TEMA 6: GESTIÓN DE LA E/S

DISPOSITIVOS DE E/S

Los dispositivos de E/S se pueden dividir de manera general en dos categorías:

- **Dispositivo de bloques:** almacena la información en bloques de tamaño fijo, cada uno con su propia dirección. Se caracterizan por la posibilidad de leer de o escribir en un bloque de forma independientemente de los demás, por lo que no existe una operación de búsqueda que nos permita identificar el bloque al que se va a acceder. Ejemplos: los discos y otros dispositivos de almacenamiento secundario.
- **Dispositivo de caracteres:** 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. Ejemplos: terminales, ratones e impresoras.

También hay dispositivos que no se adaptan a esta división. Ejemplos: relojes, tarjetas gráficas mapeadas a memoria.

Aun así, este modelo que divide los dispositivos de E/S es lo bastante general como para ser utilizado por gran pate del software del SO. Por ejemplo, el sistema de ficheros solo trabaja con dispositivos abstractos de bloques.

PRINCIPIOS DEL SOFTWARE DE E/S

La idea básica es organizar el software de E/S como una serie de capas para ocultar las peculiaridades del hardware a las capas superiores.

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.
2. **Nombres uniformes:** el nombre del fichero o dispositivo debe ser simplemente una cadena o un entero y no depender de las características del dispositivo.
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 (la CPU inicia una transferencia y realiza otras cosas hasta que se produce una interrupción). Sin embargo, los programas del usuario son mucho más fáciles de escribir si las operaciones de E/S son síncronas (bloqueantes). 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 (discos) y dispositivos de uso exclusivo (impresoras). El sistema operativo debe permitir ambas posibilidades.

Una forma de lograr estos objetivos es estructurando el software de E/S en cuatro capas.

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**.

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 para que elimine el bloqueo del proceso que inició la operación, de forma que este pueda continuar su ejecución.

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.

MANEJADORES DE DISPOSITIVOS O DRIVERS

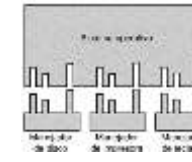
Los **manejadores de dispositivo** son los que envían órdenes a las *tarjetas controladoras* y verifican su ejecución adecuada. Cada uno de estos manejadores controla solo un tipo de dispositivo.

Para poder acceder a los registros de las controladoras, los manejadores necesitan ejecutarse en modo núcleo 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.

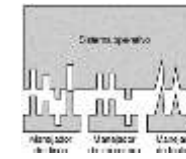
La implementación de un manejador de dispositivo no es una tarea sencilla pues supone conocer parte de las interioridades del sistema operativo y el funcionamiento exacto del dispositivo correspondiente. Esta implementación se complica si tenemos en cuenta que muchos manejadores deben ser *reentrantes*.

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.

Después de enviar la orden, pueden darse dos casos: que el manejador deba esperar hasta que el controlador realice cierto trabajo bloqueándose, o que la operación termine. Tras terminar la operación, el manejador debe verificar los errores 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.



Un último aspecto a tener en cuenta es que 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.



Los problemas de no definir una interfaz estándar son 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.

SOFTWARE DE E/S INDEPENDIENTEMENTE DEL 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. 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.
- **Nombres de los dispositivos:** el SID se encarga de asociar el nombre simbólico de cada dispositivo con el manejador adecuado.
- **Protección de los dispositivos:** se debe evitar que los usuarios no autorizados accedan a los dispositivos.
- **Proporcionar un tamaño de bloque independiente del dispositivo:** 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. Por ejemplo, el teclado no es capaz de almacenar información. 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. 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 el buffers guarda temporalmente la información para ir enviándola poco a poco.
  En el caso de los discos, interesa que ciertos bloques se almacenen temporalmente en memoria para acelerar su funcionamiento.
- **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 ya que, serán usados por los procesos de usuario para leer de o escribir información en los dispositivos físicos de bloques presentes en el sistema.

SOFTWARE DE E/S EN EL ESPACIO USUARIO

Una pequeña parte del software de E/S 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 (poniendo en orden los parámetros), del mismo modo que hay otros procedimientos de biblioteca que gestionan otras llamadas al sistema.

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.

Consideraremos como ejemplo una impresora; puede ocurrir que un proceso abra el fichero de caracteres correspondiente a la impresore y lo mantenga abierto durante horas sin llevar a cabo

ninguna actividad. Lo que se puede hacer es crear un proceso especial llamado *demonio* y un directorio espacial 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 que deben ser procesados por el demonio.

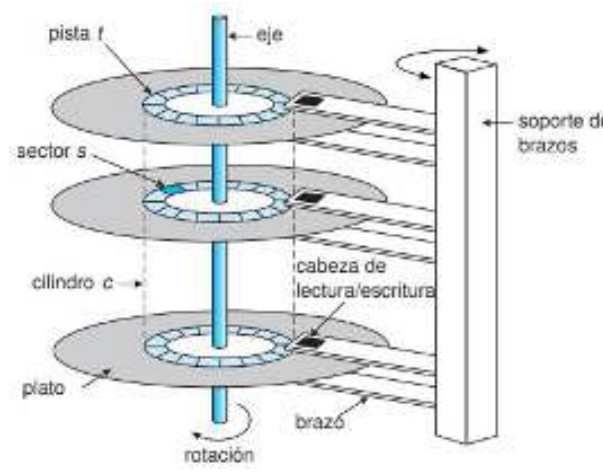
DISCOS

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.

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.

Se accede a los datos 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.



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 cabezas hacia una nueva posición, estas pueden tener acceso a un conjunto de pistas diferente. Hay tantos *cilindros* como posiciones posibles de las cabezas y el proceso de desplazar las cabezas hacia un nuevo cilindro se conoce como *búsqueda.*

Las pistas se dividen en *sectores*, que son la unidad mínima de lectura y escritura, 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. Todas las pistas no tienen por que tener el mismo numero de sectores (las pistas exteriores son más largas que las interiores).

Puesto que el número de sectores por pista es variable, el acceso a un sector por su terna se hace imposible. 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***, que no es más que su posición dentro del array.

Los pasos necesarios para acceder a el disco:

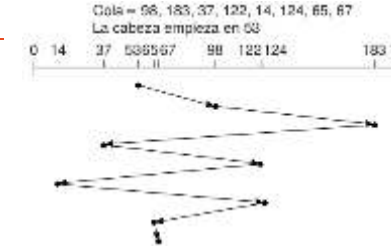
1. Mover el soporte de los brazos hasta colocar las cabezas en el cilindro adecuado (*tiempo de búsqueda*).
2. Activar la cabeza adecuada (*tiempo despreciable*).
3. Esperar a que el disco gire para que los sectores a leer o escribir pasen junto a la cabeza (*tiempo de latencia*).
4. Leer o escribir 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*.

PLANIFICACIÓN DE LOS MOVIMIENTOS DEL BRAZO

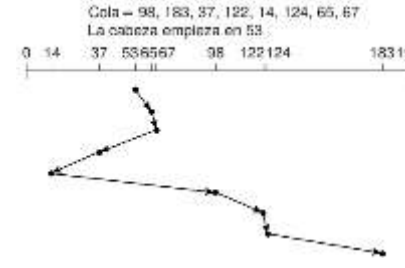
Los discos suelen tener una cola de solicitudes pendientes. Cada solicitud puede afectar a un cilindro distinto y las cabezas deberán desplazarse a dicho cilindro para procesarla. Nuestro objetivo será planificar el orden en el que se atienden las solicitudes pendientes para reducir el movimiento de las cabezas. Este objetivo debe ser logrado por el manejador del disco, que puede usar distintos algoritmos o planificadores para conseguirlo. Se supone que el manejador conoce la geometría del disco por lo que la planificación de las peticiones se hará teniendo en cuenta los cilindros a los que afecten.

PLANIFICACIÓN FCFS



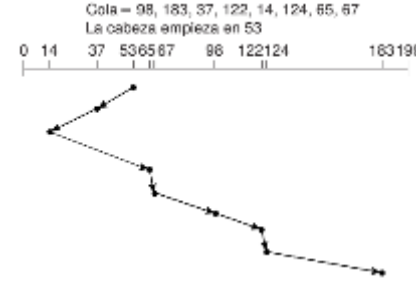
La **planificación FCFS** (*First Come, First Served*) es la más sencilla, ya que se da servicio a las solicitudes según su orden de llegada. El problema es que puede dar lugar a tiempos de servicio en promedio bastante grandes.

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 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. Tampoco es óptimo este algoritmo.

PLANIFICACIÓN SCAN O ALGORITMO DEL ASCENSOR

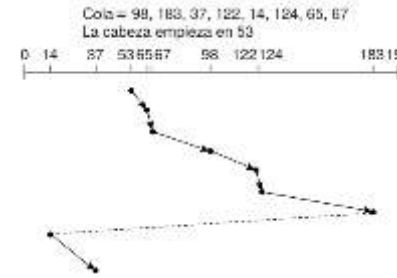


En la **planificación SCAN**, o **algoritmo del ascensor**, las cabezas de L/E se desplazan en un sentido dando servicio a las solicitudes que van encontrando en cada cilindro. Cuando no hay más solicitudes en ese sentido, se invierte el sentido para hacer lo mismo otra vez. Por tanto, en este algoritmo es necesario tener un bit que indique el sentido del movimiento.

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 esta planificación es peor, ya que suele producir más movimientos de las cabezas del disco. A cambio, tiene como ventaja que evita la localidad del algoritmo anterior.

PLANIFICACIÓN C-SCAN (SCAN CIRCULAR)



Un problema de la planificación SCAN es que puede atender muchas solicitudes que acaban de llegar antes que otras que llevan en la cola un cierto tiempo. 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.

¿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. En cualquier caso, ninguno de los algoritmos vistos es óptimo.

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 que utilizar ficheros con bloques enlazados.

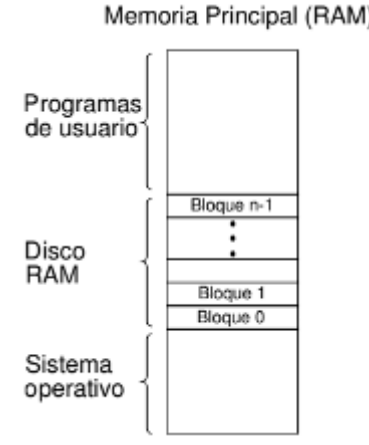
También es importante la localización de directorios. Podría ser una buena idea para mejorar el rendimiento el colocar los bloques de datos de los directorios próximos a los bloques que alojan a los nodos-i de los ficheros contenidos en esos bloques de datos.

Todos los algoritmos de planificación que hemos visto suponen que el tiempo de búsqueda es mayor que el tiempo de latencia. Sin embargo, 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.

DISCOS EN RAM

En un disco en RAM 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 ni búsquedas.

Un disco RAM se divide en *n* bloques, cada uno con el mismo tamaño que el que podemos encontrar en un bloque de un disco real. 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.



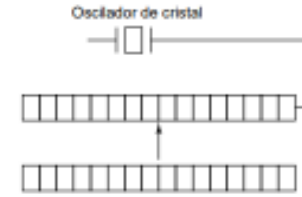
RELOJES

El software para reloj toma la forma de un manejador de dispositivo, aunque el reloj no es un dispositivo ni de bloques ni de caracteres.

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 un oscilador de cristal de cuarzo, un contador y un registro de carga. El oscilador genera una señal periódica, de 5 a 100 MHz. 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:



- **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 (f) y del valor almacenado en el registro de carga (c). 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.

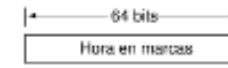
SOFTWARE PARA RELOJES

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

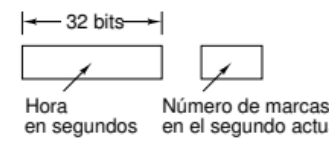
CONTROLAR LA HORA DEL DÍA

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, o desde cualquier otro punto de referencia. Podemos pensar en:

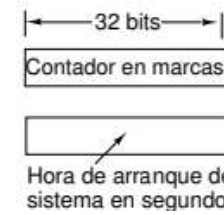
- **Contador de marcas de 64 bits:** a razón de 60 Hz, este contador puede registrar más de $9{,}8 \cdot 10^3$ millones de años.



- **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 232 segundos, que son más de 136 años.



- **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. Más tarde, al solicitar la hora del día, la hora de arranque almacenada se añade al contador de marcas para obtener la hora actual. El contador de marcas, por su parte, se desbordará transcurridos 828,5 días si suponemos una frecuencia de 60 Hz, lo que obligará a reiniciar el sistema transcurridos poco más de dos años y tres meses.



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.

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

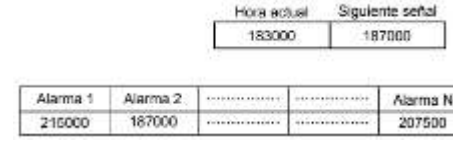
La forma más precisa de hacer esto es tener un segundo reloj e iniciarlo cada vez que se asigne la CPU a un proceso. Cuando el proceso deja la CPU, se puede leer el valor del contador del segundo reloj para saber el tiempo durante el cual se ha ejecutado. Dicho valor se puede sumar a un campo de la entrada de la tabla de procesos correspondiente a dicho proceso.

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.

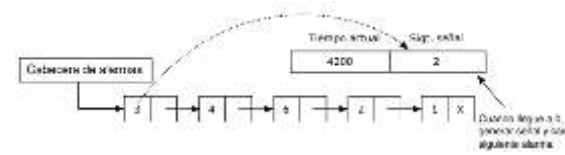
ALARMAS

Un proceso puede solicitar al sistema operativo que le envíe una señal, interrupción, mensaje o algo similar después de cierto intervalo de tiempo. En Unix es la llamada al sistema `alarm()` que un proceso puede usar para pedirle al sistema operativo que le avise transcurrido cierto número de segundos. Para controlar todas las **alarmas** solicitadas:

1. Si el manejador de reloj dispone de relojes físicos suficientes, se podría utilizar un reloj independiente para cada solicitud.
2. Se simulan 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 de un reloj virtual pendiente, así como una variable «*siguiente señal*» que indique el tiempo de la próxima señal. 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. Se simulan varios relojes mediante una lista ligada de todas las solicitudes pendientes. 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.



CRONÓMETROS GUARDIANES

Los **cronómetros guardianes** son alarmas establecidas por parte del propio sistema operativo. Se evita el retraso que supondría tener que activar el motor en cada operación si se parara inmediatamente tras 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.

Todas estas funciones deben realizarse con rapidez, ya que se han de repetir varias veces por segundo.