5

ENTRADA/SALIDA

Una de las principales funciones de un sistema operativo es la de controlar todos los dispositivos de E/S (Entrada/Salida). El sistema operativo debe enviar comandos a los dispositivos, atender las interrupciones y gestionar los errores. También debe proporcionar una interfaz entre los dispositivos y el resto del sistema que sea sencilla y fácil de utilizar. Hasta donde sea posible, la interfaz debe ser la misma para todos los dispositivos (independencia del dispositivo). El código de E/S representa una fracción significativamente grande del sistema operativo completo. La forma en la cual el sistema operativo gestiona la E/S es el tema de este capítulo.

Este capítulo está organizado de la siguiente manera. En primer lugar vamos a ver algunos de los principios del hardware de E/S, para después fijarnos en el software de E/S en general. El software de E/S puede estructurarse en capas, cada una de las cuales tiene encomendada una tarea bien definida. Vamos a fijarnos en esas capas para ver qué es lo que hacen y cómo encajan unas con otras.

Siguiendo a esa introducción, vamos a pasar a ver varios dispositivos de E/S en detalle: discos, relojes, teclados y pantallas. Para cada dispositivo vamos a ver tanto su hardware como su software. Finalmente vamos a considerar la gestión de la energía.

5.1 PRINCIPIOS DEL HARDWARE DE E/S

Diferentes personas ven el hardware de E/S de diferentes maneras. Los ingenieros electrónicos lo ven en términos de chips, cables, fuentes de alimentación, motores y todos los demás componentes físicos que componen el hardware. Los programadores lo ven en términos de la interfaz que presenta al software – los comandos que el hardware acepta, las funciones que lleva a cabo y los informes de error que pueden ser devueltos. En este libro nos concierne lo que tenga que ver con la programación de los dispositivos de E/S, no su diseño, su construcción o su mantenimiento, por lo que nuestro interés se restringe a cómo se programa el hardware y no a cómo funciona por dentro. Sin embargo, a menudo la programación de muchos dispositivos de E/S está conectada íntimamente con su operación interna. En las siguientes tres secciones vamos a proporcionar una pequeña base general sobre el hardware de E/S en lo relativo a la programación. Puede verse como una revisión y una expansión del material introductorio de la sección 1.4.

5.1.1 Dispositivos de E/S

En términos generales, los dispositivos de E/S pueden clasificarse en dos categorías: **dispositivos de bloques** y **dispositivos de caracteres**. Un dispositivo de bloques es uno que almacena la información en bloques de tamaño fijo, cada uno con su propia dirección. El tamaño de los bloques varía desde 512 bytes a 32768 bytes. La propiedad esencial de un dispositivo de bloques es que es posible leer o escribir cada bloque independientemente de todos los demás. Los discos son los dispositivos de bloques más comunes.

Si la examinamos más de cerca, la frontera entre los dispositivos que son direccionables por bloques y aquéllos que no lo son, no está nítidamente definida. Todo el mundo está de acuerdo en que un disco es un dispositivo direccionable por bloques, debido a que, sin importar dónde se encuentre posicionado el brazo del disco, siempre es posible situarse sobre otro cilindro y esperar a que el bloque requerido rote hasta pasar por debajo de la cabeza de lectura/escritura. Consideremos ahora una unidad de cinta utilizada para hacer backups del disco. Las cintas contienen una secuencia de bloques. Si en un momento dado deseamos que la unidad de cinta lea el bloque *N*, siempre podemos rebobinar la cinta e ir leyendo hacia delante hasta llegar al bloque *N*. Esta operación es análoga a la de un disco haciendo un posicionamiento, salvo que requiere mucho más tiempo. Hay que tener en cuenta también que no siempre es posible reescribir un bloque en el medio de una cinta. Pero incluso aunque fuera posible utilizar las cintas como dispositivos de bloques de acceso directo, eso sería forzar bastante las cosas: las cintas normalmente no se usan de esa manera.

El otro tipo de dispositivos de E/S es el de los dispositivos de caracteres. Un dispositivo de caracteres proporciona o acepta un flujo de caracteres, sin tener en cuenta ninguna estructura de bloque. No es un dispositivo direccionable y no cuenta con ninguna operación de posicionamiento. Las impresoras, los interfaces de red, los ratones (para señalar en la pantalla), las ratas (para experimentar en el laboratorio de psicología) y la mayoría de los otros dispositivos que no son similares a los discos, pueden ser vistos como dispositivos de caracteres.

Este esquema de clasificación no es perfecto. Algunos dispositivos simplemente no encajan en la clasificación. Por ejemplo, los relojes (*timers*) no son dispositivos direccionables por bloques, ni tampoco generan o aceptan flujos de caracteres. Lo único que hacen es provocar interrupciones a intervalos de tiempo bien definidos. Las pantallas con RAM de vídeo mapeada en memoria tampoco encajan bien en el modelo descrito. Sin embargo, el modelo de los dispositivos de bloques y de caracteres es una base lo suficientemente general para conseguir que una buena parte del software de E/S del sistema operativo sea independiente del dispositivo. Por ejemplo, el sistema de ficheros trata sólo con dispositivos de bloques abstractos, dejando para el software de nivel inferior la parte dependiente del dispositivo.

Las velocidades de los dispositivos de E/S abarcan un rango enormemente amplio, lo que ejerce una considerable presión sobre el software para que consiga responder siempre correctamente a lo largo de varios órdenes de magnitud en las velocidades de transferencia de los datos. La Figura 5-1 muestra las velocidades de transferencia de algunos dispositivos usuales. La mayoría de estos dispositivos tienden a ser cada día más rápidos.

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Figura 5-1. Velocidad de transferencia de datos de algunos dispositivos, redes y buses típicos.

5.1.2 Controladores de dispositivos

Las unidades de E/S constan normalmente de un componente mecánico y un componente electrónico. En muchos casos es posible separar las dos partes para tener un diseño más modular y general. El componente electrónico se denomina **controlador del dispositivo** o **adaptador**. En los ordenadores personales, este componente suele adoptar la forma de una tarjeta de circuito impreso (**tarjeta controladora**) que puede insertarse en una ranura de expansión. El componente mecánico es el dispositivo mismo. Esta organización se muestra en la Figura 1-5.

La tarjeta controladora está provista usualmente de un conector en el cual puede conectarse un cable que va al dispositivo. Muchas controladoras pueden manejar dos, cuatro o incluso ocho dispositivos idénticos. Si la interfaz entre la controladora y el dispositivo es un interfaz estándar, ya sea un estándar ANSI, IEEE o ISO oficial, o un estándar *de facto*, eso permite que cualquier fabricante de hardware pueda manufacturar controladores o dispositivos que se ajusten a esa interfaz. Por ejemplo, muchas compañías de hardware fabrican unidades de disco compatibles con la interfaz IDE o SCSI.

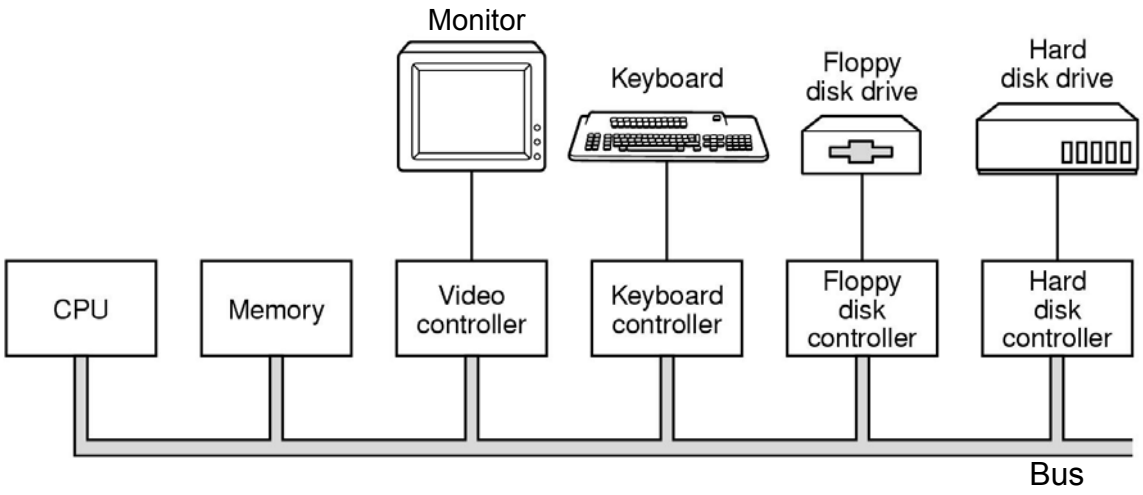

Figura 1-5. Algunos de los componentes de un ordenador personal sencillo.

La interfaz entre el controlador y el dispositivo es a menudo una interfaz de muy bajo nivel. Por ejemplo, un disco puede formatearse con 256 sectores de 512 bytes por pista. Sin embargo, lo que en realidad sale de la unidad es un flujo de bits en serie que comienza por un **preámbulo**, seguido de los 4096 bits de un sector y terminando con una suma de verificación (*checksum*), también llamada un **código de corrección de errores** (**ECC**; *Error-Correcting Code*). El preámbulo se escribe cuando se formatea el disco, y contiene el número de cilindro y de sector, el tamaño del sector y datos similares, así como información de sincronización.

La tarea del controlador consiste en convertir ese flujo de bits en serie en un bloque de bytes y realizar cualquier corrección de errores que sea necesaria. Normalmente primero se ensambla el bloque de bytes, bit a bit, en un búfer que está dentro del controlador. Una vez comprobado su checksum y declarado el bloque libre de errores, puede procederse a copiarlo en la memoria principal.

A un nivel igual de bajo, el controlador de un monitor también opera como un dispositivo de bits en serie: lee de la memoria bytes que contienen los caracteres a visualizar y genera las señales que sirven para modular el haz de electrones del CRT para producir la escritura en la pantalla. El controlador genera también las señales que hacen que el haz del CRT efectúe un retrazado horizontal al terminar cada barrido de una línea, así como las señales que realizan el retrazado vertical una vez que se ha barrido toda la pantalla. Si no fuera por el controlador del CRT, el programador del sistema operativo tendría que programar de forma expícita el barrido analógico del tubo de imagen del monitor. Con el controlador, el sistema operativo inicializa el controlador con unos pocos parámetros, tales como el número de caracteres o píxeles por línea y el número de líneas de la pantalla, y deja que el controlador sea realmente quien se encargue de dirigir el haz del CRT.

5.1.3 E/S mapeada en memoria

Cada controlador tiene unos cuantos registros que le sirven para comunicarse con la CPU. Escribiendo en estos registros, el sistema operativo puede ordenar al dispositivo que suministre datos, acepte datos, se encienda o apague a sí mismo, o realice alguna otra acción. Leyendo de estos registros, el sistema operativo puede averiguar en qué estado se encuentra el dispositivo, si está preparado o no para aceptar un nuevo comando, etc.

Además de los registros de control, muchos dispositivos tienen un búfer de datos que el sistema operativo puede leer y escribir. Por ejemplo, en muchos ordenadores la manera usual de visualizar píxeles en la pantalla es mediante una RAM de vídeo (que es básicamente un búfer de datos) en la que los programas o el sistema operativo pueden escribir.

Dicho lo anterior surge la pregunta de cómo se comunica la CPU con los registros de control y los búferes de datos de los dispositivos. Existen dos alternativas. Con el primer enfoque, a cada registro de control se le asigna un número de **puerto de E/S**, que es un número entero de 8 o 16 bits. Utilizando una instrucción especial de E/S como

```
IN REG,PUERTO
```

la CPU puede leer el registro de control PUERTO y almacenar el resultado en un registro REG de la CPU. Similarmente, con

```
OUT PUERTO,REG
```

La CPU puede escribir el contenido de REG en un registro de control. La mayoría de los primeros ordenadores – incluyendo casi todos los mainframes, como el IBM 360 y todos sus sucesores – funcionaban de esta manera.

En este esquema, los espacios de direcciones para la memoria y para E/S son distintos, como se muestra en la Figura 5-2(a). Las instrucciones

```
IN R0,4
```

y

```
MOV R0,4
```

son completamente distintas en este diseño. La primera lee el contenido del puerto de E/S 4 y lo coloca en R0, mientras que la segunda lee el contenido de la palabra de memoria 4 y lo coloca en R0. Los 4s en estos dos ejemplos se refieren a espacios de direcciones distintos que no tienen relación alguna entre sí.

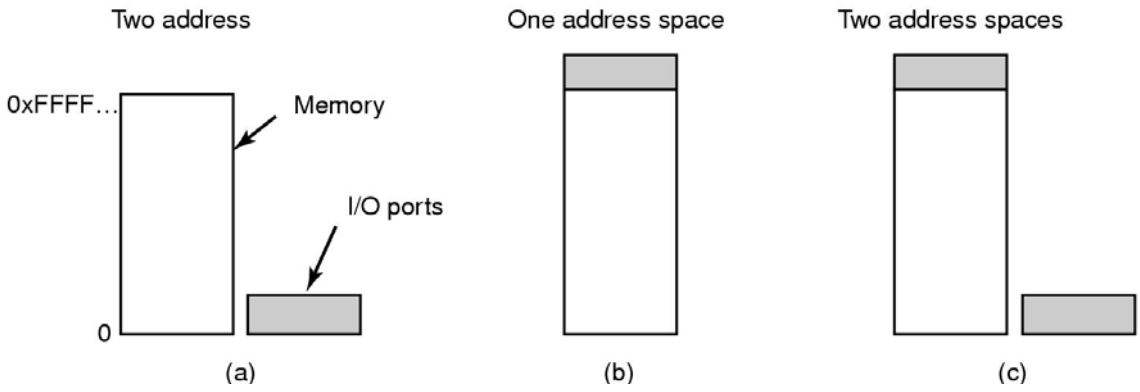

Figura 5-2. (a) Espacios de E/S y de memoria separados.
(b) E/S con mapeada en memoria. (c) Híbrido.

El segundo enfoque, introducido con el miniordenador PDP-11, consiste en mapear todos los registros de control dentro del espacio de memoria, como se muestra en la Figura 5-2(b). A cada registro de control se le asigna una dirección de memoria única a la cual no se asigna memoria. Este sistema se denomina **E/S mapeada en memoria**. Usualmente, las direcciones asignadas a los registros de control están en la parte más alta del espacio de direcciones. En la Figura 5-2(c) se muestra un esquema híbrido, con búferes de datos de E/S mapeados en memoria y puertos de E/S separados para los registros de control. El Pentium utiliza esta arquitectura en los ordenadores compatibles con el IBM PC, reservando las direcciones entre 640K y 1M para los búferes de datos de los dispositivos, además de los puertos de E/S del 0 al 64K.

¿Cómo funcionan estos dos esquemas? En todos los casos, cuando la CPU quiere leer una palabra, sea de la memoria o de un puerto de E/S, coloca la dirección que necesita en las líneas de dirección del bus y luego activa una señal READ sobre una línea de control del bus. Se usa una segunda línea de señal para indicar si la dirección se refiere al espacio de E/S o al de memoria. Si se indica el espacio de memoria, es la memoria quien responde a la solicitud; si se indica el espacio de E/S, es el dispositivo de E/S quien responde a la solicitud. Si sólo hay un espacio de memoria [como en la Figura 5-2(b)], cada módulo de memoria y cada dispositivo de E/S compara las líneas de dirección con el rango de direcciones a las que atiende. Si la dirección está dentro de ese rango, responde a la solicitud. Puesto que nunca se asigna ninguna dirección tanto a la memoria como a un dispositivo de E/S, no hay ambigüedades ni conflictos.

Los dos esquemas para direccionar los controladores tienen diferentes ventajas y desventajas. Comencemos con las ventajas de la E/S mapeada en memoria. Primera ventaja, si se necesitan instrucciones de E/S especiales para leer y escribir en los registros de control del dispositivo, el acceso a ellos requerirá el uso de código en ensamblador, porque no hay manera de ejecutar una instrucción IN o OUT en C o C++. El tener que llamar a procedimientos que realicen estas operaciones (escritos en código ensamblador) representa una sobrecarga adicional para controlar la E/S. En contraste, con la E/S mapeada en memoria, los registros de control del dispositivo no son más que variables en la memoria y pueden direccionarse en C de la misma manera que cualquier otra variable. Por lo tanto, con E/S mapeada en memoria, es posible escribir completamente en C el driver del dispositivo. Sin E/S mapeada en memoria, siempre será necesario escribir algunas partes del driver en código ensamblador.

Segunda ventaja, con E/S mapeada en memoria no se requiere ningún mecanismo de protección especial para evitar que los procesos de usuario realicen directamente la E/S. Lo único que necesita hacer el sistema operativo es cuidar de que la porción del espacio de direcciones que contiene los registros de control nunca se incluya en el espacio de direcciones virtual de ningún usuario. Mejor aún, si cada dispositivo tiene sus registros de control en una página distinta del espacio de direcciones, el sistema operativo podrá dar control a un usuario sobre algunos dispositivos específicos y no sobre otros con sólo incluir las páginas deseadas en su tabla de páginas. Tal esquema permite colocar diferentes drivers de dispositivo en diferentes espacios de direccionamiento, con lo cual no sólo se reduce el tamaño del núcleo sino que también se evita que un controlador interfiera con otros.

Tercera ventaja, con E/S mapeada en memoria cualquier instrucción que pueda hacer referencia a la memoria puede también referenciar registros de control. Por ejemplo, si en el repertorio de instrucciones hay una instrucción, TEST, que comprueba si una palabra de memoria es 0, esa misma intrucción también podrá usarse para determinar si un registro de control es 0, lo que podría indicar que el dispositivo está desocupado y puede aceptar un nuevo comando. El código en lenguaje ensamblador podría tener el aspecto que se muestra a continuación:

```
BUCLE:  TEST PUERTO_4   // comprueba si el puerto 4 es 0
        BEQ LISTO       // si es 0, saltar a LISTO:
        BRANCH BUCLE    // de lo contrario, continuar testeando
LISTO:
```

Si no utilizásemos E/S mapeada en memoria, primero deberíamos leer el registro de control dentro de la CPU mediante una instrucción IN, para luego realizar la comprobación, lo que significa dos instrucciones en lugar de una. En el caso del bucle anterior, la adición de una instrucción más haría un poco más lenta la detección de un dispositivo desocupado.

En el diseño de los ordenadores, prácticamente todo conduce a situaciones de compromiso, y ese es el caso también aquí. La E/S mapeada en memoria también tiene sus desventajas. Primera desventaja, la mayoría de los ordenadores actuales tienen algún tipo de caché de memoria. Cargar en la caché un registro de control de dispositivo sería desastroso. Consideremos la ejecución del bucle en código ensamblador visto anteriormente utilizando una caché. La primera referencia a PUERTO_4 haría que su valor se cargase en la caché. Todas las referencias posteriores a PUERTO_4 simplemente tomarían ese valor de la caché y no se preocuparían por consultar el dispositivo. De esa manera, cuando por fin el dispositivo esté listo, el software no tendrá forma de saberlo. El bucle no terminará nunca.

Para evitar esta situación con E/S mapeada en memoria, el hardware debe contar con la capacidad de deshabilitar de manera selectiva el uso de la caché, por ejemplo, para una página específica. Esto aumenta la complejidad tanto del hardware como del sistema operativo, que tiene que administrar el uso selectivo de la caché.

Segunda desventaja, si sólo hay un espacio de direcciones, todos los módulos de memoria y todos los dispositivos de E/S tienen que examinar todas las referencias a la memoria para determinar a cuáles deben responder. Si el ordenador tiene un único bus, como en la Figura 5-3(a), es sencillo obligar a todo el mundo a examinar cada dirección.

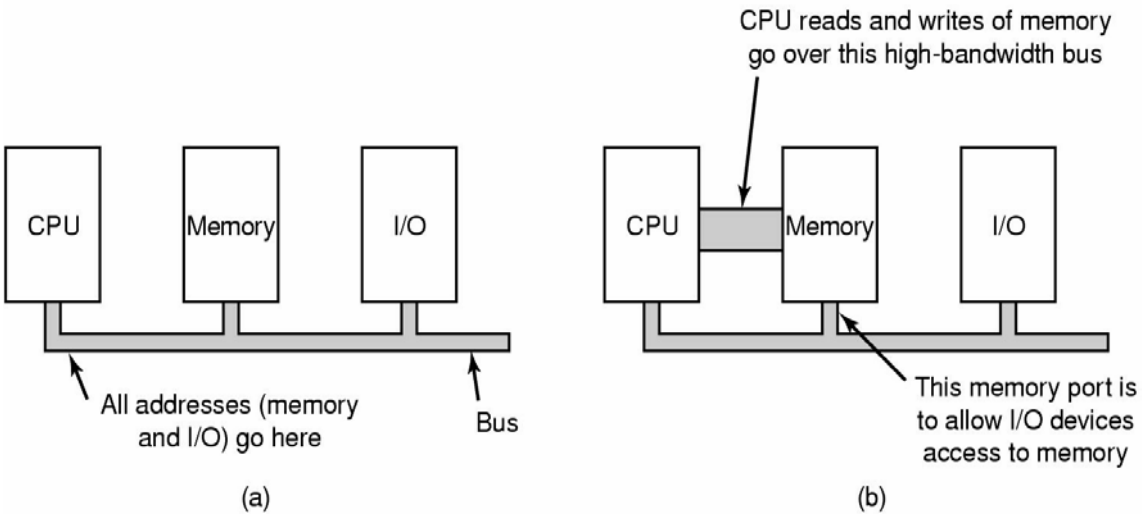

Figura 5-3. (a) Arquitectura de un solo bus. (b) Arquitectura de memoria de bus dual.

Sin embargo, en los ordenadores personales modernos la tendencia es tener un bus de alta velocidad dedicado a la memoria, como se muestra en la Figura 5-3(b), recurso con el que, por cierto, también cuentan los mainframes. Este bus está diseñado para optimizar el rendimiento de la memoria, sin verse afectado por la lentitud de los dispositivos de E/S. Los sistemas Pentium cuentan incluso con tres buses externos (memoria, PCI e ISA), como se muestra en la Figura 1-11.

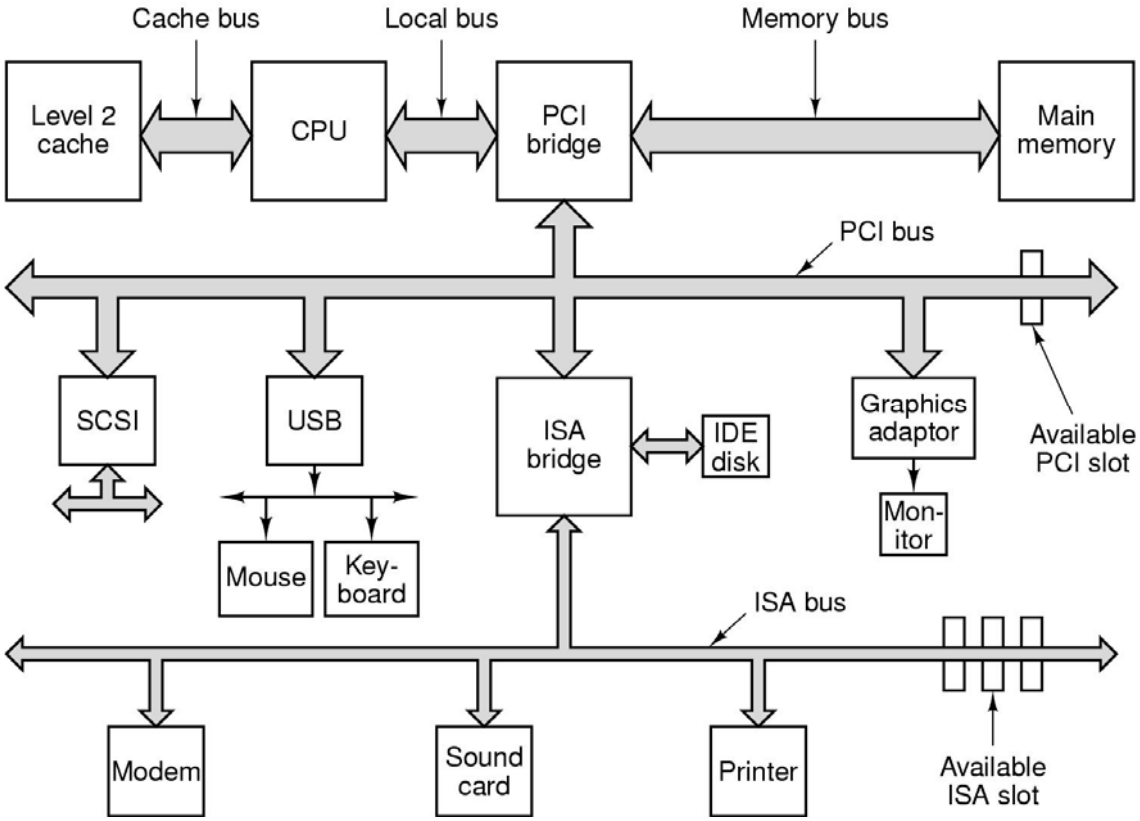

Figura 1-11. Estructura de un sistema Pentium grande.

El problema de tener un bus de memoria aparte en las máquinas con E/S mapeada en memoria es que los dispositivos de E/S no pueden ver las direcciones de memoria a su paso por el bus de memoria, por lo que no tienen la posibilidad de responder. Una vez más, es preciso tomar medidas especiales para lograr que la E/S mapeada en memoria funcione en un sistema con múltiples buses. Una posibilidad es enviar primero todas las referencias a la memoria. Si la memoria no responde a una referencia, la CPU prueba con los otros buses. Este diseño puede funcionar pero requiere un aumento de la complejidad del hardware.

Una segunda posibilidad de diseño es colocar un dispositivo fisgón (*snooping*) en el bus de memoria para que pase todas las direcciones presentadas a dispositivos de E/S potencialmente interesados. El problema aquí es que los dispositivos de E/S seguramente no podrán procesar las peticiones con la misma rapidez que puede hacerlo la memoria.

Una tercera posibilidad, que es la que se usa en la configuración Pentium de la Figura 1-11, es filtrar las direcciones en el chip puente del PCI. Este chip contiene registros de rango que se cargan en el momento del arranque de la máquina. Por ejemplo, el rango de 640K a 1M podría marcarse como *no de memoria*. Las direcciones que caen en alguno de los intervalos marcados como *no de memoria* se envían por el bus PCI, en vez de enviarse a la memoria. La desventaja de este esquema es que durante el arranque de la máquina es preciso determinar qué direcciones de memoria no son realmente tales. Así pues, cada esquema tiene argumentos a favor y en contra, por lo que es inevitable hacer compromisos y concesiones.

5.1.4 Acceso directo a memoria

Independientemente de que tenga o no E/S mapeada en memoria, la CPU necesita direccionar los controladores de dispositivo para intercambiar datos con ellos. La CPU puede solicitar datos del controlador de E/S byte a byte, pero haciéndolo así estaría desperdiciándose mucho tiempo de CPU. Por ese motivo normalmente se utiliza un esquema diferente, denominado **acceso directo a memoria** (**DMA**; *Direct Memory Access*). El sistema operativo sólo puede utilizar DMA si el hardware dispone de un controlador de DMA, por lo que la mayoría de los sistemas cuentan con él. A veces ese controlador está integrado en los controladores de disco o en otros controladores, pero tal diseño significa tener un controlador de DMA por cada dispositivo. Más comúnmente, se tiene un único controlador de DMA (por ejemplo en la placa madre) para regular las transferencias con múltiples dispositivos, a menudo de forma concurrente.

Sea cual sea su ubicación física, el controlador de DMA tiene acceso al bus del sistema independientemente de la CPU, como se muestra en la Figura 5-4. El controlador contiene varios registros en los que la CPU puede leer y escribir. Éstos incluyen un registro de dirección de memoria, un registro contador de bytes y uno o más registros de control. Los registros de control especifican el puerto de E/S que se utilizará, la dirección de la transferencia (leyendo del dispositivo de E/S o escribiendo en el dispositivo de E/S), la unidad de transferencia (un byte a la vez o una palabra a la vez) y el número de bytes que se transferirán en cada ráfaga.

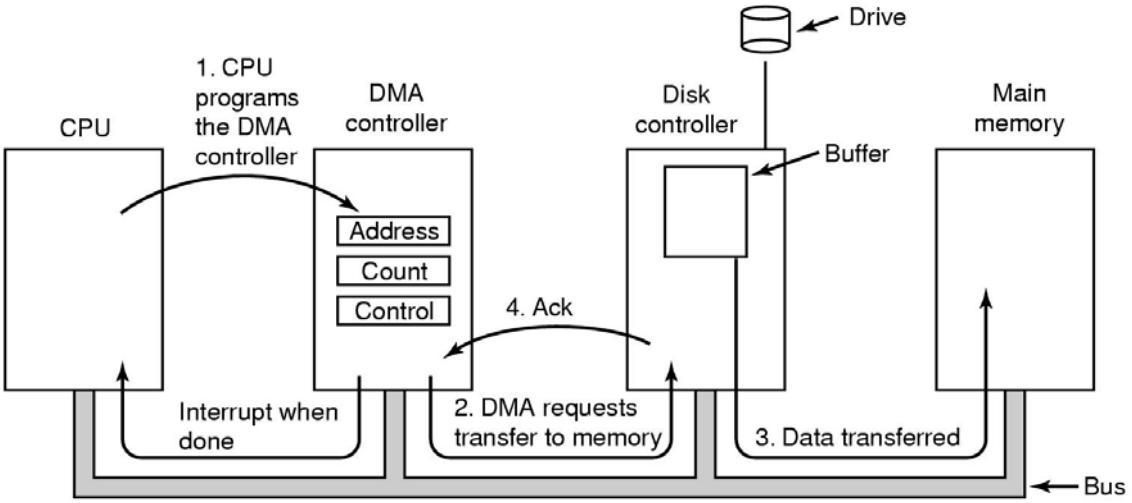

Figura 5-4. Funcionamiento de una transferencia con DMA.

Para explicar cómo funciona el DMA, veamos primero cómo se realizan las lecturas del disco cuando no se utiliza DMA. Primero, el controlador lee el bloque (uno o más sectores) de la unidad, bit a bit en serie, hasta que esté todo el bloque en el búfer interno del controlador. Luego, calcula el checksum para comprobar que no se produjeron errores al leer, y provoca una interrupción. Cuando el sistema operativo comienza a ejecutarse, puede leer ya el bloque de disco del búfer de la controladora byte a byte (o palabra a palabra), ejecutando un bucle, en el que en cada iteración lee un byte (o una palabra) de un registro de control del controlador y lo almacena en la memoria principal.

Cuando se utiliza DMA, el procedimiento es diferente. Primero la CPU programa el controlador de DMA, escribiendo los valores apropiados en sus registros para que sepa qué debe transferir y a dónde debe transferirse (paso 1 en la Figura 5-4). La CPU envía también un

comando al controlador del disco para indicarle que lea los datos del disco en su búfer interno y compruebe el checksum. Cuando haya datos válidos en el búfer del controlador del disco, el DMA puede comenzar.

El controlador de DMA inicia la transferencia enviando por el bus una petición de lectura al controlador de disco (paso 2). Esta petición de lectura es similar a cualquier otra petición de lectura, y el controlador del disco no sabe ni le importa si proviene de la CPU o de una controladora de DMA. Típicamente, la dirección de memoria en la que va a escribirse está ya en las líneas de dirección del bus, así que cuando el controlador del disco toma la siguiente palabra de su búfer interno, sabe donde escribirla. La escritura en memoria es otro ciclo de bus estándar (paso 3). Cuando termina la escritura, el controlador del disco envía una señal de acuse al controlador de DMA, también por el bus (paso 4). Después, el controlador de DMA incrementa la dirección de memoria a utilizar y decrementa el contador de bytes. Si el contador de bytes sigue siendo todavía mayor que 0, se repiten los pasos 2 a 4 hasta que el contador llega a valer 0. En ese momento, el controlador de DMA interrumpe a la CPU para avisarle de que se ha completado la transferencia. Cuando el sistema operativo tome el control, no tendrá que copiar el bloque del disco en la memoria; el bloque ya está ahí.

Los controladores de DMA varían mucho en cuanto a su sofisticación. Los más sencillos realizan una transferencia de cada vez, como acabamos de describir. Otros más complejos pueden programarse para manejar varias transferencias simultáneas. Tales controladores cuentan internamente con múltiples conjuntos de registros, un conjunto por cada canal. La CPU comienza cargando cada conjunto de registros con los parámetros pertinentes para su transferencia. Cada transferencia debe utilizar un controlador de dispositivo diferente. Después de transferir cada palabra (pasos 2 a 4 de la Figura 5-4), el controlador de DMA decide a qué dispositivo atenderá a continuación. Podría configurarse para aplicar un algoritmo de tipo round-robin (asignando turnos de forma circular) o un esquema de prioridades para dar preferencia a algunos dispositivos sobre otros. Es posible tener pendientes al mismo tiempo varias solicitudes a diferentes controladores de dispositivo, supuesto que exista una forma inequívoca de distinguir los diferentes acuses provenientes de los controladores. Por esa razón es frecuente la utilización de una línea de acuse del bus diferente para cada canal de DMA.

Muchos buses pueden operar de dos modos: modo palabra a palabra y modo bloque. Algunos controladores de DMA pueden operar también en cualquiera de los dos modos. En el primer modo el funcionamiento es el que acabamos de describir: el controlador de DMA solicita la transferencia de una palabra y la obtiene. Si la CPU también quiere el bus, tendrá que esperar. Este mecanismo se denomina **robo de ciclo** (*cycle stealing*) porque el controlador del dispositivo de vez en cuando quita furtivamente un ciclo de bus a la CPU, retrasándola ligeramente. En el modo bloque, el controlador de DMA solicita al controlador del dispositivo que adquiera el bus, realice una serie de transferencias y libere el bus. Esta forma de funcionamiento se denomina **modo ráfaga** (*burst mode*). Es más eficiente que el robo de ciclo porque la adquisición del bus requiere cierto tiempo y en el modo ráfaga se transfieren varias palabras al precio de una única adquisición del bus. La desventaja del modo ráfaga es que puede bloquear a la CPU y a otros dispositivos durante un periodo de tiempo considerable si se está transfiriendo una ráfaga muy larga.

En el modelo que hemos estado analizando, conocido a veces como **modo fly-by**, el controlador de DMA pide al controlador de dispositivo que transfiera el dato directamente a la memoria principal. Un modo alternativo que utilizan algunos controladores de DMA consiste en pedir al controlador del dispositivo que envíe la palabra al controlador de DMA, el cual realiza entonces una segunda petición de bus para escribir la palabra en el lugar de destino. Este esquema requiere un ciclo de bus extra por cada palabra transferida, pero es más flexible ya que permite realizar también transferencias de dispositivo a dispositivo e, incluso, de memoria a memoria (realizando primero una solicitud de lectura de la memoria y luego una solicitud de escritura en memoria sobre una dirección diferente).

La mayoría de los controladores de DMA utilizan direcciones físicas de memoria para sus transferencias. Utilizar direcciones físicas requiere que el sistema operativo convierta la dirección virtual del búfer de memoria deseado en una dirección física, y escriba esa dirección física en el registro de dirección del controlador de DMA. Un esquema alternativo que se utiliza en algunos (más bien pocos) controladores de DMA consiste en escribir direcciones virtuales en el controlador. En ese caso el controlador de DMA debe utilizar la MMU para efectuar la traducción de la dirección virtual a física. Sólo tiene sentido poner las direcciones virtuales en el bus si la MMU forma parte de la memoria (lo cual es posible pero poco común), en vez de estar integrada con la CPU.

Mencionamos anteriormente que el disco primero lee los datos en su búfer interno antes de que pueda comenzar el DMA. El lector seguramente se estará preguntando por que motivo el controlador del disco no almacena simplemente los bytes en la memoria principal tan pronto como los recibe del disco. En otras palabras, ¿por qué necesita el controlador tener un búfer interno? Hay dos razones. La primera es que al utilizar un búfer interno, el controlador puede comprobar el *checksum* (suma de verificación) antes de iniciar una transferencia. Si el checksum es incorrecto, se comunicará el error y no se realizará la transferencia.

La segunda razón es que una vez que ha comenzado una transferencia de disco, los bits van llegando inexorablemente del disco a un ritmo constante que no tiene en cuenta si el controlador está listo para recibirlos o no. Si se encomienda al controlador la escritura de los datos directamente en la memoria, dicho controlador necesitará poder utilizar el bus del sistema una vez por cada palabra transferida. Si alguna de esas veces el bus resulta estar ocupado porque otro dispositivo lo esta utilizando (por ejemplo, en modo ráfaga), el controlador tendrá que esperar, y si la siguiente palabra de disco llega antes de que la primera se escriba en la memoria, el controlador tendrá que ponerla en algún lado. Si el bus está muy ocupado, el controlador tendrá que guardar provisionalmente un buen número de palabras y realizar muchas tareas administrativas adicionales. En cambio, si el bloque se coloca en un búfer interno, el bus sólo se necesita cuando comienza el DMA, por lo que el diseño del controlador es mucho más sencillo al no tener que transferir cada palabra en un tiempo crítico. (Algunas controladoras antiguas sí que realizaban la transferencia directamente a la memoria con tan solo un pequeño búfer interno, pero cuando el bus estaba muy ocupado podía llegar a suceder que la transferencia se viera suspendida por un error de desbordamiento del búfer.)

No todos los ordenadores utilizan DMA. El argumento en contra es que la CPU principal a menudo es mucho más rápida que el controlador de DMA y puede realizar el trabajo mucho más rápidamente (cuando el factor limitante no es la rapidez del dispositivo de E/S). Si la CPU (rápida) no tiene otra cosa que hacer, no tiene objeto hacer que espere a que termine el controlador de DMA (lento). Además, prescindir del controlador de DMA y dejar que la CPU realice todo el trabajo por software ahorra dinero, lo cual es importante en los ordenadores de gama baja.

5.1.5 Repaso de las interrupciones

En la sección 1.4.3 dimos una breve introducción sobre el tema de las interrupciones, pero es necesario conocer más cosas sobre ellas. En un ordenador personal típico, la estructura de las interrupciones es como se muestra en la Figura 5-5. A nivel de hardware, las interrupciones funcionan como sigue. Cuando un dispositivo de E/S termina el trabajo que se le encomendó, provoca una interrupción (suponiendo que el sistema operativo ha habilitado las interrupciones). Esto lo hace aplicando una señal a una línea del bus que se le ha asignado. El chip controlador de interrupciones situado en la placa madre detecta esa señal y decide lo que se va a hacer a continuación.

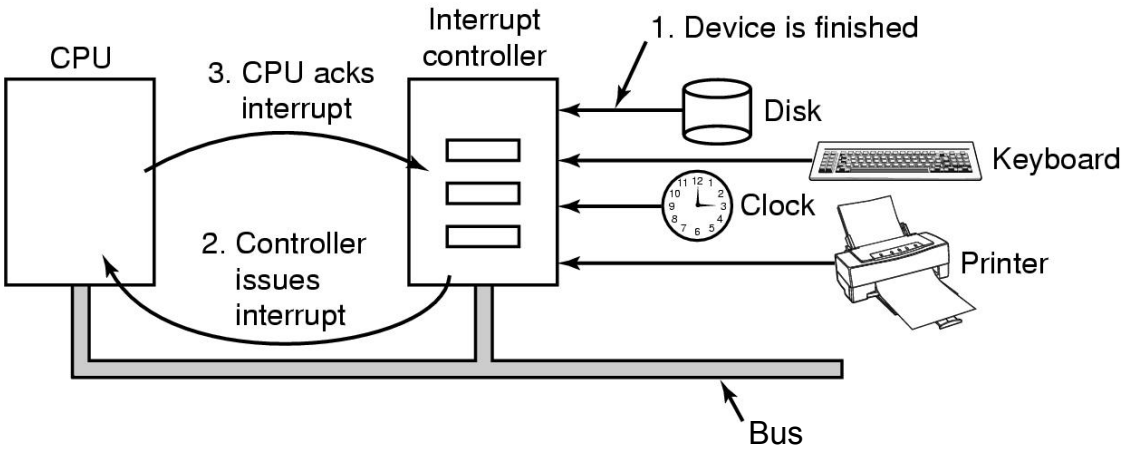

Figura 5-5. Forma en la que se produce una interrupción. En la realidad las conexiones entre los dispositivos y el controlador de interrupciones utilizan líneas del bus en vez de cables dedicados.

Si no hay otras interrupciones pendientes, el controlador de interrupciones procesa la interrupción inmediatamente. Si está atendiéndose alguna otra interrupción en ese momento, o si otro dispositivo ha realizado una petición simultánea sobre una línea de petición de interrupción de mayor prioridad, el primer dispositivo será ignorado momentáneamente. En este caso, el dispositivo seguirá aplicando la señal de interrupción al bus hasta que reciba de la CPU el servicio deseado.

Para gestionar la interrupción, el controlador vuelca un número en las líneas de dirección del bus especificando qué dispositivo requiere atención y aplica una señal que interrumpe a la CPU.

La señal de interrupción provoca que la CPU deje lo que estaba haciendo y comience a hacer alguna otra cosa. El número que está en las líneas de dirección se usa como un índice de una tabla llamada la **tabla de vectores de interrupción** para extraer un nuevo contador de programa, el cual apunta al comienzo del correspondiente procedimiento de servicio de la interrupción. Normalmente tanto los traps, como las interrupciones utilizan el mismo mecanismo a partir es este punto, y a menudo comparten la misma tabla de vectores de interrupción. La ubicación de la tabla de vectores de interrupción puede estar cableada en la máquina o puede estar en cualquier lugar de la memoria, en cuyo caso habrá un registro en la CPU (cargado por el sistema operativo) que apuntará a su comienzo.

Poco después de comenzar a ejecutarse, el procedimiento de servicio de la interrupción efectúa un ciclo de reconocimiento de la interrupción, escribiendo cierto valor en uno de los puertos de E/S del controlador de interrupciones. Este reconocimiento (o acuse de la recepción) de la interrupción le dice al controlador que, habiéndose aceptado ya su petición de interrupción, queda libre para poder solicitar una nueva interrupción. La demora por parte de la CPU del reconocimiento de la interrupción hasta que esté lista para gestionar la siguiente interrupción permite evitar muchas condiciones de carrera involucrando a múltiples interrupciones casi simultáneas. Como comentario, algunos ordenadores (antiguos) no cuentan con un controlador de interrupciones centralizado, por lo que cada controlador de dispositivo debe solicitar directamente sus propias interrupciones.

El hardware siempre guarda cierta información antes de comenzar la ejecución del procedimiento de servicio de la interrupción. La información que se guarda y el lugar donde se guarda varían mucho de unas CPUs a otras. Como mínimo debe guardarse el contador de programa para poder retomar la ejecución del proceso interrumpido. En el otro extremo, podrían guardarse todos los registros visibles y un gran número de registros internos.

El problema es dónde guardar esa información. Una opción sería ponerla en registros internos que el sistema operativo pueda leer cuando lo necesite. Este enfoque tiene la desventaja de que no puede enviarse el acuse de la aceptación de la interrupción al controlador de interrupciones hasta no haber leído toda la información potencialmente relevante almacenada, ya que una segunda interrupción podría sobrescribir los registros internos donde se guarda el estado previo a la interrupción. Esta estrategia provoca largos tiempos muertos, durante los que las interrupciones están inhibidas, con el riesgo de pérdida de nuevas interrupciones y datos.

Consecuentemente la mayoría de las CPUs guardan la información en la pila. Sin embargo, este enfoque también tiene sus problemas. Para empezar: ¿la pila de quién? Si se utiliza la pila actual, posiblemente sería la pila de un proceso de usuario. En ese caso cabe la posibilidad de que el puntero de pila tenga un valor incorrecto, lo cual provocaría un error fatal cuando el hardware intente escribir palabras en ella. Pero incluso siendo correcto el puntero de pila, podría apuntar al final de una página, de manera que tras varias escrituras en la memoria podría atravesarse la frontera de la página, generándose una falta de página. El tener una falta de página mientras el núcleo está procesando una interrupción crea un problema todavía mayor: ¿dónde se guarda el estado actual mientras se resuelve la falta de página?

Si se utiliza la pila del supervisor (la pila del núcleo), aumenta mucho la probabilidad de que el puntero de pila sea válido y apunte a una página que no pueda causar problemas. Sin embargo, el cambio a modo supervisor podría requerir un cambio de contexto de la MMU y probablemente anularía la validez de la mayor parte o la totalidad de la caché y la TLB. La recarga de toda esa información, estática o dinámicamente incrementa el tiempo necesario para procesar una interrupción y por lo tanto desperdicia tiempo de CPU.

Otro problema se debe al hecho de que casi todas las CPUs modernas están altamente segmentadas (tienen un *pipeline* con muchas etapas) y muchas veces son superescalares (paralelas internamente). En los sistemas más antiguos, una vez que terminaba de ejecutarse una instrucción, el microprograma o el hardware comprobaba si había una interrupción pendiente. En tal caso, el contador de programa y la PSW se guardaban en la pila y comenzaba la secuencia de interrupción. Después de ejecutar el manejador de la interrupción, se realizaba el proceso inverso, sacando de la pila la PSW y el contador de programa anteriores y continuando con el proceso interrumpido.

Este modelo supone implícitamente que si se produce una interrupción inmediatamente después de alguna instrucción, eso significa que todas las instrucciones anteriores junto con la última se han ejecutado por completo, y que las instrucciones siguientes no han comenzado a

ejecutarse. En las máquinas más antiguas esa suposición era siempre válida. En las modernas podría no serlo.

Para empezar, consideremos el modelo de pipeline de la Figura 1-6(a). ¿Qué sucede si se presenta una interrupción cuando el pipeline está lleno (que es lo normal)? En ese caso hay varias instrucciones que están en diversas etapas intermedias de su ejecución. Cuando se presenta la interrupción, es muy probable que el valor del contador de programa no refleje la frontera correcta entre las instrucciones ejecutadas y las no ejecutadas. Lo más probable es que refleje la dirección de la siguiente instrucción que se extraerá de la memoria (y se introducirá en el pipeline) en vez de la dirección de la siguiente instrucción que la unidad de ejecución va a procesar.

Como consecuencia, aun en el caso (dudoso) de que exista una frontera bien definida entre las instrucciones que se han ejecutado y las que no lo han hecho, el hardware puede ser incapaz de saber dónde está esa frontera. Por tanto, cuando el sistema operativo deba retornar de una interrupción, no puede simplemente comenzar a llenar el pipeline a partir de la dirección contenida en el contador de programa, sino que debe determinar de alguna manera cuál fue la última instrucción que se ejecutó, lo que supone a menudo realizar una tarea de software compleja como es el análisis del estado de la máquina.

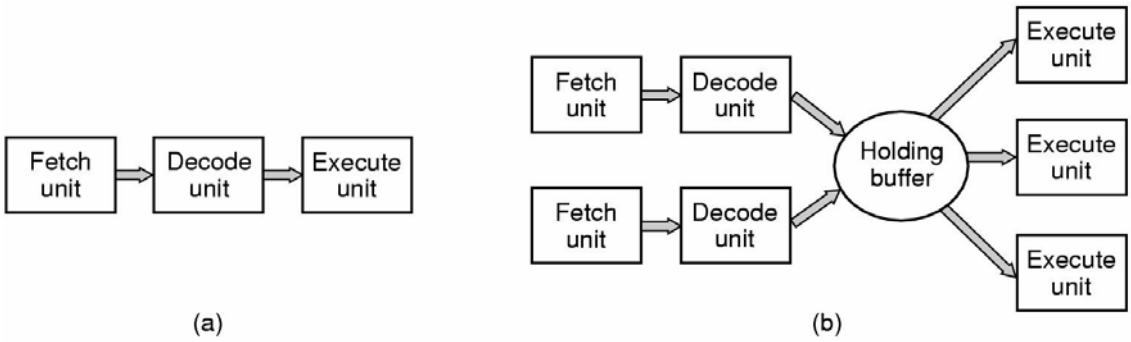

Figura 1-6. (a) Pipeline de tres etapas. (b) CPU superescalar.

Aunque esta situación es mala, las interrupciones en una máquina superescalar, como la de la Figura 1-6(b), son incluso peores. Dado que las instrucciones podrían ejecutarse desordenadas, la probabilidad de que no exista una frontera bien definida entre las instrucciones ejecutadas y las no ejecutadas es mucho mayor. Así podría suceder perfectamente que ya se hubieran ejecutado las instrucciones 1, 2, 3, 5 y 8, pero no todavía las instrucciones 4, 6, 7, 9, 10 y siguientes. Además, puede ser que el contador de programa esté apuntado actualmente a cualquiera de las instrucciones 9, 10 u 11.

Una interrupción que deja a la máquina en un estado bien definido se denomina una **interrupción precisa** (Walter y Cragon, 1995). Una interrupción de ese tipo tiene cuatro propiedades:

1. El contador de programa (PC) se guarda en un lugar conocido.
2. Todas las instrucciones anteriores a aquella a la que apunta el PC se han ejecutado ya por completo.
3. No ha sido ejecutada ninguna instrucción posterior a aquella a la que apunta el PC.
4. Se conoce el estado de ejecución de la instrucción a la que apunta el PC.

En cuanto a la propiedad 3 hay que aclarar que no se prohibe el inicio de la ejecución de instrucciones posteriores a aquella a la que apunta el PC, pero cualquier cambio que produzcan tales instrucciones en los registros o en la memoria deberá deshacerse antes de que la

interrupción ceda el control a su rutina de tratamiento de la interrupción. Se permite que la instrucción a la que apunta el PC haya terminado ya su ejecución, pero también se permite que todavía no lo haya hecho. Sin embargo, debe quedar claro de qué caso se trata. Si la interrupción es de E/S, es frecuente que todavía no se haya iniciado la instrucción; pero si la interrupción es en realidad una excepción o una falta de página, el PC por lo general apunta a la instrucción que provocó el fallo, para poder continuar con su ejecución después.

Una interrupción que no cumple estos requisitos se denomina **una interrupción imprecisa** y complica extremadamente la vida del escritor del sistema operativo, quien ahora no sólo tiene que determinar lo que ha sucedido, sino también lo que no ha sucedido todavía. Las máquinas con interrupciones imprecisas suelen vomitar a la pila una gran cantidad de información sobre el estado interno para que el sistema operativo tenga la posibilidad de dilucidar lo que estaba sucediendo. El tener que guardar una gran cantidad de información en la memoria en cada interrupción hace que las interrupciones sean lentas y peor aún la recuperación tras ellas. Esto conduce a la irónica situación de que a veces las rapidísimas CPUs superescalares no son apropiadas para las aplicaciones en tiempo real debido a la lentitud de las interrupciones.

Algunos ordenadores se diseñan de modo que algunas interrupciones de E/S (y excepciones) sean precisas y otras no. Por ejemplo, si las interrupciones de E/S son precisas, el que las excepciones por errores de programación fatales sean imprecisas no representa ningún problema, ya que en ese caso el proceso que se estaba ejecutando se aborta directamente sin que haya ninguna necesidad de continuar con su ejecución. Algunas máquinas tienen un bit que puede activar el sistema operativo para forzar a que todas las interrupciones sean precisas. El inconveniente de activar ese bit es que fuerza a la CPU a llevar un registro minucioso de todo lo que está haciendo y a mantener copias sombra de los registros de manera que pueda generar una interrupción precisa en cualquier instante. Toda esta sobrecarga de trabajo afecta negativamente al rendimiento de la máquina.

Algunas máquinas superescalares, como el Pentium Pro y todas sus sucesoras, tienen interrupciones precisas para permitir que los programas antiguos escritos para el 386, 486 y Pentium I funcionen correctamente (la superescalaridad se introdujo en el Pentium Pro; el Pentium I sólo tenía dos pipelines). El precio que se paga por tener interrupciones precisas es una lógica de interrupciones extremadamente compleja dentro de la CPU para asegurar que cuando el controlador de interrupciones indique que desea provocar una interrupción, se permita terminar a todas las instrucciones que hayan llegado hasta cierto punto y no se permita a ninguna instrucción posterior tener ningún efecto perceptible sobre el estado de la máquina. Aquí el precio no se paga en tiempo sino en área de chip y en complejidad del diseño. Si no se requiriesen interrupciones precisas para garantizar la compatibilidad hacia atrás, este área del chip podría aprovecharse para hacer más grandes las cachés internas del chip, haciendo así más rápida a la CPU. Por otra parte, las interrupciones imprecisas hacen que el sistema operativo sea mucho más complicado y lento, por lo que es difícil determinar qué enfoque es realmente mejor.

5.2 PRINCIPIOS DEL SOFTWARE DE E/S

Vamos a dejar ahora a un lado el hardware de E/S pasando a echar un vistazo al software de E/S. Trataremos primero los objetivos del software de E/S y luego las distintas formas en las que puede llevarse a cabo la E/S desde el punto de vista del sistema operativo.

5.2.1 Objetivos del software de E/S

Un concepto clave en el diseño del software de E/S es lo que se conoce como la **independencia del dispositivo**, lo que significa que debe ser posible escribir programas capaces de acceder a cualquier dispositivo de E/S sin tener que especificar por adelantado de qué dispositivo se trata. Por ejemplo, un programa que tome su entrada de un fichero debe poder leerlo tanto de un disquete, como de un disco duro, como de un CD-ROM sin tener que modificar el programa para cada dispositivo diferente. Similarmente, debe ser posible que un comando del shell como

```
sort < entrada > salida
```

funcione con la entrada proveniente de un disquete, un disco IDE, un disco SCSI o el teclado, y enviando la salida a cualquier tipo de disco o a la pantalla. Corresponde al sistema operativo resolver los problemas causados por el hecho de que todos esos dispositivos son en realidad diferentes y requieren secuencias de comandos muy distintas para leer o escribir.

El objetivo de **denominación uniforme de ficheros y dispositivos** está estrechamente relacionado con la independencia del dispositivo. El nombre de un fichero o dispositivo debe ser simplemente una cadena de caracteres o un entero y no depender en absoluto del dispositivo. En UNIX, todos los discos pueden integrarse en la jerarquía del sistema de ficheros con total libertad, de manera que el usuario no necesita saber qué nombre corresponde a qué dispositivo. Por ejemplo, un disquete puede **montarse** en el directorio */usr/ast/backup* de manera que copiando cualquier fichero en el directorio */usr/ast/backup/lunes*, estamos realmente copiando dicho fichero en el disquete. De esta manera todos los ficheros y dispositivos se direccionan del mismo modo: mediante su nombre de camino (absoluto o relativo).

Otro aspecto importante del software de E/S es el **manejo de errores**. En general, los errores deben tratarse tan cerca del hardware como sea posible. Si el controlador descubre un error de lectura, él mismo debe tratar de corregirlo en un primer momento. Si no puede, será el driver del dispositivo quien deberá tratar de corregirlo, por ejemplo repitiendo el intento de lectura del bloque. Muchos errores son transitorios, tales como los errores de lectura provocados por una partícula de polvo en la cabeza de lectura, y desaparecen si se repite la operación. Sólo debe informarse del problema a las capas superiores cuando las capas inferiores no puedan solucionar el problema por sí mismas. En muchos casos, la recuperación de los errores puede realizarse de forma transparente en los niveles más bajos, sin necesidad de que los niveles superiores se enteren siquiera de que tuvo lugar el error.

También otro aspecto clave son las transferencias **síncronas** (bloqueantes) frente a las **asíncronas** (dirigidas por interrupciones). Casi toda la E/S física es asíncrona – la CPU pone en marcha la transferencia y se pone a hacer alguna otra cosa hasta que llega la interrupción. Los programas de usuario son mucho más fáciles de escribir si las operaciones de E/S son bloqueantes – después de una llamada al sistema `read`, el programa se suspende

automáticamente hasta que los datos estén disponibles en el búfer. Corresponde al sistema operativo hacer que las operaciones que realmente están controladas por interrupciones parezcan bloqueantes desde la perspectiva de los programas de usuario.

Otra cuestión que corresponde al software de E/S es el **almacenamiento intermedio de los datos** (*buffering*). A menudo los datos provenientes de un dispositivo no pueden almacenarse directamente en su destino final. Por ejemplo, cuando llega un paquete por la red, el sistema operativo no sabe a donde dirigirlo hasta que no lo guarda en algún sitio y lo examina. Además, algunos dispositivos están sujetos a severas restricciones de tiempo real (por ejemplo, los dispositivos de audio digital o las grabadoras de CDs), por lo que los datos deben colocarse primeramente en un búfer de salida para desacoplar la velocidad a la que se llena el búfer con la velocidad a la que se vacía, de manera que el dispositivo de destino nunca se encuentre con el búfer vacío (*buffer underruns*). El uso de búferes requiere numerosas operaciones de copiado y tiene a menudo un importante impacto sobre el rendimiento de la E/S.

El último concepto que mencionaremos aquí es el de dispositivos compartibles frente a dispositivos dedicados. Algunos dispositivos de E/S, como los discos, pueden ser utilizados simultáneamente por muchos usuarios. No hay ningún problema porque varios usuarios tengan ficheros abiertos en el mismo disco simultáneamente. Otros dispositivos, como las unidades de cinta, tienen que estar dedicados a un único usuario hasta que ese usuario termine. Más tarde podrá asignarse la unidad de cinta a otro usuario. Ciertamente no puede funcionar bien que dos o más usuarios escriban bloques entremezclados de forma aleatoria en la misma cinta. La introducción de dispositivos dedicados (no compartidos) introduce también diversos problemas, como los interbloqueos. Una vez más, el sistema operativo debe ser capaz de manejar tanto dispositivos compartidos como dedicados de forma que no se produzcan problemas.

5.2.2 E/S programada

Hay tres formas fundamentalmente distintas de hacer E/S. En esta sección vamos a echar un vistazo a la primera (E/S programada). En las dos secciones siguientes examinaremos las otras (E/S dirigida por interrupciones y E/S utilizando DMA). La forma más sencilla de E/S consiste en dejar que la CPU haga todo el trabajo. Este método se denomina **E/S programada**.

Resulta más sencillo ilustrar la E/S programada mediante un ejemplo. Consideremos un proceso de usuario que desea imprimir la cadena de ocho caracteres “ABCDEFGH” por la impresora. Lo primero que hace es formar la cadena de caracteres en un búfer en el espacio de usuario, como se muestra en la Figura 5-6(a).

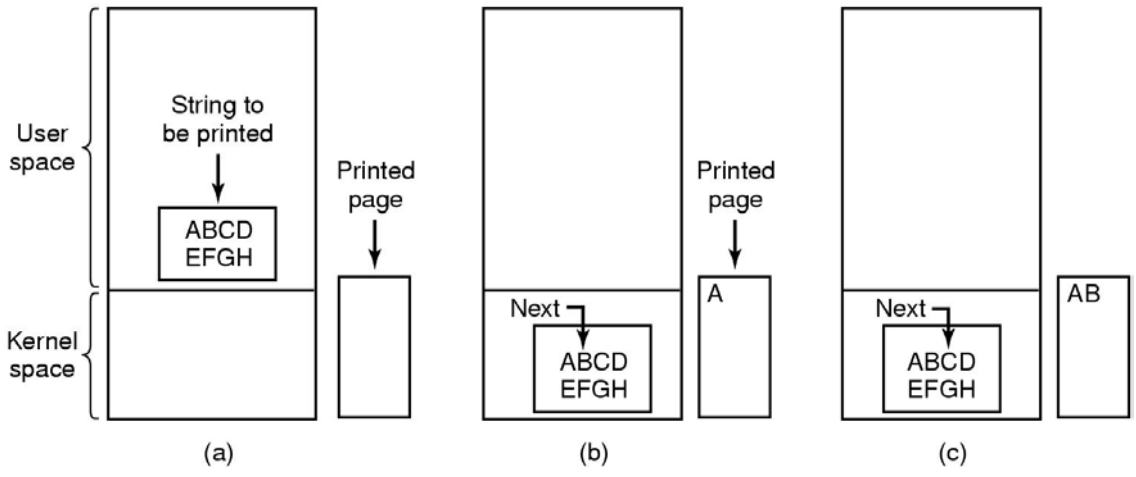

Figura 5-6. Pasos en la impresión de una cadena.

A continuación el proceso de usuario debe solicitar poder utilizar la impresora para escribir, haciendo una llamada al sistema para abrirla. Si actualmente la impresora está siendo utilizada por otro proceso, esta llamada no tendrá éxito y, dependiendo del sistema operativo y de los parámetros de la llamada, devolverá un código de error o se bloqueará hasta que esté disponible la impresora. Una vez que el proceso de usuario consiga la impresora, realizará una llamada al sistema diciéndole al sistema operativo que imprima la cadena.

Seguidamente lo más normal es que el sistema operativo copie el búfer que contiene la cadena en una tabla, llamémosla *p*, en el espacio del núcleo, donde puede acceder más fácilmente a la cadena (porque el núcleo podría tener que cambiar el mapa de memoria para poder acceder al espacio de usuario). Luego comprueba si la impresora está disponible actualmente. Si no lo está, esperará hasta que lo esté. Tan pronto como la impresora esté disponible, el sistema operativo copia el primer carácter en el registro de datos de la impresora, en este ejemplo utilizando E/S mapeada en memoria. Esta acción activa la impresora. Es posible que el carácter no aparezca todavía porque algunas impresoras esperan a que se complete una línea o toda una página antes de imprimir nada, manteniendo en el búfer los caracteres recibidos. Sin embargo en la Figura 5-6(b) vemos que se ha imprimido el primer carácter y que el sistema considera la "B" como el siguiente carácter a imprimir.

Tan pronto como el sistema operativo termina de copiar el primer carácter a la impresora, comprueba si está lista para aceptar el siguiente. Generalmente, la impresora tiene un segundo registro, que informa de su estado. El propio acto de escribir en el registro de datos provoca que el estado sea el de "no preparada". Cuando el controlador de la impresora termina de procesar el carácter actual, indica su disponibilidad activando algún bit de su registro de estado o colocando algún valor especial en él.

En este momento el sistema operativo espera a que la impresora esté lista otra vez. Cuando eso sucede, imprime el siguiente carácter, como se muestra en la Figura 5-6(c). Este ciclo continúa hasta que se imprime toda la cadena. Finalmente se devuelve el control al proceso de usuario.

Las acciones realizadas por el sistema operativo se resumen en la Figura 5-7. Primero se copian los datos en el núcleo. Luego el sistema operativo entra en un bucle enviando los caracteres de uno en uno. El funcionamiento esencial de la E/S programada, que ilustra claramente la figura, es que después de enviar un carácter a la impresora, la CPU muestrea continuamente el estado del dispositivo para ver si está listo para aceptar otro. Este comportamiento se denomina a menudo *polling* o **espera activa** (*busy waiting*).

```
copy_from_user(buffer, p, count);              /* p es el bufer del nucleo */
for (i=0; i < count; i++) {                    /* repetir para cada caracter */
   while (*printer_status_reg != READY) ;      /* repetir hasta que este lista */
   *printer_data_register = p[i];              /* envia el caracter */
}
return_to_user( );
```

Figura 5-7. Escritura de una cadena en la impresora utilizando E/S programada.

La E/S programada es sencilla pero tiene la desventaja de mantener a la CPU ocupada todo el tiempo hasta que termina la E/S. Si el tiempo requerido para "imprimir" un carácter es muy corto (porque lo único que hace la impresora es copiar el nuevo carácter a un búfer interno), la espera activa resulta apropiada. En un sistema empotrado donde la CPU no tiene ninguna otra cosa que hacer, la espera activa resulta igualmente razonable. Sin embargo, en los sistemas más complejos, donde la CPU tiene otras tareas que realizar mientras dura la E/S, la espera activa es ineficiente. Es necesario un método mejor para realizar la E/S.

5.2.3 E/S por interrupciones

Consideremos ahora el caso en el cual tenemos que imprimir la cadena de caracteres utilizando una impresora que no guarda temporalmente los caracteres en un búfer, sino que imprime inmediatamente cada carácter según le llega. Si suponemos que la impresora puede imprimir 100 caracteres/segundo, tenemos que cada carácter se imprime en unos 10 milisegundos. Eso significa que después de escribir un carácter en el registro de datos de la impresora, la CPU permanecerá dando vueltas en un bucle ocioso (que no realiza trabajo útil) durante 10 milisegundos a la espera de que se le permita enviar el siguiente carácter. Ese tiempo es más que suficiente para realizar un cambio de contexto y ejecutar algún otro proceso durante esos 10 milisegundos que se desperdiciarían de otra manera.

La forma de permitir que la CPU haga otra cosa mientras espera a que la impresora esté lista es utilizar interrupciones. Cuando se efectúa la llamada al sistema para imprimir la cadena, el búfer se copia al espacio del núcleo, como vimos antes, y el primer carácter se copia a la impresora tan pronto como esté dispuesta a aceptar uno. En ese momento la CPU llama al planificador y pasa a ejecutarse algún otro proceso. El proceso que pidió que se imprimiera la cadena quedará bloqueado hasta que termine de imprimirse toda la cadena. En la Figura 5-8(a) se muestra el trabajo que se realiza durante la llamada al sistema.

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY);
*printer_data_register = p[0];
scheduler( ) ;
```

(a)

```
if (count == 0) {
  unblock_user( ) ;
} else {
  *printer_data_register = p[i] ;
  count = count – 1;
  i = i + 1;
}
acknowledge_interrupt( ) ;
return_from_interrupt( );
```

(b)

Figura 5-8. Escritura de una cadena de caracteres por la impresora utilizando E/S dirigida por interrupciones. (a) Código ejecutado cuando se realiza la llamada al sistema para imprimir. (b) Rutina de tratamiento de la interrupción.

Cuando la impresora termina de imprimir el carácter y está preparada para aceptar el siguiente, genera una interrupción que detiene el proceso actual y guarda su estado. A continuación se ejecuta la rutina de tratamiento de la interrupción (también llamada a veces procedimiento de servicio de la interrupción o manejador de la interrupción). En la Figura 5-8(b) se muestra una versión poco elaborada de este código. Si ya no hay más caracteres que imprimir, el manejador de la interrupción realiza alguna acción para desbloquear al usuario. En caso contrario, envía a la impresora el siguiente carácter, acusa recibo de la interrupción al controlador y retorna al proceso que se estaba ejecutando justo antes de la interrupción, que continúa desde el mismo punto donde se había quedado.

5.2.4 E/S por DMA

Una desventaja obvia de la E/S controlada por interrupciones es que tiene lugar una interrupción por cada carácter. Las interrupciones ocupan su tiempo, por lo que este esquema desperdicia cierta cantidad de tiempo de CPU. Una solución es utilizar DMA. Aquí la idea consiste en dejar que el controlador de DMA envíe los caracteres a la impresora uno a uno, sin que la CPU tenga que intervenir. En esencia, el DMA es E/S programada, sólo que el controlador de DMA es el que realiza todo el trabajo y no la CPU principal. En la Figura 5-9 se presenta un bosquejo del código.

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
sheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Figura 5-9. Impresión de una cadena utilizando DMA. (a) Código que se ejecuta cuando se efectúa la llamada al sistema para imprimir. (b) Rutina de tratamiento de la interrupción.

La gran ventaja del DMA es que se reduce el número de interrupciones, pasando de una interrupción por carácter, a una única interrupción por búfer impreso. Si hay muchos caracteres para imprimir y las interrupciones son lentas, el ahorro de tiempo de CPU puede suponer una mejora considerable del sistema. Por otra parte, el controlador de DMA suele ser mucho más lento que la CPU principal. Si el controlador de DMA no puede operar el dispositivo a su máxima velocidad, o si la CPU de todas maneras no tiene nada que hacer mientras espera la interrupción del DMA, puede ser preferible utilizar E/S dirigida por interrupciones o incluso E/S programada.

5.3 CAPAS DEL SOFTWARE DE E/S

El software de E/S está organizado típicamente en cuatro capas, como se muestra en la Figura 5-10. Cada capa tiene encomendada una función bien definida y ofrece a las capas adyacentes una interfaz igualmente bien definida. La funcionalidad y las interfaces difieren de un sistema a otro, motivo por el cual nuestro análisis siguiente, que examina todas las capas comenzando por la más baja, no es específico de ninguna máquina concreta.

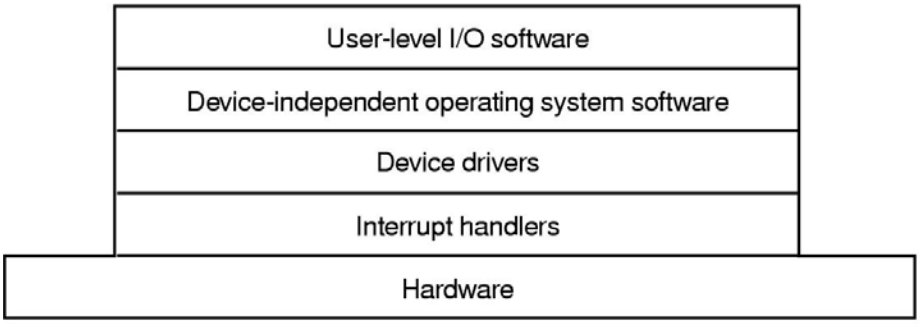

Figura 5-10. Capas del software del sistema de E/S.

5.3.1 Rutinas de tratamiento de las interrupciones

Aunque a veces es útil la E/S programada, en la mayoría de los sistemas de E/S las interrupciones son de esas cosas inevitables que tiene la vida. No obstante es preciso mantener ocultas las interrupciones en las profundidades del sistema operativo, reduciendo al mínimo la parte del sistema operativo que tiene conocimiento de ellas. La mejor manera de ocultar las interrupciones es hacer que el driver que pone en marcha una operación de E/S se bloquee hasta que se complete la E/S y se produzca la interrupción. El driver puede bloquearse a sí mismo por ejemplo ejecutando una operación bajar sobre un semáforo, o un wait sobre una variable de condición o un receive sobre un mensaje, o algo similar.

Cuando llega la interrupción, la rutina de tratamiento hace lo necesario para atender a la interrupción, tras lo cual puede desbloquear el driver que programó esa interrupción. En algunos casos la rutina de tratamiento simplemente ejecutará un subir sobre un semáforo. En otros casos, ejecutará un signal sobre una variable de condición de un monitor. En otros casos diferentes, enviará un mensaje al driver bloqueado. En absolutamente todos los casos, el efecto neto de la interrupción será que un driver que antes estaba bloqueado pasará ya a poder ejecutarse. Este modelo funciona mejor si los drivers se estructuran como procesos del núcleo, con sus propios estados, pilas y contadores de programa.

Por supuesto, la realidad no es tan sencilla. Procesar una interrupción no consiste tan solo en tomar la interrupción, ejecutar un subir sobre algún semáforo y ejecutar una instrucción IRET para retornar de la interrupción al proceso anterior. Se requiere por parte del sistema operativo la realización de mucho más trabajo que vamos a esbozar como una serie de pasos que deben realizarse por software después de que el hardware acepte la interrupción. Los detalles dependen mucho del sistema concreto, por lo que algunos de los pasos siguientes podrían no ser necesarios en una máquina dada, pudiendo ser necesarios otros pasos que no están incluidos en la lista. Además, en algunas máquinas los pasos podrían tener lugar en un orden muy diferente.

1. Guardar los registros (incluida la PSW) que no haya guardado aún el hardware de interrupciones.

2. Establecer el contexto adecuado para la ejecución de la rutina de tratamiento de la interrupción. Esto podría implicar establecer la TLB, la MMU y una tabla de páginas.

3. Disponer una pila para su uso por parte de la rutina de tratamiento de la interrupción.

4. Enviar el acuse de la recepción de la interrupción al controlador de interrupciones. Si no hay un controlador de interrupciones centralizado, volver a habilitar las interrupciones.

5. Copiar los registros de donde se guardaron (posiblemente de alguna pila) a la tabla de procesos.

6. Ejecutar la rutina de tratamiento de la interrupción, la cual necesitará leer la información contenida en los registros del controlador de dispositivo que interrumpió.

7. Escoger el proceso que se ejecutará a continuación. Si la interrupción provocó que algún proceso de alta prioridad que estaba bloqueado pasara a estar listo, podría suceder que ese proceso fuera escogido para ejecutarse ahora.

8. Establecer el contexto de la MMU para el proceso que se ejecutará a continuación. También podría ser necesario preparar la TLB.

9. Cargar los registros del nuevo proceso, incluida su PSW.

10. Comenzar a ejecutar el nuevo proceso.

Como puede apreciarse, el procesamiento de las interrupciones dista mucho de ser trivial y requiere un número considerable de instrucciones de la CPU, sobre todo en las máquinas en las que se utiliza memoria virtual siendo preciso preparar tablas de páginas o guardar el estado de la MMU (por ejemplo los bits *R* y *M*). En algunas máquinas podría ser necesario reajustar también la TLB y la caché de la CPU al conmutar de modo usuario a modo supervisor, lo que puede suponer muchos ciclos de máquina adicionales.

5.3.2 Drivers de dispositivo

Anteriormente en este capítulo hemos echado un vistazo a lo que hacen los controladores de dispositivo. Vimos que cada controlador tiene algunos registros de dispositivo utilizados para enviar comandos al dispositivo o algunos registros de dispositivo utilizados para conocer su estado, o ambos. El número de registros de dispositivo y la naturaleza de los comandos varía radicalmente de un dispositivo a otro. Por ejemplo, un controlador de ratón tiene que aceptar información del ratón indicándole la distancia que se ha desplazado y qué botones están presionados. En contraste, un controlador de disco necesita información sobre los sectores, las pistas, los cilindros, las cabezas, la dirección de movimiento del brazo, los motores, el tiempo de estabilización de las cabezas y todos los demás aspectos mecánicos que se requieren para que funcione correctamente. Obviamente estos dos controladores tienen que ser muy diferentes.

Como consecuencia, cada dispositivo de E/S conectado a un ordenador necesita algún código específico de ese dispositivo que lo controle. Ese código, denominado el **driver del dispositivo**, está normalmente escrito por el fabricante del dispositivo que lo proporciona junto con el dispositivo. Dado que cada sistema operativo necesita sus propios drivers, los fabricantes suelen proporcionar sus drivers para varios de los sistemas operativos más utilizados.

Cada driver de dispositivo maneja normalmente un tipo de dispositivo o, cuando más, una clase de dispositivos estrechamente relacionados. Por ejemplo, usualmente un driver de disco SCSI puede manejar varios discos SCSI de diferentes tamaños y diferentes velocidades, y quizá también un CD-ROM SCSI. Por otra parte, un ratón y un joystick son tan diferentes que casi siempre necesitan drivers diferentes. Sin embargo no existe ninguna restricción técnica que

impida que un driver de dispositivo controle varios dispositivos que no tengan relación entre sí; simplemente resulta que no es una buena idea.

Para acceder al hardware del dispositivo, es decir a los registros del controlador, normalmente es necesario que el driver forme parte del núcleo del sistema operativo, al menos con las arquitecturas actuales. Realmente, es posible construir drivers que se ejecuten en el espacio del usuario, con llamadas al sistema para leer y escribir en los registros del dispositivo. De hecho, ese diseño podría ser una buena idea, ya que conseguiría aislar al núcleo de los drivers y a los drivers entre sí. Haciendo las cosas así podría eliminarse una de las principales causas de caídas del sistema: drivers con errores que interfieren con el núcleo de una manera u otra. Sin embargo, dado que los sistemas operativos actuales esperan que los drivers se ejecuten en el núcleo, ése es el modelo que vamos a considerar aquí.

Puesto que los diseñadores de cualquier sistema operativo saben que se instalarán en él fragmentos de código (drivers) escritos por otras personas, es preciso que utilicen en su diseño una arquitectura que permita tal instalación. Eso significa tener un modelo bien definido de lo que hace un driver y de cómo interactúa con el resto del sistema operativo. Los drivers de dispositivo se ubican comúnmente por debajo del resto del sistema operativo, como se ilustra en la Figura 5-11.

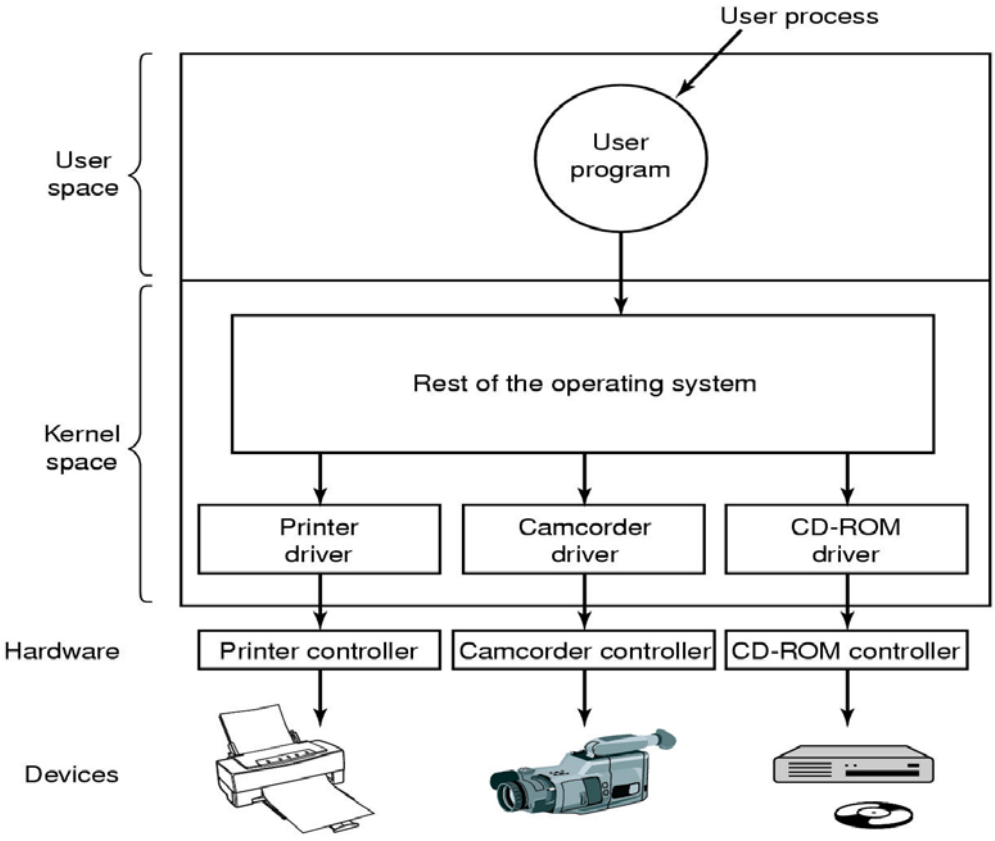

Figura 5-11. Ubicación lógica de los drivers de dispositivo. En realidad, toda la comunicación entre los drivers y los controladores de dispositivo se realiza a través del bus.

Usualmente los sistemas operativos clasifican los drivers en unas cuantas categorías. Las más comunes son los **dispositivos de bloques**, como los discos, que contienen múltiples bloques de datos susceptibles de direccionarse independientemente, y los **dispositivos de caracteres**, como los teclados e impresoras, que generan o aceptan un flujo de caracteres.

La mayoría de los sistemas operativos definen una interfaz estándar que todos los drivers de bloques deben soportar y una segunda interfaz estándar que todos los drivers de caracteres deben soportar. Tales interfaces consisten en varios procedimientos que el resto del sistema operativo puede invocar para pedir al driver que realice algún trabajo. Los procedimientos más comunes son los que leen un bloque (dispositivo de bloques) o los que escriben una cadena de caracteres (dispositivo de caracteres).

En algunos sistemas, el sistema operativo es un único programa binario que contiene todos los drivers que pueda necesitar compilados dentro de él. Este esquema fue la norma durante años en los sistemas UNIX debido a que se ejecutaban en centros de cálculo en los que los dispositivos de E/S raramente cambiaban. Si se añadía un nuevo dispositivo el administrador del sistema simplemente recompilaba el núcleo con el nuevo controlador para obtener un nuevo binario.

Con la llegada de los ordenadores personales, con su miríada de dispositivos de E/S, este modelo dejo de funcionar bien. Pocos usuarios son capaces de recompilar o reenlazar el núcleo incluso si cuentan con el código fuente o con los módulos objeto, lo cual no siempre es el caso. Por ese motivo, comenzando con MS-DOS los sistemas operativos cambiaron a un modelo en el que los drivers se cargan dinámicamente en el sistema durante la ejecución. Cada sistema diferente maneja la carga de los drivers de manera diferente.

Un driver de dispositivo tiene varias funciones. La más obvia es la de aceptar peticiones de lectura o escritura abstractas enviadas por el software independiente del dispositivo y controlar que se lleven a cabo, pero existen también otras funciones que debe realizar. Por ejemplo, el driver debe inicializar el dispositivo, en caso de ser necesario. También puede tener que controlar su consumo de energía eléctrica y mantener un registro de eventos.

Muchos drivers de dispositivo tienen una estructura general similar. Un driver típico comienza comprobando los parámetros de entrada para ver si son válidos. Si no lo son, se devuelve un error. Si los parámetros son válidos, puede ser necesaria una traducción de términos abstractos a concretos. En el caso de un driver de disco, esto puede significar convertir un número de bloque lineal en los números de cabeza, pista, sector y cilindro correspondientes de acuerdo con la geometría del disco.

Luego el driver puede comprobar si el dispositivo está actualmente en uso. En tal caso, la petición deberá encolarse para su procesamiento posterior. Si el dispositivo está desocupado, debe examinarse el estado del hardware para ver si puede atenderse ya la petición. Puede ser necesario encender el dispositivo o poner en marcha un motor antes de comenzar las transferencias. Una vez que está encendido el dispositivo y listo pata trabajar, es cuando comienza el control propiamente dicho del dispositivo.

Controlar el dispositivo significa enviarle una secuencia de comandos. El driver es el lugar donde se determina la secuencia de comandos, dependiendo de la tarea a realizar. Después de que el driver determina qué comandos tiene que enviar, comienza a escribirlos en los registros de dispositivo del controlador. Después de escribir cada comando en el controlador, puede ser necesario comprobar si el controlador ha aceptado el comando y está preparado para aceptar el siguiente. Esto se repite hasta terminar de enviar toda la secuencia de comandos. Algunos controladores son capaces de leer y procesar por si mismos toda una lista enlazada de comandos (en la memoria) sin ninguna ayuda del sistema operativo posterior a la petición de procesamiento de la lista.

Una vez que se envían los comandos pueden darse dos situaciones. En muchos casos el driver del dispositivo debe esperar hasta que el controlador realice algún trabajo, por lo que se bloquea a si mismo hasta que llega la interrupción que lo desbloquea. Sin embargo, en otros casos, la operación termina de inmediato, por lo que el driver del dispositivo no necesita

bloquearse. Un ejemplo de la segunda situación es el desplazamiento vertical (*scroll*) de la pantalla en modo texto, el cual sólo requiere escribir unos cuantos bytes en los registros del controlador. No se requiere ningún movimiento mecánico, por lo que toda la operación puede completarse en nanosegundos.

En el primer caso, el driver bloqueado debe ser despertado por la interrupción. En el segundo caso, el driver nunca se duerme. En ambos casos, una vez que se completa la operación, el driver del dispositivo debe comprobar si se ha producido algún error. Si todo fue bien, el controlador dispone ya de los datos que debe comunicar al software independiente del dispositivo (por ejemplo, un bloque que acaba de leerse). Finalmente, el driver devuelve cierta información de estado a quien lo invocó para informarle de si todo salió bien o de si hubo errores, y cuáles fueron. Si hay más peticiones pendientes en la cola, se selecciona una de ellas y se arranca. Si por el contrario la cola está vacía, el driver se bloquea a la espera de la siguiente petición.

Este sencillo modelo no es más que una burda aproximación a la realidad. Hay muchos factores que hacen que el código sea mucho más complicado. Por ejemplo, un dispositivo de E/S podría terminar mientras un driver está ejecutándose, interrumpiendo por tanto al driver. La interrupción puede provocar que se ejecute un driver de dispositivo. De hecho, podría provocar que se ejecutase nuevamente el driver actual. Por ejemplo, mientras el driver de red está procesando un paquete entrante, podría llegar otro paquete. Consecuentemente, los drivers deben ser **reentrantes**, lo que significa que un driver que está ejecutándose, debe tener prevista la posibilidad de que se le invoque una segunda vez antes de que haya terminado la primera llamada.

En un sistema que permite la conexión en caliente es posible añadir o quitar dispositivos mientras el ordenador está funcionando. Como resultado, puede suceder que mientras un driver está ocupado leyendo de algún dispositivo, el sistema operativo informe de que el usuario ha quitado repentinamente el dispositivo del sistema. En ese caso no solamente es necesario abortar la transferencia de E/S actual sin dañar ninguna estructura de datos del núcleo, sino que deberán eliminarse con sumo cuidado del sistema todas las peticiones pendientes para el dispositivo ahora desaparecido, comunicando la mala noticia a quienes hicieron esas peticiones. Además, la adición inesperada de nuevos dispositivos podría obligar al núcleo a reasignar recursos (por ejemplo, líneas de solicitud de interrupción), quitándole algunos recursos al driver y dándole otros nuevos a cambio.

Los drivers a menudo necesitan interactuar con el resto del núcleo. Aunque desde los drivers no pueden realizarse llamadas al sistema, usualmente sí que se les permite que realicen llamadas a ciertos procedimientos del núcleo. Por ejemplo, usualmente hay llamadas para asignar y liberar páginas de memoria para utilizarlas como búferes. Otras llamadas útiles son necesarias para gestionar la MMU, los timers, el controlador de DMA, el controlador de interrupciones, etc.

5.3.3 Software de E/S independiente del dispositivo

Aunque una parte del software de E/S es específica para los dispositivos concretos existentes en el sistema, otras partes son independientes del dispositivo. La frontera exacta entre los drivers y el software independiente del dispositivo depende del sistema (y del dispositivo), porque algunas funciones que podrían realizarse con independencia del dispositivo en realidad se llevan a cabo en los drivers por cuestiones de eficiencia u otras razones. Las funciones que se muestran en la Figura 5-12 se realizan típicamente en el software independiente del dispositivo.

Interfaz uniforme con los drivers de los dispositivos
Buffering
Informar de los errores
Asignación y liberación de dispositivos dedicados
Proporcionar un tamaño de bloque independiente del dispositivo

Figura 5-12. Funciones del software de E/S independiente del dispositivo.

La función básica del software independiente del dispositivo es realizar las operaciones de E/S que son comunes a todos los dispositivos y presentar una interfaz uniforme al software a nivel de usuario. A continuación examinaremos los aspectos anteriores con más detalle.

Interfaz uniforme con los drivers de los dispositivos

Una cuestión principal en un sistema operativo es cómo conseguir que todos los dispositivos de E/S y sus drivers tengan un aspecto más o menos similar. Si la interfaz con los discos, impresoras, teclados, etc., es muy diferente para cada caso, cada vez que se añada un nuevo dispositivo al sistema será preciso modificar el sistema operativo para ese nuevo dispositivo. No es una buena idea el que haya que hacer este tipo de retoques técnicos en el sistema operativo cada vez que se añada un nuevo dispositivo.

Un aspecto de esta cuestión es la interfaz entre los drivers de dispositivo y el resto del sistema operativo. En la Figura 5-13(a) se ilustra una situación en la que cada driver de dispositivo tiene una interfaz diferente con el sistema operativo, lo que significa que las funciones del driver que el sistema puede invocar difieren de un driver a otro. También podría significar que las funciones del núcleo que necesita el driver difieren también de un controlador a otro. En conjunto, todo esto significa que la interfaz con cada nuevo driver requiere un elevado nuevo esfuerzo de programación.

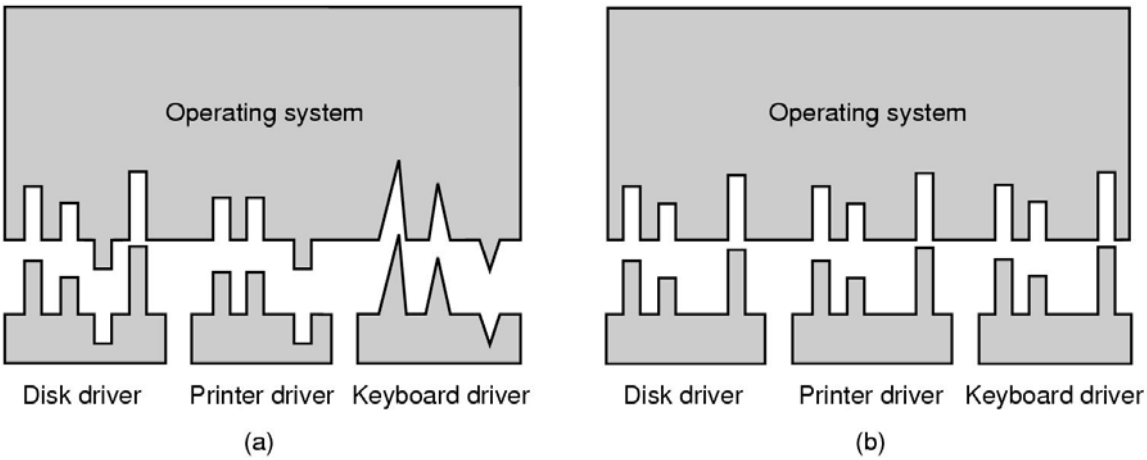

Figura 5-13. (a) Sin una interfaz estándar con los drivers.
(b) Con una interfaz estándar con los drivers.

En contraste, en la Figura 5-13(b) se muestra un diseño diferente en el cual todos los drivers tienen la misma interfaz. En este caso resulta mucho más fácil añadir un nuevo driver,

siempre y cuando se ajuste a la interfaz con los drivers existente. Además en este caso los escritores de drivers conocen perfectamente lo que se espera de ellos (es decir, qué funciones debe proporcionar el driver y qué funciones del núcleo puede invocar el driver). En la práctica, no todos los dispositivos son absolutamente idénticos, pero usualmente sólo hay un pequeño número de tipos de dispositivos e incluso esos tipos son generalmente casi el mismo. Por ejemplo, incluso los dispositivos de bloques y de caracteres tienen muchas funciones en común.

Otro aspecto de tener una interfaz uniforme es la forma en la que se nombran los dispositivos de E/S. El software independiente del dispositivo se encarga de mapear los nombres simbólicos de los dispositivos sobre el driver correcto. Por ejemplo, en UNIX un nombre de dispositivo, como */dev/disk0*, especifica unívocamente el i-nodo de un fichero especial, y ese i-nodo contiene el **número de dispositivo mayor**, que se utiliza para localizar el driver apropiado. El i-nodo contiene también el **número de dispositivo menor**, que se pasa como parámetro al driver en orden a especificar la unidad de la que se va a leer o a escribir. Todos los dispositivos tienen número de dispositivo mayor y menor, y el acceso a los drivers se realiza utilizando el número de dispositivo mayor para seleccionar el driver.

La protección es algo que está íntimamente relacionado con la forma de nombrar a los dispositivos. ¿Cómo puede impedir el sistema que los usuarios puedan acceder a dispositivos para los que no tienen derecho a acceder? Tanto en UNIX como en Windows 2000 los dispositivos aparecen en el sistema de ficheros como objetos con nombre, lo que significa que las reglas de protección habituales para los ficheros son también aplicables a los dispositivos de E/S. El administrador del sistema puede establecer así los permisos correctos para cada dispositivo.

Buffering

Por varias razones la utilización de búferes es otra cuestión importante, tanto para los dispositivos de bloques como para los de caracteres. Para entender una de ellas consideremos un proceso que quiere leer datos desde un módem. Una posible estrategia para tratar el flujo de caracteres que llegan es hacer que el proceso de usuario realice una llamada al sistema read y se bloquee a la espera del siguiente carácter. La llegada de cada carácter provoca una interrupción. La rutina de tratamiento de esa interrupción entrega el carácter al proceso de usuario, desbloqueándolo a continuación. Después de dejar el carácter en algún lado, el proceso lee otro carácter y se bloquea de nuevo. Este modelo se ilustra en la Figura 5-14(a).

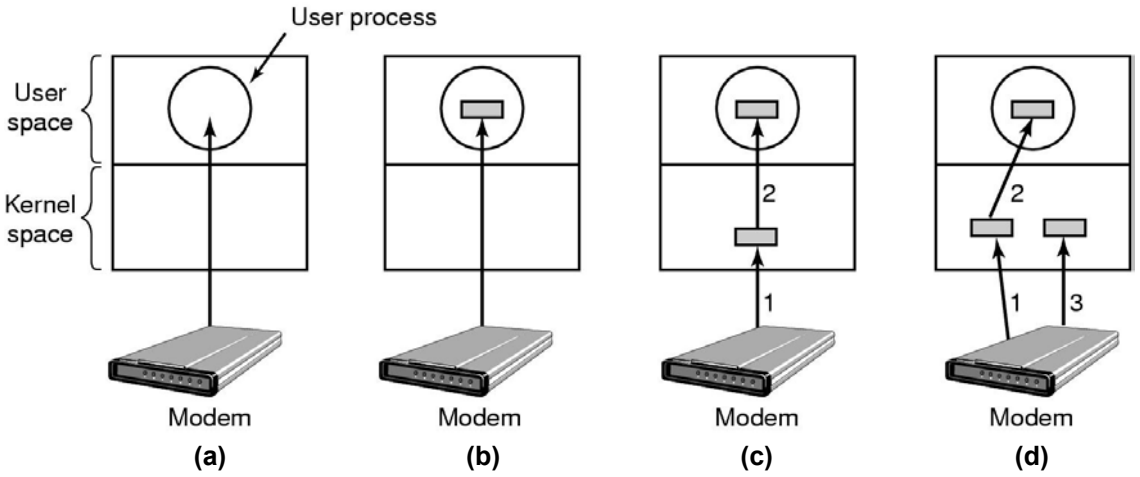

Figura 5-14. (a) Entrada sin búfer. (b) Buffering en el espacio del usuario. (c) Buffering en el núcleo seguido de copia en el espacio de usuario. (d) Doble buffering en el núcleo.

El problema con esa forma de hacer negocios es que el proceso de usuario tiene que ponerse en marcha y bloquearse por cada carácter que llega. Poner en marcha un proceso muchas veces durante cortos lapsos de tiempo resulta ineficiente, por lo que este no es un buen diseño.

En la Figura 5-14(b) se ilustra una mejora que consiste en que el proceso de usuario proporciona un búfer de *n* caracteres en el espacio de usuario y realiza una lectura de *n* caracteres. La rutina de tratamiento de la interrupción deja los caracteres que llegan en ese búfer hasta que se llena, momento en el que despierta al proceso de usuario. Este esquema es mucho más eficiente que el anterior, pero también tiene un inconveniente: ¿qué sucede si en el momento en que llega un carácter las páginas donde está el búfer se encuentran en el disco al haber sido víctimas de los algoritmos de sustitución? Podríamos bloquear las páginas del búfer en la memoria, pero si muchos procesos comienzan a bloquear páginas en la memoria, tendremos una reducción en la reserva de páginas disponibles con una consiguiente degradación del rendimiento del sistema.

Un tercer enfoque consiste en crear un búfer dentro del núcleo y hacer que el manejador de la interrupción deje allí los caracteres, como se muestra en la Figura 5-14(c). Cuando el búfer se llena, se carga del disco la página que contiene el búfer del usuario, si es necesario, y el búfer del núcleo se copia allí en una única operación. Este esquema es mucho más eficiente.

Sin embargo, incluso este esquema tiene un problema: ¿qué sucede con los caracteres que llegan mientras se está cargando del disco la página que contiene el búfer del usuario? Puesto que el búfer está lleno, no tenemos ningún sitio dónde dejarlos. Una salida es tener un segundo búfer en el núcleo. Después de que se llene el primer búfer, y mientras se copia al espacio del usuario, se utiliza el segundo búfer, como se muestra en la Figura 5-14(d). Cuando se llene el segundo búfer, podrá copiarse en el búfer de usuario (suponiendo que el usuario lo pidió). Mientras el segundo búfer se está copiando en el espacio de usuario, el primero puede utilizarse para guardar los nuevos caracteres. De esta manera los dos búferes van alternándose: mientras uno se está copiando en el espacio de usuario, el otro está acumulando las nuevas entradas. Este esquema de utilización de búferes se denomina **doble buffering**.

Los búferes son también importantes en las operaciones de salida. Consideremos, por ejemplo, la forma en la que realiza la salida de datos hacia el módem siguiendo el modelo de la Figura 5-14(b). El proceso de usuario ejecuta una llamada al sistema write para enviar a la salida *n* caracteres. En ese momento, el sistema tiene dos opciones. Puede bloquear al usuario hasta que todos los caracteres se hayan escrito, pero eso podría significar mucho tiempo sobre una lenta línea de teléfono. Por otro lado, el sistema también podría liberar al usuario inmediatamente y realizar la E/S mientras el proceso de usuario sigue con sus cálculos, pero esto nos conduce a un problema todavía peor: ¿cómo puede saber el proceso de usuario cuándo termina la operación de salida y cuándo puede volver a utilizar el búfer? El sistema podría generar una señal o una interrupción software, pero ese estilo de programación es difícil y muy propenso a sufrir condiciones de carrera. Una solución mucho mejor es que el núcleo copie los datos en un búfer del núcleo, de forma análoga a como se hace en la Figura 5-14(c) (pero en el otro sentido) y desbloquee inmediatamente al proceso. Ahora no importa cuándo termina la E/S real; el usuario puede volver a utilizar el búfer en el instante mismo en que se le desbloquea.

El uso de búferes es una técnica extensamente utilizada, pero que también tiene sus desventajas. Si los datos van pasando sucesivamente por demasiados búferes, se produce una merma en el rendimiento. Consideremos por ejemplo la red de la Figura 5-15 y supongamos que un usuario realiza una llamada al sistema para escribir en la red. El núcleo copia el paquete a un búfer del núcleo para que el usuario pueda seguir trabajando de inmediato (paso 1).

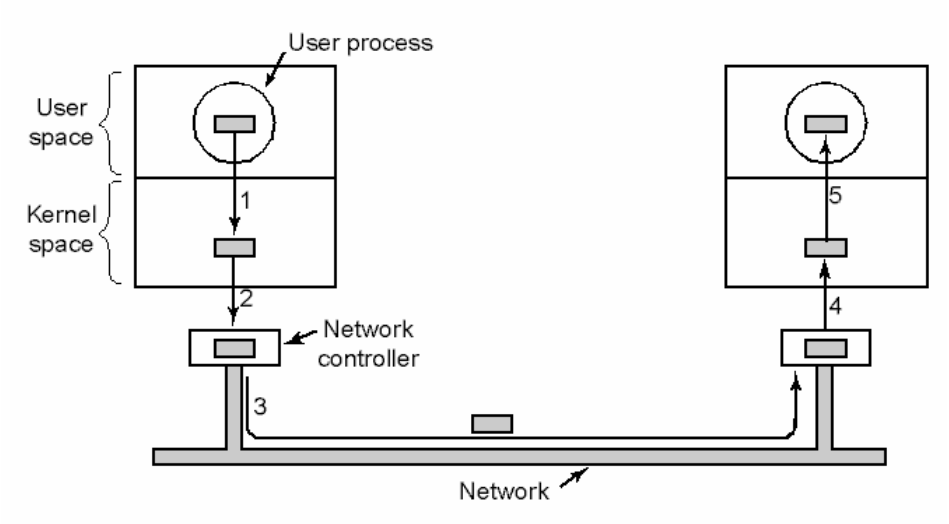

Figura 5-15. Múltiples copias sucesivas de un paquete enviado a través de una red.

Cuando se llama al driver, éste se encarga de copiar el paquete al controlador para proceder a su envío (paso 2). La razón por la cual el driver no envía directamente el paquete a la línea desde la memoria del núcleo es que una vez que se inicia la transmisión de un paquete ésta debe continuar a una velocidad uniforme. El driver no puede garantizar que vaya a conseguir acceder a la memoria a una velocidad uniforme, ya que los canales de DMA y otros dispositivos de E/S podrían estar robando a la CPU muchos ciclos de memoria. Cualquier retraso excesivo en la lectura de una palabra, echará a perder todo el paquete. Este problema se evita almacenando previamente a su transmisión todo el paquete dentro del controlador.

Después de copiar el paquete en el búfer interno del controlador, se copia a través de la red (paso 3). Los bits llegan al receptor poco después de haber sido enviados, así que, casi inmediatamente después de enviarse el último bit, ese bit llega al receptor, donde el paquete queda almacenado en un búfer dentro del controlador. A continuación el paquete se copia al búfer del núcleo del receptor (paso 4). Finalmente, se copia al búfer del proceso receptor (paso 5). Usualmente, el receptor envía entonces un acuse de recibo. Cuando el transmisor recibe el acuse de recibo, puede iniciar la transmisión del siguiente paquete. Sin embargo debe quedar claro que todas esas copias van a reducir considerablemente la velocidad de transmisión porque todos los pasos deben proceder secuencialmente unos después de otros.

Informe de errores

Los errores son mucho más comunes en el contexto de E/S que en otros contextos. Cuando se presentan, el sistema operativo debe solventarlos lo mejor que pueda. Muchos errores son específicos del dispositivo por lo que su tratamiento sólo puede llevarlo a cabo el correspondiente driver. No obstante el marco general del tratamiento de los errores es independiente del dispositivo.

Una clase particular de errores de E/S es la de los errores de programación. Éstos se presentan cuando un proceso solicita algo imposible, como escribir en un dispositivo de entrada (teclado, ratón, escáner, etc.) o leer de un dispositivo de salida (impresora, ploter, etc.). Otros errores frecuentes consisten en proporcionar una dirección de búfer u otro parámetro inválido, o en especificar un dispositivo inválido (por ejemplo, el disco 3 cuando el sistema sólo tiene dos discos). La medida a tomar ante tales errores es sencilla: devolver un código de error al proceso que efectuó la llamada al sistema para hacer E/S.

Otra clase de errores es la clase de los verdaderos errores de E/S, como tratar de escribir en un sector del disco que está dañado o tratar de leer de una cámara de vídeo que acaba de apagarse. En esas circunstancias, corresponde al driver determinar qué es lo que hay que hacer. Si el driver no sabe lo que hacer, tendrá que comunicar el problema al software independiente del dispositivo.

Lo que haga ese software dependerá del entorno y de la naturaleza del error. Si se trata de un simple error de lectura y la operación ha sido solicitada por un usuario interactivo, podría desplegarse una ventana de diálogo para preguntarle qué es lo que desea que se haga en respuesta al error. Las opciones podrían incluir reintentar el acceso cierto número de veces, ignorar el error o terminar el proceso que solicitó la E/S. Si no hay ningún usuario al que poder consultar, probablemente la única opción viable sea dar por fallida la llamada al sistema devolviendo un código de error.

Sin embargo, algunos errores no pueden tratarse de esta manera. Por ejemplo, podría haberse destruido una estructura de datos crucial, como el directorio raíz o la lista de bloques libres, y en tal caso, el sistema tendrá que mostrar un mensaje de error y detenerse completamente para evitar que el daño se propague.

Asignar y liberar dispositivos dedicados

Algunos dispositivos, como las grabadoras de CD-ROM, sólo pueden ser utilizados por un proceso a la vez. Corresponde al sistema operativo examinar las solicitudes de uso de los dispositivos y aceptarlas o rechazarlas, dependiendo de si el dispositivo solicitado está disponible o no. Una forma sencilla de gestionar estas solicitudes es requerir a los procesos que realicen directamente llamadas `open` sobre los ficheros especiales de los dispositivos. Si el dispositivo no está disponible el `open` falla. El cierre del fichero especial libera el dispositivo.

Un enfoque alternativo sería contar con mecanismos especiales para solicitar y liberar dispositivos dedicados. Un intento de adquirir un dispositivo que no está disponible, bloqueará al proceso, en lugar de fallar. Los procesos bloqueados esperan en una cola asociada al dispositivo. Tarde o temprano, el dispositivo solicitado estará disponible y se permitirá que el primer proceso de la cola lo adquiera y continúe con su ejecución.

Tamaño del bloque independiente del dispositivo

Los diferentes discos podrían tener diferentes tamaños de sector. Corresponde al software independiente del dispositivo ocultar estas diferencias y proporcionar un tamaño de bloque uniforme a las capas superiores, por ejemplo, tratando varios sectores como un único bloque lógico. De esta manera, las capas superiores sólo tratarán con dispositivos abstractos, todos los cuales utilizan el mismo tamaño de bloque lógico, independiente del tamaño del sector físico. Similarmente, algunos dispositivos de caracteres suministran sus datos byte a byte (como los módems), mientras que otros suministran los suyos en unidades más grandes (como las interfaces de red). Por supuesto, es posible ocultar también estas diferencias.

5.3.4 Software de E/S en el espacio de usuario

Aunque casi todo el software de E/S está dentro del sistema operativo, una pequeña porción consiste en bibliotecas enlazadas junto con los programas de usuario, e incluso en programas enteros que se ejecutan fuera del núcleo. Los procedimientos de biblioteca normalmente realizan llamadas al sistema, incluyendo las correspondientes a la E/S. Cuando un programa en C contiene la llamada

```
contador = write(fd, buffer, nbytes) ;
```

el procedimiento de biblioteca *write* debe enlazarse con el programa por lo que estará contenido en el programa binario presente en la memoria en tiempo de ejecución. Es obvio que el conjunto de todos estos procedimientos de biblioteca forma parte del sistema de E/S.

Si bien estos procedimientos no hacen mucho más que colocar sus parámetros en el lugar apropiado para la llamada al sistema, hay otros procedimientos de E/S que sí efectúan un trabajo real. En particular, son estos procedimientos de biblioteca quienes realizan el formateo de las entradas y salidas. Un ejemplo en C es *printf*, que toma como entrada un string de formato y posiblemente algunas variables, construye un string ASCII y finalmente llama a write para enviar ese string a la salida. Como un ejemplo de *printf*, consideremos la instrucción

```
printf(" El cuadrado de %3d es %6d\n", i, i*i) ;
```

Esta instrucción formatea un string consistente en el string de 16 caracteres " El cuadrado de " seguido del valor de *i* como un string de 3 caracteres, del string de 4 caracteres " es ", de i^2 como un string de 6 caracteres y de finalmente un carácter de salto de línea.

Un ejemplo de un procedimiento similar para la entrada es *scanf*, que lee la entrada y la almacena en variables descritas en un string de formato que utiliza la misma sintaxis que *printf*. La biblioteca estándar de E/S contiene varios procedimientos que implican E/S, y todos se ejecutan como parte de los programas de usuario.

No todo el software de E/S en el nivel de usuario consiste en procedimientos de biblioteca. Otra categoría importante es el sistema de spooling. El **spooling** es una forma de tratar los dispositivos dedicados en un sistema con multiprogramación. Consideremos un dispositivo que típicamente se gestiona mediante spooling: una impresora. Aunque desde el punto de vista técnico sería fácil permitir que el proceso de usuario abriese el fichero especial de caracteres correspondiente a la impresora, eso tendría el peligro de que un proceso podría abrir el fichero sin hacer nada con él durante horas. Durante ese tiempo ningún otro proceso podría imprimir nada a pesar de que la impresora no está realmente siendo utilizada.

En vez de eso, lo que se hace es crear un proceso especial, llamado **demonio** (*daemon*), y un directorio especial, llamado **directorio de spooling**. Para imprimir un fichero, lo primero que hace un proceso es generar el fichero completo que desea imprimir, colocándolo a continuación en el directorio de spooling. Corresponde al demonio, que es el único proceso autorizado para usar el fichero especial de la impresora, imprimir los ficheros que están en el directorio. Al proteger el fichero especial contra el uso directo por parte de los usuarios, se elimina el problema de que uno de ellos lo mantenga abierto durante un tiempo innecesariamente largo.

El spooling no sólo se utiliza con las impresoras, sino que también se utiliza en otras situaciones. Por ejemplo a menudo se utiliza un demonio de red para transferir ficheros por una red. Si un usuario quiere enviar un fichero a través de la red, debe colocarlo en un directorio de spooling de la red. Más tarde, el demonio de la red cogerá el fichero y lo transmitirá. Un uso particular de la transmisión de ficheros mediante spooling es el sistema de news USENET. Esta

red consta de millones de máquinas de todo el mundo comunicándose a través de Internet. Existen miles de grupos de news sobre numerosos temas. Para publicar un mensaje de news, el usuario invoca un programa de news, el cual acepta el mensaje a publicar y lo deposita en un directorio de spooling para transmitirlo más tarde a otras máquinas. Todo el sistema de news se ejecuta fuera del sistema operativo.

La Figura 5-16 resume el sistema de E/S, mostrando todas las capas y las principales funciones de cada capa. Comenzando por la base, las capas son: el hardware, las rutinas de tratamiento de las interrupciones, los drivers de dispositivo, el software independiente del dispositivo y finalmente, los procesos de usuario.

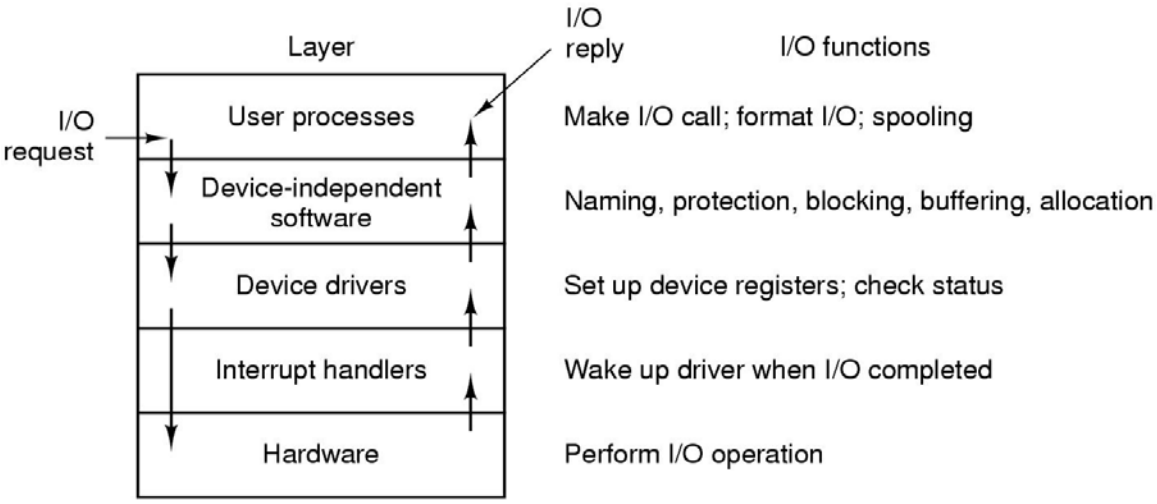

Figura 5-16. Capas del sistema de E/S y funciones principales de cada capa.

Las flechas de la Figura 5-16 muestran el flujo de control. Por ejemplo, cuando un programa de usuario trata de leer un bloque de un fichero se invoca al sistema operativo para que lleve a cabo la llamada. El software independiente del dispositivo busca el bloque en el búfer de la caché, por ejemplo. Si el bloque requerido no está allí, se invoca al driver del dispositivo para que envíe la solicitud al hardware para que obtenga el bloque del disco. Luego el proceso se bloquea hasta que se completa la operación del disco.

Cuando el disco termina, el hardware genera una interrupción. El manejador de la interrupción se ejecuta para descubrir qué es lo que ha sucedido, es decir, qué dispositivo quiere que lo atiendan en ese momento. Luego revisa el estado del dispositivo y despierta al proceso dormido para que finalice la solicitud de E/S y permita que pueda continuar el proceso de usuario.

5.4 DISCOS

Vamos a comenzar a estudiar algunos dispositivos de E/S reales. Empezaremos con los discos para continuar después con los relojes, teclados y pantallas.

5.4.1 Hardware del disco

Existe un enorme variedad de tipos de discos. Los más comunes son los discos magnéticos (discos duros y disquetes), caracterizándose por el hecho de que las lecturas y escrituras son igual de rápidas, lo que los hace ideales como memoria secundaria (paginación, sistemas de ficheros, etc.). A veces se utilizan conjuntamente varios de esos discos para obtener dispositivos de almacenamiento altamente fiables. Son también importantes de cara a la distribución de programas, datos y películas diversos tipos de discos ópticos (CD-ROMs, CD-Regrabables y DVDs). En las siguientes secciones describiremos primero el hardware y luego el software de estos dispositivos.

Discos magnéticos

Los discos magnéticos están organizados en cilindros, cada uno de los cuales contiene tantas pistas como cabezas tenga apiladas verticalmente. Las pistas se dividen en sectores, siendo el número de sectores alrededor de cada circunferencia típicamente de 8 a 32 en los disquetes, y de hasta varios cientos en los discos duros. El número de cabezas varía entre 1 y 16.

Algunos discos magnéticos tienen tan poca electrónica que tan solo pueden suministrar un simple flujo de bits en serie. En estos discos, el controlador realiza la mayor parte del trabajo. En otros discos, en particular en los discos **IDE** (*Integrated Drive Electronics*; Electrónica Integrada en la Unidad), la propia unidad de disco contiene un microcontrolador que realiza una parte del trabajo, permitiendo al controlador real del dispositivo trabajar con un conjunto de comandos de más alto nivel.

Una característica del dispositivo que tiene importantes implicaciones para el driver del disco es la posibilidad de que el controlador realice posicionamientos de las cabezas lectoras de dos o más unidades de disco al mismo tiempo, lo que se denomina **posicionamiento de cabezas solapado** (*overlapped seeks*). Mientras el controlador y el software están esperando a que se complete un posicionamiento en una unidad, el controlador puede iniciar otro posicionamiento en otra unidad. Muchos controladores también pueden leer o escribir en una unidad mientras realizan posicionamientos en otras unidades, pero un controlador de disquete no puede leer o escribir en dos unidades al mismo tiempo. (Leer o escribir requiere que el controlador transfiera bits en una escala de tiempo de microsegundos, por lo que una transferencia requiere casi toda su potencia de cómputo.) La situación es diferente en el caso de los discos duros que tienen controladores integrados, y en un sistema con varios de esos discos duros todos ellos pueden operar simultáneamente, al menos en cuanto a realizar transferencias entre el disco y el búfer del controlador. Sin embargo, sólo puede realizarse una transferencia entre el controlador y la memoria principal a la vez. La capacidad de realizar dos o más operaciones al mismo tiempo puede reducir considerablemente el tiempo de acceso medio.

En la Figura 5-17 se comparan los parámetros del medio de almacenamiento estándar en el PC original de IBM con los parámetros de un disco duro moderno para poner de manifiesto lo mucho que han cambiado los discos en las dos últimas décadas. Es interesante destacar que no todos los parámetros han mejorado en la misma medida. El tiempo de posicionamiento medio es siete veces mejor, la velocidad de transferencia es 1300 veces mejor, mientras que la capacidad ha mejorado en un factor de 50.000. Esta diferencia se debe a que las mejoras en las partes móviles han sido relativamente graduales, mientras que la densidad de bits de las superficies de grabación ha experimentado un aumento muy superior.

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 µsec

Figura 5-17. Parámetros de disco correspondientes al disquete original de 360 KB del IBM PC y al disco duro de Western Digital WD 18300.

Algo que debe tenerse muy presente al estudiar las especificaciones de los discos duros modernos es que la geometría que se especifica, que es la que utiliza el software del driver, podría ser diferente del formato físico. En los discos más antiguos el número de sectores por pista era el mismo para todos los cilindros. Los discos modernos están divididos en zonas, teniendo las zonas exteriores más sectores que las interiores. La Figura 5-18(a) ilustra un disco pequeño con dos zonas. La zona exterior tiene 32 sectores por pista; la interior, 16. Un disco real, como el WD 18300, suele tener 16 zonas, aumentando el número de sectores aproximadamente un 4% de una zona a la siguiente, a medida que nos movemos desde la zona más interna a la más externa.

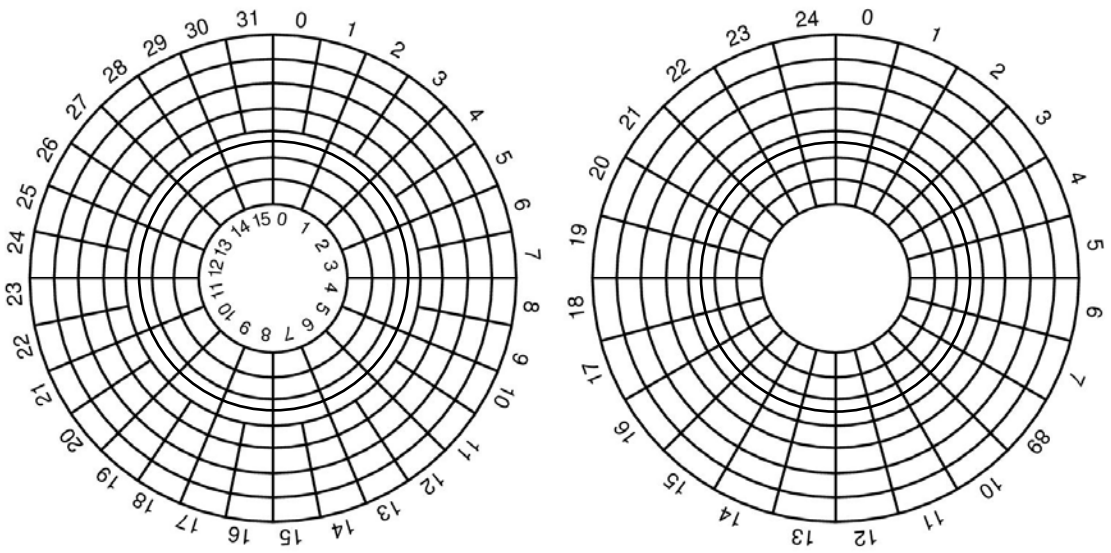

Figura 5-18. (a) Geometría física de un disco con dos zonas. (b) Una posible geometría virtual para este disco.

Con el fin de ocultar los detalles sobre el número de sectores que hay en cada pista, la mayoría de los discos modernos tiene una geometría virtual que se presenta al sistema operativo. El software actúa como si hubiera *x* cilindros, *y* cabezas y *z* sectores por pista. Luego, el controlador establece la correspondencia entre una petición del sector (*x*, *y*, *z*) y el cilindro, cabeza y sector reales. En la Figura 5-18(b) se muestra una posible geometría virtual para el disco físico de la Figura 5-18(a). En ambos casos el disco tiene 192 sectores (4×32+4×16 y 8×24), sólo que la disposición publicada es diferente de la real.

En los ordenadores basados en el Pentium, los valores máximos para esos tres parámetros suelen ser (65535, 16 y 63), debido a la necesidad de mantener la compatibilidad hacia atrás con las limitaciones del IBM PC original. Sobre esa máquina se utilizaron campos de 16, 4 y 6 bits respectivamente para especificar dichos números, con cilindros y sectores numerados a partir de 1 y cabezas numeradas a partir de 0. Con estos parámetros y 512 bytes por sector, el disco más grande posible tiene una capacidad de 31,5 GB. Para poder superar ese límite, muchos discos modernos soportan ahora un sistema denominado **direccionamiento de bloque lógico** (LBA), en el cual los sectores del disco se numeran de forma consecutiva a partir de 0, sin tener en cuenta para nada la geometría física del disco.

RAID

El rendimiento de las CPUs ha estado aumentando de manera exponencial durante la última década, duplicándose prácticamente cada 18 meses. No ha sucedido lo mismo con el rendimiento de los discos. En los años setenta, los tiempos de posicionamiento de las cabezas eran de 50 a 100 ms. Ahora dichos tiempos son de poco menos de 10 ms. En la mayoría de las industrias técnológicas (como la automovilística o la de la aviación), una mejora en el rendimiento en un factor de 5 a 10 en dos décadas sería digno de celebrar, pero en la industria de los ordenadores resulta poco menos que vergonzoso. Así, la brecha entre el rendimiento de la CPU y el de los discos se ha ido agrandando con el paso del tiempo.

Como hemos visto, cada vez se está usando más el procesamiento paralelo para acelerar el rendimiento de la CPU. Desde hace años, a muchas personas se les ha ocurrido que la E/S en paralelo también podría ser una buena idea. En su artículo de 1988, Patterson y otros sugirieron seis organizaciones específicas de disco que podrían servir para mejorar el rendimiento de los discos, su fiabilidad o ambas cosas (Patterson y otros, 1988). Estas ideas no tardaron en ser adoptadas por la industria y han dado pie a una nueva clase de dispositivo de E/S denominado **RAID**. Patterson y otros definieron un RAID como un **array redundante de discos baratos** (*Redundant Array of Inexpensive Disks*), pero la industria redefinió la I de RAID de modo que significara “independientes” en lugar de “baratos” (¿tal vez para poder utilizar discos caros?). Puesto que también se necesitaba un villano (como en RISC contra CISC, que también se debe a Patterson), el malo de la película fue el **SLED** (*Single Large Expensive Disk*; un único disco grande y caro).

La idea básica que hay tras el RAID es la de instalar una caja llena de discos junto a un ordenador (típicamente un servidor grande), sustituir la tarjeta controladora de disco por una controladora RAID, copiar los datos al RAID y luego continuar con la operación normal. En otras palabras, un RAID debe verse igual que un SLED a los ojos del sistema operativo, pero tiene un mayor rendimiento y una mayor fiabilidad. Puesto que los discos SCSI tienen buen rendimiento, bajo precio y capacidad para que una única controladora opere hasta 7 unidades (15 en el caso del SCSI ancho), es natural que la mayoría de los RAID consistan de una tarjeta controladora RAID SCSI y una caja de discos SCSI que el sistema operativo ve como un único disco grande. De esta manera, no se requieren cambios en el software para utilizar el RAID, lo cual es un punto importante a tener en cuenta en el momento de la compra para muchos administradores de sistemas.

Los RAID se presentan al software como un único disco, pero además, todos los RAID tienen la propiedad de que los datos se distribuyen entre las unidades de disco para permitir realizar operaciones en paralelo. Patterson y otros definieron varios esquemas diferentes para hacer esto, los cuales se conocen ahora como RAID nivel 0 hasta RAID nivel 5. Existen además unos cuantos niveles más que no trataremos. El término “nivel” no es muy apropiado, ya que no se trata de una jerarquía; simplemente hay seis organizaciones diferentes posibles.

En la Figura 5-19(a) se ilustra el RAID nivel 0. En este caso el disco virtual simulado por el RAID se considera dividido en tiras (*strips*) de k sectores cada una, con los sectores de 0 a $k - 1$ en la tira 0, los sectores de k a $2k - 1$ en la tira 1, y así de forma sucesiva. Para $k = 1$, cada tira es un único sector; para $k = 2$, cada tira tiene dos sectores, etc. La organización RAID de nivel 0 escribe las tiras consecutivas repartiéndolas entre los discos por turno circular, como se muestra en la Figura 5-19(a) para un RAID con cuatro unidades de disco. Esta distribución de los datos entre varias unidades de disco se denomina **grabación en tiras** (*striping*). Por ejemplo, si el software envía un comando para leer un bloque de datos que consta de cuatro tiras consecutivas, comenzando al principio de una tira, el controlador RAID descompondrá este comando en cuatro comandos, uno para cada uno de los cuatro discos, haciéndolos operar en paralelo. Tenemos por tanto E/S paralela sin que el software sea consciente de ello.

El RAID de nivel 0 funciona mejor si las peticiones son grandes; cuanto más grandes, mejor. Si se pide un bloque mayor que el número de unidades de disco multiplicado por el tamaño de la tira, algunas unidades recibirán múltiples peticiones de sectores, por lo que cuando terminen de atender la primera petición, iniciarán la segunda. Corresponde al controlador descomponer la petición inicial y enviar los comandos adecuados a los discos adecuados en el orden correcto, ensamblando luego correctamente los resultados en la memoria. El rendimiento es excelente y la implementación es directa.

El RAID de nivel 0 funciona peor con sistemas operativos que habitualmente piden datos un sector de cada vez. Los resultados son correctos, pero no hay ningún paralelismo y, por lo tanto, tampoco ninguna mejora del rendimiento. Otra desventaja de esta organización es que la fiabilidad es potencialmente peor que la de un SLED. Si un RAID consiste en cuatro discos, cada uno con un tiempo medio de fallo de 20.000 horas, fallará alguna unidad aproximadamente una vez cada 5000 horas perdiéndose irremisiblemente los datos. Un SLED con un tiempo medio de fallo de 20.000 horas sería cuatro veces más fiable. Puesto que no hay redundancia en este diseño, en realidad no se trata de un verdadero RAID.

La siguiente opción, RAID de nivel 1, mostrada en la Figura 5-19(b), es ya un verdadero RAID. Todos los discos están duplicados, de modo que hay cuatro discos primarios y cuatro de respaldo (backup). Al escribir, todas las tiras se escriben dos veces. Al leer, puede utilizarse cualquiera de las copias, distribuyendo la carga entre más unidades de disco. Consecuentemente, el rendimiento de las escrituras no es mejor que con una sola unidad de disco, pero el rendimiento de las lecturas puede ser hasta dos veces mejor. La tolerancia a fallos es excelente: Si una unidad deja de funcionar, simplemente se utiliza la copia en su lugar. La recuperación consiste simplemente en instalar una nueva unidad de disco y copiar en ella la unidad de respaldo entera.

A diferencia de los niveles 0 y 1, que trabajan con tiras de sectores, el RAID de nivel 2 trabaja sobre una base de palabra, o incluso sobre una base de byte. Imaginemos que dividimos cada byte del disco virtual en un par de *nibbles* de 4 bytes cada uno, añadiendo un código de Hamming a cada nibble para formar una palabra de 7 bits, de la cual los bits 1, 2 y 4 son bits de paridad. Imaginemos también que las siete unidades de disco de la Figura 5-19(c) se sincronizan en términos de la posición de sus brazos y de su posición rotacional. Entonces sería posible escribir estas palabras Hamming de 7 bits repartiéndolas entre las siete unidades, un bit por unidad.

El ordenador CM-2 de Thinking Machines utilizaba este esquema, tomando palabras de datos de 32 bits y añadiéndoles 6 bits de paridad para formar una palabra Hamming de 38 bits, más un bit extra para la paridad de las palabras, repartiendo cada palabra entre 39 unidades de disco. La capacidad de transferencia de datos total era enorme, porque en el tiempo normal de escritura de un sector podían escribirse 32 sectores de datos. Además, la pérdida de una unidad no provoca ningún problema grave, porque equivale a perder un bit de cada palabra de 39 bits leída, algo que el código Hamming puede resolver sobre la marcha.

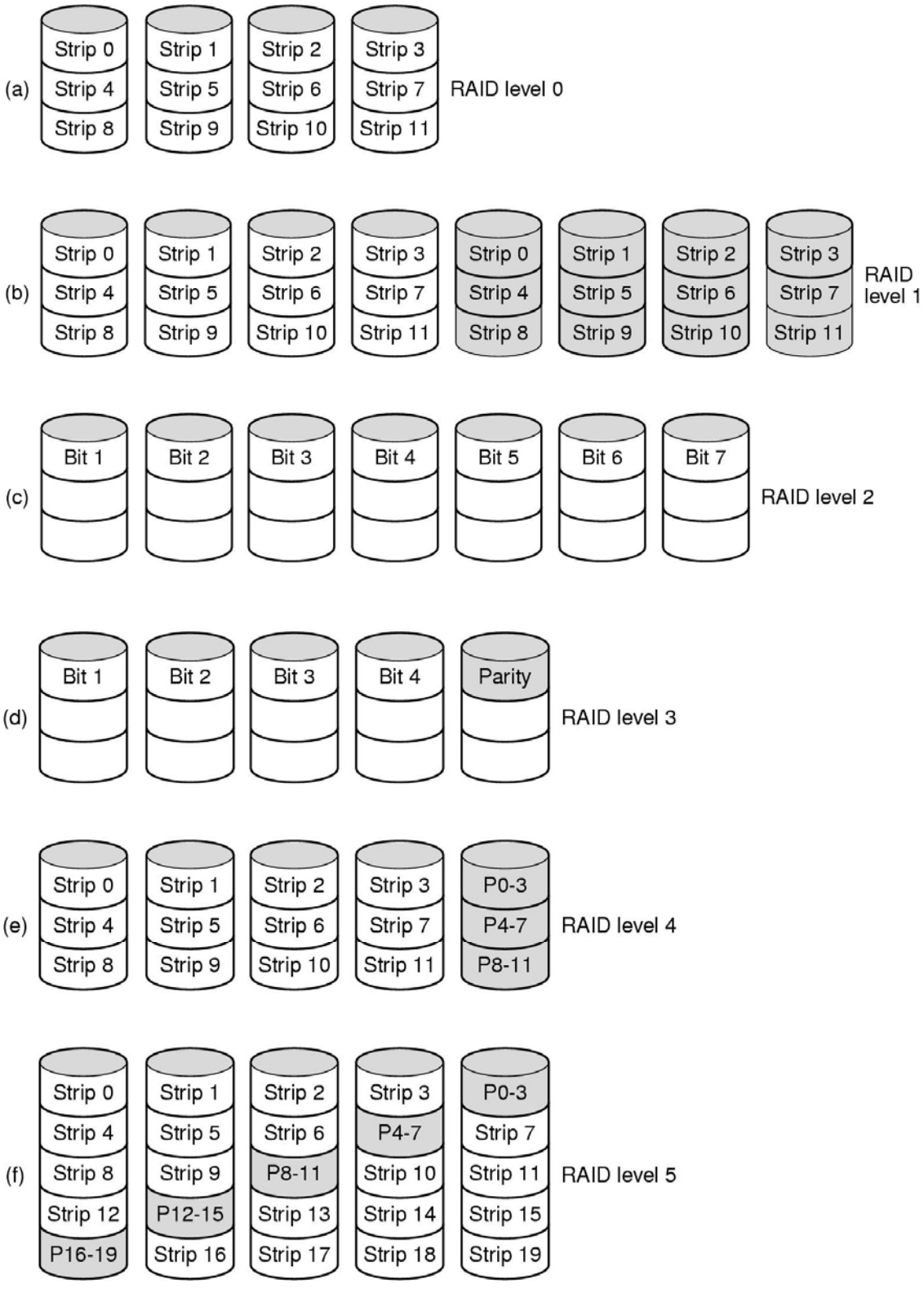

Figura 5-19. Niveles RAID 0 a 5. Las unidades de respaldo y paridad se muestran sombreadas.

En el lado de los inconvenientes, este esquema requiere que todas las unidades de disco estén sincronizadas rotacionalmente, y sólo tiene sentido si el número de unidades es considerable (incluso con 32 unidades de datos y seis unidades de paridad, la sobrecarga adicional es del 19%). También exige mucho de la controladora, que debe calcular una suma de verificación Hamming en el tiempo de lectura normal de cada bit.

El RAID de nivel 3 es una versión simplificada del RAID de nivel 2, y se ilustra en la Figura 5-19(d). Aquí se calcula un único bit de paridad por cada palabra de datos y se escribe en un disco de paridad. Al igual que en RAID nivel 2, las unidades deben estar perfectamente sincronizadas, porque las palabras de datos individuales están repartidas sobre varios discos.

A primera vista, podría parecer que utilizar un único bit de paridad sólo ofrece detección de errores, sin posibilidad de corregirlos. En el caso de errores aleatorios no detectados, esta observación es cierta. Sin embargo, en el caso de que deje de funcionar una unidad de disco, sí que es posible corregir completamente los errores de un bit correspondientes al conocerse la posición del bit erróneo. Si una unidad de disco deja de funcionar, el controlador simplemente hace como si todos sus bits fueran ceros. Si una palabra tiene un error de paridad, eso quiere decir que el bit de la unidad inoperante era realmente un 1, por lo que se corrige. Aunque los niveles RAID 2 y 3, ofrecen tasas de transferencia de datos muy altas, el número de peticiones individuales de E/S que pueden manejar por segundo no es mejor que con una única unidad de disco.

Los RAIDs de nivel 4 y 5 trabajan de nuevo con tiras, no con palabras individuales con paridad, y no requieren unidades de disco sincronizadas. RAID nivel 4 [vea la Figura 5-19(e)] se parece a RAID nivel 0 pero con una paridad tira a tira escrita en una unidad de disco adicional. Por ejemplo, si cada tira tiene una longitud de *k* bytes, se calcula el OR EXCLUSIVO de todas las tiras, y el resultado es una tira de paridad de *k* bytes. Si una unidad de disco deja de funcionar, los bytes perdidos pueden recalcularse a partir de la unidad de paridad.

Este diseño protege contra la pérdida de una unidad de disco, pero tiene un rendimiento muy pobre cuando lo que se realizan son pequeñas actualizaciones. Si se modifica un sector, es necesario leer todas las unidades para poder recalcular la paridad, que tendrá que volver a escribirse. Alternativamente, pueden leerse los datos de usuario antiguos y los datos de paridad viejos y calcular una nueva paridad a partir de ellos. Incluso con esta optimización, una pequeña actualización requiere dos lecturas y dos escrituras.

Como consecuencia de la pesada carga que asume la unidad de paridad, dicha unidad puede convertirse en un verdadero cuello de botella. Este cuello de botella se elimina en el RAID nivel 5 distribuyendo los bits de paridad de manera uniforme entre todas las unidades, por turno circular, como se muestra en la Figura 5-19(f). Sin embargo, en el caso de que un disco deje de funcionar, la reconstrucción de sus contenidos es un proceso complejo.

CD-ROMs

Recientemente han comenzado a estar disponibles los discos ópticos (en oposición a los magnéticos), teniendo densidades de grabación mucho más altas que los discos magnéticos convencionales. Los discos ópticos se desarrollaron originalmente para grabar programas de televisión, pero puede dárseles un uso más estético como dispositivos de almacenamiento para ordenadores. Debido a su capacidad, potencialmente enorme, los discos ópticos han sido el tema de un gran número de investigaciones y han experimentado una evolución increíblemente rápida.

Los discos ópticos de primera generación fueron inventados por el conglomerado de electrónica Philips, de los Países Bajos, para almacenar películas en ellos. Su diámetro era de 30 cm y se vendían con el nombre LaserVision, pero no tuvieron éxito, salvo en Japón.

En 1980, Philips, junto con Sony, desarrolló el CD (*Compact Disc*) que rápidamente sustituyó al disco de vinilo de 33 1/3 rpm para grabar música (excepto entre los conocedores, quienes todavía prefieren el vinilo). Los detalles técnicos precisos del CD se publicaron como un estándar internacional oficial (ISO 10149), conocido popularmente como el **Libro Rojo**, por el color de su portada. Los estándares internacionales son emitidos por la organización internacional para la estándarización, que es la contrapartida internacional de los organismos nacionales de normas como ANSI, DIN, etc. Cada estándar tiene su propio número ISO. Lo que se busca al publicar las especificaciones del disco y de la unidad como un estándar internacional es hacer posible la compatibilidad entre los CDs de diferentes productores de música y los reproductores de diferentes fabricantes de aparatos electrónicos. Todos los CDs tienen un diámetro de 120 mm y un espesor de 1,2 mm, con un agujero de 15 mm en el centro. El CD de audio fue el primer medio de almacenamiento digital que tuvo éxito en el mercado de masas. Se supone que duran 100 años. Estamos esperando que alguien compruebe esto en el año 2080 y nos informe de cómo le ha ido al primer lote de CDs.

Un CD se prepara utilizando un láser infrarrojo de alta potencia para quemar agujeros de 0,8 micras de diámetro en un disco maestro recubierto de vidrio. A partir de ese disco maestro, se fabrica un molde, con pequeñas protuberancias donde estaban los agujeros realizados con el láser. En este molde se inyecta resina de policarbonato fundida que toma la forma de un CD con el mismo patrón de agujeros que el disco maestro de vidrio. Luego se deposita sobre el policarbonato una capa muy fina de aluminio reflectante que se cubre con una laca protectora y finalmente una etiqueta. Las depresiones en el sustrato de policarbonato se denominan **fosos** (*pits*); las áreas no quemadas entre los fosos se denominan **llanos** (*lands*).

Cuando se lee un CD en un reproductor, un diodo láser de baja potencia ilumina los fosos y llanos a medida que el disco gira bajo el láser, utilizando una luz infrarroja con una longitud de onda de 0,78 micras. La luz del láser incide por el lado del policarbonato, de modo que los fosos sobresalen hacia el láser como protuberancias en la superficie por lo demás plana. Puesto que los fosos tienen una altura igual a una cuarta parte de la longitud de onda de la luz del láser, la luz que se refleja de un foso está desfasada media longitud de onda respecto a la que refleja la superficie circundante. Como resultado las dos partes se interfieren de forma destructiva y devuelven menos luz al fotodetector del reproductor que la luz que se refleja de un llano. Así es como el reproductor distingue un foso de un llano. Aunque podría parecer más sencillo utilizar un foso para registrar un 0 y un llano para registrar un 1 (o viceversa), resulta más fiable utilizar una transición foso/llano o llano/foso para un 1 y su ausencia para un 0, por lo que éste es el esquema que se usa.

Los fosos y llanos se graban en una única espiral continua que comienza cerca del agujero y avanza una distancia de 32 mm hacia el borde. La espiral describe 22.188 revoluciones alrededor del disco (aproximadamente 600 por mm). Si se desenrollara, tendría una longitud de 5,6 km. La espiral se ilustra en la Figura 5-20.

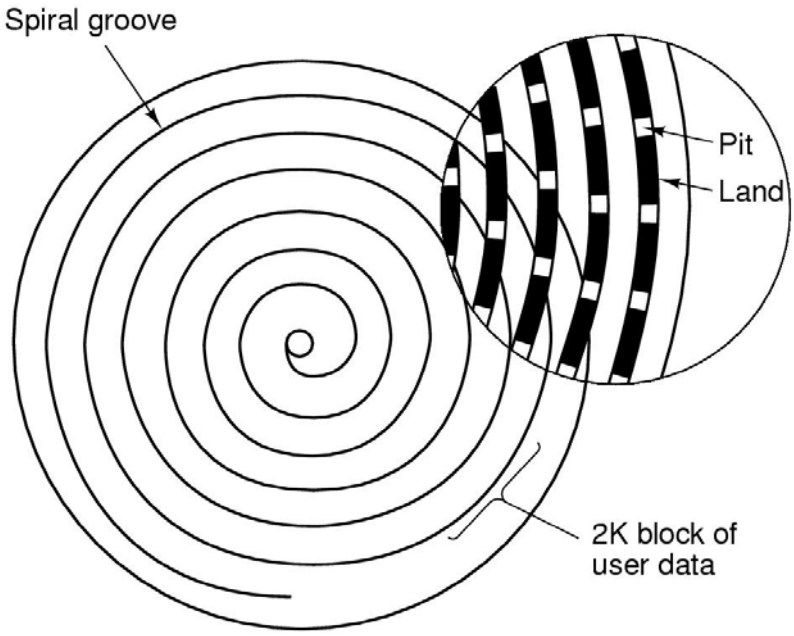

Figura 5-20. Estructura de grabación de un disco compacto o CD-ROM.

Para conseguir que la música se reproduzca a un ritmo uniforme, es necesario que los fosos y llanos fluyan a una velocidad lineal constante. Consecuentemente la velocidad de rotación del CD debe reducirse continuamente a medida que la cabeza lectora se aleja del centro del CD hacia el exterior. En la parte más interna, la velocidad de rotación es de 530 rpm para conseguir una velocidad lineal de 120 cm/s; en la parte más externa la velocidad de rotación debe bajar a 200 rpm para mantener la misma velocidad lineal bajo la cabeza. Una unidad de disco con velocidad lineal constante es muy distinta de una unidad de disco magnético, que opera a una velocidad angular constante, independiente del lugar donde la cabeza esté actualmente posicionada. Además, 530 rpm no es comparable con las velocidades de 3600 a 7200 rpm a las que giran la mayoría de los discos magnéticos.

En 1984, Philips y Sony se percataron de la posibilidad de utilizar los discos compactos para almacenar datos destinados a un ordenador, por lo que publicaron el **Libro Amarillo**, que define un estándar preciso para lo que ahora se conoce como **CD-ROM** (*Compact Disk - Read Only Memory*). A fin de aprovechar el ya entonces considerable mercado de los CDs de audio, los CD-ROMs debían tener el mismo tamaño físico que los CDs de audio, ser mecánica y ópticamente compatibles con ellos, y producirse utilizando las mismas máquinas de estampado por inyección de policarbonato. La consecuencia de esta decisión no sólo fue la necesidad de contar con motores lentos de velocidad variable, sino también que el costo de fabricación de un CD-ROM enseguida se hizo mucho menor de un dólar en volúmenes de producción moderados.

Lo que el Libro Amarillo definió fue el formateo de los datos del ordenador. También se mejoró la capacidad de corrección de errores del sistema, lo que era un paso esencial porque, si bien a los amantes de la música no les importa perder algún que otro bit, los amantes de los ordenadores tienden a ser muy quisquillosos al respecto. El formato básico de un CD-ROM consiste en codificar cada byte con un símbolo de 14 bits. Como vimos antes, son suficientes 14 bits para codificar mediante Hamming un byte de 8 bits, sobrando dos bits. De hecho se emplea un sistema de codificación aún más potente. La transformación de 14 a 8 durante la lectura se realiza por hardware mediante la consulta de unas tablas.

En el siguiente nivel hacia arriba, un grupo de 42 símbolos consecutivos forma una **trama** de 588 bits. Cada trama contiene 192 bits de datos (24 bytes). Los 396 bits restantes se utilizan para corrección de errores y control. Hasta aquí el esquema es idéntico para los CDs de audio y los CD-ROMs.

Lo que el Libro Amarillo añade es el agrupamiento de 98 tramas en un **sector de CD-ROM**, como se muestra en la Figura 5-21. Todo sector de CD-ROM comienza con un preámbulo de 16 bytes, de los cuales los primeros 12 son 00FFFFFFFFFFFFFFFFFF00 (hexadecimal) para hacer posible que el reproductor reconozca el principio de un sector de CD-ROM. Los tres bytes siguientes contienen el número de sector, que se necesita porque el posicionamiento de la cabeza lectora en un CD-ROM con su espiral de datos única es mucho más difícil que un disco magnético con sus pistas concéntricas uniformes. Para posicionarse, el software de la unidad calcula aproximadamente el lugar al que debe ir, mueve la cabeza allí y luego espera hasta detectar un preámbulo que le dirá lo cerca o lejos que está del punto de destino. El último byte del preámbulo es el modo.

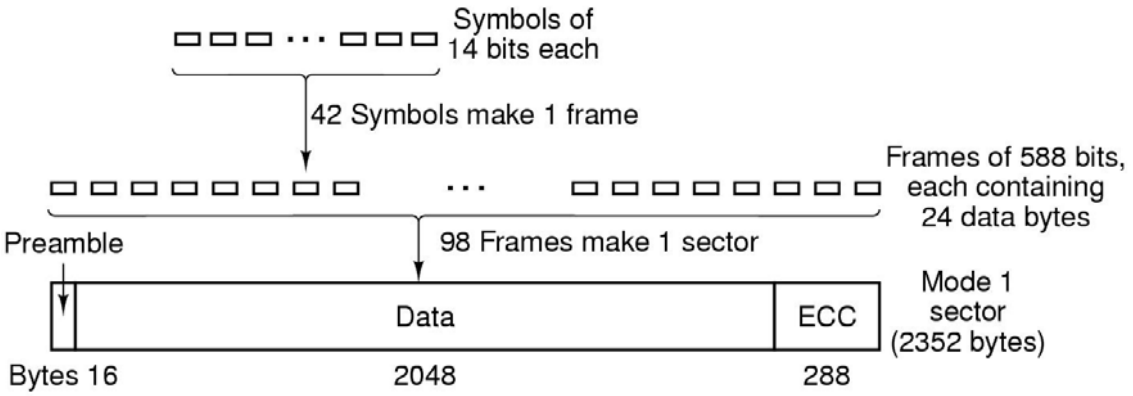

Figura 5-21. Disposición lógica de los datos en un CD-ROM.

El Libro Amarillo define dos modos. El modo 1 utiliza la disposición de la Figura 5-21, con un preámbulo de 16 bytes, 2048 bytes de datos y un código de 288 bytes para corrección de errores (un código Reed-Solomon intercalado). El modo 2 combina los campos de datos y de ECC en un campo de datos de 2336 bytes para aquellas aplicaciones que no necesitan corrección de errores (o no disponen de tiempo para realizarla), tales como audio y vídeo. Cabe señalar que para mantener una excelente fiabilidad, se utilizan tres esquemas separados de corrección de errores: dentro de un símbolo, dentro de una trama y dentro de un sector de CD-ROM. Los errores de un único bit se corrigen en el nivel más bajo; los errores de ráfaga cortos se corrigen en el nivel de trama y los errores residuales se capturan en el nivel de sector. El precio que se paga por esa fiabilidad es que se requieren 98 tramas de 588 bits (7203 bytes) para una carga útil de datos de sólo 2048 bytes; una eficiencia de tan solo el 28%.

Las unidades de CD-ROM de una sola velocidad (1x) operan a 75 sectores/s, lo que significa una tasa de datos de 153.600 bytes/s en el modo 1 y 175.200 bytes/s en el modo 2. Las unidades de doble velocidad (2x) son dos veces más rápidas, y así sucesivamente hasta la velocidad más alta. Una unidad de 40x puede entregar datos a razón de 40 × 153.600 bytes/s, suponiendo que tanto la interfaz de la unidad como el bus y el sistema operativo pueden manejar esa tasa de datos. Un CD de audio estándar tiene espacio para 74 minutos de música que, si se utiliza para datos en modo 1, da una capacidad de 681.984.000 bytes. Esta cifra suele expresarse como 650 MB ya que 1 MB equivale a 2^{20} bytes (1.048.576 bytes), no 1.000.000 bytes.

Una unidad de CD-ROM de 32x (4.915.200 bytes/s) no es rival para una unidad de disco magnético Fast SCSI-2, que opera a 10 MB/s. Aunque muchas unidades de CD-ROM utilizan la interfaz SCSI son muy frecuentes las unidades de CD-ROM IDE. Si pensamos que el tiempo de posicionamiento de la cabeza lectora suele ser de varios cientos de milisegundos, resulta obvio que las unidades de CD-ROM no están en la misma categoría de rendimiento que las de disco magnético, a pesar de su gran capacidad.

En 1986, Philips dio un nuevo golpe con el **Libro Verde**, añadiendo gráficos y capacidad para intercalar audio, vídeo y datos en el mismo sector, característica indispensable para los CD-ROMs multimedia.

La última pieza del rompecabezas de los CD-ROMs es el sistema de ficheros. Para poder utilizar el mismo CD-ROM en diferentes ordenadores, era preciso llegar a un acuerdo en lo tocante a los sistemas de ficheros en CD-ROM. A fin de lograr este acuerdo, los representantes de muchas compañías de ordenadores se reunieron en Lake Tahoe en lo alto de High Sierras, en la frontera entre los estados de California y Nevada, e idearon un sistema de ficheros al que denominaron **High Sierra** y que más adelante se convirtió en un nuevo estándar internacional (ISO 9660). Hay tres niveles. El nivel 1 utiliza nombres de fichero de hasta 8 caracteres seguidos opcionalmente por una extensión de hasta tres caracteres (el convenio de MS-DOS para nombrar ficheros). Los nombres de fichero sólo pueden contener letras mayúsculas, dígitos y el trazo de subrayado. Los directorios pueden anidarse hasta una profundidad de ocho, pero los nombres de directorio no pueden contener extensiones. El nivel 1 exige que todos los ficheros sean contiguos, lo que no resulta un problema en un medio que se graba una sola vez. Cualquier CD-ROM que se ajuste a ISO 9660 nivel 1 puede leerse utilizando MS-DOS, un ordenador Apple, un ordenador UNIX o casi cualquier otro ordenador. Los productores de CD-ROMs ven este hecho como una gran ventaja.

ISO 9660 nivel 2 permite nombres de hasta 32 caracteres, y el nivel 3 permite ficheros no contiguos. Las extensiones Rock Ridge (denominadas así caprichosamente por el pueblo que figura en la película de Gene Wilder, *Blazing Saddles*) permiten nombres muy largos (para UNIX), UIDs, GIDs y enlaces simbólicos, pero los CD-ROMs que no se ajustan al nivel 1 no pueden leerse en todos los ordenadores.

Los CD-ROMs se han vuelto extremadamente populares para publicar juegos, películas, enciclopedias, atlas y trabajos de referencia de todo tipo. En la actualidad, casi todo el software comercial se vende en CD-ROM. Su combinación de alta capacidad y bajo coste de fabricación los hace apropiados para innumerables aplicaciones.

CDs grabables

Inicialmente, el equipamiento necesario para producir un CD-ROM maestro (o un CD de audio) era extremadamente costoso. Pero como suele suceder en la industria de los ordenadores, nada permanece caro durante mucho tiempo. A mediados de los años noventa, las grabadoras de CD con un tamaño no mayor que un reproductor de CD eran periféricos comunes disponibles en la mayoría de las tiendas de ordenadores. Estos dispositivos seguían siendo diferentes de los discos magnéticos ya que una vez grabados, los CD-ROMs no podían borrarse. Sin embargo, rápidamente encontraron un nicho propio como medio de backup para grandes discos duros y también permitiendo a individuos o a empresas nacientes fabricar sus propios CD-ROMs en lotes pequeños o crear discos maestros para entregarlos a plantas comerciales de duplicación de CDs a gran escala. Estas unidades se conocen como CD-R (*CD-Recordable*).

Físicamente, los CD-Rs comienzan siendo discos vírgenes de policarbonato de 120 mm parecidos a los CD-ROMs, excepto en que contienen un surco de 0,6 mm de anchura para guiar el láser durante la escritura. El surco tiene una excursión sinusoidal de 0,3 mm a una frecuencia de exactamente 22,05 kHz para proporcionar una retroalimentación continua de forma que la

velocidad de rotación pueda ser precisamente monitorizada y ajustada si es necesario. Los CD-Rs tienen una apariencia similar a los CD-ROMs, excepto que son de color dorado en la parte de arriba, en lugar de ser plateados. El color dorado se debe a la utilización de oro en lugar de alumnio en la capa reflectante. A diferencia de los CDs plateados, que tienen depresiones físicas, en los CD-Rs la diferencia de reflectividad de los fosos y llanos debe simularse. Esto se consigue añadiendo una capa de colorante entre el policarbonato y la capa de oro reflectante, como se muestra en la Figura 5-22. Se utilizan dos tipos de colorante: cianina, que es verde, y ftalocianina, que es de color naranja amarillento. Los químicos pueden enfrascarse en argumentaciones interminables acerca de cuál es mejor. Estos colorantes son similares a los empleados en fotografía, lo cual explica por qué Kodak y Fuji son fabricantes destacados de discos CD-R en blanco.

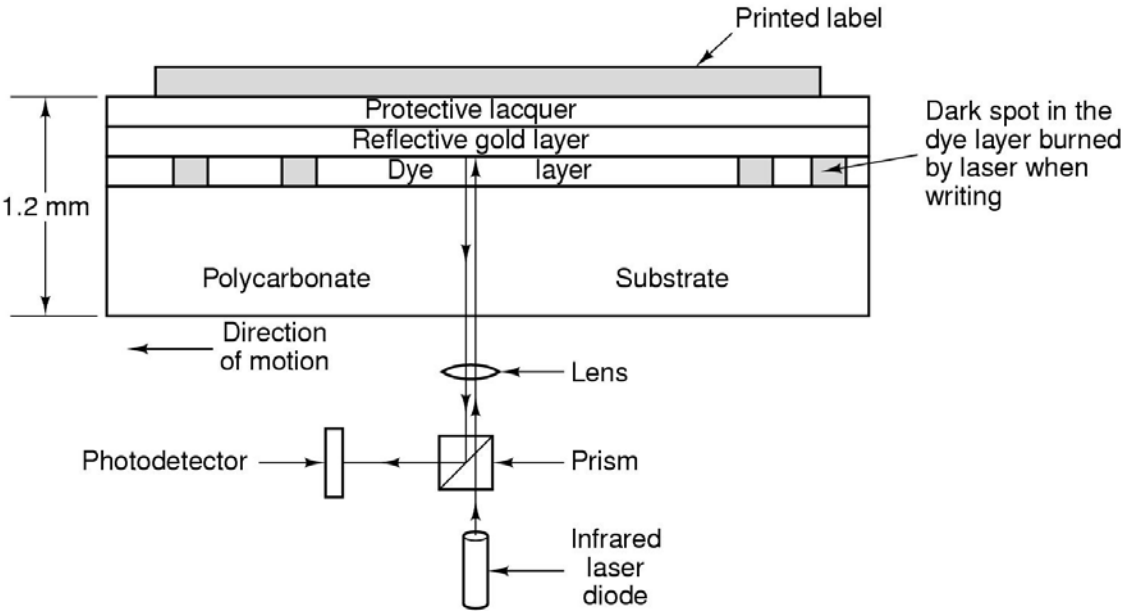

Figura 5-22. Sección de un disco CD-R y su láser (no está a escala). Un CD-ROM plateado tiene una estructura similar, sin la capa de colorante y con una capa agujereada de aluminio en lugar de la capa de oro.

En su estado inicial, la capa de colorante es transparente y permite que la luz láser pase y se refleje en la capa de oro. Para escribir, la potencia del láser del CD-R se aumenta a 8-16 mW. Cuando el haz incide sobre un punto del colorante, lo calienta y rompe un enlace químico. Este cambio en la estructura molecular crea una mancha oscura. Cuando se lee (a 0,5 mW), el fotodetector nota la diferencia entre las manchas oscuras donde se calentó el colorante y las áreas transparentes donde está intacto. Esta diferencia se interpreta como la diferencia entre fosos y llanos, incluso cuando se lee con un lector de CD-ROM normal o incluso con un reproductor de CDs de audio.

Ninguna nueva especie de CDs podría enorgullecerse de su origen sin un libro de color, por lo que al CD-R le corresponde el **Libro Naranja**, publicado en 1989. Este documento define el CD-R y también un formato nuevo, el **CD-ROM XA**, que permite escribir los CD-Rs de forma incremental, unos cuantos sectores hoy, unos cuantos mañana y otros pocos el mes que viene. Un grupo de sectores consecutivos escritos de una vez se denomina una **pista de CD-ROM**.

Uno de los primeros usos del CD-R fue para el Kodak PhotoCD. En este sistema, el cliente lleva un rollo de película fotográfica expuesta y su viejo PhotoCD al procesador fotográfico, recibiendo más tarde el mismo PhotoCD con las nuevas fotografías añadidas después de las antiguas. El nuevo lote de fotografías, que se crea digitalizando los negativos, se

escribe en el PhotoCD como una pista de CD-ROM separada. La escritura incremental era necesaria porque, cuando se introdujo ese producto, los discos CD-R vírgenes eran demasiado caros como para utilizar uno nuevo para cada rollo de película fotográfica.

Sin embargo la escritura incremental crea un nuevo problema. Antes del Libro Naranja, todos los CD-ROMs tenían una única **tabla de contenidos del volumen** (**VTOC**; *Volume Table Of Contents*) al principio. Ese esquema no funciona con las escrituras incrementales (es decir, multipista). La solución ofrecida por el Libro Naranja fue dar a cada pista de CD-ROM su propia VTOC. Los ficheros listados en la VTOC pueden incluir algunos de los ficheros de pistas anteriores, o todos. Cuando se inserta el CD-R en la unidad, el sistema operativo busca en todas las pistas del CD-ROM hasta encontrar la VTOC más reciente, que muestra el estado actual del disco. Al incluirse en la VTOC algunos de los ficheros de pistas anteriores, pero no todos, es posible crear la ilusión de que se han borrado algunos ficheros. Las pistas pueden agruparse en **sesiones**, dando pie a CD-ROMs **multisesión**. Los reproductores de CD de audio estándar no pueden reproducir CDs multisesión porque esperan una única VTOC al principio.

Cada pista tiene que escribirse de una vez en una única operación continua sin paradas. Como consecuencia, el disco duro del que provienen los datos debe ser lo suficientemente rápido como para suministrarlos a tiempo. Si los ficheros que van a copiarse están dispersos por todo el disco, los tiempos de posicionamiento del brazo pueden provocar que cese temporalmente el flujo de datos al CD-R ocasionando el vaciado del búfer (*buffer underrun*). El resultado del vaciado del búfer en medio de la grabación de la pista es un bonito y brillante (pero bastante caro) posavasos, o un disco volador dorado de 120 mm. El software del CD-R normalmente ofrece la opción de juntar todos los ficheros a escribir en una única imagen de CD-ROM contigua, de 650 MB, antes de grabar el CD-R, pero ese proceso normalmente duplica el tiempo de escritura real, requiere 650 MB de espacio libre en el disco, y tampoco protege contra los discos duros que, habiéndose calentado demasiado, empiezan a fallar hasta el punto de que el pánico les obliga a realizar una laboriosa recalibración térmica.

Los CD-Rs hacen posible que las personas individuales y las pequeñas compañías puedan copiar fácilmente CD-ROMs (y CDs de audio), generalmente violando los derechos de autor de quien los produjo. Se han ideado varios esquemas para obstaculizar tal piratería e impedir que un CD-ROM pueda leerse utilizando cualquier software que no sea del productor. Uno de ellos consiste en grabar todas las longitudes de los ficheros en el CD-ROM como siendo de varios gigabytes, frustrando cualquier intento de copiar los ficheros en un disco duro utilizando el software de copia estándar. Las longitudes verdaderas están implícitas en el software del productor u ocultas (posiblemente cifradas) en el CD-ROM en un lugar inesperado. Otro esquema utiliza ECCs intencionalmente incorrectos en sectores seleccionados, con la expectativa de que el software de copia de CDs “corrija” los errores. El software de la aplicación verifica los ECCs negándose a trabajar si los encuentra corregidos. Otras posibilidades son el uso de huecos no estándar entre las pistas y otros “defectos” físicos.

CDs regrabables

Aunque la gente está acostumbrada a otros medios en los que sólo puede escribirse una vez como el papel y la película fotográfica, existe demanda de un CD-ROM reescribible. Una tecnología que ya está disponible es el **CD-RW** (**CD-ReWritable**), que utiliza discos del mismo tamaño que los CD-Rs. Sin embargo, en lugar de un colorante como cianina o ftalocianina, los CD-RW utilizan una aleación de plata, indio, antimonio y teluro para la capa de grabación. Esta aleación tiene dos estados estables: cristalino y amorfo, con diferentes reflectividades.

Las unidades de CD-RW utilizan láseres con tres niveles de potencia. En la potencia más alta, el láser funde la aleación, convirtiéndola del estado critalino, altamente reflectante, al estado amorfo, de baja reflectividad, para simular un foso. En la potencia media, la aleación se

funde y vuelve a solidificarse en su estado cristalino natural, para convertirse otra vez en un llano. En la potencia más baja el estado del material se detecta (para su lectura) pero no tiene lugar ninguna transición de fase.

La razón por la que los CD-RWs no han sustituido a los CD-Rs es que los discos CD-RW vírgenes son mucho más caros que los CD-Rs. Además, para las aplicaciones de backup de los discos duros resulta una gran ventaja el hecho de que, una vez grabado, un CD-R no pueda borrarse accidentalmente.

DVD

El formato de CD/CD-ROM básico se ha estado utilizando desde alrededor de 1980. La tecnología ha mejorado desde entonces, por lo que ahora son económicamente factibles discos ópticos de mayor capacidad, existiendo una gran demanda de ellos. A Hollywood le encantaría sustituir las cintas de vídeo analógico por discos digitales, ya que ofrecen una calidad superior, su fabricación es más económica, duran más, ocupan menos espacio en los anaqueles de las tiendas de vídeo y no tienen que rebobinarse. Las compañías de electrónica de consumo estaban buscando un nuevo producto que arrasase, y muchas compañías de ordenadores querían añadir características multimedia a su software.

Esta combinación de tecnología y demanda por parte de tres industrias inmensamente ricas y poderosas ha dado origen al **DVD**, que originalmente era el acrónimo de **disco de vídeo digital** pero que ahora oficialmente es **Disco Digital Versátil**. Los CDs. utilizan el mismo diseño general que los CDs, con discos de policarbonato de 120 mm moldeados por inyección que contienen fosos y llanos, se iluminan con un diodo láser y se leen con un fotodetector. Lo nuevo es el uso de:

1. Fosos más pequeños (de 0,4 micras en lugar de 0,8 micras en los CDs).

2. Una espiral más apretada (0,74 micras entre pistas frente a las 1,6 de los CDs).

3. Un láser rojo (a 0,65 micras en lugar de 0,78 micras en los CDs).

En conjunto, estas mejoras aumentan la capacidad siete veces, hasta los 4,7 GB. Una unidad DVD a 1x opera a 1,4 MB/s (compárese con los 150 KB/s de los CDs). Desafortunadamente el cambio a los láseres rojos empleados en los supermercados implica que los reproductores de DVD requieren un segundo láser o un sistema óptico de conversión complicado para poder leer los CDs y CD-ROMs existentes, algo que quizá no todas las unidades puedan incluir. Además, algunas unidades de DVD no pueden leer CD-Rs o CD-RWs.

¿Es suficiente con 4,7 GB? Quizá. Utilizando compresión MPEG-2 (estandarizada en el ISO 13346), un disco DVD de 4,7 GB puede contener 133 minutos de vídeo en pantalla completa y con alta definición (720 × 480), así como pistas sonoras en hasta ocho idiomas y subtítulos en 32 idiomas más. Cerca del 92% de todas las películas que se han hecho en Hollywood duran menos de 133 minutos. No obstante, algunas aplicaciones como juegos multimedia o trabajos de referencia podrían necesitar más, y a Hollywood le gustaría poder poner varias películas en el mismo disco, por lo que se han definido cuatro formatos:

1. Un solo lado, una sola capa (4,7 GB).

2. Un solo lado, doble capa (8,5 GB).

3. Dos lados, una sola capa (9,4 GB).

4. Dos lados, doble capa (17 GB).

¿Por qué tantos formatos? En una sola palabra: política. Philips y Sony querían discos de un solo lado y doble capa para la versión de alta capacidad, pero Toshiba y Time Warner querían discos de dos lados y una sola capa. Philips y Sony no creían que la gente estaría dispuesta a dar la vuelta a los discos, y Time Warner no creía factible eso de colocar dos capas de un mismo lado. El compromiso: todas las combinaciones, pero el mercado determinará cuáles sobrevivirán.

La tecnología de capa dual tiene una capa reflectante en el fondo, y más arriba una capa semirreflectante. Dependiendo del lugar donde se enfoque el láser, el haz rebotará en una capa o en la otra. La capa inferior necesita fosos y llanos un poco más grandes para que pueda leerse de forma fiable, por lo que su capacidad es un poco más baja que la de la capa superior.

Los discos de dos lados se fabrican tomando dos discos de un solo lado, de 0,6 mm de espesor, y pegándolos reverso con reverso. Para que el espesor de todas las versiones sea el mismo, un disco de un solo lado consta de un disco de 0,6 mm pegado a un sustrato en blanco (o tal vez en el futuro uno que tenga grabados 133 minutos de anuncios con la esperanza de que la gente sienta curiosidad por ver qué es lo que hay ahí abajo). La estructura del disco de dos lados y doble capa se ilustra en la Figura 5-23.

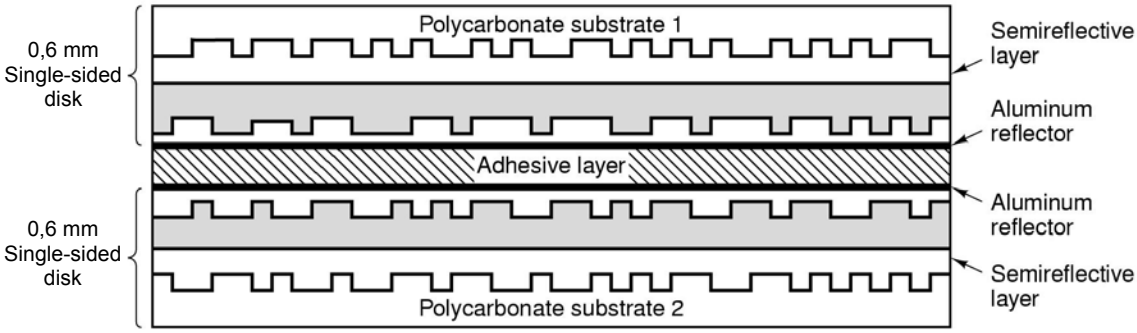

Figura 5-23. Disco DVD de dos lados y doble capa.

El DVD fue ideado por un consorcio de 10 compañías de electrónica de consumo, siete de ellas japonesas, en estrecha cooperación con los principales estudios hollywoodienses (algunos de los cuales son propiedad de las compañías japonesas de electrónica del consorcio). Las industrias de los ordenadores y de telecomunicaciones no fueron invitadas a la fiesta, y el enfoque resultante fue el uso del DVD para el alquiler de películas y para presentaciones de productos. Por ejemplo, entre las funciones estándar está la de saltarse en tiempo real las escenas escabrosas (de modo que los padres puedan convertir una película clasificada para adultos en una que los niños pequeños puedan ver sin peligro), sonido de seis canales y manejo de *Pan-and-Scan*. Esta última función permite al reproductor de DVD decidir de forma dinámica cómo recortar los bordes izquierdo y derecho de las películas (cuya proporción de aspecto es de 4:3).

Otra cosa que quizá no se le habría ocurrido a la industria de los ordenadores es una premeditada incompatibilidad entre los discos destinados a Estados Unidos y los destinados a Europa, además de estándares para otros continentes. Hollywood exigió esta "función" porque las películas siempre se exiben primero en Estados Unidos y luego se envían a Europa una vez que los vídeos salen a la venta en Estados Unidos. Lo que se buscaba era garantizar que las tiendas de vídeo europeas no pudieran comprar vídeos en Estados Unidos demasiado pronto, pues de lo contrario, se reducirían las entradas de taquilla por la exhibición de películas nuevas en las salas de cine europeas. Si Hollywood controlara el destino de la industria de los ordenadores, habríamos tenido disquetes de 3,5 pulgadas en Estados Unidos y de 9 cm en Europa.

Formateo del disco

Un disco duro consiste en una pila de platos de aluminio, aleación o vidrio de 5,25 o 3,5 pulgadas de diámetro (o incluso más pequeños en los ordenadores portátiles). En cada plato está depositada una delgada capa de óxido metálico magnetizable. Después de la fabricación, no hay ninguna información en ninguna parte del disco.

Para que el disco pueda utilizarse es necesario que cada plato reciba un **formato de bajo nivel** realizado por software. El formato consiste de una serie de pistas concéntricas, cada una de las cuales contiene cierto número de sectores, con cortos espacios vacíos entre ellos. En la Figura 5-24 se muestra el formato de un sector.

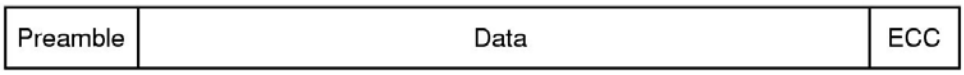

Figura 5-24. Un sector de disco.

El preámbulo comienza con cierto patrón de bits que permite al hardware reconocer el principio del sector. También contiene los números de cilindro y de sector y alguna otra información. El programa de formateo de bajo nivel determina el tamaño de la porción de datos. La mayoría de los discos utilizan sectores de 512 bytes. El campo ECC contiene información redundante que puede servir para corregir errores de lectura. El tamaño y el contenido de este campo varía de fabricante a fabricante, dependiendo de cuanto espacio de disco está dispuesto a sacrificar el diseñador a cambio de una mayor fiabilidad, y del grado de complejidad del código ECC que puede manejar la controladora. Un campo ECC de 16 bytes no es inusual. Además, todos los discos duros tienen asignados un cierto número de sectores de reserva que sirven para sustituir los sectores con defectos de fabricación que puedan aparecer.

Cuando se aplica el formato de bajo nivel el sector 0 de cada pista se desplaza respecto del sector 0 de la pista anterior. Este desplazamiento, denominado **sesgo del cilindro** (*cilinder skew*), tiene por objeto mejorar el rendimiento. La idea es permitir que el disco lea varias pistas en una sola operación continua sin perder datos. La naturaleza del problema puede apreciarse observando la Figura 5-18(a). Supongamos que una petición necesita que se lean 18 sectores comenzando por el sector 0 de la pista más interna. La lectura de los primeros 16 sectores tarda una rotación del disco, pero a continuación es necesario mover el brazo una pista hacia fuera para leer el sector 17. Para cuando la cabeza termina de moverse una pista, el sector 0 ya ha pasado de largo, de modo que se necesita esperar toda una rotación para leer ese sector. El problema se elimina desplazando los sectores como se muestra en la Figura 5-25.

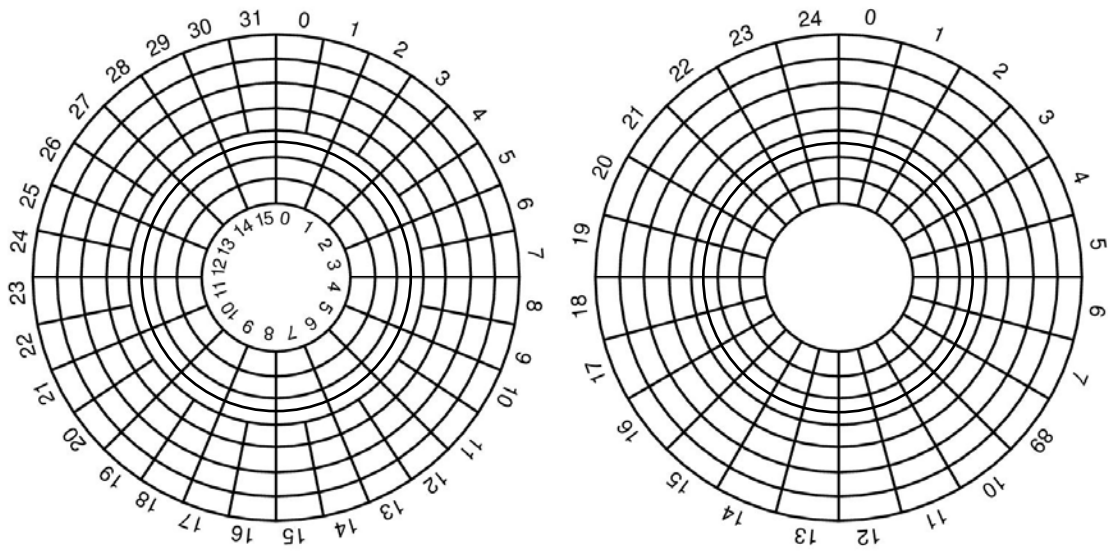

Figura 5-18. (a) Geometría física de un disco con dos zonas. (b) Una posible geometría virtual para este disco.

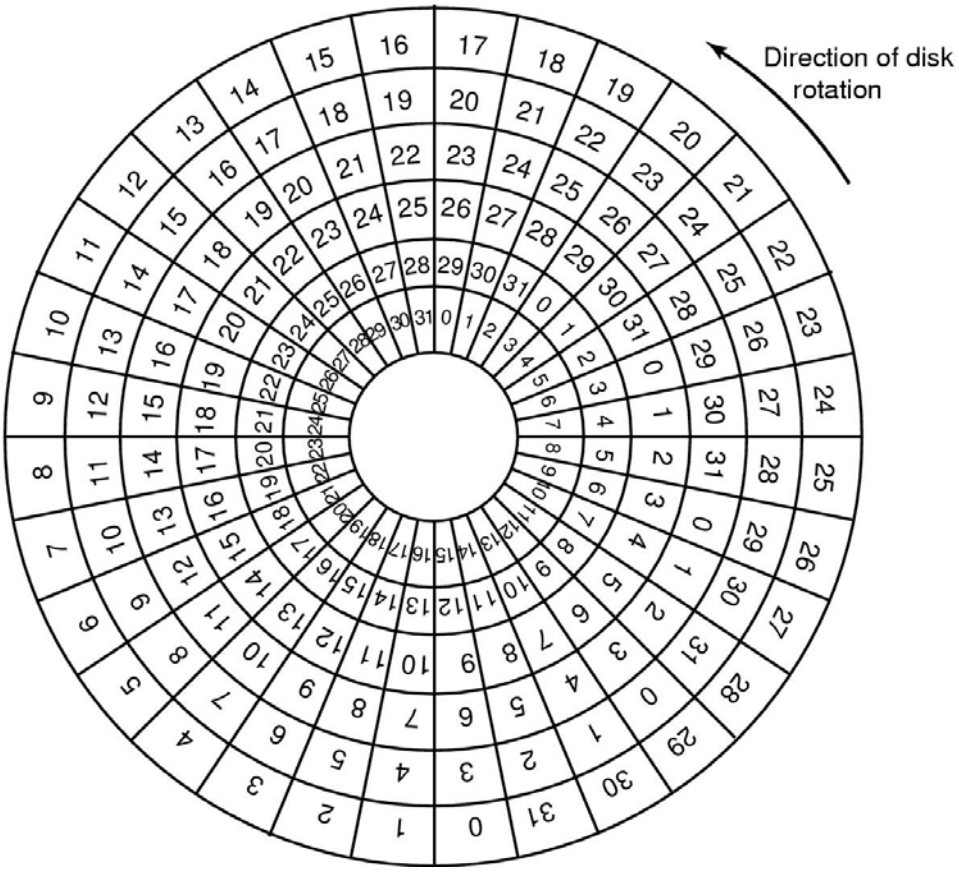

Figura 5-25. Una ilustración del sesgo del cilindro.

La cantidad de sesgo del cilindro depende de la geometría de la unidad. Por ejemplo, una unidad de 10.000 rpm realiza una rotación completa cada 6 milisegundos. Si una pista contiene 300 sectores, pasará un sector bajo la cabeza cada 20 microsegundos. Si el tiempo de posicionamiento desde una pista a la siguiente es de 800 microsegundos, eso significa que pasarán 40 sectores durante el posicionamiento, por lo que el sesgo del cilindro debe ser de 40 sectores, en vez de los tres sectores que se muestran en la Figura 5-25. Vale la pena mencionar que la conmutación entre cabezas tarda también un tiempo finito, de manera que hay un **sesgo de la cabeza** además de un sesgo del cilindro, pero el sesgo de la cabeza no es muy grande.

Como resultado del formateo de bajo nivel, la capacidad del disco se reduce, dependiendo de los tamaños del preámbulo, del espacio vacío entre sectores y del ECC, así como del número de sectores reservados para sustituir los sectores defectuosos. A menudo la capacidad tras ese formateo es un 20% menor que la capacidad inicial. Los sectores de reserva no cuentan de cara a la capacidad tras el formateo, por lo que todos los discos de un tipo dado tienen exactamente la misma capacidad cuando salen de la fábrica, independientemente de cuántos sectores defectuosos tengan en realidad (si el número de sectores defectuosos excede el número de sectores de reserva, la unidad se rechaza y no sale de la fábrica).

Existe una considerable confusión en lo que respecta a la capacidad de los discos, porque algunos fabricantes anuncian la capacidad no formateada para aparentar que sus discos son más grandes de lo que en realidad lo son. Por ejemplo, consideremos una unidad de disco cuya capacidad no formateada es de 20×10^9 bytes. Esta unidad podría venderse como un disco de 20 GB. Sin embargo, tras el formateo, quizás sólo queden $2^{34} \cong 17{,}2 \times 10^9$ bytes disponibles para guardar datos. Por si no hubiera poca confusión, es probable que el sistema operativo informe de esa capacidad como 16,0 GB, no 17,2 GB, porque el software considera que 1 GB es 2^{30} (1.073.741.824) bytes, y no 10^9 (1.000.000.000) bytes.

Para empeorar todavía más las cosas, en el mundo de la comunicación de datos 1 Gbps significa 1.000.000.000 bits/segundo porque el prefijo *giga* realmente significa 10^9 (después de todo, un kilómetro son 1000 metros, no 1024 metros). Sólo al hablar de tamaños de memorias y de discos los prefijos kilo, mega, giga y tera significan 2^{10}, 2^{20}, 2^{30} y 2^{40}, respectivamente.

El formateo afecta también al rendimiento. Si un disco de 10.000 rpm tiene 300 sectores de 512 bytes por pista, tardará 6 milisegundos en leer los 153.600 bytes de una pista, lo que significa una velocidad de transferencia de datos de 25.600.000 bytes/segundo, o 24,4 MB/s. Será imposible alcanzar una velocidad mayor, sea cual sea el tipo de interfaz presente, incluso aunque sea una interfaz SCSI que opere a 80 MB/segundo o a 160 MB/segundo.

En realidad, para poder leer de forma continuada a esas velocidades se requiere un gran búfer en el controlador. Por ejemplo, consideremos un controlador cuyo búfer tiene capacidad para un sector y al que se ha enviado un comando para leer dos sectores consecutivos. Después de leer el primer sector del disco y realizar el cálculo de su ECC, hay que transferir los datos a la memoria principal. Mientras tiene lugar esa transferencia, el siguiente sector está pasando velozmente bajo la cabeza. Cuando se completa la copia a la memoria, el controlador tiene que esperar casi el tiempo de una rotación entera hasta que el segundo sector vuelva a pasar debajo de la cabeza.

Este problema puede eliminarse numerando los sectores de una forma intercalada cuando se formatea el disco. En la Figura 5-26(a) vemos el patrón de numeración usual (ignorando aquí el sesgo del cilindro). En la Figura 5-26(b) vemos un **intercalamiento sencillo** (*single interleaving*), que da a la controladora un respiro entre sectores consecutivos para que le dé tiempo a copiar el búfer en la memoria principal.

Si el proceso de copiado es muy lento, podría ser necesario utilizar **doble intercalamiento** como en la Figura 5-26(c). Si el controlador tiene un búfer con capacidad para un solo sector, el intercalamiento es una técnica necesaria, independientemente de si la copia del búfer a la memoria principal la realiza el controlador, la CPU principal o un chip de DMA, ya que de todos modos la copia tardará algún tiempo. Para evitar la necesidad del intercalamiento de los sectores el controlador tendría que ser capaz de guardar en el búfer toda una pista. Muchos controladores modernos pueden hacerlo.

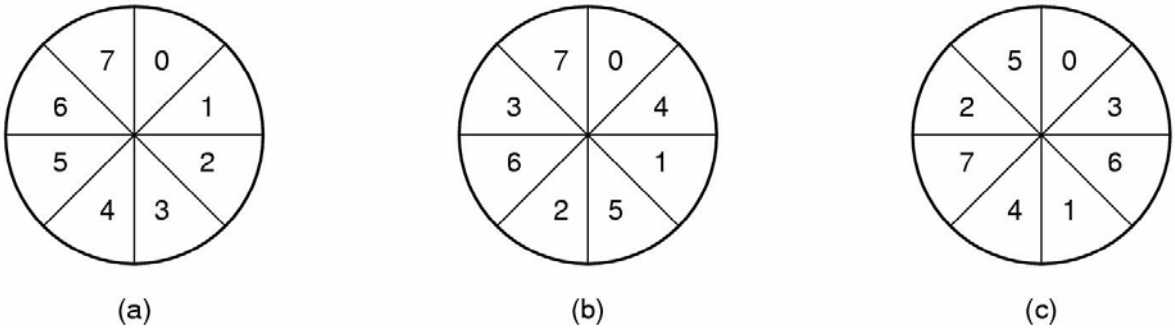

Figura 5-26. (a) Sin intercalamiento. (b) Intercalamiento sencillo. (c) Intercalamiento doble.

Una vez completado el formateo de bajo nivel, el disco se divide en particiones. Desde el punto de vista lógico, cada partición es como si fuera un disco separado. En el Pentium y en la mayoría de los demás ordenadores, el sector 0 contiene el **registro de arranque maestro** (*master boot record*), que contiene el programa de arranque junto con la tabla de particiones al final del sector. La tabla de particiones contiene para cada partición su sector de comienzo y su tamaño. En el Pentium la tabla de particiones tiene espacio para sólo cuatro particiones. Si todas ellas son para Windows, recibirán los nombres C:, D:, E: y F:, y se tratan como unidades separadas. Si tres de ellas son para Windows y una es para UNIX, Windows llamará a sus particiones C:, D: y E:. El primer CD-ROM será entonces F:. Para poder arrancar desde el disco duro, una partición debe marcarse como activa en la tabla de particiones.

El paso final en la preparación de un disco para su uso consiste en realizar un **formateo de alto nivel** a cada partición (de forma separada). Esta operación establece el bloque de arranque, la gestión de la memoria libre (lista de bloques libres o mapa de bits), el directorio raíz y un sistema de ficheros vacío. También se introduce un código en la entrada apropiada de la tabla de particiones para indicar qué sistema de ficheros se está utilizando en la partición, ya que cada sistema operativo utiliza normalmente un sistemas de ficheros incompatible (por razones históricas). En este momento es posible ya por fin arrancar el sistema.

Cuando se enciende el ordenador, comienza ejecutándose el BIOS, el cual lee del disco el registro de arranque maestro y le cede el control a su programa de arranque. Este programa de arranque averigua entonces qué partición está activa, lee el sector de arranque de esa partición y lo ejecuta. El sector de arranque contiene un pequeño programa que busca en el directorio raíz de la partición un cierto programa (el sistema operativo o bien un cargador de autoarranque más grande). Ese programa se carga en la memoria y se ejecuta.

5.4.2 Algoritmos de planificación del brazo del disco

En esta sección echaremos un vistazo a algunas cuestiones relacionadas con los drivers de disco en general. Comenzaremos considerando el tiempo que se requiere para leer o escribir un bloque de disco. El tiempo requerido viene determinado por tres factores:

1. Tiempo de posicionamiento (el tiempo que se tarda en mover el brazo hasta el cilindro correcto).
2. Retraso (latencia) rotacional (el tiempo que tarda el sector correcto en girar hasta pasar por debajo de la cabeza de lectura).
3. Tiempo de transferencia de datos real.

Para la mayoría de los discos, el tiempo de posicionamiento (*seek time*) domina los otros dos tiempos, por lo que una reducción en el tiempo de posicionamiento medio puede mejorar substancialmente el rendimiento del sistema.

Si el driver del disco sólo acepta una petición de cada vez y las atiende en el orden de llegada, es decir, First-Come First-Served (FCFS), poco puede hacerse para optimizar el tiempo de posicionamiento. Sin embargo, si el disco está sometido a una carga intensa de peticiones, podría utilizarse otra estrategia. Es muy probable que mientras el brazo esté moviéndose para atender una petición, otros procesos generen nuevas peticiones de disco. Muchos drivers de disco mantienen una tabla, indexada por número de cilindro, con todas las peticiones pendientes para cada cilindro encadenadas en una lista enlazada encabezada por las entradas de la tabla.

Dado este tipo de estructura de datos, podemos mejorar el algoritmo de planificación primera en llegar, primera en ser servida. Para ver cómo, consideremos un disco imaginario con 40 cilindros. Supongamos que llega una petición para leer un bloque en el cilindro 11, y que mientras el brazo está moviéndose hacia el cilindro 11, llegan nuevas peticiones para los cilindros 1, 36, 16, 34, 9 y 12, en ese orden. Esas peticiones se colocan en la tabla de peticiones pendientes, con una lista enlazada distinta para cada cilindro. Las peticiones se muestran en la Figura 5-27.

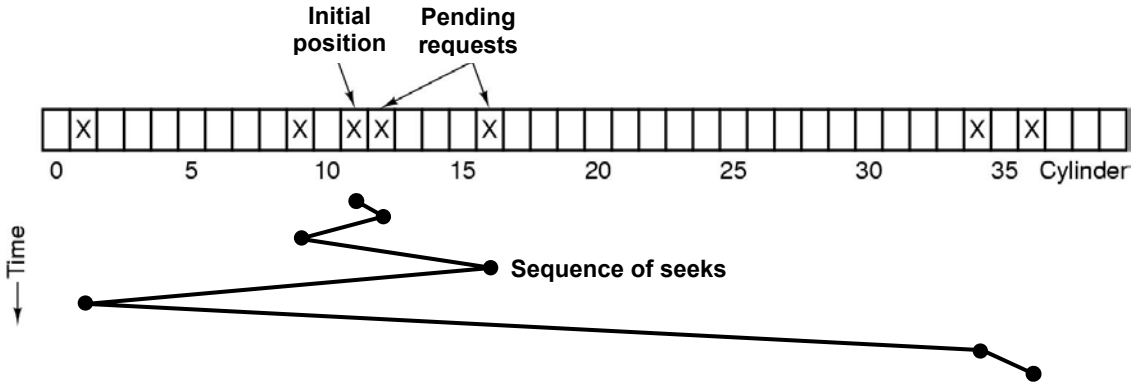

Figura 5-27. Algoritmo de planificación del disco Shortest Seek First (SSF).

Cuando termina de atenderse la petición actual (correspondiente al cilindro 11), el driver del disco tiene que elegir la petición que atenderá a continuación. Si utiliza FCFS, irá a continuación al cilindro 1, luego al 36, y así todas las demás. Este algoritmo requerirá movimientos del brazo de 10, 35, 20, 18, 25 y 3 cilindros respectivamente, para un total de 111 cilindros.

Alternativamente, el driver del disco podría atender siempre a continuación la petición más cercana, con el fin de minimizar el tiempo de posicionamiento. Dadas las peticiones de la Figura 5-27, la secuencia es 12, 9, 16, 1, 34 y 36, como indica la línea en zigzag de la parte baja de la Figura 5-27. Con esta sucesión, los movimientos del brazo son de 1, 3, 7, 15, 33 y 2, para un total de 61 cilindros. Este algoritmo, **Shortest Seek First** (**SSF**), reduce el movimiento total del brazo en casi la mitad comparado con FCFS.

Desafortunadamente SSF tiene un problema. Supongamos que siguen llegando más peticiones mientras se están procesando las peticiones de la Figura 5-27. Por ejemplo, si después de posicionarnos sobre el cilindro 16 está presente una nueva petición para el cilindro 8, esa solicitud tendrá prioridad sobre la del cilindro 1. Si luego llega una petición para el cilindro 13, el brazo se dirigirá entonces al cilindro 13, no al 1. Si el disco está sometido a mucha carga de peticiones, el brazo tenderá a permanecer en el medio del disco la mayor parte del tiempo, mientras que las peticiones para ambos extremos tendrán que esperar hasta que una fluctuación estadística en la carga de peticiones provoque que no haya ninguna petición de cilindros de la zona central. En consecuencia las peticiones alejadas del centro del disco podrían recibir un mal servicio con este algoritmo. En este caso se produce un conflicto entre el objetivo de maximizar el rendimiento y el objetivo de ser justos evitando discriminar algún tipo de peticiones.

Los rascacielos también tienen que enfrentarse con esta situación comprometida. El problema de planificar el ascensor de un rascacielos es similar al de planificar el brazo de un disco. Continuamente llegan peticiones llamando al ascensor para que acuda a los pisos (cilindros) al azar. El ordenador que controla el ascensor fácilmente puede seguir la pista de la secuencia en la cual los usuarios oprimieron el botón de llamada, atendiéndolos utilizando FCFS o SSF.

Sin embargo, la mayoría de los ascensores utilizan un algoritmo diferente para conciliar los objetivos en conflicto de justicia y eficiencia. El ascensor siempre continúa su avance en el mismo sentido (hacia arriba o hacia abajo) hasta que no queden peticiones pendientes en ese sentido, momento en el cual el ascensor cambia de sentido. Este algoritmo conocido tanto en el mundo de los discos como en el de los ascensores como el **algoritmo del ascensor**, requiere que el software mantenga un bit: el bit del sentido actual del movimiento, ARRIBA o ABAJO. Cuando termina de atenderse una petición, el driver del disco o del ascensor comprueba el bit. Si es ARRIBA, el brazo o la cabina se mueve hasta la siguiente petición pendiente hacia arriba. Si no hay peticiones pendientes en posiciones más altas, se invierte el bit del sentido del movimiento. Una vez que el bit se establece a ABAJO, el movimiento es hasta la siguiente posición hacia abajo solicitada, si la hay.

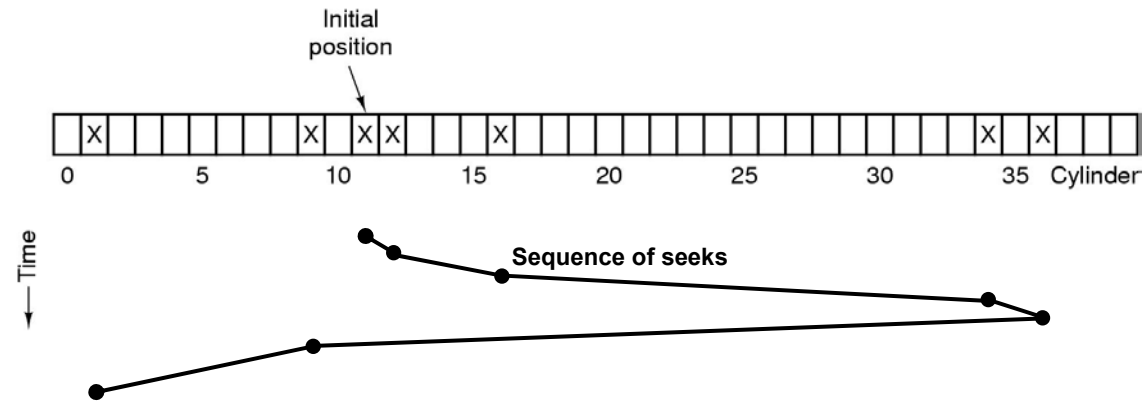

Figura 5-28. Algoritmo del ascensor para la planificación de peticiones de disco.

La Figura 5-28 muestra el algoritmo del ascensor aplicado a las mismas siete peticiones que en la Figura 5-27, suponiendo que el bit de sentido del movimiento es inicialmente ARRIBA. El orden en el que se atienden las peticiones es 12, 16, 34, 36, 9 y 1, lo que significa movimientos del brazo de 1, 4, 18, 2, 27 y 8, para un total de 60 cilindros. En este caso el algoritmo del ascensor es ligeramente mejor que el SSF, aunque normalmente es peor. Una propiedad agradable que tiene el algoritmo del ascensor es que, dado cualquier conjunto de peticiones, existe una cota superior fija sobre el movimiento total, que es exactamente el doble del número de cilindros.

Una pequeña modificación de ese algoritmo que tiene menor varianza en los tiempos de respuesta (Teory, 1972) consiste en explorar siempre en el mismo sentido. Una vez que se ha servido el cilindro de número más alto con una petición pendiente, el brazo se dirige al cilindro de número más bajo que tenga una petición pendiente, para luego continuar moviéndose hacia arriba. En definitiva es como si el cilindro de número más bajo estuviera justo a continuación del cilindro de número más alto.

Algunos controladores de disco permiten que el software conozca el número del sector que está actualmente pasando bajo la cabeza. Con un controlador así, es posible realizar otra optimización. Si hay dos o más peticiones pendientes para el mismo cilindro, el driver puede enviar una petición para el sector que pasará primero bajo la cabeza. Obsérvese que cuando un cilindro comprende varias pistas, dos peticiones consecutivas pueden corresponder a pistas distintas sin que eso represente ninguna penalización; el controlador puede seleccionar cualquiera de sus cabezas instantáneamente, ya que la selección de cabeza lectora no requiere ni movimiento del brazo ni retraso rotacional.

Si el disco tiene la propiedad de que su tiempo de posicionamiento es mucho más corto que su latencia rotacional, deberá utilizarse una estrategia de optimización diferente. Las peticiones pendientes deberán ordenarse por número de sector, y tan pronto como el siguiente sector esté a punto de pasar bajo la cabeza, el brazo deberá posicionarse sobre la pista correcta para leer o escribir en él.

En los discos duros modernos, los retrasos por posicionamiento y rotación dominan de tal manera el rendimiento que resulta ineficiente leer tan solo uno o dos sectores de cada vez. Por esa razón, muchos controladores de disco siempre leen y guardan en la caché varios sectores, incluso aunque sólo se haya solicitado uno. Normalmente, cualquier petición de lectura de un sector provocará que se lea ese sector y buena parte de la pista que lo contiene, o toda, dependiendo del espacio que esté disponible en la memoria caché del controlador. Por ejemplo, el disco descrito en la Figura 5-17 tiene una caché de 2 o 4 MB. El controlador determina dinámicamente el uso que se dará a la caché. En su modo más sencillo, la caché se divide en dos secciones, una para las lecturas y otra para las escrituras. Si puede satisfacerse una lectura posterior con el contenido de la caché de la controladora, será posible remitir de inmediato los datos solicitados.

Vale la pena señalar que la caché del controlador de disco es completamente independiente de la caché del sistema operativo. La caché del controlador normalmente contiene bloques que realmente no se han solicitado, pero que resultaba conveniente leer porque dio la casualidad de que pasaron bajo la cabeza lectora como un efecto secundario de alguna otra lectura. En contraste, cualquier caché mantenida por el sistema operativo contendrá bloques que se leyeron explícitamente y que el sistema operativo considera que podrían volverse a necesitar en un futuro cercano (por ejemplo, un bloque de disco que contiene un bloque de directorio).

Cuando un mismo controlador controla varias unidades de disco, el sistema operativo debe mantener una tabla de peticiones pendientes distinta para cada unidad. Siempre que cualquier unidad esté ociosa, el driver deberá ordenarle que vaya moviendo su brazo hacia el cilindro que se va a necesitar a continuación (supuesto que el controlador permita el

posicionamiento solapado). Cuando termina la transferencia en curso, puede comprobarse si alguna unidad de disco está ya posicionada sobre el cilindro correcto. Si una o más lo están, puede iniciarse la siguiente transferencia en una unidad que ya esté posicionada sobre el cilindro correcto. Si ninguno de los brazos está todavía en el lugar correcto, el driver deberá ordenar un nuevo posicionamiento en la unidad que acaba de terminar su transferencia quedándose a la espera de la siguiente interrupción para ver cual de los brazos llega primero a su destino.

Es importante darse cuenta de que todos los algoritmos de planificación de disco anteriores suponen tácitamente que la geometría real del disco es idéntica a la geometría virtual. Si no lo fuese, no tendrá sentido planificar las peticiones de disco porque el sistema operativo realmente no puede saber si el cilindro 40 está más cerca del cilindro 39 que el cilindro 200. Por otra parte, si el controlador de disco puede aceptar múltiples solicitudes pendientes, podrá utilizar internamente esos algoritmos de planificación. En ese caso, los algoritmos siguen siendo válidos, pero a un nivel más bajo, dentro del propio controlador.

5.4.3 Tratamiento de los errores

Los fabricantes de discos siempre están empujando los límites de la tecnología, aumentando las densidades lineales de grabación de bits. Una pista situada en el medio de un disco de 5,25 pulgadas tiene una circunferencia de aproximadamente 300 milímetros. Si la pista contiene 300 sectores de 512 bytes, la densidad lineal de grabación puede ser de unos 5000 bits/milímetro tomando en cuenta el hecho de que se pierde algo de espacio debido a los preámbulos, los ECCs y los huecos entre sectores. La grabación de 5000 bits/milímetro requiere un sustrato extremadamente uniforme y un recubrimiento de óxido muy fino. Desafortunadamente, no es posible fabricar un disco con tales especificaciones que no tenga defectos. Tan pronto como la tecnología de fabricación mejora hasta el punto en el que es posible operar impecablemente con tales densidades, los diseñadores del disco se pasan a densidades más altas para aumentar la capacidad. En consecuencia vuelven a aparecer los defectos.

Los defectos de fabricación introducen sectores defectuosos, es decir, sectores que no permiten leer correctamente el valor que supone que se acaba de escribir en ellos. Si el defecto es muy pequeño, digamos de unos cuantos bits, es posible seguir utilizando el sector defectuoso dejando que el ECC corrija los errores continuamente. Si el defecto es mayor, no será posible enmascarar el error.

Hay dos enfoques generales para tratar los bloques defectuosos: que se ocupe de ellos el controlador o que se ocupe de ellos el sistema operativo. En el primer enfoque, antes de que el disco salga de la fábrica, se testea escribiéndose en el disco una lista conteniendo los sectores defectuosos. Cada sector defectuoso se sustituye por uno de los de repuesto.

Hay dos formas de realizar esa sustitución. En la Figura 5-29(a) vemos una pista de disco con 30 sectores de datos y dos de repuesto, donde el sector 7 tiene un defecto. El controlador puede reasignar el número 7 a uno de los sectores de repuesto, como se muestra en la Figura 5-29(b). La otra posibilidad es desplazar todos los sectores una posición hacia arriba, como se muestra en la Figura 5-29(c). En ambos casos el controlador tiene que saber qué sector es cada uno. Puede mantenerse al tanto de esa información con la ayuda de tablas internas (una por pista) o reescribiendo los preámbulos de modo que proporcionen los números de sector reajustados. Si se reescriben los preámbulos, el método de la Figura 5-29(c) implica más trabajo (porque hay que reescribir 23 preámbulos) pero en última instancia produce un mejor rendimiento porque sigue siendo posible leer toda una pista en una única rotación del disco.

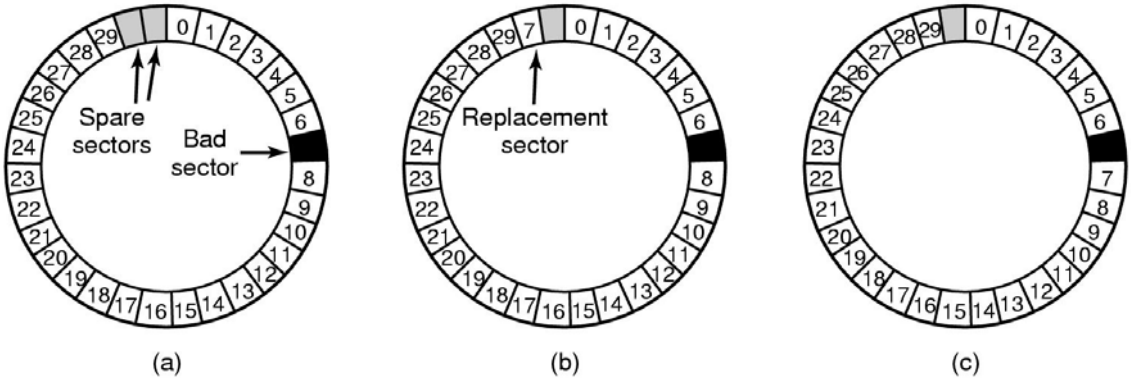

Figura 5-29. (a) Una pista de un disco con un sector defectuoso. (b) Sustitución del sector defectuoso por uno de repuesto. (c) Desplazamiento de todos los sectores para saltarse el sector defectuoso.

Los errores también pueden surgir durante el funcionamiento normal tiempo después de que se haya instalado la unidad de disco. La primera línea de defensa al presentarse un error que no puede resolver el ECC consiste en reintentar la operación. Algunos errores de lectura son transitorios, es decir, están causados por partículas de polvo que se sitúan temporalmente bajo la cabeza y desaparecen en un segundo intento. Si el controlador detecta la aparición de errores persistentes en el acceso a cierto sector, puede sustituirlo por un sector de repuesto antes de que el sector original quede completamente inutilizable. De esa manera se evita la pérdida de datos sin que ni el sistema operativo ni el usuario lleguen siquiera a enterarse del problema. Normalmente es necesario utilizar el método de la Figura 5-29(b) al encontrarse todos los demás sectores ya ocupados con datos. En ese caso la utilización del método de la Figura 5-29(c) podría requerir no sólo la reescritura de los preámbulos, sino también la copia de todos los datos.

Ya dijimos anteriormente que hay dos enfoques a la hora de tratar los errores: tratarlos en el controlador o tratarlos en el sistema operativo. Si el controlador no tiene capacidad para remapear de forma transparente los sectores como hemos visto anteriormente, deberá ser el sistema operativo quien lo haga por software. Eso significa que el sistema operativo deberá crear una lista de sectores defectuosos, bien sea leyéndola del disco o simplemente testeando él mismo todo el disco. Una vez que el sistema operativo sabe qué sectores son defectuosos, puede proceder a construir las tablas de remapeo de los bloques. En el caso de que el sistema operativo quiera utilizar el enfoque de la Figura 5-29(c) deberá desplazar un sector hacia arriba los datos de los sectores 7 a 29.

Cuando el sistema operativo realiza el remapeo de los sectores deber asegurarse de que no haya sectores defectuosos en los ficheros y tampoco en la lista o en el mapa de bits de bloques libres. Una forma de hacerlo es crear un fichero secreto integrado por todos los sectores defectuosos. Si este fichero no se incorpora al sistema de ficheros, los usuarios no podrán leerlo accidentalmente (o, lo que sería peor, liberar sus bloques).

Sin embargo, todavía queda el problema de los backups (copias de respaldo). Si se hace una copia de seguridad de un disco fichero a fichero, es importante que el programa de backup no trate de copiar el fichero de bloques defectuosos. Para evitarlo, el sistema operativo tiene que ocultar dicho el fichero de bloques defectuosos tan bien que ni siquiera un programa de backup pueda encontrarlo. Si el disco se copia sector a sector en vez de fichero a fichero, será muy difícil, si no imposible, evitar errores de lectura durante el backup. La única esperanza es que el programa de backup sea lo bastante inteligente como para darse por vencido después de 10 lecturas fallidas, continuando con el siguiente sector.

Los sectores defectuosos no son la única fuente de errores. También pueden presentarse errores de posicionamiento del brazo provocados por problemas mecánicos. El controlador sigue la pista de la posición del brazo internamente. Para realizar un posicionamiento, el controlador envía una serie de pulsos al motor del brazo, un pulso por cilindro, a fin de mover el brazo al nuevo cilindro. Una vez que el brazo llega a su destino, el controlador lee el número de cilindro actual del preámbulo del siguiente sector. Si el brazo no está en el lugar correcto, es que ha ocurrido un error de posicionamiento.

La mayoría de los controladores de disco duro corrigen automáticamente los errores de posicionamiento, pero la mayoría de los controladores de disquete (incluido el del Pentium) tan solo activan un bit de error y dejan el resto como responsabilidad del driver. El driver trata el error enviando un comando recalibrate para mover el brazo tan lejos como pueda ir y resetea el estado interno del controlador para que considere como cilindro actual el cilindro 0. Normalmente eso resuelve el problema. En otro caso será necesario reparar la unidad.

Como hemos visto, el controlador es en realidad un pequeño ordenador especializado, incluyendo software, variables, búferes y, de vez en cuando, errores (*bugs*). A veces sucede que una secuencia inusual de sucesos tales como una interrupción en una unidad, exactamente en el momento en que se envía un comando recalibrate a otra unidad puede disparar el error provocando que la controladora entre en un bucle infinito u olvide lo que estaba haciendo en ese momento. Los diseñadores de los controladores normalmente tienen previsto lo peor y proporcionan un pin en el chip que, cuando se activa, fuerza al controlador a olvidarse de todo lo que estaba haciendo y a resetearse a si mismo. Si todo lo demás falla, el driver del disco puede establecer un bit para activar esta señal y resetear el controlador. Si eso no resuelve el problema, lo único que el driver del dispositivo puede hacer es mostrar un mensaje y darse por vencido.

La recalibración de un disco produce un ruidillo raro pero por lo demás normalmente no causa ningún problema. Sin embargo, hay una situación en la que la recalibración representa un serio problema: los sistemas con restricciones de tiempo real. Cuando se está reproduciendo un vídeo almacenado en un disco duro, o cuando se están grabando ficheros de un disco duro en un CD-ROM es esencial que los bits lleguen desde el disco duro a un ritmo uniforme. Bajo esas circunstancias, las recalibraciones insertan huecos en el flujo de bits y son por lo tanto inaceptables. Para tales aplicaciones existen unidades de disco especiales, llamadas **discos AV** (**discos audiovisuales**), que nunca necesitan recalibrarse.

5.4.4 Almacenamiento Estable

Como hemos visto, en los discos a veces aparecen errores que convierten repentinamente sectores buenos en sectores defectuosos, o que destruyen unidades completas de forma inesperada. Los RAIDs protegen frente a la pérdida de unos cuantos sectores o incluso de toda una unidad, pero no nos protegen frente a errores de escritura que consigan grabar datos inicialmente erróneos. Tampoco nos protegen frente a caídas del sistema durante las escrituras, las cuales corrompen los datos originales sin reemplazarlos por nuevos datos.

En algunas aplicaciones es esencial que los datos nunca se pierdan ni corrompan, aunque se presenten errores de disco o de la CPU. Idealmente, un disco debería trabajar todo el tiempo sin errores. Desafortunadamente, la realidad no es así. Lo que sí es factible es tener un subsistema de disco que tenga la siguiente propiedad: cuando se pone en marcha una operación de escritura, o bien termina habiéndose escrito correctamente los datos, o bien termina sin efecto comunicando un fallo y dejando los datos existentes en el disco intactos. Un sistema así se denomina un **sistema de almacenamiento estable** y se implementa por software (Lampson y Sturgis, 1979). A continuación describimos la idea original con ligeras variaciones.

Antes de describir el algoritmo, es importante tener un modelo claro de los posibles errores. El modelo supone que cuando se escribe un bloque en un disco (uno o más sectores), la escritura es correcta o incorrecta, y que ese error puede detectarse en una lectura subsiguiente examinando los valores de los campos ECC. En principio, nunca es posible garantizar la detección de errores porque con un campo ECC de, digamos, 16 bytes para proteger un sector de 512 bytes existen 2^{4096} posibles valores de los datos y sólo 2^{128} posibles valores de ECC. Por tanto, si un bloque sufre alteraciones durante la escritura pero el ECC no, existen billones de billones de combinaciones incorrectas que producen el mismo ECC. Si por casualidad se presenta cualquiera de ellas, no será posible detectar el error. En general, la probabilidad de que unos datos al azar tengan el ECC de 16 bytes correcto es de 2^{-128}, una cifra lo bastante pequeña como para considerarla cero, aunque en realidad no lo sea.

El modelo considera posible que un sector escrito correctamente pueda estropearse espontáneamente, dejando de poder leerse. Sin embargo se supone que tales sucesos son tan raros que la probabilidad de que el mismo sector se estropee en una segunda unidad (independiente) durante un intervalo de tiempo razonable (por ejemplo, un día) es lo suficientemente pequeña como para poder ignorarla.

El modelo también supone que la CPU puede fallar, en cuyo caso simplemente para. Cualquier escritura en disco que esté en progreso en el momento del fallo también se detendrá, dejando datos incorrectos en un sector y un ECC incorrecto que más tarde podrá detectarse. Bajo todas esas condiciones puede conseguirse un sistema de almacenamiento estable 100% fiable en el sentido de que las escrituras o bien funcionan de forma correcta, o dejan intactos los datos que había anteriormente. Por supuesto que esto no nos protege frente a desastres físicos, tales como que se produzca un terremoto que provoque que el ordenador caiga por una grieta de 100 metros y se hunda en un río de lava hirviente. Mediante software resulta imposible recuperarse de una situación así.

El almacenamiento estable utiliza un par de discos idénticos en los que los bloques correspondientes colaboran para formar un bloque libre de errores. En ausencia de errores, los bloques correspondientes en ambas unidades son iguales. Leyendo cualquiera de los dos bloques se obtiene el mismo resultado. Para conseguir el almacenamiento estable, se definen las siguientes tres operaciones:

1. **Escrituras estables.** Una escritura estable consiste en escribir primero el bloque en la unidad 1, y luego leerlo para verificar que se escribió correctamente. Si eso no fue así, se repite la escritura y la posterior lectura *n* veces hasta que funcionen. Tras *n* fracasos consecutivos, el bloque se remapea en un bloque de repuesto, repitiéndose la operación hasta que se tenga éxito, sin importar cuántos bloques de repuesto sea preciso utilizar. Una vez que se ha logrado escribir correctamente en la unidad 1, se escribe y se relee el bloque correspondiente en la unidad 2, reintentándolo varias veces si es necesario, hasta lograrlo. En ausencia de fallos de la CPU, al terminarse la escritura estable se habrá escrito correctamente el bloque y se habrá verificado en ambas unidades.

2. **Lecturas estables.** Una lectura estable lee primero el bloque de la unidad 1. Si eso produce un ECC incorrecto, la lectura vuelve a intentarse hasta *n* veces. Si en todas esas lecturas se obtiene un ECC incorrecto, se lee el bloque correspondiente de la unidad 2. Dado el hecho de que una escritura estable con éxito deja dos copias correctas del bloque, y dada nuestra suposición de que es insignificante la probabilidad de que el mismo bloque se estropee espontáneamente en ambas unidades dentro de un intervalo de tiempo razonable, una lectura estable siempre tiene éxito.

3. **Recuperación después de caídas.** Después de una caída del sistema, un programa de recuperación explora ambos discos comparando bloques correspondientes. Si un par de bloques son correctos e iguales, no se hace nada. Si uno de ellos tiene un error de ECC, el bloque erróneo se sobrescribe con el bloque correcto correspondiente. Si un par de bloques son aparentemente correctos pero diferentes, el bloque de la unidad 1 se escribe en la unidad 2.

En ausencia de fallos de la CPU, este esquema siempre funciona ya que las escrituras estables siempre escriben dos copias válidas de cada bloque y se supone que los errores espontáneos nunca ocurren en ambos bloques correspondientes al mismo tiempo. Pero, ¿qué sucede si la CPU falla durante una escritura estable? Todo depende del momento preciso en que haya ocurrido el fallo. Hay cinco posibilidades, que se ilustran en la Figura 5-30.

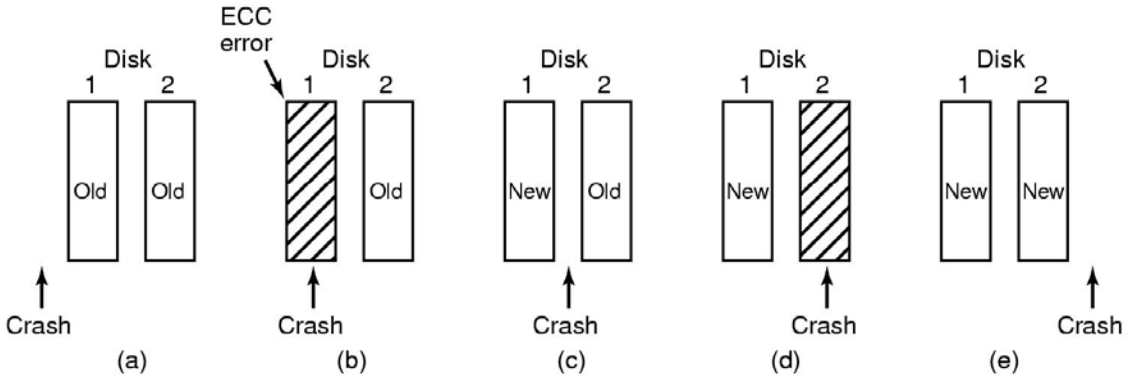

Figura 5-30. Análisis de la influencia de las caídas del sistema sobre las escrituras estables.

En la Figura 5-30(a), el fallo de la CPU se presenta antes de que se escriba cualquiera de las copias del bloque. Durante la recuperación no se modifica ninguna de ellas por lo que seguirá existiendo el antiguo valor, lo cual es correcto.

En la Figura 5-30(b), la CPU falla durante la escritura en la unidad 1, destruyendo los contenidos del bloque. Sin embargo, el programa de recuperación detecta este error y restaura el bloque de la unidad 1 a partir del bloque correspondiente de la unidad 2. Por tanto el efecto del fallo se anula, restaurándose completamente el estado anterior a la primera escritura.

En la Figura 5-30(c), el fallo de la CPU se presenta después de que se ha escrito en la unidad 1 pero antes de escribir en la unidad 2. En ese momento se ha rebasado ya el punto de no retorno: el programa de recuperación copia el bloque de la unidad 1 a la unidad 2, y la escritura se completa con éxito.

La Figura 5-30(d) es similar a la Figura 5-30(b): durante la recuperación, el bloque defectuoso se sobrescribe con el bloque correcto de la unidad 1. Otra vez, el valor final de ambos bloques es el nuevo.

Finalmente, en la Figura 5-30(e) el programa de recuperación ve que ambos bloques son iguales, por lo que no modifica ninguno, completándose con éxito la escritura.

Son posibles varias optimizaciones y mejoras sobre este esquema. Para empezar, es factible, pero costoso, comparar todos los bloques por pares después de un fallo. Una mejora enorme consiste en mantenerse al tanto de qué bloque se está escribiendo durante una escritura

estable, de modo que sólo sea necesario verificar un bloque durante la recuperación. Algunos ordenadores cuentan con una pequeña cantidad de **RAM no volátil** que es una memoria CMOS especial alimentada por una batería de litio. Tales baterías duran años, posiblemente toda la vida útil del ordenador. A diferencia de la memoria principal, cuyo contenido se pierde después de un fallo, el contenido de la RAM no volátil no se pierde en esos casos. Normalmente se guarda en ella la hora del día (incrementándose automáticamente mediante un circuito especial), y gracias a esto es por lo que los ordenadores siguen manteniendo la hora correcta incluso después de apagarse.

Supongamos que se cuenta con unos cuantos bytes de RAM no volátil que el sistema operativo puede usar para sus fines. La escritura estable puede colocar en la RAM no volátil el número del bloque que está a punto de actualizar, antes de iniciar la escritura. Una vez terminada con éxito la escritura estable, el número de bloque en la RAM no volátil se sobrescribe con un número de bloque inválido, por ejemplo -1. Bajo estas condiciones, después de una caída del sistema el programa de recuperación podrá consultar la RAM no volátil para saber si se estaba en medio de una escritura estable durante el fallo y, en tal caso, saber exactamente qué bloque se estaba escribiendo cuando se cayó el sistema. A continuación puede verificarse ya la corrección y consistencia de las dos copias del bloque.

Si no se cuenta con RAM no volátil, de todas maneras puede simularse como sigue. Al principio de una escritura estable, se sobrescribe un bloque de disco fijado de la unidad 1 con el número del bloque que se quiere escribir de manera estable. Luego se lee ese bloque para verificarlo. Una vez que se comprueba que es correcto se escribe y verifica el bloque correspondiente en la unidad 2. Cuando termine correctamente la escritura estable, ambos bloques se sobrescriben con un número de bloque inválido y se verifican. Después de un fallo, también será fácil determinar con este esquema si se estaba en medio de una escritura estable durante la caída. Por supuesto, esta técnica requiere ocho operaciones de disco adicionales para escribir un bloque estable, por lo que sólo deberá utilizarse cuando de verdad sea indispensable.

Vale la pena destacar un último punto. Supusimos que sólo uno de los bloques de un par de bloques dado puede estropearse espontáneamente en un mismo día. Si pasan suficientes días, podría estropearse también el otro bloque. Por tanto, debe efectuarse una exploración completa de ambos discos una vez al día, reparando cualquier daño que se detecte. De esta manera, al comienzo de cada mañana ambos discos siempre serán idénticos. Incluso si ambos bloques de un par se estropean dentro de un periodo de unos pocos días, se conseguirá reparar todos los errores correctamente.

5.10 INVESTIGACIÓN SOBRE ENTRADA/SALIDA

Se están realizando numerosas investigaciones sobre la entrada/salida, pero la mayoría de ellas se concentran sobre dispositivos específicos, y no sobre la E/S en general. A menudo el objetivo es mejorar el rendimiento de una forma u otra.

Los sistemas de disco son un ejemplo a destacar. Los algoritmos de planificación del brazo del disco más antiguos utilizan un modelo de disco que realmente ya no es aplicable, así Worthington y otros (1994) echaron un vistazo a los modelos que corresponden a los discos modernos. RAID es un tema de moda del que se están ocupando muchos investigadores en relación con diversos aspectos de estos sistemas. Álvarez y otros (1997) estudiaron la forma de mejorar la tolerancia a fallos, como hizo Blaum y otros (1994). Cao y otros (1994) examinaron la idea de tener un controlador paralelo en un RAID. Wilkes y otros (1996) describieron un sistema RAID avanzado que construyeron para HP. Tener múltiples unidades de disco requiere una buena planificación paralela, por lo que también se está investigando este tema (Chen y Towsley, 1996; y Kallahalla y Varman, 1999). Lumb y otros (2000) han argumentado a favor de precargar datos durante el tiempo que transcurre desde que el brazo termina su posicionamiento hasta que el sector requerido pasa por debajo de la cabeza. Incluso mejor que aprovechar el tiempo de latencia rotacional para hacer trabajo útil es eliminar completamente la rotación utilizando un dispositivo de almacenamiento microelectromecánico de estado sólido (Griffin y otros., 2000; y Carley y otros, 2000) o almacenamiento holográfico (Orlov, 2000). Otra tecnología nueva que hay que tener muy presente es el almacenamiento magneto óptico (McDaniel, 2000).

El terminal SLIM ofrece una versión moderna del antiguo sistema de tiempo compartido, con toda la computación realizándose de forma centralizada y poniendo a disposición de los usuarios terminales que simplemente gestionan la pantalla, el ratón, el teclado, y nada más (Schmidt y otros, 1999). La diferencia principal con respecto al tiempo compartido de los viejos tiempos es que en lugar de conectar el terminal al ordenador mediante un modém de 9600 bps, se utiliza una Ethernet a 10 Mbps, que proporciona suficiente ancho de banda para una interfaz gráfica completa a disposición del usuario.

Las GUIs se han estandarizado en gran medida, pero todavía se está trabajando mucho en ese campo, por ejemplo en el uso de entradas de voz (Malkewitz, 1998; Manaris y Harkreader, 1998; Slaughter y otros, 1998; y Van Buskirk y LaLomia, 1995). La estructura interna de la GUI es también tema de investigación (Taylor y otros, 1995).

Dado el gran número de ciéntíficos informáticos que tienen ordenadores portátiles y dada la microscópica duración de la batería en la mayoría de ellos, no debe sorprendernos el gran interés que hay en utilizar técnicas de software para administrar y ahorrar la energía de las baterías (Ellis, 1999; Flinn y Satyanarayanan, 1999; Kravets y Krishnan, 1998; Lebeck y otros, 2000; Lorch y Smith, 1996, y Lu y otros, 1999).

5.11 RESUMEN

La entrada/salida es un tema que se descuida a menudo, a pesar de su importancia. Una gran parte de cualquier sistema operativo corresponde a la E/S. Hay tres formas de realizar E/S. En primer lugar está la E/S programada, en la cual la CPU principal transmite o recibe cada byte o palabra, debiendo a continuación esperar dando vueltas dentro de un pequeño bucle hasta que sea posible enviar o recibir el siguiente byte. En segundo lugar está la E/S dirigida por interrupciones, en la cual la CPU inicia una transferencia de E/S para un carácter o palabra, pasando a hacer otra cosa hasta que llega una interrupción que le avisa de que ya terminó la E/S. En tercer lugar está el DMA, en el cual un chip aparte gestiona la transferencia completa de todo un bloque de datos, produciendo una única interrupción cuando termina la transferencia de todo el bloque.

La E/S puede estructurarse en cuatro niveles: las rutinas de tratamiento de las interrupciones, los drivers de los dispositivos, el software de E/S independiente del dispositivo y las bibliotecas de E/S y los programas de *spooling* que se ejecutan en el espacio de usuario. Los drivers de los dispositivos se encargan de los detalles de operación de los dispositivos, así como de presentar interfaces uniformes al resto del sistema operativo. El software de E/S independiente del dispositivo se encarga de cosas como informar de los errores o disponer los búferes que resulten convenientes.

Hay diversos tipos de discos, incluyendo discos magnéticos, RAIDs y varios tipos de discos ópticos. En muchos casos es posible utilizar algoritmos de planificación del brazo del disco para mejorar el rendimiento, pero la presencia de geometrías virtuales complica mucho las cosas. Si se emparejan dos discos, puede construirse un medio de almacenamiento estable con ciertas propiedades útiles.

Los relojes sirven para llevar el control del tiempo real, limitar el tiempo de ejecución de los procesos, implementar temporizadores vigilantes y llevar la contabilidad de los recursos utilizados.

Los terminales orientados a caracteres tienen varios aspectos relacionados con los caracteres especiales que pueden introducirse y las secuencias de escape especiales que pueden generarse. Las entradas pueden estar en modo crudo o en modo elaborado, dependiendo del grado de control que quiera tener el programa sobre ellas. Las secuencias de escape en la salida controlan el movimiento del cursor y permiten insertar y borrar texto en la pantalla.

Muchos ordenadores personales utilizan GUIs para visualizar sus salidas. Estas interfaces se basan en el paradigma WIMP: ventanas, iconos, menús y dispositivo señalador. Los programas basados en GUI están controlados normalmente por eventos, siendo los eventos de teclado, de ratón y de otro tipo enviados al programa para que los procese tan pronto como ocurran.

Hay varios tipos de terminales de red. Uno de los más populares es el que ejecuta X, un sistema sofisticado que puede utilizarse para construir diversas GUIs. Una alternativa a X Windows es una interfaz de bajo nivel que simplemente envía píxeles crudos a través de la red. Los experimentos realizados con el terminal SLIM muestran que esta técnica produce un rendimiento sorprendentemente bueno.

Por último, la administración de la energía es una cuestión principal para los ordenadores portátiles debido a la limitada duración de las baterías. El sistema operativo puede utilizar diversas técnicas para reducir el consumo de energía. Los programas también pueden ayudar sacrificando algo en la calidad del servicio a cambio de que las baterías duren más.