cualquier bloque puede serle asignado. Por tanto, no hay problemas de fragmentación externa.

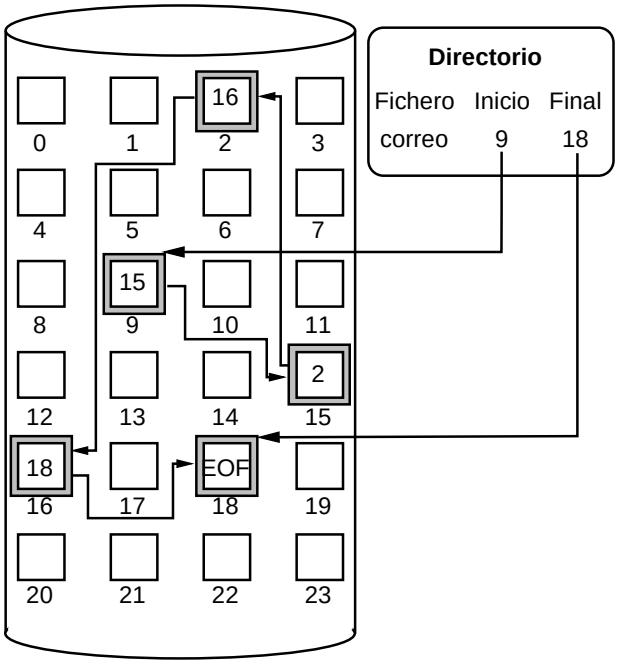

Figura 4.9: Asignación enlazada

Este tipo de organización sólo permite el acceso secuencial a los ficheros, ya que para acceder a un bloque determinado hay que leer los anteriores. Dado que los bloques pueden estar dispersos por todo el dispositivo, se puede realizar una desfragmentación para agrupar los bloques del fichero y acelerar el acceso.

Un problema más grave es la fiabilidad. Dado que los bloques de un fichero están enlazados mediante punteros, si uno de ellos se pierde, perderíamos el resto de la información. Una solución parcial es la utilización de listas doblemente enlazadas o almacenar el nombre del fichero y el número de bloque relativo dentro de cada bloque; estos métodos requieren aún más sobrecarga para cada fichero.

Los sistemas operativos MS-DOS y OS/2 utilizan una variante del método de asignación enlazada. En ellos, en vez de guardarse la información sobre cuál es el siguiente bloque del fichero en los propios bloques de datos se construye lo que se llama una **tabla de asignación de ficheros** (FAT). En cada partición del disco se reserva un espacio para contener esta tabla. Ésta tiene una entrada por cada bloque del disco, y está indexada por el número de bloque.

Las dos primeras entradas de la FAT están reservadas para contener el tamaño del disco; el resto describe la utilización de los bloques. El contenido de cada entrada puede ser:

- Bloque disponible (FREE).
- Ultimo bloque del fichero (EOF).
- Siguiente bloque del fichero (n).
- Bloque en mal estado (BAD).

Cada entrada del directorio contiene el número del primer bloque asignado al fichero; éste se utiliza como punto de entrada en la FAT. A partir de éste, cada entrada de la FAT contiene el número del siguiente bloque, hasta llegar al final del fichero. En el caso del fichero A (figura 4.10), éste comienza en el bloque 6. La entrada número 6 de la FAT contiene el número del siguiente bloque asignado al fichero, el 8; esta entrada apunta al siguiente bloque, el 4; la entrada número 4 almacena el número del siguiente bloque, el 2; como éste es el último bloque asignado al fichero, la entrada número 2 de la FAT contiene el código de fin de fichero (EOF).

El problema de la FAT es que la información sobre la ubicación de los ficheros en el disco se combina al azar en la misma tabla. Esto quiere decir que se necesita toda la FAT para abrir un fichero.

La asignación enlazada resuelve los problemas de la fragmentación externa y la declaración del tamaño máximo del fichero que conlleva la asignación contigua. Sin embargo, no soporta de forma eficiente el acceso directo a los ficheros.

4.4.2.3. Asignación indexada

La asignación indexada resuelve los problemas de los esquemas anteriores, es decir, soporta el acceso directo sin sufrir fragmentación externa. Es la forma más utilizada actualmente de asignación de espacio a los ficheros.

Al igual que la asignación encadenada, ésta permite asignar dinámicamente los bloques a un fichero según sus necesidades. La información de qué bloques componen un fichero se mantiene en su **bloque índice**. La entrada i-ésima del bloque índice apunta al i-ésimo bloque del fichero. La figura 4.11 muestra este esquema de asignación.

Este tipo de asignación permite tanto el acceso secuencial como el directo. Para acelerar los accesos, el bloque índice se almacena en memoria principal en el momento en que se abre el fichero, manteniéndose en ella hasta que se cierra. De este modo, al trabajar con un fichero disponemos en memoria principal de cuál es la localización de los distintos bloques de datos que posee, sin necesidad de requerir dos accesos a disco para localizar un bloque de datos.

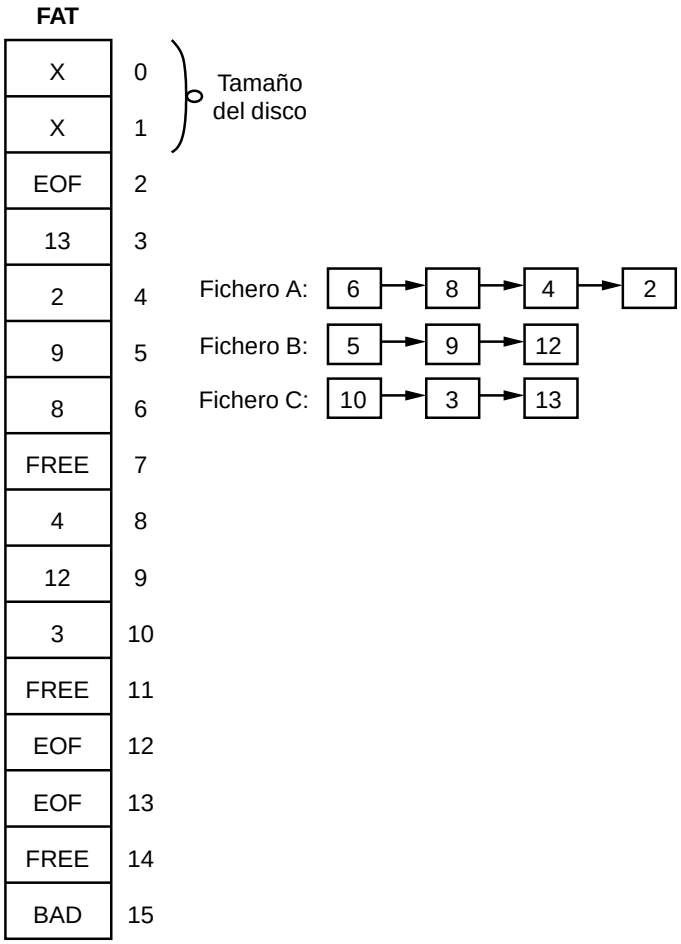

Figura 4.10: Tabla de asignación de ficheros

Normalmente los bloques índices no forman parte del directorio, sino que éste mantiene un puntero al mismo. Con esta estrategia, el fichero tiene tantos bloques de datos como necesite, además cada uno es usado íntegramente para almacenar datos, sin necesidad de almacenar un puntero como en el caso de la asignación enlazada.

Un problema que plantea la asignación indexada es el gasto extra de espacio para mantener los bloques índice. El caso más extremo se da cuando tenemos un fichero que sólo ocupa un bloque de datos; con este método de asignación necesitaremos dos bloques, el de datos y el índice.

Otra cuestión es cómo de grande puede ser el bloque índice. Puesto que cada fichero tiene su propio bloque índice, nos interesará que éste sea lo más pequeño posible, para no desperdiciar mucho espacio. Sin embargo, si el bloque índice es demasiado pequeño, no será capaz de mantener todos los punteros necesarios en el caso de que tengamos ficheros grandes. Por tanto, habrá que diseñar algún tipo de mecanismo para tratar este problema. Veamos algunas posibilidades:

Esquema enlazado Normalmente un bloque índice es un bloque de disco. Si te-

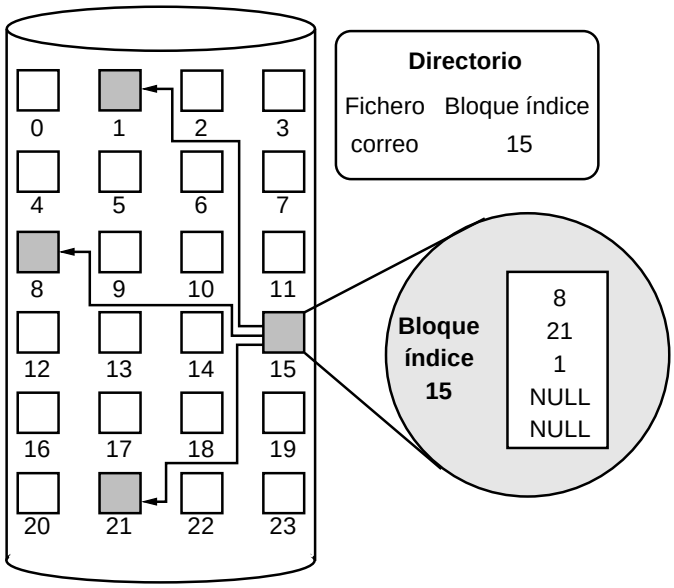

Figura 4.11: Asignación indexada

nemos un fichero grande que necesita más de un bloque índice para mantener las direcciones de los bloques que lo componen, los bloques índices se enlazan entre sí. Es decir, para ficheros pequeños sólo se utilizará un bloque índice, para ficheros grandes se utilizarán tantos como sean necesarios, todos ellos enlazados entre sí a través de la última entrada de cada bloque índice.

Índice multinivel Una variante del método anterior es usar un bloque índice separado para apuntar a los bloques índice. Para acceder a un bloque, el sistema operativo utiliza el índice de primer nivel para encontrar el índice de segundo nivel, y éste para encontrar los bloques de datos. Este esquema se podría continuar con un tercer o cuarto nivel, dependiendo del tamaño máximo de los ficheros que se quiera manejar.

Esquema combinado Otra alternativa es utilizar un bloque en el que se mantienen punteros directos a bloques de datos y punteros indirectos. Éstos apuntan a bloques que contienen punteros a bloques de datos.

4.4.3. Gestión del espacio libre

El sistema de gestión de ficheros debe llevar el control de qué bloques están libres y cuáles ocupados. Normalmente se utilizan dos técnicas, **mapas de bits** y **listas enlazadas**.

4.4.3.1. Mapa de bits

Frecuentemente, el control del espacio libre se lleva a cabo mediante un **mapa de bits**, donde cada bloque se representa con un bit. Si el bloque está libre, el bit está a 0, y si está asignado, el bit estará a 1. La figura 4.12 ilustra esta técnica. El tamaño del mapa dependerá del número de bloques del disco. Así, para un disco de 1 GiB, con bloques de 4 KiB, serán necesarios 262144 bits. Por tanto, se necesitarían 8 bloques para almacenar el mapa de bits.

Bloques libres: 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 27
Mapa de bits: 11000011000000111001111110001111 1...

Figura 4.12: Mapa de bits para la gestión del espacio libre

La principal ventaja de este método es que es relativamente simple y eficiente para encontrar n bloques libres consecutivos. Sin embargo, los mapas de bits son ineficientes a menos que se mantenga el mapa completo en memoria principal.

4.4.3.2. Lista enlazada

Otra posibilidad es enlazar todos los bloques libres del disco usando punteros. Este método no gasta mucho espacio porque no necesita una tabla de asignación del disco, sino simplemente un puntero al primer bloque libre. El resto de los punteros se almacenan en los propios bloques libres (figura 4.13).

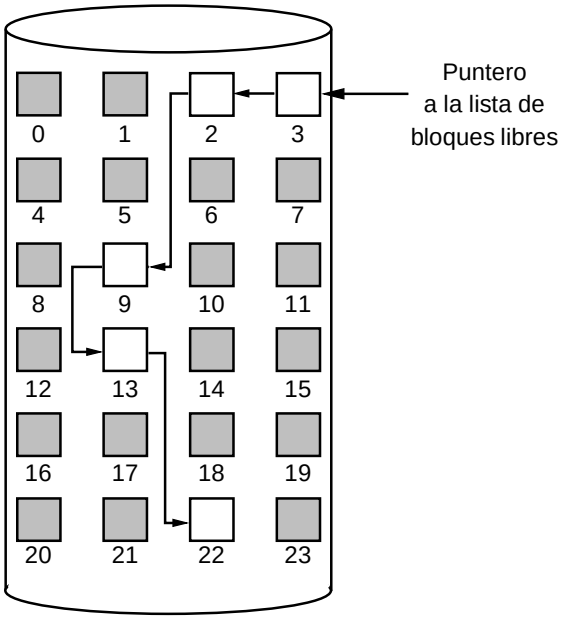

Figura 4.13: Lista de espacio libre enlazada

Este método no es muy eficiente ya que para recorrer la lista tenemos que leer cada bloque, lo que requiere una operación de E/S. Por este motivo, para optimizar el tiempo de búsqueda de un conjunto de bloques libres, se han realizado diversas implementaciones. Una primera forma es el **agrupamiento**. En este caso se usa un primer bloque que almacena las direcciones de n bloques libres (tantos como quepan en él), donde la última dirección apunta al siguiente bloque que contiene direcciones de bloques libres, y así sucesivamente. De este modo, se hallan rápidamente las direcciones de un gran número de bloques libres.

El inconveniente es el espacio que requiere esta técnica. Supongamos que la implementamos en el mismo disco visto en el mapa de bits, es decir, un disco de 1 GiB, con bloques de 4 KiB, donde cada dirección requiere 32 bits. Cada bloque puede contener un total de 1024 direcciones de bloques. Luego para poder contener los 262144 bloques del disco, serán necesarios 257 bloques.

Otra posibilidad consiste en aprovechar los bloques libres contiguos. Es lo que se denomina **recuento**. Esta técnica almacena la dirección de un bloque libre junto con el número de bloques libres contiguos. De este modo, cada entrada requiere más espacio que antes, pero la lista será más corta, dado que el contador generalmente será mayor que uno.

4.4.4. Implementación de directorios

Un directorio es un tipo de fichero especial, que contiene información de la ubicación de los distintos ficheros que se catalogan en él. La implementación de los directorios es muy importante en el sistema de gestión de ficheros, debido a que toda operación que se quiera realizar sobre un fichero requiere acceder previamente al directorio.

El método más sencillo de implementación de un directorio es el de una lista lineal de sus entradas. Así, crear un fichero consiste en añadir una entrada al final de la lista. Borrar un fichero consiste en eliminar su entrada de la lista; ese espacio puede ser posteriormente utilizado por un nuevo fichero, o bien pasar la última entrada a dicha posición para reducir el tamaño del directorio. El problema que plantea esta implementación es la operación de búsqueda de un fichero, ya que al no estar la lista ordenada, tenemos que hacer una búsqueda secuencial en el directorio. El mantenerla ordenada para facilitar la búsqueda nos complicaría los métodos de inserción y borrado de ficheros.

Una alternativa que permite mejorar los tiempos de búsqueda sin perjudicar las operaciones de inserción y borrado, sería la utilización de una tabla de dispersión. El inconveniente de esta implementación es que la tabla de dispersión presenta un tamaño fijo, y la función de dispersión depende de éste.

Con independencia del tipo de implementación que tenga el directorio, los sistemas operativos actuales suelen mantener en memoria principal, mediante la caché de disco, la información de los directorios de uso más reciente.

Otro de los aspectos a considerar es la estructura de las distintas entradas. Ésta depende del tipo de asignación empleado (apartado 4.4.2) pues determina la mínima información que debe residir en ellas. Sin embargo, la estructura de éstas varía mucho de un sistema a otro.

En la figura 4.14 se muestra una entrada de un directorio en MS-DOS, indicándose entre paréntesis el número de bytes de cada componente. Cada entrada tiene 32 bytes que se emplean para guardar la siguiente información:

- Nombre del fichero.
- Extensión.
- Atributos.
- Fecha y hora de creación o de última modificación.
- Número del primer bloque del fichero.
- Tamaño.

El número del primer bloque del fichero se emplea como índice en la FAT para conocer el resto de los bloques que forman el fichero. Los directorios en MS-DOS son ficheros y pueden contener un número arbitrario de entradas. La estructura de la entrada de un directorio en LINUX se verá en el apartado 4.8.4.

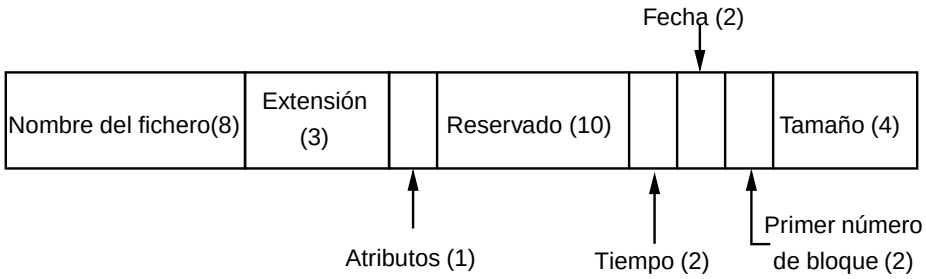

Figura 4.14: Entrada de un directorio en MS-DOS

4.5. Fiabilidad del sistema de ficheros

El sistema de ficheros, además de llevar el control del espacio libre y la asignación de bloques a los ficheros, debe proporcionar mecanismos de protección de la información almacenada en ellos. Estas técnicas no evitan la destrucción física de los datos, pero pueden permitir recuperarlos, cuando se origina su pérdida, y ayudarnos a mantener la consistencia de la información almacenada. En este apartado vamos a analizar algunos de los aspectos implicados en la protección del sistema de ficheros.

4.5.1. Copias de seguridad

Es importante hacer copias de seguridad de los discos con frecuencia. El medio empleado para realizar la copia dependerá del volumen de información a guardar, así se podrán emplear discos flexibles, discos duros, CD-ROM, cintas magnéticas, etc.

Las copias de seguridad pueden ser de distinto tipo dependiendo de la información que se copie; se pueden hacer copias de toda la información que contiene el disco, o bien sólo de aquellos ficheros que han sido modificados desde la última copia de seguridad; éstas son las llamadas **copias incrementales**. Para poder realizarlas, el sistema debe guardar para cada fichero la fecha en que se realizó la última copia.

4.5.2. Consistencia del sistema de ficheros

Muchos sistemas de ficheros cuando realizan operaciones sobre los ficheros modifican la copia existente en memoria principal y posteriormente actualizan la memoria secundaria. Si antes de realizar esta actualización se produce un fallo en el sistema, la información en memoria secundaria puede quedar en un estado inconsistente ya que no coincide con las modificaciones hechas por el usuario. Este problema se agrava si afecta a las estructuras de control del sistema de ficheros.

Un sistema de ficheros se dice que es **consistente** cuando todas las modificaciones realizadas por el usuario se reflejan en el disco.

El sistema operativo puede proporcionar alguna utilidad que nos permita verificar la consistencia del sistema de ficheros. En el apartado 4.8.8 se estudia la herramienta que incorpora el sistema operativo LINUX.

4.6. Rendimiento del sistema de ficheros

Al estudiar cómo se puede mejorar el rendimiento de un sistema de ficheros vamos a tener en cuenta dos aspectos fundamentales que influyen en la mejora de éste:

- Reducción del número de accesos al disco, que se consigue mediante la caché de disco (estudiado en el tema anterior).
- Reducción del movimiento del brazo del disco, esto lo podemos conseguir por medio de la planificación de las peticiones de acceso al disco (estudiados en el tema anterior) y agrupando los bloques que forman parte de un fichero.

La forma de realizar la agrupación de bloques va a depender tanto del esquema de asignación como de la forma en que se lleva el control del espacio libre. Si se emplea una asignación contigua, el propio esquema nos proporciona el agrupamiento de los bloques. Las otras estrategias de asignación, enlazada o indexada, presentan

problemas de dispersión de bloques, por lo que si el fichero necesita más bloques, hemos de intentar asignárselos cercanos al resto. Por tanto, el objetivo es evitar la desfragmentación de los ficheros por todo el disco. Esto dependerá del mecanismo empleado para gestionar el espacio libre. Si se emplea un mapa de bits, es fácil la elección de un bloque libre que esté lo más cerca posible del anterior. Si se utiliza una lista libre, el agrupamiento es más complejo.

Otro obstáculo para el rendimiento en los sistemas de asignación indexada es que se requiere un acceso extra al disco para leer el bloque índice. La ubicación de los bloques índice con respecto a los bloques de datos va a influir en el rendimiento. Si se sitúan en un extremo del disco, como se muestra en la figura 4.15(a), la distancia media entre el bloque índice y los bloques de datos será la mitad del número de pistas, lo que requiere desplazamientos largos.

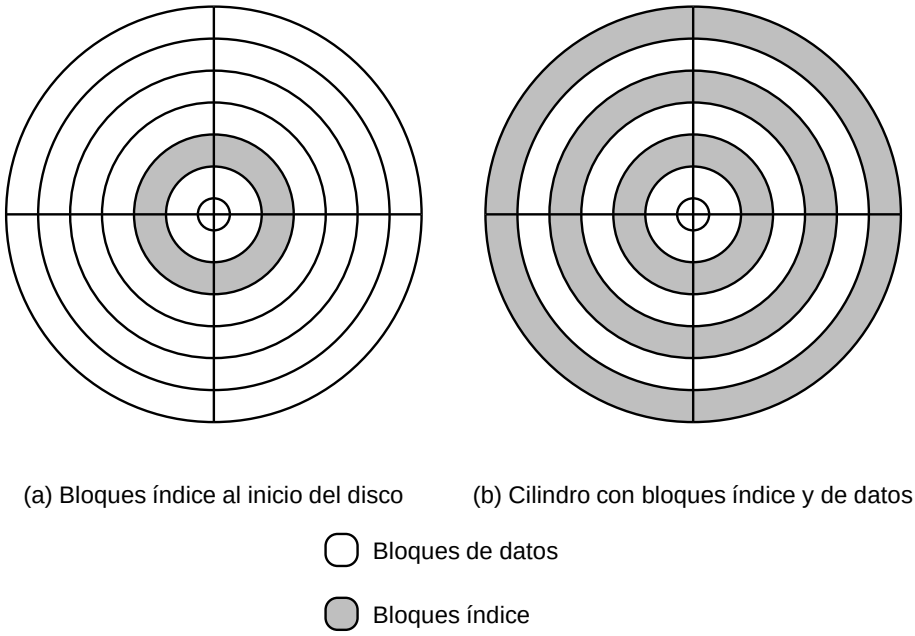

Figura 4.15: Optimización del rendimiento en asignación indexada

Si el disco se divide en grupos de cilindros, de forma que cada uno disponga de un conjunto de bloques índice y de datos, se puede intentar asignar a un fichero los bloques de datos y el bloque índice en el mismo grupo de cilindros. Si no fuese posible, se elegiría del grupo de cilindros más cercano. Esto es lo que se muestra en la figura 4.15(b).

4.7. Sistema de ficheros en LINUX

4.8. Sistemas de ficheros en LINUX

Hasta la aparición de la versión 2.4 del núcleo de Linux, éste proporciona un sistema de ficheros nativo, `ext2` y varios sistemas de ficheros por compatibilidad con otros sistemas operativos, tales como fat, vfat, ntfs, sysv, nfs, minix, etc. Pero a partir de la aparición de dicha versión del núcleo, éste ofrece una gran selección de sistemas de ficheros, ext2, ext3, ReiserFS, JFS y XFS.

En este apartado vamos a considerar las características del sistema de ficheros etx2, y posteriormente se describirán las principales diferencias que presentan estos nuevos sistemas de ficheros con respecto a éste.

4.8.1. Estructura lógica

Como la mayoría de los sistemas operativos modernos, LINUX organiza su sistema de ficheros como una jerarquía de directorios, que se denomina **árbol de directorios**. Como se muestra en la figura 4.16, existe un directorio especial, llamado **directorio raíz**, que está situado en la parte más alta de la jerarquía. Cada directorio del sistema de ficheros `ext2` puede contener tres clases de ficheros:

Ficheros ordinarios Contienen datos.

Ficheros especiales Proporcionan acceso a los dispositivos de E/S.

Directorios Contienen información acerca de conjuntos de ficheros y se utilizan para localizar un fichero a partir de su nombre.

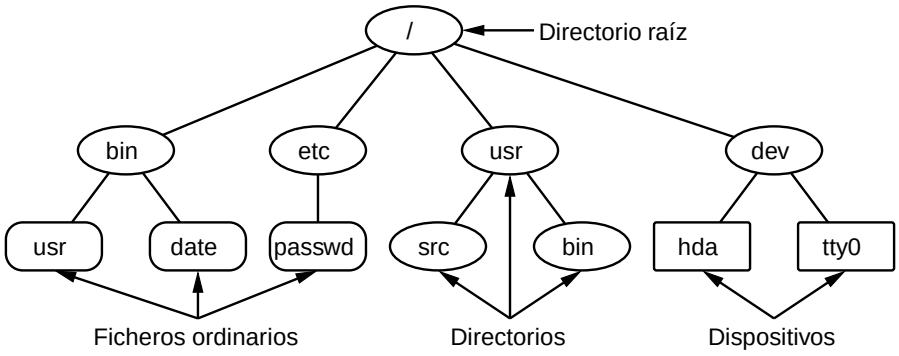

Figura 4.16: Jerarquía de directorios y ficheros

Esa jerarquía con un único directorio raíz puede estar formada por varios sistemas de ficheros residentes en distintas particiones de disco duro u otros dispositivos de

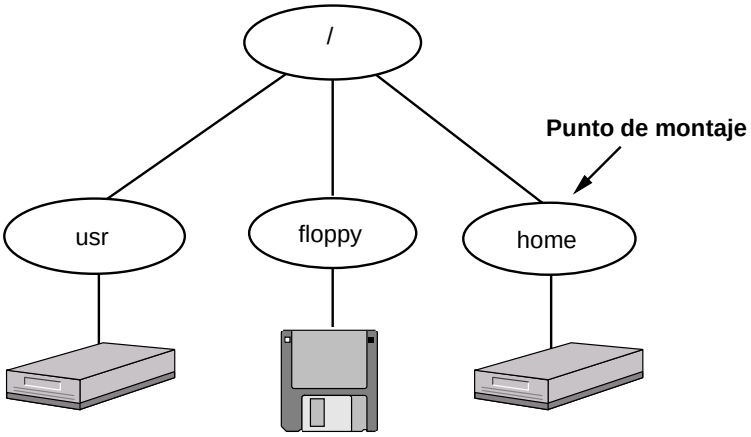

Figura 4.17: Sistema de ficheros constituido por tres dispositivos

bloque. La forma de conseguir esa estructura lógica de un único árbol de directorios es a través de las operaciones de **montaje**[1] de sistemas de ficheros. De este modo:

- Existe un único sistema de ficheros raíz que se monta cuando el sistema arranca.

- Los demás sistemas de ficheros se montan en directorios que deben existir previamente en la jerarquía. Los directorios donde se montan los sistemas de ficheros se denominan **puntos de montaje**. Éste es el único modo que tiene el usuario para acceder a los ficheros que hay en una partición de disco duro, CD-ROM, disco flexible, cinta, etc. Si el punto de montaje elegido tiene ya un contenido, éste queda oculto hasta que se desmonte el sistema de ficheros.

Por tanto, el usuario ve un único sistema de ficheros, pero éste está realmente compuesto por varios sistemas de ficheros físicos, cada uno de ellos situado en un dispositivo o partición diferente, pudiendo ser incluso sistemas distintos. La figura 4.17 muestra esta situación. El sistema `VFS`[2] es el que se encarga de crear ese árbol de directorios, de forma que cuando se accede a un fichero, el sistema `VFS` transmite la petición del usuario al sistema de ficheros físico correspondiente. Podemos decir, que el sistema `VFS` se comporta como un sistema de sistemas de ficheros.

Cuando se para el sistema se desmontan todos los sistemas de ficheros. Del mismo modo, si se dispone de un sistema montado en un dispositivo extraíble (disquete o CD-ROM) y queremos extraerlo, es necesario desmontarlo previamente.

[1] Montar (*mount*): Integrar en la jerarquía de directorios.
[2] *Virtual File System*.

4.8.2. Estructura física del fichero

El sistema de ficheros `ext2` presenta un esquema de asignación indexada, por lo que los bloques de datos de un fichero pueden estar dispersos por todo el disco. Para llevar el control de cuáles son los bloques que pertenecen a un fichero, se emplea una estructura de control que recibe el nombre de **nodo índice** o **nodo-i**. Éste contiene información de control del fichero, así como sobre los bloques que ocupa éste. Está diseñado como muestra la figura 4.18, ocupando 128 bytes.

Entre la información de control que posee sobre el fichero se encuentra:

- Identificación del propietario del fichero, y el grupo de usuarios al que pertenece.
- Tipo de fichero (fichero ordinario, especial o directorio).
- Bits de protección.
- Fechas de creación, último acceso y última modificación.
- Número de enlaces duros al fichero.
- Tamaño del fichero en bytes y en bloques.

Además, el nodo índice incluye la **tabla de contenidos** utilizada en la localización de los datos de un fichero en el disco. Su diseño permite mantener una estructura del nodo índice pequeña y, a la vez, permitir ficheros grandes.

La tabla de contenidos dispone de 15 punteros, de los cuales 12 de ellos apuntan directamente a bloques de datos del fichero. Así, para ficheros pequeños (no más de 12 bloques) sólo se necesita del nodo índice. Para ficheros de mayor tamaño, las tres últimas entradas de la tabla de contenidos apuntan a bloques de direcciones de bloques de datos. Así, el puntero indirecto simple apunta a un bloque que contiene direcciones de bloques de datos, el puntero indirecto doble apunta a un bloque que contiene direcciones de otros bloques, que a su vez contienen direcciones de bloques de datos y así sucesivamente.

El número de direcciones que hay dentro de un bloque depende del tamaño del bloque. Por ejemplo, si se dispone de bloques de 4 KiB, dado que las direcciones de éstos son de 32 bits, un bloque podría apuntar a otros 1024 bloques. De este modo, el tamaño máximo de un fichero sería:

$$(12 + 1024 + 1024^2 + 1024^3) \times 4 \text{ KiB} \approx 4 \text{ TiB}$$

Aunque es posible direccionar más de 4 TiB, dado que el nodo índice almacena la longitud del fichero en bytes, éste se limita a 4 TiB.

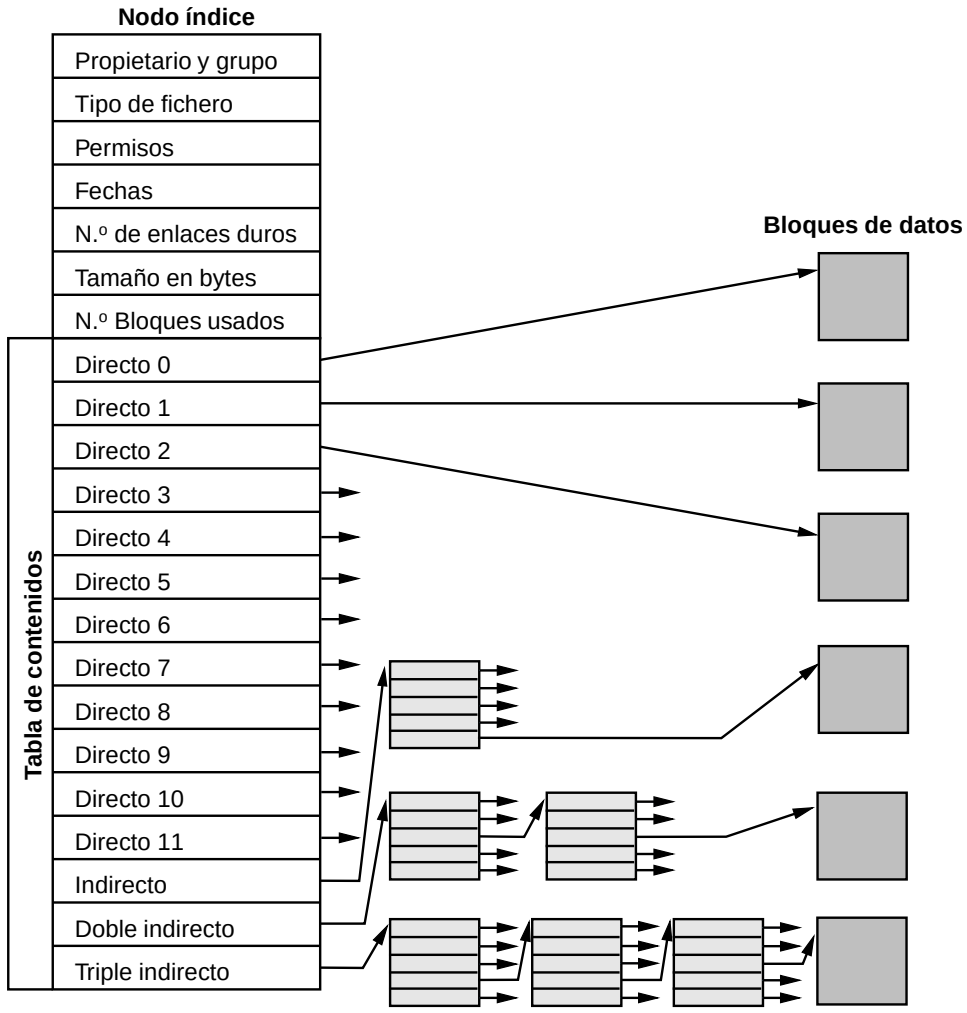

Figura 4.18: Estructura del nodo índice del sistema de ficheros `ext2`

4.8.3. Estructura física de un sistema de ficheros

El sistema de ficheros `ext2` divide la partición o dispositivo de bloques donde reside en **grupos de bloques** o **cilindros**, como se muestra en la figura 4.19.

Sector de arranque	Grupo de bloques 0	Grupo de bloques 1	... ...	Grupo de bloques N

Figura 4.19: Estructura física del sistema de ficheros `ext2`

El **sector de arranque** ocupa el principio de la partición o dispositivo de bloques y puede contener el código de arranque (*bootstrap*) que es leído por la máquina para iniciar el sistema operativo.

La estructura de un grupo de bloques se representa en la figura 4.20. En ella se

observa que cada uno contiene:

Superbloque Estructura de control del sistema de ficheros.

Descriptores de grupos Estructuras de control de los grupos de bloques.

Mapas de bits Permiten una gestión rápida tanto de los bloques de datos como de los nodos-i disponibles en el sistema de ficheros.

Tabla de nodos-i En el sistema de ficheros `ext2` cada fichero se representa por un nodo-i. Esta tabla contiene los nodos-i que se almacenan en el grupo de bloques.

Bloques de datos Contienen los datos de los ficheros.

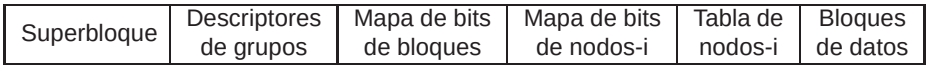

Figura 4.20: Estructura de un grupo de bloques

Podemos comprobar que aunque cada grupo presenta la misma estructura, sólo los grupos de bloques impares son los que contienen una copia redundante de las estructuras de control del sistema de ficheros (superbloque y descriptores de grupos). De todas ellas, el núcleo sólo utiliza la que se encuentra en el grupo 1, las otras sólo se emplean como copias de seguridad, permitiendo aumentar la fiabilidad.

Además, el sistema de ficheros se encuentra repartido entre los distintos grupos de bloques, de forma que cada uno contiene una parte de los nodos-i y bloques de datos del total disponible.

4.8.3.1. El superbloque

El sistema de ficheros se representa globalmente mediante el superbloque. Esta estructura contiene toda la información necesaria para que el núcleo pueda llevar el control del sistema de ficheros. La información contenida en el superbloque se puede agrupar en tres tipos:

- Parámetros fijos que se determinan durante la creación del sistema de ficheros y no pueden variar.
- Parámetros ajustables que se pueden modificar una vez creado el sistema de ficheros.
- Información del estado actual del sistema de ficheros.

Parámetros fijos

- Número mágico.
- Tamaño del bloque.
- Número de nodos-i y de bloques de datos en el sistema de ficheros.
- Número de nodos-i y de bloques de datos por grupo.
- Primer nodo-i.

Parámetros ajustables

- Número de bloques reservados para el administrador.
- Identificación del usuario y grupo de usuarios que pueden hacer uso de los bloques reservados al administrador.
- Número de veces que se tiene que montar para realizar una comprobación de la consistencia del sistema de ficheros.
- Número de segundos que puede transcurrir sin realizarse una comprobación de la consistencia del sistema de ficheros.
- Comportamiento del núcleo cuando se encuentra un error en el sistema de ficheros.

Estado actual del sistema de ficheros

- Número de bloques libres.
- Número de nodos-i libres.
- Fecha del último cambio efectuado en el sistema de ficheros.
- Fecha en la que se montó por última vez.
- Número de veces que ha sido montado.
- Fecha de la última comprobación de consistencia del sistema.
- Tipo de error detectado en el sistema de ficheros.

Cuadro 4.2: Información del superbloque

La tabla 4.2 muestra los distintos tipos de información que se puede encontrar en el superbloque.

Con respecto a los parámetros fijos, el número mágico se emplea para identificar el tipo de sistema de ficheros en las operaciones de montaje. En el caso del sistema `ext2` es el `0xEF53`.

El tamaño del bloque puede ser 1, 2 ó 4 KiB. La elección de un valor u otro dependerá del tamaño del sistema de ficheros. Este parámetro es el valor clave para la determinación de la estructura física del sistema de ficheros. La tabla 4.3 muestra la relación existente entre el tamaño del bloque, el número de bloques por grupo y por nodo índice.

Tamaño del bloque	Nº bloques/grupo	Nº bloques/nodo-i
1 KiB	8192	4
2 KiB	16384	3
4 KiB	32768	2

Cuadro 4.3: Influencia del tamaño del bloque en la estructura física del sistema `ext2`

Un conjunto de información importante es la que emplea la orden e2fsck, encargada de comprobar la consistencia del sistema de ficheros. Así, cada vez que el sistema se monta se incrementa una variable, de forma que cuando llegue a un número máximo se realiza una comprobación del sistema. Este límite de operaciones de montaje se establece por omisión a 20. Con objeto de evitar que éstas se hagan muy tardías, en el superbloque se almacena también el número de segundos que debe transcurrir como máximo entre dos comprobaciones consecutivas.

Un aspecto importante en el sistema `ext2` es la posibilidad de reservar una serie de bloques del disco para el administrador del sistema. Por omisión se reserva el 5 % de los bloques totales del sistema de ficheros. Esto permite que el sistema no se llene al 100 %, puesto que esto podría provocar problemas en su funcionamiento. Además, es posible especificar qué usuario o grupo de usuarios pueden hacer uso de esos bloques.

4.8.3.2. Descriptores de grupo

Cada grupo de bloques se representa en el sistema por el **descriptor de grupo**. Todos los descriptores de grupo de un sistema de ficheros forman una **tabla de descriptores**. Ésta se almacena, al igual que el superbloque, en los grupos de bloques impares por motivos de fiabilidad, aunque sólo se emplea la copia del grupo 1. La información que se encuentra en un descriptor de grupo se puede agrupar en dos tipos:

- Parámetros fijos que se determinan durante la creación del sistema de ficheros y no pueden variar.

- Estadísticas de uso del grupo de bloques.

Parámetros fijos
■ Número de bloques ocupados por el mapa de bits de bloques.
■ Número de bloques del mapa de bits de nodos-i.
■ El número del bloque donde comienza la tabla de nodos-i para el grupo de bloques.
Estadísticas de uso
■ Número de bloques y nodos-i libres en el grupo de bloques.
■ Número de directorios existentes en el grupo de bloques.

Cuadro 4.4: Información del descriptor de grupo

La tabla 4.4 muestra la información que se puede encontrar en un descriptor. Las estadísticas sobre el uso de los recursos de un grupo de bloques son empleadas por el núcleo para distribuir de forma homogénea los ficheros y directorios entre los distintos grupos de bloques.

4.8.3.3. Mapas de bits

La gestión de los recursos disponibles del sistema de ficheros, nodos índice y bloques de datos, se realiza mediante mapas de bits. Cada grupo de bloques posee un mapa de bits para los nodos índice y otro para los bloques de datos que se encuentran en él.

El tamaño del mapa de bits viene especificado en el descriptor del grupo. Normalmente se suele emplear un bloque, puesto que si disponemos de bloques de 1 KiB, el mapa de bits puede gestionar hasta $1024 \times 8 = 8192$ bloques o nodos índice en el grupo de bloques.

4.8.4. Directorios

Es un fichero cuyos datos son una secuencia de entradas de longitud variable, que se maneja como una lista enlazada. La estructura de cada entrada se muestra en la figura 4.21.

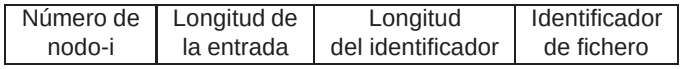

Figura 4.21: Entrada de un directorio del sistema de ficheros `ext2`

El identificador de fichero le da un nombre a éste dentro de un directorio. El número máximo de caracteres que puede tener un identificador de fichero es de 255.

Con objeto de facilitar la gestión de la lista de entradas, el tamaño de éstas debe ser múltiplo de 4 bytes, rellenándose de «ceros» los espacios libres. Por este motivo, cada entrada almacena tanto la longitud del identificador como la de la propia entrada.

Las entradas de un directorio se almacenan igual que los datos de cualquier fichero ordinario, usando la estructura de nodo-i y los niveles de bloques directos e indirectos.

Las dos primeras entradas de un directorio son siempre el directorio actual y el padre, denotados por . y .., respectivamente. Estas entradas nunca se pueden eliminar de un directorio.

4.8.5. Enlaces

Una entrada de un directorio es un enlace a un fichero, dado que cada entrada apunta a un nodo-i. En el sistema de ficheros `ext2` puede haber múltiples enlaces a un mismo fichero, es decir, podemos tener un fichero con varios nombres. Todos los enlaces a un mismo fichero tienen el mismo número-i. Los enlaces pueden residir en directorios diferentes (figura 4.22(a)). Como sólo hay una copia del fichero, cualquier cambio que se haga en éste a través de cualquiera de sus enlaces es visible para todos los demás.

Estos enlaces se suelen denominar **enlaces duros** y presentan dos restricciones; no se pueden crear enlaces duros entre directorios, ni entre dos ficheros que estén en sistemas de ficheros diferentes. La razón para esta última restricción es que cada sistema de ficheros tiene su propio conjunto de nodos-i, por tanto, los números-i sólo son únicos dentro de un mismo sistema de ficheros.

El sistema de ficheros `ext2` proporciona otra forma de enlace, el simbólico. Cuando se crea un **enlace simbólico** a un fichero se está creando un nuevo fichero, cuyo contenido es el camino del fichero enlazado. Así, cuando hacemos referencia al enlace simbólico el sistema averigua el camino del fichero al que realmente vamos a acceder (figura 4.22(b)).

El sistema de ficheros `ext2` proporciona dos tipos de enlaces simbólicos: lento y rápido. El enlace lento crea una entrada de directorio, y almacena el camino en un bloque de datos, apuntado por el nodo-i.

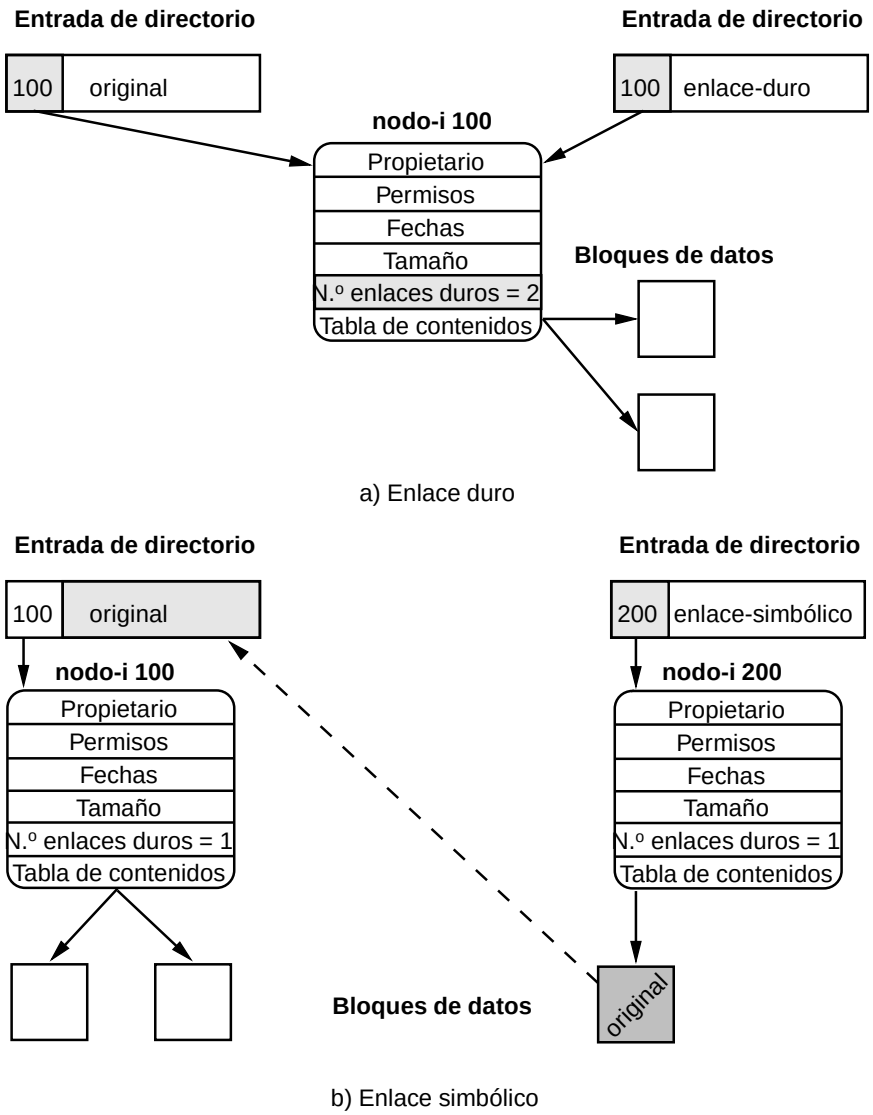

Figura 4.22: Enlaces duros y simbólicos en el sistema de ficheros `ext2`

El enlace rápido consiste en que el nodo-i en lugar de tener una tabla de contenidos almacena el camino del enlace en esa tabla. El nombre del fichero enlazado ha de tener una longitud menor de 60 caracteres, que es el espacio disponible en la tabla de contenidos (15 entradas de 4 bytes cada una).

Los enlaces simbólicos no presentan las restricciones de los enlaces duros a la hora de crearlos. Se pueden establecer entre ficheros que estén en sistemas de ficheros diferentes, y también entre directorios, lo cual permite a los administradores de sistemas mover parte de la jerarquía de directorios de un sitio a otro de forma transparente para los usuarios.

El inconveniente de los enlaces simbólicos es que consumen más espacio que los duros, puesto que necesitan un nodo-i y pueden necesitar un bloque de datos.

4.8.6. Búsqueda de un fichero

Un nombre de fichero en `ext2` es una sucesión de identificadores de ficheros separados por el carácter `/`. Cuando se le proporciona un nombre de fichero, el sistema debe localizar a partir de éste su nodo-i para poder acceder a sus bloques de datos. Para ello debe realizar una búsqueda a lo largo de todos los directorios que componen el nombre del fichero.

Supongamos que le damos el nombre absoluto `/home/linux/ext2`. El primer nodo-i que necesita es el del directorio raíz, el cuál se encuentra en el superbloque. A partir de éste accede a sus bloques de datos donde se encuentran almacenadas sus entradas.

A continuación busca la entrada `home` en el directorio raíz con el fin de hallar su nodo-i y localizar las entradas del directorio para poder buscar el siguiente componente del nombre, `linux`. Cuando encuentra la entrada `linux`, ésta tiene el nodo-i del directorio `/home/linux`. A partir de este nodo índice se puede hallar el contenido del directorio y buscar `ext2`. Con su nodo-i podemos acceder a los bloques del fichero deseado. Este proceso de búsqueda se ilustra en la figura 4.23.

Los nombres de ruta relativos se buscan de la misma forma que los absolutos, sólo que comenzando desde el directorio de trabajo y no desde el directorio raíz. Todo directorio tiene entradas para los ficheros . y .. que se colocan ahí cuando se crea el directorio. La entrada . tiene el número de nodo-i del directorio actual y la entrada .. tiene el número de nodo-i del directorio padre. Por tanto, un procedimiento que busca a `../fps/prog.c` simplemente busca a .. en el directorio de trabajo, halla el número de nodo-i del directorio padre y rastrea ese directorio hasta encontrar `fps`.

4.8.7. Optimización del sistema de ficheros

El sistema de ficheros `ext2` utiliza un esquema de asignación indexada, lo que puede producir que los bloques de un fichero se encuentren muy dispersos en el disco, con el consiguiente incremento del tiempo de acceso debido al mayor desplazamiento de la cabeza lectora.

Con objeto de reducir los tiempos de acceso y, por consiguiente, mejorar el rendimiento, se realiza una optimización que consiste en intentar almacenar en el mismo grupo de bloques el nodo índice y los bloques de datos de un mismo fichero. Así, si se van a asignar los primeros bloques a un fichero se inicia la búsqueda dentro del mismo grupo donde se encuentra su nodo índice, y si se va a ampliar el tamaño del fichero, se inicia la búsqueda a partir del último bloque asignado al fichero. De este modo, cuando se intenta acceder al fichero el movimiento de la cabeza lectora es el menor posible.

El sistema `ext2` también realiza una preasignación de bloques. Para ello aprovecha la estructura del mapa de bits para la gestión de bloques libres. Dado que un byte del mapa lleva el control de 8 bloques, cuando se busca un bloque libre para

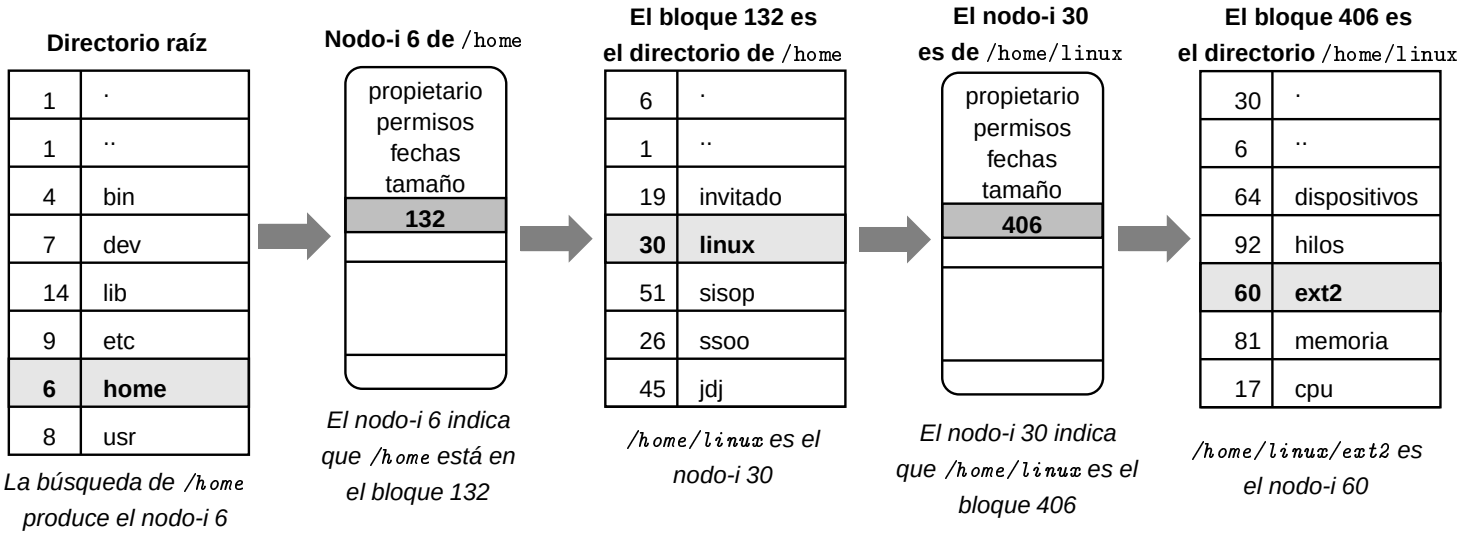

Figura 4.23: Las etapas en la búsqueda de `/home/linux/ext2`

un fichero se intenta asignar 8 bloques libres consecutivos, de forma que en el mapa de bits de bloques se representa con un byte de 8 bits libres.

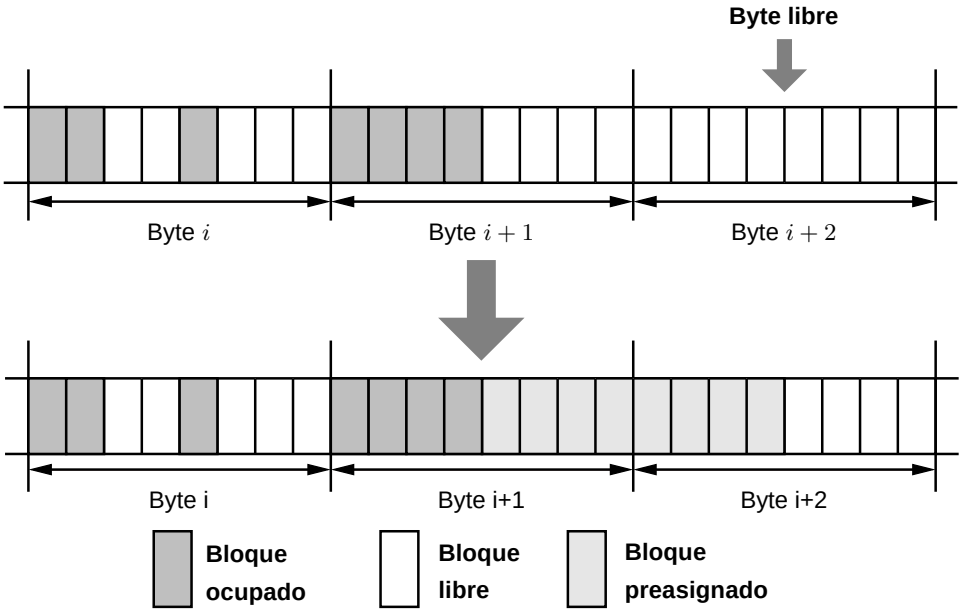

Figura 4.24: Preasignación de bloques

La búsqueda de bloques libres se realiza en dos fases. Primero se busca un byte libre completo en el mapa de bits. Si se encuentra, se realiza una búsqueda hacia atrás para no dejar espacio libre entre el byte y el último bloque asignado. Esta búsqueda en bytes permite que el fichero disponga de bloques adyacentes acelerando el acceso secuencial al fichero. Si no se encuentra un byte libre se buscan 8 bits libres en el mapa de bits para asignárselos al fichero.

La figura 4.24 muestra el resultado de una búsqueda de un bloque libre. De los tres bytes del mapa de bits que se ilustran, el byte $i+2$ es el que se encuentra libre. Una vez que lo localiza, retrocede hacia atrás aprovechando el espacio libre anterior.

Esta preasignación es simplemente una reserva de bloques, puesto que cuando el fichero se cierra los bloques preasignados se devuelven al mapa de bits como libres. De este modo, cualquier fichero si necesita más bloques de datos puede ocuparlos.

4.8.8. Consistencia del sistema de ficheros

Para verificar la consistencia, el sistema `ext2` proporciona la orden `e2fsck` cuyo funcionamiento se describe a continuación.

Permite verificar tanto los bloques como los nodos-i. Para verificar la consistencia de un bloque tenemos que ver, en primer lugar, en qué estado se puede encontrar. Un bloque puede estar libre o bien pertenecer a algún fichero. En el caso de no

encontrarse en ninguna de esas situaciones excluyentes, estamos en un sistema inconsistente.

Para probar la consistencia del sistema se crea una tabla con dos contadores por bloque, valiendo ambos inicialmente cero. El primer contador lleva la cuenta de las veces que un bloque está presente en un fichero, es decir, los que están ocupados; el segundo registra la frecuencia con la cual está presente en el mapa de bits, es decir, los que están libres.

Después el programa lee todos los nodos-i. Comenzando desde un nodo-i, es posible construir una lista de todos los números de bloque que se utilizan en el fichero correspondiente. Conforme se lee cada número de bloque, se incrementa el primer contador. Después el programa examina el mapa de bits, con objeto de encontrar todos los bloques que no están en uso, incrementado el segundo contador.

Una vez finalizado, se pueden presentar las siguientes situaciones:

- Cada bloque tendrá un 1 en un único contador, como se ilustra en la figura 4.25(a). En este caso el sistema de ficheros es consistente.
- Sin embargo, si ha ocurrido un fallo, la tabla podría aparecer como en la figura 4.25(b), donde el bloque 5 no aparece en ninguna de ellas. Estos bloques que faltan no hacen un daño real, pero desperdician espacio y, por lo tanto, reducen la capacidad del disco. La solución a este problema es fácil, simplemente el programa e2fsck lo marca como libre.
- El error más grave que puede aparecer es que un bloque de datos pertenezca a dos o más ficheros, como se muestra en la figura 4.25(c) con el bloque 7. Si alguno de estos ficheros se borra, el bloque 7 se liberará, lo cual nos llevará a una situación en la que el bloque está libre y en uso al mismo tiempo. La acción que debe llevar a cabo el programa e2fsck consiste en copiar el contenido del bloque 7 a un bloque libre y asignar un bloque a cada fichero. Los propietarios de los ficheros deberán ser informados del error para que inspeccionen el daño causado.

Además de revisar los bloques, se debe verificar el sistema de directorios. En este caso, también se utilizan dos contadores, pero por nodo-i, en vez de por bloques. El programa e2fsck realiza un recorrido recursivo de todos los directorios del sistema de ficheros, comenzando en el directorio raíz. Cada vez que encuentra un fichero en un directorio incrementa el primer contador y almacena en el segundo el número de enlaces duros que aparece en su nodo-i.

Cuando termina compara los dos contadores, que si el sistema es consistente deben coincidir. Pueden aparecer dos tipos de errores, el contador de enlaces en el nodo-i puede ser mayor o menor que el número de entradas en los directorios.

Si el número de enlaces es mayor, puede provocar un desperdicio de espacio ya que si se borran todos los enlaces al fichero, tanto el nodo-i como los bloques de datos seguirían marcados como ocupados. Si el número de enlaces es menor el error

Bloque	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Uso	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Libre	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(a) Estado consistente

Bloque	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Uso	1	1	0	1	0	0	1	1	1	0	0	1	1	1	0	0
Libre	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(b) Falta un bloque

Bloque	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Uso	1	1	0	1	0	1	1	2	1	0	0	1	1	1	0	0
Libre	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(c) Bloque de datos duplicado

Figura 4.25: Estados de consistencia de un sistema de ficheros `ext2`

es más grave, puesto que podría desaparecer el fichero sin borrar todos sus enlaces. La solución en ambos casos consiste en almacenar en el número de enlaces del nodo-i el valor del contador que hemos obtenido.

Estas dos operaciones, la verificación de bloques y la de directorios, a menudo se integran por razones de eficiencia (es decir, sólo se requiere un paso por los nodos-i).

4.8.9. Características de los nuevos sistemas de ficheros de LINUX

4.8.9.1. Ext3

Ext3 fue concebido por Stephen Tweedie, y a diferencia del resto de los sistemas de ficheros de nueva generación, no está diseñado a partir de cero, sino que está basado en Ext2. La diferencia fundamental entre Ext2 y Ext3, es que éste último soporta *journaling*, el *journal* es una estructura interna del disco en la que se introducen los metadatos, es decir, la información sobre los datos que residen en el disco. Así, cuando hay que hacer una búsqueda de los metadatos no hace falta buscarlos en todo el disco, sino que basta con ver las entradas del *journal*. De este modo, se reduce considerablemente el tiempo que dura esta búsqueda. La actualización de un sistema de ficheros Ext2 a Ext3 es muy sencilla y fiable, y también se puede hacer en el sentido contrario.

4.8.9.2. ReiserFS

Las grandes ventajas de ReiserFS con respecto a Ext2 son:

- Aprovecha mejor el espacio del disco.

- Proporciona mejor rendimiento del acceso al disco.
- La recuperación es más rápida después de una caída del sistema.

ReiserSF aprovecha mejor el espacio del disco por varias razones; en el caso de los ficheros pequeños, los datos y la información sobre ellos se guardan juntos sin necesidad de mantener un puntero al nodo índice. Por otro lado, el espacio de disco no se asigna en unidades de 1 a 4 KiB, sino en la unidad necesaria. Otra ventaja es la asignación dinámica de nodos-i, con lo que no se limita la cantidad máxima de ficheros y directorios del sistema de ficheros.

El mejor rendimiento en el acceso al disco duro se logra porque para los ficheros pequeños tanto los datos como la información del nodo índice se guardan juntos, por lo que basta con un único acceso al disco duro para obtener toda la información necesaria.

El mecanismo de *journaling* hace que la comprobación de un sistema de ficheros se reduzca a unos pocos segundos, incluso para sistemas de ficheros grandes.

4.8.9.3. JFS

JFS (*Journalig File System*) fue desarrollado por IBM para AIX. La versión 1.0.0 para LINUX apareció en el año 2001. JFS está diseñado para cumplir las exigencias del entorno de un servidor de alto rendimiento en el que sólo cuenta el funcionamiento. Al ser un sistema de ficheros de 64 bits, soporta ficheros grandes y particiones LFS (*Large File Support*, lo cual es una ventaja para los entornos de servidor.

Debido al mecanismo de *journaling* proporciona una recuperación rápida después de una caída del sistema. También proporciona una mejor utilización del espacio del disco mediante la asignación dinámica de nodos-i.

También proporciona diversas estructuras de directorios. Para los directorios pequeños se permite el almacenamiento de su contenido en el nodo índice. En directorios más grandes utiliza árboles B+, que facilitan la administración de éstos.

4.8.9.4. XFS

XFS fue pensado orignalmente como sistema de ficheros para sistemas operativos IRIX. XFS es un sistema *journaling* de 64 bits de gran rendimiento adaptado a las necesidades extremas de los sistemas actuales. También está indicado para el trabajo con ficheros grandes y ofrece un buen rendimiento con el hardware de última generación. Algunas de las funciones más importates de XFS son las siguientes:

- Manejo de grupos de asignación.
- Alto rendimiento y eficiente administración del espacio del disco.

- Utiliza preasignación para evitar la fragmentación del sistema de ficheros.

En el momento de la creación de un sistema de ficheros XFS, el dispositivo de bloques en el que reside el sistema de ficheros, se divide en ocho o más campos de igual tamaño, denominados **grupos de asignación**. Cada grupo de asignación tiene sus propios nodos índice y sus bloques de datos. Se pueden considerar a estos grupos como sistemas de ficheros dentro de sistemas de ficheros. Este concepto de grupos de asignación autónomos cumple las exigencias de los sistemas con varios procesadores.

Una función única y característica de XFS es la denominada **asignación retardada**. XFS no guarda en los bloques de disco la información hasta que no es estrictamente necesario. Con esto ahorra operaciones de escritura, ya que algunos datos temporales no tendrán que ser escritos nunca porque han quedado obsoletos. Por otro lado también retrasa la decisión de dónde almacenar los datos para evitar la fragmentación. Por contra, es probable que sea mayor la pérdida de datos si se produce una caída del sistema.

Antes de la escritura de los datos en el sistema de ficheros, XFS reserva el espacio de disco necesario para un fichero que vaya a ser asignado. De esta forma, se reduce la fragmentación del sistema de ficheros y aumenta el rendimiento, ya que el contenido de los ficheros no está disperso por todo el disco.

4.8.10. Llamadas al sistema

La tabla 4.5 muestra las distintas llamadas al sistema relacionadas con el sistema de ficheros en LINUX.

Llamada	Descripción
creat	Crea un nuevo fichero.
open	Abre fichero.
read	Leer de un fichero.
write	Escribir en un fichero.
lseek	Cambiar la posición.
close	Cerrar un fichero.
mknod	Crea un fichero especial.
chdir	Cambio de directorio.
chroot	Cambiar de directorio raíz.
chown, fchown	Cambiar el dueño.
chmod, fchmod	Cambiar los permisos.
mount	Montar un sistema de ficheros.
umount	Desmontar un sistema de ficheros.
stat, fstat, lstat	Obtener información del nodo-i.
link	Crea un enlace duro.
unlink	Elimina un enlace.

Cuadro 4.5: Llamadas al sistema relacionadas con el sistema de ficheros

4.9. Resumen

Los sistemas operativos proporcionan una abstracción muy útil para tratar con la información que reside en los dispositivos de almacenamiento, los ficheros. Un fichero es una colección de datos con un nombre que suele residir en un dispositivo de almacenamiento secundario como un disco o una cinta. La parte del sistema operativo que los trata se denomina sistema de ficheros, y proporciona a los usuarios y a las aplicaciones servicios relacionados con el uso de éstos.

En los sistemas de ficheros podemos encontrar dos elementos fundamentales, los ficheros y los directorios. Estos últimos son también ficheros que contienen información sobre otros catalogados en ellos.

Se pueden distinguir dos aspectos fundamentales de los sistemas de ficheros, la forma en que éstos aparecen ante los usuarios o interfaz del sistema de ficheros, y los algoritmos y estructuras de datos que deben crearse para establecer la correspondencia entre el sistema de ficheros que ve el usuario y el real.

Para los usuarios los puntos de interés de un sistema de ficheros son la forma de nombrarlos, las operaciones que pueden hacerse sobre ellos, cómo se protegen, su estructura lógica y la forma de acceso. Con respecto a los directorios les interesa la información que almacenan, las operaciones que pueden realizarse y la forma en que pueden organizarse los ficheros que se catalogan en ellos.

Otros aspectos con los que los usuarios no tratan directamente, pero que son de gran importancia para el rendimiento del sistema son: el método de asignación de espacio a los ficheros, la forma de llevar el control del espacio libre y el método empleado para implementar los directorios.

El sistema de ficheros debe proporcionar mecanismos para asegurar la consistencia de los datos almacenados en él, en este capítulo se estudia el proporcionado por el sistema Linux.