CAPÍTULO 11

Gestión de E/S y planificación del disco

◆ ◆

Probablemente, la E/S es el aspecto más intrincado en el diseño de un sistema operativo. Dado que existe una gran variedad de dispositivos y de aplicaciones de los mismos, es difícil desarrollar una solución general uniforme.

Este capítulo comienza con un breve estudio de los dispositivos de E/S y la organización del sistema de E/S. Estos temas, que generalmente pertenecen al ámbito de la arquitectura de computadores, establecen una base para el estudio de la E/S desde el punto de vista del sistema operativo.

En la siguiente sección se estudian aspectos de diseño del sistema operativo, incluyendo los objetivos de diseño y la manera como se puede organizar el sistema de E/S. A continuación, se examinará el uso de buffers en las operaciones de E/S; uno de los servicios básicos de E/S proporcionados por el sistema operativo es un sistema de gestión de buffers, que mejora el rendimiento general.

Las siguientes secciones del capítulo estarán dedicadas a la E/S del disco magnético. En los sistemas contemporáneos, esta forma de E/S es la más importante, resultando fundamental para el rendimiento tal como lo percibe el usuario. Se comienza desarrollando un modelo del rendimiento de la E/S del disco y, a continuación, se estudian diversas técnicas que pueden utilizarse para mejorar el rendimiento.

Por último, un apéndice de este capítulo resume las características de los dispositivos de almacenamiento secundario, incluyendo el disco magnético y la memoria óptica.

◆ ◆

11.1. DISPOSITIVOS DE E/S

Como se mencionó en el Capítulo 1, los dispositivos externos dedicados a la E/S en un computador se pueden agrupar, a grandes rasgos, en tres categorías:

- **Legibles para el usuario.** Adecuados para la comunicación con el usuario del computador. Algunos ejemplos son las impresoras y terminales de visualización gráfica, que constan de pantalla, teclado y, posiblemente, otros dispositivos como un ratón.
- **Legibles para la máquina.** Adecuados para la comunicación con equipamiento electrónico. Algunos ejemplos son las unidades de discos y de cintas, los sensores, los controladores y los activadores.
- **Comunicación.** Adecuados para la comunicación con dispositivo remotos. Algunos ejemplos son los controladores de una línea digital y los módems.

Hay grandes diferencias entre las distintas categorías e incluso dentro de cada una. Entre las diferencias fundamentales se encuentran las siguientes:

- **Velocidad de transferencia de datos.** Puede haber diferencias de varios órdenes de magnitud entre las velocidades de transferencia de datos. La Figura 11.1 muestra algunos ejemplos.
- **Aplicación.** El uso al que está destinado un dispositivo tiene influencia en el software y en las políticas del sistema operativo y de las herramientas que le dan soporte. Por ejemplo, un disco utilizado para almacenar ficheros requiere el soporte de software de gestión de ficheros. Sin embargo, un disco utilizado como almacenamiento de respaldo para las páginas en un esquema de memoria virtual depende del uso del hardware y el software de memoria virtual. Además, estas dos distintas aplicaciones del disco tienen un impacto sobre los algoritmos de planificación de disco (que se estudiarán más adelante en este capítulo). Como un ejemplo adicional, considere un terminal que puede utilizar un usuario normal o un administrador del

sistema. Estos usos implican diferentes niveles de privilegios y posiblemente prioridades diferentes en el sistema operativo.

- **Complejidad de control.** Una impresora requiere una interfaz de control sencilla. Un disco es mucho más complejo. El efecto de estas diferencias en el sistema operativo se filtra hasta cierto punto por el módulo de E/S que controla el dispositivo, como se estudiará en la próxima sección.
- **Unidad de transferencia.** Los datos pueden transferirse como un flujo de bytes o caracteres (por ejemplo, la E/S de un terminal) o en bloques de mayor tamaño (por ejemplo, la E/S de un disco).
- **Representación de datos.** Los dispositivos utilizan diferentes esquemas de codificación de datos, incluyendo diferencias en el código del carácter y en las convenciones sobre la paridad.
- **Condiciones de error.** La naturaleza de los errores, el modo en que se notifican, sus consecuencias y el rango disponible de respuestas difieren considerablemente de un dispositivo a otro.

Esta diversidad hace difícil lograr un enfoque coherente y uniforme para la E/S, tanto desde el punto de vista del sistema operativo como de los procesos de usuario.

11.2. ORGANIZACIÓN DEL SISTEMA DE E/S

La Sección 1.7 presentó tres técnicas para llevar a cabo la E/S:

- **E/S programada.** El procesador envía un mandato de E/S, a petición de un proceso, a un módulo de E/S; a continuación, ese proceso realiza una espera activa hasta que se complete la operación antes de continuar.

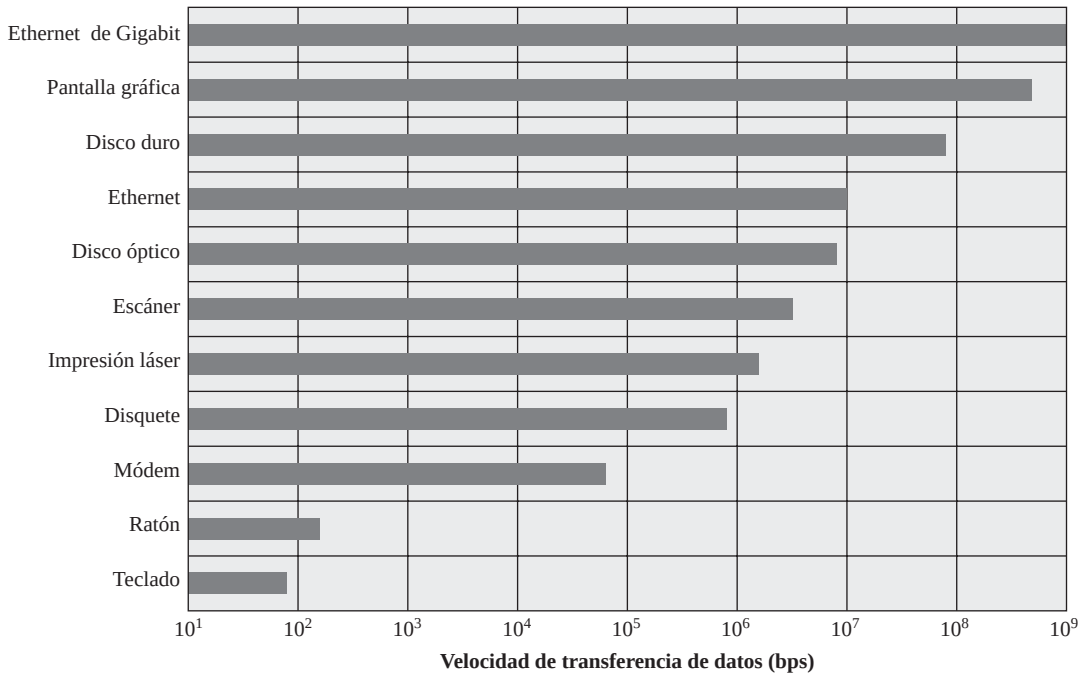


Figura 11.1. Velocidad típica de transferencia de datos de E/S.

- **E/S dirigida por interrupciones.** El procesador emite un mandato de E/S a petición de un proceso y continúa ejecutando las instrucciones siguientes, siendo interrumpido por el módulo de E/S cuando éste ha completado su trabajo. Las siguientes instrucciones pueden ser del mismo proceso, en el caso de que ese proceso no necesite esperar hasta que se complete la E/S. En caso contrario, se suspende el proceso en espera de la interrupción y se realiza otro trabajo.
- **Acceso directo de memoria (*Direct Memory Access*, DMA).** Un módulo de DMA controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador manda una petición de transferencia de un bloque de datos al módulo de DMA y resulta interrumpido sólo cuando se haya transferido el bloque completo.

La Tabla 11.1 indica la relación entre estas tres técnicas. En la mayoría de los computadores, el DMA es la forma de transferencia predominante, a la que el sistema operativo debe dar soporte.

Tabla 11.1. Técnicas de E/S.

	Sin interrupciones	**Con interrupciones**
Transferencia de E/S a memoria a través del procesador	E/S programada	E/S dirigida por interrupciones
Transferencia directa de E/S a memoria		Acceso directo a memoria (DMA)

LA EVOLUCIÓN DEL SISTEMA DE E/S

Según los computadores han ido evolucionando, ha habido una tendencia a una creciente complejidad y sofisticación de los componentes individuales, y esta tendencia en ninguna otra parte es tan evidente como en las funciones de E/S. Las etapas de esta evolución se pueden resumir de la siguiente manera:

1. El procesador controla directamente un dispositivo periférico. Esta situación se presenta en dispositivos simples controlados por un microprocesador.
2. Se añade un controlador o módulo de E/S. El procesador usa E/S programada sin interrupciones. Con este paso, el procesador se independiza de los detalles específicos de las interfaces de los dispositivos externos.
3. Se utiliza la misma configuración que en la etapa anterior, pero empleando interrupciones. El procesador no necesita gastar tiempo esperando a que se realice una operación de E/S, incrementando de esta manera la eficiencia.
4. Al módulo de E/S se le da control directo de la memoria mediante DMA. Con ello, puede mover un bloque de datos a la memoria sin involucrar el procesador, excepto al principio y al final de la transferencia.
5. Se mejora el módulo de E/S para convertirse en un procesador independiente, con un juego de instrucciones especializadas adaptadas a la E/S. La unidad central de procesamiento (CPU) hace que el procesador ejecute un programa de E/S residente en la memoria principal. El procesador de E/S lee y ejecuta estas instrucciones sin la intervención del procesador. Esto permite que el procesador especifique una secuencia de actividades de E/S, siendo interrumpido sólo cuando se termine la secuencia completa.

6. El módulo de E/S tiene su propia memoria local y es, de hecho, un computador por derecho propio. Con esta arquitectura, se pueden controlar un gran conjunto de dispositivos de E/S, con una intervención mínima por parte del procesador. Un uso común de esta arquitectura ha sido controlar la comunicación con terminales interactivos. El procesador de E/S se encarga de la mayoría de las tareas involucradas en el control de los terminales.

Según se avanza a lo largo de este proceso de evolución, cada vez hay una mayor parte de las tareas de E/S que se realizan sin la intervención del procesador. El procesador va quedando relevado gradualmente de estas tareas relacionadas con la E/S, mejorando el rendimiento. En las dos últimas etapas (5 y 6), se produce un cambio principal con la introducción del concepto de un módulo de E/S capaz de ejecutar un programa.

Una nota sobre la terminología: en todos los módulos descritos en las etapas desde la cuarta a la sexta, el uso del término *acceso directo a memoria* (*Direct Access Memory*, DMA) es apropiado, debido a que todos estos tipos de módulos implican el control directo de la memoria principal por parte del módulo de E/S. Asimismo, el módulo de E/S de la quinta etapa se denomina usualmente **canal de E/S**, y el de la sexta etapa **procesador de E/S**; sin embargo, cada término es, en ocasiones, aplicado a ambas situaciones. En la última parte de esta sección, se utilizará el término *canal de E/S* para referirse a ambos tipos de módulos de E/S.

ACCESO DIRECTO A MEMORIA

La Figura 1.12 indica, en términos generales, la lógica del DMA. La unidad de DMA es capaz de imitar al procesador, tomando el control del bus del sistema tal como lo hace un procesador. La unidad de DMA necesita hacerlo para transferir los datos desde y hacia la memoria usando el bus del sistema.

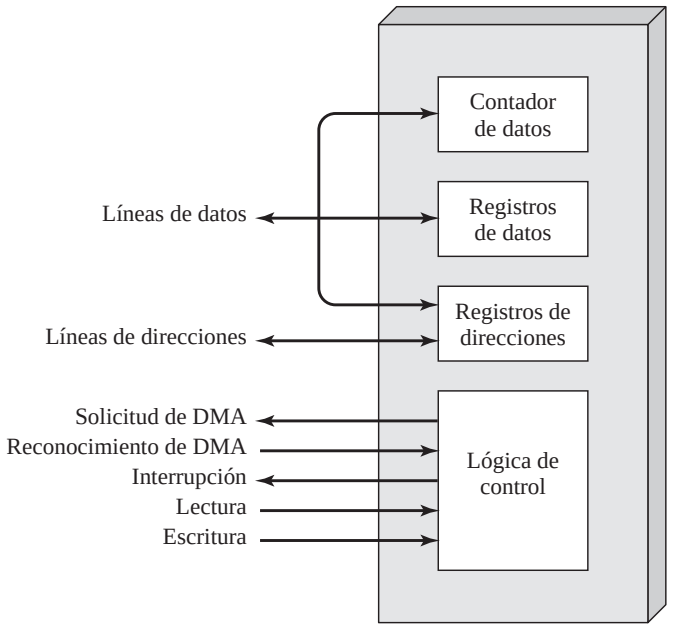


Figura 11.2. Típico diagrama de bloques del DMA.

La técnica del DMA funciona de la manera que se describe a continuación. Cuando el procesador quiere leer o escribir un bloque de datos, envía un mandato al módulo de DMA especificándole la siguiente información:

- Si se trata de una operación de lectura o de escritura, utilizando para ello la línea de control de lectura o escritura que existe entre el procesador y el módulo de DMA.
- La dirección del dispositivo de E/S involucrado, comunicándoselo mediante las líneas de datos.
- La dirección inicial de memoria que se pretende leer o escribir, comunicándoselo mediante las líneas de datos y almacenándose en el registro de dirección del módulo de DMA.
- El número de palabras que se van a leer o escribir, comunicándoselo de nuevo mediante las líneas de datos y almacenándose en el registro contador de datos.

A continuación, el computador continúa con otro trabajo. Ha delegado esta operación de E/S al módulo de DMA. El módulo de DMA transfiere el bloque completo de datos, palabra a palabra, directamente desde la memoria o hacia a ella, sin pasar por el procesador. Cuando se completa la transferencia, el módulo DMA envía una señal de interrupción al procesador. Por tanto, el procesador está involucrado sólo al principio y al final de la transferencia (Figura 1.19c).

El mecanismo de DMA se puede configurar de varias maneras. En la Figura 11.3 se muestran algunas alternativas. En el primer ejemplo, todos los módulos comparten el mismo bus de sistema. El módulo de DMA, actuando como un procesador subordinado, usa E/S programada para intercambiar datos entre la memoria y un módulo de E/S a través del módulo de DMA. Esta configuración, aunque pueda ser económica, es claramente ineficiente: al igual que ocurre con la E/S programada controlada por el procesador, cada transferencia de una palabra consume dos ciclos de bus (petición de transferencia seguida por la transferencia).

El número de ciclos de bus requeridos puede recortarse sustancialmente integrando el DMA y las funciones de E/S. Como indica la Figura 11.3b, esto significa que hay un camino entre el módulo de DMA y uno o más módulos de E/S que no incluye el bus del sistema. La lógica del DMA puede ser realmente parte de un módulo de E/S, o puede ser un módulo separado que controla uno o más módulos de E/S. Este concepto se puede llevar un paso más allá conectando los módulos de E/S al módulo DMA utilizando un bus de E/S (Figura 11.3c). Esto reduce a sólo uno el número de interfaces de E/S en el módulo de DMA y proporciona una configuración fácilmente expansible. En todos estos casos (Figuras 11.3 b y c), el módulo de DMA utiliza el bus del sistema, que comparte con el procesador y la memoria principal, sólo para intercambiar datos con la memoria y señales de control con el procesador. El intercambio de datos entre los módulos de DMA y de E/S tiene lugar fuera del bus del sistema.

11.3. ASPECTOS DE DISEÑO DEL SISTEMA OPERATIVO

OBJETIVOS DE DISEÑO

Hay dos objetivos de suma importancia en el diseño del sistema de E/S: eficiencia y generalidad. La **eficiencia** es importante debido a que las operaciones de E/S usualmente significan un cuello de botella en un computador. Examinando de nuevo la Figura 11.1, se observará que la mayoría de los dispositivos de E/S son extremadamente lentos comparados con la memoria principal y el procesador. Una manera de afrontar este problema es la multiprogramación, que, como ya se estudió previamente, permite que algunos procesos esperen por la finalización de operaciones de E/S mientras se

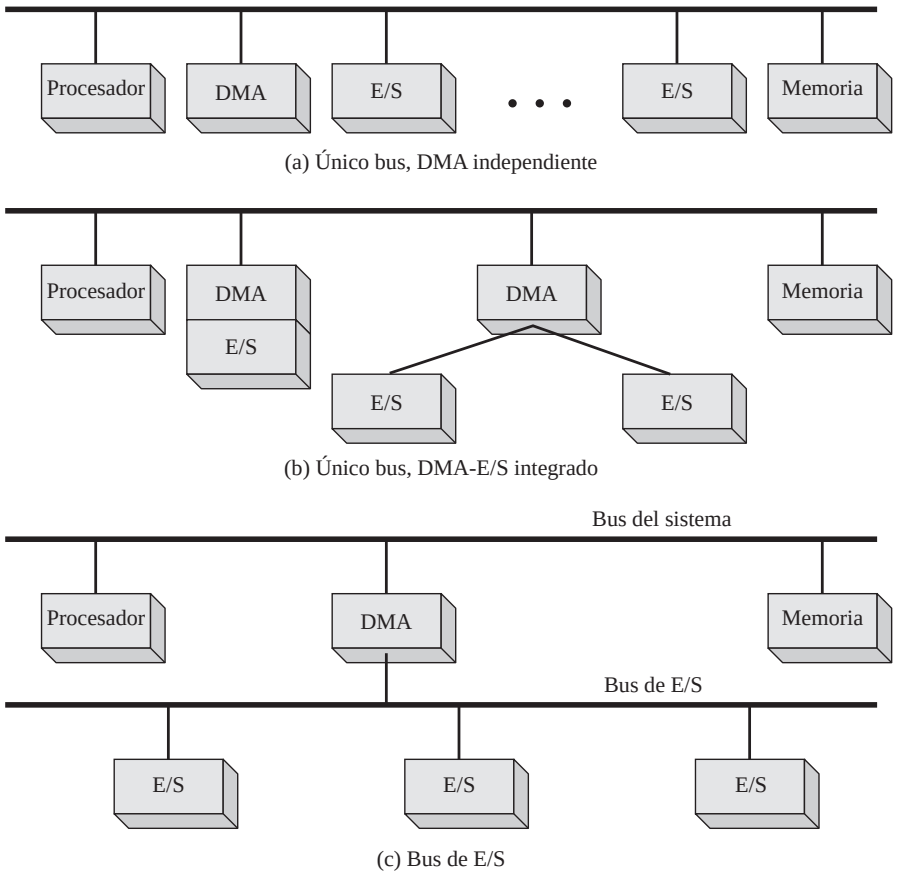


Figura 11.3. Configuraciones de DMA alternativas.

está ejecutando otro proceso. Sin embargo, a pesar del enorme tamaño de la memoria principal de las máquinas de hoy en día, se dará con cierta frecuencia la situación en la que la E/S no puede seguir el ritmo del procesador. Se puede utilizar el intercambio para poder tener más procesos listos de manera que se pueda mantener ocupado al procesador, pero esto en sí mismo es una operación de E/S. Por tanto, se ha dedicado un considerable esfuerzo en el diseño de la E/S para plantear esquemas que mejoren la eficiencia de E/S. El área que ha recibido la mayor atención, por su importancia, es la E/S de disco, por lo que gran parte de este capítulo estará dedicado al estudio de la eficiencia de la E/S de disco.

El otro objetivo principal es la **generalidad.** En aras de la simplicidad y la eliminación de errores, es deseable manejar todos los dispositivos de una manera uniforme. Esta afirmación se aplica tanto al modo en que los procesos ven los dispositivos de E/S como a la manera en que el sistema operativo gestiona los dispositivos y las operaciones de E/S. Debido a la diversidad de las características de los dispositivos, es difícil en la práctica alcanzar una total generalidad. Lo que hay que hacer es utilizar una estrategia modular jerárquica para diseñar las funciones de E/S. Esta estrategia esconde la mayoría de los detalles del dispositivo de E/S en las rutinas de nivel inferior de manera que los procesos de usuario y los niveles más altos del sistema operativo contemplan los dispositivos en términos de funciones generales, tales como lectura, escritura, abrir, cerrar, establecer un cerrojo y liberarlo. A continuación, se estudia esta estrategia.

ESTRUCTURA LÓGICA DEL SISTEMA DE E/S

En el Capítulo 2, en el estudio de la estructura del sistema, se enfatizó la naturaleza jerárquica de los sistemas operativos modernos. La filosofía jerárquica se basa en que las funciones del sistema operativo deberían estar separadas dependiendo de su complejidad, su escala de tiempo característica y su nivel de abstracción. Esta estrategia conduce a una organización del sistema operativo en una serie de niveles. Cada nivel realiza un subconjunto relacionado de las funciones requeridas del sistema operativo y se apoya en el nivel inferior subyacente para realizar funciones más básicas y ocultar los detalles de esas funciones, proporcionando servicios al siguiente nivel superior. Idealmente, los niveles se deberían definir de manera que los cambios en un nivel no requieran cambios en otros niveles. Por tanto, se ha descompuesto un problema en varios subproblemas más manejables.

En general, los niveles inferiores tratan con una escala de tiempo mucho más corta. Algunas partes del sistema operativo deben interaccionar directamente con el hardware del computador, donde los eventos pueden tener una escala de tiempo tan breve como unos pocos nanosegundos. En el otro extremo del espectro, se sitúan las partes del sistema operativo que se comunican con el usuario, que emite mandatos a un ritmo mucho más sosegado, quizás uno cada pocos segundos. El uso de un conjunto de niveles se ajusta perfectamente a este entorno.

Aplicando esta filosofía específicamente al sistema de E/S se llega al tipo de organización sugerida en la Figura 11.4 (compárese con la Tabla 2.4). Los detalles de organización dependerán del tipo de dispositivo y su aplicación. Las tres estructuras lógicas más importantes están representadas en la figura. Por supuesto, un sistema operativo puede que no se ajuste exactamente a estas estructuras. Sin embargo, los principios generales son válidos, por lo que el esquema de E/S de la mayoría de los sistemas operativos encaja aproximadamente en el descrito.

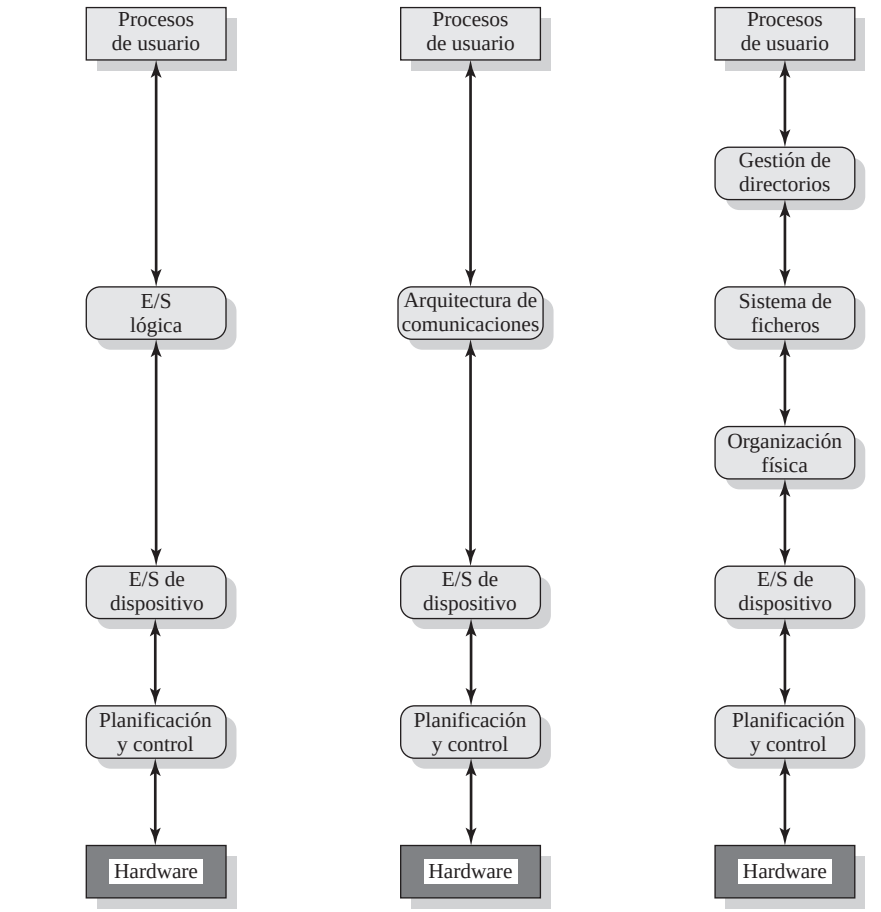
Considérese en primer lugar el caso más simple, el de un dispositivo periférico local que se comunica de una manera sencilla, tal como un flujo de bytes o registros (Figura 11.4a). Los niveles involucrados son los siguientes:

- **E/S lógica.** El módulo de E/S lógica trata a los dispositivos como un recurso lógico y no se ocupa de los detalles del control real del dispositivo. El módulo de E/S lógica se ocupa de la gestión de las tareas generales de E/S para los procesos de usuario, permitiéndolos tratar con el dispositivo en términos de un identificador de dispositivo y con mandatos sencillos como abrir, cerrar, leer y escribir.
- **E/S de dispositivo.** La operaciones requeridas y los datos (caracteres en los *buffers*, registros, etc.) se convierten en las secuencias apropiadas de instrucciones de E/S, mandatos del canal y órdenes del controlador. Se pueden utilizar técnicas de uso de *buffers* para mejorar la utilización.
- **Planificación y control.** La gestión real de la cola y la planificación de las operaciones de E/S se producen en este nivel, así como el control de las operaciones. Por tanto, en este nivel se manejan las interrupciones y se recoge el estado de la E/S y se informa del mismo. Este es el nivel de software que realmente interactúa con el módulo de E/S y, por tanto, con el hardware del dispositivo.

Para un dispositivo de comunicación, la estructura de E/S (Figura 11.4b) se parece mucho a la anteriormente descrita. La diferencia fundamental es que el módulo de E/S lógica se reemplaza por una arquitectura de comunicación, que puede a su vez consistir de varios niveles. Un ejemplo es TCP/IP, que se estudiará en el Capítulo 13.

La Figura 11.4c muestra una estructura representativa para la gestión de E/S en un dispositivo de almacenamiento secundario que proporciona soporte a un sistema de ficheros. Los tres niveles que no se han presentado previamente son:

- **Gestión de directorios.** En este nivel, los nombres simbólicos de los ficheros se convierten en identificadores que o bien hacen referencia directamente al fichero o indirectamente a través de un descriptor de fichero o una tabla de índices. Este nivel también se ocupa de las operaciones de usuario que afectan al directorio de ficheros, tales como añadir, borrar y reorganizar.
- **Sistema de ficheros.** Este nivel trata con la estructura lógica de los ficheros y con las operaciones que pueden especificar los usuarios, como abrir, cerrar, leer y escribir. Los derechos de acceso se gestionan también en este nivel.
- **Organización física.** De la misma manera que las direcciones de memoria virtual deben convertirse en direcciones físicas de memoria principal teniendo en cuenta la estructura de segmentación y paginación, las referencias lógicas a ficheros y registros se deben convertir en direcciones físicas del almacenamiento secundario, teniendo en cuenta la estructura de pistas



(a) Dispositivo periférico local (b) Puerto de comunicación (c) Sistema de fichero

Figura 11.4. Un modelo de organización de E/S.

y sectores del dispositivo de almacenamiento secundario. La asignación de espacio del almacenamiento secundario y de *buffers* del almacenamiento principal se tratan también en este nivel.

Dada la importancia del sistema de ficheros, se dedicará un cierto tiempo en este capítulo y en el próximo a estudiar sus diversos componentes. El estudio de este capítulo se centra en los tres niveles inferiores, mientras que en el Capítulo 12 se presentarán los dos niveles superiores.

11.4. UTILIZACIÓN DE BUFFERS DE E/S

Supóngase que un proceso de usuario desea leer bloques de datos de una cinta de uno en uno, teniendo cada bloque una longitud de 512 bytes. Los datos tienen que leerse en una zona de datos del espacio de direcciones del proceso de usuario que corresponde con las direcciones virtuales desde la 1000 hasta 1511. La manera más sencilla sería enviar un mandato de E/S (algo similar a Leer_Bloque[1000, cinta]) a la unidad de cinta y, a continuación, esperar hasta que los datos estén disponibles. La espera puede ser activa (comprobando continuamente el estado del dispositivo) o, de manera más práctica, suspendiendo el proceso hasta que se produzca una interrupción.

Esta estrategia conlleva dos problemas. Primero, el programa se queda esperando a que se complete la relativamente lenta operación de E/S. El segundo problema es que esta estrategia de E/S interfiere con las decisiones de intercambio del sistema operativo. Las posiciones virtuales desde la 1000 a la 1511 deben permanecer en la memoria principal durante el curso de la transferencia del bloque. En caso contrario, se pueden perder algunos datos. Si se utiliza paginación, al menos la página que contiene las posiciones de memoria requeridas debe fijarse como residente en la memoria principal. Por tanto, aunque algunas páginas del proceso pueden expulsarse al disco, es imposible expulsar completamente al proceso, incluso aunque lo precise el sistema operativo. Nótese también que hay riesgo de que se produzca el interbloqueo de un único proceso. Si un proceso emite un mandato de E/S, se suspende esperando el resultado, y, a continuación, se expulsa antes del inicio de la operación, el proceso se bloquea esperando el evento de E/S y la operación de E/S se bloquea esperando a que el proceso se traiga de nuevo a memoria. Para evitar este interbloqueo, la memoria de usuario involucrada en la operación de E/S debe quedarse residente en la memoria principal inmediatamente antes de que se emita la petición de E/S, incluso aunque la operación de E/S se encole y no pueda ejecutarse durante algún tiempo.

Las mismas consideraciones se aplican a una operación de salida. Si se está transfiriendo un bloque desde un área del proceso de usuario directamente a un módulo de E/S, el proceso se bloquea durante la transferencia y el proceso no puede expulsarse.

Para evitar estas sobrecargas e ineficiencias, es a veces conveniente realizar las transferencias de entrada antes de que se hagan las peticiones correspondientes y llevar a cabo las transferencias de salida un cierto tiempo después de que se haya hecho la petición. Esta técnica se conoce como E/S con *buffers*. En esta sección, se analizarán algunos esquemas de gestión de *buffers* que proporcionan los sistemas operativos para mejorar el rendimiento del sistema.

A la hora de analizar las distintas estrategias de gestión de *buffers*, es a veces importante hacer una distinción entre dos tipos de dispositivos de E/S: orientados a bloques y orientados a flujo de caracteres. Un dispositivo **orientado a bloques** almacena información en bloques que son usualmente de tamaño fijo realizándose las transferencias de bloque en bloque. Generalmente, es posible hacer referencia a los datos mediante su número de bloque. Los discos y las cintas son ejemplos de dispositivos orientados a bloques. Un dispositivo **orientado a flujo de caracteres** transfiere los datos, tanto de entrada como de salida, como un flujo de bytes, sin estructura de bloques. Los terminales, las impresoras, los puertos de comunicación, el ratón y otros dispositivos apuntadores, y

la mayoría de los dispositivos que no son de almacenamiento secundario están orientados a flujos de caracteres.

BUFFER ÚNICO

El tipo más sencillo de esquema que puede proporcionar el sistema operativo es el *buffer* único (Figura 11.5b). Cuando un proceso de usuario emite una petición de E/S, el sistema operativo asigna un *buffer* para la operación en la parte de sistema de la memoria principal.

Para dispositivos orientados a bloques, el esquema con un *buffer* único se puede describir de la siguiente manera: las transferencias de entrada usan el *buffer* del sistema. Cuando se completa la transferencia, el proceso mueve el bloque al espacio de usuario e inmediatamente pide otro bloque. A esto se le denomina lectura adelantada, o entrada anticipada; esta operación se realiza con la esperanza de que se acabará necesitando ese bloque. Para muchos tipos de cómputos, se trata de una suposición razonable durante la mayor parte del tiempo porque normalmente se accede a los datos de forma secuencial. Sólo al final de una secuencia de procesamiento se leerá innecesariamente un bloque.

Esta estrategia proporcionará generalmente una cierta mejora en el rendimiento comparada con la alternativa de no usar *buffers* en el sistema. El proceso de usuario puede estar procesando un bloque de datos mientras se está leyendo el bloque siguiente. El sistema operativo es capaz de expulsar

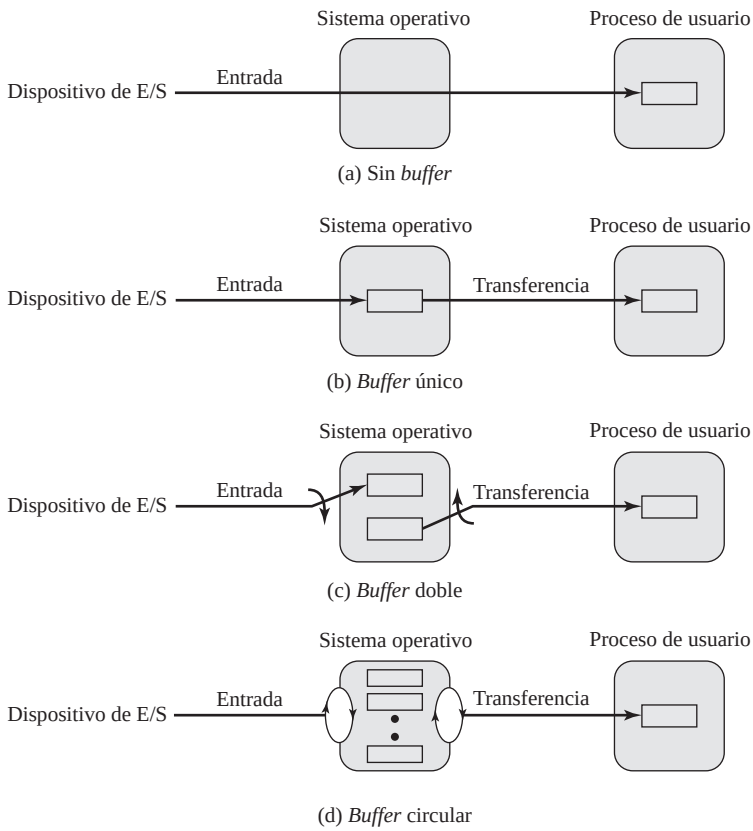


Figura 11.5. Esquemas de uso de *buffers* de E/S (entrada/salida).

al proceso puesto que se está llevando a cabo la operación de entrada en la memoria del sistema en vez de en la memoria del proceso de usuario. Esta técnica, sin embargo, complica la lógica en el sistema operativo. El sistema operativo debe hacer un seguimiento de la asignación de *buffers* del sistema a los procesos de usuario. También afecta a la lógica del intercambio: si la operación de E/S afecta al mismo disco que se está usando para el intercambio, prácticamente no tiene sentido encolar las escrituras en disco requeridas para expulsar el proceso sobre ese mismo dispositivo. Este intento de expulsar el proceso y liberar memoria principal no comenzará en sí mismo hasta que termine la operación de E/S previa, en cuyo momento la expulsión del proceso al disco puede haber dejado de ser apropiada.

Se pueden aplicar consideraciones similares a la salida orientada a bloques. Cuando se transfieren los datos a un dispositivo, en primer lugar, se copian desde el espacio de usuario hasta el *buffer* del sistema, desde donde se escribirán definitivamente. El proceso solicitante puede en este momento continuar o ser expulsado cuando sea necesario.

[KNUT97] sugiere una comparación de rendimiento, un tanto tosca pero bastante informativa, entre usar un *buffer* único y no usar ninguno. Supóngase que T es el tiempo requerido para leer un bloque y que P es el tiempo de proceso que transcurre entre sucesivas peticiones de lectura. Sin *buffer*, el tiempo de ejecución por bloque es básicamente $T + P$. Con un *buffer* único, el tiempo es el máximo de $[P, T] + M$, donde M es el tiempo requerido para mover los datos desde el *buffer* del sistema a la memoria de usuario. En la mayoría de los casos, el tiempo de ejecución por bloque es sustancialmente menor usando un *buffer* único que no usando ninguno.

En el caso de la E/S orientada a flujo de caracteres, el esquema de *buffer* único se puede utilizar en un modo de operación línea a línea o en uno byte a byte. El modo de operación línea a línea es apropiado para los terminales en modo texto, en los que el texto se va desplazando verticalmente (a veces denominados terminales no inteligentes). Con este tipo de terminal, la entrada del usuario en el terminal es línea a línea, usando un retorno de carro para indicar el final de una línea, y, de manera similar, la salida es también línea a línea. Una impresora de líneas es otro ejemplo de este tipo de dispositivos. La operación byte a byte se utiliza en los terminales en modo formulario, donde cada tecla pulsada es significativa, y para muchos otros periféricos, como los sensores y los controladores.

En el caso de E/S línea a línea, se puede utilizar el *buffer* para almacenar una única línea. El proceso de usuario se suspende durante la entrada de datos, esperando la llegada de una línea completa. Con respecto a la salida, el proceso de usuario puede copiar una línea de salida en el *buffer* y continuar el procesamiento. No se necesita suspender el proceso a menos que tenga una segunda línea de salida que enviar al *buffer* antes de que se vacíe el mismo al terminar la primera operación de salida. En el caso de E/S byte a byte, la interacción entre el sistema operativo y el proceso de usuario sigue el modelo productor-consumidor estudiado en el Capítulo 5.

BUFFER DOBLE

Se puede hacer una mejora sobre la técnica del *buffer* único asignando a la operación dos *buffers* del sistema (Figura 11.5c). Con este nuevo esquema, un proceso transfiere datos a (desde) un *buffer* mientras el sistema operativo vacía (o llena) el otro. Esta técnica se conoce como ***buffer* doble** o **intercambio de *buffers***.

En cuanto a la transferencia orientada a bloques, se puede estimar aproximadamente el tiempo de ejecución como el máximo de $[P, T]$. Por tanto, es posible mantener el dispositivo orientado a bloques trabajando a toda velocidad si $P \leq T$. Por otro lado, si $P > T$, el *buffer* doble asegura que el proceso no tendrá que esperar la finalización de la E/S. En cualquier caso, se alcanza una mejora con respecto al uso de un *buffer* único. De nuevo, esta mejora conlleva un incremento de la complejidad.

Con respecto a la entrada orientada a flujo de caracteres, se plantean de nuevo los dos modos de operación alternativos. En el caso de la E/S línea a línea, el proceso de usuario no necesita suspenderse debido a una operación de entrada o salida, a menos que el proceso vaya por delante del *buffer* doble. En el caso de operación byte a byte, el *buffer* doble no ofrece ninguna ventaja adicional sobre un *buffer* único de doble longitud. En ambos casos, se utiliza el modelo productor-consumidor.

BUFFER CIRCULAR

Un esquema de *buffer* doble debería suavizar el flujo de datos entre un dispositivo de E/S y un proceso. Si el interés está centrado en el rendimiento de un determinado proceso, se desearía que la operación de E/S fuera capaz de mantener el ritmo del proceso. El *buffer* doble puede ser inadecuado si el proceso realiza ráfagas rápidas de E/S. En este caso, el problema puede aliviarse frecuentemente utilizando más de dos *buffers*.

Cuando se utilizan más de dos *buffers*, al conjunto de *buffers* se le denomina *buffer* circular (Figura 11.5d), siendo cada *buffer* individual una unidad del *buffer* circular. Se trata simplemente de un modelo productor-consumidor con un *buffer* acotado, como se estudió en el Capítulo 5.

LA UTILIDAD DEL USO DE *BUFFERS*

El uso de *buffers* es una técnica que amortigua los picos en la demanda de E/S. Sin embargo, por muchos *buffers* que se utilicen, estos no permitirán a un dispositivo de E/S mantener el ritmo de un proceso indefinidamente cuando la demanda media del proceso sea mayor que la que puede servir el dispositivo de E/S. Incluso con múltiples *buffers*, todos los *buffers* acabarán llenándose y el proceso tendrá que esperar después de procesar cada fragmento de datos. Sin embargo, en un entorno de multiprogramación, donde hay diversas actividades de E/S y distintos procesos que hay que atender, el uso de *buffers* es una técnica que puede incrementar la eficiencia del sistema operativo y el rendimiento de los procesos individuales.

11.5. PLANIFICACIÓN DEL DISCO

Durante los últimos cuarenta años, el incremento de la velocidad de los procesadores y de la memoria principal ha sobrepasado con creces el de la velocidad de acceso al disco, incrementándose en aproximadamente dos órdenes de magnitud mientras que la velocidad del disco lo ha hecho en un orden de magnitud. El resultado es que los discos son actualmente al menos cuatro órdenes de magnitud más lentos que la memoria principal. Además, se espera que esta diferencia aumente en el futuro inmediato. Por tanto, el rendimiento del subsistema de almacenamiento de disco es de vital importancia, por lo que se han realizado muchas investigaciones en esquemas para mejorar ese rendimiento. En esta sección, se resaltan algunos de los aspectos fundamentales y se revisan las técnicas más importantes. Dado que el rendimiento del sistema de discos está estrechamente asociado a aspectos de diseño del sistema de ficheros, este estudio continuará en el Capítulo 12.

PARÁMETROS DE RENDIMIENTO DEL DISCO

Los detalles reales de la operación de E/S del disco dependen del computador, del sistema operativo, y de la naturaleza del hardware del canal de E/S y del controlador del disco. En la Figura 11.6 se muestra un diagrama general de tiempos de una transferencia de E/S de disco.

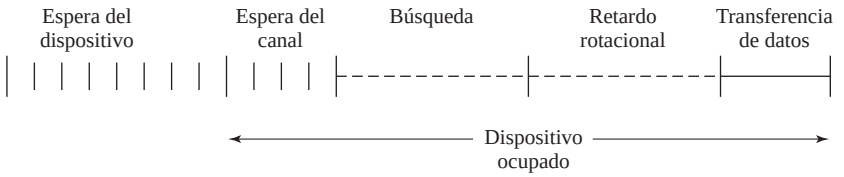


Figura 11.6. Diagrama de tiempos de una transferencia de E/S de disco.

Cuando está en funcionamiento la unidad de disco, el disco rota a una velocidad constante. Para leer o escribir, la cabeza se debe posicionar en la pista deseada y en el principio del sector requerido de dicha pista[1]. La selección de pista implica el movimiento de la cabeza en un sistema de cabeza móvil o la selección de manera electrónica de una cabeza en un sistema de cabeza fija. En un sistema de cabeza móvil, el tiempo que se tarda en situar la cabeza en la pista se denomina **tiempo de búsqueda**. En cualquier caso, una vez que se selecciona la pista, el controlador del disco espera hasta que el sector apropiado rote debajo de la cabeza. El tiempo que tarda en llegar el comienzo del sector hasta debajo de la cabeza se conoce como **retardo rotacional**, o latencia rotacional. La suma del tiempo de búsqueda, en el caso de que lo haya, y el retardo rotacional dan lugar al **tiempo de acceso**, que es el tiempo que se tarda en llegar a estar en posición para realizar la lectura o escritura. Una vez que la cabeza está en posición, la operación de lectura o escritura se realiza cuando el sector se mueve debajo de la cabeza; ésta es la parte de la transferencia de datos de la operación; el tiempo requerido para la transferencia es el **tiempo de transferencia**.

Además del tiempo de acceso y de transferencia, hay varios retardos por esperas en colas normalmente asociadas con una operación de E/S del disco. Cuando un proceso emite una petición de E/S, debe esperar, en primer lugar, en una cola del dispositivo hasta que esté disponible. En ese momento, se asigna el dispositivo al proceso. Si el dispositivo comparte un único canal de E/S o un conjunto de canales de E/S con otras unidades de disco, puede haber una espera adicional hasta que el canal esté disponible. En ese punto, se realiza la búsqueda para comenzar el acceso al disco.

En algunos sistemas de tipo *mainframe*, se utiliza una técnica conocida como detección de posición rotacional (*Rotational Position Sensing*, RPS). Este mecanismo funciona de la forma siguiente: cuando se ha generado el mandato de búsqueda, se libera el canal para manejar otras operaciones de E/S. Cuando se completa la búsqueda, el dispositivo determina cuándo los datos rotan debajo de la cabeza. Cuando ese sector se aproxima a la cabeza, el dispositivo intenta restablecer el camino de comunicación de vuelta a la máquina. Si la unidad de control o el canal están ocupados con otra operación de E/S, el intento de volver a conectarse falla y el dispositivo debe rotar una revolución completa antes de que pueda intentar volverse a conectar, a lo que se denomina un fallo de RPS. Esta situación representa un elemento de retardo adicional que debe añadirse a la línea de tiempo de la Figura 11.6.

Tiempo de búsqueda

El tiempo de búsqueda es el tiempo requerido para mover el brazo del disco a la pista requerida. El tiempo de búsqueda consta de dos componentes fundamentales: el tiempo de arranque inicial y el tiempo que se tarda en atravesar las pistas que tienen que cruzarse una vez que el brazo de acceso

[1] *Véase* el Apéndice 11A para un estudio de la organización y el formato del disco.

empieza a moverse. Por desgracia, el tiempo para atravesar las pistas no es una función lineal del número de pistas sino que incluye un tiempo de establecimiento (el tiempo que transcurre desde que se posiciona la cabeza sobre la pista prevista hasta que se confirma su identificación).

Se han producido muchas mejoras gracias al desarrollo de componentes de disco más pequeños y ligeros. Hace algunos años, un disco normal tenía un diámetro de 14 pulgadas (36 cm.), mientras que el tamaño más común actualmente es de 3,5 pulgadas (8,9 cm.), reduciendo la distancia que tiene que moverse el brazo. Un tiempo normal de búsqueda medio en los discos duros actuales está por debajo de los 10 ms.

Retardo rotacional

Los discos, excepto los flexibles, rotan a velocidades que van desde las 3600 rpm (para dispositivos portátiles como cámaras digitales) hasta 15.000 rpm, en el momento en el que se escribió este libro; a esta última velocidad se produce una revolución cada 4 ms. Por tanto, en promedio, el retardo rotacional será de 2 ms. Los discos flexibles normalmente giran a una velocidad entre 300 y 600 rpm. Por tanto, el retardo medio estará entre 100 y 50 ms.

Tiempo de transferencia

El tiempo de transferencia de un disco depende de la velocidad de rotación de acuerdo con la siguiente expresión:

$$T = \frac{b}{rN}$$

donde

T = tiempo de transferencia

b = número de bytes que se van a transferir

N = número de bytes en una pista

r = velocidad de rotación, en revoluciones por segundo

Por tanto, el tiempo de acceso total medio se puede expresar de la siguiente manera:

$$T_a = T_b + \frac{1}{2r} + \frac{b}{rN}$$

donde T_b es el tiempo de búsqueda medio.

Una comparación de tiempos

Con los parámetros previamente definidos, se examinarán dos operaciones de E/S diferentes para mostrar el peligro de confiar en los valores medios. Considere un disco con un tiempo de búsqueda medio de 4ms. según su fabricante, una velocidad de rotación de 7.500 rpm y sectores de 512 bytes con 500 sectores por pista. Supóngase que se quiere leer un fichero que consta de 2.500 sectores, lo que significa un total de 1,28 Mbytes. Se pretende estimar el tiempo total de transferencia.

En primer lugar, supóngase que el fichero se almacena de la forma más compacta posible en el disco. Es decir, el fichero ocupa todos los sectores de 5 pistas adyacentes (5 pistas × 500 sectores/pista = 2.500 sectores). Esto se conoce como *organización secuencial*. El tiempo para leer la primera pista es el siguiente:

Búsqueda media	4	ms
Retardo rotacional	4	ms
Lectura de 500 sectores	8	ms
	16	ms

Supóngase que las restantes pistas se pueden leer ahora sin básicamente tiempo de búsqueda. Es decir, la operación de E/S puede seguir el flujo del disco. Entonces, como mucho, se necesita tener en cuenta el retardo rotacional para cada pista sucesiva. De este modo, se lee cada pista sucesiva en 4 + 8 = 12 ms. Para leer el fichero entero sería necesario:

$$\text{Tiempo total} = 16 + 4 \times 12 = 64 \text{ ms} = 0{,}064 \text{ segundos}$$

A continuación, se calculará el tiempo requerido para leer los mismos datos utilizando un acceso aleatorio en lugar de un acceso secuencial; es decir, los accesos a los sectores se distribuyen de manera aleatoria en el disco. Por cada sector, se tiene:

Búsqueda media	4	ms
Retardo rotacional	4	ms
Lectura de 1 sector	0,016	ms
	8,016	ms

$$\text{Tiempo total} = 2500 \times 8{,}016 = 20.040 \text{ ms} = 20{,}04 \text{ segundos}$$

Es evidente que el orden en el que se leen los sectores del disco tiene un efecto tremendo en el rendimiento de la E/S. En el caso del acceso a un fichero en el que se leen o se escriben múltiples sectores, se tiene algún control sobre el modo en el que se distribuyen los sectores de datos en el disco, como se estudiará en el próximo capítulo. Sin embargo, incluso en el caso del acceso a un fichero, en un entorno de multiprogramación, habrá diversas peticiones de E/S compitiendo por el mismo disco. Por tanto, es importante examinar de qué maneras se puede mejorar el rendimiento de la E/S del disco con respecto al logrado con un acceso al disco puramente aleatorio.

POLÍTICAS DE PLANIFICACIÓN DEL DISCO

En el ejemplo que se acaba de describir, la diferencia en el rendimiento se debe al tiempo de búsqueda. Si las peticiones de acceso a los sectores involucran una selección aleatoria de pistas, el rendimiento del sistema de E/S del disco será el peor posible. Para mejorar el asunto, se necesita reducir el tiempo medio gastado en las búsquedas.

Considere la situación normal en un entorno de multiprogramación, en el que el sistema operativo mantiene una cola de peticiones por cada dispositivo de E/S. Así, en el caso de un único disco, habrá varias peticiones de E/S (lecturas y escrituras) de diversos procesos en la cola. Si se seleccionan elementos de la cola en un orden aleatorio, se puede predecir que las pistas se van a visitar aleatoriamente, proporcionando un rendimiento deficiente. Esta **planificación aleatoria** es útil como un punto de referencia para evaluar otras técnicas.

La Figura 11.7 compara el rendimiento de varios algoritmos de planificación usando un ejemplo de una secuencia de peticiones de E/S. El eje vertical corresponde con las pistas en el disco. El eje horizontal corresponde con el tiempo o, de manera equivalente, el número de pistas atravesadas. En esta figura, se asume que la cabeza del disco está situada inicialmente en la pista 100. En este ejemplo, se supone un disco con 200 pistas y que la cola de peticiones del disco incluye peticiones aleatorias. Las pistas solicitadas, en el orden recibido por el planificador del disco, son 55, 58, 39, 18, 90, 160, 150, 38 y 184. La Tabla 11.2a muestra los resultados en forma de tabla.

Tabla 11.2. Comparación de algoritmos de planificación de disco.

(a) FIFO (comenzando en la pista 100)		(b) SSTF (comenzando en la pista 100)		(c) SCAN (comenzando en la pista 100, en la dirección de números de pista crecientes)		(d) C-SCAN (comenzando en la pista 100, en la dirección de números de pista crecientes)	
Próxima pista accedida	**Número de pistas atravesadas**	**Próxima pista accedida**	**Número de pistas atravesadas**	**Próxima pista accedida**	**Número de pistas atravesadas**	**Próxima pista accedida**	**Número de pistas atravesadas**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
longitud media de búsqueda	55,3	**longitud media de búsqueda**	27,5	**longitud media de búsqueda**	27,8	**longitud media de búsqueda**	35,8

Primero en entrar, primero en salir

La forma más sencilla de planificación corresponde con el algoritmo del primero en entrar, primero en salir (*First In-First Out*, FIFO), que procesa los elementos de la cola en orden secuencial. Esta estrategia tiene la ventaja de ser equitativa, porque toda petición se acaba sirviendo y, además, las peticiones se sirven en el orden recibido. La Figura 11.7a muestra el movimiento del brazo del disco con el algoritmo FIFO.

Con esta planificación, si sólo hay unos pocos procesos que requieren acceso al disco y una gran parte de las peticiones corresponden con sectores agrupados de ficheros, se puede prever un buen rendimiento. Sin embargo, esta técnica tendrá con frecuencia resultados similares a la planificación aleatoria en cuanto al rendimiento, en el caso de que haya muchos procesos compitiendo por el disco. Por tanto, puede ser beneficioso considerar una política de planificación más sofisticada. En la Tabla 11.3 se enumeran varias políticas de planificación, que se analizarán a continuación.

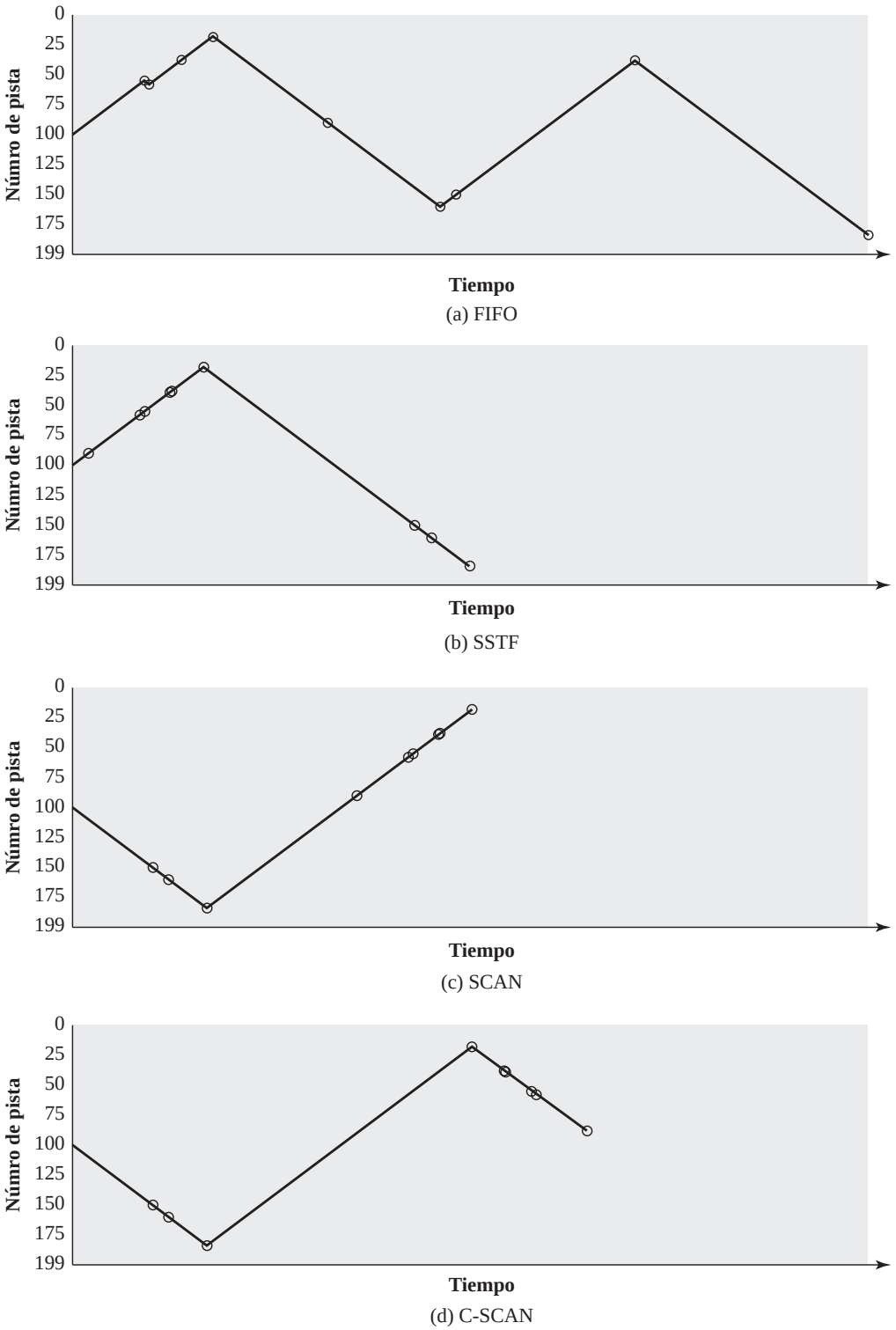


Figura 11.7. Comparación de algoritmos de planificación del disco (*véase* la Tabla 11.3).

Tabla 11.3. Algoritmos de planificación del disco.

Nombre	Descripción	Comentarios
	Selección según el solicitante	
PA	Planificación aleatoria	Para análisis y simulación
FIFO	Primero en entrar, primero en salir	El más equitativo de todos
PRI	Prioridad por proceso	Control externo a la gestión de la cola de disco
LIFO	Último en entrar, primero en salir	Maximiza proximidad y uso de recursos
	Selección según el elemento solicitado	
SSTF	Primero el de tiempo de servicio más corto	Buena utilización; colas pequeñas
SCAN	Recorrido del disco bidireccional	Mejor distribución del servicio
C-SCAN	Recorrido unidireccional con retorno rápido	Menor variabilidad de servicio
SCAN-de-N-pasos	SCAN de *N* registros cada vez	Garantía de servicio
FSCAN	SCAN-de-*N*-pasos con *N* = tamaño de la cola al principio del ciclo de SCAN	Sensible a la carga

Prioridad

En un sistema basado en la prioridad (PRI), la planificación está fuera del control del software de gestión de disco. Esta estrategia no está diseñada para optimizar la utilización del disco sino para satisfacer otros objetivos dentro del sistema operativo. Con frecuencia se les da mayor prioridad a los trabajos por lotes de corta duración y a los trabajos interactivos que a los trabajos más largos que requieren un procesamiento más prolongado. Esto permite que muchos trabajos cortos salgan del sistema rápidamente, pudiendo proporcionar un tiempo de respuesta interactiva adecuado. Sin embargo, los trabajos más largos pueden tener que esperar un tiempo excesivamente largo. Además, una política de este tipo podría llevar a que los usuarios tomaran medidas en su contra, dividiendo sus trabajos en partes más pequeñas para engañar al sistema. Este tipo de política tiende a ser inadecuada para los sistemas de base de datos.

Último en entrar, primero en salir

Sorprendentemente, la política de seleccionar siempre la petición más reciente tiene cierto mérito. En sistemas de procesamiento de transacciones, asignar el dispositivo al usuario más reciente debería producir poco o ningún movimiento del brazo durante un desplazamiento a lo largo de un fichero secuencial. Aprovechar la proximidad mejora el rendimiento y reduce la longitud de la cola. Mientras un trabajo utilice activamente el sistema de ficheros, se procesará tan rápido como sea posible. Sin embargo, si el disco se mantiene ocupado debido a una gran carga de trabajo, hay una evidente posibilidad de inanición. Una vez que un proceso ha insertado una petición de E/S en la cola y pierde su posición en la parte delantera de la misma, el proceso no podrá recuperar dicha posición hasta que se terminen las peticiones que hay delante en la cola.

Las planificaciones FIFO, por prioridad y LIFO (último en entrar, primero en salir) están basadas solamente en atributos de la cola o del demandante. Si el planificador conoce la posición de pista actual, se puede emplear una planificación basada en el elemento solicitado. A continuación se examinan estas políticas.

Primero el de tiempo de servicio más corto

La política SSTF (*Shortest Service Time First*, primero el de tiempo de servicio más corto) consiste en seleccionar la petición de E/S del disco que requiera un menor movimiento del brazo desde su posición actual. De ese modo, siempre se realiza una selección de manera que se produzca un tiempo de búsqueda mínimo. Evidentemente, seleccionar siempre el tiempo de búsqueda mínimo no garantiza que sea mínimo el tiempo de búsqueda medio correspondiente a varios movimientos del brazo. Sin embargo, este esquema debería proporcionar un rendimiento mejor que el algoritmo FIFO. Dado que el brazo puede moverse en dos direcciones, se puede utilizar un algoritmo aleatorio para resolver los casos de empate debido a la existencia de distancias iguales.

La Figura 11.7b y la Tabla 11.2b muestran el rendimiento de SSFT en el mismo ejemplo que se usó con la planificación FIFO.

Scan

Con la excepción de la planificación FIFO, todas las políticas descritas hasta ahora pueden dejar alguna petición sin servir hasta que se vacíe completamente la cola de peticiones. Es decir, podrían llegar continuamente nuevas peticiones que se seleccionarían antes de una petición existente. Una alternativa sencilla que impide este tipo de inanición es el algoritmo SCAN, también conocido como el algoritmo del ascensor porque funciona de manera muy similar a como lo hace un ascensor.

Con el algoritmo SCAN, el brazo sólo debe moverse en una dirección, satisfaciendo todas las peticiones pendientes que encuentre en su camino, hasta que alcanza la última pista en esa dirección o hasta que no haya más peticiones en esa dirección. A esta última mejora se le denomina política LOOK. En ese momento, la dirección de servicio se invierte y la búsqueda continúa en la dirección opuesta, sirviendo de nuevo todas las peticiones en orden.

La Figura 11.7c y la Tabla 11.2c muestran la política SCAN. Como se puede observar, la política SCAN se comporta prácticamente igual que la política SSTF. De hecho, si se hubiera supuesto que el brazo se estaba moviendo en la dirección de las pistas con números más bajos al comienzo del ejemplo, el patrón de planificación habría sido el mismo para SSTF y SCAN. Sin embargo, se trata de un ejemplo estático en el que no se añade ningún elemento nuevo a la cola. En cualquier caso, incluso si la cola está cambiando dinámicamente, SCAN será similar a SSTF a menos que el patrón de peticiones sea inusual.

Nótese que la política SCAN no favorece en su tratamiento al área del disco que se ha atravesado más recientemente. Por tanto, no aprovecha la proximidad tan bien como lo hacen SSTF o incluso LIFO.

No es difícil observar que la política SCAN favorece a los trabajos cuyas peticiones corresponden con las pistas más cercanas tanto a las pistas interiores como a las exteriores, así como a los trabajos que han llegado más recientemente. El primer problema se puede evitar mediante la política C-SCAN, mientras que el segundo problema lo soluciona la política SCAN-de-*N*-pasos.

C-Scan

La política C-SCAN (SCAN circular) restringe la búsqueda a una sola dirección. Por tanto, después de visitar la última pista en una dirección, el brazo vuelve al extremo opuesto del disco y la búsqueda comienza de nuevo. Esto reduce el retardo máximo que pueden experimentar las nuevas peticiones. Con SCAN, si el tiempo previsto para una búsqueda de la pista más interna a la más externa es t, el intervalo de servicio previsto para los sectores en la periferia es $2t$. Con C-SCAN, el intervalo es del orden de $t + s_{máx}$, siendo $s_{máx}$ el tiempo de búsqueda máximo.

La Figura 11.7d y la Tabla 11.2d muestran el comportamiento de C-SCAN.

SCAN-de-*N*-pasos y FSCAN

Con SSFT, SCAN y C-SCAN, es posible que el brazo no pueda moverse durante un periodo de tiempo considerable. Por ejemplo, si uno o más procesos tienen elevadas tasas de acceso a una determinada pista, pueden monopolizar el dispositivo entero debido a sucesivas peticiones a esa pista. Los discos de alta densidad con múltiples superficies se van a ver probablemente más afectados por esta característica que los discos de densidad más baja y/o los discos con solo una o dos superficies. Para evitar este «estancamiento del brazo», la cola de peticiones del disco puede dividirse en segmentos, tal que en cada momento se está procesando un segmento hasta que se complete. Los algoritmos SCAN-de-*N*-pasos y FSCAN son dos ejemplos de esta estrategia.

La política de SCAN-de-*N*-pasos divide la cola de peticiones del disco en varias colas de longitud *N*. En cada momento se procesa solo una cola, utilizando SCAN. Mientras se está procesando una cola, las nuevas peticiones se deben añadir a otra cola. Si hay menos de *N* peticiones disponibles al final de una búsqueda, todas ellas se procesan en la próxima búsqueda. Para valores grandes de *N*, el rendimiento de SCAN-de-*N*-pasos se aproxima al de SCAN; para un valor de $N = 1$, se convierte en una política FIFO.

FSCAN es una política que utiliza dos colas. Cuando comienza una búsqueda, todas las peticiones están incluidas en una de las colas, estando la otra vacía. Durante la búsqueda, todas las nuevas peticiones se incluyen en la otra cola. Por tanto, se difiere el servicio de las nuevas peticiones hasta que se han procesado todas las peticiones antiguas.

11.6. RAID

Como se estudió anteriormente, el ritmo de mejora en el rendimiento del almacenamiento secundario ha sido considerablemente menor que en el caso de los procesadores y la memoria principal. Esta diferencia ha hecho del sistema de almacenamiento en disco probablemente el centro principal de interés para mejorar el rendimiento general del computador.

Como ocurre con otras áreas vinculadas con el rendimiento del computador, los diseñadores del almacenamiento de disco saben bien que si un determinado componente ya no puede mejorarse más, para lograr mejoras adicionales en el rendimiento se van a tener que utilizar múltiples componentes en paralelo. En el caso del almacenamiento de disco, esto lleva al desarrollo de vectores de discos que funcionan de manera independiente y en paralelo. Usando múltiples discos, se pueden manejar en paralelo las peticiones de E/S independientes, siempre que los datos solicitados residan en distintos discos. Además, una única petición de E/S se puede ejecutar en paralelo si el bloque de datos que se pretende acceder está distribuido a lo largo de múltiples discos.

Cuando se usan múltiples discos, hay muchas maneras de organizar los datos y la redundancia que se puede añadir para mejorar la fiabilidad. Esto podría dificultar el desarrollo de esquemas de ba-

ses de datos que se puedan utilizar en diversas plataformas y sistemas operativos. Afortunadamente, la industria ha definido un esquema estándar para el diseño de bases de datos en múltiples discos, conocido como RAID (*Redundant Array of Independent Disks*, Vector redundante de discos independientes). El esquema RAID consta de siete niveles[2], del cero al seis. Estos niveles no implican una relación jerárquica, sino que designan a diversas arquitecturas de diseño que comparten tres características comunes:

1. RAID corresponde con un conjunto de unidades físicas de disco tratado por el sistema operativo como un único dispositivo lógico.
2. Los datos están distribuidos a lo largo de las unidades físicas de un vector.
3. La capacidad de redundancia del disco se utiliza para almacenar información de paridad, que garantiza que los datos se pueden recuperar en caso de que falle un disco.

Los detalles de las características enumeradas en segundo y tercer lugar difieren en los distintos niveles RAID. RAID 0 y RAID 1 no incluyen la tercera característica.

El término *RAID* fue originalmente acuñado en un artículo de un grupo de investigadores de la universidad de California en Berkeley [PATT88][3]. El artículo definió varias configuraciones y aplicaciones de los discos RAID e introdujo las definiciones de los niveles RAID que se usan todavía hoy en día. Esta nueva estrategia remplazó los dispositivos de disco de gran capacidad por múltiples dispositivos de menor capacidad distribuyendo los datos de manera que se permitan los accesos simultáneos a los datos de múltiples dispositivos. Con ello, se mejora el rendimiento de E/S y se facilita el crecimiento gradual en la capacidad.

La contribución original de la propuesta RAID es abordar efectivamente la necesidad de redundancia. Aunque permitir que múltiples cabezas y activadores trabajen simultáneamente logra una mayor tasa de E/S y de transferencia, el uso de múltiples dispositivos incrementa la probabilidad de fallo. Para compensar esta pérdida de fiabilidad, RAID utiliza información de paridad almacenada que posibilita la recuperación de los datos perdidos debido a un fallo de un disco.

A continuación, se examinará cada uno de los niveles RAID. La Tabla 11.4, de [MASS97], proporciona un pequeño resumen de los siete niveles. En la tabla, el rendimiento de E/S se muestra tanto en términos de capacidad de transferencia de datos (es decir, posibilidad de mover datos) como de tasa de peticiones de E/S (es decir, capacidad de satisfacer peticiones de E/S), ya que estos niveles RAID presentan diferencias relativas inherentes con respecto a estos dos parámetros. El punto fuerte de cada nivel RAID está resaltado en color. La Figura 11.8 es un ejemplo que muestra el uso de los siete esquemas RAID para proporcionar una capacidad de datos equivalente a la de cuatro discos sin redundancia. La figura resalta la distribución de los datos de usuario y de redundancia e indica los requisitos de almacenamiento relativos de los distintos niveles. Se hará referencia a esta figura a lo largo del siguiente estudio.

---

[2] Algunos investigadores y algunas compañías han definido niveles adicionales. Sin embargo, los siete niveles definidos en esta sección son los conocidos universalmente.

[3] En ese artículo las siglas RAID significaban vector redundante de discos económicos (*Redundant Array of Inexpensive Disks*). El término *económico* se utilizaba para contrastar entre los discos pequeños relativamente baratos usados en el vector RAID y la alternativa consistente en un único disco caro y grande (SLED; *Single Large Expensive Disk*). La opción SLED es simplemente una cosa del pasado, puesto que actualmente se usa una tecnología de disco similar tanto para las configuraciones RAID como para las que no lo son. Por consiguiente, la industria ha adoptado el término *independiente* para enfatizar el hecho de que el vector RAID significa mejoras de rendimiento y fiabilidad significativas.

Tabla 11.4. Niveles RAID.

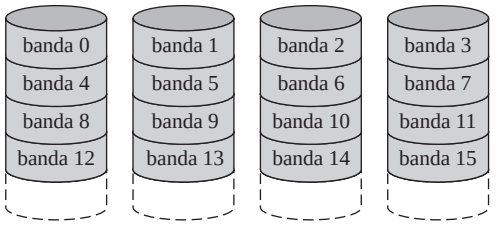
Categoría	Nivel	Descripción	Discos implicados	Disponibilidad de datos	Capacidad de transferencia para datos de E/S grandes	Tasa para peticiones de E/S pequeñas
En bandas	0	Sin redundancia	*N*	Inferior a un único disco	Muy alta	Muy alta tanto para lecturas como para escrituras
Espejo	1	Discos duplicados	2*N*, 3*N*, etc.	Mayor que RAID 2, 3, 4 o 5; menor que RAID 6	Mayor que un único disco para lecturas; similar a un único disco para escrituras	Hasta el doble de un único disco para lecturas; similar a un único disco para escrituras
Acceso paralelo	2	Redundancia mediante Código Hamming	*N* + *m*	Mucho mayor que un único disco; mayor que RAID 3, 4 o 5	La mayor de todas las alternativas mostradas	Aproximadamente el doble de un único disco
	3	Paridad intercalada a nivel de bit	*N* + 1	Mucho mayor que un único disco; comparable a RAID 2, 4 ó 5	La mayor de todas las alternativas mostradas	Aproximadamente el doble de un único disco
Acceso independiente	4	Paridad intercalada a nivel de bloque	*N* + 1	Mucho mayor que un único disco; comparable a RAID 2, 3 o 5	Similar a RAID 0 para lecturas; significativamente inferior a un único disco para escrituras	Similar a RAID 0 para lecturas; significativamente inferior a un único disco para escrituras
	5	Paridad distribuida e intercalada a nivel de bloque	*N* + 1	Mucho mayor que un único disco; comparable a RAID 2, 3 o 4	Similar a RAID 0 para lecturas; inferior a un único disco para escrituras	Similar a RAID 0 para lecturas; generalmente inferior a un único disco para escrituras
	6	Paridad dual distribuida e intercalada a nivel de bloque	*N* + 2	La mayor de todas las alternativas mostradas	Similar a RAID 0 para lecturas; inferior a RAID 5 para escrituras	Similar a RAID 0 para lecturas; significativamente inferior a RAID 5 para escrituras

RAID DE NIVEL 0

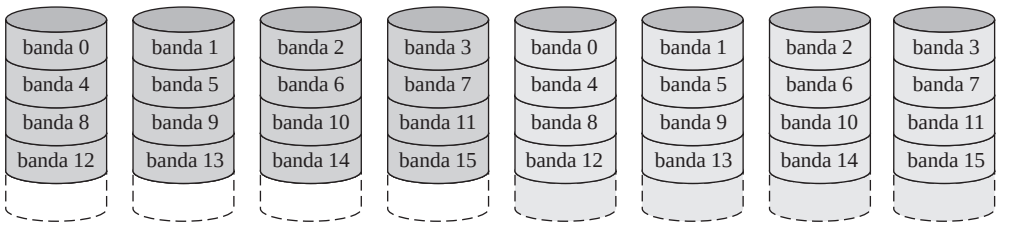
RAID de nivel 0 no es un verdadero miembro de la familia RAID, puesto que no incluye redundancia para mejorar la fiabilidad. Sin embargo, hay algunas aplicaciones, tales como algunas que se usan en los supercomputadores, en los que el rendimiento y la capacidad son primordiales, siendo más importante el bajo coste que la fiabilidad.

En RAID 0, los datos de los usuarios y del sistema están distribuidos a lo largo de todos los discos del vector. Esto tiene una importante ventaja sobre el uso de un único disco grande: si están pendientes dos peticiones de E/S diferentes que solicitan dos bloques de datos distintos, hay una gran probabilidad de que los bloques pedidos estén en diferentes discos. Por tanto, se pueden llevar a cabo las dos peticiones en paralelo, reduciendo el tiempo de espera en la cola de E/S.

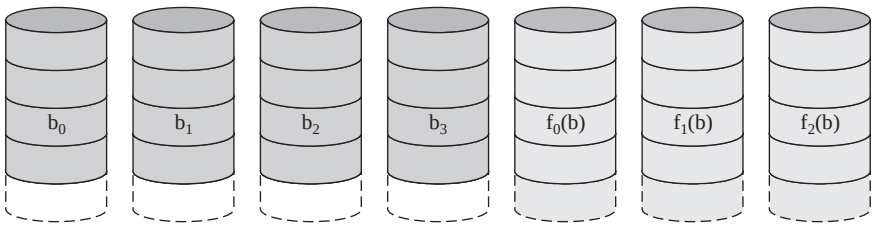
Como ocurre con todos los otros niveles, RAID 0 hace algo más que simplemente distribuir los datos a lo largo de un vector de discos: Los datos están distribuidos *en bandas* (en inglés, *strips*) a lo largo de los discos disponibles. Esto se comprenderá mejor considerando la Figura 11.8. A todos los efectos, es como si los datos de los usuarios y del sistema estuvieran todos almacenados en un único disco lógico. El disco lógico está dividido en bandas; estas bandas pueden ser bloques físicos, sectores o alguna otra unidad. Las bandas se asignan de forma rotatoria a discos físicos consecutivos en el vector RAID. A un conjunto de bandas lógicamente consecutivas tal que a cada



(a) RAID 0 (sin redundancia)



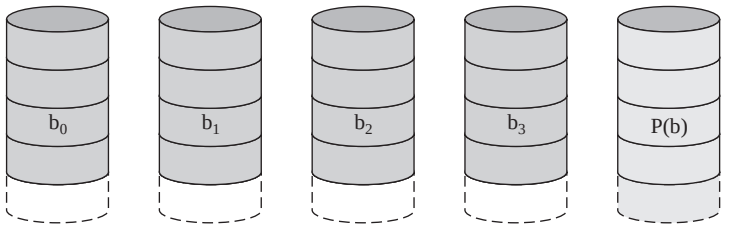
(b) RAID 1 (discos espejo)



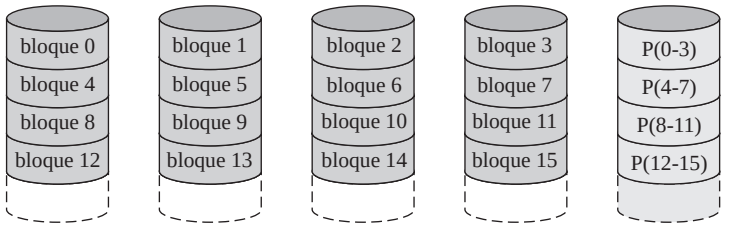
(c) RAID 2 (redundancia mediante código Hamming)

Figura 11.8. Niveles RAID (página 1 de 2).

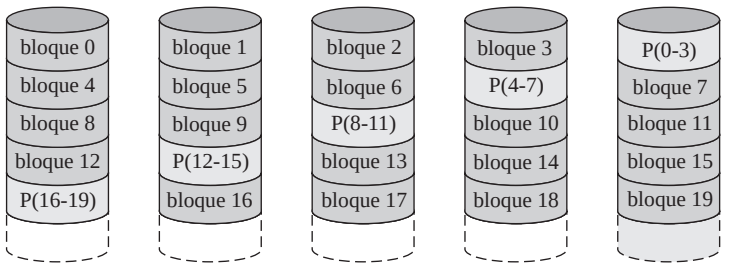
miembro del vector de discos se le asigna exactamente una banda se le llama *lista* (en inglés, *stripe*). En un vector de *n* discos, las primeras *n* bandas lógicas se almacenan físicamente como la primera banda de cada uno de los *n* discos; las segundas *n* bandas lógicas se distribuyen constituyendo la segunda banda de cada disco; y así sucesivamente. La ventaja de esta distribución es que si una única petición de E/S consiste de múltiples bandas contiguas lógicamente, se pueden manejar en paralelo hasta *n* bandas de esta petición, reduciendo considerablemente el tiempo de transferencia de E/S.



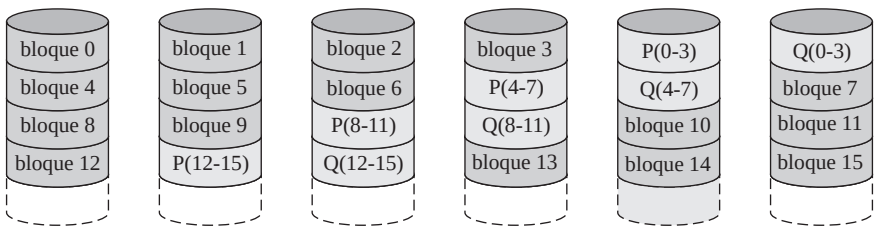
(d) RAID 3 (paridad a nivel de bit)



(e) RAID 4 (paridad distribuida a nivel de bloque)



(f) RAID 5 (paridad distribuida a nivel de bloque)



(g) RAID 6 (redundancia dual)

Figura 11.8. Niveles RAID (página 2 de 2).

RAID 0 para una elevada capacidad de transferencia de datos El rendimiento de cualquiera de los niveles RAID depende de forma crítica de los patrones de peticiones que haya en el sistema y de la distribución de los datos. Estos aspectos se pueden tratar más claramente en el nivel RAID 0, donde el impacto de la redundancia no interfiere en el análisis. En primer lugar, considérese el uso de RAID 0 para lograr una tasa de transferencia de datos elevada. Para que las aplicaciones experimenten esa alta tasa de transferencia, se deben cumplir dos requisitos. Primero, debe existir una elevada capacidad de transferencia a lo largo de todo el camino entre la memoria de la máquina y las unidades de disco individuales, incluyendo los buses internos del controlador, los buses de E/S del sistema, los adaptadores de E/S y los buses de memoria de la máquina.

El segundo requisito es que la aplicación debe hacer peticiones de E/S que accedan eficientemente al vector de discos. Este requisito se cumple si la mayoría de las peticiones corresponde con una gran cantidad de datos contiguos lógicamente, comparado con el tamaño de una banda. En este caso, una única petición de E/S involucra la transferencia de datos en paralelo desde múltiples discos, incrementando la tasa de transferencia efectiva comparada con una transferencia de un único disco.

RAID 0 para una elevada tasa de peticiones de E/S En un entorno orientado a transacciones, el usuario está normalmente más preocupado por el tiempo de respuesta que por la velocidad de transferencia. En el caso de una petición de E/S individual de una cantidad pequeña de datos, el tiempo de E/S está dominado por el movimiento de las cabezas del disco (tiempo de búsqueda) y el movimiento del disco (latencia rotacional).

En un entorno de transacciones, habrá cientos de peticiones de E/S por segundo. Un vector de discos puede proporcionar tasas elevadas de ejecución de E/S repartiendo la carga de E/S entre los múltiples discos. El reparto de carga efectivo sólo se alcanza si normalmente hay múltiples peticiones de E/S pendientes. Esto, a su vez, implica que hay múltiples aplicaciones independientes o una sola aplicación basada en transacciones que es capaz de realizar múltiples peticiones de E/S asíncronas. El rendimiento estará también influido por el tamaño de la banda. Si el tamaño de la banda es relativamente grande, de manera que una única petición de E/S sólo involucre un único acceso a disco, se pueden manejar en paralelo múltiples peticiones de E/S pendientes, reduciendo el tiempo de espera en la cola de cada petición.

RAID DE NIVEL 1

El nivel RAID 1 se diferencia de los niveles del 2 al 6 en el modo en que se logra la redundancia. En estos otros esquemas RAID, se utiliza algún tipo de cálculo de paridad para incluir la redundancia, mientras que en RAID 1, se consigue la redundancia mediante la simple estrategia de duplicar todos los datos. La Figura 11.8b muestra la distribución de datos utilizada, que es igual que la del esquema RAID 0. Sin embargo, en este caso cada banda lógica se asigna a dos discos físicos separados, de manera que cada disco en el vector tenga un disco duplicado que contenga los mismos datos. El esquema RAID 1 se puede implementar también sin la distribución de datos, aunque esto sea menos común.

Hay varios aspectos positivos en la organización RAID 1:

1. Una petición de lectura puede servirse de cualquiera de los dos discos que contienen los datos pedidos, aquél que implique un valor mínimo del tiempo de búsqueda más la latencia rotacional.
2. Una petición de escritura requiere actualizar ambas bandas, pero esto se puede hacer en paralelo. Por tanto, el rendimiento de la escritura lo establece el de la escritura más lenta (es decir, el que implica mayor valor del tiempo de búsqueda más la latencia rotacional). Sin embargo, no hay

una «penalización de escritura» con RAID 1. Los niveles RAID del 2 al 6 implican el uso de bits de paridad. Por tanto, cuando se actualiza una única banda, el software de gestión del vector debe primero calcular y actualizar los bits de paridad además de la banda real involucrada.

3. La recuperación de un fallo es sencilla. Cuando un dispositivo falla, todavía se puede acceder a los datos en el segundo dispositivo.

La principal desventaja de RAID 1 es el coste: requiere el doble de espacio de disco que el correspondiente al disco lógico proporcionado. Debido a esto, una configuración RAID 1 está probablemente limitada a dispositivos que almacenan software y datos del sistema u otros ficheros altamente críticos. En estos casos, RAID 1 proporciona una copia de respaldo en tiempo real de todos los datos de manera que, en el caso de un fallo de disco, todos los datos críticos están inmediatamente disponibles.

En un entorno de transacciones, RAID 1 puede alcanzar tasas elevadas de peticiones de E/S si la mayor parte de las peticiones son de lectura. En esta situación, el rendimiento de RAID 1 puede acercarse al doble del proporcionado por el esquema RAID 0. Sin embargo, si una parte sustancial de las peticiones de E/S son de escritura, puede que no haya ganancia de rendimiento con respecto a RAID 0. RAID 1 puede también proporcionar un mejor rendimiento que RAID 0 para aplicaciones en las que hay transferencias intensivas de datos con un elevado porcentaje de lecturas. La mejora se produce si la aplicación puede dividir cada petición de lectura de manera que participen ambos grupos de discos.

RAID DE NIVEL 2

Los niveles RAID 2 y 3 utilizan una técnica de acceso paralelo. En un vector de acceso paralelo, todos los miembros del disco participan en la ejecución de cada petición de E/S. Normalmente, los ejes de las distintas unidades se sincronizan de manera que en todo momento la cabeza de cada disco esté en la misma posición en todos los discos.

Como ocurre con los otros esquemas RAID, se utiliza una distribución de datos en bandas. En el caso de los esquemas RAID 2 y 3, las bandas son muy pequeñas, a menudo un único byte o palabra. Con RAID 2, se calcula un código de corrección de error con los bits correspondientes de cada disco de datos, almacenándose los bits del código resultante en las correspondientes posiciones de bit en los múltiples discos de paridad. Normalmente, se utiliza un Código Hamming, que es capaz de corregir errores en un único bit y detectar errores en dos.

Aunque RAID 2 requiere menos discos que RAID 1, sigue siendo bastante costoso. El número de discos redundantes es proporcional al logaritmo del número de discos de datos. En una única lectura, se acceden todos los discos de manera simultánea. El controlador del vector recibe los datos pedidos y el código de corrección de error asociado. Si hay un error en un único bit, el controlador puede detectar y corregir el error instantáneamente, con lo que el tiempo de acceso de lectura no se ralentiza. En una única escritura, la operación de escritura debe acceder a todos los discos de datos y de paridad para llevarse a cabo.

El esquema RAID 2 sólo sería una opción efectiva en un entorno en el que se produjeran muchos errores de disco. Dada la alta fiabilidad de las unidades de disco y de los discos propiamente dichos, el esquema RAID 2 es excesivo y no se implementa en la práctica.

RAID DE NIVEL 3

El esquema RAID 3 se organiza de una manera similar al usado en RAID 2. La diferencia estriba en que RAID 3 requiere sólo un disco redundante, con independencia del tamaño del vector de discos.

RAID 3 emplea acceso paralelo, teniendo los datos distribuidos en pequeñas bandas. En lugar de un código de corrección de errores, se calcula un bit de paridad simple para el conjunto de bits almacenados en la misma posición en todos los discos de datos.

Redundancia En el caso de que ocurra un fallo en un dispositivo, se accede al dispositivo de paridad y se reconstruyen los datos desde los dispositivos restantes. Una vez que se remplaza el dispositivo que falló, los datos perdidos pueden restaurarse en el nuevo dispositivo y la operación se reanuda.

La reconstrucción de datos es simple. Considere un vector con cinco unidades de tal manera que los dispositivos que van desde X0 a X3 contienen datos, mientras que la unidad X4 es el disco de paridad. La paridad para el bit *i*-ésimo se calcula de la siguiente manera:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

donde $\oplus$ es una función O-exclusivo.

Supóngase que falla el dispositivo X1. Si se suma $X4(i) \oplus X1(i)$ a ambos lados de la ecuación precedente, se obtiene:

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

Por consiguiente, los contenidos de cada banda de datos en X1 pueden regenerarse a partir de los contenidos de las bandas correspondientes en los restantes discos del vector. Este principio es aplicable a los niveles RAID desde el 3 al 6.

Si se produce un fallo en un disco, todos los datos están todavía disponibles en lo que se denomina modo reducido. En este modo, para las lecturas, los datos perdidos se regeneran sobre la marcha utilizando el cálculo del O-exclusivo. Cuando se escriben datos en un vector RAID 3 en modo reducido, se debe mantener la coherencia de paridad para una posterior regeneración. El retorno a un modo de operación pleno requiere que se remplace el disco que falló y se regenere el contenido completo de dicho disco en el disco nuevo.

Rendimiento Dado que los datos están distribuidos en pequeñas bandas, el esquema RAID 3 puede alcanzar velocidades de transferencia de datos muy elevadas. Cualquier petición de E/S involucrará la transferencia en paralelo de datos de todos los discos de datos. Para transferencias grandes, la mejora en el rendimiento es especialmente notable. Por otro lado, sólo se puede ejecutar una petición de E/S en cada momento. Por tanto, en entorno de transacciones, se ve afectado el rendimiento.

RAID DE NIVEL 4

Los niveles RAID de 4 a 6 utilizan una técnica de acceso independiente. En un vector de acceso independiente, cada disco del vector opera independientemente, de manera que se pueden servir en paralelo peticiones de E/S independientes. Debido a esto, los vectores de acceso independiente son más adecuados para las aplicaciones que requieren tasas elevadas de peticiones de E/S y relativamente menos adecuados para aquéllas que necesitan tasas elevadas de transferencias de datos.

Como ocurre en los otros esquemas RAID, se utiliza una distribución de datos en bandas. En el caso de los esquemas RAID desde el 4 al 6, las bandas son relativamente grandes. En RAID 4, se calcula bit a bit una banda de paridad a partir de las bandas correspondientes de cada disco de datos, almacenándose los bits de paridad resultantes en la banda correspondiente del disco de paridad.

El esquema RAID 4 implica una penalización a las escrituras cuando son de tamaño pequeño. Cada vez que se produce una escritura, el software de gestión del vector, además de modificar los da-

tos de usuario involucrados, debe actualizar los datos de paridad correspondientes. Considere un vector con cinco unidades de tal manera que los dispositivos que van desde X0 a X3 contienen datos, mientras que la unidad X4 es el disco de paridad. Supóngase que se realiza una escritura que sólo involucra una banda en el disco X1. Inicialmente, para cada bit *i* se cumplirá la siguiente relación:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \quad (11.1)$$

A continuación, se muestra el resultado después de la actualización, donde se ha marcado con una comilla los bits potencialmente modificados.

$$\begin{aligned} X4(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \\ &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) \\ &= X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \oplus X1(i) \oplus X1'(i) \\ &= X4(i) \oplus X1(i) \oplus X1'(i) \end{aligned}$$

El conjunto precedente de ecuaciones se deriva como se explica a continuación. La primera línea muestra que un cambio en X1 afectará también al disco de paridad X4. En la segunda línea, se suman los términos [$\oplus$ X1(i) $\oplus$ X1(i)]. Dado que el O-exclusivo de cualquier valor consigo mismo es 0, esto no afecta a la ecuación. Sin embargo, es un paso intermedio conveniente para llegar a la tercera línea, donde simplemente se cambia el orden de los términos. Finalmente, se utiliza la ecuación (11.1) para remplazar los cuatro primeros términos por X4(i).

Para calcular la nueva paridad, el software de gestión del vector debe leer las bandas de datos y paridad antiguas. A continuación, puede actualizar estas dos bandas con los nuevos datos y la nueva paridad calculada. Por consiguiente, cada escritura en una banda requiere dos lecturas y dos escrituras.

En el caso de una escritura de E/S de mayor tamaño que involucre bandas en todas las unidades de disco, la paridad se calcula fácilmente utilizando sólo los nuevos bits de datos. Por tanto, el dispositivo de paridad se puede actualizar en paralelo a los dispositivos de datos sin requerir lecturas o escrituras adicionales.

En cualquier caso, cada operación de escritura debe involucrar el disco de paridad, que, por tanto, puede llegar a ser un cuello de botella.

RAID DE NIVEL 5

El esquema RAID 5 se organiza de manera similar al RAID 4. La diferencia estriba en que el esquema RAID 5 distribuye las bandas de paridad a través de todos los discos. La asignación habitual usa un esquema rotatorio, como se muestra en la Figura 11.8f. Para un vector de *n* discos, la banda de paridad está en un disco diferente para las *n* primeras listas, y, a continuación, se repite este patrón de asignación.

La distribución de las bandas de paridad a través de todos los dispositivos evita el potencial cuello de botella de E/S debido a la existencia de un único disco de paridad que aparece en el esquema RAID 4.

RAID DE NIVEL 6

El esquema RAID 6 fue propuesto en un artículo posterior por los investigadores de Berkeley [KATZ89]. En el esquema RAID 6, se realizan dos cálculos de paridad diferentes, almacenándose en

bloques separados de distintos discos. Por tanto, un vector RAID 6, cuyos datos de usuario requieran *N* discos, necesitará *N* + 2 discos.

La Figura 11.8g muestra este esquema. P y Q son dos algoritmos de comprobación de datos diferentes. Uno de los dos es el cálculo O-exclusivo usado en los esquemas RAID 4 y 5. Sin embargo, el otro es un algoritmo de comprobación de datos independiente. Esto permite regenerar los datos incluso si fallan dos discos que contienen datos de usuario.

La ventaja del esquema RAID 6 es que proporciona una extremadamente alta disponibilidad de datos. Tendrían que fallar tres discos dentro del intervalo correspondiente al tiempo medio de reparación (*Mean Time To Repair*, MTTR) para causar una pérdida de datos. Por otro lado, el esquema RAID 6 incurre en una penalización de escritura sustancial, debido a que cada escritura afecta a dos bloques de paridad.

11.7. CACHE DE DISCO

En la Sección 1.6 y en el Apéndice 1A, se resumen los fundamentos de las memorias cache. El término *memoria cache* se utiliza normalmente aplicado a una memoria que es más pequeña y más rápida que la memoria principal y que se interpone entre la memoria principal y el procesador. Dicha memoria cache reduce el tiempo medio de acceso a memoria explotando el principio de la proximidad.

El mismo principio se puede aplicar a la memoria del disco. Específicamente, una cache de disco es un *buffer* en memoria principal para almacenar sectores del disco. La cache contiene una copia de algunos de los sectores del disco. Cuando se hace una petición de E/S solicitando un determinado sector, se comprueba si el sector está en la cache del disco. En caso afirmativo, se sirve la petición desde la cache. Debido al fenómeno de la proximidad de referencias, cuando se lee un bloque de datos en la cache para satisfacer una única petición de E/S, es probable que haya referencias a ese mismo bloque en el futuro.

CONSIDERACIONES DE DISEÑO

Hay varios aspectos de diseño que son de interés. En primer lugar, cuando se satisface una petición de E/S de la cache de disco, se deben entregar los datos de la cache al proceso solicitante. Esta operación se puede hacer o bien copiando el bloque de datos almacenado en la memoria principal asignada a la cache de disco hasta la memoria asignada al proceso de usuario, o bien utilizando simplemente la técnica de la memoria compartida pasando un puntero al bloque correspondiente de la cache de disco. Esta última estrategia ahorra el tiempo de la transferencia de memoria a memoria y también permite el acceso compartido por parte de otros procesos utilizando el modelo de lectores/escritores descrito en el Capítulo 5.

Un segundo aspecto de diseño está relacionado con la estrategia de remplazo. Cuando se trae un nuevo sector a la cache de disco, se debe remplazar uno de los bloques existentes. Se trata de un problema idéntico al presentado en el Capítulo 8, donde se requería un algoritmo de remplazo de páginas. Se han probado diversos algoritmos. El algoritmo más frecuentemente utilizado es el del menos recientemente usado (*Least Recently Used*, LRU): se remplaza el bloque que ha estado en la cache más tiempo sin ser accedido. Lógicamente, la cache consiste de una pila de bloques, estando el bloque más recientemente accedido en lo más alto de la pila. Cuando se accede a un bloque en la cache, se mueve de su posición actual en la pila hasta la cima de la misma. Cuando se trae un bloque desde la memoria secundaria, se elimina el bloque que estaba al final de la pila, situándose el bloque entrante en la cima de la pila. Naturalmente, no es necesario mover realmente estos bloques dentro de la memoria principal, puesto que se puede asociar una pila de punteros a la cache.

Otra posibilidad es el algoritmo del **menos frecuentemente usado (*Least Frequently Used*, LFU).** Se remplaza el bloque del conjunto que ha experimentado la menor cantidad de referencias. El algoritmo LFU puede implementarse asociando un contador con cada bloque. Cuando se trae un bloque, se le asigna un contador con un valor igual a 1, incrementando el contador en 1 por cada referencia al bloque. Cuando se requiere un remplazo, se selecciona el bloque con un contador más pequeño. Intuitivamente, podría parece que LFU es más apropiado que LRU porque utiliza información más pertinente de cada bloque en el proceso de selección.

Un algoritmo LFU sencillo conlleva el siguiente problema. Puede ocurrir que algunos bloques se accedan en términos globales de forma relativamente infrecuente, pero cuando se hace referencia a ellos, se producen referencias repetidas durante cortos intervalos de tiempo debido a la proximidad, de manera que se genera un contador de referencias elevado. Cuando se acaba el intervalo, el valor del contador de referencias puede llevar a conclusiones erróneas, no reflejando el grado de probabilidad de que el bloque se acceda de nuevo en un breve plazo de tiempo. Por tanto, el efecto de la proximidad puede realmente causar que el algoritmo LFU tome decisiones inadecuadas.

Para resolver esta deficiencia del algoritmo LFU, en [ROBI90] se propone una técnica denominada remplazo basado en la frecuencia. Por motivos pedagógicos, considere una versión simplificada, mostrada en la Figura 11.9a. Los bloques se organizan lógicamente en una pila, como en el algoritmo LRU. Una cierta porción de la parte superior de la pila se considera separada como una sección nueva. Cuando hay un acierto en la cache, el bloque accedido se mueve a la parte superior de la pila. Si el bloque ya estaba en la sección nueva, su contador de referencias no se incrementa; en caso contrario, se incrementa en 1. Si la sección nueva es suficientemente grande, se consigue que permanezcan inalterados los contadores de referencias de los bloques que se acceden repetidamente dentro de un breve intervalo de tiempo. En caso de fallo, se escoge para remplazar el bloque con el contador de referencias más bajo que no esté en la sección nueva, seleccionando el bloque usado menos recientemente en caso de empate.

Los autores explican que esta estrategia sólo logra una ligera mejora con respecto al algoritmo LRU. El problema es el siguiente:

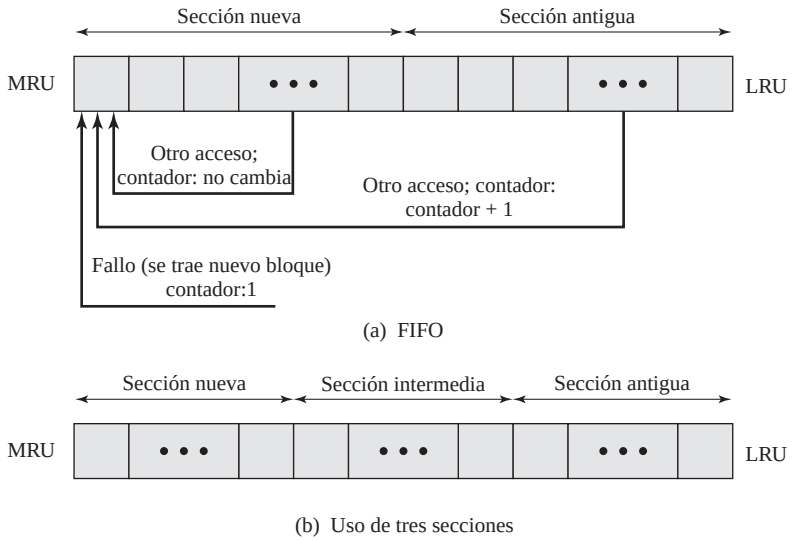


Figura 11.9. Remplazo basado en la frecuencia.

1. En un fallo de cache, se incluirá un nuevo bloque en la sección nueva, con un contador igual a 1.
2. El contador continúa con un valor igual a 1 mientras el bloque permanezca en la sección nueva.
3. Pasado un cierto tiempo, el bloque sale de la sección nueva, con su contador todavía igual a 1.
4. Si no se vuelve a acceder al bloque con cierta prontitud, es muy probable que se remplace porque tiene necesariamente el menor contador de referencias entre todos los bloques que no están en la sección nueva. En otras palabras, parece que no hay un intervalo suficientemente largo para que los bloques que abandonan la sección nueva puedan aumentar el valor de su contador de referencias incluso aunque se accedan con una cierta frecuencia.

Una mejora adicional afronta este problema: dividir la pila en tres secciones: nueva, intermedia y antigua. Como antes, los contadores de referencias no se incrementan en el caso de bloques incluidos en la sección nueva. Sin embargo, sólo se seleccionan para remplazo los bloques en la sección antigua. Asumiendo una sección intermedia grande, este esquema proporciona a los bloques que se accedan con una cierta frecuencia una oportunidad para aumentar el valor de sus contadores de referencias antes de llegar a ser candidatos al remplazo. Las simulaciones realizadas por los autores indican que esta política mejorada es significativamente mejor que LRU o LFU.

Con independencia de cada estrategia de remplazo específica, el remplazo puede realizarse bajo demanda o planificado por anticipado. En el primer caso, sólo se remplaza un sector cuando se necesita un hueco. En el último caso, se generan simultáneamente varios huecos. El motivo de esta última estrategia está relacionado con la necesidad de volver a escribir los sectores en el disco. Si se trae a la cache un sector y sólo se lee, cuando se remplace no será necesario escribirlo de nuevo en el disco. Sin embargo, si se actualiza el sector, es necesario volverlo a escribir antes de remplazarlo. En este último caso, es razonable agrupar las escrituras y ordenarlas para minimizar el tiempo de búsqueda.

CONSIDERACIONES DE RENDIMIENTO

En este caso se aplican las mismas consideraciones de rendimiento analizadas en el Apéndice 1A. El aspecto del rendimiento de la cache se reduce en sí mismo a una cuestión de si se puede alcanzar una determinada tasa de fallos. Esto dependerá del comportamiento de las referencias al disco con respecto a la proximidad y del algoritmo de remplazo, así como de otros factores de diseño. Sin embargo, la tasa de fallos principalmente está en función del tamaño de la cache del disco. La Figura 11.10 resume los resultados de varios estudios usando LRU, uno para un sistema UNIX ejecutado en un VAX [OUST85] y otro para sistemas operativos de *mainframes* de IBM [SMIT85]. La Figura 11.11 muestra los resultados de simulaciones del algoritmo de remplazo basado en la frecuencia. Una comparación de las dos figuras resalta uno de los riesgos de este tipo de evaluación del rendimiento. Las figuras parecen mostrar que el algoritmo LRU supera al algoritmo de remplazo basado en la frecuencia. Sin embargo, cuando se comparan patrones de referencias idénticos utilizando la misma estructura de cache, el algoritmo de remplazo basado en la frecuencia es superior. Por tanto, la secuencia exacta de patrones de referencia, así como los aspectos de diseño relacionados como el tamaño de bloque, tendrá una profunda influencia en el rendimiento alcanzado.

11.8. E/S DE UNIX SVR4

En UNIX, cada dispositivo de E/S está asociado con un fichero especial, que lo gestiona el sistema de ficheros y se lee y escribe de la misma manera que los ficheros de datos de usuario. Esto proporciona una interfaz bien definida y uniforme para los usuarios y los procesos. Para leer o escribir

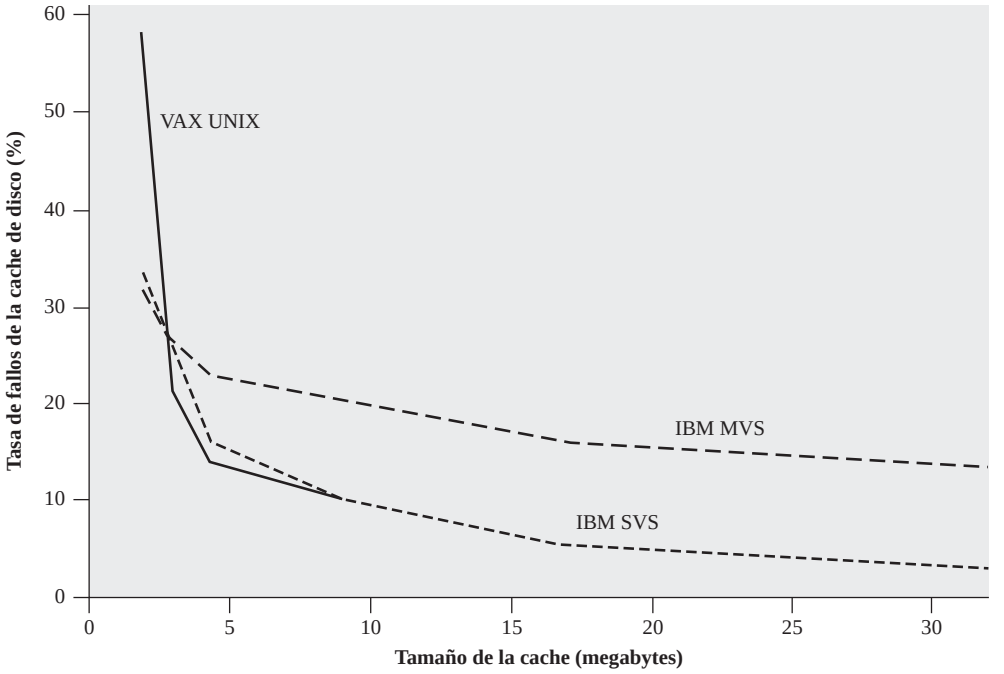


Figura 11.10. Algunos resultados del rendimiento de una cache de disco usando LRU.

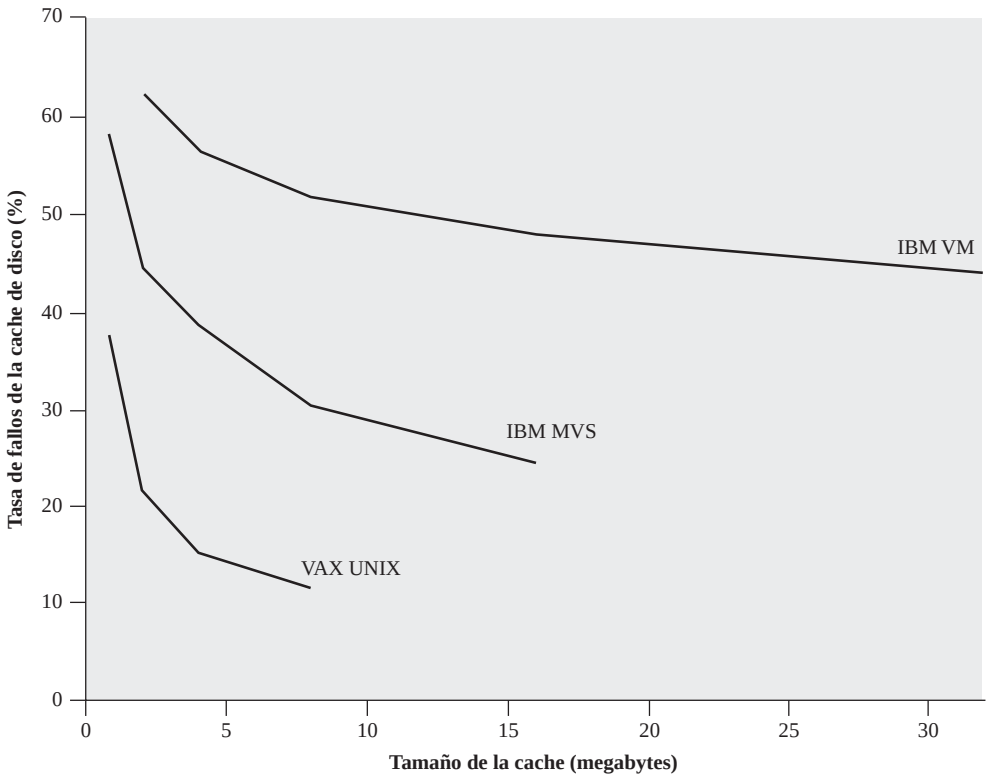


Figura 11.11. Rendimiento de la cache de disco utilizando un remplazo basado en la frecuencia [ROBI90].

de un dispositivo, se realizan peticiones de lectura o escritura en el fichero especial asociado con el dispositivo.

La Figura 11.12 muestra la estructura lógica del sistema de E/S. El subsistema de ficheros gestiona los ficheros en los dispositivos de almacenamiento secundario. Además, sirve como interfaz a los dispositivos para los procesos, debido a que se tratan como ficheros.

Hay dos tipos de E/S en UNIX: con *buffer* y sin *buffer*. La E/S con *buffer* pasa a través de los *buffers* del sistema, mientras que la E/S sin *buffer* normalmente involucra al sistema de DMA, de manera que la transferencia tiene lugar directamente entre el módulo de E/S y el área de E/S del proceso. En la E/S con *buffer*, se utilizan dos tipos de *buffers*: caches de *buffers* del sistema y colas de caracteres.

CACHE DE *BUFFERS*

La cache de *buffers* de UNIX es esencialmente una cache de disco. Las operaciones de E/S sobre el disco se manejan a través de la cache de *buffers*. La transferencia de datos entre la cache de *buffers* y el espacio del proceso de usuario siempre se realiza utilizando DMA. Dado que tanto la cache de *buffers* como el área de E/S del proceso están en la memoria principal, se utiliza en este caso el sistema de DMA para realizar una copia de memoria a memoria. Esta operación no consumirá ningún ciclo del procesador, pero consume ciclos del bus.

Para gestionar la cache de *buffers* se usan tres listas:

- **Lista de libres.** Lista de todos los huecos de la cache (en UNIX a un hueco se le denomina *buffer*, cada hueco almacena un sector del disco) que están disponibles para su asignación.
- **Lista de dispositivos.** Lista de todos los *buffers* que están actualmente asociados con cada disco.
- **Cola de E/S del manejador.** Lista de los *buffers* sobre los que se está realmente realizando E/S o esperando por la misma para un determinado dispositivo.

Todos los *buffers* deberían estar en la lista de libres o en la cola de E/S del manejador. Una vez que se asocia un *buffer* a un dispositivo, permanece asociado al mismo incluso aunque esté en la lista de libres, hasta que se reutilice realmente y pase a estar vinculado con otro dispositivo. Estas listas se gestionan usando punteros asociados a cada *buffer* en vez de utilizar listas físicamente independientes.

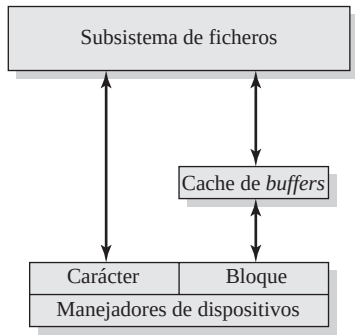


Figura 11.12. Estructura de la E/S de UNIX.

Cuando se hace una referencia a un número de bloque físico de un determinado dispositivo, el sistema operativo primero comprueba si el bloque está en la cache de *buffers*. Para minimizar el tiempo de búsqueda, la lista de dispositivos se organiza como una tabla *hash*, utilizando una técnica similar al desbordamiento con encadenamiento estudiado en el Apéndice 8A (Figura 8.27b). La Figura 11.13 muestra la organización general de la cache de *buffers*. Hay una tabla *hash* de longitud fija que contiene punteros a la cache de *buffers*. Cada referencia a un (dispositivo#, bloque#) se corresponde con una determinada entrada en la tabla *hash*. El puntero de dicha entrada apunta al primer *buffer* de la cadena. Un puntero *hash* incluido en cada *buffer* señala al siguiente *buffer* en la cadena correspondiente a esa entrada de la tabla *hash*. Por tanto, se cumple que para todas las referencias (dispositivo#, bloque#) que se corresponden con la misma entrada de la tabla *hash*, si el bloque correspondiente está en la cache de *buffers*, ese *buffer* estará en la cadena vinculada a esa entrada de la tabla *hash*. Por consiguiente, la longitud de la búsqueda en la cache de *buffers* se reduce por un factor de orden *N*, siendo *N* la longitud de la tabla *hash*.

Para el remplazo de un bloque, se utiliza el algoritmo del menos recientemente usado. Una vez que a un *buffer* se le ha asignado un bloque de disco, no puede utilizarse para otro bloque hasta que se hayan usado más recientemente los restantes *buffers*. La lista de libres mantiene el orden requerido por el algoritmo.

COLA DE CARACTERES

Los dispositivos orientados a bloques, como el disco y la cinta, se pueden gestionar eficientemente usando la cache de *buffers*. Sin embargo, los dispositivos orientados a caracteres, como los terminales y las impresoras, requieren otro tipo de *buffers*: las colas de caracteres. Una cola de caracteres, en

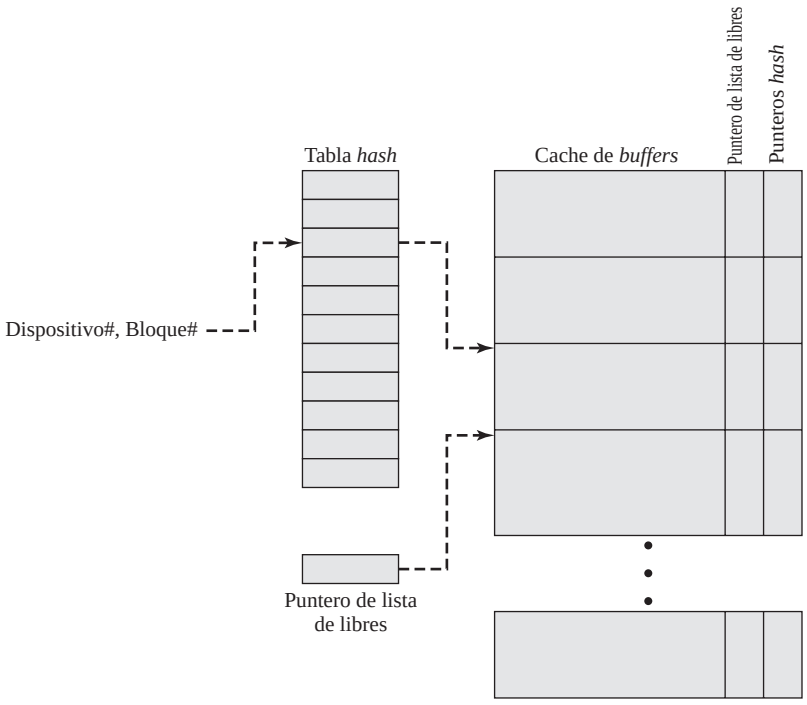


Figura 11.13. Organización de la cache de *buffers* en UNIX.

ciertos casos, es escrita por el dispositivo de E/S y leída por el proceso, mientras que en otras ocasiones es el proceso el que escribe y el dispositivo el que lee. En ambos casos, se utiliza el modelo productor/consumidor presentado en el Capítulo 5. Por tanto, las colas de caracteres sólo se pueden leer una vez; cuando se lee un carácter, de hecho, se destruye. Esto contrasta con la cache de *buffers*, que puede leerse múltiples veces y, por tanto, corresponde con el modelo lectores/escritores (también estudiado en el Capítulo 5).

E/S SIN *BUFFER*

La E/S sin *buffer*, que consiste simplemente en una operación de DMA entre el dispositivo y el espacio del proceso, es siempre el método más rápido de realizar E/S por parte de un proceso. Un proceso que está realizando E/S sin *buffer* queda residente en la memoria principal, no pudiendo resultar expulsado. Esto reduce la oportunidad de usar el intercambio al fijar como residente parte de la memoria principal, lo que conlleva una disminución del rendimiento global del sistema. Asimismo, el dispositivo de E/S queda asociado al proceso durante la duración de la transferencia, no estando disponible mientras tanto para otros procesos.

DISPOSITIVOS DE UNIX

Entre las categorías de dispositivos reconocidos por UNIX se encuentran las siguientes:

- Dispositivos de disco.
- Dispositivos de cinta.
- Terminales.
- Líneas de comunicación.
- Impresoras.

La Tabla 11.5 muestra los tipos de E/S apropiados para cada tipo de dispositivo. Los dispositivos de disco, que se utilizan abundantemente en UNIX, son orientados a bloques y, en principio, suelen tener un rendimiento razonablemente bueno. Por tanto, la E/S de estos dispositivos suele ser sin *buffer* o usando la cache de *buffers*. Los dispositivos de cinta son funcionalmente similares a los dispositivos de disco utilizando unos esquemas de E/S similares.

Tabla 11.5. E/S de dispositivos en UNIX.

	E/S sin *buffer*	Cache de *buffers*	Cola de caracteres
Unidad de disco	X	X	
Unidad de cinta	X	X	
Terminales			X
Líneas de comunicación			X
Impresoras	X		X

Dado que los terminales requieren un intercambio relativamente lento de caracteres, la E/S de terminal utiliza normalmente la cola de caracteres. De manera similar, las líneas de comunicación re-

quieren el procesamiento en serie de bytes de datos de entrada o salida, por lo que se manejan mejor con colas de caracteres. Por último, el tipo de E/S utilizado por una impresora dependerá generalmente de su velocidad. Las impresoras lentas utilizan normalmente colas de caracteres, mientras que una rápida puede emplear E/S sin *buffer*. En una impresora rápida puede utilizarse una cache de *buffers*. Sin embargo, dado que los datos que van a la impresora nunca se reutilizan, la sobrecarga de la cache de *buffers* es innecesaria.

11.9. E/S DE LINUX

En términos generales, el sistema de E/S del núcleo de Linux es muy similar al de otras implementaciones de UNIX, como es el caso de SVR4. El núcleo de Linux asocia un fichero especial con cada manejador de dispositivo de E/S, distinguiéndose entre dispositivos de bloques, de caracteres y de red. En esta sección, se estudiarán varias características del sistema de E/S de Linux.

PLANIFICACIÓN DE DISCO

El planificador de disco por defecto en Linux 2.4 se le conoce con el nombre de ascensor de Linus, que es una variación del algoritmo LOOK estudiado en la Sección 11.3. En Linux 2.6, además del algoritmo del ascensor, se han incluido dos algoritmos adicionales: el planificador de E/S basado en plazos y el planificador de E/S previsor [LOVE04b]. A continuación, se estudiará cada uno de ellos.

EL PLANIFICADOR DEL ASCENSOR

El planificador del ascensor mantiene una única cola con las peticiones de lectura y escritura en el disco, realizando operaciones de ordenamiento y agrupamiento sobre la cola. En términos generales, el planificador del ascensor mantiene la lista de peticiones ordenadas por el número de bloque. De esta manera, cuando se manejan las peticiones de disco, el dispositivo se mueve en una única dirección, satisfaciendo cada petición según la encuentra. Esta estrategia general se mejora de la siguiente manera. Cuando se añade una nueva petición a la cola, se consideran en este orden las siguientes cuatro operaciones:

1. Si existe una petición pendiente en la cola de tal manera que la nueva petición corresponde con el mismo sector del disco o uno inmediatamente adyacente al requerido por dicha petición previa, la petición existente y la nueva se mezclan en una sola.
2. Si hay una petición en la cola que es suficientemente antigua, la nueva petición se inserta al final de la cola.
3. Si hay una posición adecuada, la nueva petición se sitúa en el orden correspondiente.
4. Si no hay una posición adecuada, la nueva petición se sitúa al final de la cola.

PLANIFICADOR BASADO EN PLAZOS

La segunda operación de la lista precedente intenta evitar la inanición de una petición, pero no es muy efectiva [LOVE04a]. No intenta servir peticiones en un plazo de tiempo determinado, sino que simplemente deja de insertar las peticiones en orden después de un plazo conveniente. En el esquema del ascensor se manifiestan dos problemas. El primer problema es que se puede retrasar una petición

de un bloque distante durante un tiempo considerable debido a que la cola se actualiza dinámicamente. Por ejemplo, considere el siguiente flujo de peticiones de los bloques de disco: 20, 30, 700 y 25. El planificador del ascensor modifica el orden de estas peticiones de manera que se sitúan en la cola en el orden 20, 25, 30 y 700, estando la petición del bloque 20 en la cabeza de la cola. Si llega una secuencia continua de peticiones que corresponden con bloques de baja numeración, la petición del bloque 700 se retrasa indefinidamente.

Un problema incluso más serio es el de la distinción entre peticiones de lectura y de escritura. Normalmente, una petición de escritura se realiza asíncronamente. Es decir, una vez que un proceso solicita una petición de escritura, no necesita esperar hasta que realmente se lleve a cabo la petición. Cuando una aplicación solicita una escritura, el núcleo copia los datos en un *buffer* apropiado, que se escribirá cuando se considere oportuno. Una vez que se copian los datos en el *buffer* del núcleo, la aplicación puede continuar. Sin embargo, en muchas operaciones de lectura, el proceso, antes de continuar, debe esperar hasta que se entreguen los datos pedidos a la aplicación. Por tanto, un flujo de peticiones de escritura (por ejemplo, para escribir en el disco un fichero grande) puede bloquear una petición de lectura durante un tiempo considerable y, con ello, bloquear también al proceso.

Para resolver estos problemas, se utiliza el planificador de E/S basado en plazos que usa tres colas (Figura 11.14). Cada nueva petición se incluye en la cola ordenada del ascensor, como en el algoritmo previo. Asimismo, esa misma petición se sitúa al final de una cola FIFO de lectura en el caso de una petición de lectura o de una cola FIFO de escritura si se trata de una petición de escritura. Por tanto, las colas de lectura y de escritura almacenan una lista de peticiones en el orden en que éstas se hicieron. Asociado con cada petición hay un tiempo de expiración, con un valor por defecto de 0,5 segundos en caso de una petición de lectura y de 5 segundos en el de una escritura. Generalmente, el planificador extrae peticiones de la cola ordenada. Cuando se completa una petición, se elimina de la cabeza de la cola ordenada y también de la cola FIFO correspondiente. Sin embargo, cuando se cumple el tiempo de expiración del elemento de la cabeza de una de las colas FIFO, el planificador pasa a dar servicio de esa cola FIFO, extrayendo la petición expirada, junto con algunas de las siguientes peticiones de la cola. Según se sirve cada petición, se borra de la cola ordenada.

El esquema del planificador de E/S basado en plazos supera el problema de la inanición y también el problema de las lecturas frente a las escrituras.

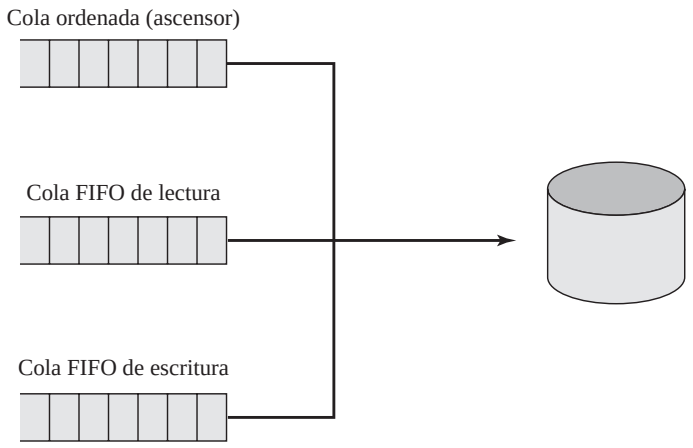


Figura 11.14. El planificador de E/S basado en plazos de Linux.

PLANIFICADOR DE E/S PREVISOR

El planificador del ascensor original y el basado en plazos están diseñados para servir una nueva petición tan pronto como se completa la petición existente, manteniendo, por tanto, el disco lo más ocupado que sea posible. Esta misma política se aplica a todos los algoritmos de planificación estudiados en la Sección 11.5. Sin embargo, esta política puede ser contraproducente si hay numerosas peticiones de lectura síncronas. Normalmente, una aplicación esperará hasta que se complete una petición de lectura y estén los datos disponibles antes de realizar la siguiente petición. El pequeño retardo que hay entre que se reciben los datos de la última lectura y la solicitud de la siguiente lectura permite al planificador dedicarse a otra petición pendiente y servir esa petición.

Gracias al principio de la proximidad, es probable que las lecturas sucesivas del mismo proceso se encuentren en bloques de disco que estén los unos cerca de los otros. Si el planificador tuviera un retardo de un breve periodo de tiempo después de servir una petición de lectura, de manera que pudiera comprobar si se hace una nueva petición de lectura cercana, el rendimiento global del sistema podría mejorarse. Ésta es la filosofía en la que se basa el planificador previsor, propuesto en [IYER01], e implementado en Linux 2.6.

En Linux, el planificador previsor está superpuesto sobre el planificador basado en plazos. Cuando se sirve una petición de lectura, el planificador previsor causa que el sistema de planificación se retrase hasta 6 milisegundos, dependiendo de la configuración. Durante este pequeño retardo, hay una oportunidad apreciable de que la aplicación que solicitó la última petición de lectura genere otra petición de lectura en la misma región del disco. En caso de que sea así, esa petición se servirá inmediatamente. Si no se produce esa petición de lectura, el planificador continúa utilizando el algoritmo de planificación basado en plazos.

[LOVE04b] muestra los resultados de dos pruebas del algoritmo de planificación de Linux. La primera prueba consistió en la lectura de un fichero de 200 MB mientras se hace una larga escritura secuencial ejecutando en segundo plano. En la segunda prueba se realizó una lectura de un fichero grande ejecutando en segundo plano mientras que se leen todos los ficheros del árbol de código fuente del núcleo. En la siguiente tabla se muestran los resultados de las pruebas:

Planificador de E/S y núcleo	Prueba 1	Prueba 2
Ascensor de Linus en 2.4	45 segundos	30 minutos y 28 segundos
Planificador de E/S basado en plazos en 2.6	40 segundos	3 minutos y 30 segundos
Planificador de E/S previsor en 2.6	4,6 segundos	15 segundos

Como se puede apreciar, la mejora del rendimiento depende de la naturaleza de la carga de trabajo. Sin embargo, en ambos casos, el planificador previsor proporciona una mejora muy considerable.

CACHE DE PÁGINAS DE LINUX

En Linux 2.2 y en las versiones anteriores, el núcleo mantiene una cache de páginas para las lecturas y escrituras de los ficheros ordinarios del sistema de ficheros y para las páginas de memoria virtual, y una cache de *buffers* independiente para la E/S de bloques. En Linux 2.4 y en las versiones posteriores, hay una única cache de páginas unificada que está involucrada en todo el tráfico entre el disco y la memoria principal.

La cache de páginas conlleva dos beneficios. En primer lugar, cuando llega el momento de escribir en el disco las páginas modificadas, se puede agrupar un conjunto de las mismas ordenándolas

adecuadamente y escribiéndolas, por tanto, eficientemente. En segundo lugar, gracias al principio de la proximidad temporal, las páginas incluidas en la cache de páginas se accederán probablemente de nuevo antes de ser expulsadas de la cache, evitando de esta forma una operación de E/S de disco.

Las páginas modificadas se escriben en el disco en dos situaciones:

- Cuando la cantidad de memoria libre llega a ser menor que un determinado umbral, el núcleo reduce el tamaño de la cache de páginas liberando memoria que va a añadirse al conjunto de memoria libre disponible en el sistema.
- Cuando las páginas modificadas envejecen más allá de un determinado umbral, se escriben en el disco varias páginas modificadas.

11.10. E/S DE WINDOWS

La Figura 11.15 muestra el gestor de E/S de Windows. Este gestor es responsable de todo el sistema de E/S del sistema operativo y proporciona una interfaz uniforme a la que todos los tipos de manejadores pueden llamar.

MÓDULOS DE E/S BÁSICOS

El gestor de E/S consta de cuatro módulos:

- **Gestor de cache.** El gestor de cache maneja la gestión de la cache para todo el subsistema de E/S, proporcionando un servicio de cache en memoria principal para todos los componentes de sistemas de ficheros y de red. Se puede incrementar y decrementar dinámicamente el tamaño de la cache dedicada a una determinada actividad según varíe la cantidad de la memoria física disponible. El gestor de cache incluye dos servicios para mejorar el rendimiento general:
  - **Escritura perezosa.** El sistema registra las actualizaciones sólo en la cache y no en el disco. Posteriormente, cuando el grado de utilización del procesador es bajo, el gestor de cache escribe los cambios en el disco. Si se actualiza un determinado bloque de cache mientras tanto, hay un ahorro neto.
  - **Compromiso perezoso.** Este servicio es similar a la escritura perezosa pero para procesamiento de transacciones. En vez de registrar de manera inmediata que una transacción se ha completado con éxito, el sistema almacena en la cache la información comprometida y, posteriormente, un proceso ejecutando en segundo plano la escribe en el *log* del sistema de ficheros.

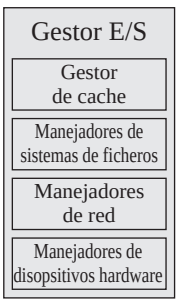


Figura 11.15. Gestor de E/S de Windows.

- **Manejadores de sistemas de ficheros.** El gestor de E/S trata a un manejador de sistema de ficheros de la misma manera que a otro manejador y encamina los mensajes destinados a determinados volúmenes al manejador software correspondiente a ese adaptador de dispositivo.
- **Manejadores de red.** Windows incluye una gestión de red integrada y proporciona soporte para aplicaciones distribuidas.
- **Manejadores de dispositivos hardware.** Estos manejadores acceden a los registros hardware de los dispositivos periféricos a través de puntos de entrada de bibliotecas dinámicamente enlazadas del Ejecutivo de Windows. Existe un conjunto de estas rutinas para cada plataforma en la que puede ejecutar Windows; gracias a que los nombres de las rutinas son los mismos para todas las plataformas, el código fuente de los manejadores de dispositivos de Windows se puede transportar a distintos tipos de procesadores.

E/S SÍNCRONA Y ASÍNCRONA

Windows ofrece dos modos de operación para la E/S: asíncrono y síncrono. El modo asíncrono se utiliza para optimizar el rendimiento de la aplicación, siempre que sea posible. Con la E/S asíncrona, una aplicación inicia una operación de E/S prosiguiendo su ejecución mientras se lleva a cabo la petición de E/S. Con la E/S síncrona, la aplicación se bloquea hasta que se completa la operación de E/S.

La E/S asíncrona es más eficiente, desde el punto de vista del hilo que solicita la operación, ya que le permite continuar ejecutando mientras que el gestor de E/S encola la operación de E/S y, posteriormente, se realiza dicha operación. Sin embargo, la aplicación que invoca la operación de E/S asíncrona necesita alguna manera de determinar cuándo se completa la operación. Windows proporciona cuatro técnicas diferentes para la notificación de la finalización de una operación de E/S:

- **Activación de un objeto dispositivo del núcleo.** Con esta estrategia, se activa un indicador asociado a un objeto dispositivo cuando se completa una operación en ese objeto. El hilo que solicitó la operación de E/S puede continuar ejecutando hasta que alcance un punto donde debe parar a la espera de que se complete la operación de E/S. En ese instante, el hilo puede esperar hasta que se complete la operación y, a continuación, continuar su ejecución. Esta técnica es sencilla y fácil de utilizar pero no es apropiada para manejar múltiples peticiones de E/S. Por ejemplo, si un hilo necesita realizar múltiples acciones simultáneas sobre un único fichero, como leer un fragmento del fichero y escribir en otra parte del mismo, con esta técnica, el hilo no puede distinguir entre la finalización de la lectura y el de la escritura. Sabría simplemente que alguna operación de E/S solicitada sobre este fichero se ha completado.
- **Activación de un objeto evento del núcleo.** Esta técnica permite que haya múltiples peticiones de E/S simultáneas sobre un dispositivo o fichero. El hilo crea un evento por cada petición. Posteriormente, el hilo puede esperar por una sola de estas peticiones o por todas ellas.
- **E/S con alerta.** Esta técnica utiliza una cola asociada a un hilo, conocida como la cola de llamadas a procedimientos asíncronos (*Asynchronous Procedure Call*, APC). En este caso, el hilo realiza peticiones de E/S y el gestor de E/S sitúa los resultados de estas peticiones en la cola APC del hilo solicitante.
- **Puertos de finalización de E/S.** Esta técnica se utiliza en un servidor de Windows para optimizar el uso de los hilos. Esencialmente, consiste en que está disponible un conjunto de hilos para su uso, de manera que no es necesario crear un nuevo hilo para manejar una nueva petición.

RAID SOFTWARE

Windows proporciona dos tipos de configuraciones RAID, definidas en [MS96] de la siguiente forma:

- **RAID hardware.** Discos físicos independientes agrupados en uno o más discos lógicos por el hardware del controlador de disco o del armario de almacenamiento de disco.
- **RAID software.** Espacio de disco que no es contiguo agrupado en una o más particiones lógicas por el manejador de discos software tolerante a fallos, FTDISK.

En el RAID hardware, la interfaz del controlador maneja la creación y la regeneración de la información redundante. El RAID software, disponible en Windows Server, implementa la funcionalidad RAID como parte del sistema operativo y puede utilizarse con cualquier conjunto de múltiples discos. El sistema de RAID software implementa RAID 1 y RAID 5. En el caso del RAID 1 (duplicado de discos), los dos discos que contienen las particiones primaria y duplicada pueden estar asociados al mismo controlador de disco o a diferentes. A esta última configuración se la denomina *duplexing de disco*.

11.11. RESUMEN

La interfaz de un computador al mundo exterior corresponde con su arquitectura de E/S. Esta arquitectura está diseñada para proporcionar un medio sistemático de controlar la interacción con el mundo exterior y proveer al sistema operativo de la información que necesita para gestionar eficientemente la actividad de E/S.

El sistema de E/S se divide generalmente en varios niveles, de forma que los niveles inferiores tratan con los detalles que están más cercanos a las funciones físicas que se van a realizar y los niveles superiores tratan con la E/S de una manera lógica y genérica. Como resultado, los cambios en los parámetros del hardware no van a afectar a la mayoría del software de E/S.

Un aspecto fundamental de la E/S es la utilización de *buffers* que gestiona el sistema de E/S en lugar de los procesos de aplicación. El uso de *buffers* amortigua las diferencias entre la velocidad interna del computador y la velocidad de los dispositivos de E/S. El uso de *buffers* también desvincula la transferencia real de E/S del espacio de direcciones del proceso de aplicación. Esto permite al sistema operativo más flexibilidad a la hora de realizar sus funciones de gestión de memoria.

El aspecto de la E/S que tiene un mayor impacto en el rendimiento general del sistema es la E/S de disco. Por ello, ha habido un mayor auge de la investigación y el diseño en esta área que en otros tipos de E/S. La planificación de disco y la cache de disco constituyen dos de las estrategias más frecuentemente utilizadas para mejorar el rendimiento de E/S del disco.

En cualquier momento, puede haber una cola de peticiones de E/S en el mismo disco. El objetivo de la planificación del disco es satisfacer estas peticiones de manera que se minimice el tiempo de búsqueda mecánica del disco y, con ello, se mejore el rendimiento. En esta planificación entran en juego aspectos tales como la distribución física de las peticiones pendientes, así como consideraciones sobre la proximidad.

Una cache de disco es un *buffer*, almacenado usualmente en la memoria principal, que funciona como una cache de bloques de disco entre la memoria de disco y el resto de la memoria principal. Gracias al principio de la proximidad, el uso de una cache de disco debería reducir considerablemente el número de transferencias de E/S de bloques entre la memoria principal y el disco.

11.12. LECTURAS Y SITIOS WEB RECOMENDADOS

En la mayoría de los libros de arquitectura de computadores se pueden encontrar estudios generales de la E/S del computador, como [STAL03] y [PATT98]. [MEE96a] proporciona un buen estudio de la tecnología de grabación de los discos y las cintas. [MEE96b] se centra en las técnicas de almacena-

miento de datos en los sistemas de disco y cinta. [WIED87] contiene un excelente estudio de los aspectos relacionados con el rendimiento de disco, incluyendo aquéllos vinculados con la planificación del disco. [NG98] examina los aspectos de rendimiento del hardware del disco. [CAO96] analiza el uso de la cache de disco y la planificación de disco. [WORT94] y [SELT90] son buenos estudios de algoritmos de planificación de disco, que incluyen análisis del rendimiento.

[ROSC03] proporciona un completo resumen de todos los tipos de sistemas de memoria externa, incluyendo una moderada cantidad de detalles técnicos de cada uno. Otro interesante estudio, con mayor énfasis en la interfaz de E/S y menos en los dispositivos en sí mismos, es [SCHW96]. [PAI00] es una descripción pedagógica sobre un esquema integrado de gestión de *buffers* y cache en el sistema operativo.

[DELL00] proporciona un estudio detallado de los manejadores de dispositivos de Windows junto con una profunda revisión de toda la arquitectura de E/S de Windows.

Un excelente estudio de la tecnología RAID, escrita por los inventores del concepto de RAID, es [CHEN94]. En [MASS97] se presenta un estudio más detallado de la *RAID Advisory Board*, una asociación de distribuidores y consumidores de productos relacionados con RAID. [CHEN96] analiza el rendimiento del esquema RAID. Otro interesante artículo es [FRIE96]. [DALT96] describe en detalle el sistema de RAID software de Windows NT.

CAO96 Cao, P.; Felten, E.; Karlin, A.; y Li, K. «Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling.» *ACM Transactions on Computer Systems*, Noviembre 1996.

CHEN94 Chen, P.; Lee, E.; Gibson, G.; Katz, R.; y Patterson, D. «RAID: High-Performance, Reliable Secondary Storage.» *ACM Computing Surveys*, Junio 1994.

CHEN96 Chen, S., y Towsley, D. «A Performance Evaluation of RAID Architectures.» *IEEE Transactions on Computers*, Octubre 1996.

DALT96 Dalton, W., *et al*. *Windows NT Server 4: Security, Troubleshooting, and Optimization*. Indianapolis, IN: New Riders Publishing, 1996.

DELL00 Dekker, E., y Newcomer, J. *Developing Windows NT Device Drivers: A Programmer's Handbook*. Reading, MA: Addison Wesley, 2000.

FRIE96 Friedman, M. «RAID Keeps Going and Going and ...» *IEEE Spectrum*, Abril 1996.

MASS97 Massiglia, P. (editor). *The RAID Book: A Storage System Technology Handbook*. St. Peter, MN: The Raid Advisory Board, 1997.

MEE96A Mee, C., y Daniel, E. eds. *Magnetic Recording Technology*. New York: McGraw-Hill, 1996.

MEE96B Mee, C., y Daniel, E. eds. *Magnetic Storage Handbook*. New York: McGraw-Hill, 1996.

NG98 Ng, S. «Advances in Disk Technology: Performance Issues». *Computer*, Mayo 1989.

PAI00 Pai, V.; Druschel, P.; y Zwaenepoel, W. «IO-Lite: A Unified I/O Buffering and Caching System». *ACM Transactions on Computer Systems*, Febrero 2000.

PATT98 Patterson, D., y Hennessy, J. *Computer Organization and Design: The Hardware/Software Interface*. San Mateo, CA: Morgan Kaufmann, 1998.

ROSC03 Rosch, W. *The Winn L. Rosch Hardware Bible*. Indianapolis, IN: Sams, 2003.

SCHW96 Schwaderer, W., y Wilson, A. *Understanding I/O Subsystems*. Milpitas, CA: Adaptec Press, 1996.

SELT90 Seltzer, M.; Chen, P.; y Ousterhout, J. «Disk Scheduling Revisited.» *Proceedings, USENIX Winter Technical Conference*, Enero 1990.

STAL03 Stallings, W. *Computer Organization and Architecture*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2000.

WIED87 Wiederhold, G. *File Organization for Database Design*. New York: McGraw-Hill, 1987.

WORT94 Worthington, B.; Ganger, G.; y Patt, Y. «Scheduling Algorithms for Modern Disk Drives.» *ACM SiGMETRICS*, Mayo 1994.

SITIOS WEB RECOMENDADOS

- **Caracterización y optimización de la E/S.** Un sitio dedicado a la educación e investigación en el área del diseño y del rendimiento de E/S. Incluye herramientas y cursos útiles. Gestionado por la universidad de Illinois.

11.13. TÉRMINOS CLAVE, CUESTIONES DE REPASO Y PROBLEMAS

TÉRMINOS CLAVE

acceso directo a memoria (DMA)	disco duro	paquete de discos
bloque	disco extraíble	pista
buffer circular	disco flexible	procesador de E/S
buffer de E/S	disco magnético	retardo rotacional
cabeza de lectura/escritura	disco no extraíble	sector
cache de disco	dispositivo orientado a bloques	tiempo de acceso al disco
canal de E/S	dispositivo orientado a flujo de caracteres	tiempo de búsqueda
CD-R	Entrada/salida (E/S)	tiempo de transferencia
CD-ROM	E/S de dispositivo	vector redundante de discos independientes (RAID)
CD-RW	E/S dirigida por interrupciones	
cilindro	E/S lógica	
disco de cabeza fija	E/S programada	
disco de cabeza móvil	hueco	
disco digital versátil (DVD)		

CUESTIONES DE REPASO

11.1. Cite ejemplos de recursos reutilizables y consumibles.

11.2. Enumere y defina brevemente tres técnicas para realizar E/S.

11.3. ¿Cuál es la diferencia entre la E/S lógica y la E/S de dispositivos?

11.4. ¿Cuál es la diferencia entre los dispositivos orientados a bloques y los orientados a flujos de caracteres? Proponga algunos ejemplos de cada tipo.

11.5. ¿Por qué se debería mejorar el rendimiento utilizando para E/S un *buffer* doble en lugar de un único *buffer*?

11.6. ¿Qué elementos de retardo están involucrados en una lectura o escritura de disco?

11.7. Defina brevemente las políticas de planificación de disco mostradas en la Figura 11.7.

11.8. Defina brevemente los siete niveles RAID.

11.9. ¿Cuál es el tamaño habitual del sector del disco?

PROBLEMAS

11.1. Considere un programa que accede a un único dispositivo de E/S y compare la E/S sin *buffer* con el uso de un *buffer*. Muestre que el uso del *buffer* puede reducir el tiempo de ejecución en un factor de dos como máximo.

11.2. Generalice el resultado del Problema 11.1 para el caso de un programa que utiliza n dispositivos.

11.3. Realice el mismo tipo de análisis que el que se muestra en la Tabla 11.2 para la siguiente secuencia de peticiones de pistas de disco: 27, 129, 110, 186, 147, 41, 10, 64 y 120. Asuma que la cabeza del disco está situada inicialmente en la pista 100 y se está moviendo en la dirección de números de pista crecientes. Repita el mismo análisis, pero ahora suponiendo que la cabeza del disco se está moviendo en la dirección contraria.

11.4. Considere un disco con N pistas numeradas de 0 a (N –1) y suponga que los sectores pedidos se distribuyen aleatoria y uniformemente sobre el disco. Se pretende calcular el número medio de pistas atravesadas en una búsqueda.

a) En primer lugar, calcule la probabilidad de que se produzca una búsqueda de longitud j cuando la cabeza está actualmente situada sobre la pista t. *Sugerencia*: hay que determinar el número total de combinaciones, teniendo en cuenta que todas las posiciones de pista tienen la misma probabilidad de ser el destino de la búsqueda.

b) A continuación, calcule la probabilidad de que se produzca una búsqueda de longitud K. *Sugerencia:* Este cálculo implica la suma de todas las posibles combinaciones de movimientos de K pistas.

c) Calcule el número medio de pistas atravesadas en una búsqueda, usando la fórmula que permite obtener el valor esperado:

$$\mathrm{E}[x] = \sum_{i=0}^{N-1} i \sum \Pr[x = i]$$

d) Demuestre que para valores grandes de N, el número medio de pistas atravesadas en una búsqueda es aproximadamente igual a $N/3$.

11.5. Se ha propuesto la siguiente ecuación tanto para la memoria cache como para la memoria cache de disco:

$$T_S = T_C + M \times T_D$$

Generalice esta ecuación para una jerarquía de memoria con N niveles en lugar de únicamente con 2.

11.6. Dado el algoritmo de remplazo basado en la frecuencia (*véase* la Figura 11.11), se define F_{nueva}, $F_{\text{intermedia}}$ y F_{antigua} como la fracción de la cache que corresponde con las secciones nueva, intermedia y antigua, respectivamente. Evidentemente, se cumplirá que $F_{\text{nueva}} + F_{\text{intermedia}} + F_{\text{antigua}} = 1$. Analice de qué tipo de política se trata si se cumple:

a) $F_{\text{antigua}} = 1 - F_{\text{nueva}}$

b) $F_{\text{antigua}} = 1 /$ (tamaño de la cache)

11.7. ¿Cuál es la velocidad de transferencia de una unidad de cinta magnética de nueve pistas si la velocidad de cinta es de 120 pulgadas por segundo y la densidad de la misma es de 1.600 bits lineales por pulgada?

11.8. Sea una bobina de cinta de 2.400 pies; y una zona de separación entre registros de 0,6 pulgadas, en donde la cinta se detiene entre dos lecturas; una aceleración/deceleración lineal durante los arranques/paradas en las zonas de separación entre registros; y las otras caracterísiticas de la cinta iguales a las del Problema 11.7. Los datos en la cinta están organizados en registros físicos, tal que cada registro físico contiene un número fijo de unidades definidas por el usuario, llamadas registros lógicos.

a) ¿Cuánto tiempo llevará leer una cinta completa de registros lógicos de 120 bytes agrupados en bloques de 10 por cada registro físico?

b) Lo mismo que en el Apartado (a), pero en este caso agrupados en bloques de 30.

c) ¿Cuántos registros lógicos almacenará la cinta usando cada uno de los dos factores de agrupamiento de bloques mencionados anteriormente?

d) ¿Cuál es la tasa de transferencia global efectiva para cada uno de los dos factores de agrupamiento de bloques mencionados anteriormente?

e) ¿Cuál es la capacidad de la cinta?

11.9. Calcule cuánto espacio de disco (en sectores, pistas y superficies) se requiere para almacenar los registros lógicos leídos en el Problema 11.8b si el disco tiene sectores de tamaño fijo de 512 bytes/sector, 96 sectores/pista, 110 pistas por superficie y 8 superficies útiles. Ignore cualquier tipo de registro (o registros) de cabecera de fichero o de índices de pista, y suponga que un registro no se puede extender sobre dos sectores.

11.10. Considere el sistema de disco descrito en el Problema 11.9 y suponga que el disco rota a 360 rpm. En este sistema, un procesador lee un sector del disco utilizando E/S dirigida por interrupciones, produciéndose una interrupción por cada byte. Si se tarda 2,5 μs en procesar cada interrupción, ¿cuál es el porcentaje de tiempo que el procesador dedica a manejar la E/S (sin tener en cuenta el tiempo de búsqueda)?

11.11. Repita el Problema 11.10 usando DMA y suponiendo una interrupción por sector.

11.12. Un computador de 32 bits tiene dos canales selectores y un canal multiplexor. Cada canal selector gestiona dos unidades de disco magnético y dos de cinta magnética. El canal multiplexor tiene conectados dos impresoras de líneas, dos lectores de tarjetas y diez terminales VDT. Suponga las siguientes velocidades de transferencia:

Dispositivo de disco	800 Kbytes/s
Dispositivo de cinta magnética	200 Kbytes/s
Impresora de líneas	6,6 Kbytes/s
Lector de tarjetas	1,2 Kbytes/s
VDT	1 Kbytes/s

Estime la máxima velocidad de transferencia de E/S conjunta en este sistema.

11.13. Debería ser evidente que la distribución de datos en bandas en los discos puede mejorar la tasa de transferencia de datos cuando el tamaño de la banda es pequeño comparado con el tamaño de la petición de E/S. Asimismo, debería ser evidente que el esquema RAID 0 proporciona una mejora en el rendimiento comparándolo con un único disco grande, ya que pueden manejarse en paralelo múltiples peticiones de E/S. Sin embargo, en este último caso, ¿es necesaria la distribución de datos en bandas? Es decir, ¿el uso de esta técnica mejora la tasa de peticiones de E/S respecto a la proporcionada por un vector de discos de características similares pero que no utiliza dicha técnica?

APÉNDICE 11A DISPOSITIVOS DE ALMACENAMIENTO EN DISCO

DISCO MAGNÉTICO

Un disco es un plato circular construido de metal o plástico que está cubierto con un material magnético. Los datos se graban en el disco y posteriormente se recuperan del mismo mediante una bobina conductora llamada **cabeza**. Durante una operación de lectura o escritura, la cabeza está estacionaria mientras que el plato rota debajo de ella.

El mecanismo de escritura se basa en el hecho de que cuando la electricidad fluye a través de una bobina se produce un campo magnético. Se envían pulsos a la cabeza, registrándose patrones magnéticos en la superficie subyacente, que serán diferentes dependiendo de si la corriente es positiva o negativa. El mecanismo de lectura se basa en que un campo magnético moviéndose con respecto a una bobina produce una corriente eléctrica en la bobina. Cuando la superficie del disco pasa debajo de la cabeza, se genera una corriente de la misma polaridad que la que se grabó previamente.

ORGANIZACIÓN Y FORMATO DE LOS DATOS

La cabeza es un dispositivo relativamente pequeño capaz de leer y escribir sobre una parte del plato que rota debajo de la misma. En consecuencia, la organización de datos en el plato consiste en un conjunto concéntrico de anillos, llamados **pistas**. Cada pista tiene la misma anchura que la cabeza. Hay miles de pistas en cada superficie.

La Figura 11.16 muestra esta distribución de datos. Las pistas adyacentes están separadas por **huecos**. Esto impide, o al menos minimiza, los errores debidos a que la cabeza no esté alineada o, simplemente, a que existan interferencias en los campos magnéticos.

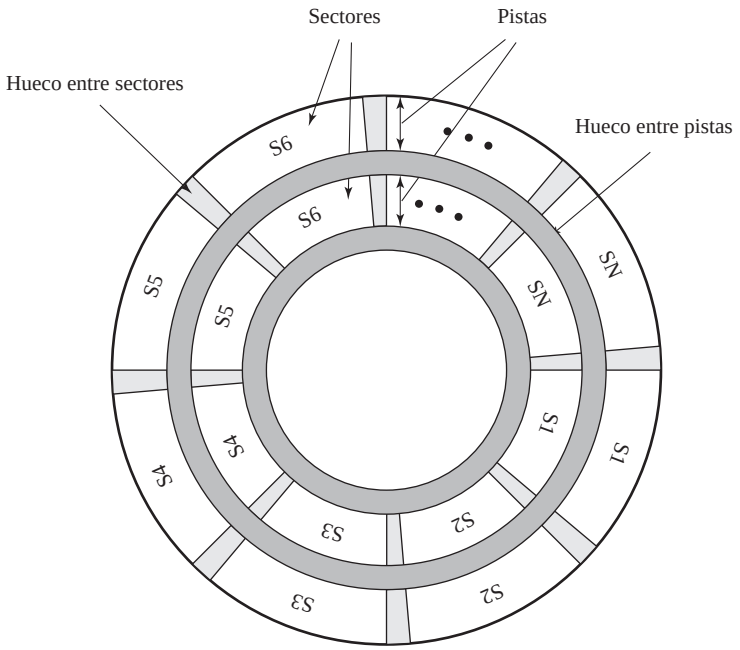
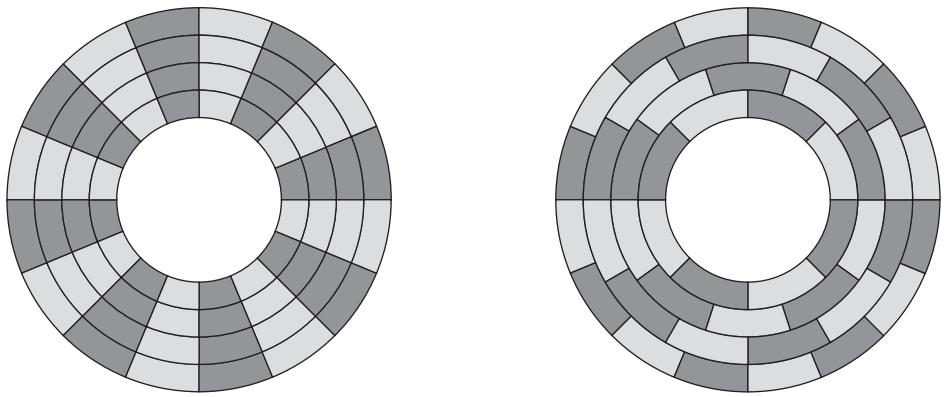


Figura 11.16. Disposición de los datos en el disco.

Los datos se transfieren del disco en **sectores** (Figura 11.16). Normalmente, hay cientos de sectores por pista, que pueden ser de longitud fija o variable. En la mayoría de los sistemas contemporáneos, los sectores utilizan una longitud fija de 512 bytes, siendo prácticamente universal el tamaño del sector. Para evitar imponer requisitos de precisión irrazonables en el sistema, los sectores adyacentes están separados por huecos entre las pistas, además de existir huecos entre los sectores de la misma pista.

Un bit que esté próximo al centro de un disco rotando pasa a través de un punto fijo (como una cabeza de lectura-escritura) a menos velocidad que un bit situado en la parte exterior. Por tanto, se debe encontrar alguna manera de compensar esta variación en la velocidad de manera que la cabeza pueda leer todos los bits a la misma velocidad. Esto se puede lograr incrementando el espacio entre los bits de información grabada en distintos segmentos del disco. Gracias a ello, la información puede accederse a la misma velocidad rotando el disco a una velocidad fija, conocida como **velocidad angular constante (*Constant Angular Velocity*, CAV)**. La Figura 11.17a muestra la disposición de un disco que utiliza **CAV**. El disco está dividido en varios sectores en forma de tarta y en una serie de pistas concéntricas. La ventaja de utilizar CAV es que se puede hacer referencia directamente a cada bloque individual de datos usando un número de pista y sector. Para mover la cabeza desde su posición actual a una dirección específica, sólo se necesita un corto movimiento de la cabeza a la pista correspondiente y una breve espera hasta que el sector apropiado gire debajo de la cabeza. La desventaja de CAV es que la cantidad de datos que se pueden almacenar en las pistas exteriores, que tienen mayor longitud, es la misma que se puede almacenar en las pistas interiores, que son más cortas.

Dado que la **densidad**, en bits por pulgada lineal, se incrementa al moverse desde las pistas más externas hasta las más internas, la capacidad de almacenamiento del disco en un sistema CAV sencillo está limitada por la densidad de grabación máxima que puede lograrse en las pistas más internas. Para incrementar la densidad, los sistemas de disco duro modernos utilizan una técnica conocida como **grabación en múltiples zonas**, en la que la superficie está dividida en varias zonas concéntricas (normalmente, 16). Dentro de una zona, el número de bits por pista es constante. Las zonas más alejadas del centro contienen más bits (más sectores) que las zonas que están más cerca del centro. Esto permite una mayor capacidad de almacenamiento global con el coste de una circuitería algo más compleja. Según se mueve la cabeza del disco de una zona a otra, cambia la longitud (a lo largo de la pista) de cada bit, causando un cambio en la temporización de las lecturas y escrituras. La Figura 11.17b muestra la naturaleza de la grabación en múltiples zonas; en esta figura, cada zona tiene una anchura de una sola pista.



(a) Velocidad angular constante (b) Grabación en múltiples zonas

Figura 11.17. Comparación de métodos de disposición del disco.

Se necesita alguna forma de localizar la posición de cada sector dentro de una pista. Lógicamente, debe de haber algún punto de inicio en la pista y una manera de identificar el inicio y el final de cada sector. Estos requisitos se manejan por medio de datos de control grabados en el disco. Por tanto, el disco se formatea con algunos datos adicionales utilizados sólo por el controlador de disco y que no están accesibles para el usuario.

CARACTERÍSTICAS FÍSICAS

La Tabla 11.6 detalla las características principales que distinguen los diversos tipos de discos magnéticos. En primer lugar, la cabeza puede estar fija o moverse con respecto a la dirección radial del plato. En un **disco con cabeza fija**, hay una cabeza de lectura/escritura por pista. Todas las cabezas están montadas en un brazo rígido que se extiende a lo largo de las pistas. En un **disco de cabeza móvil**, hay sólo una cabeza de lectura/escritura por cada superficie. También en este caso la cabeza está montada en un brazo. Dado que la cabeza debe de ser capaz de situarse sobre cualquier pista, el brazo puede extenderse o retraerse con este propósito.

Tabla 11.6. Características físicas de los sistemas de disco.

Movimiento de la cabeza	**Platos**
Cabeza fija (una por pista)	Único plato
Cabeza móvil (una por superficie)	Múltiples platos
Carácter portátil del disco	**Mecanismo de la cabeza**
Disco no extraíble	Contacto (disco flexible)
Disco extraíble	Separación fija
	Separación aerodinámica (Winchester)
Número de caras por plato	
Sólo una cara	
Doble cara	

El disco en sí mismo está montado en un dispositivo de disco, que consta de un brazo, un eje giratorio que rota el disco y la electrónica necesaria para leer y escribir datos binarios. Un **disco no extraíble** está montado permanentemente en el dispositivo de disco; el disco duro de un computador personal es un disco no extraíble. Un **disco extraíble** puede ser extraído y remplazado por otro disco. La ventaja de este tipo de discos es que permiten que esté disponible una cantidad ilimitada de datos con un número limitado de sistemas de disco. Además, un disco de este tipo se puede mover desde un computador a otro. Los discos flexibles y los cartuchos ZIP son ejemplos de discos extraíbles.

En la mayoría de los discos, la cubierta magnética está aplicada en ambos lados del plato, a lo que se denomina **doble cara**. Algunos discos más económicos utilizan discos de **una sola cara**.

Algunos dispositivos de disco incluyen **múltiples platos** apilados verticalmente separados por menos de una pulgada, existiendo múltiples brazos (Figura 11.18). Los discos con múltiples platos emplean una cabeza móvil, con una cabeza de lectura/escritura por cada superficie del plato. Todas las cabezas están fijas mecánicamente, de manera que todas están a la misma distancia del centro del disco y se mueven conjuntamente. Así, en todo momento, todas las cabezas se posicionan sobre pistas que están a la misma distancia del centro del disco. Al conjunto de todas las pistas en la misma posición relativa en el plato se le denomina **cilindro**. Por ejemplo, todas las pistas sombreadas en la Figura 11.19 son parte de un cilindro.

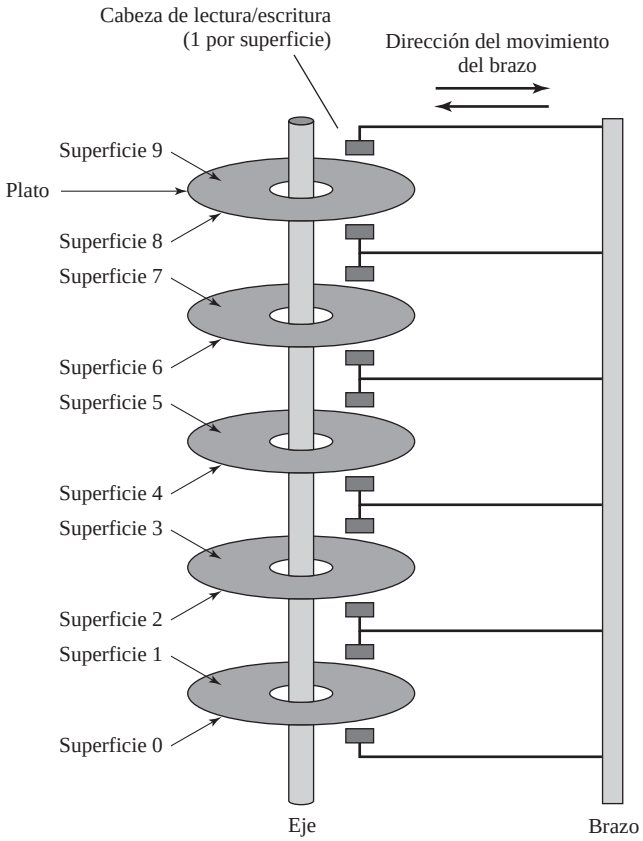


Figura 11.18. Componentes de un dispositivo de disco.

Por último, el mecanismo de la cabeza establece una clasificación de los discos en tres tipos. Tradicionalmente, la cabeza de lectura/escritura se posiciona a una distancia fija sobre el plato, dejando una bolsa de aire. En el otro extremo, existe un mecanismo de cabeza que entra realmente en contacto físico con el medio durante una operación de lectura/escritura. Este mecanismo se usa en el **disco flexible**, que es un plato pequeño y flexible, constituyendo el tipo de disco más económico.

Para comprender el tercer tipo de disco, se necesita comentar la relación existente entre la densidad de datos y el tamaño de la bolsa de aire. La cabeza debe generar o detectar un campo electromagnético de magnitud suficiente para leer y escribir apropiadamente. Cuanto más estrecha es la cabeza, más cerca debe estar de la superficie del plato para poder funcionar. Una cabeza más estrecha implica pistas más estrechas y, por tanto, mayor densidad de datos, lo que es deseable. Sin embargo, cuanto más cerca esté la cabeza del disco, mayor es el riesgo de error por impurezas e imperfecciones. El desarrollo del **Disco Winchester** significó un avance considerable en esta tecnología. Las cabezas Winchester se utilizan en montajes sellados de dispositivos que están prácticamente libres de contaminantes. Están diseñados para operar más cerca de la superficie del disco que las cabezas del disco rígido convencional, permitiendo una mayor densidad de datos. La cabeza es realmente una lámina aerodinámica que descansa ligeramente en la superficie del plato cuando el disco está inmóvil. La presión del aire generada por un disco cuando está girando es suficiente para hacer que la lámina se eleve por encima de la superficie. El sistema sin contactos resultante puede construirse de manera que se usen

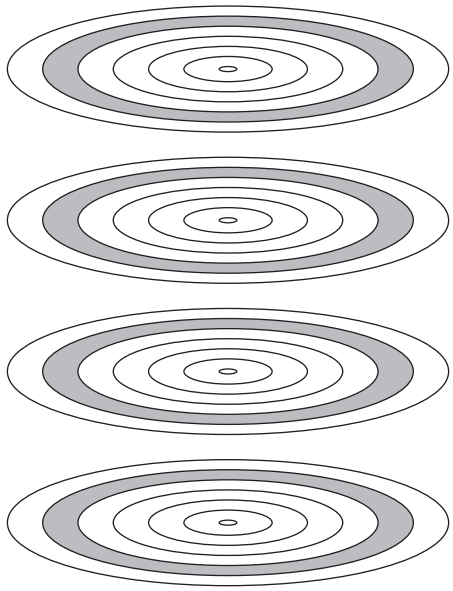


Figura 11.19. Pistas y cilindros.

cabezas más estrechas que operen más cerca de la superficie del plato que las cabezas de los disco duros convencionales[4].

La Tabla 11.7 muestra los parámetros de típicos discos de cabezas móviles de alto rendimiento actuales.

MEMORIA ÓPTICA

En 1983, se presentó uno de los productos de consumo de mayor éxito de todos los tiempos: el sistema de audio digital en disco compacto (*Compact Disk*, CD). El CD es un disco que no puede borrarse y que puede almacenar más de 60 minutos de información de audio en una cara. El enorme éxito comercial del CD permitió el desarrollo de la tecnología de almacenamiento de disco óptico de bajo coste que ha revolucionado el almacenamiento de datos del computador. Desde entonces, se han presentado diversos sistemas de disco óptico (*véase* la Tabla 11.8). A continuación, se presenta brevemente cada uno de ellos.

[4] Como asunto de interés historico, el término Winchester lo utilizó originalmente IBM como un nombre de código para el modelo de disco 3340 antes de ser anunciado públicamente. El 3340 era un paquete de discos extraíble con las cabezas selladas dentro del paquete. El término se aplica hoy en día a cualquier dispositivo de disco cuyas unidades están selladas y que usa un diseño de cabeza aerodinámico. El disco Winchester se usa comúnmente en computadores personales y estaciones de trabajo, donde se le llama **disco duro**.

Tabla 11.7. Parámetros de algunos discos duros típicos.

Características	Seagate Barracuda 180	Seagate Cheetah X15-36LP	Seagate Barracuda 36ES	Toshiba HDD1242	IBM Microdrive
Aplicación	Servidor de alta capacidad	Servidor de alto rendimiento	Computador personal de gama baja	Computador portátil	Dispositivos portátiles de mano
Capacidad	181,6 GB	36,7 GB	18,4 GB	5 GB	1 GB
Tiempo mínimo de búsqueda de pista a pista	0,8 ms	0,3 ms	1,0 ms	—	1,0 ms
Tiempo medio de búsqueda	7,4 ms	3,6 ms	9,5 ms	15 ms	12 ms
Velocidad de rotación	7.200 rpm	15K rpm	7.200 rpm	4.200 rpm	3.600 rpm
Retardo rotacional medio	4,17 ms	2 ms	4,17 ms	7,14 ms	8,33 ms
Tasa de transferencia máxima	160 MB/ s	522 a 709 MB/s	25 MB/s	66 MB/s	13,3 MB/s
Bytes por sector	512	512	512	512	512
Sectores por pista	793	485	600	63	—
Pistas por cilindro (número de caras de los platos)	24	8	2	2	2
Cilindros (número de pistas en una cara de un plato)	24.247	18.479	29.851	10.350	—

Tabla 11.8. Diversos tipos de discos ópticos.

CD	Disco compacto. Un disco que no puede borrarse y que almacena información de audio digitalizada. El sistema estándar usa discos de 12 cm. y puede grabar más de 60 minutos de tiempo de reproducción ininterrumpida.
CD-ROM	Disco compacto de memoria de sólo lectura (*Read Only Memory*). Un disco que no puede borrarse, utilizado para almacenar datos del computador. El sistema estándar usa discos de 12 cm. y puede almacenar más de 650 Mbytes.
CD-R	CD grabable (*Recordable*). Similar a un CD-ROM. El usuario sólo puede escribir una vez en el disco.
CD-RW	CD modificable (*Rewritable*). Similar a un CD-ROM. El usuario puede borrar y modificar el disco múltiples veces.
DVD	Disco Digital Versátil (*Digital Versatile Disk*). Una tecnología para producir una representación comprimida y digitalizada de información de vídeo, así como de grandes volúmenes de otros datos digitales. Se usan discos de 8 y de 12 cm. de diámetro, con una capacidad de hasta 17 Gbytes usando doble cara. El DVD básico es de sólo lectura (DVD-ROM).
DVD-R	DVD grabable (*Recordable*). Similar a un DVD-ROM. El usuario sólo puede escribir una vez en el disco. Sólo se pueden usar discos de una única cara.
DVD-RW	DVD modificable (*Rewritable*). Similar a un DVD-ROM. El usuario puede borrar y modificar el disco múltiples veces. Sólo se pueden usar discos de una única cara.

CD-ROM

Tanto el CD de audio como el CD-ROM (*Compact Disk Read Only Memory*, disco compacto de sólo lectura) usan una tecnología similar. La diferencia principal es que los reproductores de CD-ROM son más robustos y tienen mecanismos de corrección de errores que aseguran que los datos se transfieren correctamente desde el disco al computador. Ambos tipos de discos se fabrican de la misma manera. El disco está hecho de una resina, como, por ejemplo, policarbonato. La información grabada digitalmente (ya sea música o datos del computador) se imprime como una serie de agujeros microscópicos en la superficie del policarbonato. Esta operación se realiza, en primer lugar, con un láser de alta intensidad enfocado con precisión, creando un disco maestro. El maestro se usa, a su vez, para hacer una matriz que permita imprimir copias en policarbonato. A continuación, la superficie agujereada se cubre con una superficie muy reflectante, normalmente de aluminio o de oro. Esta superficie brillante está protegida del polvo y de los rasguños por una cubierta superior de material acrílico transparente. Por último, se puede realizar una serigrafía de una etiqueta sobre el material acrílico.

La información se lee de un CD o de un CD-ROM mediante un láser de baja intensidad alojado en el reproductor de disco óptico, o unidad del dispositivo. El láser ilumina a través de la cubierta protectora transparente mientras que un motor gira el disco que va pasando a través del mismo. La in-

tensidad de la luz del láser reflejada cambia cuando enfoca a un agujero. Este cambio lo detecta un foto sensor y lo convierte en una señal digital.

Recuerde que en un disco magnético la información se graba en pistas concéntricas. Si se usa en el disco magnético un sistema de velocidad angular constante (*Constant Angular Velocity*, CAV), que es más sencillo, el número de bits por pista es constante. Para lograr un incremento de la densidad, se usa un esquema de grabación en múltiples zonas, en el que la superficie está dividida en varias zonas, de manera que las más alejadas del centro contienen más bits que las más cercanas al mismo. Aunque esta técnica incrementa la capacidad, sigue sin ser óptima.

Para alcanzar mayor capacidad, el CD y el CD-ROM no organizan la información en pistas concéntricas. En su lugar, el disco contiene una única pista en espiral, comenzando cerca del centro y girando en forma de espiral hacia el extremo exterior del disco. Los sectores cercanos al exterior del disco tienen la misma longitud que los que están cerca del interior. Por tanto, la información está empaquetada uniformemente a lo largo del disco en segmentos del mismo tamaño, que se acceden a la misma velocidad rotando el disco a una velocidad variable. El láser lee los agujeros a una **velocidad lineal constante** (***Constant Linear Velocity*, CLV**). El disco rota más lentamente cuando se accede al extremo más exterior que cuando se accede cerca del centro. Por tanto, de igual manera la capacidad de una pista como el retardo rotacional se incrementan para las posiciones más cercanas al extremo más exterior del disco. La capacidad del CD-ROM es aproximadamente de 680 MB.

El CD-ROM es apropiado para la distribución de grandes cantidades de datos a un gran número de usuarios. Sin embargo, debido al gasto generado en el proceso inicial de escritura, no es apropiado para una sola aplicación. Comparándolo con los discos magnéticos tradicionales, el CD-ROM tiene tres ventajas principales:

- La capacidad de almacenamiento de información es mayor en el disco óptico.
- El disco óptico junto con la información almacenada en el mismo puede ser replicada masivamente de forma económica, a diferencia de lo que ocurre con un disco magnético. Para reproducir la base de datos almacenada en un disco magnético hay que copiarla disco a disco utilizando dos unidades de disco.
- El disco óptico es extraíble, permitiendo utilizar el disco en sí mismo como almacenamiento para el archivo permanente de información. La mayoría de los discos magnéticos no son extraíbles. En los discos magnéticos que no son extraíbles la información debe copiarse primero en cinta antes de que el dispositivo de disco, y el propio disco, pueda utilizarse para almacenar nueva información.

Las desventajas del CD-ROM son las siguientes:

- Es sólo de lectura y no puede actualizarse.
- Tiene un tiempo de acceso mucho mayor que el dispositivo de disco magnético, tanto como medio segundo.

CD GRABABLE

Para adaptarse a las aplicaciones en las que sólo se necesita una copia, o un pequeño número de copias, de un conjunto de datos, se ha desarrollado el CD que permite muchas lecturas pero sólo una escritura, que se denomina CD grabable (CD-R, CD *Recordable*). En un CD-R, se fabrica un disco de modo que posteriormente se puede escribir una vez con un rayo láser de moderada intensidad. Por

tanto, con un controlador de disco un poco más caro que el de CD-ROM, además de leer el disco, el consumidor puede escribir una vez en el mismo.

El substrato del CD-R es similar, aunque no idéntico, al CD o CD-ROM. En el caso del CD y el CD-ROM, la información se graba mediante agujeros en la superficie del medio, que cambia su capacidad reflectante. En un CD-R, el substrato incluye una capa con tinte. El tinte se utiliza para cambiar la capacidad reflectante, siendo activado por un láser de alta intensidad. El disco resultante se puede leer en un dispositivo de CD-R o en uno de CD-ROM.

El disco óptico CD-R está indicado para el archivo permanente de documentos y ficheros. Proporciona un registro permanente de grandes volúmenes de datos de usuario.

CD MODIFICABLE

El disco óptico CD-RW (*Rewritable*) se puede escribir y reescribir repetidas veces, como ocurre con un disco magnético. Aunque se han intentado usar diversas estrategias, la única técnica de carácter puramente óptico (en contraposición con la técnica magneto-óptica, que se presenta posteriormente) que se ha mostrado efectiva se denomina cambio de fase. El disco de cambio de fase utiliza un material que presenta dos niveles reflectantes significativamente diferentes en dos estados de fase distintos. Hay un estado amorfo, en el que las moléculas presentan una orientación aleatoria que apenas refleja la luz, y un estado cristalino que tiene una superficie lisa que refleja bien la luz. Un rayo de luz láser puede cambiar el material de una fase a otra. La desventaja primordial de los discos ópticos de cambio de fase es que el material acaba perdiendo de forma definitiva y permanente sus propiedades necesarias. Los materiales actuales se pueden utilizar durante un número de ciclos de borrado que van desde los 500.000 hasta 1.000.000.

El CD-RW tiene una ventaja evidente con respecto al CD-ROM y al CD-R: puede reescribirse y, por tanto, usarse como un verdadero almacenamiento secundario. Por ello, compite con el disco magnético. Una ventaja fundamental de los discos ópticos es que los márgenes de tolerancia en su fabricación son mucho menos estrictos que en los discos magnéticos de alta capacidad. Por tanto, presentan una mayor fiabilidad y una vida más prolongada.

DISCO DIGITAL VERSÁTIL

Con la gran capacidad del disco digital versátil (*Digital Versatile Disk*, DVD), la industria electrónica ha encontrado por fin un sustituto aceptable de la cinta analógica de vídeo VHS. El DVD remplazará a la cinta de vídeo utilizada en los grabadores de casetes de vídeo (*Video Cassette Recorder*, VCR) y, más importante en el ámbito de esta presentación, remplazará al CD-ROM en los computadores personales y servidores. EL DVD introduce al vídeo en la era digital. Muestra películas con una calidad de imagen impresionante, pudiendo accederse de manera aleatoria como en el CD audio, que también puede reproducirse en las máquinas DVD. En el disco DVD se pueden almacenar enormes volúmenes de datos: actualmente siete veces más que en un CD-ROM. Con la enorme capacidad de almacenamiento del DVD y su increíble calidad, los juegos de PC se harán más realistas y el software pedagógico incorporará un mayor uso del vídeo. Junto con el despertar de esta tecnología, se producirá una nueva explosión en el tráfico en Internet y en las intranets corporativas, según se vaya incorporando este material en los sitios web. La mayor capacidad del DVD se debe a las siguientes tres diferencias con respecto al CD-ROM:

- En el DVD los bits se almacenan más próximos entre sí. El espacio entre los bucles de una espiral en un CD es de 1,6 μm y la distancia mínima entre los agujeros a lo largo de la espiral es de 0,834 μm. El DVD utiliza un láser con una longitud de onda más corta alcanzando un espa-

cio de bucle de 0,74 μm y una distancia mínima entre agujeros de 0,4 μm. El resultado de estas dos mejoras es un incremento en la capacidad de aproximadamente siete veces la del CD-ROM, hasta cerca de los 4,7 GB.

- El DVD emplea una segunda capa de agujeros sobre un substrato encima de la primera capa. Un DVD de capa dual tiene una capa semi-reflectora encima de la capa reflectora, de manera que ajustando el foco, los lectores láser de los dispositivos DVD pueden leer cada capa separadamente. Esta técnica casi dobla la capacidad del disco, hasta casi 8,5 GB. La menor capacidad reflectora de la segunda capa limita su capacidad de almacenamiento por lo que no se llega al doble de capacidad.
- El DVD puede tener dos caras, mientras que en el CD los datos se graban en una sola cara, lo que proporciona una capacidad total de 17 GB.

Como ocurre con el CD, el DVD se presenta en versiones modificables, así como de sólo lectura (Tabla 11.8).

CAPÍTULO 12

Gestión de ficheros

En la mayoría de las aplicaciones, el fichero es el elemento central. Con la excepción de las aplicaciones de tiempo real y algunas aplicaciones especializadas, la entrada a la aplicación se realiza mediante un fichero y en prácticamente todas las aplicaciones, la salida se guarda en un fichero para un almacenamiento a largo plazo o para su acceso posterior por parte del usuario u otros programas.

Los ficheros tienen vida fuera de cualquier aplicación individual que los utilice como entrada y/o salida. Los usuarios desean poder acceder a los ficheros, guardarlos y mantener la integridad de sus contenidos. Con el fin de lograr estos objetivos, prácticamente todos los sistemas operativos proporcionan programas que se ejecutan como aplicaciones privilegiadas. Sin embargo, como mínimo, un sistema de gestión de ficheros necesita servicios especiales del sistema operativo; como máximo, el sistema de gestión de ficheros completo se considera parte del sistema operativo. Por tanto, es apropiado considerar los elementos básicos de la gestión de ficheros en este libro.

Se comenzará con un primer análisis, pasando a continuación a mostrar varios esquemas de organización de ficheros. Aunque la organización de ficheros está generalmente fuera del ámbito del sistema operativo, es esencial tener una comprensión general de las alternativas comunes para apreciar algunos de los compromisos de diseño relacionados con la gestión de ficheros. El resto del capítulo describe otros temas de la gestión de ficheros.

12.1. DESCRIPCIÓN BÁSICA

FICHEROS Y SISTEMAS DE FICHEROS

Desde el punto de vista del usuario, una de las partes más importantes de un sistema operativo es el sistema de ficheros. El sistema de ficheros proporciona las abstracciones de recursos típicamente asociadas con el almacenamiento secundario. El sistema de ficheros permite a los usuarios crear colecciones de datos, llamadas ficheros, con propiedades deseables, tales como las siguientes:

- **Existencia a largo plazo.** Los ficheros se almacenan en disco u otro almacenamiento secundario y no desaparece cuando un usuario se desconecta.
- **Compartible entre procesos.** Los ficheros tienen nombres y pueden tener permisos de acceso asociados que permitan controlar la compartición.
- **Estructura.** Dependiendo del sistema de ficheros, un fichero puede tener una estructura interna que es conveniente para aplicaciones particulares. Adicionalmente, los ficheros se pueden organizar en estructuras jerárquicas o más complejas para reflejar las relaciones entre los mismos.

Cualquier sistema de ficheros proporciona no sólo una manera de almacenar los datos organizados como ficheros, sino una colección de funciones que se pueden llevar a cabo sobre ficheros. Algunas operaciones típicas son las siguientes:

- **Crear.** Se define un nuevo fichero y se posiciona dentro de la estructura de ficheros.
- **Borrar.** Se elimina un fichero de la estructura de ficheros y se destruye.
- **Abrir.** Un fichero existente se declara «abierto» por un proceso, permitiendo al proceso realizar funciones sobre dicho fichero.
- **Cerrar.** Un determinado proceso cierra un fichero, de forma que no puede volver a realizar determinadas funciones sobre el mismo, a no ser que vuelva a abrirlo.

- **Leer.** Un proceso lee de un fichero todos los datos o una porción de ellos.
- **Escribir.** Un proceso actualiza un fichero, bien añadiendo nuevos datos que expanden el tamaño del fichero, bien cambiando los valores de elementos de datos existentes en el fichero.

Típicamente, un sistema de ficheros mantiene un conjunto de atributos asociados al fichero. Estos incluyen el propietario, tiempo de creación, tiempo de última modificación, privilegios de acceso, etc.

ESTRUCTURA DE UN FICHERO

Cuatro términos aparecen normalmente cuando se habla sobre ficheros:

- Campo.
- Registro.
- Fichero.
- Base de datos.

Un **campo** es el elemento básico de los datos. Un campo individual contiene un único valor, tal como el apellido de un empleado, una fecha o el valor de un sensor. Se caracteriza por su longitud y el tipo de datos (por ejemplo, ASCII, cadena de caracteres, decimal). Dependiendo del diseño del fichero, el campo puede tener una longitud fija o variable. En este último caso, el campo está formado normalmente por dos o tres subcampos: el valor real almacenado, el nombre del campo, y en algunos casos, la longitud del campo. En otros casos de campos de longitud variable, la longitud del campo se indica mediante el uso de símbolos de demarcación especiales entre campos.

Un **registro** es una colección de campos relacionados que pueden tratarse como una unidad por alguna aplicación. Por ejemplo, un registro de empleado podría contener campos tales como nombre, número de seguridad social, clasificación de trabajo, fecha de contratación, etc. De nuevo, dependiendo del diseño, los registros pueden ser de longitud fija o variable. Un registro tendrá una longitud variable si alguno de sus campos tiene longitud variable o si el número de campos puede variar. En este último caso, cada campo se acompaña normalmente de un nombre de campo. En cualquier caso, el registro completo incluye normalmente un campo longitud.

Un **fichero** es una colección de campos similares. El fichero se trata como una entidad única por parte de los usuarios y las aplicaciones. Los ficheros se pueden referenciar por nombre. Dichos ficheros se pueden crear y borrar. Las restricciones de control de acceso normalmente se aplican a nivel del fichero. Es decir, en un sistema compartido, el acceso a los ficheros completos es permitido o denegado a los usuarios y los programas. En algunos sistemas más sofisticados, tales controles se realizan a nivel de registro o incluso a nivel de campo.

Una **base de datos** es una colección de datos relacionados. Los aspectos esenciales de una base de datos son que la relación que exista entre los elementos de datos sea explícita y que la base de datos se diseña para su uso por parte de varias aplicaciones diferentes. Una base de datos podría contener toda la información relacionada con una organización o proyecto, tal como información de negocio o de estudio científico. La base de datos está formada por uno o más tipos de ficheros. Normalmente, hay un sistema de gestión de base de datos separado del sistema operativo, aunque hace uso de algunos programas de gestión de ficheros.

Los usuarios y las aplicaciones desean utilizar ficheros. Las operaciones típicas que deben soportarse incluyen:

- **Obtener_Todos.** Obtener todos los registros de un fichero. Esta operación se requerirá por aplicaciones que deban procesar toda la información de un fichero de una vez. Por ejemplo, una aplicación que produzca un resumen de la información existente en un fichero necesitaría obtener todos sus registros. Esta operación se asocia frecuentemente con el término *procesamiento secuencial*, porque todos los registros se acceden en secuencia.
- **Obtener_Uno.** Esta operación solicita un único registro. Las aplicaciones interactivas y orientadas a transacciones necesitan esta operación.
- **Obtener_Siguiente.** Esta operación solicita el «siguiente» registro, en alguna secuencia lógica con respecto al registro más recientemente leído. Algunas aplicaciones interactivas, tales como el rellenado de formularios, podrían requerir este tipo de operaciones. Un programa que realiza una búsqueda podría también utilizar este tipo de operación.
- **Obtener_Anterior.** Similar a la operación *Obtener_Siguiente*, pero en este caso el registro al que se accede es el anterior al más recientemente leído.
- **Insertar_Uno.** Insertar un nuevo registro en el fichero. Puede ser necesario que el nuevo registro encaje en una posición específica, para preservar una secuencia del fichero.
- **Borrar_Uno.** Borrar un registro existente. Ciertos enlaces u otras estructuras de datos podrían tener que actualizarse para preservar la secuencia del fichero.
- **Actualizar_Uno.** Obtener un registro, actualizar uno o más de sus campos, y reescribir el registro actualizado en el fichero. De nuevo, puede ser necesario preservar la secuenciación con esta operación. Si la longitud del registro ha cambiado, la operación de actualización es generalmente más difícil que si se preserva la longitud.
- **Obtener_Varios.** Obtener varios registros. Por ejemplo, una aplicación o usuario podría desear obtener todos los registros que satisfacen un cierto conjunto de criterios.

La naturaleza de las operaciones que se realizan más habitualmente sobre ficheros influirá en la forma en que se organiza un fichero, como se discutirá en la Sección 12.2.

Debería destacarse que no todos los sistemas de ficheros utilizan el conjunto de estructuras discutido en esta subsección. En UNIX y sistemas similares, la estructura de fichero básica es sólo un flujo de caracteres. Por ejemplo, un programa C se almacena como un fichero pero no tiene campos físicos, registros u otras estructuras.

SISTEMAS DE GESTIÓN DE FICHEROS

Un sistema de gestión de ficheros es aquel conjunto de software de sistema que proporciona servicios a los usuarios y aplicaciones en el uso de ficheros. Típicamente, la única forma en la que un usuario o aplicación puede acceder a los ficheros es a través del sistema de gestión de ficheros. Esto elimina la necesidad de que el usuario o programador desarrolle software de propósito especial para cada aplicación. Además, proporciona al sistema una forma consistente y bien definida de controlar su recurso más importante. [GROS86] sugiere los siguientes objetivos de un sistema de gestión de ficheros:

- Satisfacer las necesidades de gestión de datos y requisitos del usuario, lo que incluye el almacenamiento de datos y la capacidad de llevar a cabo las operaciones anteriormente mencionadas.
- Garantizar, hasta donde sea posible, que los datos del fichero son válidos.
- Optimizar el rendimiento, desde el punto de vista del sistema en términos de productividad y desde el punto de vista del usuario en términos de tiempo de respuesta.

- Proporcionar soporte de E/S a una variedad de tipos de dispositivos de almacenamiento.
- Minimizar o eliminar la potencial pérdida de datos.
- Proporcionar un conjunto estándar de rutinas de interfaces de E/S a los procesos.
- Proporcionar soporte de E/S a múltiples usuarios, en el caso de sistemas multiusuarios.

Respecto al primer punto, satisfacer los requisitos del usuario depende de una variedad de aplicaciones y del entorno en el cual el sistema de computación se utilizará. Para un sistema interactivo, de propósito general, las siguientes características constituyen un conjunto mínimo de requisitos:

1. Cada usuario debería poder crear, borrar, leer, escribir y modificar ficheros.
2. Cada usuario tendría acceso controlado a los ficheros de otros usuarios.
3. Cada usuario podría controlar qué tipos de accesos se permiten a los ficheros de los usuarios.
4. Cada usuario debe poder reestructurar los ficheros de los usuarios en una forma apropiada al problema.
5. Cada usuario debe ser capaz de mover datos entre ficheros.
6. Cada usuario debe poder copiar y recuperar los ficheros de usuario en caso de daño.
7. Cada usuario debe poder acceder a los ficheros utilizando nombres simbólicos.

Estos objetivos y requisitos deben tenerse en mente cuando se discutan los sistemas de gestión de ficheros.

Arquitectura de un sistema de ficheros

Una forma de conocer el ámbito de la gestión de ficheros es analizar la organización de software típica, como sugiere la Figura 12.1. Por supuesto, distintos sistemas se organizarán de forma diferente, pero esta organización es razonablemente representativa. En el nivel más bajo, los **manejadores de dispositivos** se comunican directamente con los dispositivos periféricos o sus controladores o canales. Un controlador de dispositivo es el responsable de iniciar las operaciones de E/S de un dispositivo y procesar la finalización de una petición de E/S. Para las operaciones sobre ficheros, los dispositivos típicos son los discos y las cintas. Los controladores de dispositivos se consideran normalmente parte del sistema operativo.

El siguiente nivel se denomina **sistema de ficheros básico**, o nivel de **E/S físico**. Esta es la interfaz primaria con el entorno fuera del sistema de computación. Trata con bloques de datos que son intercambiados con discos o sistemas de cintas. Por tanto, este nivel se encarga de la colocación de aquellos bloques del dispositivo de almacenamiento secundario y el *buffering* de dichos bloques en memoria principal. No se encarga de interpretar el contenido de los datos o la estructura de los ficheros. El sistema de ficheros básico es frecuentemente considerado parte del sistema operativo.

El **supervisor de E/S básico** se encarga de todas las iniciaciones y finalizaciones de E/S. En este nivel, las estructuras de control tratan con los dispositivos de E/S, la planificación y el estado de los ficheros. El supervisor de E/S básico selecciona el dispositivo en el cual se van a llevar a cabo las operaciones, basándose en el fichero particular seleccionado. También se encarga de la planificación de disco y cinta para optimizar el rendimiento. A este nivel, se asignan los *buffers* de E/S y la memoria secundaria. El supervisor de E/S básico es parte del sistema operativo.

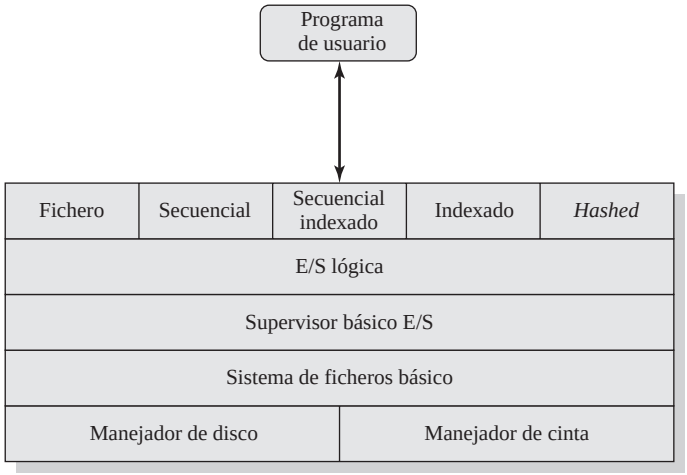


Figura 12.1. Arquitectura software de un sistema de ficheros.

La **E/S lógica** permite a los usuarios y a las aplicaciones acceder a los registros. Por tanto, mientras que el sistema de ficheros básico trata con bloques de datos, el módulo de E/S lógica trata con registros de ficheros. La capa de E/S lógica proporciona una capacidad de E/S de propósito general, a nivel de registros y mantiene datos básicos sobre los ficheros.

El nivel del sistema de ficheros más cercano al usuario es frecuentemente denominado **método de acceso**. Proporciona una interfaz estándar entre las aplicaciones y los sistemas de ficheros y dispositivos que contienen los datos. Diferentes métodos de acceso reflejan diferentes estructuras de ficheros y diferentes formas de acceder y procesar los datos. Algunos de los métodos de acceso más comunes se muestran en la Figura 12.1 y se describen brevemente en la Sección 12.2.

Funciones de gestión de ficheros

Otra forma de ver las funciones de un sistema de ficheros se muestra en la Figura 12.2. Sigamos este diagrama de izquierda a derecha. Los usuarios y programas de aplicaciones interaccionan con el sistema de ficheros por medio de mandatos para crear y borrar ficheros y realizar operaciones sobre los ficheros. Antes de realizar cualquier operación, el sistema de ficheros debe identificar y localizar el fichero seleccionado. Esto requiere el uso de algún tipo de directorio que se utilice para describir la ubicación de todos los ficheros, más sus atributos. Adicionalmente, la mayoría de los sistemas compartidos fuerza el control de acceso de usuario: sólo se permite determinado acceso particular a los usuarios autorizados. Las operaciones básicas que puede realizar un usuario o aplicación se realizan a nivel de registro. El usuario o aplicación ve el fichero como una estructura que organiza los registros, tales como una estructura secuencial (por ejemplo, los registros personales se almacenan alfabéticamente por el apellido). Por tanto, para traducir los mandatos de usuario en mandatos de manipulación de ficheros específicos, debe emplearse el método de acceso apropiado para esta estructura de ficheros.

Mientras que los usuarios y las aplicaciones se preocupan por los registros, la E/S se realiza a nivel de bloque. Por tanto, los registros de un fichero se deben convertir en bloque en la salida y volver a convertir en estructura de registro después de la entrada. Para dar soporte de bloque de E/S, se necesitan varias funciones. Se debe gestionar el almacenamiento secundario. Esto supone asignar ficheros a bloques libres de almacenamiento secundario así como conocer qué bloques están disponibles para nuevos ficheros y utilizados en ficheros existentes. Adicionalmente, se deben planificar las peticiones

de E/S de bloques individuales; este tema se trató en el Capítulo 11. Tanto la planificación de disco como la asignación de ficheros están relacionadas con la optimización del rendimiento. Como se podría esperar, estas funciones, por tanto, necesitan considerarse simultáneamente. Más aún, la optimización dependerá de la estructura de los ficheros y los patrones de acceso. De acuerdo a esto, desarrollar un sistema de gestión de ficheros óptimo, desde el punto de vista del rendimiento, es una tarea excesivamente complicada.

La Figura 12.2 sugiere una división entre las responsabilidades del sistema de gestión de ficheros considerado como una utilidad del sistema y las responsabilidades del sistema operativo, siendo el punto de intersección el procesamiento de registros. Esta división es arbitraria: se utilizan distintas técnicas en diversos sistemas.

El resto de este capítulo describe algunos de los aspectos de diseño sugeridos en la Figura 12.2. Se comienza con una discusión de las organizaciones de los ficheros y los métodos de acceso. Aunque este tema está fuera del ámbito de lo que se considera responsabilidad del sistema operativo, es imposible discutir aspectos de diseño sin una apreciación de la organización y el acceso a los ficheros. Por tanto, se describe el concepto de directorio. El sistema operativo, en nombre del sistema de gestión de ficheros, gestiona frecuentemente los directorios. Los temas restantes tratan sobre los aspectos físicos de E/S de gestión de ficheros y son tratados apropiadamente como aspectos de diseño de sistemas operativos. Uno de estos temas es la forma en la cual los registros lógicos se organizan en bloques físicos. Finalmente, hay temas relacionados con la asignación de ficheros en almacenamiento secundario y la gestión de almacenamiento secundario libre.

12.2. ORGANIZACIÓN Y ACCESO A LOS FICHEROS

En esta sección, se utiliza el término *organización de fichero* para referirse a la estructura lógica de los registros determinados por la forma en la que se acceden. La organización física del fichero en al-

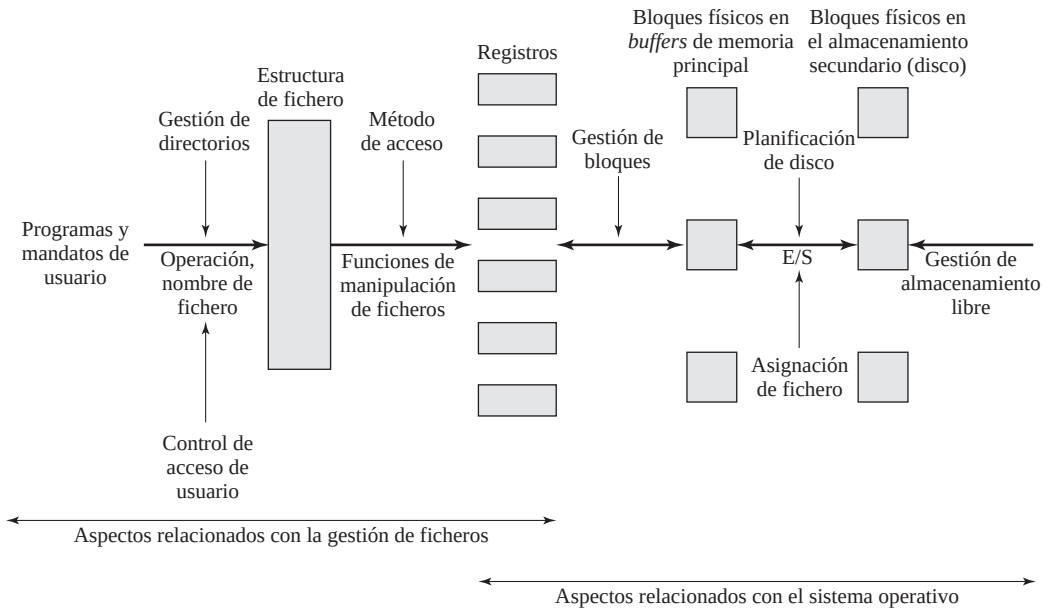


Figura 12.2. Ciclo de instrucción básico.

macenamiento secundario depende de la estrategia de bloques y de asignación de ficheros, temas tratados posteriormente en este capítulo.

Para escoger una organización de ficheros, son importantes varios criterios:

- Tiempo de acceso corto.
- Facilidad de actualización.
- Economía de almacenamiento.
- Mantenimiento sencillo.
- Fiabilidad.

La prioridad relativa de estos criterios dependerá de las aplicaciones que utilizarán el fichero. Por ejemplo, si un fichero se va a procesar sólo en lotes, con acceso a todos los registros cada vez, entonces el acceso rápido a un único registro no es un requisito. Un fichero almacenado en CD-ROM nunca se actualizará, y por tanto la facilidad de actualización no es un aspecto a tener en cuenta en este caso.

Estos criterios pueden entrar en conflicto. Por ejemplo, para facilitar la economía de almacenamiento, debería haber mínima redundancia en los datos. Por otro lado, la redundancia es una medida primaria para incrementar la velocidad de acceso a los datos. Un ejemplo lo constituye el uso de índices.

El número de organizaciones de ficheros alternativas que se han implementado o simplemente propuesto es inmanejablemente largo, incluso para un libro dedicado a los sistemas de ficheros. En este breve resumen, se describen cinco organizaciones fundamentales. La mayoría de las estructuras utilizadas en los sistemas reales cae dentro de una de estas categorías y se puede implementar con una combinación de estas organizaciones. Las cinco organizaciones, de las cuales las cuatro primeras se muestran en la Figura 12.3, son:

- La pila.
- El fichero secuencial.
- El fichero secuencial indexado.
- El fichero indexado.
- El fichero de acceso directo o *hash*.

La Tabla 12.1 resume los aspectos de rendimiento relativo de estas cinco organizaciones[1].

LA PILA

La forma menos complicada de organización de ficheros se puede denominar *pila*. Los datos se almacenan en el orden en el que llegan. Cada registro está formado por un conjunto de datos. El propósito de la pila es simplemente acumular la masa de datos y guardarlos. Los registros podrían tener diferentes campos o similares campos en diferentes órdenes. Por tanto, cada campo debe ser autodescrip-

[1] La tabla emplea la notación «O mayúscula», utilizada para caracterizar la complejidad de tiempo de los algoritmos. Una explicación de esta notación se encuentra en un documento en el sitio web de este libro.

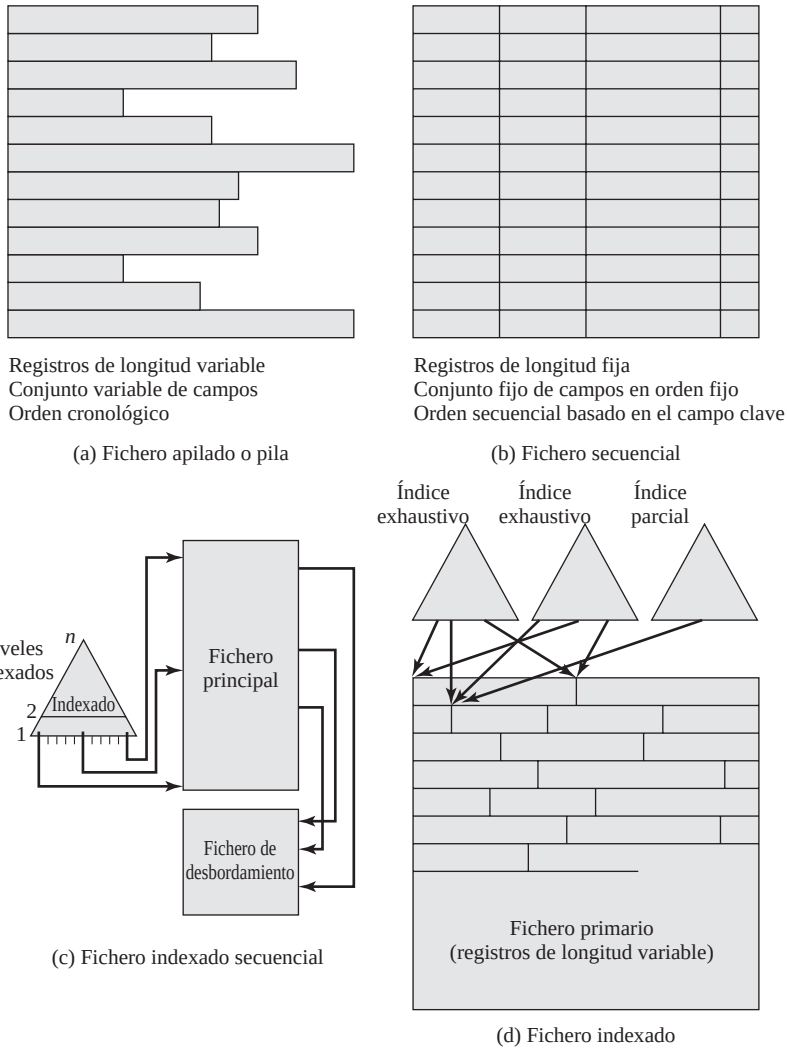


Figura 12.3. Organizaciones comunes de ficheros.

tivo, incluyendo el nombre del campo y el valor. La longitud de cada campo debe ser implícitamente indicada por delimitadores, explícitamente incluida como un subcampo o conocida por defecto para cada tipo de campo.

Dado que no hay estructura en este tipo de fichero, el acceso a los registros se hace mediante búsqueda exhaustiva. Es decir, si se desea encontrar un registro que contiene un campo particular con un valor particular, es necesario examinar cada registro en la pila hasta encontrar el registro deseado o terminar de recorrer todo el fichero. Si se desea encontrar todos los registros que contienen un campo particular o contienen dicho campo con un valor particular, entonces se debe recorrer el fichero entero.

Los ficheros pila se utilizan cuando los datos se recogen y almacenan antes del procesamiento o cuando los datos no son fáciles de organizar. Este tipo de ficheros utiliza el espacio adecuadamente cuando los datos varían en tamaño y estructura, es perfectamente adecuado para búsquedas exhausti-

Tabla 12.1. Grados de rendimiento de las cinco organizaciones de ficheros básicas [WIED87].

Método de fichero	Espacio		Actualización		Adquisición		
	Atributos		Tamaño de registro		Único registro	Sub-conjunto	Exhaustivo
	Variable	Fijo	Igual	Mayor			
Pila	A	B	A	E	E	D	B
Secuencial	F	A	D	F	F	D	A
Indexado secuencial	F	B	B	D	B	D	B
Indexado	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A = Excelente, muy adecuado para este propósito ≈ O(r)
B = Bueno ≈ O(o x r)
C = Adecuado ≈ O(r log n)
D = Requiere algún esfuerzo extra ≈ O(n)
E = Posible con esfuerzo extremo ≈ O(r x n)
F = No razonable para este propósito ≈ $O(n^{>1})$

donde

r = tamaño de los resultados
o = número de registros que desbordan
n = número de registros del fichero

vas y es fácil de actualizar. Sin embargo, más allá de estos usos limitados, este tipo de fichero es inadecuado para la mayoría de las aplicaciones.

EL FICHERO SECUENCIAL

La forma más común de estructura de fichero es el fichero secuencial. En este tipo de ficheros, se utiliza un formato fijo para los registros. Todos los registros son de igual tamaño y están compuestos por el mismo número de campos de longitud fija en un orden específico. Debido a que la longitud y la posición de cada campo son conocidas, sólo se necesita almacenar los valores de los campos; el nombre y longitud de cada campo son atributos de la estructura del fichero.

Un campo particular, normalmente el primer campo de cada registro, se denomina **campo clave**. El campo clave identifica de forma única el registro; por tanto, los valores de la clave de diferentes registros son siempre diferentes. Más aún, los registros se almacenan en secuencia según la clave: orden alfabético para una clave de texto y orden numérico para una clave numérica.

Los ficheros secuenciales se utilizan normalmente en aplicaciones en lotes y son generalmente óptimos para dichas aplicaciones cuando implican el procesamiento de todos los registros (por ejemplo, una aplicación bancaria o de nóminas). La organización de fichero secuencial es el único que se almacena fácilmente en cinta y en disco.

Para aplicaciones interactivas que suponen consultas y/o actualizaciones de registros individuales, el fichero secuencial proporciona un pobre rendimiento. Los accesos requieren una búsqueda en

el fichero para encontrar una clave. Si el fichero entero, o una porción grande del fichero, se pueden llevar a memoria principal simultáneamente, se pueden utilizar técnicas de búsqueda más eficientes. No obstante, acceder a un registro en un fichero secuencial grande implica un procesamiento y retrasos considerables. Las adiciones a los ficheros también presentan problemas. Típicamente, un fichero secuencial se almacena en orden secuencial simple de los registros dentro de los bloques. Es decir, la organización física del fichero en cinta o discos encaja directamente con la organización lógica del fichero. En este caso, el procedimiento normal es colocar nuevos registros en un fichero de pila separado, denominado fichero registro o fichero de transacciones. Periódicamente, un sistema de actualización lleva a cabo la mezcla entre el registro y el fichero maestro para producir un nuevo fichero en la secuencia de claves correcta.

Una alternativa es organizar el fichero secuencial físicamente como una lista enlazada. Uno o más registros se almacenan en cada bloque físico. Cada bloque del disco contiene un puntero al siguiente bloque. La inserción de nuevos registros implica una manipulación de punteros, pero no requiere que los nuevos registros ocupen una posición de bloque físico específica. Por tanto, se obtienen ventajas a cambio de incrementar el procesamiento y la sobrecarga.

EL FICHERO SECUENCIAL INDEXADO

Una técnica popular para eliminar las desventajas del fichero secuencial es utilizar los ficheros secuenciales indexados. El fichero secuencial indexado mantiene las características clave del fichero secuencial: los registros se organizan en secuencia, basándose en un campo clave. Dos características se añaden: un índice al fichero que da soporte al acceso aleatorio y un fichero de desbordamiento. El índice proporciona una capacidad de búsqueda para alcanzar rápidamente la vecindad de un registro deseado. El fichero de desbordamiento es similar al fichero registro, utilizado con un fichero secuencial, pero se integra de tal forma que un registro en el fichero de desbordamiento se localiza mediante un puntero desde su registro predecesor.

En la estructura secuencial indexada más sencilla, se utiliza un único nivel de indexación. El índice en este caso es un fichero secuencial simple. Cada registro del fichero índice está formado por dos campos: un campo clave, que es el mismo que el campo clave del fichero principal y un puntero al fichero principal. Para encontrar un campo específico, se busca el índice que contenga el mayor valor clave que sea igual o preceda al valor de clave deseado. La búsqueda continúa en el fichero principal en la ubicación indicada por el puntero.

Para comprobar la efectividad de esta técnica, considere un fichero secuencial con 1 millón de registros. Buscar un valor clave particular requerirá medio millón de accesos a los registros de media. Supóngase que se construye un índice que contiene 1000 entradas, con las claves del índice más o menos distribuidas uniformemente en el fichero principal. Encontrar un registro llevará una media de 500 accesos al fichero índice seguido por 500 accesos al fichero principal. La longitud de búsqueda media se reduce de 500.000 a 1000.

Las adiciones al fichero se gestionan de la siguiente forma: cada registro del fichero principal contiene un campo adicional no visible a la aplicación, que es un puntero al fichero de desbordamiento. Cuando se inserta un nuevo registro en el fichero, se añade al fichero de desbordamiento. Se actualiza el registro del fichero principal que inmediatamente precede al nuevo registro en secuencia lógica para contener un puntero al nuevo registro del fichero de desbordamiento. Si el registro inmediatamente precedente está a su vez en el fichero de desbordamiento, entonces se actualiza el puntero de dicho registro. Al igual que el fichero secuencial, el fichero secuencial indexado es ocasionalmente mezclado con el fichero de desbordamiento en modo *batch*.

El fichero secuencial indexado reduce enormemente el tiempo requerido para acceder a un único registro, sin sacrificar la naturaleza secuencial del fichero. Para procesar el fichero entero secuencial-

mente, los registros del fichero principal se procesan en secuencia hasta que se encuentra un puntero al fichero de desbordamiento; a continuación, se accede de forma continua en el fichero de desbordamiento hasta que se encuentra un puntero nulo, momento en el cual se continúa accediendo al fichero principal desde el lugar en que se dejó.

Para proporcionar incluso mayor eficiencia en el acceso, se pueden utilizar múltiples niveles de indexación. A continuación, el menor nivel de indexación se trata como un fichero secuencial y se crea un fichero índice de mayor nivel para dicho fichero. Se construye un índice de bajo nivel con 10.000 entradas. Entonces se puede construir un índice de mayor nivel por cada 100 entradas. La búsqueda comienza en el índice de mayor nivel (longitud media = 50 accesos) para encontrar un punto de entrada en el índice de menor nivel. Por tanto, este índice se procesa (longitud media = 50) para encontrar un punto de entrada en el fichero principal, en el cual se busca de nuevo. Por tanto, la longitud media de búsqueda se ha reducido de 500.000 a 1000, y de 1000 a 150.

EL FICHERO INDEXADO

El fichero secuencial indexado elimina una de las limitaciones del fichero secuencial: el procesamiento efectivo se limita a las búsquedas que se basan en un único campo del fichero. Cuando es necesario buscar por algún otro atributo que no sea el campo clave, ambas formas de ficheros secuenciales son inadecuadas. En algunas aplicaciones, esta flexibilidad es deseable.

Para lograr esta flexibilidad, se necesita una estructura que emplea múltiples índices, uno por cada tipo de campo que puede estar sujeto a una búsqueda. En el fichero indexado general, se abandonan los conceptos de secuencialidad y clave única. Los registros se acceden sólo a través de sus índices. El resultado es que no hay restricción en la colocación de los registros siempre que al menos un puntero en un índice se refiera a dicho registro. Además, se pueden emplear registros de longitud variable.

Se utilizan dos tipos de índice. Un índice exhaustivo contiene una entrada por cada registro del fichero principal. Para facilitar la búsqueda, el índice a su vez está organizado como un fichero secuencial. Un índice parcial contiene entradas a registros donde el campo de interés existe. Con registros de longitud variable, algunos registros no contendrán todos los campos. Cuando se añade un nuevo registro al fichero principal, todos los ficheros índices deben actualizarse.

Los ficheros índices se utilizan frecuentemente en aplicaciones donde la temporización de la información es crítica y donde los datos casi nunca se procesan exhaustivamente. Ejemplos de este tipo de aplicación son los sistemas de reservas de aerolíneas y los sistemas de control de inventario.

EL FICHERO DE ACCESO DIRECTO O *HASH*

El fichero de acceso directo, o *hash*, explota la capacidad encontrada en los discos para acceder directamente a cualquier bloque de una dirección conocida. Al igual que los ficheros secuenciales y secuenciales indexados, se requiere una clave para cada registro. Sin embargo, en este tipo de ficheros no existe el concepto de ordenación secuencial.

El fichero directo hace uso de una función *hash* sobre un valor clave. Esta función se explicó en el Apéndice 8A. La Figura 8.27b muestra el tipo de organización *hash* con un fichero de desbordamiento que se utiliza normalmente en un fichero *hash*.

Los ficheros directos se utilizan frecuentemente cuando se requiere un acceso muy rápido, los registros son de tamaño fijo y los registros se acceden de uno en uno. Ejemplos de este tipo de estructura son los directorios, las tablas de precios, los inventarios y las listas de nombres.

12.3. DIRECTORIOS

CONTENIDO

Asociado con cualquier sistema de gestión de ficheros y colección de ficheros, se encuentra el concepto de directorio. El directorio contiene información sobre los ficheros, incluyendo atributos, ubicación y propiedad. Gran parte de esta información, especialmente la que concierne a almacenamiento, la gestiona el sistema operativo. El directorio es a su vez un fichero, accesible por varias rutinas de gestión de ficheros. Aunque parte de la información de los directorios está disponible para los usuarios y las aplicaciones, esto se proporciona generalmente de forma indirecta por las rutinas del sistema.

La Tabla 12.2 muestra la información normalmente utilizada en el directorio por cada fichero del sistema. Desde el punto de vista del usuario, el directorio proporciona una proyección entre los nombres de ficheros, conocidos para los usuarios y las aplicaciones, y los ficheros en sí. Por tanto, cada entrada del fichero incluye el nombre del fichero. Prácticamente todos los sistemas tratan con diferentes tipos de ficheros y distintas organizaciones de ficheros, y esta información también se proporciona. Una importante categoría de información sobre cada fichero trata sobre el almacenamiento, incluyendo su ubicación y tamaño. En sistemas compartidos, es también importante proporcionar información que se utilice para controlar el acceso a los ficheros. Típicamente, un usuario es el propietario del fichero y puede conceder ciertos privilegios de acceso a otros usuarios. Finalmente, se utiliza información de uso para gestionar la utilización actual del fichero y registrar la historia de su uso.

ESTRUCTURA

La forma en la que la información de la Tabla 12.2 se almacena difiere ampliamente entre varios sistemas. Parte de la información se puede almacenar en un registro cabecera asociado con el fichero; esto reduce la cantidad de almacenamiento requerido para el directorio, haciendo más fácil almacenar el directorio o parte del directorio en memoria principal, a fin de incrementar la velocidad.

La forma más sencilla de estructura para un directorio es una lista de entradas, una por cada fichero. Esta estructura se podría representar como un fichero secuencial simple, con el nombre del fichero actuando como clave. En algunos sistemas iniciales monousuario, se ha utilizado esta técnica. Sin embargo, esta técnica es inadecuada cuando múltiples usuarios comparten el sistema o cuando un único usuario tiene muchos ficheros.

Para comprender los requisitos de la estructura de un fichero, es útil considerar los tipos de operaciones que se pueden llevar a cabo sobre directorios:

- **Buscar.** Cuando un usuario o aplicación referencia un fichero, el directorio debe permitir encontrar la entrada correspondiente a dicho fichero.
- **Crear fichero.** Cuando se crea un nuevo fichero, se debe añadir una entrada al directorio.
- **Borrar fichero.** Cuando se borra un fichero, se debe eliminar una entrada del directorio.
- **Listar directorio.** Se puede solicitar ver el directorio completo o una porción del mismo. Generalmente, el usuario solicita esta petición y como resultado obtiene un listado de todos los ficheros de los cuales es propietario, más algunos de los atributos de cada fichero (por ejemplo, información de control de acceso, información de uso).
- **Actualizar directorio.** Debido a que algunos atributos se almacenan en el directorio, un cambio en uno de estos atributos requiere un cambio en la entrada de directorio correspondiente.

Tabla 12.2. Elementos de información de un directorio.

	Información básica
Nombre de fichero	Nombre escogido por el creador (usuario o programa). Debe ser único dentro de un directorio específico.
Tipo de fichero	Por ejemplo: texto, binario, módulo de carga, etc.
Organización de fichero	Para sistemas que soportan diferentes organizaciones.
	Información de direccionamiento
Volumen	Indica el dispositivo en el cual se almacena el fichero.
Dirección inicial	Dirección física inicial en almacenamiento secundario (por ejemplo, cilindro, pista y número de bloque en disco).
Tamaño utilizado	Tamaño actual del fichero en bytes, palabras o bloques.
Tamaño asignado	Tamaño máximo del fichero.
	Información de control de acceso
Propietario	Usuario que tiene el control del fichero. El propietario puede conceder/denegar acceso a otros usuarios y cambiar estos privilegios.
Información de acceso	Una versión sencilla de este elemento incluye el nombre de usuario y clave para cada usuario autorizado.
Acciones permitidas	Controla la lectura, escritura, ejecución y la transmisión a través de la red.
	Información de uso
Fecha de creación	Fecha en la que el fichero se coloca por vez primera en el directorio.
Identidad del creador	Normalmente aunque no necesariamente el propietario actual.
Fecha de último acceso de lectura	Fecha de la última vez que se leyó un registro.
Identidad de último lector	Usuario que hizo la última lectura.
Fecha de último acceso de modificación	Fecha de la última actualización, inserción o borrado.
Identidad de último modificador	Usuario que hizo la última modificación.
Fecha de la última copia de seguridad	Fecha de la última vez que el fichero fue copiado en otro medio de almacenamiento.
Uso actual	Información sobre la actividad actual sobre el fichero, tal como proceso o procesos que tienen el fichero abierto, si está bloqueado por un proceso y si el fichero ha sido actualizado en memoria principal pero no en disco.

La lista sencilla no es adecuada para dar soporte a estas operaciones. Considérese las necesidades de un único usuario. Los usuarios podrían tener muchos tipos de ficheros, incluyendo ficheros de procesamiento de texto, gráficos, hojas de cálculo, etc. Al usuario le podría gustar tener estos ficheros organizados por proyecto, por tipo, o de alguna forma conveniente. Si el directorio es una lista secuencial simple, no proporciona ayuda para organizar los ficheros y fuerza a los usuarios a tener cuidado de no utilizar el mismo nombre para dos tipos de ficheros diferentes. El problema se agrava en un sistema compartido. El nombrado único se convierte en un problema serio. Más aún, es difícil conceder porciones del directorio completo a los usuarios cuando no hay una estructura inherente en el directorio.

Una primera solución para resolver estos problemas sería pasar a un esquema de dos niveles. En este caso, hay un directorio por cada usuario y un directorio maestro. El directorio maestro tiene una entrada por cada directorio usuario, proporcionando información sobre dirección y control de acceso. Cada directorio de usuario es una lista simple de los ficheros de dicho usuario. Esto implica que los nombres deben ser únicos sólo dentro de la colección de los ficheros de un único usuario y que el sistema de ficheros puede fácilmente asegurar las restricciones de acceso de los directorios. Sin embargo, aún no proporciona ayuda a los usuarios para estructurar su colección de ficheros.

Una técnica más potente y flexible, que es casi universalmente adoptada, es utilizar una estructura jerárquica en forma de árbol (Figura 12.4). Como en la técnica anterior, hay un directorio maestro, que tiene bajo dicho directorio varios directorios de usuario. Cada uno de estos directorios de usuario, a su vez, podría tener subdirectorios y ficheros como entradas. Esto se cumple para todos los niveles: es decir, en cada nivel, un directorio podría estar formado por subdirectorios y/o ficheros.

Falta describir cómo se organiza cada directorio y subdirectorio. El enfoque más sencillo, por supuesto, es almacenar cada directorio como un fichero secuencial. Si los directorios contuvieran un gran número de entradas, dicha organización llevaría a unos tiempos de búsqueda innecesariamente largos. En dicho caso, es preferible una estructura *hash*.

NOMBRADO

Los usuarios necesitan poder referenciar un fichero mediante un nombre simbólico. Claramente, cada fichero en el sistema debe tener un nombre único a fin de que las referencias al mismo no sean ambiguas. Por otro lado, es inaceptable obligar a que los usuarios proporcionen nombres únicos, especialmente en un sistema compartido.

El uso de un directorio estructurado en forma de árbol minimiza la dificultad de asignar nombres únicos. Cualquier fichero del sistema se puede localizar siguiendo un camino desde el directorio raíz o maestro y bajando por las ramas hasta alcanzar el fichero. El conjunto de nombres de directorios, finalizando en el nombre del fichero, constituye un **nombre de camino** para el fichero. Por ejemplo,

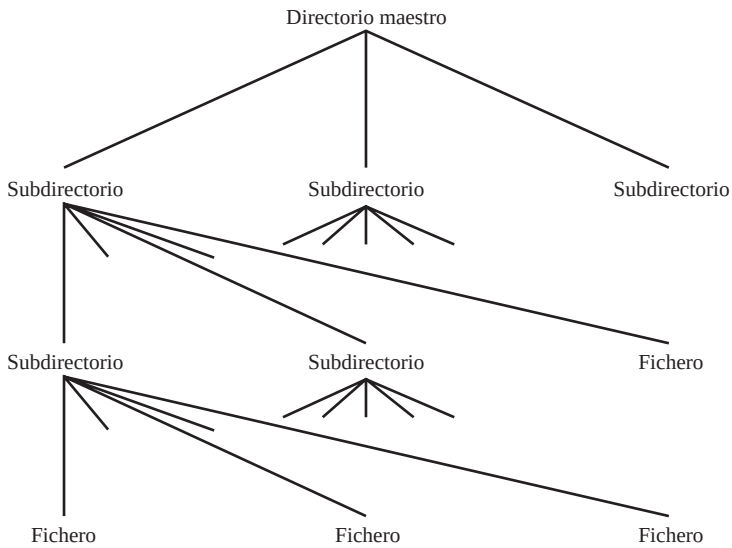


Figura 12.4. Directorio estructurado en forma de árbol.

el fichero de la esquina inferior izquierda de la Figura 12.5 tiene el camino /Usuario_B/Texto/Unidad_A/ABC. La barra se utiliza para delimitar nombres en la secuencia. El nombre del directorio maestro es implícito, porque todos los nombres comienzan en dicho directorio. Obsérvese que es perfectamente aceptable tener varios ficheros con el mismo nombre, siempre que ambos ficheros tengan nombres de camino únicos, lo que equivale a decir que el mismo nombre de fichero se puede utilizar en diferentes directorios. En el ejemplo, hay otro fichero en el sistema con el nombre ABC, pero cuyo nombre completo es /Usuario_B/Dibujo/ABC.

Aunque los nombres de camino facilitan la selección de los nombres de fichero, sería complicado para el usuario tener que escribir el camino completo cada vez que se hace una referencia a un fichero. Normalmente, un usuario interactivo o un proceso está asociado con un directorio actual, que se suele denominar **directorio de trabajo**. Los ficheros se pueden referenciar de forma relativa al direc-

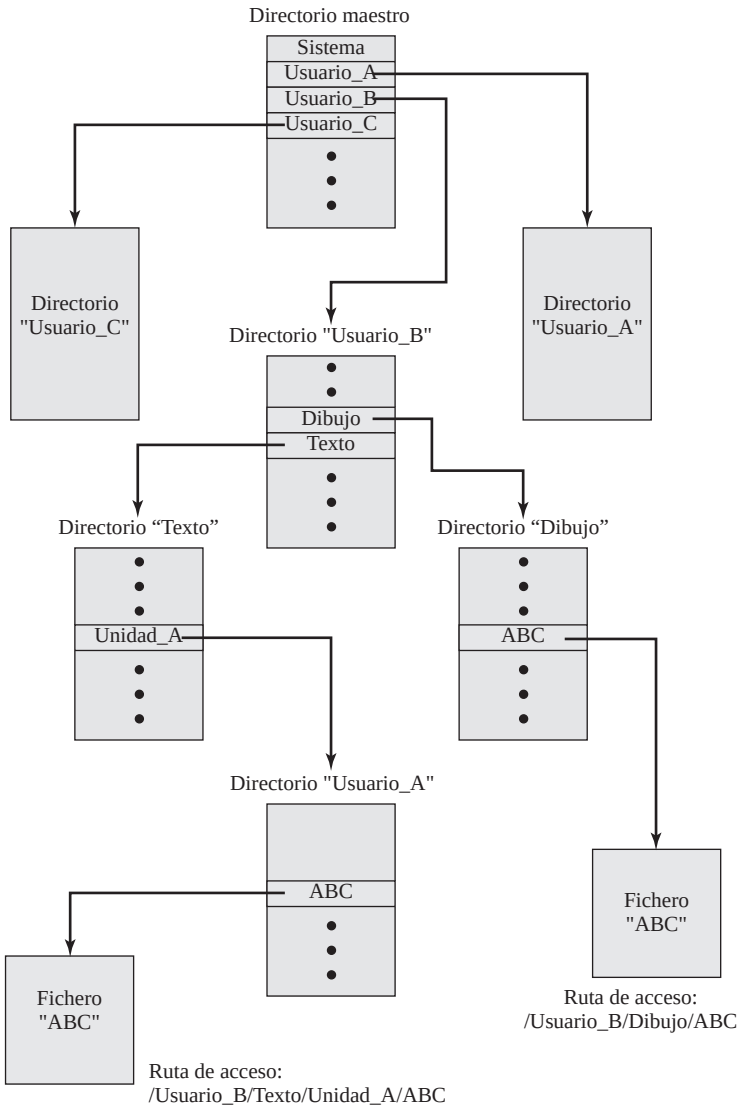


Figura 12.5. Ejemplo de directorio estructurado en forma de árbol.

torio de trabajo. Por ejemplo, si el directorio de trabajo del usuario B es «Texto», entonces el nombre Unidad_A/ABC es suficiente para identificar al fichero de la esquina inferior izquierda de la Figura 12.5. Cuando un usuario interactivo se conecta o cuando se crea un proceso, su directorio de trabajo por omisión es el directorio inicial del usuario. Durante la ejecución, el usuario puede navegar hacia arriba y hacia abajo en el árbol para cambiarse a un directorio de trabajo diferente.

12.4. COMPARTICIÓN DE FICHEROS

En un sistema multiusuario, existe casi siempre el requisito de permitir que los ficheros se compartan entre varios usuarios. Existen dos aspectos relacionados: derechos de acceso y la gestión de acceso simultáneo.

DERECHOS DE ACCESO

El sistema de ficheros debería proporcionar una herramienta flexible para permitir la compartición de ficheros extensiva entre los usuarios. El sistema de ficheros debería proporcionar varias opciones de tal forma que el acceso a un fichero particular se pueda controlar. Típicamente, a los usuarios o grupos de usuarios se les concede ciertos derechos de acceso a un fichero. Se ha utilizado un amplio rango de derechos de acceso. La siguiente lista es representativa de los derechos de acceso que se asignan a un usuario particular para un determinado fichero:

- **Ninguno.** El usuario no puede incluso conocer la existencia del fichero, y por tanto, tampoco puede acceder a él. Para forzar esta restricción, el usuario no tiene permiso de lectura del directorio en el cual se incluye este fichero.
- **Conocimiento.** El usuario puede determinar si el fichero existe y quién es su propietario. El usuario entonces es capaz de solicitar al propietario derechos de acceso adicionales.
- **Ejecución.** El usuario puede cargar y ejecutar un programa pero no copiarlo. Los programas propietarios utilizan normalmente estas restricciones.
- **Lectura.** El usuario puede leer el fichero para cualquier propósito, incluyendo copia y ejecución. Algunos sistemas son capaces de forzar una distinción entre ver y copiar. En el primer caso, el usuario puede ver el contenido del fichero, pero no puede realizar una copia.
- **Adición.** El usuario puede añadir datos al fichero, frecuentemente sólo al final, pero no puede modificar o borrar cualquiera de los contenidos del fichero. Este derecho es útil para recolectar datos de varias fuentes.
- **Actualización.** El usuario puede modificar, borrar o añadir datos al fichero. Esto normalmente incluye escribir el fichero al inicio, reescribirlo completa o parcialmente y borrar todos o una porción de los datos. Algunos sistemas diferencian entre distintos grados de actualización.
- **Cambio de protección.** El usuario puede cambiar los derechos de acceso otorgados a otros usuarios. Normalmente, sólo el propietario del fichero goza de este derecho. En algunos sistemas, el propietario puede extender este derecho a otros. Para prevenir del abuso de este mecanismo, el propietario del fichero normalmente puede especificar qué derechos podría cambiar un usuario con este tipo de permiso.
- **Borrado.** El usuario puede borrar el fichero del sistema de ficheros.

Estos derechos se pueden considerar como una jerarquía, con cada uno de los derechos conteniendo a aquellos que le preceden. Por tanto, si un usuario particular tiene el permiso de actualización

para un fichero concreto, dicho usuario también tendrá los siguientes derechos: conocimiento, ejecución, lectura y adición.

A un usuario se le considera propietario de un determinado fichero, normalmente a la persona que inicialmente creó un fichero. El propietario tiene todos los derechos de acceso listados previamente y puede conceder permisos a otros usuarios. Se pueden proporcionar diferentes accesos a distintas clases de usuarios:

- **Usuario específico.** Usuarios individuales que se designan por el identificador del usuario.
- **Grupos de usuarios.** Un conjunto de usuarios que no se definen individualmente. El sistema debe tener alguna forma de gestionar la membresía de los grupos de usuarios.
- **Todos.** Todos los usuarios que tienen acceso a este sistema. Dichos ficheros se consideran ficheros públicos.

ACCESO SIMULTÁNEO

Cuando se garantiza acceso de adición o actualización de un fichero a más de un usuario, el sistema operativo o sistema de gestión de ficheros debe forzar una disciplina. Una técnica de fuerza bruta consiste en permitir al usuario bloquear el fichero completo cuando se va a actualizar. Un control de grano más fino implica el bloqueo de registros individuales durante la actualización. Esencialmente, este es el problema de los lectores/escritores discutido en el Capítulo 5. Se deben tratar aspectos de exclusión mutua e interbloqueos a la hora de diseñar capacidades de acceso compartidas.

12.5. BLOQUES Y REGISTROS

Como se indica en la Figura 12.2, los registros son las unidades lógicas de acceso de un fichero estructurado, mientras que los bloques son las unidades de E/S con almacenamiento secundario. Para que la E/S se pueda realizar, los registros se deben organizar como bloques[2].

Hay varios aspectos a considerar. Primero, ¿los bloques deberían ser de longitud fija o variable? En la mayoría de los sistemas, los bloques tienen longitud fija. Esto simplifica la E/S, la asignación de *buffers* en memoria principal y la organización de bloques en almacenamiento secundario. A continuación, ¿cuál debería ser el tamaño relativo de un bloque comparado con el tamaño de registro medio? El compromiso es el siguiente: cuanto mayor sea el bloque, más registros se transferirán en una operación de E/S. Si se procesa un fichero de forma secuencial, esto supone una ventaja, porque se reduce el número de operaciones de E/S utilizando bloques mayores, y por tanto, acelerando el procesamiento. Por otro lado, si los registros se acceden de forma aleatoria y no se observa ninguna proximidad de referencias, entonces utilizar bloques más grandes supone transferencias innecesarias de registros no utilizados. Sin embargo, si se combina la frecuencia de operaciones secuenciales con la potencialidad de proximidad de referencias, se puede decir que el tiempo de transferencia de E/S se reduce utilizando bloques mayores. La preocupación viene por el hecho de que bloques más grandes requieren *buffers* de E/S mayores, haciendo la gestión de *buffers* más difícil.

Dado el tamaño de un bloque, se pueden utilizar tres métodos:

---

[2] En contraposición a un fichero que se trata sólo como una ristra de bytes, como en el sistema de ficheros UNIX.

- **Bloques fijos.** Se utilizan registros de longitud fija y se almacenan en un bloque un número integral de registros. Podría haber espacio no utilizado al final de cada bloque. Esto se denomina fragmentación interna.
- **Bloques expandidos de longitud variable.** Se utilizan registros de longitud variable y se empaquetan en bloques sin dejar espacio no utilizado. Por tanto, algunos registros deben expandirse a lo largo de dos bloques, con su continuación indicada por un puntero al bloque sucesor.
- **Bloques no expandidos de longitud variable.** Se utilizan registros de longitud variable, pero no se emplea expansión. Hay espacio malgastado en la mayoría de los bloques debido a la incapacidad para utilizar el resto de un bloque si el siguiente registro es mayor que el espacio no utilizado restante.

La Figura 12.6 ilustra estos métodos asumiendo que los ficheros se almacenan en bloques secuenciales del disco. El efecto no cambiaría si se utilizara algún otro esquema de asignación de ficheros (*véase* Sección 12.6).

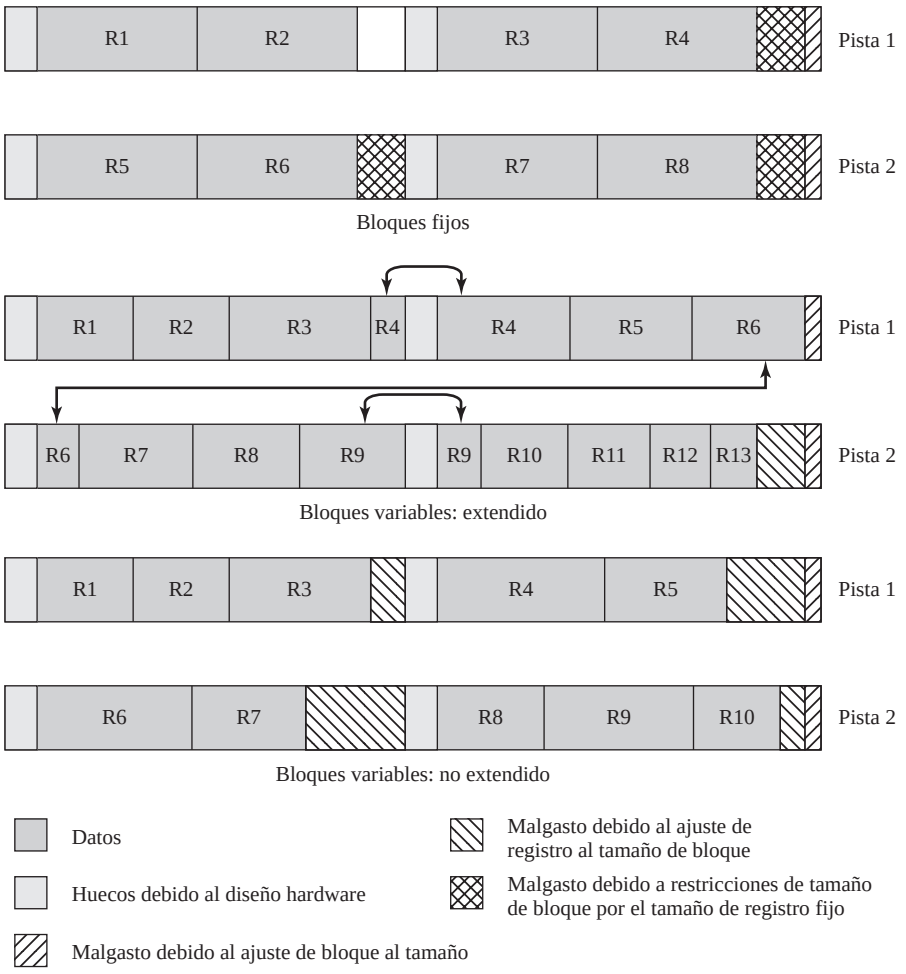


Figura 12.6. Métodos de asignación de registros a bloques [WIED87].

Utilizar bloques fijos es el modo común para ficheros secuenciales con registros de longitud fija. Los bloques expandidos de longitud variable son eficientes respecto al almacenamiento y no limitan el tamaño de los registros. Sin embargo, esta técnica es difícil de implementar. Los registros que expanden dos bloques requieren dos operaciones de E/S y los ficheros son difíciles de actualizar, sin tener en cuenta la organización. Los bloques no expandidos de longitud variable implican espacio malgastado y limitan el tamaño del registro al tamaño de un bloque.

La técnica de bloques y registros utilizada podría interaccionar con el hardware de memoria virtual, si ésta se utilizara. En un entorno de memoria virtual, sería deseable utilizar la página como unidad básica de transferencia. Las páginas son generalmente bastante pequeñas, de forma que es impracticable tratar una página como un bloque para bloques no expandidos. Análogamente, algunos sistemas combinan múltiples páginas para crear un bloque más grande con propósitos de E/S. Esta técnica se utiliza para ficheros VSAM en mainframes IBM.

12.6. GESTIÓN DE ALMACENAMIENTO SECUNDARIO

En almacenamiento secundario, un fichero está compuesto por una colección de bloques. El sistema operativo o sistema de gestión de ficheros es responsable de asignar bloques a los ficheros. Esto supone dos aspectos relacionados con la gestión. Primero, se debe asignar espacio de almacenamiento secundario a los ficheros y segundo, es necesario guardar una traza del espacio disponible para su asignación. Veremos que estas dos tareas están relacionadas; es decir, la técnica seleccionada para asignación de ficheros podría influir en la técnica seleccionada para gestión del espacio libre. Más aún, se verá que existe una interacción entre la estructura de los ficheros y las políticas de asignación.

Se comenzará esta sección analizando alternativas para asignación de ficheros en un único disco. A continuación se describen aspectos relacionados con la gestión de espacio libre y finalmente se discutirá la fiabilidad.

ASIGNACIÓN DE FICHEROS

Varios aspectos están involucrados en la asignación de ficheros:

1. Cuando se crea un fichero nuevo, ¿se asigna de una vez el espacio máximo requerido para el fichero?
2. El espacio se asigna a un fichero como una o más unidades contiguas, lo que se denomina porción. El tamaño de una porción puede ir desde un único bloque al fichero completo. ¿Qué tamaño de porción debería utilizarse para asignación de ficheros?
3. ¿Qué clase de estructura de datos o tabla se utiliza para guardar traza de las porciones asignadas para un fichero? Un ejemplo de dicha estructura es una **tabla de asignación de ficheros** (*File Allocation Table*, FAT), encontrado en DOS y otros sistemas.

Se examinarán estos aspectos a continuación.

Preasignación frente a asignación dinámica

Una política de preasignación requiere que el tamaño máximo de un fichero sea declarado en tiempo de creación de fichero. En varios casos, tales como las compilaciones de programas, la producción de ficheros de datos resumen, o la transferencia de un fichero desde otro sistema sobre la red de comuni-

cación, se puede estimar este valor de forma fiable. Sin embargo, para muchas aplicaciones, es difícil sino imposible, estimar fiablemente el tamaño máximo potencial del fichero. En dichos casos, los usuarios y los programadores de aplicaciones tenderían a sobrestimar el tamaño de fichero de forma que no se queden sin espacio. Esto claramente supone malgasto desde el punto de vista de la asignación de espacio de almacenamiento. Por tanto, hay ventajas en el uso de la gestión dinámica, que asigna espacio a un fichero en porciones cuando se necesite.

Tamaño de porción

El segundo aspecto listado es el tamaño de la porción asignado a un fichero. En un extremo, se puede asignar una porción suficientemente grande para contener el fichero completo. En el otro extremo, para el espacio en el disco se puede asignar un bloque cada vez. Para escoger un tamaño de porción, debe existir un compromiso entre la eficiencia desde el punto de vista de un único fichero y la eficiencia del sistema completo. [WIED87] lista cuatro aspectos a considerar en este compromiso:

1. La contigüidad del espacio incrementa el rendimiento, especialmente para operaciones *Obtener_Siguiente* y para transacciones ejecutándose en un sistema operativo orientado a transacciones.
2. Utilizar un gran número de porciones pequeñas incrementa el tamaño de las tablas necesarias para gestionar la información de asignación.
3. Utilizar porciones de tamaño fijo (por ejemplo, bloques) simplifica la reasignación de espacio.
4. Utilizar porciones de tamaño variable o pequeñas de tamaño fijo minimiza el espacio malgastado debido a la sobreasignación.

Por supuesto, estos elementos interaccionan y se deben considerar conjuntamente. Por tanto, existen dos alternativas principales:

- **Porciones variables, grandes y contiguas.** Esta alternativa proporciona el mejor rendimiento. El tamaño variable evita malgastar espacio, y las tablas de asignación de ficheros son pequeñas. Sin embargo, el espacio es difícil de reutilizar.
- **Bloques.** Pequeñas porciones fijas proporcionan mayor flexibilidad. Podrían requerir grandes tablas o estructuras complejas para su asignación. La contigüidad ha sido abandonada como meta primaria; los bloques se asignan según se necesite.

Cada opción es compatible con preasignación o asignación dinámica. En el caso de porciones variables, grandes y contiguas, al fichero se le preasigna un grupo de bloques contiguos. Esto elimina la necesidad de una tabla de asignación de ficheros; todo lo que se requiere es un puntero al primer bloque y el número de bloques asignados. En el caso de los bloques, todas las porciones requeridas se asignan a la vez. Esto significa que la tabla de asignación de ficheros para el fichero es de tamaño fijo.

Con porciones de tamaño variable, es necesario preocuparse de la fragmentación del espacio libre. Este aspecto se trató cuando se consideró la memoria principal particionada en el Capítulo 7. Las siguientes estrategias son posibles:

- **Primer ajuste.** Escoger el primer grupo contiguo no utilizado de bloques de tamaño suficiente desde una lista de bloques libres.
- **Siguiente ajuste.** Escoger el grupo más pequeño no utilizado que sea de suficiente tamaño.

- **Ajuste más próximo.** Escoger el grupo no utilizado de tamaño suficiente que sea más cercano a la asignación previa para el fichero de manera que se incremente la proximidad.

No está claro qué estrategia es mejor. La dificultad de modelar estrategias alternativas es el hecho de que muchos factores interaccionen, incluyendo tipos de ficheros, patrones de acceso a ficheros, grado de multiprogramación, otros factores de rendimiento del sistema, *caching* de disco, planificación de disco y otros.

Métodos de asignación de ficheros

Habiendo analizado los aspectos de preasignación frente a la asignación dinámica y el tamaño de las porciones, estamos en posición de considerar métodos específicos de asignación de ficheros. Tres métodos son de uso común: contiguo, encadenado e indexado. La Tabla 12.3 resume algunas de las características de cada método.

Tabla 12.3. Métodos de asignación de ficheros.

	Contiguos	**Encadenado**	**Indexado**	
¿Preasignación?	Necesaria	Posible	Posible	
¿Porciones de tamaño fijo o variable?	Variable	Bloques fijos	Bloques fijos	Variable
Tamaño de porción	Grande	Pequeño	Pequeño	Medio
Frecuencia de asignación	Una vez	Pequeña a alta	Alta	Baja
Tiempo a asignar	Medio	Largo	Corto	Medio
Tamaño de tabla de asignación de ficheros	Una entrada	Una entrada	Grande	Medio

Con **asignación contigua**, se asigna un único conjunto contiguo de bloques en tiempo de creación de los ficheros (Figura 12.7). Por tanto, hay una estrategia de preasignación que utiliza porciones de tamaño variable. La tabla de asignación de ficheros necesita sólo una entrada para cada fichero, mostrando el bloque inicial y la longitud del fichero. La asignación contigua es la mejor desde el punto de vista del fichero secuencial individual. Múltiples bloques se pueden leer de una vez para mejorar el rendimiento de E/S en procesamiento secuencial. Es también fácil obtener un único bloque. Por ejemplo, si un fichero comienza en el bloque b y se quiere acceder al bloque i-ésimo del fichero, su ubicación en almacenamiento secundario es simplemente $b + i - 1$. La asignación contigua presenta algunos problemas. Existirá fragmentación externa, haciendo difícil encontrar bloques contiguos de espacio de suficiente longitud. De vez en cuando, será necesario llevar a cabo un algoritmo de compactación para liberar espacio adicional en el disco (Figura 12.8). Además, con preasignación, es necesario declarar el tamaño del fichero en el tiempo de creación, con los problemas mencionados anteriormente.

En el extremo opuesto de la asignación contigua está la **asignación encadenada** (Figura 12.9). Típicamente, la asignación se realiza a nivel de bloques individuales. Cada bloque contiene un puntero al siguiente bloque en la cadena. De nuevo, la tabla de asignación de ficheros necesita sólo una entrada para cada fichero, mostrando el bloque inicial y la longitud del fichero. Aunque

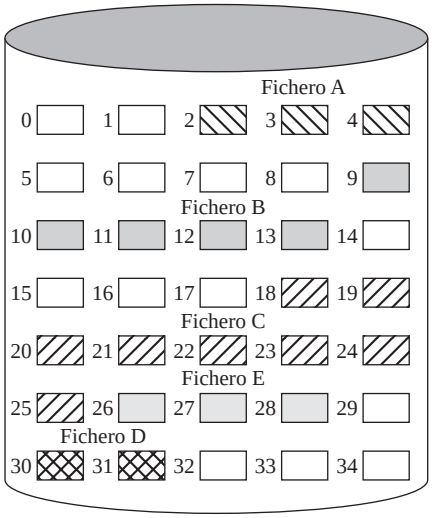


Tabla de asignación de ficheros

Nombre de fichero	Bloque inicial	Longitud
Fichero A	2	3
Fichero B	9	5
Fichero C	18	8
Fichero D	30	2
Fichero E	26	3

Figura 12.7. Asignación de fichero contiguo.

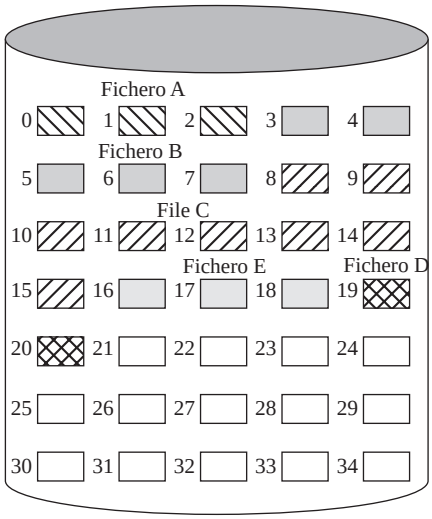


Tabla de asignación de ficheros

Nombre de fichero	Bloque inicial	Longitud
Fichero A	0	3
Fichero B	3	5
Fichero C	8	8
Fichero D	19	2
Fichero E	16	3

Figura 12.8. Asignación de fichero contiguo (después de la compactación).

la preasignación es posible, es más común asignar bloques cuando se necesita. La selección de bloques es ahora una cuestión sencilla: cualquier bloque libre se puede añadir a una cadena. No hay fragmentación externa de la que preocuparse porque sólo se necesita un bloque cada vez. Este tipo de organización física se adapta mejor a ficheros secuenciales que se procesan secuencialmente. Seleccionar un bloque individual de un fichero requiere seguir la cadena hasta alcanzar el bloque deseado.

Una consecuencia del encadenamiento, tal como se describe, es que no existe principio de proximidad. Por tanto, si es necesario traer varios bloques de fichero a la vez, como en el procesamiento secuencial, se requiere una serie de accesos a diferentes partes del disco. Esto es tal vez un efecto más

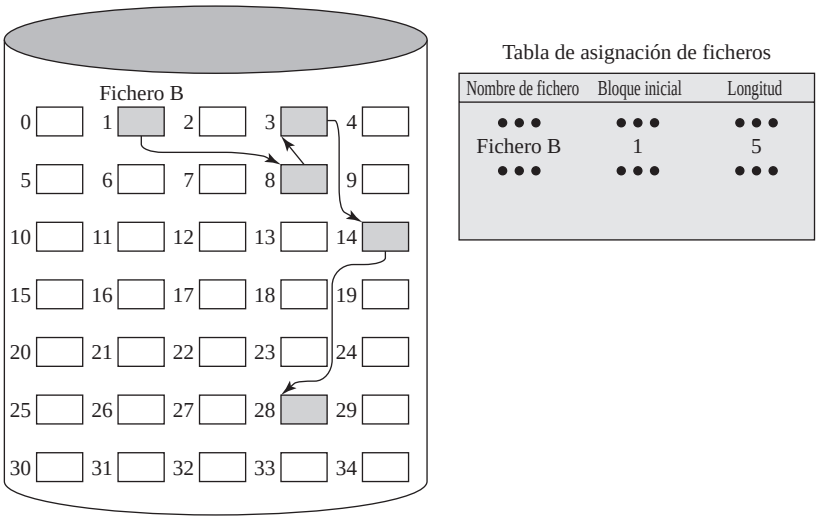


Figura 12.9. Asignación encadenada.

significativo en un sistema monousuario, pero también podría ser preocupante en el caso de un sistema compartido. Para resolver este problema, algunos sistemas consolidan ficheros periódicamente (Figura 12.10).

La **asignación indexada** resuelve muchos de los problemas de la asignación contigua y encadenada. En este caso, la tabla de asignación de ficheros contiene un índice separado de un nivel por cada fichero; el índice tiene una entrada por cada porción asignada al fichero. Típicamente, los índices de fichero no se almacenan físicamente como parte de la tabla de asignación de ficheros. Por el contrario, el índice de ficheros para un fichero se guarda en un bloque separado y la entrada para fichero en la tabla de asignación de ficheros apunta a dicho bloque. La asignación puede realizarse me-

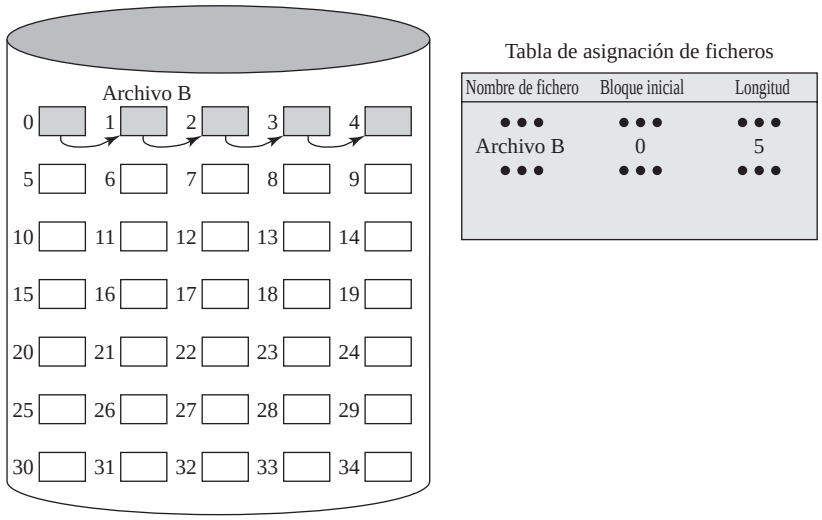


Figura 12.10. Asignación encadenada (después de la consolidación).

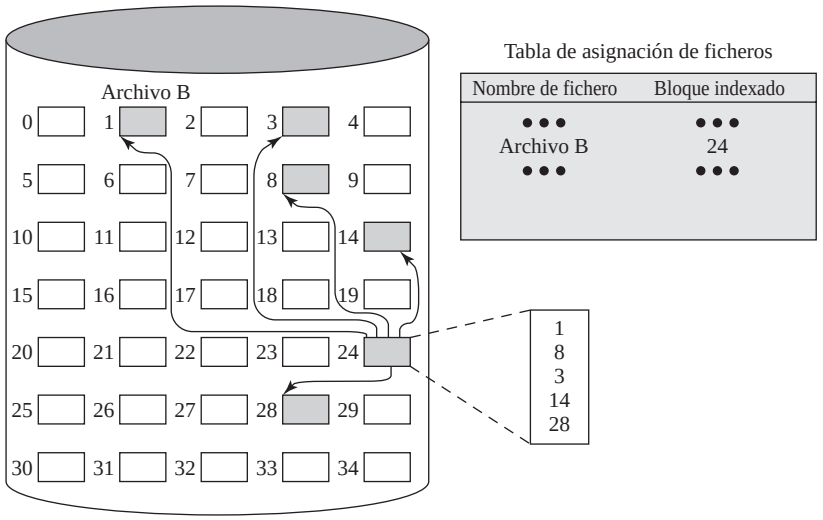


Figura 12.11. Asignación indexada con porciones de bloques.

diante bloques de tamaño fijo (Figura 12.11) o porciones de tamaño variable (Figura 12.12). La asignación por bloques elimina la fragmentación externa, mientras que la asignación por porciones de tamaño variable mejora la proximidad. En cualquier caso, la consolidación de ficheros se puede realizar de vez en cuando. La consolidación de ficheros reduce el tamaño del índice en el caso de porciones de tamaño variable, pero no en el caso de asignación de bloques. La asignación indexada da soporte tanto a acceso secuencial como directo a los ficheros y por tanto es la forma más popular de asignación de ficheros.

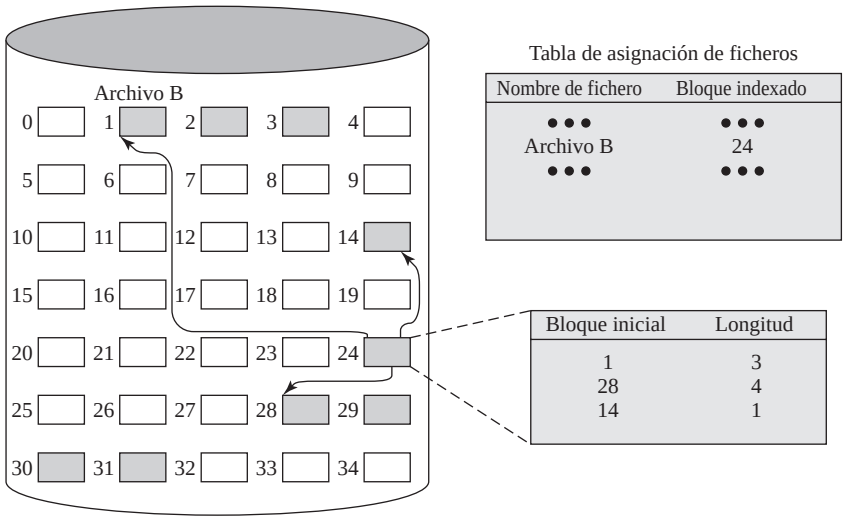


Figura 12.12. Asignación indexada con porciones de tamaño variable.

GESTIÓN DE ESPACIO LIBRE

De la misma forma que se asigna el espacio a los ficheros, así también se debe gestionar el espacio que no está actualmente asignado a ningún fichero. Para llevar a cabo cualquiera de las técnicas de asignación de ficheros descritas previamente, es necesario saber qué bloques están disponibles en el disco. Por tanto se necesita una **tabla de asignación de disco** en adición a la tabla de asignación de ficheros. Se discutirán aquí varias técnicas que se han implementado.

Tablas de bits

Este método utiliza un vector que está formado por un bit por cada bloque en el disco. Cada entrada 0 corresponde a un bloque libre y cada 1 corresponde a un bloque en uso. Por ejemplo, para la composición de disco de la Figura 12.7, se necesita un vector de longitud 35 con el siguiente contenido:

00111000011111000011111111111011000

Una tabla de bits tiene la ventaja de que es relativamente fácil encontrar un bloque libre o un grupo contiguo de bloques libres. Por tanto, una tabla de bits trabaja bien con cualquiera de los métodos de asignación de ficheros descrito. Otra ventaja es que esta estructura es tan pequeña como sea posible. La cantidad de memoria (en bytes) requerida para un mapa de bits de bloques es

$$\frac{\text{Tamaño de disco en bytes}}{8 \times \text{tamaño de bloque del sistema de ficheros}}$$

Por tanto, para un disco de 16 Gbytes con bloques de 512 bits, la tabla de bits ocupa 4 Mbytes. ¿Se puede almacenar la tabla de bits de 4 Mbytes en memoria principal? Si es así, entonces a la tabla de bits se puede acceder sin necesidad de acceder a disco. Pero incluso con los tamaños de memoria de hoy, 4 Mbytes es una cantidad considerable de memoria principal para dedicar a una única función. La alternativa es poner la tabla de bits en disco. Pero una tabla de bits de 4 Mbytes requeriría alrededor de 8000 bloques de disco. No se puede hacer una búsqueda sobre esta cantidad de espacio de disco cada vez que se necesita un bloque, de tal forma que una tabla de bits residente en memoria es indicada.

Incluso cuando la tabla de bits está en memoria principal, una búsqueda exhaustiva de la tabla puede ralentizar el rendimiento del sistema de ficheros hasta un grado inaceptable. Esto es especialmente cierto cuando el disco está casi lleno y hay pocos bloques libres restantes. De forma análoga, la mayoría de los sistemas de ficheros que utilizan tablas de bits mantienen estructuras de datos auxiliares que resumen los contenidos de subrangos de la tabla de bits. Por ejemplo, la tabla se podía dividir lógicamente en varios subrangos de igual tamaño. Una tabla resumen podría incluir, para cada subrango, el número de bloques libres y el número de bloques libres contiguos de tamaño máximo. Cuando el sistema de ficheros necesita varios bloques contiguos, puede analizar la tabla resumen para encontrar un subrango apropiado y entonces buscar dentro de dicho subrango.

Porciones libres encadenadas

Las porciones libres se pueden encadenar utilizando un puntero y valor de longitud en cada porción libre. Este método tiene una sobrecarga de espacio insignificante, porque no se necesita una tabla de asignación de disco, sino simplemente un puntero al comienzo de la cadena y la longitud de la primera porción. Este método es apropiado para todos los métodos de asignación de ficheros. Si se asigna un

bloque cada vez, simplemente hay que escoger el bloque libre en la cabeza de la cadena y ajustar el primer puntero o el valor de la longitud. Si la asignación se hace de una porción de longitud variable, se debe utilizar un algoritmo de primer ajuste: se cargan las cabeceras de las porciones para determinar la siguiente porción libre apropiada en la cadena. De nuevo, se ajustan el puntero y los valores de longitud.

Este método tiene sus propios problemas. Después de cierto uso, el disco se quedará bastante fragmentado y muchas porciones serán de la longitud de un único bloque. También obsérvese que cada vez que se asigne un bloque, se necesita leer el bloque primero para recuperar el puntero al nuevo primer bloque libre antes de escribir datos en dicho bloque. Si se necesitan asignar muchos bloques individuales de una vez para una operación sobre ficheros, esto ralentiza en gran medida la creación de ficheros. Similarmente, borrar ficheros altamente fragmentados es una tarea que consume mucho tiempo.

Indexación

La técnica de indexación trata el espacio libre como un fichero y utiliza una tabla de índices tal y como se describió en la asignación de ficheros. Por motivos de eficiencia, el índice se debería utilizar en base a porciones de tamaño variable en lugar de bloques. Por tanto, hay una entrada en la tabla por cada porción libre en el disco. Esta técnica proporciona soporte eficiente a todos los métodos de asignación de ficheros.

Lista de bloques libres

En este método, a cada bloque se le asigna un número secuencialmente y la lista de los números de todos los bloques libres se mantiene en una porción reservada del disco. Dependiendo del tamaño del disco, se necesitarán 24 o 32 bits para almacenar un único número de bloque, de tal forma que el tamaño de la lista de bloques libres es 24 o 32 veces el tamaño de la correspondiente tabla de bits y por tanto debe almacenarse en disco y no en memoria principal. Sin embargo, este es un método satisfactorio. Considérense los siguientes puntos:

1. El espacio en disco dedicado a la lista de bloques libres es menor que el 1% del espacio total de disco. Si se utiliza un número de bloque de 32 bits, entonces la penalización de espacio es de 4 bytes por cada bloque de 512 bytes.
2. Aunque la lista de bloques libres es demasiado grande para almacenarla en memoria principal, hay dos técnicas efectivas para almacenar una pequeña parte de la lista en memoria principal.

    a) La lista se puede tratar como una pila (Apéndice 1B) con los primeros miles de elementos de la pila residentes en memoria principal. Cuando se asigna un nuevo bloque, se saca de la pila, que está en memoria principal. Similarmente, cuando se desasigna un bloque, se coloca en la pila. Cuando la porción de la pila en memoria se llena o se vacía, hay sólo una transferencia entre disco y memoria principal. Por tanto, esta técnica da acceso muy rápido en la mayoría de las ocasiones.

    b) La lista se puede tratar como una cola FIFO, con unos pocos miles de entradas desde la cabeza al final de la cola en memoria principal. Se asigna un bloque tomando la primera entrada de la cabeza de la cola y se desasigna añadiéndolo al final de la cola. Sólo hay una transferencia entre disco y memoria principal cuando la porción en memoria de la cabeza de la cola se vacía o la porción en memoria del final de la cola se llena.

En cualquiera de las estrategias listadas en el punto precedente (pila o cola FIFO), un hilo en segundo plano puede ordenar lentamente la lista en memoria o listas para facilitar la asignación contigua.

FIABILIDAD

Considérese el siguiente escenario:

1. El usuario A solicita una asignación de ficheros para añadir a un fichero existente.
2. La petición se concede y el disco y las tablas de asignación de ficheros se actualizan en memoria principal pero todavía no en disco.
3. El sistema falla y consecuentemente se reinicia.
4. El usuario B solicita una asignación de ficheros y se asigna espacio en disco que solapa la última asignación al usuario A.
5. El usuario A accede a una porción solapada a través de una referencia que se almacena dentro del fichero de A.

Esta dificultad surge debido al hecho de que el sistema mantiene una copia de la tabla de asignación de disco y la tabla de asignación de ficheros en memoria principal por motivos de eficiencia. Para prevenir este tipo de error, cuando se solicita una asignación de ficheros se pueden llevar a cabo los siguientes pasos:

1. Bloquear la tabla de asignación de disco en disco. Esto previene a otro usuario de causar alteraciones a la tabla hasta que esta asignación se complete.
2. Buscar espacio disponible en la tabla de asignación de disco. Esto supone que una copia de la tabla de asignación de disco siempre se guarda en memoria principal. Si no, debe primero traerse de disco.
3. Asignar espacio, actualizar la tabla de asignación de disco y actualizar el disco. Actualizar el disco supone escribir la tabla de asignación de disco en disco. Para la asignación de disco encadenado, también supone actualizar algunos punteros en disco.
4. Actualizar la tabla de asignación de disco y actualizar el disco.
5. Desbloquear la tabla de asignación de disco.

Esta técnica evitará errores. Sin embargo, cuando se asignan frecuentemente pequeñas porciones, el impacto en el rendimiento será substancial. Para reducir esta sobrecarga, se podría utilizar un esquema de asignación en lotes de almacenamiento. En este caso, se obtiene un lote de porciones libres en el disco para asignación. Las correspondientes porciones de disco se marcan «en uso». La asignación utilizando este lote puede realizarse en memoria principal. Cuando se finalice el lote, la tabla de asignación de disco se actualiza en disco y se puede adquirir un nuevo lote. Si ocurriera un fallo en el sistema, porciones del disco marcados «en uso» deben limpiarse de alguna forma antes de que se puedan reasignar. La técnica de limpieza dependerá de las características particulares del sistema de ficheros.

12.7. GESTIÓN DE FICHEROS DE UNIX

En el sistema de ficheros UNIX, se pueden distinguir seis tipos de ficheros:

- **Regulares u ordinarios.** Contiene datos arbitrarios en cero o más bloques de datos. Los ficheros regulares contienen información introducida por un usuario, una aplicación o una utilidad del sistema. El sistema de ficheros no impone ninguna estructura interna a un fichero regular sino que lo trata como una ristra de bytes.

- **Directorios.** Contiene una lista de nombres de ficheros más punteros a nodos-i asociados (nodos índice), descritos posteriormente. Los directorios se organizan jerárquicamente (Figura 12.4). Los directorios son realmente ficheros ordinarios con privilegios de protección de escritura especiales de tal forma que sólo el sistema de ficheros puede escribirlos, mientras que los programas de usuario tienen acceso de lectura.
- **Especiales.** No contienen datos, sino que proporcionan un mecanismo para asociar dispositivos físicos a nombres de ficheros. Se utilizan nombres de ficheros para acceder a los dispositivos periféricos, tales como terminales e impresoras. Cada dispositivo de E/S se asocia con un fichero especial, como se discutió en la Sección 11.8.
- **Tuberías con nombre.** Como se discutió en la Sección 6.7, una tubería es una utilidad de comunicación entre procesos. Una tubería guarda en un *buffer* los datos de su entrada de forma que un proceso que lea de la salida de la tubería reciba los datos del mismo modo que si leyera de una cola FIFO.
- **Enlaces.** En esencia, un enlace es un nombre alternativo de fichero para un fichero existente.
- **Enlaces simbólicos.** Se trata de un fichero de datos que contiene el nombre del fichero al que enlaza.

Esta sección trata la gestión de ficheros ordinarios, que corresponden a lo que la mayoría de los sistemas trata como ficheros.

NODOS-I

Todos los tipos de ficheros UNIX se administran por el sistema operativo mediante los nodos-i. Un nodo-i (nodo índice) es una estructura de control que contiene la información clave necesaria de un fichero particular para el sistema operativo. Varios nombres de ficheros se pueden asociar con un único nodo-i, pero un nodo-i activo se asocia con exactamente un fichero, y cada fichero es controlado por exactamente un nodo-i.

Los atributos del fichero así como sus permisos y otra información de control se almacenan en el nodo-i. La Tabla 12.4 lista los atributos de ficheros almacenados en el nodo-i de una implementación UNIX típica.

En el disco, hay una tabla de nodos-i, o lista de nodos-i, que contiene los nodos-i de todos los ficheros del sistema de ficheros. Cuando se abre un fichero, se trae su nodo-i a memoria principal y se almacena en una tabla de nodos-i residente en memoria.

ASIGNACIÓN DE FICHEROS

La asignación de ficheros se realiza a nivel de bloque. La asignación es dinámica, es decir, cuando se necesita, en lugar de utilizar preasignación. Por tanto, los bloques de un fichero en disco no son necesariamente contiguos. Se utiliza un método indexado para guardar traza de cada fichero, con parte del índice almacenado en el nodo-i del fichero. El nodo-i incluye 39 bytes de información de dirección que se organizan como trece direcciones de 3 bytes o punteros. Las primeras 10 direcciones apuntan a los primeros 10 bloques de datos del fichero. Si el fichero es mayor de 10 bloques, se utilizan uno o más niveles de indirección como se indica a continuación:

La dirección undécima en el nodo-i apunta a un bloque en disco que contiene la siguiente porción del índice. Esto se conoce como bloque indirecto simple. Este bloque contiene los punteros a siguientes bloques en el fichero.

Tabla 12.4. Información de un nodo-i UNIX residente en disco.

Modo de fichero	Espacio de 16 bits que almacena los permisos de acceso y ejecución asociados con el fichero.
	12-14 Tipo de fichero (regular, directorio, especial de caracteres o bloques, tubería FIFO)
	9-11 *Flags* de ejecución
	8 Permiso de lectura para el propietario
	7 Permiso de escritura para el propietario
	6 Permiso de ejecución para el propietario
	5 Permiso de lectura para el grupo del propietario
	4 Permiso de escritura para el grupo del propietario
	3 Permiso de ejecución para el grupo del propietario
	2 Permiso de lectura para el resto de usuarios
	1 Permiso de escritura para el resto de usuarios
	0 Permiso de ejecución para el resto de usuarios
Número de enlaces	Número de directorios que referencian a este nodo-i
Identificación del propietario	Propietario individual del fichero
Identificación de grupo del propietario	Grupo del propietario asociado con este fichero
Tamaño de fichero	Número de bytes del fichero
Direcciones de fichero	39 bytes de información de direcciones
Último acceso	Fecha del último acceso al fichero
Última modificación	Fecha de la última modificación del fichero
Nodo-i modificado	Fecha de la última modificación del nodo-i

Si el fichero contiene más bloques, la duodécima dirección del nodo-i apunta a un bloque indirecto doble. Este bloque contiene una lista de direcciones de bloques indirectos simples adicionales. Cada uno de los bloques indirectos simples, a su vez, contiene punteros a bloques de ficheros.

Si el fichero contiene todavía más bloques, la dirección decimotercera en el nodo-i apunta a un bloque indirecto triple que constituye un tercer nivel de indexación. Este bloque apunta a bloques indirectos dobles adicionales.

Todo esto se muestra en la Figura 12.13. La primera entrada del nodo-i contiene información sobre este fichero o directorio (Tabla 12.4). Las entradas restantes son las direcciones recién descritas. El número total de bloques de datos en el fichero depende de la capacidad de los bloques de tamaño fijo en el sistema. En UNIX System V, la longitud de un bloque es 1 Kbyte y cada bloque puede contener un total de 256 direcciones de bloque. Por tanto, el tamaño máximo de un fichero con este esquema es cerca de 16 Gbytes (Tabla 12.5).

Este esquema tiene varias ventajas:

1. El nodo-i es de tamaño fijo y relativamente pequeño y por tanto, se puede almacenar en memoria principal durante periodos largos.

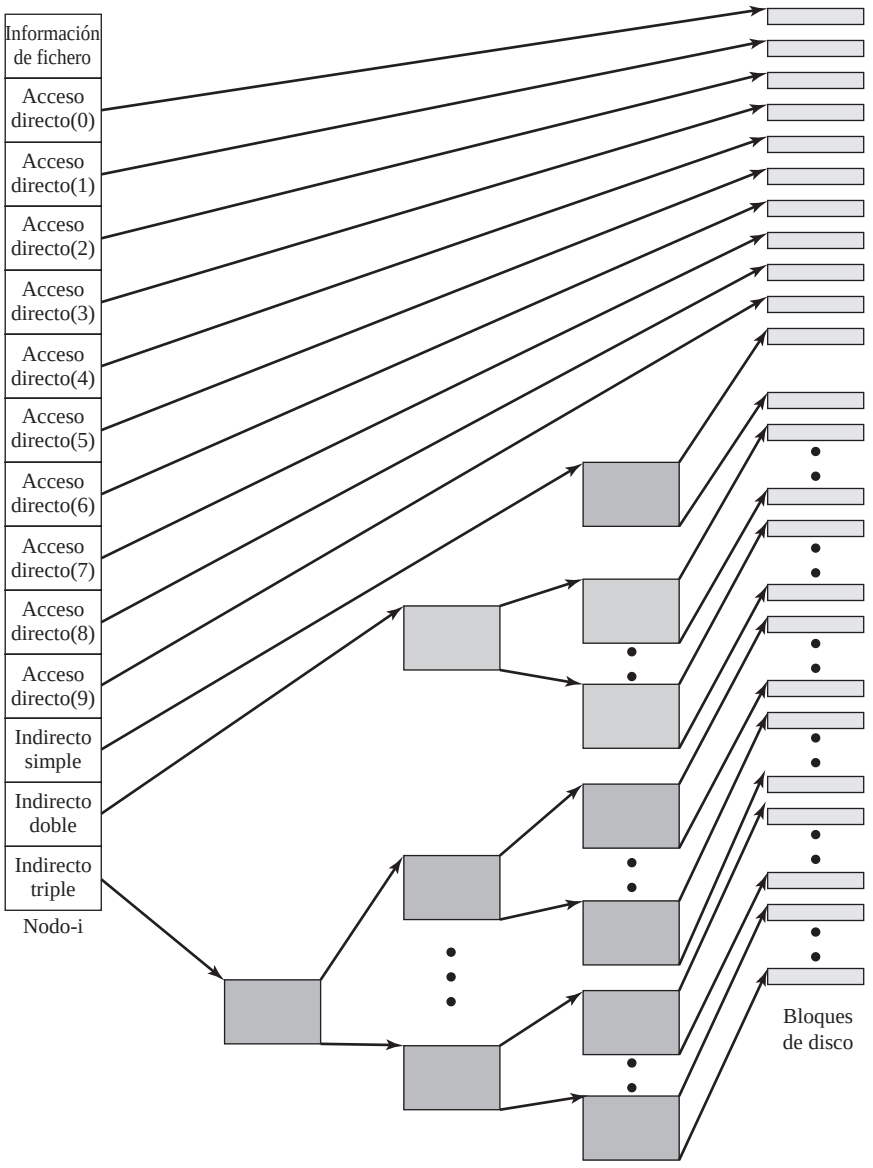


Figura 12.13. Disposición de un fichero UNIX en un disco.

Tabla 12.5. Capacidad de un fichero UNIX.

Nivel	Número de bloques	Número de bytes
Directo	10	10K
Simple indirecto	256	256K
Doble indirecto	256 x 256 = 65K	65M
Triple indirecto	256 x 65K = 16M	16G

2. A los ficheros más pequeños se puede acceder con poca o ninguna indirección, reduciendo el procesamiento y el tiempo de acceso a disco.
3. El tamaño máximo teórico de un fichero es suficientemente grande para satisfacer prácticamente todas las aplicaciones.

DIRECTORIOS

Los directorios se estructuran como un árbol jerárquico. Cada directorio contiene ficheros y/u otros directorios. Un directorio que se encuentra dentro de otro directorio se denomina subdirectorio. Como se mencionó anteriormente, un directorio es simplemente un fichero que contiene una lista de nombres de ficheros más punteros a nodos-i asociados. La Figura 12.14 muestra la estructura global. Cada entrada de directorio (entradaD) contiene un nombre para el fichero asociado o subdirectorio más un entero llamado el número-i (número índice). Cuando se accede al fichero o directorio, su número-i se utiliza como un índice en la tabla de nodos-i.

ESTRUCTURA DEL VOLUMEN

Un sistema de ficheros UNIX reside en un único disco lógico o partición de disco y se compone de los siguientes elementos:

- **Bloque de arranque.** Contiene el código requerido para arrancar el sistema operativo.
- **Superbloque.** Contiene atributos e información sobre el sistema de ficheros, tal como el tamaño de la partición y el tamaño de la tabla de nodos-i.
- **Tabla de nodos-i.** La colección de nodos-i para cada fichero.
- **Bloques de datos.** El espacio de almacenamiento disponible para los ficheros de datos y subdirectorios.

12.8. SISTEMA DE FICHEROS VIRTUAL LINUX

Linux incluye una utilidad versátil y potente para gestión de ficheros, diseñado para soportar una gran variedad de sistemas de gestión de ficheros y estructuras de ficheros. El enfoque usado en Linux consiste en hacer uso del **sistema de ficheros virtual (VFS)**, que presenta una única y uniforme interfaz de sistema de ficheros para los procesos de usuario. VFS define un modelo de ficheros común que es capaz de representar cualquier característica general y comportamiento de un sistema de ficheros concebible. VFS asume que los ficheros son objetos de un sistema de almacenamiento masivo del computador que comparten propiedades básicas sin tener en cuenta el sistema de ficheros concreto o el hardware subyacente. Los ficheros tienen nombres simbólicos que les permiten identificarse de forma única dentro de un directorio específico en el sistema de ficheros. Un fichero tiene un propietario, protección frente a accesos o modificaciones no autorizadas y otras propiedades. Un fichero se puede crear, leer, escribir o borrar. Para cualquier sistema de ficheros específico, se necesita un módulo de proyección que transforme las características del sistema de ficheros real a las características esperadas por el sistema de ficheros virtual.

La Figura 12.15 indica los ingredientes clave de la estrategia del sistema de ficheros Linux. Un proceso de usuario invoca una llamada al sistema de ficheros (por ejemplo, lectura) utilizando el esquema de ficheros VFS. VFS convierte esta llamada en una llamada al sistema de ficheros interno (del núcleo) que se pasa a una función de proyección del sistema de ficheros específico [por ejemplo,

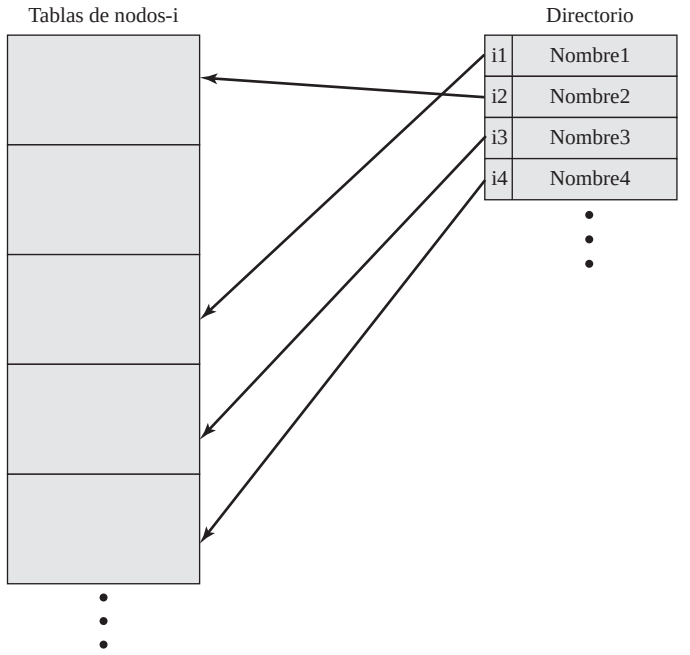


Figura 12.14. Directorios y nodos-i UNIX.

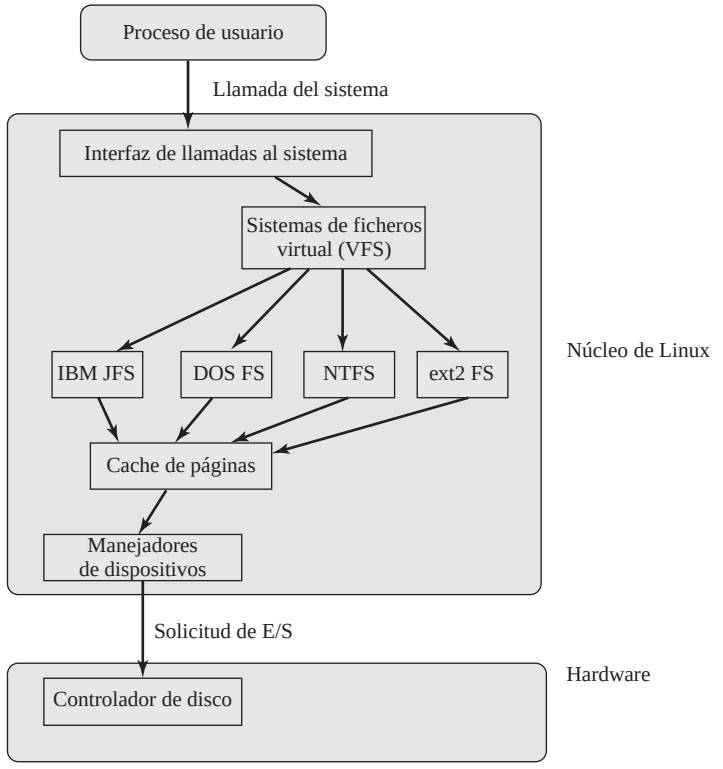


Figura 12.15. Contexto del sistema de ficheros virtual de Linux.

el sistema de ficheros JFS (*Journaling File System*) de IBM]. En la mayoría de los casos, la función de proyección es simplemente una proyección de las llamadas funcionales del sistema de ficheros desde un esquema a otro. En algunos casos, la función de proyección es más compleja. Por ejemplo, algunos sistemas de ficheros utilizan una tabla de asignación de ficheros (*File Allocation Table*, FAT), que almacena la posición de cada fichero en el árbol de directorios. En estos sistemas de ficheros, los directorios no son ficheros. En cualquier caso, la llamada original al sistema de ficheros se traduce en una llamada que es nativa para el sistema de ficheros destino. El software del sistema de ficheros destino es invocado entonces para llevar a cabo la función requerida sobre el fichero o directorio, bajo su control y almacenamiento secundario. Los resultados de la operación se comunican de nuevo al usuario de una forma similar.

La Figura 12.16 indica el papel que VFS juega dentro del núcleo de Linux. Cuando un proceso inicia una llamada al sistema orientada a ficheros (por ejemplo, lectura), el núcleo llama a una función en VFS. Esta función gestiona los aspectos independientes del sistema de ficheros e inicia una llamada a una función en el código del sistema de ficheros destino. Esta llamada pasa a través de una función de proyección que convierte la llamada VFS en una llamada al sistema de ficheros destino. VFS es independiente de cualquier sistema de ficheros, de modo que la implementación de una función de proyección debe ser parte de la implementación de un sistema de ficheros en Linux. El sistema de ficheros destino convierte la petición del sistema de ficheros en instrucciones orientadas a dispositivo que se pasan al controlador del dispositivo mediante funciones de *cache* de páginas.

VFS es un esquema orientado a objetos. Debido a que está escrito en C, en lugar de en un lenguaje que dé soporte a la programación de objetos (como C++ o Java), los objetos VFS se implementan simplemente como estructuras de datos C. Cada objeto contiene tanto datos como punteros a las funciones implementadas del sistema de ficheros que operan sobre los datos. Los cuatro tipos de objetos primarios en VFS son los siguientes:

- **Objeto superbloque.** Representa un sistema de ficheros montado específico.
- **Objeto nodo-i.** Representa un fichero específico.
- **Objeto entrada de directorio.** Representa una entrada de directorio específica.
- **Objeto de fichero.** Representa un fichero abierto asociado con un proceso.

Este esquema se basa en los conceptos utilizados en el sistema de ficheros de UNIX, tal y como se describió en la Sección 12.7. Los conceptos clave del sistema de ficheros de UNIX a recordar son los siguientes. Un sistema de ficheros está compuesto por una organización jerárquica de directorios.

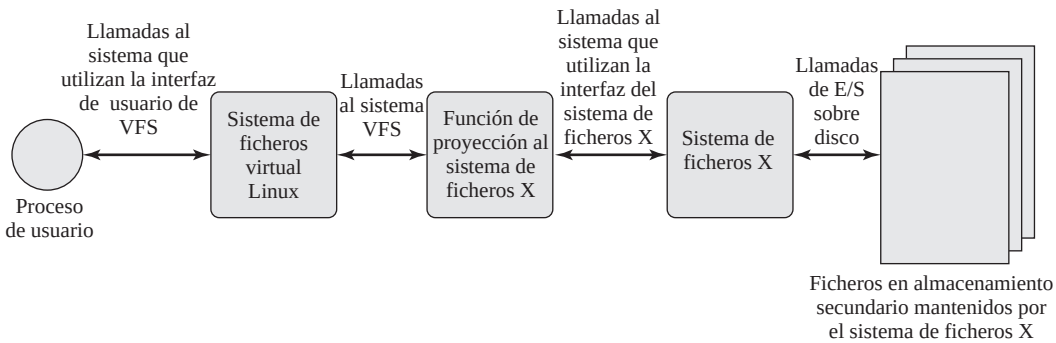


Figura 12.16. Concepto del sistema de ficheros virtual de Linux.

Un directorio es análogo a una carpeta y puede contener ficheros y/u otros directorios. Debido a que un directorio puede contener otros directorios, se forma una estructura de árbol. Un camino a través de la estructura de árbol desde la raíz está formado por una secuencia de entradas de directorio, acabando en una entrada de directorio (entradaD) o un nombre de fichero. En UNIX, un directorio se implementa como un fichero que lista los ficheros y directorios contenidos en él. Por tanto, las operaciones de ficheros se pueden llevar a cabo tanto sobre ficheros como directorios.

EL OBJETO SUPERBLOQUE

El objeto superbloque almacena información que describe un sistema de ficheros específico. Típicamente, el superbloque corresponde al superbloque del sistema de ficheros o bloque de control del sistema de ficheros, que se almacena en un sector especial en el disco.

El objeto superbloque está formado por varios elementos de datos. Ejemplos de estos elementos incluyen los siguientes:

- El dispositivo sobre el cual el sistema de ficheros está montado.
- El tamaño de bloque básico del sistema de ficheros.
- El *dirty flag*, que indica que se ha cambiado el superbloque pero no se ha escrito a disco.
- Tipo de fichero.
- *Flags*, como el de sólo lectura.
- Puntero al directorio raíz del sistema de ficheros.
- Lista de ficheros abiertos.
- Semáforo para controlar el acceso al sistema de ficheros.
- Lista de operaciones de superbloque.

El último elemento en la lista precedente se refiere a un objeto de operaciones contenido en el objeto superbloque. El objeto operación define los métodos de objeto (funciones) que el núcleo puede invocar sobre el objeto superbloque. Los métodos definidos para el objeto superbloque incluyen las siguientes:

- *** read_inode.** Leer un nodo-i específico desde un sistema de ficheros montado.
- * **write_inode.** Escribir un nodo-i dado a disco.
- * **put_inode.** Obtener un nodo-i.
- * **delete_inode.** Borrar un nodo-i del disco.
- * **notify_change.** Invocado cuando se cambian los atributos.
- * **put_super.** Llamado por VFS cuando el sistema de ficheros está desmontado, para obtener el superbloque dado.
- * **write_super.** Invocado cuando VFS decide que el superbloque necesita escribirse en disco.
- * **statfs.** Obtiene las estadísticas del sistema de ficheros.
- * **remount_fs.** Llamado por VFS cuando el sistema de ficheros es montado de nuevo con nuevas opciones de montaje.
- * **clear_inode.** Obtener nodo-i y borrar cualquier página que contenga datos relacionados.

EL OBJETO NODO-I

Un nodo-i está asociado con cada fichero. El objeto nodo-i contiene toda la información sobre un determinado fichero excepto su nombre y el contenido real del fichero. Los elementos contenidos en un objeto nodo-i incluyen el propietario, grupo, permisos, tiempos de acceso para un fichero, tamaño de los datos que contiene y número de enlaces.

El objeto nodo-i también incluye un objeto operaciones de nodo-i, que describe las funciones implementadas en el sistema de ficheros que VFS puede invocar sobre el nodo-i. Los métodos definidos por el objeto nodo-i incluyen los siguientes:

- **create.** Crear un nodo-i nuevo para un fichero regular asociado con un objeto entradaD en algún directorio.
- **lookup.** Buscar en un directorio el nodo-i correspondiente a un nombre de fichero.
- **mkdir.** Crear un nuevo nodo-i para un directorio asociado a un objeto entradaD en algún directorio.

EL OBJETO ENTRADAD

Una entradaD (entrada de directorio) es un componente específico de una ruta. El componente puede ser un nombre de directorio o un nombre de fichero. El objeto entradaD facilita el acceso a los ficheros y directorios y se utilizan en una *cache* de entradaD para dicho propósito.

EL OBJETO FICHERO

El objeto fichero se utiliza para representar un fichero abierto por un proceso. El objeto se crea en respuesta a la llamada al sistema *open()* y se destruye en respuesta a la llamada al sistema *close()*. El objeto fichero está formado por un conjunto de elementos, que incluye los siguientes:

- Objeto entradaD asociado con el fichero.
- Sistema de ficheros que contiene el fichero.
- Contador de uso del objeto fichero.
- Identificador del usuario.
- Identificador de grupo del usuario.
- Puntero de posición del fichero, que es la posición actual en el fichero desde la cual tendrá lugar la siguiente operación.

El objeto fichero también incluye un objeto operaciones de nodo-i, que describe las funciones implementadas del sistema de ficheros que VFS puede invocar sobre el objeto fichero. Los métodos definidos para el objeto fichero incluyen la lectura, escritura, apertura, creación y bloqueo.

12.9. SISTEMA DE FICHEROS DE WINDOWS

Windows da soporte a varios sistemas de ficheros, incluyendo el sistema FAT (*File Allocation Table*: tabla de asignación de ficheros) que ejecutan en Windows 95, MS-DOS y OS/2. Pero los desarrolla-

dores de Windows también diseñaron un nuevo sistema de ficheros, el sistema de ficheros de Windows (NTFS), que está pensado para alcanzar requisitos de altas prestaciones en estaciones de trabajo y servidores. Ejemplos de aplicaciones de altas prestaciones incluyen las siguientes:

- Aplicaciones cliente/servidor tales como los servidores de ficheros, servidores de computación y servidores de bases de datos.
- Ingeniería intensiva de recursos y aplicaciones científicas.
- Aplicaciones de red para grandes sistemas corporativos.

CARACTERÍSTICAS CLAVE DE NTFS

NTFS es un sistema de ficheros flexible y potente, construido, como se verá, en un modelo de sistema de ficheros elegantemente simple. Las características más notables de NTFS incluyen las siguientes:

- **Recuperación.** Uno de los requisitos más importantes del nuevo sistema de ficheros Windows es la capacidad de recuperarse frente a errores en el sistema y los fallos de disco. En el caso de dichos fallos, NTFS es capaz de reconstruir volúmenes de disco y devolverlos a un estado consistente. Esto se lleva a cabo utilizando un modelo de procesamiento de transacciones para los cambios en el sistema de ficheros; cada cambio significativo se trata como una acción atómica que se realiza de forma completa o no se lleva a cabo en absoluto. Cada transacción que está en proceso cuando se produce un fallo es a continuación terminada completamente o bien se deja el sistema como antes de su ejecución. Adicionalmente, NTFS utiliza almacenamiento redundante para datos del sistema de ficheros críticos, de forma que un fallo en el sector de un disco no cause la pérdida de datos que describen la estructura y estado del sistema de ficheros.
- **Seguridad.** NTFS utiliza el modelo de objetos de Windows para forzar la seguridad. Un fichero abierto se implementa como un objeto fichero con un descriptor de seguridad que define sus atributos de seguridad.
- **Discos y ficheros grandes.** NTFS soporta discos y ficheros muy grandes de forma más eficiente que la mayoría del resto de los sistemas de ficheros, incluyendo FAT.
- **Múltiples flujos de datos.** Los contenidos reales de un fichero se tratan como un flujo de bytes. En NTFS es posible definir múltiples flujos de datos para un único fichero. Un ejemplo de la utilidad de esta característica es que permite que sistemas Macintosh remotos utilicen Windows para almacenar y recuperar ficheros. En Macintosh, cada fichero tiene dos componentes: los datos del fichero y un contenedor de recursos que tiene información sobre el fichero. NTFS trata estos dos componentes como dos flujos de datos.
- **Facilidad general de indexación.** NTFS asocia una colección de atributos con cada fichero. El conjunto de descripciones de fichero en el sistema de gestión de ficheros se organiza como una base de datos relacional, de forma que los ficheros se pueden indexar por cualquier atributo.

VOLÚMENES NTFS Y ESTRUCTURA DE FICHEROS

NTFS hace uso de los siguientes conceptos de almacenamiento de disco:

- **Sector.** La unidad física de almacenamiento más pequeña en el disco. El tamaño de los datos en bytes es una potencia de 2 y es casi siempre 512 bytes.

- **Agrupación.** Uno o más sectores contiguos (próximos entre sí en la misma pista). El tamaño de la agrupación en sectores es una potencia de 2.
- **Volumen.** Una partición lógica de un disco, formada por una o más agrupaciones y utilizada por un sistema de ficheros para asignar espacio. En cualquier momento, un volumen está formado por información del sistema de ficheros, una colección de ficheros y cualquier espacio restante adicional sin asignar del volumen que se puede asignar a los ficheros. Un volumen puede ser todo o parte de un único disco o se puede extender entre múltiples discos. Si se emplea RAID 5 hardware o software, un volumen está compuesto por tiras a través de múltiples discos. El tamaño máximo de un volumen en NTFS es de 2^{64} bytes.

La agrupación es la unidad fundamental de asignación en NTFS, el cual no reconoce sectores. Por ejemplo, supóngase que cada sector es 512 bytes y el sistema se configura con dos sectores por agrupación (una agrupación = 1K bytes). Si un usuario crea un fichero de 1600 bytes, se asignan dos agrupaciones al fichero. Posteriormente, si el usuario actualiza el fichero a 3200 bytes, se asignan otras dos agrupaciones. Las agrupaciones asignadas a un fichero no necesitan ser contiguas; es posible fragmentar un fichero en el disco. Actualmente, el tamaño de fichero máximo soportado por NTFS es 2^{32} agrupaciones, lo que equivale a un máximo de 2^{48} bytes. Una agrupación puede tener como máximo 2^{16} bytes.

El uso de agrupaciones para la asignación hace a NTFS independiente del tamaño de sector físico. Esto habilita a NTFS a soportar fácilmente discos no estándares que no tengan un tamaño de sector de 512 bytes y a soportar eficientemente discos y ficheros muy grandes utilizando un tamaño de agrupación más grande. La eficiencia procede del hecho de que los sistemas de ficheros deben guardar traza de cada agrupación asignada a cada fichero; con agrupaciones mayores, hay menos elementos que gestionar.

La Tabla 12.6 muestra los tamaños de agrupación por omisión para NTFS. Los tamaños por omisión dependen del tamaño del volumen. El tamaño de agrupación que se utiliza para un volumen particular lo establece NTFS cuando el usuario solicita que se formatee un volumen.

Tabla 12.6. Particiones y tamaños de agrupaciones de Windows NTFS.

Tamaño de volumen	Sectores por agrupación	Tamaño de agrupación
≤ 512 Mbytes	1	512 bytes
512 Mbytes - 1 Gbyte	2	1K
1 Gbyte – 2 Gbytes	4	2K
2 Gbytes – 4 Gbytes	8	4K
4 Gbytes – 8 Gbytes	16	8K
8 Gbytes – 16 Gbytes	32	16K
16 Gbytes – 32 Gbytes	64	32K
> 32 Gbytes	128	64K

ESTRUCTURA DE UN VOLUMEN NTFS

NTFS utiliza un enfoque notablemente simple pero potente para organizar información de un volumen en el disco. Cada elemento del volumen es un fichero, y cada fichero está formado por una colección de atributos. Incluso el contenido de un fichero se trata como un atributo. Con esta estructura sencilla, unas pocas funciones de propósito general son suficientes para organizar y gestionar un sistema de ficheros.

La Figura 12.17 muestra la estructura de un volumen NTFS, que está formado por cuatro regiones. Los primeros sectores de un volumen están ocupados por el **sector de arranque de la partición** (aunque se llame sector, puede estar formado por un máximo de 16 sectores), que contiene información sobre la estructura del volumen, las estructuras del sistema de ficheros así como la información de arranque de inicio y el código. Esto es seguido por la **tabla maestra de ficheros** (*Master File Table*, MFT), que contiene información sobre todos los ficheros y carpetas (directorios) de este volumen NTFS así como la información sobre espacio disponible no asignado. En esencia, el MFT es una lista de todos los contenidos de este volumen NTFS, organizada como un conjunto de filas en una estructura de base de datos relacional.

Siguiendo al MFT hay una región, típicamente de 1 Mbyte de longitud, conteniendo los **ficheros del sistema**. Entre los ficheros de esta región se encuentran los siguientes:

- **MFT2.** Un espejo de las tres primeras filas del MFT, utilizado para garantizar el acceso al MFT en el caso de un fallo de un único sector.
- **Fichero registro.** Una lista de pasos de transacciones utilizadas para la recuperación en NTFS.
- **Mapa de bits de las agrupaciones.** Una representación del volumen, mostrando qué agrupaciones están en uso.
- **Tabla de definición de atributos.** Define los tipos de atributos soportados en este volumen e indica si se pueden indexar o si se pueden recuperar durante una operación de recuperación del sistema.

Tabla maestra de ficheros

El corazón del sistema de ficheros Windows es el MFT. El MFT se organiza como una tabla de filas de longitud variable, llamadas registros. Cada fila describe un fichero o una carpeta de este volumen, incluyendo el propio MFT, que se trata como un fichero. Si un fichero es suficientemente pequeño, el fichero completo se localiza en una fila del MFT. En otro caso, la fila para dicho fichero contiene información parcial y el resto del fichero se encuentra en otras agrupaciones disponibles del volumen. Un puntero a dichas agrupaciones se encuentra en la fila MFT del fichero.

Cada registro del MFT está formado por un conjunto de atributos que sirve para definir las características del fichero (o carpeta) y los contenidos del mismo. La Tabla 12.7 lista los atributos que se pueden encontrar en una fila, habiéndose sombreado los atributos obligatorios.

RECUPERACIÓN

NTFS hace posible llevar el sistema de ficheros a un estado consistente cuando ha habido un error de sistema o un fallo de disco. Los elementos clave que soportan la recuperación son los siguientes (Figura 12.18):

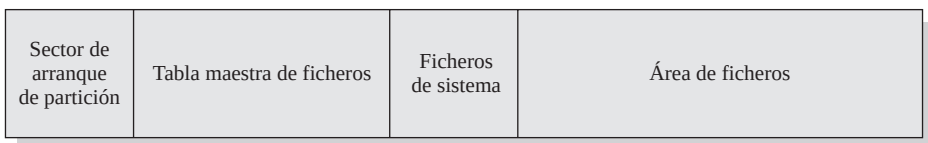


Figura 12.17. Disposición del volumen de NTFS.

Tabla 12.7. Tipos de atributos de ficheros y directorios de Windows NTFS.

Tipo de atributos	Descripción
Información estándar	Incluye los atributos de acceso (sólo lectura, lectura y escritura, etc.); sellos de tiempo, incluyendo la fecha de creación y última modificación; y cuantos directorios apuntan al fichero (número de enlaces).
Lista de atributos	Una lista de atributos que componen el fichero y la referencia de ficheros del registro de ficheros MFT en el que se localiza cada atributo. Utilizado cuando todos los atributos no caben en un único registro de ficheros MFT.
Nombre de fichero	Un fichero o directorio debe tener uno o más nombres
Descriptor de seguridad	Especifica quién es el propietario del fichero y quién puede acceder a él.
Datos	Los contenidos del fichero. Un fichero tiene un conjunto de atributos sin nombre por omisión y podría tener uno o más atributos de datos con nombre.
Raíz índice	Usado para implementar carpetas
Asignación índice	Usado para implementar carpetas
Información de volumen	Incluye información relacionada con el volumen, tal como la versión y el nombre del volumen.
Mapa de bits	Proporciona un mapa que representa registros en uso del MFT o carpeta

Nota: Las filas coloreadas se refieren a los atributos de ficheros requeridos; los otros atributos son opcionales.

- **Gestor de E/S.** Incluye el controlador NTFS, que gestiona las funciones básicas de apertura, cierre, lectura y escritura de NTFS. Adicionalmente, el módulo software de RAID FT-DISK se puede configurar para su uso.
- **Servicio de fichero de registro.** Mantiene un registro de las escrituras de disco. El fichero de registro se utiliza para recuperar un volumen formateado NTFS en el caso de un fallo del sistema.
- **Gestor de cache.** Responsable del *caching* de las lecturas y escrituras de fichero para incrementar el rendimiento. El gestor de *cache* optimiza la E/S de disco utilizando escritura diferida y técnicas de transacciones diferidas, descritas en la Sección 11.8.
- **Gestor de memoria virtual.** NTFS accede a ficheros en *cache* mediante la proyección de las referencias al fichero a las referencias a memoria virtual, leyendo y escribiendo en la memoria virtual.

Es importante observar que los procedimientos de recuperación utilizados por NTFS se diseñan para recuperar los datos del sistema de ficheros, no los contenidos del fichero. Por tanto, debido a un error, el usuario nunca debería perder un volumen o la estructura del directorio/fichero. Sin embargo, el sistema de ficheros no garantiza los datos del usuario. Proporcionar recuperación completa, incluyendo los datos de usuario, implicaría el uso de una facilidad de recuperación más elaborada y que supone un mayor tiempo de ejecución.

La esencia de la capacidad de recuperación de NTFS se encuentra en el uso de registros (*logging*). Cada operación que altera un sistema de ficheros se trata como una transacción. Cada subope-

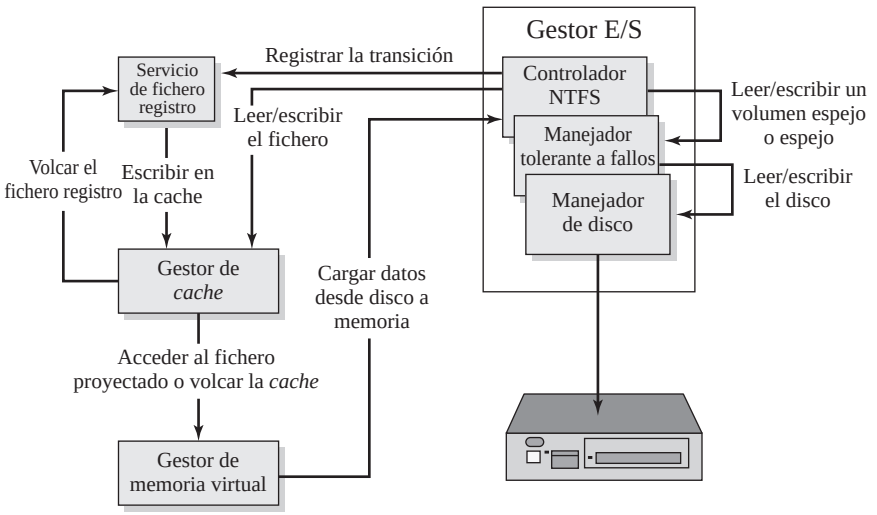


Figura 12.18. Componentes de Windows NTFS.

ración de una transacción que altera estructuras de datos del sistema de ficheros importantes se graba en un fichero de registro antes de grabarse en el volumen del disco. Utilizando el registro, una transacción parcialmente completada en el momento del error se puede rehacer posteriormente o deshacer cuando el sistema se recupera.

En términos generales, estos son los pasos tomados para asegurar la recuperación, como se describe en [CUST94]:

1. NTFS primero llama al registro del sistema de ficheros para grabar en el registro de la cache cualquier transacción que modificará la estructura del volumen.
2. NTFS modifica el volumen (en la cache).
3. El gestor de la cache llama al registro del sistema de ficheros para volcar el fichero registro al disco.
4. Una vez que el registro se actualiza de forma segura en el disco, el gestor de cache vuelca los cambios al disco.

12.10. RESUMEN

Un sistema de gestión de ficheros es un conjunto de software de sistema que proporciona servicios a usuarios y aplicaciones en el uso de ficheros, incluyendo accesos a ficheros, mantenimiento de directorios y control de acceso. El sistema de gestión de ficheros se ve típicamente como un servicio del sistema que el sistema operativo sirve, en lugar de ser parte del sistema operativo en sí. Sin embargo, en cualquier sistema, al menos parte de las funciones de gestión de ficheros se lleva a cabo por el sistema operativo.

Un fichero está formado por una colección de registros. La forma en la que estos registros se acceden determina su organización lógica, y hasta cierto punto su organización física en disco. Si un fichero se procesa primariamente como una entidad, la organización secuencial es la más sencilla y la más apropiada. Si se necesita acceso secuencial, pero también se desea acceso aleatorio al fichero in-

dividual, un fichero secuencial indexado puede proporcionar el mejor rendimiento. Si los accesos al fichero son principalmente aleatorios, un fichero indexado o *hash* puede ser el más apropiado.

Sea cual sea la estructura del fichero escogida, se necesita un servicio de directorios. Esto permite que los ficheros se organicen de forma jerárquica. Esta organización es útil para que el usuario guarde traza de los ficheros y es útil para que los sistemas de gestión de ficheros proporcionen control de acceso y otros servicios al usuario.

Los registros de los ficheros, incluso si son de tamaño fijo, generalmente no están conformes al tamaño de un bloque de disco físico. Por tanto, se necesita alguna especie de estrategia de asignación de bloques. Un compromiso entre la complejidad, el rendimiento y la utilización del espacio determina la estrategia de asignación de bloques a utilizar.

Una función clave de cualquier esquema de gestión de ficheros es la gestión del espacio de disco. Parte de esta función es la estrategia para asignar bloques de disco a un fichero. Varios métodos pueden utilizarse, y una gran variedad de estructuras de datos se utilizan para guardar traza de la asignación para cada fichero. Adicionalmente, se debe gestionar el espacio en disco que no se ha asignado. Esta última función consiste principalmente en mantener una tabla de asignación de disco que indique qué bloques están libres.

12.11. LECTURAS RECOMENDADAS

Existen varios libros buenos sobre gestión de ficheros. Todos ellos se enfocan en sistemas de gestión de ficheros, pero también tratan aspectos relacionados con el sistema operativo. Quizás el más útil es [WIED87], que sigue un enfoque cuantitativo para la gestión de ficheros y trata todos los aspectos descritos en la Figura 12.2, desde la planificación de disco a la estructura de ficheros. [LIVA90] hace hincapié en las estructuras de los ficheros, proporcionando una buena y abundante revisión de análisis de rendimiento comparativos. [GROS86] proporciona una visión equilibrada en aspectos relacionados con los métodos de E/S de ficheros y métodos de acceso a ficheros. También contiene una descripción general de todas las estructuras de control necesitadas por un sistema de ficheros. Éstas proporcionan una lista útil para el diseño de los sistemas de ficheros. [FOLK98] enfatiza el procesamiento de los ficheros, tratando los temas de mantenimiento, búsqueda, ordenación y compartición.

El sistema de ficheros Linux se examina detalladamente en [LOVE04] y [BOVE03]. Una buena descripción se recoge en [RUBI97].

[CUST94] proporciona una buena descripción del sistema de ficheros NT. [NAGA97] cubre el material más detalladamente.

BOVE03 Bovet, D., and Cesati, M. *Understanding the Linux Kernel.* Sebastopol, CA: O'Reilly, 2003.

CUST94 Custer, H. *Inside the Windows NT File System.* Redmond, WA: Microsoft Press, 1994.

FOLK98 Folk, M., and Zoellick, B. *File Structures: An Object-Oriented Approach with* C++. Reading, MA:Addison-Wesley, 1998.

GROS86 Grosshans,D. *File Systems: Design and Implementation.* Englewood Cliffs, NJ: Prentice Hall, 1986.

LIVA90 Livadas, P. *File Structures: Theory and Practice.* Englewood Cliffs, NJ: Prentice Hall, 1990.

LOVE04 Love, R. *Linux Kernel Development.* Indianapolis, IN: Sams Publishing, 2004.

NAGA97 Nagar, R. *Windows NT File System Internals.* Sebastopol, CA: O'Reilly, 1997.

RUBI97 Rubini, A. «The Virtual File System in Linux.» *Linux Journal*, May 1997.

WIED87 Wiederhold, G. *File Organization for Database Design.* New York: McGraw-Hill, 1987.

12.12. TÉRMINOS CLAVE, CUESTIONES DE REPASO Y PROBLEMAS

TÉRMINOS CLAVE

asignación de fichero	directorio de trabajo o actual	pila
asignación de fichero indexado	fichero	registro
asignación de ficheros contigua	fichero de acceso directo o *hash*	ruta del nombre
asignación de ficheros encadenada	fichero indexado	sistema de gestión de ficheros
base de datos	fichero secuencial	tabla de asignación de disco
bloque	fichero secuencial indexado	tabla de asignación de fichero
campo	método de acceso	tabla de bits
campo clave	nodo-i	
directorio	nombre de fichero	

CUESTIONES DE REPASO

12.1. ¿Cuál es la diferencia entre un campo y un registro?

12.2. ¿Cuál es la diferencia entre un fichero y una base de datos?

12.3. ¿Qué es un sistema de gestión de ficheros?

12.4. ¿Qué criterios son importantes a la hora de escoger una organización de ficheros?

12.5. Liste y defina brevemente cinco organizaciones de ficheros.

12.6. ¿Por qué es menor el tiempo de búsqueda medio de un registro en un fichero para un fichero secuencial indexado que para un fichero secuencial?

12.7. ¿Cuáles son las operaciones típicas que se pueden realizar sobre un directorio?

12.8. ¿Cuál es la relación entre una ruta de un fichero y un directorio de trabajo?

12.9. ¿Cuáles son los derechos de acceso típicos que se pueden conceder o denegar a un usuario particular sobre un fichero particular?

12.10. Liste y defina brevemente tres tipos de bloques utilizados.

12.11. Liste y defina brevemente tres métodos de asignación de ficheros.

PROBLEMAS

12.1. Defina:

B = tamaño de bloque

R = tamaño de registro

P = tamaño de puntero de bloque

F = factor de bloques; número esperado de registros dentro de un bloque

Defina una fórmula para F de acuerdo a los tres tipos de bloques dibujados en la Figura 12.6.

12.2. Un esquema para evitar el problema de la preasignación frente al malgasto o falta de contigüidad es asignar porciones de tamaño incremental a medida que el fichero crece. Por ejemplo, comenzando con un tamaño de un bloque y doblando la porción por cada asignación. Considérese un fichero con *n* registros con un factor de bloques *F*. Supóngase que se utiliza un índice simple de un nivel tal como una tabla de asignación de ficheros

a) Obtenga un límite superior del número de entradas en la tabla de asignación de ficheros como una función de *F* y *n*.

b) ¿Cuál es la cantidad máxima de espacio que no se asigna en ningún momento?

12.3. ¿Qué organizaciones de fichero escogería para maximizar la eficiencia en términos de velocidad de acceso, uso de espacio de almacenamiento y facilidad de actualización (añadir/borrar/modificar) cuando los datos:

a) no se actualizan frecuentemente y se acceden frecuentemente en orden aleatorio?

b) se actualizan frecuentemente y se acceden en su totalidad relativamente frecuente?

c) se actualizan frecuentemente y se acceden frecuentemente en orden aleatorio?

12.4. Los directorios se pueden implementar como «ficheros especiales» que sólo se pueden acceder de una forma limitada o como ficheros de datos ordinarios. ¿Cuáles son las ventajas y desventajas de cada opción?

12.5. Algunos sistemas operativos tienen un sistema de ficheros estructurado en forma de árbol, pero limitan la profundidad del árbol hasta un número pequeño de niveles. ¿Qué efecto tiene este límite sobre los usuarios? ¿Cómo simplifica esto el diseño del sistema operativo (si lo hace)?

12.6. Considérese un sistema de ficheros jerárquico en el que el espacio de disco libre se guarda en una lista de espacio libre.

a) Supóngase que se pierde el puntero al espacio libre. ¿Puede el sistema reconstruir la lista de espacio libre?

b) Sugiera un esquema que asegure que el puntero nunca se pierde como resultado de un único fallo de memoria.

12.7. Considérese la organización de un fichero UNIX representado por el nodo-i (Figura 12.13). Asúmase que hay 12 punteros directos a bloque, un puntero indirecto simple, otro doble y otro triple en cada nodo-i. Asúmase también que el tamaño de bloque del sistema y el tamaño de sector del disco son ambos de 8K. Si el puntero a bloque es de 32 bits, con 8 bits utilizados para identificar el disco físico y 24 bits para identificar el bloque físico, contestar a las siguientes preguntas:

a) ¿Cuál es el tamaño máximo de fichero soportado por este sistema?

b) ¿Cuál es el tamaño máximo de partición soportado por este sistema?

c) Asumiendo que sólo se conoce que el nodo-i del fichero está ya en memoria principal, ¿cuántos accesos a disco se requieren para acceder al byte de la posición 13.423.956?