APUNTES DE

Introducción a los Sistemas Operativos

2º DE GRADO EN INGENIERÍA INFORMÁTICA

TEMA 6. GESTIÓN DE LA E/S

CURSO 2021/2022

ÍNDICE GENERAL

CAPÍTULO 6

GESTIÓN DE LA E/S

ESQUEMA DE CONTENIDOS

- Dispositivos de bloques y caracteres.
- Objetivos del software de E/S.
- Arquitectura del software de E/S.
- Planificación de disco.
- Discos RAM.
- Relojes programables.
- Funciones del manejador de reloj.

Desde el punto de vista del sistema operativo, nos interesa la programación de los dispositivos de entrada/salida (E/S), no su diseño, construcción o mantenimiento. Por ello, nos fijaremos en la interfaz que presenta el hardware al software: órdenes que acepta el hardware, funciones que realiza y errores de los que puede informar.

6.1 DISPOSITIVOS DE E/S

Los dispositivos de E/S se pueden dividir de manera general en dos categorías: dispositivos de bloques y dispositivos de caracteres. Un *dispositivo de bloques* almacena la información en bloques de tamaño fijo, cada uno con su propia dirección. La propiedad esencial de estos dispositivos es la posibilidad de leer de o escribir en un bloque de forma independiente de los demás, por lo que existe una operación de búsqueda que nos permite indicar el bloque al que se va a acceder. Ejemplos de dispositivos que se clasifican dentro de esta categoría son los discos y otros muchos dispositivos de almacenamiento secundario.

Un *dispositivo de caracteres*, por su parte, envía o recibe un flujo de caracteres, sin atenerse a una estructura de bloques. En estos dispositivos no es posible utilizar direcciones ni hay una operación de búsqueda. Las terminales, impresoras, interfaces de red, ratones y otros muchos dispositivos son de caracteres.

A veces, la frontera entre los dispositivos que se manejan mediante direcciones de bloque y los que no no está bien definida. Por ejemplo, una cinta es un dispositivo de caracteres que podría funcionar como un disco, aunque de forma mucho más lenta, ya que es posible rebobinar la cinta y avanzar después hasta cierta posición.

También hay dispositivos que no se adaptan a esta división. Un ejemplo son los relojes, que no tienen direcciones de bloques ni aceptan o generan flujos de caracteres; solo producen interrupciones periódicas. Otro ejemplo son las tarjetas gráficas mapeadas a memoria, en las que la memoria de vídeo es parte del espacio de direcciones de la memoria principal del ordenador.

Aun así, este modelo que divide los dispositivos de E/S en dispositivos de bloques y de caracteres es lo bastante general como para ser utilizado por gran parte del software del sistema operativo para trabajar de forma independiente del dispositivo. Por ejemplo, el sistema de ficheros solo trabaja con dispositivos abstractos de bloques, y encarga la parte que depende del dispositivo a un software de menor nivel llamado manejador de dispositivo, como veremos en la siguiente sección.

6.2 PRINCIPIOS DEL SOFTWARE DE E/S

La idea básica es organizar el software de E/S como una serie de capas, en donde las capas inferiores se encarguen de ocultar las peculiaridades del hardware a las capas superiores de forma que estas últimas se preocupen por presentar una interfaz limpia, agradable y regular a los usuarios. Analicemos estas metas y la forma de llegar a ellas.

6.2.1 Objetivos del software de E/S

El software de E/S persigue la consecución de los siguientes 5 objetivos:

1. *Independencia de dispositivo*: el sistema operativo debe crear conceptos suficientemente generales que no dependan de las características particulares de cada dispositivo. Por ejemplo, el sistema operativo puede crear sobre los dispositivos de bloques (discos duros, discos SSD, memorias USB, CD-ROMs, DVDs, etc.) el concepto de sistema de ficheros, con una estructura bien definida y operaciones para manejar dicha estructura. Para el programador, el sistema de ficheros se comporta siempre de la misma forma sin importar si dicho sistema de ficheros se encuentra sobre un CD-ROM (en un lector que tiene un motor que se debe arrancar y parar) o un disco duro (que está girando continuamente). De esta forma, es posible escribir programas que puedan acceder a ficheros en cualquier dispositivo de bloques sin tener que modificar los programas para cada tipo de dispositivo.

2. *Nombres uniformes*: relacionado con el objetivo anterior, el nombre del fichero o dispositivo debe ser simplemente una cadena o un entero y no depender de las características del dispositivo. Por ejemplo, en Linux, el fichero `/dev/sda` representa y da nombre al primer disco duro disponible, sin importar qué características tenga este. También, el hecho de que muchos dispositivos se representen mediante ficheros especiales de bloques y caracteres permite acceder a ellos a través de un concepto bien conocido como es el de fichero.

3. *Manejo de errores*: los errores deben manejarse lo más cerca posible del hardware. Si el controlador descubre un error, debe tratar de corregirlo en la medida de lo posible. Si no puede, debe tratar de corregirlo el manejador de dispositivo. Solo en el caso de que las capas inferiores no puedan resolver el problema, se debe informar a los niveles superiores.

4. *Conversión de las transferencias asíncronas en síncronas*: la mayor parte de la E/S es asíncrona, es decir, la CPU inicia una transferencia y realiza otras cosas

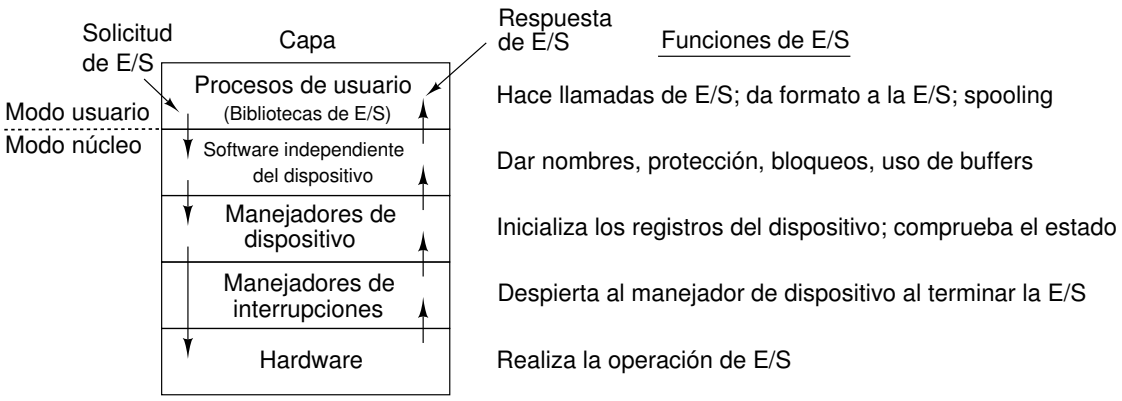

Figura 6.1. Capas del sistema de E/S y las principales funciones de cada capa. Las flechas muestran el flujo de control.

hasta que se produce una interrupción que indica que la transferencia ha terminado. Sin embargo, los programas del usuario son mucho más fáciles de escribir si las operaciones de E/S son síncronas, es decir, bloqueantes; por ejemplo, tras una llamada a `read()`, el programa se bloquea hasta que los datos están disponibles en el buffer. En resumen, el sistema operativo debe hacer que operaciones controladas por interrupciones parezcan síncronas para el usuario.

5. *Compartición de recursos*: hay dispositivos que se pueden compartir, como los discos (varios usuarios pueden tener ficheros abiertos en el mismo disco al mismo tiempo) y dispositivos de uso exclusivo, como las impresoras (un usuario que toma el control de la impresora debe liberarla solo cuando haya terminado de imprimir todo su trabajo). El sistema operativo debe permitir ambas posibilidades.

Una forma de lograr estos objetivos es estructurando el software de E/S en cuatro capas, como se puede ver en la figura 6.1. Observa cómo los manejadores de interrupciones no intervienen en las peticiones de E/S, pero sí en las respuestas. En los siguientes apartados vamos a describir en detalle cada una de estas capas y qué objetivo consigue cada una.

6.2.2 Manejadores de interrupciones

Las interrupciones son difíciles de manejar porque se pueden producir en cualquier momento. Por ello, deben esconderse en lo más profundo del sistema operativo de forma que solo una pequeña parte de él sepa de su existencia. A esta parte se le llama *manejador de interrupciones*. Una forma de ocultarlas es la siguiente:

- Un proceso que inicia una operación de E/S se bloquea hasta que termine la E/S y ocurra la interrupción. Un proceso puede bloquearse a sí mismo, mediante una llamada al sistema especial, o ser bloqueado de forma automática por el propio sistema operativo. Este segundo caso es el de Unix en el que un proceso se bloquea en su parte del núcleo cediendo la CPU a otro proceso.

- Al producirse la interrupción, el manejador de interrupciones la atiende para saber qué operación de E/S ha hecho que se produzca la interrupción. Después, avisa al manejador de dispositivo correspondiente (ver sección 6.2.3) para que elimine el bloqueo del proceso que inició la operación, de forma que este pueda continuar su ejecución. En el caso de Unix, el proceso continuará en su parte del núcleo para terminar de llevar a cabo la operación de E/S antes de volver a su parte de usuario.

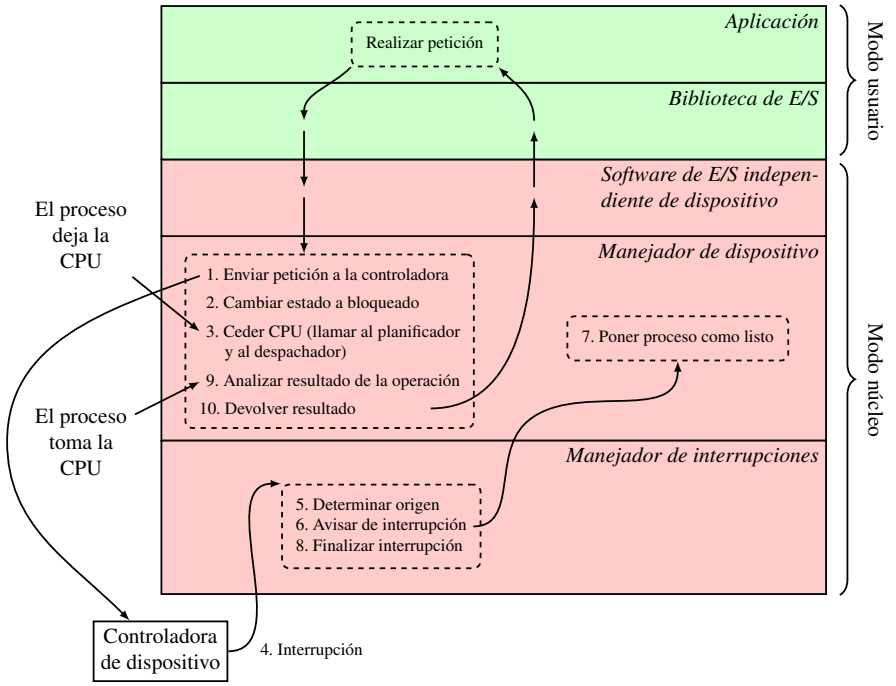

Figura 6.2. Camino seguido por una petición y tratamiento de su interrupción.

Por tanto, el efecto real de la interrupción será que un proceso antes bloqueado podrá continuar su ejecución, por lo que este proceso en ningún caso será consciente de la existencia de dicha interrupción (ver figura 6.2).

6.2.3 Manejadores de dispositivo

Todo el código que depende del funcionamiento concreto de los dispositivos aparece en los *manejadores de dispositivo* o *drivers*. Cada uno de estos manejadores controla solo un tipo de dispositivo o, a lo más, una clase de dispositivos similares.

Los manejadores de dispositivo son los que envían órdenes a las tarjetas controladoras y verifican su ejecución adecuada. Así, el manejador del disco es la única parte del sistema operativo que conoce el número de registros de la tarjeta controladora de disco y el uso que tienen estos. Para poder acceder a los registros de las controladoras, los manejadores necesitan ejecutarse en modo núcleo (al menos, en los sistemas operativos monolíticos) por lo que, desde un punto de vista lógico, se les considera parte del núcleo del sistema operativo. Generalmente, el fabricante de un dispositivo proporciona manejadores para diferentes sistemas operativos, aunque también hay manejadores que el propio sistema operativo ya incluye en su código.

La implementación de un manejador de dispositivo no es una tarea sencilla pues supone conocer parte de las interioridades del sistema operativo y el funcionamiento exacto del dispositivo correspondiente. Esta implementación se complica si tenemos en cuenta que muchos manejadores deben ser *reentrantes*, ya que, durante su ejecución para atender la petición de un proceso, es posible que un manejador se vuelva a ejecutar para atender las peticiones de otros procesos.

En general, la función de un manejador de dispositivo es la de aceptar las solicitudes abstractas que le hace el software independiente de dispositivo (ver sección 6.2.4) y verificar la ejecución de dichas solicitudes. Por ejemplo, en el caso de un disco, una solicitud común puede ser leer o escribir el bloque n, ya que el manejador del disco hará que este se vea como un array lineal de bloques de igual tamaño (ver sección 6.3).

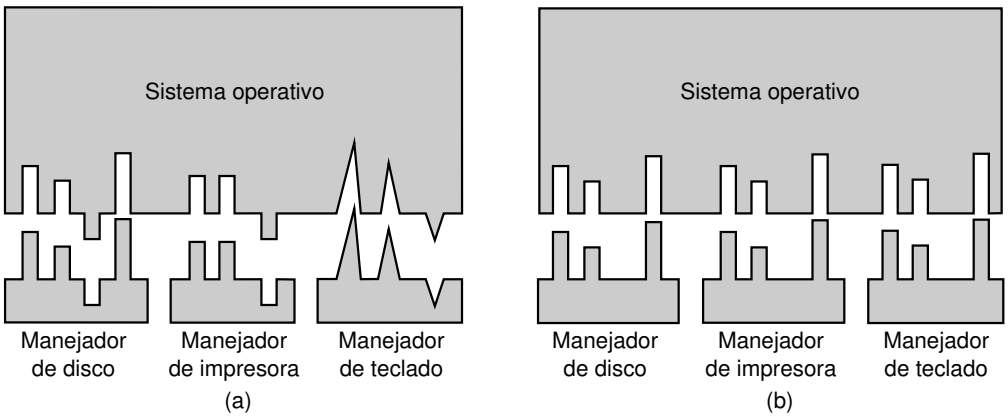

Figura 6.3. (a) Sistema operativo sin una interfaz uniforme de manejadores. (b) Sistema operativo con una interfaz uniforme de manejadores.

Después de enviar la orden (u órdenes), pueden darse dos casos: que el manejador deba esperar hasta que el controlador realice cierto trabajo, bloqueándose hasta que ocurra una interrupción, o que la operación termine sin retraso, por lo que el manejador no se bloquea. Tras terminar la operación, el manejador debe verificar los errores (corrigiendo los que pueda) para informar después convenientemente al software independiente del dispositivo. Si existen otras solicitudes pendientes, debe seleccionar e iniciar alguna de ellas; si no, se puede bloquear en espera de una solicitud (todo esto depende, como siempre, del diseño del sistema operativo).

Un último aspecto a tener en cuenta es que, para poder trabajar tanto con los dispositivos presentes como futuros, el sistema operativo necesita definir una interfaz a la que se deben adaptar todos sus manejadores. Esta interfaz está formada por una serie de funciones genéricas que el resto del sistema operativo puede invocar para pedirle a un cierto manejador que realice una determinada operación de E/S sobre el dispositivo que gestiona. De esta manera, para poder trabajar con un nuevo dispositivo que se conecte al sistema, basta con que el manejador del mismo se adapte a la interfaz definida por el sistema operativo, ya que, al hacerlo así, el nuevo dispositivo se podrá usar a través de las mismas funciones que ya se usan para el resto de dispositivos. Esta idea se muestra en la figura 6.3(b) donde se ve que los manejadores de los diferentes dispositivos se adaptan a una misma interfaz. La figura 6.3(a), en cambio, muestra el caso de un sistema operativo que no define una misma interfaz para todos los manejadores. Los problemas de no definir una interfaz estándar son, entre otros, que las funciones que se usan para invocar a un manejador cambian de uno a otro y que la programación de un nuevo manejador para el sistema operativo conlleva un esfuerzo considerable.

6.2.4 Software de E/S independiente de dispositivo

Aunque una parte del software de E/S es dependiente de los dispositivos que gestiona, una gran parte de él es independiente. La frontera exacta entre los manejadores y el software independiente de dispositivo (SID) depende del sistema, puesto que algunas de las funciones que puede llevar a cabo el SID se ejecutan en realidad en los manejadores por razones de eficiencia u otros motivos. Las funciones realizadas generalmente por el SID son:

- *Interfaz uniforme de E/S para el usuario*: a pesar de tener dispositivos y manejadores distintos, se pueden crear dispositivos abstractos para que el usuario acceda

a los dispositivos reales siempre de la misma forma. El SID selecciona el manejador adecuado en cada caso. En el caso de Unix, esta interfaz uniforme se consigue mediante el concepto de fichero, ya que muchos dispositivos se representan mediante ficheros en los que se puede leer o escribir, lo que en última instancia provoca una lectura o escritura en el dispositivo real.

- *Nombres de los dispositivos*: el SID se encarga de asociar el nombre simbólico de cada dispositivo con el manejador adecuado. En el caso de Unix, esto se hace a través de los ficheros especiales de dispositivo. Por ejemplo, `/dev/tty0` es un fichero especial de caracteres cuyo nodo-i contiene un *número mayor de dispositivo* y un *número menor de dispositivo*. Cuando un proceso abre este fichero, el sistema operativo utiliza el número mayor para localizar el manejador que debe atender las operaciones sobre el dispositivo asociado al fichero. El manejador, a su vez, utiliza el número menor para determinar el dispositivo exacto a leer o escribir (en este caso, una terminal).

- *Protección de los dispositivos*: se debe evitar que los usuarios no autorizados accedan a los dispositivos. En el caso de Unix, esta protección se consigue a través de los permisos, propietario y grupo asociados a los ficheros especiales de bloques y caracteres.

- *Proporcionar un tamaño de bloque independiente del dispositivo*: por ejemplo, se pueden agrupar o dividir sectores para conseguir un tamaño único de bloque lógico. Esta tarea también podría ser realizada por el manejador de dispositivo.

- *Uso de buffers*: el sistema operativo debe proporcionar almacenamiento temporal en memoria para los dispositivos de E/S. Hay distintas razones para ello. Por ejemplo, hay dispositivos de caracteres, como el teclado, que no son capaces de almacenar información o que envían información antes de que ningún proceso la haya solicitado. En este caso, el sistema operativo debe guardar los códigos de las teclas pulsadas para que estén disponibles cuando un proceso decida leer de teclado. Si no se hiciera así, muchas pulsaciones de teclas se perderían, lo que haría que el teclado fuera un dispositivo incómodo de usar. También hay dispositivos, como las impresoras, que no son capaces de procesar la información a la velocidad a la que un proceso puede enviarla, por lo que será necesario el uso de buffers para guardar temporalmente la información e ir enviándola poco a poco.

  En el caso de los discos, interesa que ciertos bloques se almacenen temporalmente en memoria para acelerar su funcionamiento. Además, aunque las transferencias entre disco y memoria se hagan en unidades de bloque, debemos permitir que un proceso pueda leer o escribir en un fichero una cantidad arbitraria de bytes. Así, si un proceso decide leer de un fichero byte a byte, el sistema operativo necesitará un buffer para almacenar cada bloque del fichero en memoria y luego ir dándole la información del bloque al proceso en las cantidades que este decida. De la misma manera, si el proceso decide escribir byte a byte, el bloque tendrá que llenarse en memoria antes de ser escrito en disco.

- *Asignación y liberación de los dispositivos de uso exclusivo*: hay dispositivos, como las impresoras, que no deben ser usados por dos procesos a la vez. En estos casos, el sistema operativo examina las solicitudes de uso del dispositivo y las acepta o rechaza según la disponibilidad del dispositivo solicitado.

- *Informe de errores*: el manejo de errores lo realizan los manejadores, pero cuando un manejador no puede solucionar un error, deja su manejo al SID, que lo solucionará y/o informará de él.

Un componente importante del software independiente de dispositivo es la parte del sistema operativo que gestiona los sistemas de ficheros. Este componente trabaja sobre dispositivos abstractos de bloques proporcionados por los manejadores de dispositivo. Por lo tanto, para su funcionamiento, realmente no necesita conocer las características particulares de cada dispositivo. Sobre esos dispositivos abstractos de bloques, el sistema de ficheros implementa ficheros y directorios (tal y como vimos en el tema 4) que serán usados por los procesos de usuario para leer de o escribir información en los dispositivos físicos de bloques presentes en el sistema.

6.2.5 Software de E/S en el espacio de usuario

Aunque la mayoría del software de E/S está dentro del sistema operativo (ejecutado en modo núcleo), una pequeña parte de él está fuera y se ejecuta en modo usuario. Dos ejemplos de esto son los procedimientos de biblioteca de E/S y el sistema de *spooling*.

Bibliotecas de E/S

Algunas de las bibliotecas que se enlazan con los programas y se ejecutan en modo usuario contienen procedimientos que se encargan de llevar a cabo las llamadas al sistema de E/S, del mismo modo que hay otros procedimientos de biblioteca que gestionan otras llamadas al sistema.

Aunque hay procedimientos de biblioteca de E/S que hacen poco más que poner sus parámetros en el lugar apropiado para realizar la llamada al sistema, como pueden ser las funciones `read()` y `write()`, hay otros que realmente efectúan cierto trabajo, como `printf()`, que permite dar un cierto formato a la salida.

Sistema de spooling

El *spooling* es una forma de trabajar con los dispositivos de E/S de uso exclusivo en un sistema de multiprogramación que evita la posible monopolización de esos recursos por parte de un proceso de usuario.

Para entender su funcionamiento, consideremos como ejemplo una impresora. Aunque es fácil, desde el punto de vista técnico, dejar que los procesos de usuario abran ellos mismos el fichero especial de caracteres correspondiente a la impresora para enviarle la información a imprimir, puede ocurrir que un proceso abra el fichero y lo mantenga abierto durante horas sin llevar a cabo ninguna actividad. Durante ese tiempo, ningún otro proceso podrá escribir nada en la impresora, ya que, al ser un dispositivo de uso exclusivo, el sistema operativo rechazará cualquier otra solicitud de uso del mismo. En vez de esto, lo que se puede hacer es crear un proceso especial, llamado *demonio* (de impresión, en este ejemplo), y un directorio especial, llamado *directorio de spooling*. Para imprimir un fichero, el proceso genera primero un fichero con todo lo que quiere imprimir y lo pone en el directorio de *spooling*. Por su parte, el demonio, que es el único proceso con permiso para utilizar el fichero especial que representa a la impresora, verifica periódicamente si hay algún trabajo para imprimir en el directorio y, si es así, lo selecciona y lo imprime.

El sistema de *spooling* tiene también como ventaja que permite hacer una gestión de los trabajos (ficheros) que deben ser procesados por el demonio, ya que se puede

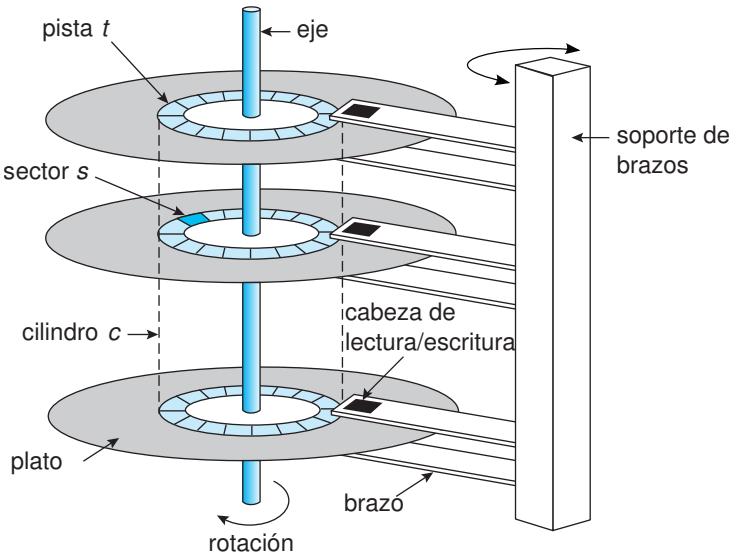

Figura 6.4. Mecanismo de un disco de cabezas móviles.

eliminar un trabajo antes de que sea procesado, se puede cambiar el orden en el que los trabajos deben ser tratados, etc.

6.3 DISCOS

Dentro de los dispositivos de E/S, los discos duros son de los más importantes, ya que permiten almacenar grandes cantidades de información de forma permanente y a un precio por gigabyte muy bajo. En esta sección vamos a estudiar su funcionamiento, aunque sin entrar en muchos detalles, y vamos a ver cómo podemos planificar las peticiones de lectura/escritura que lleguen a un disco para mejorar el rendimiento.

6.3.1 Estructura física

En un disco duro los datos se graban sobre una serie de *discos* o *platos* magnéticos. Estos discos están conectados a un eje común que gira a una velocidad muy alta, por ejemplo, 5400 vueltas o revoluciones por minuto (RPM) (ver figura 6.4).

Se accede a los datos (se leen o se escriben) mediante una serie de *cabezas de lectura/escritura*, una por cada superficie de disco. Cada cabeza solo puede acceder a los datos inmediatamente adyacentes a ella, por lo que, para leer o escribir una cierta porción de la superficie de uno de los platos, dicha porción debe estar justo debajo (o encima) de la correspondiente cabeza de lectura/escritura. El tiempo que le lleva a los datos girar desde la posición en que la se encuentran hasta la posición de la cabeza que los lee o escribe se llama *tiempo de latencia*. Una vez los datos llegan a la cabeza, estos se tienen que transferir desde o hacia la controladora, según se trate de una operación de escritura o lectura, respectivamente. Al tiempo necesario para hacer esta transferencia se le llama *tiempo de transmisión*.

Cada una de las diferentes cabezas de L/E, mientras está fija en una posición, determina una *pista* circular de datos sobre la superficie del disco que le corresponde. Las cabezas se encuentran sobre brazos los cuales están fijados a un mismo soporte, lo que hace que todos los brazos se muevan a la vez. El soporte de los brazos puede girar una pequeña cantidad de grados en ambos sentidos, haciendo que las cabezas se muevan hacia dentro o hacia fuera de los discos. Cuando el soporte de los brazos desplaza las

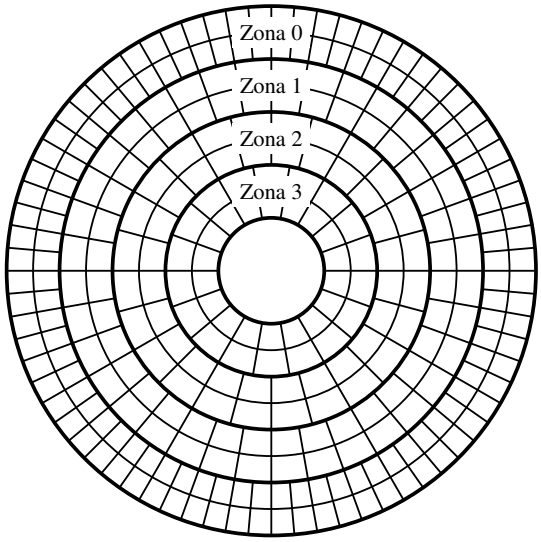

Figura 6.5. Zonas en las que se organizan las pistas en un disco duro moderno.

cabezas hacia una nueva posición, estas pueden tener acceso a un conjunto de pistas diferente. Para una posición dada de las cabezas, el conjunto de pistas definido por todas ellas se llama *cilindro*. Hay tantos cilindros como posiciones posibles de las cabezas. El proceso de desplazar las cabezas hacia un nuevo cilindro se conoce como *búsqueda*, y el tiempo que lleva el realizarla se denomina *tiempo de búsqueda*.

Como muestra también la figura 6.4, las pistas se dividen en *sectores*, que son la unidad mínima de lectura y escritura. Como ya vimos al hablar de sistemas de ficheros, los sectores suelen tener tamaños típicos de 512, 1024, 2048 o 4096 bytes.

Según la estructura que acabamos de describir, cada sector se identifica por la terna (nº de cilindro, nº de cabeza, nº de sector dentro de la pista), por lo que a veces un disco se considera un array tridimensional.

Aunque la figura 6.4 puede dar a entender que todas las pistas tienen el mismo número de sectores, en realidad, no es así, ya que las pistas exteriores son más largas que las interiores y, por tanto, contienen un mayor número de sectores por pista para aprovechar mejor la superficie de grabación. La figura 6.5 muestra cómo se organizan hoy en día las pistas de un disco. Podemos ver que existen varias zonas y que, dentro de una zona, todas las pistas tienen el mismo número de sectores.

Puesto que el número de sectores por pista es variable, el acceso a un sector por su terna (nº de cilindro, nº de cabeza, nº de sector dentro de la pista) se hace imposible, ya que eso supondría para el manejador de disco conocer la estructura interna de cada disco. Quien sí conoce la estructura interna de un disco es su controladora. Por eso, la solución pasa por que sea la propia controladora de cada disco, y no el manejador, la que exporte una interfaz que presente al disco como un array lineal de bloques. En este array, cada sector se identifica por su número LBA (*Logical Block Address*), que no es más que su posición dentro del array.

A pesar de esta división en zonas de los discos actuales, en lo que queda de sección supondremos que todas las pistas son del mismo tamaño, ya que esto simplifica la explicación sin restarle generalidad.

La estructura de un disco también se refleja en los pasos necesarios para acceder a él. Como muestra la figura 6.6, leer o escribir un sector supone realizar las siguientes tareas:

1. Mover el soporte de los brazos hasta colocar las cabezas en el cilindro adecuado

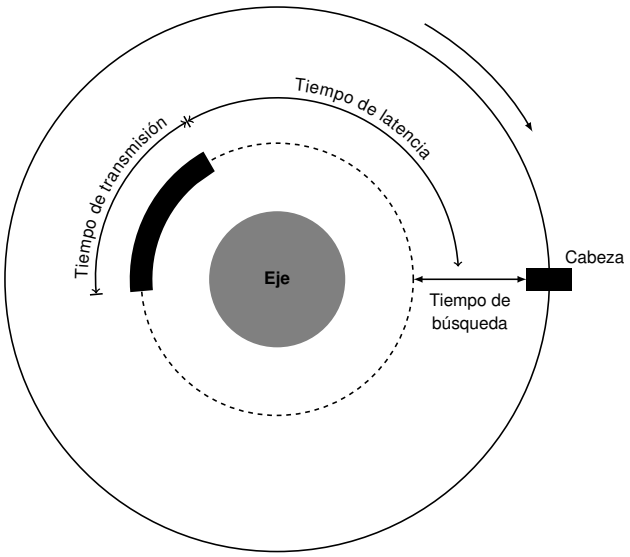

Figura 6.6. Componentes de un acceso a disco.

(tiempo de búsqueda).

2. Activar la cabeza adecuada (operación electrónica y, por tanto, de tiempo despreciable).

3. Esperar a que el disco gire para que los sectores a leer o escribir pasen junto a la cabeza (tiempo de latencia).

4. Leer o escribir los sectores en sí según van pasando junto a la cabeza (tiempo de transmisión).

El tiempo que va desde que se inicia una operación de disco hasta que esta termina se llama *tiempo de servicio* o *tiempo de acceso*. Este tiempo incluye los tiempos de búsqueda, latencia y transmisión en los que incurre la operación. De ellos, el que tiene un mayor peso es el tiempo de búsqueda, ya que la búsqueda del cilindro (es decir, el movimiento de las cabezas) suele ser el paso más costoso.

6.3.2 Planificación de los movimientos del brazo

Como acabamos de ver, de los tres tiempos que determinan un acceso a disco, el de búsqueda suele ser el más importante, por lo que la reducción del tiempo promedio de búsqueda o, lo que es lo mismo, la reducción de los movimientos de las cabezas, puede mejorar en gran medida el rendimiento del almacenamiento secundario al disminuir, a su vez, el tiempo promedio de servicio.

Los discos, como otros dispositivos de E/S, suelen tener una cola de solicitudes pendientes pues, mientras se procesa la solicitud de un proceso, pueden llegar otras solicitudes de otros procesos. Cada solicitud puede afectar a un cilindro distinto y las cabezas deberán desplazarse a dicho cilindro para procesarla. Nuestro objetivo, por tanto, será planificar el orden en el que se atienden las solicitudes pendientes para reducir el movimiento de las cabezas. Este objetivo debe ser logrado por el manejador del disco, que puede usar distintos algoritmos o planificadores para conseguirlo.

En los siguientes apartados se supone que el manejador conoce la geometría del disco, es decir, el número de cabezas, cilindros y sectores que posee, por lo que la planificación de las peticiones se hará teniendo en cuenta los cilindros a los que afecten. En

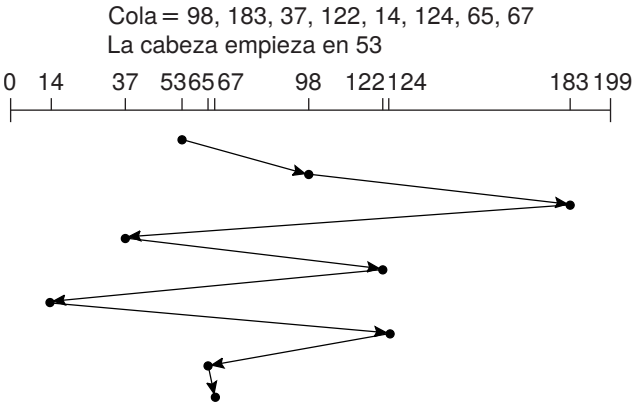

Figura 6.7. Planificación de disco FCFS.

un disco moderno, en cambio, esta geometría se desconoce, como ya hemos explicado en el apartado anterior. A pesar de ello, el funcionamiento de los algoritmos de planificación que vamos a ver a continuación sigue siendo aplicable a los discos actuales. La diferencia está en que, en lugar de tener en cuenta la distinta entre cilindros, se tiene en cuenta la distancia entre las direcciones de disco de las peticiones[1].

Planificación FCFS

La planificación FCFS (*First Come, First Served*) es la más sencilla, ya que se da servicio a las solicitudes según su orden de llegada. El problema es que puede dar lugar a tiempos de servicio en promedio bastante grandes. La figura 6.7 muestra un ejemplo de esta planificación, donde las cabezas se encuentran actualmente en el cilindro 53 y hay que atender la lista de peticiones que hay en la cola (observa que solo se indica el cilindro al que va a cada petición). Según este ejemplo, cuando se procesen todas las solicitudes, las cabezas habrán recorrido 640 cilindros. La figura nos muestra que, en algunos casos, las cabezas tienen que desplazarse un gran número de cilindros, como cuando se pasa del cilindro 183 al 37.

Planificación SSF

La planificación SSF (*Shortest Seek First*), o de la búsqueda más corta, atiende a la siguiente solicitud con cilindro más cercano al actual. Un ejemplo de esta planificación se puede ver en la figura 6.8, en donde las condiciones de partida son iguales a las del ejemplo anterior. Como se aprecia en la figura, al final las cabezas habrán recorrido 236 cilindros, que son casi tres veces menos que en la planificación FCFS.

Un problema de esta planificación es que es demasiado «local», ya que pueden llegar solicitudes que impliquen cilindros próximos al actual, por lo que estas solicitudes serán atendidas enseguida, mientras que otras que llegaron antes, con cilindros más alejados, no se atenderán. Por lo tanto, entran en conflicto las metas de reducir al mínimo el tiempo de servicio y de ser equitativos (es decir, tratar a todas las peticiones por igual y, por ende, a todos los procesos). Tampoco es un algoritmo óptimo. Así, si se hubieran

[1]Puesto que la controladora de un disco conoce la geometría exacta del mismo, también podría planificar las peticiones que recibe. Para ello, sería necesario que el disco pudiera recibir varias peticiones a la vez. Esto es precisamente lo que ocurre en los discos modernos que implementan la tecnología NCQ, siempre que el sistema operativo haya configurado el disco para que este la use durante el procesamiento de peticiones.

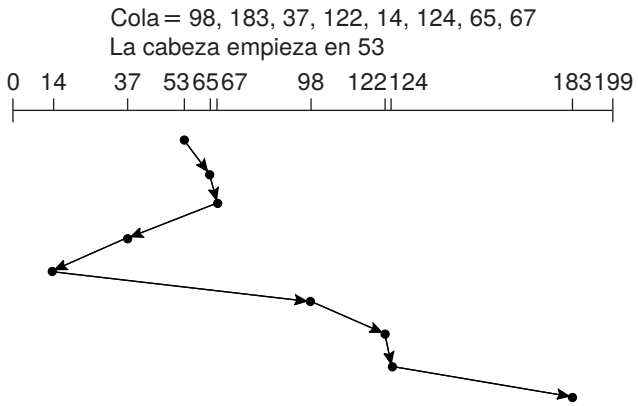

Figura 6.8. Planificación de disco SSF.

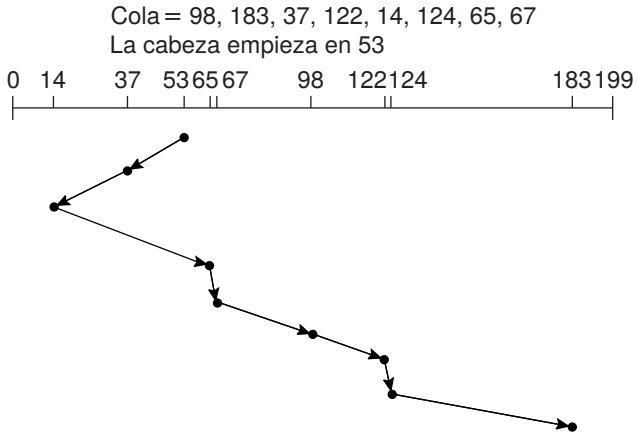

Figura 6.9. Planificación de disco SCAN.

atendido las solicitudes en el orden 37, 14, 65, 67, 98, 122, 124 y 183, las cabezas solo habrían recorrido 208 cilindros.

Planificación SCAN o algoritmo del ascensor

En la planificación SCAN, o algoritmo del ascensor, las cabezas de L/E se desplazan en un sentido dando servicio a las solicitudes que van encontrando en cada cilindro. Cuando no hay más solicitudes en ese sentido (o se llega al extremo), se invierte el sentido para hacer lo mismo otra vez. Por tanto, en este algoritmo es necesario tener un bit que indique el sentido del movimiento. La figura 6.9 muestra un ejemplo para esta planificación donde se ve que, en total, se recorren 208 cilindros.

Una propiedad interesante de este algoritmo es que, dada cualquier colección de solicitudes, la cuota máxima del total de movimientos está fijada: es el doble del número de cilindros.

En general, a pesar de que en el ejemplo dado se recorren menos cilindros que con SSF, esta planificación es peor, ya que suele producir más movimientos de las cabezas del disco. A cambio, la planificación SCAN tiene como ventaja que evita la localidad del algoritmo anterior.

Planificación C-SCAN (SCAN circular)

Un problema de la planificación SCAN (que se agrava en SSF) es que puede atender muchas solicitudes que acaban de llegar antes que otras que llevan en la cola un cierto

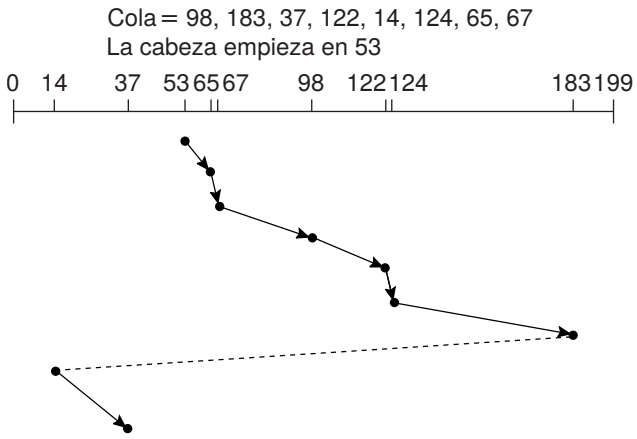

Figura 6.10. Planificación de disco C-SCAN.

tiempo. Por ejemplo, supongamos una distribución uniforme de solicitudes a lo largo del disco. Si las solicitudes se atienden según la planificación SCAN, entonces, cuando las cabezas lleguen a un extremo e inviertan el sentido, en la zona próxima a dicho extremo habrá pocas solicitudes que, además, habrán llegado hace poco, ya que los cilindros de dicha zona se atendieron recientemente. Estas solicitudes se servirán pronto a pesar de que la mayor densidad de solicitudes se encontrará en el extremo opuesto del disco, y serán también las que lleven más tiempo esperando a ser atendidas.

La planificación C-SCAN trata de ofrecer un tiempo de espera más uniforme, evitando el problema anterior. En esta planificación, las cabezas se mueven de un extremo del disco al otro, atendiendo las solicitudes que van encontrando. Al llegar al extremo opuesto, regresan de inmediato al otro extremo, sin servir ninguna solicitud, y en dicho extremo vuelven a atender las solicitudes en el mismo sentido que antes. En esencia, la planificación C-SCAN considera al disco como si fuera circular, con la última pista adyacente a la primera. La figura 6.10 muestra un ejemplo de planificación CSCAN, donde se recorren 322 cilindros en total.

¿Qué algoritmo elegir?

La planificación SSF es la que suele producir un menor movimiento de las cabezas. Sin embargo, no es adecuada para sistemas que hacen un gran uso del disco por los problemas de localidad que hemos comentado. Por eso, en los sistemas operativos actuales, se suelen usar los algoritmos SCAN y C-SCAN tal cual los hemos descrito aquí o con pequeñas modificaciones para tratar de mejorar algún objetivo concreto. En cualquier caso, ninguno de los algoritmos vistos es óptimo en el sentido de que, para conseguir ciertas metas, como reducir el número total de movimientos de las cabezas, hay que renunciar a otras, como tratar de reducir el tiempo promedio de espera de las solicitudes.

El rendimiento de cualquier algoritmo de planificación depende en gran medida de la cantidad y tipo de las solicitudes lo cual, muchas veces, viene determinado por el sistema de ficheros que se emplee. Así, el método de implementación de ficheros usado puede tener gran influencia sobre las solicitudes de disco; no es lo mismo que al fichero se le asigne un área contigua en disco, localizada en un mismo cilindro o en cilindros adyacentes (pocos movimientos de las cabezas), que utilizar ficheros con bloques enlazados o indizados, que pueden incluir bloques dispersos por todo el disco (mejor utilización del espacio en disco, pero más movimientos de la cabeza). En este último caso se podría desfragmentar periódicamente el disco para intentar colocar todos los bloques

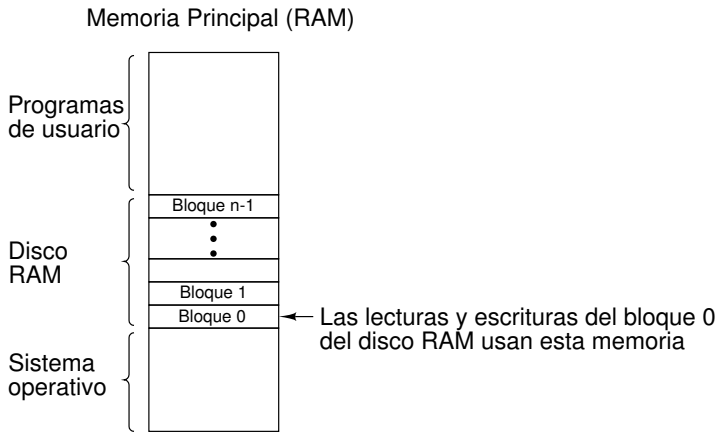

Figura 6.11. Un disco en RAM.

pertenecientes a cada fichero de forma contigua o próximos entre sí.

También es importante la localización de directorios, nodos-i, bloques indirectos, etc. Por ejemplo, en Unix, los bloques de datos de un directorio solo contienen nombres de fichero junto con los números de nodos-i asociados a esos ficheros. El resto de información de cada fichero se encuentra en su nodo-i. Podría ser una buena idea para mejorar el rendimiento (pensemos, por ejemplo, en la ejecución de una orden como `ls -l`) el colocar los bloques de datos de los directorios próximos a los bloques que alojan a los nodos-i de los ficheros contenidos en esos bloques de datos. Sistemas de ficheros como Ext2/3/4, XFS o JFS utilizan esta técnica.

Todos los algoritmos de planificación que hemos visto suponen que el tiempo de búsqueda es mayor que el tiempo de latencia. Sin embargo, si la tecnología cambia y la relación entre ambos tiempos se invierte, o surgen nuevos elementos a tener en cuenta, puede ser necesario idear nuevos algoritmos. Por ejemplo, los discos duros actuales suelen disponer de una pequeña memoria caché integrada en la controladora que usan para leer pistas enteras, o partes de ellas, cuando atienden una solicitud. De esta manera, si alguna solicitud posterior solicita uno de los sectores leído por adelantado, la petición se atenderá desde la caché. El aprovechamiento de esta caché es muy importante para mejorar el rendimiento del disco. Por eso, los algoritmos de planificación actuales tienen en cuenta este hecho y prefieren atender una solicitud que acaba de llegar, pero que está próxima a la última solicitud servida, antes que una solicitud que lleva tiempo esperando, pero que está más alejada (evidentemente, si una petición lleva esperando mucho tiempo, se atiende sin tener en cuenta otras consideraciones para no penalizarla).

6.3.3 Discos en RAM

Un dispositivo de bloques es un medio de almacenamiento con dos operaciones: leer un bloque y escribir un bloque. Por lo general, estos bloques se guardan en disco. En un disco RAM, en cambio, se utiliza una parte de la memoria principal para almacenar los bloques, lo que proporciona un acceso instantáneo a los mismos al no existir ni retraso rotacional (latencia) ni búsquedas (movimientos de las cabezas).

La figura 6.11 muestra la estructura de un disco RAM. Este se divide en n bloques, cada uno con el mismo tamaño que el que podemos encontrar en un bloque (sector) de un disco real (este tamaño lo podría indicar el usuario). Cuando el manejador recibe un mensaje para la lectura o escritura en un bloque, solo calcula el lugar de la memoria del disco en RAM donde se encuentra el bloque solicitado y lee de o escribe en él.

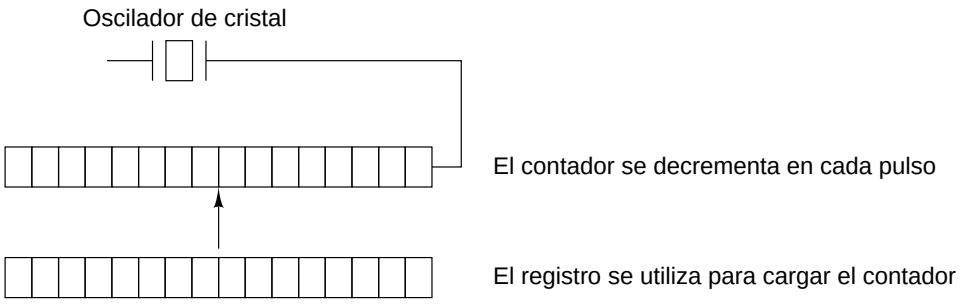

Figura 6.12. Un reloj programable.

6.4 RELOJES

Los relojes son esenciales para la operación de cualquier sistema de tiempo compartido, ya que registran la hora del día, evitan que un proceso monopolice la CPU, etc. El software para reloj toma, por lo general, la forma de un manejador de dispositivo, aunque el reloj no es un dispositivo ni de bloques ni de caracteres.

6.4.1 Hardware para relojes

Los relojes más sencillos, pero también poco comunes, son aquellos que están sujetos a la línea de corriente eléctrica de 125 o 220 voltios y provocan una interrupción por cada ciclo de voltaje, es decir, a 50 o 60 Hz.

Otro tipo de relojes más sofisticados son los *relojes programables*, que constan de tres componentes (ver figura 6.12): un oscilador de cristal de cuarzo, un contador y un registro de carga. El oscilador genera una señal periódica, de 5 a 100 MHz o incluso más, de muy alta precisión. Esta señal alimenta al contador para que cuente de forma decreciente hasta 0. Cuando el contador llega a 0, provoca una interrupción.

Los relojes programables tienen en general varios modos de operación, entre ellos, los dos siguientes:

- *Modo de disparo único*: cuando el reloj se inicia, copia el valor del registro en el contador y después decrementa el contador en cada pulso del oscilador. Cuando el contador llega a 0, provoca una interrupción y el reloj se detiene hasta que es iniciado de nuevo por el software.

- *Modo de onda cuadrada*: después de llegar a 0 y provocar la interrupción, el registro de carga se copia de manera automática en el contador y se repite el proceso. Estas interrupciones periódicas se llaman *marcas* o *tics de reloj*.

La ventaja del reloj programable es que la frecuencia de sus interrupciones se puede controlar, ya que depende de la frecuencia del oscilador de cristal de cuarzo (que suele ser fija) y del valor almacenado en el registro de carga. Así, si el oscilador tiene una frecuencia f (es decir, produce f pulsos por segundo) y el registro de carga tiene un valor c, entonces, el reloj programable producirá $\frac{f}{c}$ interrupciones por segundo en el modo de onda cuadrada.

Los chips del reloj programable contienen por lo general 2 o 3 relojes programables de forma independiente, así como otras opciones (como contar en forma ascendente, desactivar las interrupciones, etc.).

6.4.2 Software para relojes

Lo único que hace el hardware para relojes es generar interrupciones con una cierta frecuencia. Todo lo demás es realizado por el software: el *manejador del reloj*. Las funciones que desempeña este manejador son, generalmente:

- Mantener la hora del día.
- Evitar que los procesos tengan la CPU más tiempo del permitido.
- Mantener un registro del uso de la CPU.
- Controlar las alarmas que se creen para avisar a los procesos de usuario que las solicitan.
- Proporcionar cronómetros guardianes para el propio sistema operativo.

Todas estas funciones deben realizarse con rapidez, ya que se han de repetir varias veces por segundo (en cada interrupción, tic o marca de reloj). Veamos en detalle cómo realizar algunas de ellas.

Controlar la hora del día

Para controlar la hora del día basta con incrementar un contador en cada marca del reloj y registrar el tiempo que ha transcurrido desde las 00:00:00 horas del 1-1-1970, como se hace en Unix[2], o desde cualquier otro punto de referencia. Podemos pensar en diferentes esquemas para llevar un control de la hora actual contando marcas. Tres de estos posibles esquemas son los siguientes:

- Contador de marcas de 64 bits: a razón de 60 Hz, este contador puede registrar más de $9{,}8 \cdot 10^3$ millones de años (o más de $4{,}9 \cdot 10^3$ millones si el bit de signo ocupa lugar). La figura 6.13(a) muestra este caso.
- Contador que mantiene la hora del día en segundos y un contador secundario que cuenta las marcas hasta acumular un segundo. Si los contadores son de 32 bits, entonces el primero podrá almacenar hasta 2^{32} segundos, que son más de 136 años (o más de 68 años si el contador recibe un bit de signo). Por tanto, el desbordamiento del contador ocurrirá en el año 2106 (o 2038). Este esquema se puede ver en la figura 6.13(b).
- Contador de las marcas producidas desde que se arrancó el sistema y registro con el instante de arranque del sistema en segundos. En este caso, el tiempo de arranque del sistema se calcula a partir del valor actual de la hora del día y se almacena en el registro de forma conveniente (por ejemplo, como el número de segundos transcurridos desde las 00:00:00 horas del 1-1-1970 hasta el instante actual). Más tarde, al solicitar la hora del día, la hora de arranque almacenada se añade al contador de marcas para obtener la hora actual. Este caso se ilustra en la figura 6.13(c). Al igual que en el esquema anterior, con un registro de segundos de 32 bits es posible almacenar el instante de arranque del sistema hasta el año 2106. El contador de marcas, por su parte, se desbordará transcurridos 828,5 días si suponemos una frecuencia de 60 Hz, lo que obligará (al menos, en teoría) a reiniciar el sistema transcurridos poco más de dos años y tres meses.

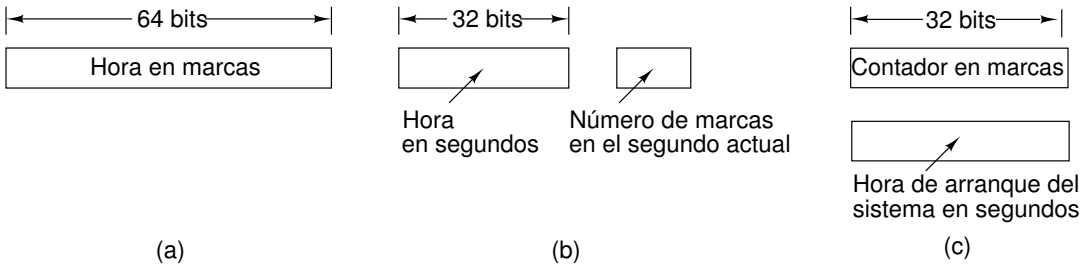

Figura 6.13. Tres formas de mantener la hora del día.

Independientemente del método que se utilice para controlar la hora, de alguna forma hay que decirle al sistema operativo la hora actual para que tenga constancia de la misma e inicialice los registros con los valores adecuados en función del esquema empleado. Una opción es que el sistema le pida la hora al usuario durante el arranque. Otra es que el propio sistema tome automáticamente la hora del *reloj de tiempo real* de la placa base. Este reloj es un pequeño circuito integrado que, a diferencia de los relojes programables que hemos descrito en la sección 6.4.1, y que producen interrupciones, mantiene la hora actual y está alimentado mediante una pequeña batería para que siga funcionando aun cuando el computador permanezca apagado.

Controlar el tiempo de ejecución de los procesos

Al asignar la CPU a un proceso, el planificador debe inicializar un contador con el valor del *quantum* de ese proceso en marcas de reloj. En cada interrupción del reloj, el manejador decrementa el contador en 1. Cuando llega a 0, el manejador llama al planificador para ceder la CPU a otro proceso.

Contabilizar el uso de la CPU

La forma más precisa de hacer esto es tener un segundo reloj (distinto del principal del sistema) e iniciarlo cada vez que se asigne la CPU a un proceso. Cuando el proceso deja la CPU (por ejemplo, porque se bloquea o es expulsado), se puede leer el valor del contador del segundo reloj para saber el tiempo durante el cual se ha ejecutado. Dicho valor se puede sumar a un campo de la entrada de la tabla de procesos correspondiente a dicho proceso. Si el proceso deja la CPU para atender una interrupción, el segundo reloj se debería parar durante el tratamiento de la misma para que no se contabilice en el proceso el tiempo que se dedica a atender la interrupción.

Otra forma más sencilla, pero menos exacta, es mantener como variable global un apuntador a la entrada de la tabla de procesos del proceso en ejecución, e incrementar un campo en dicha entrada cada vez que se produzca una marca de reloj. Esta técnica es menos exacta porque un proceso puede recibir más o menos marcas de las que le corresponden. Por ejemplo, si un proceso pasa a usar la CPU justo después de que se haya producido una marca y deja la CPU justo antes de la siguiente marca, este proceso no habrá recibido ninguna marca y, por tanto, es como si no hubiera consumido tiempo de CPU cuando en realidad estuvo en ella casi una marca completa.

Alarmas

Un proceso puede solicitar al sistema operativo que le envíe una señal, interrupción, mensaje o algo similar después de cierto intervalo de tiempo. Por ejemplo, en Unix

[2] A ese instante se le llama La Época (The Epoch).

Hora actual	Siguiente señal
183000	187000

Alarma 1	Alarma 2	.....	.....	Alarma N
215000	187000	.....	.....	207500

Figura 6.14. Simulación de varias alarmas mediante un único reloj y una tabla.

existe la llamada al sistema `alarm()` que un proceso puede usar para pedirle al sistema operativo que le avise transcurrido cierto número de segundos, cosa que él hará enviando una señal `SIG_ALARM`. Para controlar todas las alarmas solicitadas por los procesos, el sistema operativo puede usar distintas soluciones:

1. Si el manejador de reloj dispone de relojes físicos suficientes, se podría utilizar un reloj independiente para cada solicitud.

2. Como lo anterior no se suele dar, se deben simular varios relojes virtuales con un único reloj físico. Una forma de lograr esto es tener una tabla donde cada entrada contiene el tiempo de señalización (tiempo de envío de la señal) de un reloj virtual pendiente, así como una variable «siguiente señal» que indique el tiempo de la próxima señal (la primera que se producirá de entre todas las pendientes). Este esquema se muestra en la figura 6.14. Al actualizar la hora del día, el manejador de reloj verifica si ha llegado el momento de la alarma más cercana; en caso afirmativo, se envía la señal al proceso que la solicitó, se busca en la tabla la siguiente alarma por ocurrir y se actualiza la variable «siguiente señal».

3. Otra forma equivalente a la anterior, pero más eficiente si hay muchas alarmas, es simular varios relojes mediante una lista ligada de todas las solicitudes pendientes, ordenadas según el tiempo de ocurrencia (ver figura 6.15). Cada dato de la lista ligada indica el número de marcas de reloj que hay que esperar respecto a la señal anterior antes de provocar una señal. Además, existe un contador «siguiente señal» que indica cuántas marcas faltan para que se produzca la siguiente señal, correspondiente a la alarma que se encuentra en la cabeza de la lista.

   En nuestro ejemplo, tenemos cinco alarmas pendientes las cuales se dispararán en los instantes 4202 (=4200+2), 4206 (=4200+2+4), 4212 (=4200+2+4+6), 4214 (=4200+2+4+6+2) y 4215 (=4200+2+4+6+2+1). En cada marca, «siguiente señal» se decrementa. Cuando toma el valor 0, se activa la señal correspondiente a la primera alarma de la lista, se borra el elemento de la lista y «siguiente señal» toma el valor de la entrada en la cabeza de la lista (que en nuestro ejemplo sería 4).

Cronómetros guardianes

Los cronómetros guardianes son alarmas establecidas por parte del propio sistema operativo. Por ejemplo, para acceder a un DVD, el sistema debe activar el motor de la unidad lectora y esperar hasta lograr la velocidad adecuada. Al terminar la E/S, es una buena idea iniciar un cronómetro guardián y parar el motor solo en el caso de que no se realice operación alguna de E/S durante un cierto tiempo. Así, se evita el retraso que supondría tener que activar el motor en cada operación si se parara inmediatamente tras

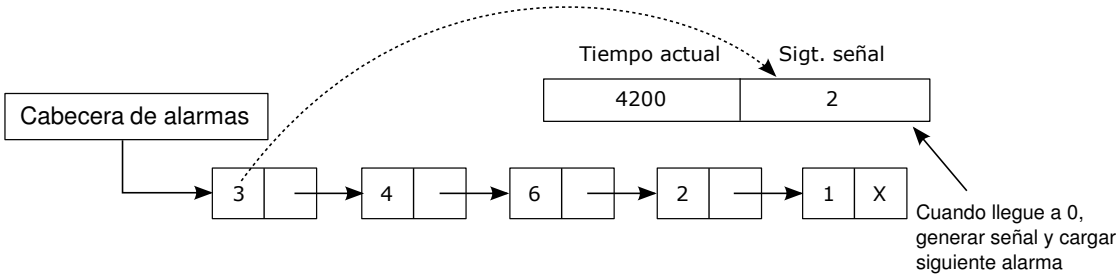

Figura 6.15. Simulación de varias alarmas mediante un único reloj y una lista. Observa que ya ha transcurrido una unidad de tiempo desde que «siguiente señal» se cargó con el número de marcas de la siguiente alarma (3 en este caso), por lo que su valor actual es 2 y la alarma se disparará en el instante 4202.

finalizar cada una, al mismo tiempo que se evita que el motor quede encendido si no hay más peticiones.

El manejador de reloj controla los cronómetros guardianes de igual manera que las alarmas de usuario. La única diferencia es que al agotar su tiempo un cronómetro guardián no provoca una señal, sino que el manejador llama a un procedimiento proporcionado por quien hizo la llamada. Puesto que todos los manejadores se encuentran en el mismo espacio de direcciones, no hay problema en realizar dicha llamada. En el núcleo las señales no existen, de ahí el uso de este mecanismo.