TEMA 5 – ENTRADA/SALIDA

Bibliografía

Este documento se ha elaborado, principalmente, a partir de las referencias bibliográficas que se exponen a continuación, y con propósito meramente académico. El alumno debería completar el contenido de este tema con bibliografía existente en la biblioteca y mediante recursos de la Web, hasta que los conceptos que se indican en el mismo sean afianzados correctamente.

[STALLINGS] W. Stallings. Sistemas operativos. 5ª edicion, Prentice Hall, Madrid, 2005.
[TANENBAUM] A. S. Tanenbaum. Sistemas operativos modernos, 3a edicion. Prentice Hall, Madrid, 2009.
[ARANDA] Joaquin Aranda Alamansa, Mª Antonia Canto Diaz, Jesús Manuel de la Cruz García, Sebastian Dormido Bencomo, Carolina Mañoso Hierro. Sistemas operativos, teoría y problemas. Editoral Sanz y Torres, S.L, 2002.

Contenido

- Dispositivos de E/S (Capítulo 5, sección 5.1.1 [Tanenbaum] y Tema 6 [Aranda]).
- Controladores de dispositivos (Capítulo 5, sección 5.1.2 [Tanenbaum]) y Tema 6, sección 6.1.1 [Aranda]).
- Técnicas del sistema de E/S (Capítulo 11, sección 11.2 [Stallings]).
- Acceso directo a memoria. (Capitulo 5, sección 5.1.4 y 5.1.5 [Tanenbaum]).
- Objetivos del software de E/S (Capítulo 5, sección 5.2.1 [Tanenbaum]).
- Drivers de los dispositivos (Capítulo 5, sección 5.3.2 [Tanenbaum]).
- Software de E/S independiente del dispositivo (Capítulo 5, sección 5.3.3 [Tanenbaum])
- Software de E/S en el espacio de usuario (Capítulo 5, sección 5.3.4 [Tanenbaum])

Entrada/Salida (E/S)

Probablemente, el sistema de E/S y el sistema de gestión de ficheros son las partes más intrincadas en el diseño de un sistema operativo. Con respecto a la E/S, el aspecto fundamental es el rendimiento. El sistema de E/S es realmente un factor clave para obtener un buen rendimiento en el sistema. Observando el modo de operación interno de un computador, se puede apreciar que la velocidad del computador continúa incrementándose y, en el caso de que un único procesador no sea todavía suficientemente rápido, las configuraciones SMP proporcionan múltiples procesadores para acelerar el trabajo. Las velocidades de acceso a la memoria interna también están incrementándose, aunque no tan rápidamente como la velocidad de procesador. No obstante, con el uso inteligente de uno, dos o incluso más niveles de cache interna, se está logrando mantener el tiempo de acceso de la memoria principal acorde con la velocidad del procesador. Sin embargo, la E/S supone un reto significativo en el rendimiento, particularmente en el caso del almacenamiento en disco.

Dispositivos de E/S

Los dispositivos de E/S se pueden dividir básicamente en dos categorías: dispositivos de bloque y dispositivos de carácter .

En un **dispositivo de bloque** (discos duros, CD-ROMs y memorias USBs) los datos se transfieren en bloques de información indivisibles cuyas características básicas son:

- Los bloques son de tamaño fijo. Los tamaños de bloque comunes varían desde 512 bytes hasta 32.768 bytes. Todas las transferencias se realizan en unidades de uno o más bloques completos.
- Un software puede leer o escribir cada bloque de manera independiente de los demás.
- Cada bloque se referencia o direcciona usando una dirección, que es única para cada uno.

El otro tipo de dispositivo de E/S es el **dispositivo de carácter** (impresoras, interfaces de red, ratón, monitores, puertos de comunicación) los cuales se comunican con la CPU por medio de bytes individuales. Tienen las siguientes características:

- No están sujetos a un estructura de bloques.
- No se pueden utilizar direcciones al no ser direccionables, no son memorias, sino que tienen un determinado buffer intermedio.
- No se pueden realizar operaciones de búsqueda.
- Envía o acepta un flujo de caracteres.

Los dispositivos de E/S cubren un amplio rango de velocidades, lo cual impone una presión considerable en el software para obtener un buen desempeño en las velocidades de transferencia de datos. La figura 5.1 muestra las velocidades de transferencia de datos de algunos dispositivos comunes.

Dispositivo	Velocidad de transferencia de datos
Teclado	10 bytes/seg
Ratón	100 bytes/seg
Módem de 56K	7 KB/seg
Escáner	400 KB/seg
Cámara de video digital	3.5 MB/seg
802.11g inalámbrico	6.75 MB/seg
CD-ROM de 52X	7.8 MB/seg
Fast Ethernet	12.5 MB/seg
Tarjeta Compact Flash	40 MB/seg
FireWire (IEEE 1394)	50 MB/seg
USB 2.0	60 MB/seg
Red SONET OC-12	78 MB/seg
Disco SCSI Ultra 2	80 MB/seg
Gigabit Ethernet	125 MB/seg
Unidad de disco SATA	300 MB/seg
Cinta de Ultrium	320 MB/seg
Bus PCI	528 MB/seg

Figura 5-1. Velocidades de transferencia de datos comunes de algunos dispositivos, redes y buses.

Controladores de dispositivos o de E/S

Por lo general, las unidades de E/S consisten en un componente mecánico y un componente electrónico. El componente electrónico se llama controlador de dispositivo, adaptador o controlador de E/S. En las computadoras personales, comúnmente tiene la forma de un chip en la tarjeta principal (placa madre) o una tarjeta de circuito integrado que se puede insertar en una ranura de expansión (PCI). El componente mecánico es el dispositivo en sí (disco, monitor, impresora,...). Esto se muestra en la figura 1-6.

Dicho esto, un controlador de dispositivo (a veces denominado también controlador o módulo de E/S) es un módulo de la computadora responsable del control de uno o más dispositivos físicos externos y del intercambio de datos entre dichos periféricos con la memoria principal o con los registros de la CPU. Por esta razón, el controlador de E/S debe poseer una interfaz interna al computador (para su conexión con la CPU y con la memoria principal), y una interfaz externa al computador (para su conexión con el dispositivo externo).

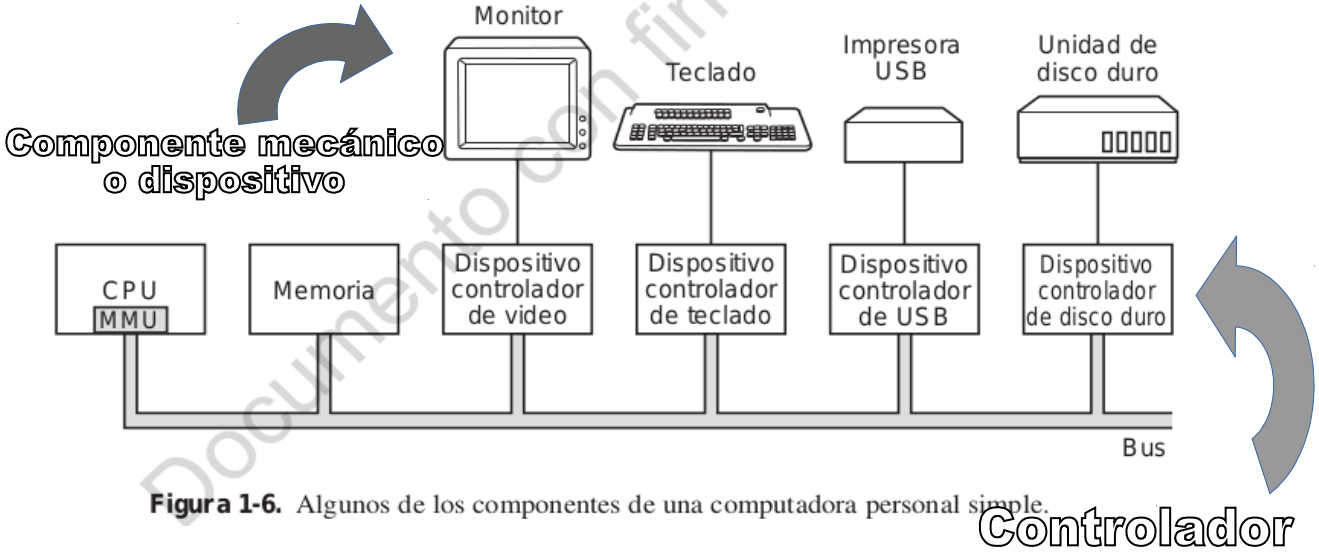
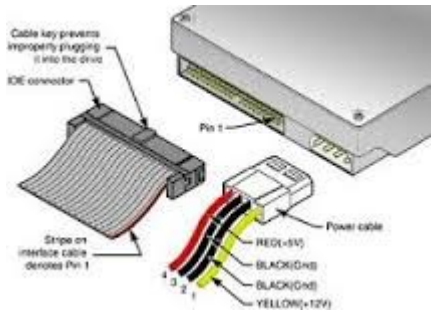


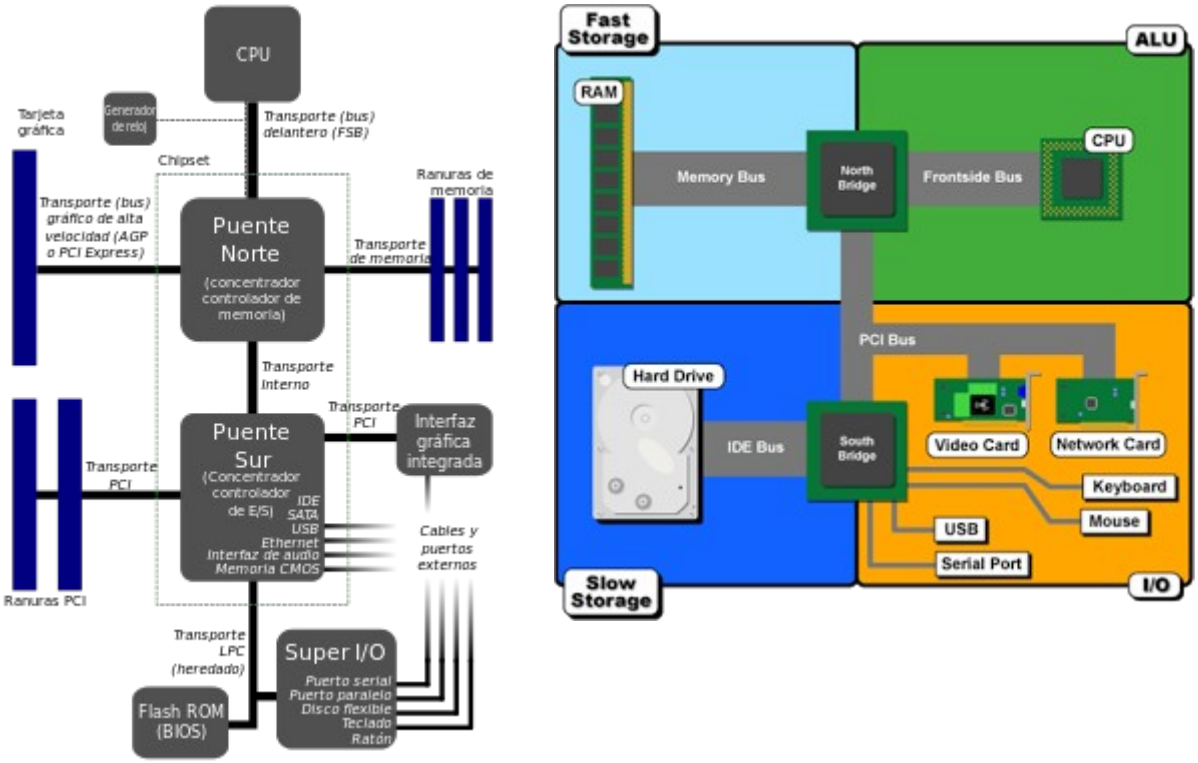
Figura 1-6. Algunos de los componentes de una computadora personal simple.

De manera breve y resumida, las principales funciones del controlador de dispositivos son las siguientes:

- Control y temporización.
- Comunicación con la CPU.
- Comunicación con el dispositivo externo.
- Almacenamiento temporal de datos, debido a las diferentes velocidades de transferencia de datos que tienen los dispositivos externos comparados con la CPU y la memoria principal.
- Detección de errores.

El controlador de dispositivo o controlador de E/S (hardware), por lo general, y dependiendo del dispositivo (teclado, disco,...), contiene un conector, en el que se puede conectar un cable que conduce al dispositivo en sí (hardware). Muchos controladores pueden manejar dos, cuatro o inclusive ocho dispositivos idénticos. Si la interfaz (hardware - conector al dispositivo en si - donde va conectado el cable del que se acaba de hablar) entre el controlador y el dispositivo es estándar, entonces las empresas pueden fabricar dispositivos que se adapten a esa interfaz. Por ejemplo, muchas empresas fabrican unidades de disco para interfaces estándar como IDE, SATA, SCSI o USB. En la siguiente imagen se muestra un ejemplo de interfaz IDE de disco duro. Para este caso, el controlador de dispositivo o módulo de E/S para el disco duro está integrado en placa (normalmente se trata del Puente Sur dentro de lo que se denomina *chipset* de la placa madre), y el interfaz estándar físico entre el controlador y el disco duro son las dos ranuras IDE (azul y a continuación negra) que aparecen en la placa madre, junto con el bus de datos con los extremos azul y negro.





Los controladores de E/S varían en complejidad y como se ha comentado anteriormente, en el número de dispositivos externos que pueden controlar. En la figura 6.3 se muestra el diagrama de bloques genérico de un controlador de E/S. El controlador de E/S se conecta con el resto del computador a través del bus del sistema. Los datos que se transfieren al controlador o desde el controlador se almacenan temporalmente en uno o más registros de datos. Algunas de las líneas de control son utilizadas para reconocer y generar las direcciones asociadas a los dispositivos que se controla. Cada controlador tiene asociada una única dirección, o un conjunto de ellas si controla a varios dispositivos. Así mismo el controlador de E/S tiene una lógica específica para la interfaz con cada periférico que controla.

.

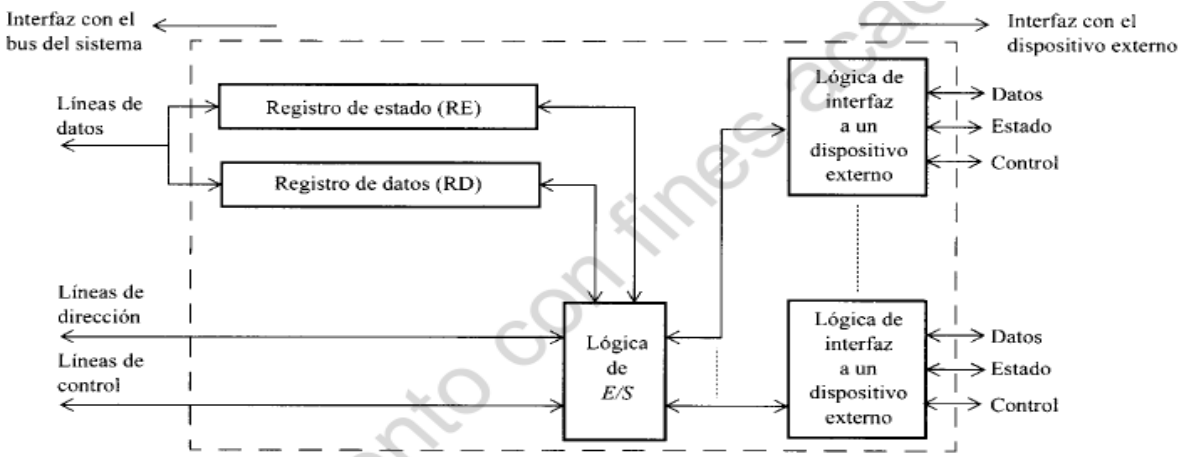


Figura 6.3: Diagrama de bloques de un controlador de *E/S*

Técnicas del sistema de entrada-salida

En el capítulo 1 se presentaron tres técnicas para llevar a cabo la E/S salida y que es conveniente repasar:

- **E/S programada.** El procesador envía un mandato de E/S, a petición de un proceso, a un módulo de E/S; a continuación, ese proceso realiza una espera activa hasta que se complete la operación antes de continuar.
- **E/S dirigida por interrupciones.** El procesador emite un mandato de E/S a petición de un proceso y continúa ejecutando otro proceso (o el mismo si no se necesita de la espera de la E/S) el cual se interrumpe por el módulo de E/S cuando éste ha completado su trabajo. Las siguientes instrucciones a ejecutar una vez se ha tratado la interrupción pueden ser del mismo proceso que se estaba ejecutando antes de la interrupción y otro diferente, por ejemplo uno que esperaba el evento que se acaba de producir y que se ha notificado mediante la señal de interrupción.
- **Acceso directo de memoria (Direct Memory Access, DMA)**. Un módulo de DMA controla el intercambio de datos entre la memoria principal y un módulo de E/S sin interrumpir a la CPU. El procesador manda una petición de transferencia de uno o varios caracteres o bloques de datos al módulo de DMA y resulta interrumpido sólo cuando se hayan transferido. Con el uso de DMA se consiguen dos cosas: 1) que los caracteres o bloques no se tengan que copiar desde el módulo de E/S a la memoria por parte de la CPU, lo que conllevaría "gasto de tiempo de CPU" y 2) ahorrar en interrupciones cuando se quiere transmitir más de un carácter o bloque. Este técnica se ampliará en la siguiente sección con respecto al Tema 1.

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

Tabla 11.1. Técnicas de E/S.

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

Acceso directo a memoria (DMA). Recordatorio y ampliación.

A continuación se recordarán y ampliarán brevemente el acceso a memoria mediante DMA y el uso de interrupciones. Ahora está en disposición de entender mejor estos conceptos que se explicaron en el primer tema.

La CPU necesita direccionar o localizar los controladores de dispositivos para intercambiar datos con ellos. La CPU puede solicitar datos de un controlador de E/S un bit a la vez, pero al hacerlo se desperdicia el tiempo de la CPU, por lo que a menudo se utiliza un esquema distinto, conocido como DMA (Acceso Directo a Memoria). El sistema operativo sólo puede utilizar DMA si el hardware tiene un controlador de DMA, que la mayoría de los sistemas tienen. El controlador DMA suele estar integrado en el *chipset* la placa base. Dependiendo del tipo de procesador que se utilice, hay arquitecturas que dividen el *chipset* en Puente Norte y Puente Sur, y concretamente el

controlador DMA se encontraría dentro del Puente Sur. En microprocesadores de última generación, como los Intel i7, esa división puede ser diferente. Lo más común es que haya un solo controlador de DMA disponible para regular las transferencias a varios dispositivos junto con el módulo o controlador de E/S.

Sin importar cuál sea su ubicación física, el controlador de DMA tiene acceso al bus del sistema de manera independiente de la CPU, como se muestra en la figura 5-4, donde se pone como ejemplo un controlador de disco duro. Contiene varios registros en los que la CPU puede escribir y leer:

- Un registro de dirección de memoria.
- Un registro contador de bytes (para saber la cantidad de información que debe transferir).
- Uno o más registros de control. Los registros de control especifican el puerto de E/S a utilizar, la dirección de la transferencia (si se va a leer del dispositivo de E/S o se va a escribir en él) y la unidad de transferencia (un byte a la vez o una palabra o bloque a la vez).

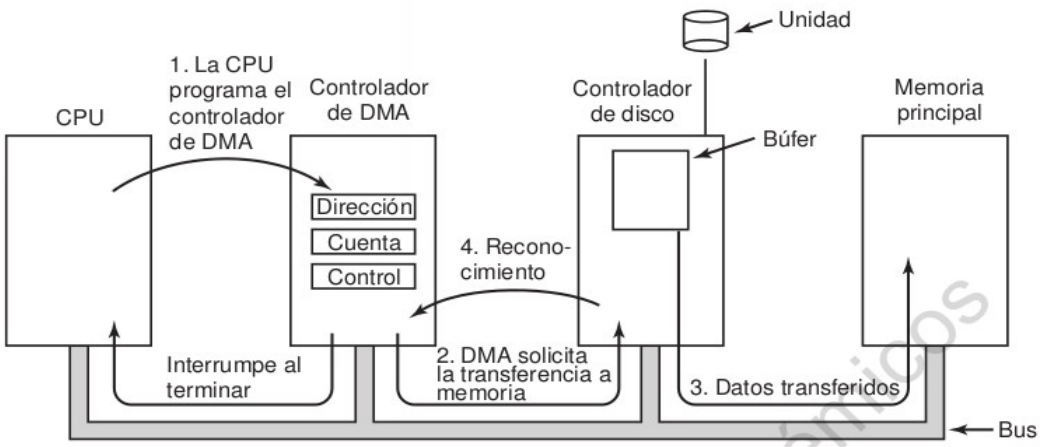


Figura 5-4. Operación de una transferencia de DMA.

Para explicar el funcionamiento del DMA, veamos primero cómo ocurren por ejemplo las lecturas de un disco duro (dispositivo de bloque) cuando no se utiliza DMA y se usa E/S por interrupciones. Primero, el controlador o módulo de E/S de disco lee el bloque (uno o más sectores) de la unidad en forma serial, bit por bit, hasta que se coloca todo el bloque completo en el búfer interno del controlador o módulo de E/S. Después calcula la suma de comprobación para verificar que no hayan ocurrido errores de lectura (algoritmo de comprobación de errores). Después, el controlador produce una interrupción en dirección al procesador. Cuando el sistema operativo atiende la interrupción puede leer el bloque de disco desde el búfer del controlador mediante la ejecución de un ciclo de reloj en el que en cada iteración se lee un byte o una palabra de un bufer del módulo de E/S y se almacena en la memoria principal. Lo mismo ocurre si hacemos esto con un dispositivo de carácter y no de bloque.

Bien, pues cuando se utiliza DMA el procedimiento es distinto. Primero la CPU programa el controlador de DMA, para lo cual establece sus registros (dirección, cuenta y control) de manera que sepa qué debe transferir y a dónde (**paso 1 en la figura 5-4**). También emite un comando al controlador de disco para indicarle que debe leer/escribir datos del/en disco en su búffer interno y verificar la suma de comprobación. Cuando hay datos válidos en el buffer del controlador de disco, puede empezar el DMA.

El controlador de DMA inicia la transferencia enviando una petición de lectura/escritura al controlador de disco mediante el bus **(paso 2**). Esta petición de lectura/escritura se ve como cualquier otra petición, por lo que el controlador de disco no sabe ni le importa si vino de la CPU o de un controlador de DMA. Por lo general, la dirección de memoria principal en la que se va a escribir (datos provenientes del disco) o de la que se va a leer (datos que se escribirán en el disco) está en las líneas de dirección del bus, por lo que cuando el controlador de disco, en el caso de escritura en memoria, obtiene la siguiente palabra de su buffer interno, sabe dónde escribir, o en el caso en que deba leer de memoria sabrá de qué dirección obtener datos para meterlos en su buffer interno y pasarlos al disco duro.

Si suponemos que hacemos una escritura, la escritura en memoria principal se realiza en otro ciclo más de bus estándar **(paso 3)**.

Cuando se completa la escritura, el controlador de disco envía una señal de reconocimiento al controlador de DMA, también a través del bus **(paso 4).** El controlador de DMA incrementa a continuación la dirección de memoria a utilizar y disminuye la cuenta de bytes a leer o escribir. Si la cuenta de bytes es aún mayor que 0, se repiten los pasos del 2 al 4.

Si la cuenta llega a 0 en ese momento, el controlador de DMA interrumpe la CPU para hacerle saber que la transferencia está completa. Cuando el sistema operativo atiende la interrupción ni tiene que copiar el bloque o bloques del controlador hasta la memoria principal, ya se encuentran allí. En caso de que el dispositivo sea de caracteres ocurre lo mismo. Por tanto con el uso de DMA hemos conseguido dos cosas: 1) que los caracteres o bloques no se tengan que copiar desde el módulo de E/S a la memoria por parte de la CPU, lo que conllevaría "gasto de tiempo de CPU" y 2) ahorrar en interrupciones cuando se quiere transmitir más de un carácter o bloque.

Los controladores de DMA varían considerablemente en cuanto a su sofisticación. Los más simples manejan una transferencia a la vez, como se describió antes, es decir, gestionan un dispositivo periférico por vez. Los más complejos se pueden programar para manejar varias transferencias para varios dispositivos. Dichos controladores tienen varios conjuntos de registros internos. La CPU empieza por cargar cada conjunto de registros con los parámetros relevantes para su transferencia. Cada transferencia debe utilizar un controlador de dispositivo distinto. Después de transferir cada palabra (los pasos 2 al 4) en la figura 5-4, el controlador de DMA decide a cuál dispositivo va a dar servicio a continuación. Se puede configurar para utilizar un algoritmo por turno rotatorio, o puede tener un diseño de esquema prioritario para favorecer a unos dispositivos sobre otros.

Puede surgir una pregunta de lo anteriormente expuesto ¿para qué necesita un controlador o módulo de E/S un buffer interno? Hay dos razones:

- En primer lugar, al utilizar el buffer interno, el controlador de disco puede verificar la suma de comprobación antes de iniciar una transferencia y si la suma de comprobación es incorrecta se señala un error y no se realiza la transferencia. Las sumas de comprobación intentan detectar errores en las transferencias, de forma que si un solo bit está mal copiado se señala un error y se aborta la transferencia o se reintenta.
- La segunda es que, una vez que se inicia una transferencia de disco, los bits siguen llegando a una velocidad constante, esté listo o no el módulo de E/S para ellos. Si el controlador tratara de escribir datos directamente en la memoria, como se ha comentado antes, tendría que pasar por el bus del sistema por cada palabra transferida. Si el bus estuviera ocupado debido a que algún otro dispositivo lo estuviera utilizando el controlador tendría que esperar. Si la siguiente palabra del disco llegara antes de que se almacenara la anterior, el controlador tendría que almacenarla en algún otro lado y para ello usa el buffer. Pueden darse casos en lo

que si el bus está muy ocupado haya que terminar una transferencia señalando un error de desbordamiento del buffer del controlador.

A medida que evolucionan las computadoras, la CPU cada vez se aísla más de las operaciones de E/S, convirtiéndose los módulos de E/S en verdaderos procesadores independientes e incluso con juegos de instrucciones propios, teniendo en muchos casos hasta su propia memoria local de gran capacidad y ejecutando sus propios programas de E/S.

Dicho esto, y asociado e inseparable a las técnicas de E/S, podemos terminar repasando muy brevemente las **interrupciones**. En un sistema de computadora personal común, la estructura de las interrupciones es como se muestra en la figura 5-5. A nivel de hardware, las interrupciones funcionan de la siguiente manera. Cuando un módulo de E/S ha terminado el trabajo que se le asignó, produce una interrupción

Para ello, impone una señal en una línea de bus que se le haya asignado. Esta señal es detectada a nivel hardware por el procesador, que después decide lo que debe hacer en función del tipo de interrupción, ejecutando para ello una rutina de interrupciones (ISR).

Si no hay otras interrupciones pendientes, la rutina de interrupciones procesa la interrupción de inmediato. Si hay otra en progreso, o si otro dispositivo ha realizado una petición simultánea en una línea de petición de interrupción de mayor prioridad en el bus, el dispositivo sólo se ignora por el momento. En este caso, continúa imponiendo una señal de interrupción en el bus hasta que la CPU la atiende.

Para manejar la interrupción, el controlador coloca un número en las líneas de dirección que especifican cuál dispositivo desea atención e impone una señal para interrumpir a la CPU. La señal de interrupción hace que la CPU deje lo que está haciendo. El número en las líneas de dirección se utiliza como índice en una tabla llamada vector de interrupciones para obtener un nuevo contador del programa. Este contador del programa apunta al inicio del procedimiento de servicio de interrupciones correspondiente (puede haber una rutina ISR por tipo de interrupción). La ubicación del vector de interrupción se puede determinar de manera estática (hardwired) en la máquina, o puede estar en cualquier parte de la memoria, con un registro de la CPU, cargado por el sistema operativo, apuntando a su origen.

Poco después de que se empieza a ejecutar el procedimiento de servicio de interrupciones o ISR éste escribe cierto valor en uno de los puertos de E/S del controlador que produjo la interrupción. Este reconocimiento indica al controlador que ya puede emitir otra interrupción y que se está tratando la actual. En el primer tema se comentaron varias técnicas de tratamiento de interrupciones que se utilizan en los sistemas operativos.

Objetivos del software de Entrada/Salida

Un concepto clave en el diseño del software de E/S se conoce como **independencia de dispositivos**. Lo que significa es que debe ser posible escribir programas que puedan acceder a cualquier dispositivo de E/S sin tener que especificar el dispositivo por adelantado. Por ejemplo, un programa que lee un archivo como entrada debe tener la capacidad de leer un archivo en el disco duro, un CD-ROM, un DVD o una memoria USB sin tener que modificar el programa para cada dispositivo distinto. Depende del sistema operativo encargarse de los problemas producidos por el hecho de que estos dispositivos en realidad son diferentes y requieren secuencias de comandos muy distintas para leer o escribir.

Un objetivo muy relacionado con la independencia de los dispositivos es la **denominación**

uniforme. El nombre de un archivo o dispositivo simplemente debe ser una cadena o un entero sin depender del dispositivo de ninguna forma. En UNIX, todos los discos se pueden integrar en la jerarquía del sistema de archivos de maneras arbitrarias, por lo que el usuario no necesita estar al tanto de cuál nombre corresponde a cuál dispositivo. Por ejemplo, una memoria USB se puede montar encima del directorio /usr/ast/respaldo, de manera que al copiar un archivo a /usr/ast/respaldo/lunes, este archivo se copie a la memoria USB. De esta forma, todos los archivos y dispositivos se direccionan de la misma forma: mediante el nombre de una ruta.

Otra cuestión importante relacionada con el software de E/S es el **manejo de errores**. En general, los errores se deben manejar lo más cerca del hardware que sea posible. Si el controlador descubre un error de lectura, debe tratar de corregir el error por sí mismo. Si no puede, entonces el software controlador del dispositivo debe manejarlo, tal vez con sólo tratar de leer el bloque de nuevo. Muchos errores son transitorios, como los errores de lectura ocasionados por pizcas de polvo en la cabeza de lectura, y comúnmente desaparecen si se repite la operación. Sólo si los niveles inferiores no pueden lidiar con el problema, los niveles superiores deben saber acerca de ello. En muchos casos, la recuperación de los errores se puede hacer de manera transparente a un nivel bajo, sin que los niveles superiores se enteren siquiera sobre el error.

Otra cuestión clave es la de las transferencias **síncronas** (de bloqueo) contra las **asíncronas** (controladas por interrupciones). La mayoría de las operaciones de E/S son asíncronas: la CPU inicia la transferencia y se va a hacer algo más hasta que llega la interrupción. Los programas de usuario son mucho más fáciles de escribir si las operaciones de E/S son de bloqueo: después de una llamada al sistema *read()*, el programa se suspende de manera automática hasta que haya datos disponibles en el búfer. Depende del sistema operativo hacer que las operaciones que en realidad son controladas por interrupciones parezcan de bloqueo para los programas de usuario.

Otra cuestión relacionada con el software de E/S es el uso de **búfer**. A menudo los datos que provienen de un dispositivo no se pueden almacenar directamente en su destino final. Por ejemplo, cuando un paquete llega de la red, el sistema operativo no sabe dónde colocarlo hasta que ha almacenado el paquete en alguna parte y lo examina. Además, algunos dispositivos tienen severas restricciones en tiempo real (por ejemplo, los dispositivos de audio digital), por lo que los datos se deben colocar en un búfer de salida por adelantado para desacoplar la velocidad a la que se llena el búfer de la velocidad a la que se vacía. Como se ha comentado anteriormente, cada tipo de dispositivo tiene su velocidad de transferencia de bits.

Drivers de dispositivos

Cada controlador o módulo de E/S tiene ciertos registros que se utilizan para enviar comandos y leer el estado de diversos dispositivos. El número de registros y la naturaleza de los comandos varían dependiendo del dispositivo que se trate. Por ejemplo, un driver de ratón tiene que aceptar información del ratón que le indica qué tanto se ha desplazado y cuáles botones están oprimidos en un momento dado. Por el contrario, un driver de disco tal vez tenga que saber todo acerca de los sectores, pistas, cilindros, cabezas, movimiento del brazo, los propulsores del motor, los tiempos de asentamiento de las cabezas y todos los demás mecanismos para hacer que el disco funcione en forma apropiada. Obviamente, estos drivers serán muy distintos.

Como consecuencia, cada dispositivo de E/S conectado a una computadora necesita cierto código específico para controlarlo. Este código, conocido como driver, es escrito por el fabricante del dispositivo y se incluye con el mismo, y junto con el módulo de E/S es necesario para gobernarlo correctamente. Como cada sistema operativo necesita sus propios drivers, los fabricantes de dispositivos por lo común los proporcionan para varios sistemas operativos populares.

Cada driver maneja un tipo de dispositivo o, a lo más, una clase de dispositivos estrechamente relacionados. Por ejemplo, un driver de disco SCSI puede manejar por lo general varios discos SCSI de distintos tamaños y velocidades, y tal vez un CD-ROM SCSI también. Por otro lado, un ratón y una palanca de mandos son tan distintos que por lo general se requieren controladores diferentes.

Para poder utilizar el hardware del dispositivo (es decir, los registros físicos del mismo), el driver por lo general tiene que formar parte del kernel del sistema operativo, cuando menos en las arquitecturas actuales. Como los diseñadores de cada sistema operativo saben qué piezas de código (drivers) escritas por terceros se instalarán en él, necesita tener una arquitectura que permita dicha instalación. Esto implica tener un modelo bien definido de lo que hace un driver y la forma en que interactúa con el resto del sistema operativo.

Generalmente los sistemas operativos clasifican los drivers en un pequeño número de categorías. Las categorías más comunes son los dispositivos de bloque como los discos, que contienen varios bloques de datos que se pueden direccionar de manera independiente, y los dispositivos de carácter como los teclados y las impresoras, que generan o aceptan un flujo de caracteres.

La mayoría de los sistemas operativos definen una interfaz estándar que todos los drivers de bloque deben aceptar. Estas interfaces consisten en varios procedimientos que el resto del sistema operativo puede llamar para hacer que el driver realice un trabajo para él. Los procedimientos ordinarios son los que se utilizan para leer un bloque (dispositivo de bloque) o escribir una cadena de caracteres (dispositivo de carácter).

En algunos sistemas, el sistema operativo es un solo programa binario que contiene compilados en él todos los drivers que requerirá. Este esquema fue la norma durante años con los sistemas UNIX, debido a que eran ejecutados por centros de computadoras y los dispositivos de E/S cambiaban pocas veces. Si se agregaba un nuevo dispositivo, el administrador del sistema simplemente recompilaba el kernel con el nuevo driver para crear un nuevo binario. Con la llegada de las computadoras personales y la multitud de dispositivos de E/S, este modelo ya no era funcional. Pocos usuarios son capaces de volver a compilar o vincular el kernel, aun si tienen el código fuente o los módulos de código objeto, que no siempre es el caso. En vez de ello, los sistemas operativos (empezando con MS-DOS) se inclinaron por un modelo en el que el driver se carga en forma dinámica en el sistema durante la ejecución. Los diferentes sistemas manejan la carga de drivers en distintas formas.

Un driver de dispositivo tiene varias funciones. La más obvia es aceptar peticiones abstractas de lectura y escritura del software independiente del dispositivo que está por encima de él (programa de usuario por ejemplo), y ver que se lleven a cabo. Pero también hay otras tantas funciones que deben realizar. Por ejemplo, el driver debe inicializar el dispositivo, si es necesario. También puede tener que administrar sus propios requerimientos y eventos del registro.

Muchos tipos de drivers de dispositivos tienen una estructura general similar. Un driver ordinario empieza por comprobar los parámetros de entrada para ver si son válidos. De no ser así se devuelve un error. Si son válidos tal vez sea necesaria una traducción de términos abstractos a concretos. Para un driver de disco, esto puede significar tener que convertir un número de bloque lineal en los números de cabeza, pista, sector y cilindro para la geometría del disco. A continuación, el driver puede comprobar si el dispositivo se encuentra en uso. De ser así, la petición se pondrá en cola para procesarla después. Si el dispositivo está inactivo, el estado del hardware se examinará para ver si la petición se puede manejar en ese momento. Tal vez sea necesario encender el dispositivo o iniciar un motor para que puedan empezar las transferencias. Una vez que el dispositivo esté encendido y listo para trabajar, puede empezar el control en sí, y si es necesario se hará uso de técnicas de E/S

como por ejemplo por DMA para transferir información.

Controlar el dispositivo significa enviarle una secuencia de comandos. Una vez que el driver sabe qué comandos va a emitir empieza a escribirlos en los registros de dispositivo mediante algunas de las técnicas de E/S estudiadas. Después de escribir cada comando, tal vez sea necesario comprobar si el dispositivo aceptó el comando y está preparado para aceptar el siguiente. Esta secuencia continúa hasta que se hayan emitido todos los comandos.

Una vez que se han emitido los comandos, se dará una de dos situaciones. En muchos casos el módulo de E/S debe esperar hasta que el dispositivo realice cierto trabajo para él, por lo que se bloquea a sí mismo hasta que llegue la interrupción para desbloquearlo. Sin embargo, en otros casos la operación termina sin retraso, por lo que el módulo de E/S no necesita bloquearse. De cualquier forma, una vez que se ha completado la operación, el módulo de E/S debe comprobar si hay errores. Si todo está bien, puede hacer que pasen datos al software independiente del dispositivo (por ejemplo, un bloque que se acaba de leer). Por último, devuelve cierta información de estado para reportar los errores de vuelta al que lo llamó. Si hay otras peticiones en la cola, ahora se puede seleccionar e iniciar una de ellas. Si no hay nada en la cola, el módulo de E/S se bloquea en espera de la siguiente petición. Note que el módulo de E/S interactúa con un dispositivo en función de cómo se lo indique el driver del mismo.

Este modelo simple es sólo una aproximación a la realidad. Muchos factores hacen el código mucho más complicado. Por ejemplo, en un sistema con “conexión en caliente” es posible agregar o eliminar dispositivos mientras la computadora está en ejecución. Como resultado, mientras un módulo de E/S está ocupado leyendo de algún dispositivo, debe poder informar que el usuario ha quitado de manera repentina ese dispositivo del sistema, de forma que cualquier petición pendiente del ahora desaparecido dispositivo debe eliminarse del sistema, avisando a los que hicieron la llamada.

Software de E/S independiente del dispositivo

Aunque parte del software de E/S es específico para cada dispositivo mediante su driver, otras partes de éste son independientes de los dispositivos. El límite exacto entre los drivers y el software independiente del dispositivo depende del sistema y del dispositivo puesto que hay algunas tareas que, aunque independientes del dispositivo, las realiza el driver, principalmente por cuestiones de eficacia.

Las funciones del software de E/S independiente del dispositivo son las siguientes:

- Realizar las tareas de E/S comunes a todos los dispositivos, proporcionando una interfaz uniforme del software a nivel de usuario.
- Asociar los nombres simbólicos de un dispositivo con el nombres adecuado.
- Proporcionar esquemas de protección a los dispositivos. En sistemas monousuario, como MS-DOS, no existe protección, pero en sistemas multiusuario o multitarea si, por ejemplo en UNIX se asocian tres bits (RWX – Lectura Escritura Ejecución) para la protección de los dispositivos.
- Proporcionar un tamaño de bloque a los niveles superiores de software, de forma que las capas superiores trabajan con tamaños de bloques lógicos independientes del tamaño del sector físico del dispositivo con el que hacen la operación de E/S.
- Utilización de buffers, de manera que aunque la E/S se haga por bloques, los procesos de

usuario puedan leer y escribir unidades arbitrarias.

- Asignación y liberación de dispositivos de uso exclusivo, como por ejemplo una impresora. Algunos de los dispositivos de entrada y salida, como los discos, pueden ser utilizados por varios usuarios al mismo tiempo. No ocurren problemas si varios usuarios tienen archivos abiertos en el mismo disco dura al mismo tiempo. Otros dispositivos como las impresoras, deben dedicarse a solo un usuario hasta concluir con el.
- Informar a los niveles superiores de errores que se puedan producir.

Interfaz uniforme para los controladores de software de dispositivos

Una importante cuestión en un sistema operativo es cómo hacer que todos los dispositivos de E/S y sus drivers se vean más o menos iguales. Si los discos, las impresoras, los teclados, etc., se conectan de distintas maneras, cada vez que llegue un nuevo dispositivo el sistema operativo deberá modificarse para éste. No es conveniente tener que modificar el sistema operativo para cada nuevo dispositivo. Un aspecto de esta cuestión es la interfaz entre los drivers de dispositivos y el resto del sistema operativo. En la figura 5-14(a) ilustramos una situación en la que cada driver de dispositivo tiene una interfaz distinta con el sistema operativo. Lo que esto significa es que las funciones de driver disponibles para que el sistema pueda llamarlas difieren de un controlador a otro. Además, podría significar que las funciones de kernel que el driver necesita también difieren de driver en driver. En conjunto, quiere decir que la interfaz para cada nuevo driver requiere mucho esfuerzo de programación.

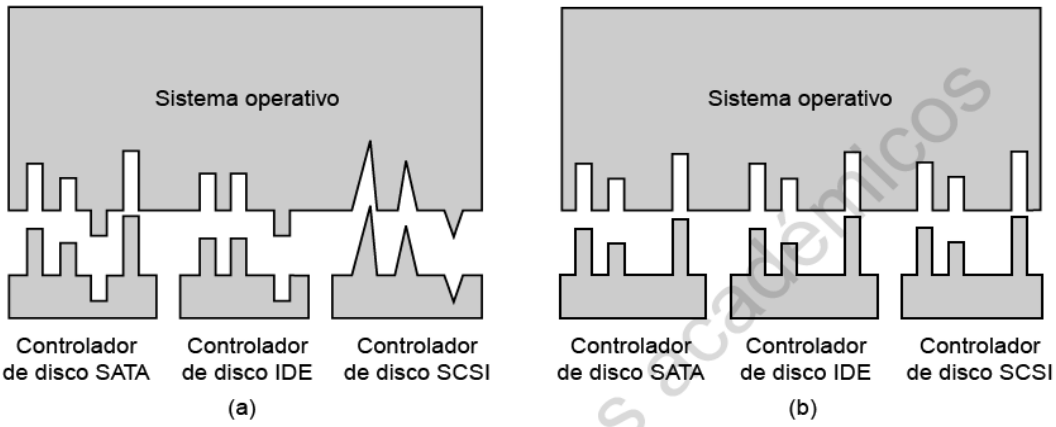


Figura 5-14. (a) Sin una interfaz de controlador estándar. (b) Con una interfaz de controlador estándar.

Por el contrario, en la figura 5-14(b) podemos ver un diseño distinto, en el que todos los drivers tienen la misma interfaz. Ahora es mucho más fácil conectar un nuevo driver, siempre y cuando sea compatible con su interfaz. Esto también significa que los escritores de los drivers saben lo que se espera de ellos. En la práctica no todos los dispositivos son absolutamente idénticos, pero por lo general hay sólo un pequeño número de tipos de dispositivos, e incluso éstos en general son casi iguales.

La manera en que funciona es la siguiente. Para cada clase de dispositivos, como los discos o las impresoras, el sistema operativo define un conjunto de funciones que el driver debe proporcionar. Para un disco, estas funciones incluyen naturalmente la lectura y la escritura, pero también encender

y apagar la unidad, aplicar formato y otras cosas relacionadas con los discos. A menudo, el driver contiene una tabla de punteros a sí mismo para estas funciones. Cuando se carga el driver, el sistema operativo registra la dirección de esta tabla de punteros a funciones, por lo que cuando necesita llamar a una de las funciones, puede hacer una llamada indirecta a través de esta tabla. Esta tabla de punteros a funciones define la interfaz entre el driver y el resto del sistema operativo. Todos los dispositivos de una clase dada (discos, impresoras, etc.) deben obedecerla.

Reporte de errores

Los errores son mucho más comunes en el contexto de la E/S que en otros. Cuando ocurren, el sis tema operativo debe manejarlos de la mejor manera posible. Muchos errores son específicos de cada dispositivo y el driver apropiado debe manejarlos, pero el marco de trabajo para el manejo de errores es independiente del dispositivo.

Los errores de programación son una clase de errores de E/S. Éstos ocurren cuando un proceso pide algo imposible, como escribir en un dispositivo de entrada (teclado, escáner, ratón, etc.) o leer de un dispositivo de salida (una impresora o un plotter, por ejemplo). Otros errores son proporcionar una dirección de búfer inválida o algún otro parámetro, y especificar un dispositivo inválido (por ejemplo, el disco 3 cuando el sistema sólo tiene dos), entre otros. La acción a tomar en estos errores es simple: sólo se reporta un código de error al que hizo la llamada.

Otra clase de errores son los de E/S reales; por ejemplo, tratar de escribir un bloque de disco dañado o tratar de leer de una cámara de video que está apagada. En estas circunstancias depende del driver determinar qué hacer. Si el driver no sabe qué hacer, puede pasar el problema de vuelta al software independiente del dispositivo. Lo que hace este software depende del entorno y la naturaleza del error. Si se trata de un error simple de lectura y hay un usuario interactivo disponible, puede mostrar un cuadro de diálogo pidiendo al usuario lo que debe hacer. Las opciones pueden incluir volver a intentar cierto número de veces, ignorar el error o eliminar el proceso que hizo la llamada. Si no hay usuario disponible, tal vez la única opción real sea hacer que la llamada al sistema falle con un código de error.

Sin embargo, algunos errores no se pueden manejar de esta forma. Por ejemplo, una estructura de datos crítica, como por ejemplo el directorio raíz puede haberse destruido. En este caso, el sistema tal vez tenga que mostrar un mensaje de error y terminar.

Asignación y liberación de dispositivos dedicados

Algunos dispositivos, como los grabadores de CD-ROM, sólo pueden ser utilizados por un solo proceso en un momento dado. Es responsabilidad del sistema operativo examinar las peticiones de uso de los dispositivos y aceptarlas o rechazarlas, dependiendo de si el dispositivo solicitado está o no disponible. El software independiente del dispositivo, en un intento por adquirir un dispositivo que no está disponible debe emitir un mensaje de error o puede bloquear al proceso que hizo la llamada, en vez de fallar. En este segundo caso los procesos bloqueados se ponen en una cola. Tarde o temprano, el dispositivo solicitado estará disponible y el primer proceso en la cola podrá adquirirlo para continuar su ejecución. A esto se le conoce también como planificación de la E/S.

Tamaño de bloque independiente del dispositivo

Los distintos discos pueden tener diferentes tamaños de sectores. Es responsabilidad del software independiente del dispositivo ocultar este hecho y proporcionar un tamaño de bloque uniforme a los niveles superiores; por ejemplo, al tratar varios sectores como un solo bloque lógico. De esta forma, los niveles superiores lidian sólo con dispositivos abstractos que utilizan todos el mismo

tamaño de bloque lógico, sin importar el tamaño del sector físico. De manera similar, algunos dispositivos de carácter envían sus datos un byte a la vez (como los módems), mientras que otros envían sus datos en unidades más grandes (como las interfaces de red). Estas diferencias también se pueden ocultar a nivel de usuario.

Software de E/S en espacio de usuario

Aunque la mayor parte del software de E/S está dentro del sistema operativo, una pequeña porción de éste consiste en bibliotecas vinculadas entre sí con programas de usuario, e incluso programas enteros que se ejecutan desde el exterior del kernel. Las llamadas al sistema, incluyendo las llamadas al sistema de E/S, se realizan comúnmente mediante procedimientos de biblioteca. Cuando un programa en C contiene, por ejemplo, la llamada

cuenta = write(da, bufer, nbytes);

el procedimiento de biblioteca *write()* se vinculará y se incluirá en el programa binario presente en memoria en tiempo de ejecución (cuando compilamos y linkamos). La colección de todos estos procedimientos de biblioteca es sin duda parte del sistema de E/S.

Ese procedimiento *write()* después podrá hacer uso de funciones asociadas a un driver de dispositivo como se comentó en secciones anteriores. Recuerde que una de las tareas de un driver es aceptar peticiones abstractas de lectura y escritura del software independiente del dispositivo que está por encima de él (programa de usuario por ejemplo), y ver que se lleven a cabo.

Aunque estos procedimientos de biblioteca hacen algo más que colocar sus parámetros en el lugar apropiado para la llamada al sistema, hay otros procedimientos de E/S que en realidad realizan un trabajo real antes de invocar a una llamada del sistema (*por ejemplo write()*). En especial, el formato de la entrada y la salida se lleva a cabo mediante procedimientos de biblioteca. Un ejemplo de C es *printf()*, que toma una cadena de formato y posiblemente unas variables como entrada, construye una cadena ASCII y después invoca a la llamada de sistema *write()* para imprimir la cadena. Como ejemplo de *printf()*, considere la instrucción

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

Esta instrucción da formato a una cadena que consiste en la cadena de 14 caracteres "El cuadrado de" seguida por el valor *i* como una cadena de 3 caracteres, después la cadena de 4 caracteres " es ", luego *i*i* como 6 caracteres, y por último un salto de línea.

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

No todo el software de E/S de bajo nivel consiste en procedimientos de biblioteca. Otra categoría importante es el sistema de colas. El uso de colas (spooling) es una manera de lidiar con los dispositivos de E/S dedicados en un sistema de multiprogramación. Considere un dispositivo común que utiliza colas: una impresora. Aunque sería técnicamente sencillo dejar que cualquier proceso de usuario abriera el archivo de caracteres especial para la impresora, suponga que un proceso lo abriera y no hiciera nada durante horas. Ningún otro proceso podría imprimir nada. En vez de ello, lo que se hace es crear un proceso especial, conocido como demonio, y un directorio especial llamado directorio de cola de impresión. Para imprimir un archivo, un proceso genera primero todo el archivo que va a imprimir y lo coloca en el directorio de la cola de impresión. Es responsabilidad del demonio, que es el único proceso que tiene permiso para usar el archivo especial de la

impresora, imprimir los archivos en el directorio. Al proteger el archivo especial contra el uso directo por parte de los usuarios, se elimina el problema de que alguien lo mantenga abierto por un tiempo innecesariamente extenso.

En la figura 5-17 se resume el sistema de E/S, donde se muestran todos los niveles y las funciones principales de cada nivel. Empezando desde la parte inferior, los niveles son el hardware (incluyendo los módulos de E/S), los manejadores de interrupciones, los drivers de dispositivos (nombrado en la figura como controlador de dispositivo), el software independiente del dispositivo y, por último, los procesos de usuario.

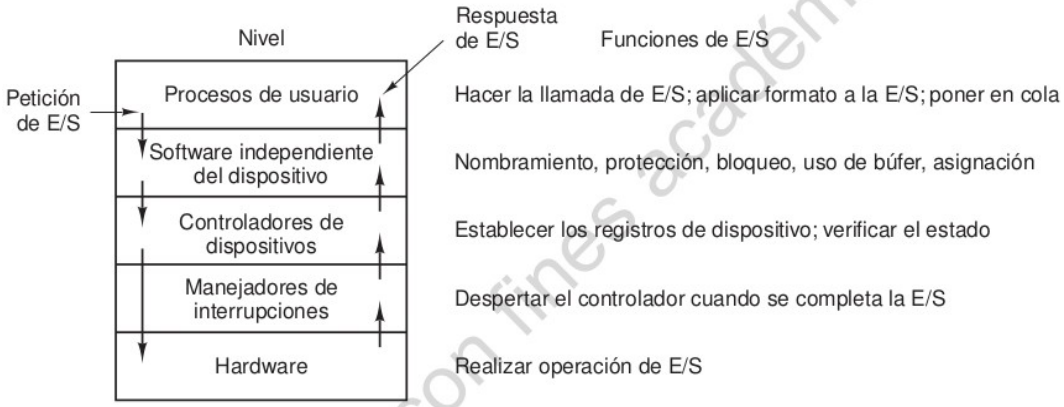


Figura 5-17. Niveles del sistema de E/S y las funciones principales de cada nivel.

Las flechas en la figura 5-17 muestran el flujo de control. Por ejemplo, cuando un programa de usuario trata de leer un bloque de un archivo, se invoca el sistema operativo para llevar a cabo la llamada. El software independiente del dispositivo llama al driver (controlador del dispositivo en la figura) para enviar la petición al hardware (incluyendo módulos de E/S) y obtenerlo del disco. Después, el proceso se bloquea hasta que se haya completado la operación de disco.

Cuando termina el disco, el hardware (incluyendo módulo de E/S) genera una interrupción. El manejador de interrupciones se ejecuta para descubrir qué ocurrió; es decir, qué dispositivo desea atención en ese momento. Después extrae el estado del dispositivo y si es necesario invocar al driver para tratar el estado se invoca. A continuación el software independiente del dispositivo despierta al proceso inactivo para que termine la petición de E/S y deje que el proceso de usuario continúe.