Interrupciones

Prácticamente todos los computadores proporcionan un mecanismo para mejorar el uso del procesador y por el cual otros módulos como los de E/S pueden interrumpir el secuenciamiento normal del mismo. La Tabla 1.1 detalla los tipos más comunes de interrupciones del secuenciamiento normal de instrucciones del procesador, las dos primeras se refieren a interrupciones software y las dos últimas a interrupciones hardware. Aquí se habla de manera general del concepto de interrupción, posteriormente se estudiará más específicamente las interrupciones de E/S, especificando tres tipos.

Tabla 1.1. Clases de interrupciones.

De programa	Generada por alguna condición que se produce como resultado de la ejecución de una instrucción, tales como un desbordamiento aritmético, una división por cero, un intento de ejecutar una instrucción de máquina ilegal, y las referencias fuera del espacio de la memoria permitido para un usuario.
Por temporizador	Generada por un temporizador del procesador. Permite al sistema operativo realizar ciertas funciones de forma regular.
De E/S	Generada por un controlador de E/S para señalar la conclusión normal de una operación o para indicar diversas condiciones de error.
Por fallo del hardware	Generada por un fallo, como un fallo en el suministro de energía o un error de paridad en la memoria.

La mayoría de los dispositivos de E/S son mucho más lentos que el procesador. Supóngase que el procesador está transfiriendo datos a una impresora (o grabando datos en un CD virgen) utilizando el esquema de ciclo de instrucción de la Figura 1.2.

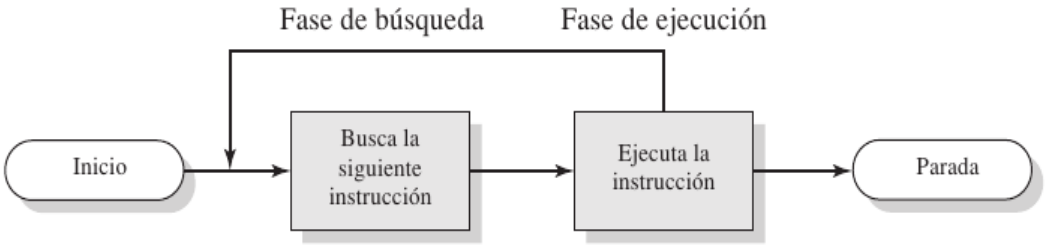
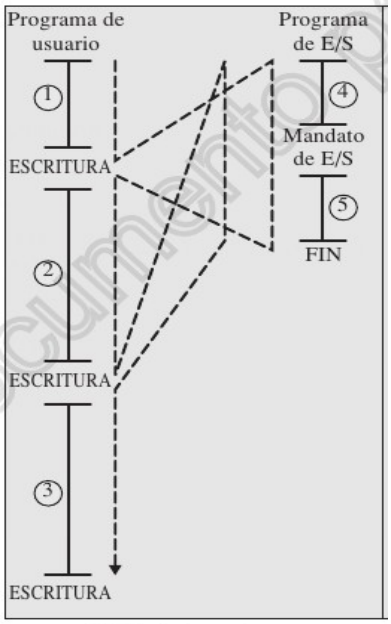


Figura 1.2. Ciclo de instrucción básico.

Suponga el envío de los datos a la memoria de la impresora. Después de cada instrucción de escritura, el procesador debe parar y permanecer inactivo hasta que la impresora lleve a cabo su labor (vacíe su buffer local). La longitud de esta pausa puede ser del orden de muchos miles o incluso millones de ciclos de instrucción. Claramente, es un enorme desperdicio de la capacidad del procesador. Para dar un ejemplo concreto, considere un computador personal que operase a 1GHz, lo que le permitiría ejecutar aproximadamente 10^9 instrucciones por segundo. Un típico disco duro tiene una velocidad de rotación de 7200 revoluciones por minuto, que corresponde con un tiempo de rotación de media pista de 4 ms., que es 4 millones de veces más lento que el procesador.

Continuando con el programa de usuario que hace una petición para imprimir en una impresora, suponga que éste realiza una serie de llamadas de ESCRITURA intercaladas con el procesamiento del resto del código que debe ejecutar (la siguiente figura muestra esta cuestión).



(a) Sin interrupciones

Los segmentos de código 1, 2 y 3 se refieren a secuencias de instrucciones que no involucran E/S. Las llamadas de ESCRITURA invocan a una o varias rutinas de E/S del núcleo cargadas en memoria principal (4 y 5), produciéndose lo siguiente:

a) Se ejecuta una secuencia de instrucciones, etiquetadas como 4 en la figura, para preparar la operación real de E/S. Esto incluye preparar los parámetros de uno o varios mandatos del núcleo para tratar con el dispositivo (impresora) y comprobar si está o no disponible, además de la realización de la transferencia de datos en si. Sin el uso de interrupciones, una vez que se emite el mandato de transferencia, el programa debe esperar a que el dispositivo de E/S realice la función solicitada, comprobando periódicamente el estado del dispositivo. El programa podría esperar simplemente realizando repetidamente una operación de comprobación para determinar si se ha realizado la operación de E/S.

b) Una vez se realiza la operación de E/S ***por parte del dispositivo*** (impresora ha vaciado su buffer y ha impreso) se produce secuencia de instrucciones, etiquetada como 5 en la figura, para indicar al procesador que ya ha terminado la operación, indicando por ejemplo el éxito o el fallo de la misma.

Debido a que la operación de E/S puede tardar un tiempo relativamente largo hasta que se completa, el procesador se queda esperando a que se termine; por ello, el programa de usuario (proceso que se esté ejecutando en ese momento) se detiene en la llamada de ESCRITURA durante un periodo de tiempo considerable.

Todavía no se ha introducido lo que es un cambio de contexto, pero cabe decir aquí, para cuando se hable de ello, que hasta ahora no se ha producido cambio de contexto en un proceso para salvar su estado, sino que ante una operación de envío o recepción de datos del procesador a otro dispositivo, se ejecutan una o varias rutinas del núcleo que están cargadas en memoria principal. Estas rutinas las ejecuta el procesador como otro programa más mediante un salto a una determinada posición de la memoria, teniendo que esperar a que se produzca la transferencia de información entre CPU y dispositivo para poder seguir con la siguiente instrucción del programa que actualmente está en ejecución.

Dicho esto podemos entonces introducir el concepto de interrupción de manera genérica. Gracias a las interrupciones el procesador podrá dedicarse a ejecutar otras instrucciones mientras que la operación de E/S se está llevando a cabo. Considere el flujo de control mostrado en la figura 1.5b de más abajo. Como anteriormente, el programa de usuario alcanza un punto en el que hace una

llamada al sistema que consiste en una llamada de ESCRITURA (puede involucrar una o varias llamadas a subrutinas del núcleo del sistema) y pone datos en el buffer de la impresora (posteriormente veremos que esto lo puede hacer un módulo de E/S). En ese momento (una vez transferidos los datos a la impresora) el procesador podría ponerse a hacer otras cosas mientras que paralelamente el dispositivo externo realiza su tarea. Cuando el dispositivo externo está listo para ser atendido, es decir, cuando la impresora ha vaciado su buffer y ha impreso los datos correspondientes, manda una señal de petición de **interrupción** al procesador. El procesador responde suspendiendo la ejecución del programa actual (se estudiará que esto requerirá un salvado de contexto), saltando a una rutina de servicio (*Interrupt Service Routine* – ***ISR***) específica de ese dispositivo de E/S, conocida como **manejador de interrupción**, y reanudando la ejecución original después de haber atendido al dispositivo. Atender al dispositivo, si no pensamos explícitamente en la impresora, puede significar varias cosas: 1) simplemente inspeccionar que la operación se ha llevado con éxito o ha fracasado, 2) en el caso de que se estén esperando datos del dispositivo para tratarlos por la CPU o almacenarlos en memoria, la ISR tendría que enviarlos al procesador (veremos después que los datos se pueden almacenar en un módulo de E/S actuando como buffer intermedio), 3) el dispositivo puede estar pidiendo al sistema algún tipo de datos, en este caso la ISR debe dar paso a la rutina o rutinas necesarias para proceder con ello.

En la Figura 1.5b anterior se indican con una "X" los puntos en los que se produce cada interrupción. Téngase en cuenta que se puede producir una interrupción en cualquier punto de la ejecución del programa principal, no sólo en una de determinada instrucción, y que dicha interrupción se produce desde el dispositivo externo hacia el procesador. En este ejemplo, una vez ejecutado el circulo 4 se pasa al circulo 2a, pero en muchos casos el procesador puede volver al programa que estaba ejecutando o ejecutar otro distinto, por ejemplo porque el proceso interrumpido necesite los datos de una E/S para continuar con la siguiente instrucción (esto requerirá salvados y restauración de contextos).

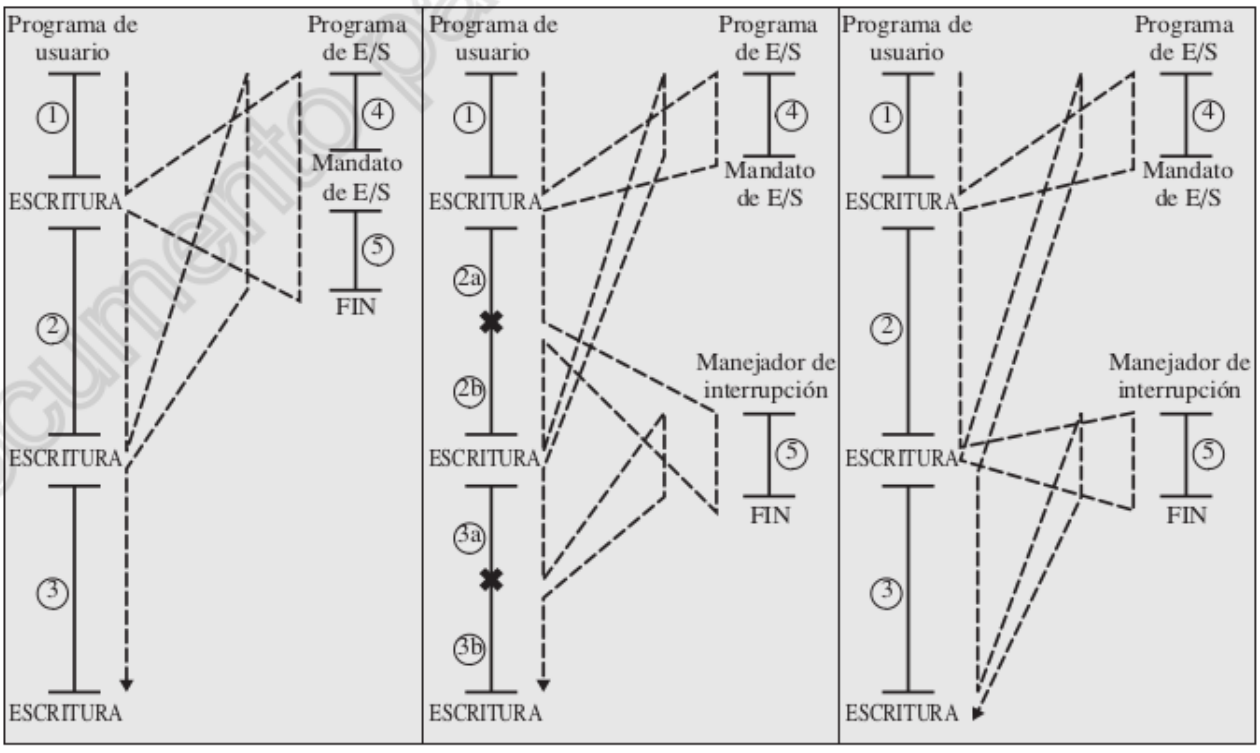


Figura 1.5. Flujo de programa del control sin interrupciones y con ellas.

De cara al programa de usuario (programa actualmente ejecutándose), una interrupción suspende la secuencia normal de ejecución. Cuando se completa el tratamiento de la interrupción por parte del

manejador de interrupciones, se reanuda la ejecución, tal y como se muestra en la Figura 1.6 (esto requerirá salvados y restauración de contextos). Por tanto, el programa de usuario no tiene que contener ningún código especial para tratar las interrupciones; el procesador y el sistema operativo son responsables de suspender el programa de usuario y, posteriormente, reanudarlo en el mismo punto.

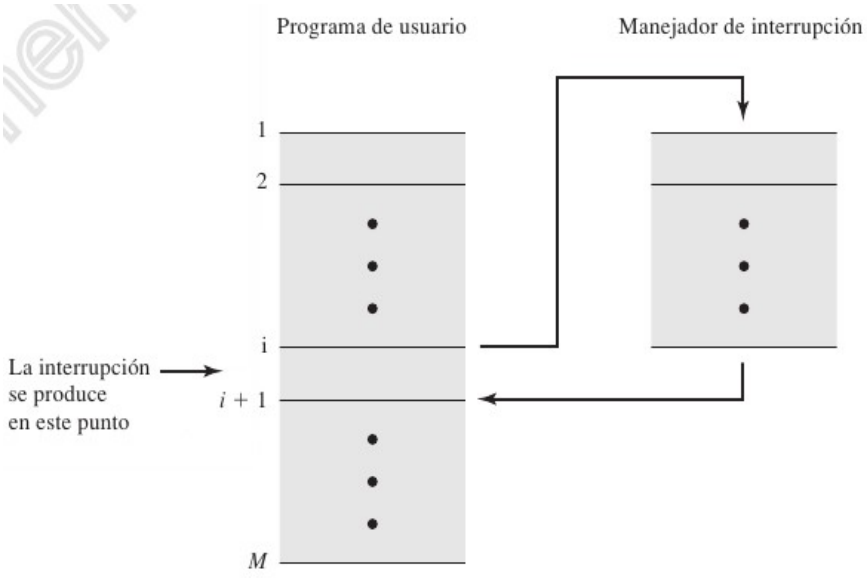


Figura 1.6. Transferencia de control mediante interrupciones.

Para poder usar los conceptos anteriormente descritos, es necesario añadir una fase de interrupción al ciclo de instrucción, como se muestra en la Figura 1.7 (compárese con la Figura 1.2).

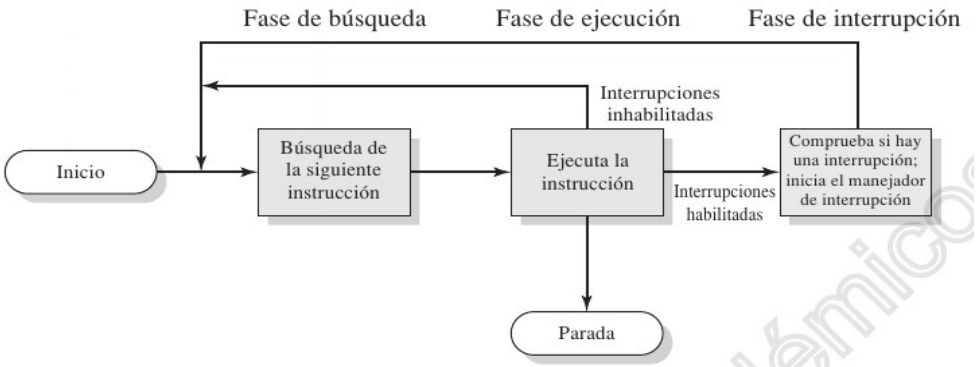


Figura 1.7. Ciclo de instrucción con interrupciones.

En la fase de interrupción, el procesador comprueba si se ha producido cualquier interrupción, hecho indicado por la presencia de una señal de interrupción (registro del procesador, suele ser una comprobación hardware) que mandó un dispositivo externo en caso de ser una interrupción hardware. Si no hay interrupciones pendientes, el procesador continúa con la fase de búsqueda y lee la siguiente instrucción del programa actual. Si está pendiente una interrupción, el procesador suspende la ejecución del programa actual y ejecuta una rutina o programa de manejado de interrupción (*Interrupt Service Routine – ISR*) comentada anteriormente, la cual determina la naturaleza de la interrupción y realiza las acciones que se requieran, volviéndose después a reanudar la ejecución del programa de usuario en el punto de la interrupción.

Para apreciar la ganancia en eficiencia, considere la Figura 1.8, que es un diagrama de tiempo basado en el flujo de control de las Figuras 1.5a y 1.5b.

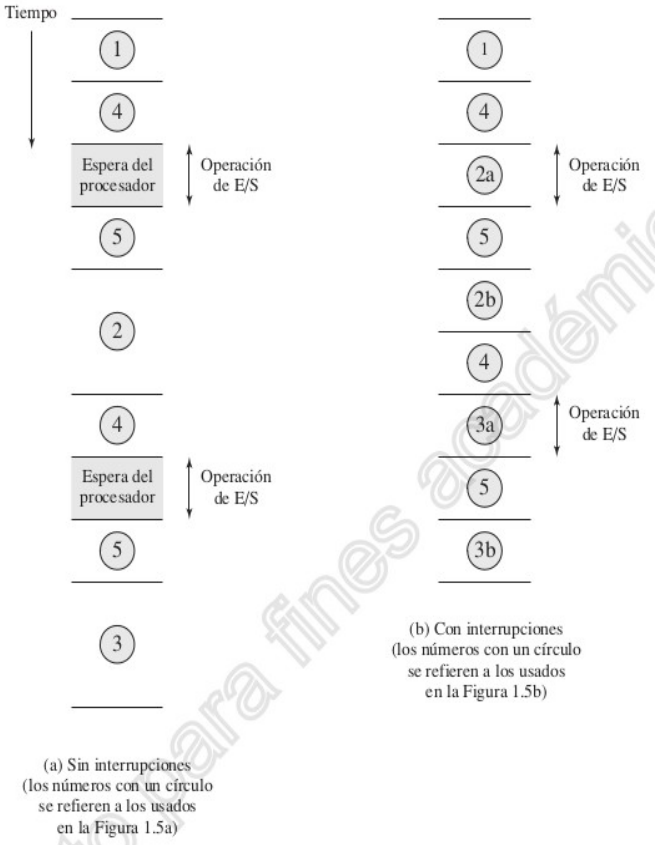


Figura 1.8. Temporización del programa: espera breve de E/S.

Salvado del contexto de un proceso debido a una interrupción

La aparición de una interrupción dispara varios eventos, tanto en el hardware del procesador como en el software. Cuando un dispositivo de E/S completa una operación de E/S (impresora que termina de imprimir un bloque de información, cliente externo que termina de enviar datos a un servidor, etc) se produce la siguiente secuencia de eventos en el hardware (suponemos que el procesador esta ejecutando normalmente alguna determinada tarea cuando se produce la interrupción):

1. El dispositivo genera una señal de interrupción hacia el procesador en una de las lineas o buses hacia el procesador habilitados para ello, modificando así un registro.

2. El procesador no sigue con la siguiente instrucción una vez ha completado la que estaba ejecutando en el momento que se produjo la interrupción, ya que comprueba si hay una petición de interrupción pendiente, determina que hay una y de qué dispositivo o tipo de interrupción se trata (consultando un vector o tabla de tipos de interrupción, la cual puede estar incorporada incluso en el hardware), y manda una señal de reconocimiento (acuse de recibo) y comprobación de estado al dispositivo que produjo la interrupción. Dependiendo de la arquitectura de computador y del diseño del sistema operativo, puede haber un único programa o rutina de interrupción o varios, uno por cada tipo de interrupción o por cada tipo de dispositivo. Si hay más de una rutina de manejo de interrupción, el procesador debe determinar cuál invocará. Esta información puede estar incluida en la señal de interrupción original o, en ocasiones, el procesador puede tener que realizar una petición al dispositivo que generó la interrupción para obtener una respuesta que contiene la información requerida.

3. En ese momento, el procesador necesita prepararse para transferir el control a la rutina de interrupción (puede haber varias, dependiendo del dispositivo). Para comenzar, necesita salvar la

información requerida para reanudar el programa actual en el momento de la interrupción. La información mínima requerida es la palabra de estado del programa (PSW) y la posición de la siguiente instrucción que se va a ejecutar, que está contenida en el contador de programa (PC). Esta información se puede apilar en la pila de control de sistema, y en otros sistemas se almacena en el bloque de control de proceso del proceso actual.

4. A continuación, el procesador carga el contador del programa con la posición del punto de entrada de la rutina de manejo de interrupción (*ISR*) que tratará a esta interrupción.

5. Una vez que se ha cargado el contador del programa, el procesador continúa con el siguiente ciclo de instrucción, que comienza con una lectura de instrucción. Dado que la lectura de la instrucción está determinada por el contenido del contador del programa, el resultado es que se transfiere el control al programa manejador de interrupción. La ejecución de este programa o *ISR* conlleva las siguientes operaciones:

> 5.1) En este momento, el PC y la PSW vinculados con el programa interrumpido se han almacenado en la pila del sistema. Sin embargo, hay otra información que se considera parte del estado del programa en ejecución. En concreto, se necesita salvar el contenido de los registros del procesador, puesto que estos registros los podría utilizar el manejador de interrupciones. Por tanto, se deben salvar todos estos valores, así como cualquier otra información de estado (es lo que conforma el Bloque de Control de Proceso o BCP). Generalmente, el manejador de interrupción comenzará salvando el contenido de todos los registros en la pila o en el BCP del proceso almacenado en la tabla de procesos del sistema.
>
> 5.2) El manejador de interrupción puede en este momento comenzar a procesar la interrupción. Esto incluirá un examen de la información de estado relacionada con la operación de E/S (éxito o fracaso) o con otro evento distinto que haya causado la interrupción (interrupción software). Asimismo, puede implicar el envío de datos del dispositivo de E/S hacia la CPU o viceversa.
>
> 5.3) Cuando se completa el procesamiento de la interrupción, se recuperan los valores de los registros salvados en la pila y se restituyen en los registros (contexto del proceso).
>
> 5.4) La última acción consiste en restituir de la pila los valores de la PSW y del contador del programa.

6) Como resultado, la siguiente instrucción que se va ejecutar corresponderá al programa previamente interrumpido. En muchas ocasiones el planificador del sistema operativo puede optar por ejecutar un programa diferente al que se interrumpió por el hecho de que tenga mayor prioridad. Luego no siempre se restaura el mismo programa que se interrumpe.

Es importante salvar toda la información de estado del programa interrumpido para su posterior reanudación (es la novedad que se ha introducido en este apartado con respecto al anterior). La interrupción puede suceder en cualquier momento y, por tanto, en cualquier punto de la ejecución de un programa de usuario. Su aparición es imprevisible.

Múltiples interrupciones

El estudio realizado hasta el momento ha tratado solamente el caso de que se produzca una única interrupción. Supóngase, sin embargo, que se producen múltiples interrupciones, es decir, mientras se esta tratando por *ISR* una interrupción se produce otra. Por ejemplo, un programa puede estar recibiendo datos de una línea de comunicación e imprimiendo resultados al mismo tiempo. La impresora generará una interrupción cada vez que completa una operación de impresión. El controlador de la línea de comunicación generará una interrupción cada vez que llega una unidad de datos. La unidad podría consistir en un único carácter o en un bloque, dependiendo de la naturaleza del protocolo de comunicaciones. En cualquier caso, es posible que se produzca una interrupción de comunicación mientras se está procesando una interrupción de la impresora.

Se pueden considerar dos alternativas a la hora de tratar con múltiples interrupciones:

1) La primera es inhabilitar las interrupciones mientras que se está procesando una interrupción. Una interrupción inhabilitada significa simplemente que el procesador ignorará cualquier nueva señal de petición de interrupción. Si se produce una interrupción durante este tiempo, generalmente permanecerá pendiente de ser procesada, de manera que el procesador sólo la comprobará después de que se rehabiliten las interrupciones. Por tanto, cuando se ejecuta un programa de usuario y se produce una interrupción, se inhabilitan las interrupciones inmediatamente. Después de que se completa la rutina de manejo de la interrupción *ISR*, se rehabilitan las interrupciones antes de reanudar el programa de usuario, y el procesador comprueba si se han producido interrupciones adicionales. Esta estrategia es válida y sencilla, puesto que las interrupciones se manejan en estricto orden secuencial (Figura 1.12a).

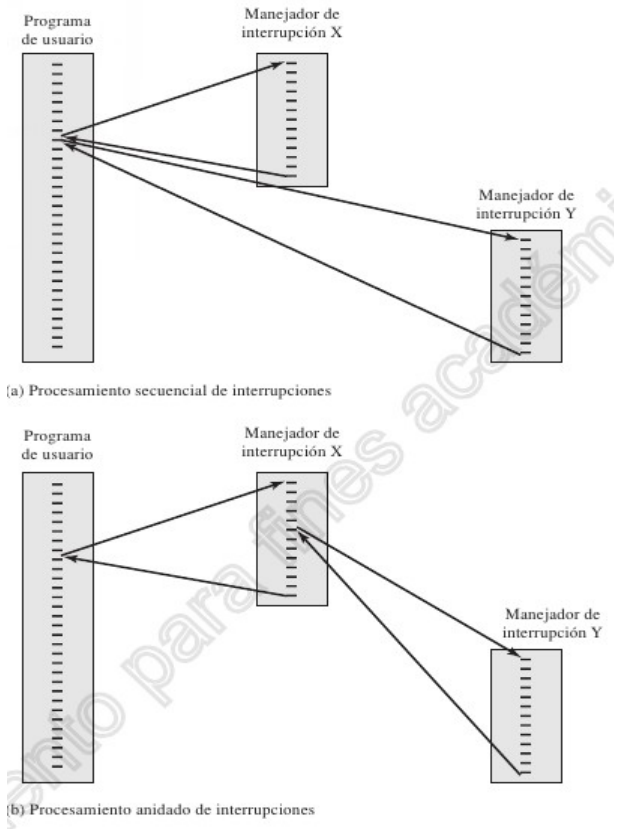


Figura 1.12. Transferencia de control con múltiples interrupciones.

La desventaja de la estrategia anterior es que no tiene en cuenta la prioridad relativa o el grado de urgencia de las interrupciones. Por ejemplo, cuando llegan datos por la línea de comunicación, se puede necesitar que se procesen rápidamente de manera que se deje sitio para otros datos que pueden llegar. Si el primer lote de datos no se ha procesado antes de que llegue el segundo, los datos pueden perderse porque el buffer del dispositivo de E/S puede llenarse y desbordarse.

2) Una segunda estrategia es definir prioridades para las interrupciones y permitir que una interrupción de más prioridad cause que se interrumpa la ejecución de un manejador de una interrupción de menor prioridad (Figura 1.12b). Como ejemplo de esta segunda estrategia, considere un sistema con tres dispositivos de E/S: una impresora, un disco y una línea de comunicación, con prioridades crecientes de 2, 4 y 5, respectivamente (suponga que un 5 significa más prioridad que un 2 y un 3). La Figura 1.13, muestra una posible secuencia. Un programa de usuario comienza en $t = 0$. En $t = 10$, se produce una interrupción de impresora; se almacena la información de usuario en la pila del sistema y la ejecución continúa en la rutina de servicio de interrupción (Interrupt Service Routine, ISR) de la impresora. Mientras todavía se está ejecutando

esta rutina, en *t = 15* se produce una interrupción del equipo de comunicaciones. Debido a que la línea de comunicación tiene una prioridad superior a la de la impresora, se sirve la petición de interrupción. Se interrumpe la ISR de la impresora, se almacena su estado en la pila y la ejecución continúa con la ISR del equipo de comunicaciones. Mientras se está ejecutando esta rutina, se produce una interrupción del disco (*t = 20*). Dado que esta interrupción es de menor prioridad, simplemente se queda en espera, y la ISR de la línea de comunicación se ejecuta hasta su conclusión. Cuando se completa la ISR de la línea de comunicación (*t = 25*), se restituye el estado previo del proceso, que corresponde con la ejecución de la ISR de la impresora. Sin embargo, antes incluso de que pueda ejecutarse una sola instrucción de esta rutina, el procesador atiende la interrupción de disco de mayor prioridad y transfiere el control a la ISR del disco. Sólo cuando se completa esa rutina (*t = 35*), se reanuda la ISR de la impresora. Cuando esta última rutina se completa (*t = 40*), se de vuelve finalmente el control al programa de usuario (previa restauración de su contexto).

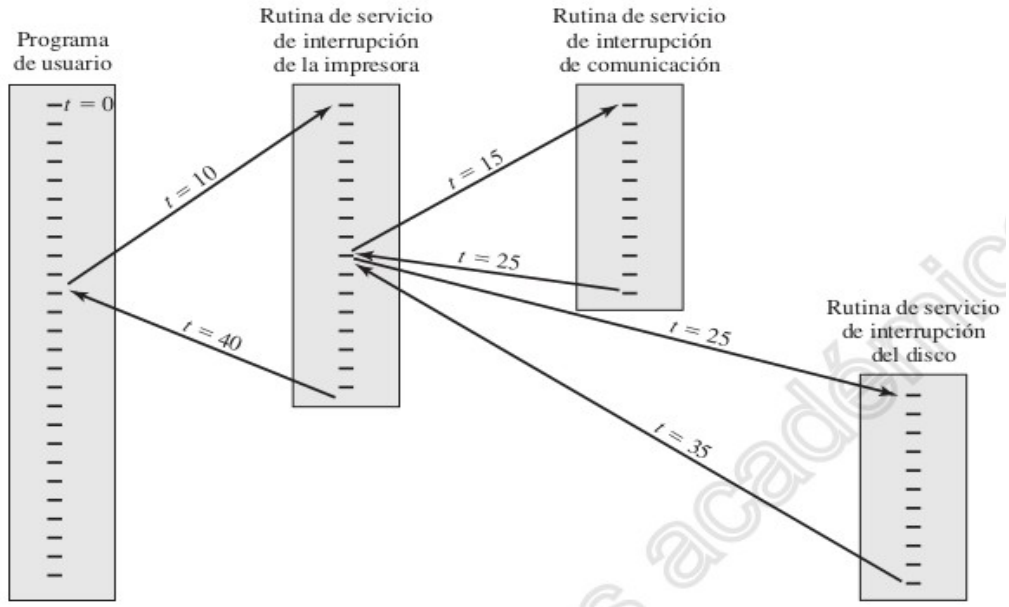


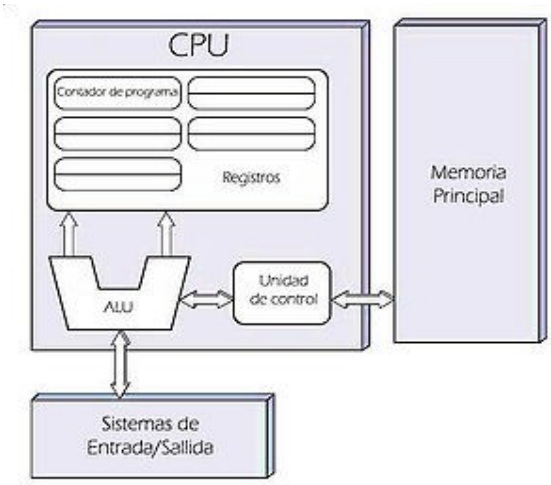
Figura 1.13. Ejemplo de secuencia de tiempo con múltiples interrupciones.

Sistema de Entrada-Salida (E/S)

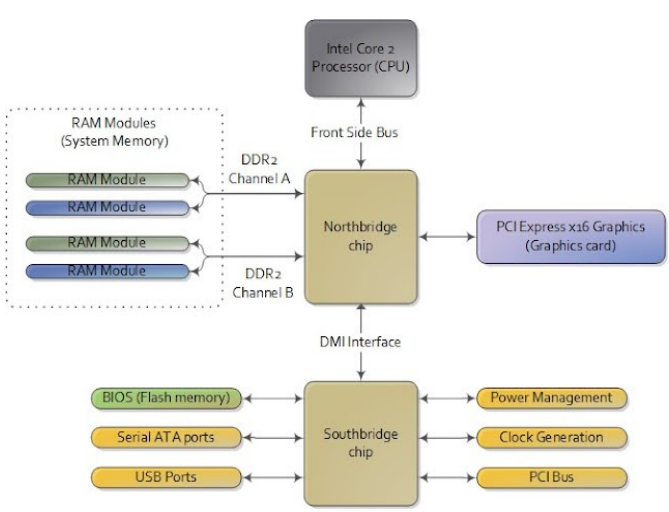
Los ordenadores actuales se basan en la arquitectura de Von Neumann. Los ordenadores con esta arquitectura constan de tres partes claramente diferencias: El procesador, la memoria, y uno o varios dispositivos de E/S. Para conectar estas partes se utiliza tres tipos básicos de buses:

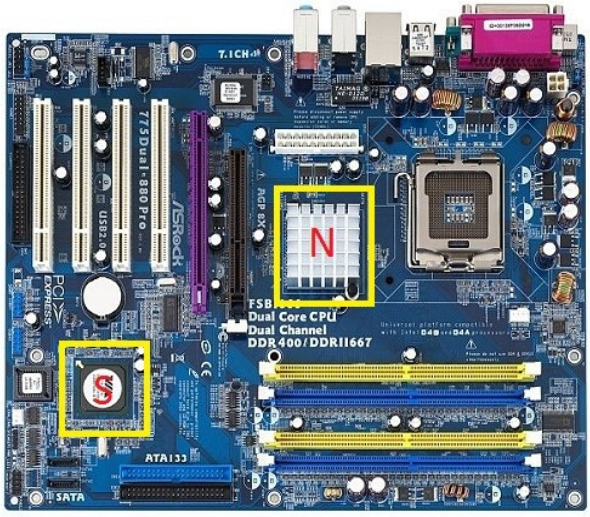
- Bus de datos, de 8, 16, 32 ó 64 bits dependiendo del modelo (64 bits para los ordenadores de última generación o actuales).
- Bus de direcciones, para poder conectar la CPU con la memoria y con los dispositivos de entrada/salida.
- Bus de control, para enviar señales que determinan cómo se comunica la CPU con el resto del sistema, es decir, sirven para llevar a cabo las normas de comunicación entre dos extremos o dispositivos.

Con esta arquitectura, debe haber por tanto, un sistema que se encargue de gestionar y comunicar a los dispositivos de E/S con el procesador, descargando a la CPU de tanto trabajo como sea posible, hablamos del sistema de E/S o módulo de E/S (indistintamente).

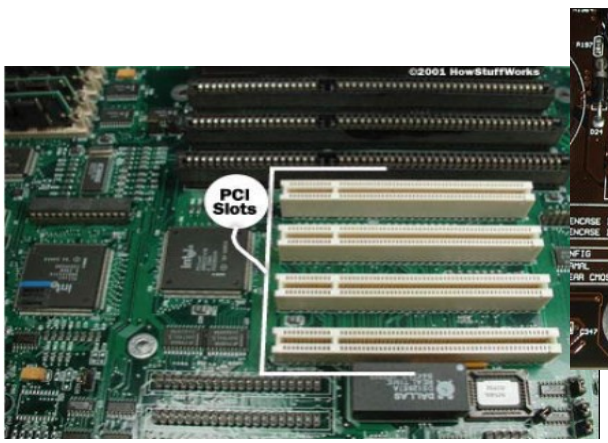


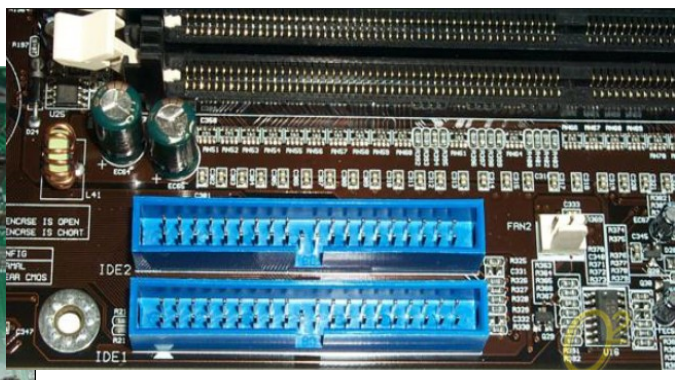
El sistema o módulo de E/S (también llamado **controlador**) es un circuito o chip integrado en la placa base, separado en muchos casos físicamente del procesador y la memoria pero unido a estos mediante buses en la misma placa. En muchos computadores este módulo es conocido como puente norte y puente sur.





Dicho chip transfiere y controla el flujo de información entre la memoria principal, el procesador y los periféricos. Los periféricos se conectan a la placa base a través de una serie de módulo físicos, por ejemplo un *slot pci* para una tarjeta de sonido o para extender puertos *usb* a una máquina que no los incluya, o *slots IDE* para discos duros.

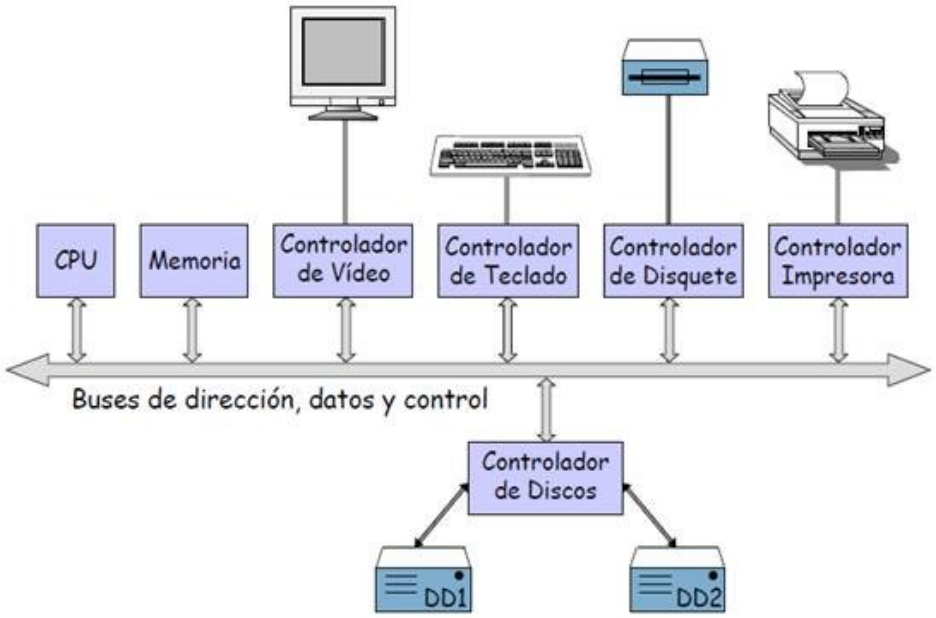




Un módulo de E/S puede controlar múltiples dispositivos (discos, impresoras, monitores, tarjetas de red, etc). El SO, normalmente, trata con el módulo de E/S, y no directamente con el dispositivo (se hizo referencia en el apartado de "Interrupciones"). Cada dispositivo de E/S está hecho por un fabricante diferente y puede tener diferentes registros y chips físicos y diferente manera de trabajar que otro dispositivo destinado al mismo uso pero de otro fabricante distinto. Se necesita entonces un conjunto de instrucciones que conformen un protocolo de comunicación entre el computador y el dispositivo físico (impresora, tarjeta gráfica externa, etc). Para ello cada fabricante crea un programa software llamado ***driver*** o **manejador de dispositivo (no confundir con *ISR*)**. Los *drivers* se implementan mediante módulos añadidos al núcleo del sistema operativo, y son objetos software con una interfaz bien definida para especificar al sistema operativo y al módulo de E/S cómo debe controlar y comunicarse con un dispositivo en particular.

El sistema de E/S tiene las siguientes funciones:

- Envío de comandos a los dispositivos, recibir sus interrupciones y ocuparse de sus errores.
- Ofrecer una interfaz entre los dispositivos y el resto del sistema, incluyendo la CPU, simple y fácil de usar.
- Optimizar la E/S del sistema. Los dispositivos de E/S son muy lentos en comparación con la CPU y si no se delega trabajo en los mismos dispositivos y en el modulo estaríamos haciendo un mal uso del procesador, ya que quedaría ocupado innecesariamente a la espera de datos en una operación de E/S.
- Permitir la conexión de nuevos dispositivos de E/S.
- Almacenamiento temporal de datos (buffer). Ya que la velocidad de acceso de la memoria es mucho más alta que la que proporcionan los dispositivos periféricos, el módulo de E/S dispone de una memoria local (rápida) con la que se comunica con la memoria y la CPU, así, puede recibir rápidamente un bloque de datos, liberar el bus, y luego escribirlo en el dispositivo a la velocidad que éste proporcione
- Detección de Errores. Debe ocuparse de detectar y comunicar a la CPU los errores mecánicos o eléctricos del dispositivo.



Técnicas de comunicación de E/S

Hay tres técnicas para llevar a cabo las operaciones de E/S: E/S programada, E/S dirigida por interrupciones y acceso directo a memoria (Direct Memory Access, DMA).

E/S programada

A continuación se expone como se realiza una operación de entrada salida sin el uso de interrupciones, pero ya usando el concepto de módulo de E/S actuando como intermediario entre procesador y dispositivo externo.

Cuando el procesador ejecuta un programa y encuentra una instrucción relacionada con la E/S (leer, escribir o simplemente comprobar el estado de un dispositivo), ejecuta esa instrucción generando uno o varios mandatos al módulo de E/S apropiado dentro del sistema de E/S. En el caso de la E/S programada, el módulo de E/S realiza la acción solicitada pero no realiza ninguna acción para avisar al procesador. Por tanto, después de que se invoca la instrucción de E/S, el procesador debe tomar un papel activo para determinar cuándo se completa la instrucción de E/S. Por este motivo, el procesador comprueba periódicamente el estado del módulo de E/S hasta que encuentra que se ha completado la operación. No confunda esto con una interrupción, aquí no se han producido interrupciones, sino que es el procesador, colaborando con el modulo de E/S, el que se comunica en dirección al dispositivo externo y no al revés, ya sea para enviar datos o para leer datos de un dispositivo.

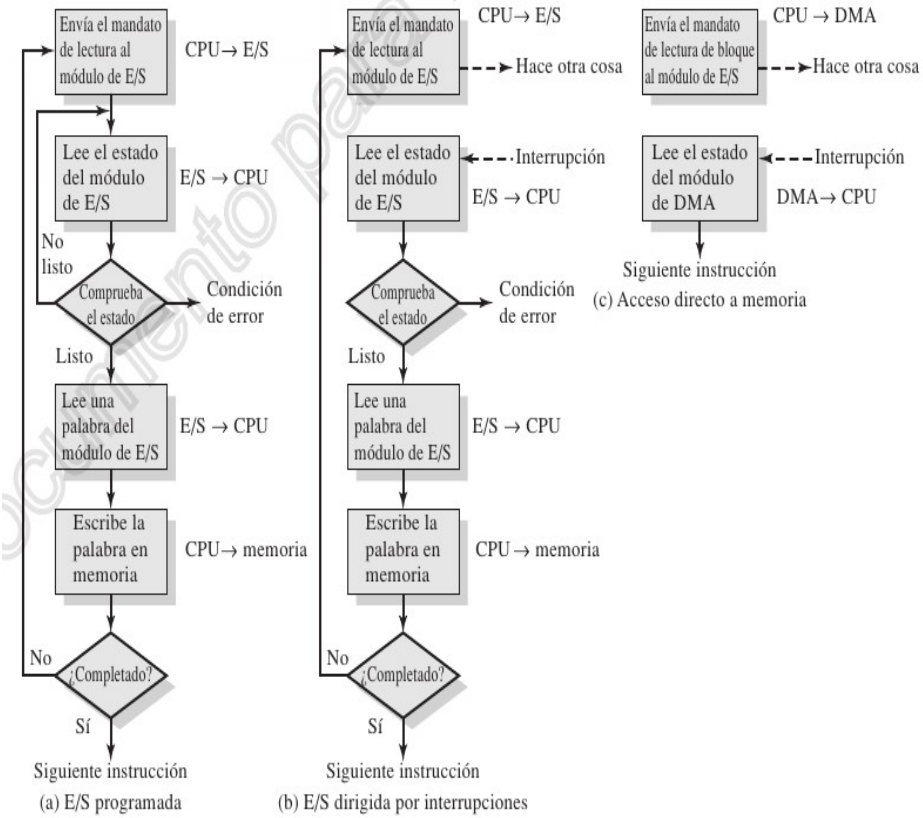


Figura 1.19. Tres técnicas para leer un bloque de datos.

El juego de instrucciones de una operación de E/S a través de un módulo de E/S incluyen las siguientes categorías:

- **Control**. Utilizadas para activar un dispositivo externo y especificarle qué debe hacer. Por ejemplo, se le puede indicar a una unidad de DVD que se rebobine o avance un registro.
- **Estado**. Utilizadas para comprobar diversas condiciones de estado asociadas a un módulo de

E/S y sus periféricos.

- **Transferencia**. Utilizadas para leer datos de un dispositivo externo o para enviar o escribor datos en el mismo.

La Figura 1.19a, se proporciona un ejemplo del uso de E/S programada para leer un bloque de datos de un dispositivo externo (p. ej. un registro de DVD) y almacenarlo en memoria. Los datos se leen palabra a palabra (por ejemplo, 16 bits). Por cada palabra que se lee, el procesador debe permanecer en un bucle de comprobación del estado hasta que determina que la palabra está disponible en el registro de datos o buffer del módulo de E/S. Este diagrama de flujo subraya las desventajas principales de esta técnica: es un proceso que consume un tiempo apreciable que mantiene al procesador ocupado innecesariamente. Ojo, aquí no se han usado interrupciones, ya que el procesador ha tenido que estar constantemente consultando al módulo de E/S para ver si había terminado o estaba lista la transferencia.

E/S dirigida por interrupciones

El problema de la E/S programada es que el procesador tiene que esperar mucho tiempo hasta que el módulo de E/S correspondiente esté listo para la recepción o la transmisión de más datos (o simplemente la comprobación de estado de un dispositivo). El procesador, mientras está esperando, debe comprobar repetidamente el estado del módulo de E/S. Como resultado, el nivel de rendimiento de todo el sistema se degrada gravemente.

Una alternativa es que el procesador genere un mandato de E/S y, acto seguido, continúe realizando algún otro trabajo útil mientras que el módulo de E/S realiza su función y se comunica con el dispositivo externo. Normalmente va a ser distinto al que esta realizando en el momento que genera el mandato de E/S, por lo que implicará un salvado de contexto del proceso actual. De nuevo cabe destacar que no debe confundir esto con una interrupción de E/S, que se produce cuando algún dispositivo quiere comunicar algo a la CPU, y no de la CPU al dispositivo.

Considere cómo funciona esta alternativa, primero desde el punto de vista del módulo de E/S. Para una operación de entrada, el módulo de E/S recibe un mandato de LECTURA del procesador. El módulo de E/S pasa entonces a leer los datos de un periférico asociado. Una vez que los datos están en el registro de datos del módulo, el módulo genera una interrupción al procesador a través de una línea de control. El módulo entonces espera hasta que el procesador pida sus datos. Cuando se hace la petición, el módulo sitúa sus datos en el bus de datos y ya está listo para otra operación de E/S.

Desde el punto de vista del procesador, las acciones correspondientes a una operación de lectura (o de escritura) son las que se describen a continuación. El procesador genera un mandato de LECTURA. Salva el contexto (por ejemplo, el contador de programa y los registros del procesador) del programa actual y lo abandona, pasando a hacer otra cosa, por ejemplo, el procesador puede estar trabajando con varios programas o procesos diferentes a la vez (aquí todavía no se ha producido interrupción). Si el planificador decide seguir con el proceso actual no se produciría el salvado de contexto comentado. Al final de cada ciclo de instrucción, el procesador comprueba si hay interrupciones (Figura 1.7). Cuando se produce la interrupción del módulo de E/S, el procesador salva el estado del programa, PSW, y el contador de programa, PC, pasando a ejecutar el manejador de interrupciones o ISR. Esto conlleva a que ISR salve el contexto del proceso que estaba ejecutando el procesador y tramita la interrupción. En este caso, el procesador se comunica con el módulo de E/S, lee la palabra de datos de dicho módulo y la almacena en memoria. A continuación, la ISR restaura el contexto del programa que había realizado el mandato de E/S (o de algún otro programa) y reanuda su ejecución.

Hasta aquí se han podido producir uno o dos cambios de contexto. Se producen dos cambios de contexto cuando la CPU en dirección dispositivo E/S decide dar trabajo al módulo de E/S y continuar haciendo otro trabajo diferente al que estaba haciendo en el momento de invocar la operación de E/S.

La Figura 1.19b muestra el uso de la E/S dirigida por interrupciones para leer un bloque de datos. La E/S dirigida por interrupciones es más eficiente que la E/S programada ya que elimina la espera innecesaria. Sin embargo, la E/S dirigida por interrupciones todavía consume mucho tiempo de procesador, puesto que cada palabra de datos que va desde la memoria al módulo de E/S o desde el módulo de E/S hasta la memoria debe pasar a través del procesador.

Casi invariablemente, habrá múltiples sub-módulos de E/S en un computador, por lo que se necesitan mecanismos para permitir que el procesador determine qué dispositivo causó la interrupción y para decidir, en caso de múltiples interrupciones, cuál debe manejar primero (la *ISR* comprueba el tipo de interrupción y decide cuál tratar primero, pudiéndose cargar subrutinas específicas para ello). En algunos sistemas, hay múltiples líneas de interrupción, de manera que cada sub-módulo de E/S usa una línea diferente. Cada línea tendrá una prioridad diferente.

Acceso directo a memoria (DMA)

La E/S dirigida por interrupciones, aunque más eficiente que la E/S programada simple, todavía requiere la intervención activa del procesador para transferir datos entre la memoria y un módulo de E/S, ya que cualquier transferencia de datos debe atravesar un camino a través del procesador. Por tanto, ambas formas de E/S sufren dos inconvenientes inherentes:

1. La tasa de transferencia de E/S está limitada por la velocidad con la que el procesador puede comprobar el estado de un dispositivo y ofrecerle servicio.

2. El procesador está involucrado en la gestión de una transferencia de E/S; se deben ejecutar varias instrucciones por cada transferencia de E/S.

Cuando se van a transferir grandes volúmenes de datos (por ejemplo grabar un CD-ROM virgen o copiar su contenido al disco duro), se requiere una técnica más eficiente: el acceso directo a memoria (Direct Memory Access, DMA). La función de DMA puede llevarla a cabo un módulo separado conectado en el bus del sistema o puede estar incluida en un módulo de E/S. Cuando el procesador desea leer o escribir un bloque de datos, genera un mandato al módulo de DMA, enviándole la siguiente información:

- Si se trata de una lectura o de una escritura.
- La dirección del dispositivo de E/S involucrado.
- La posición inicial de memoria en la que se desea leer los datos o donde se quieren escribir.
- El número de palabras que se pretende leer o escribir.

A continuación, si el procesador (más concretamente el planificador) decide continuar con otro trabajo se producirá un salvado de contexto, en caso de que decida seguir con el trabajo actual esto no se producirá. Mientras tanto, como ha delegado la operación de E/S al módulo de DMA, éste se ocupará de la misma. El módulo de DMA transferirá el bloque completo de datos, palabra a palabra, hacia la memoria o desde ella, sin pasar a través del procesador, de forma que no tendrá que hacer interrupciones constantemente cuando acaba de transferir cada bloque. Esto hace que se aproveche aún más la capacidad de procesamiento de la CPU.

Así, el procesador solo está involucrado al principio y al final de la transferencia (Fig 1.19c). Como siempre, al final de la transferencia se producirá interrupción por parte del módulo DMA, cargándose una rutina ISR y salvándose el contexto del proceso actual. Una vez se complete la comunicación entre el módulo de E/S y el procesador, se restaura el contexto del proceso que se estaba ejecutando o de algún otro en cola de espera (planificación).

Multiprogramación

Aunque no hemos hablado de ello explícitamente, al hablar de interrupciones y de entrada-salida ya sabemos cuál es el concepto de multiprogramación. En la Figura 2.4 se detalla un ejemplo de ineficiencia del procesador si no se usan interrupciones y se tiene que esperar a que se termine una operación de entrada-salida. Se representa un cálculo que corresponde a un programa que procesa un fichero con registros y realiza de media 100 instrucciones máquina por registro. En este ejemplo, el computador malgasta aproximadamente el 96% de su tiempo esperando a que los dispositivos de E/S terminen de transferir datos a y desde el fichero.

Leer un registro del fichero	15 μs
Ejecutar 100 instrucciones	1 μs
Escribir un registro al fichero	15 μs
TOTAL	31 μs

Porcentaje de utilización de la CPU = $\frac{1}{31} = 0{,}032 = 3{,}2\%$

Figura 2.4. Ejemplo de utilización del sistema.

La Figura 2.5a muestra esta situación, donde existe un único programa, lo que se denomina **monoprogramación**. El procesador ejecuta durante cierto tiempo hasta que alcanza una instrucción de E/S. Entonces debe esperar que la instrucción de E/S concluya antes de continuar.

Esta ineficiencia puede evitarse con **multiprogramación**. Supóngase que existe suficiente memoria para contener al sistema operativo y un programa de usuario. Cuando un trabajo necesita esperar por una operación de E/S, se puede asignar el procesador al otro trabajo, que probablemente no esté esperando por una operación de E/S (Figura 2.5b). Más aún, se puede expandir la memoria para que albergue tres, cuatro o más programas y pueda haber multiplexación entre todos ellos (Figura 2.5c).

Cuando el procesador trata con varios programas residentes en memoria, la secuencia en la que se ejecutan los programas dependerá de la prioridad asignada a cada uno de ellos, así como de si están esperando la finalización de una operación de E/S. Esto se conoce como **Planificación**, ya comentada en la sección “Planificación y gestión de los recursos”.

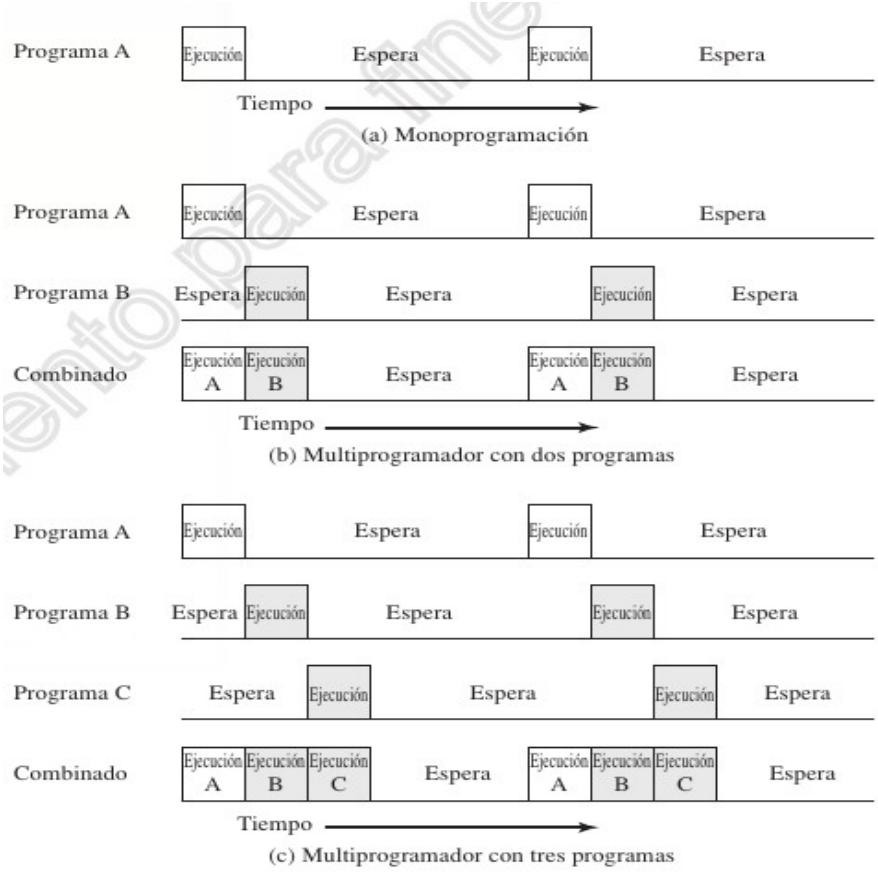


Figura 2.5. Ejemplo de multiprogramación.

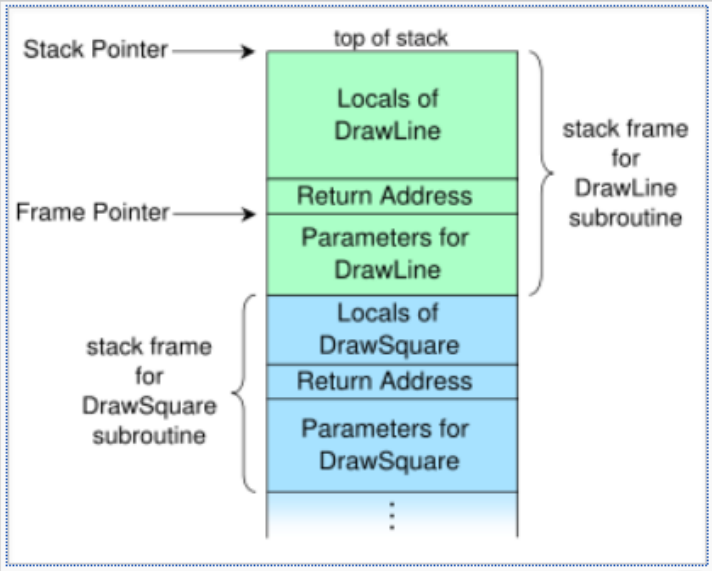
Control de procedimientos (PILA)

Una técnica habitual para controlar la ejecución de llamadas a procedimiento y los retornos de los mismos es utilizar una pila. Esta sección resume las propiedades básicas de las pilas y revisa su uso para el control de procedimientos.

Una pila es un conjunto ordenado de elementos, tal que en cada momento solamente se puede acceder a uno de ellos (el más recientemente añadido). El punto de acceso se denomina cima de la pila. El número de elementos de la pila, o longitud de la pila, es variable. Por esta razón, se conoce también a una pila como una lista de apilamiento o lista donde el último que entra es el primero que sale (Last-In First-Out, LIFO).

La implementación de una pila requiere que haya un conjunto de posiciones dedicado a almacenar los elementos de la pila. En la Figura de más abajo se muestra una técnica habitual. Se reserva en memoria principal un bloque contiguo de posiciones. La mayoría de las veces el bloque está parcialmente lleno con elementos de la pila y el resto está disponible para el crecimiento de la pila. Se necesitan tres direcciones para un funcionamiento adecuado, que habitualmente se almacenan en registros del procesador y en el BCP del proceso:

- Puntero de pila. Contiene la dirección de la cima de la pila. Si se añade un elemento (APILA) o se elimina (EXTRAE), el puntero se decrementa o se incrementa para contener la dirección de la nueva cima de la pila.
- Base de la pila. Contiene la dirección de la posición inferior en el bloque reservado. Se trata de la primera posición que se utiliza cuando se añade un elemento a una pila vacía. Si se hace un intento de extraer un elemento cuando la pila está vacía, se informa del error.
- Límite de la pila. Contiene la dirección del otro extremo, o cima, del bloque reservado. Si se hace un intento para apilar un elemento cuando la pila está llena, se indica el error.



Estructura de la pila de llamadas.
En la figura se ve una pila, creciendo de abajo hacia arriba.
La subrutina `DrawSquare` es llamada y se crea un *stack frame* para ella (en azul). Luego, `DrawSquare` llama a la subrutina `DrawLine`, la cual tiene su propio *stack frame* (en verde).
El *stack frame* de cada subrutina tiene, en este caso, tres partes: * una dirección de retorno que indica la siguiente dirección a ejecutar después de que termine la subrutina, * los parámetros con que fue llamada la subrutina (que se cargan antes de llamarla), * y un espacio reservado para las variables y las constantes locales de la subrutina.

Una técnica habitual para gestionar las llamadas y los retornos de los procedimientos es utilizar una pila. Cuando el procesador ejecuta una llamada, se almacena (apila) la dirección de retorno en la pila del proceso llamador (siguiente instrucción a ejecutar en el regreso). Cuando se ejecuta un retorno, se utiliza la dirección de la cima de la pila y se elimina (extrae) esa dirección de la pila. La Figura 1.27 muestra el uso de la pila para los procedimientos anidados presentados en la Figura 1.26. Es también necesario con frecuencia pasar parámetros en una llamada a procedimiento. Una posibilidad es almacenar los parámetros en la memoria justo después de las instrucciones de llamada. En este caso, el retorno debe estar en la posición siguiente a los parámetros. Cuando el procesador ejecuta una llamada, no sólo apila la dirección de retorno, sino también los parámetros que se desean pasar al procedimiento llamado. El procedimiento invocado puede acceder a los parámetros en la pila. Al retornar, los parámetros de retorno se pueden almacenar también en la pila, debajo de la dirección de retorno. El conjunto completo de parámetros, incluyendo la dirección de retorno, que se almacena en una invocación de procedimiento se denomina marco de pila.

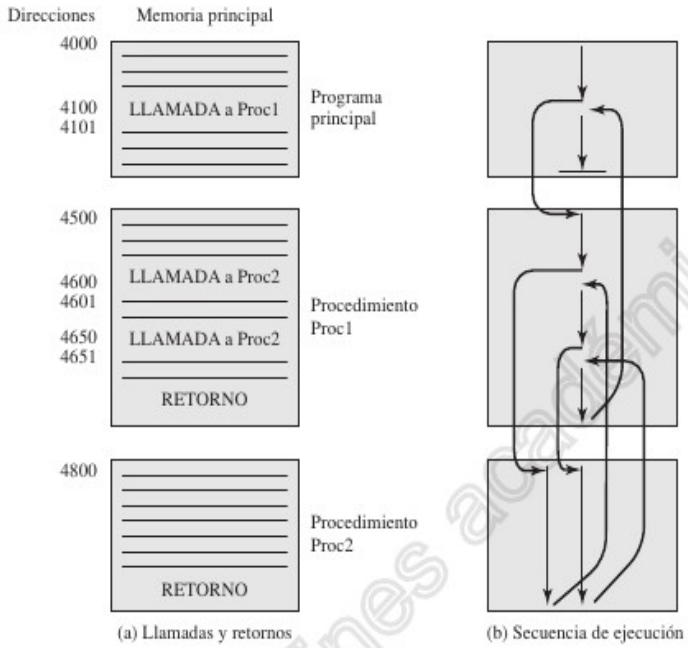


Figura 1.26. Procedimientos anidados.

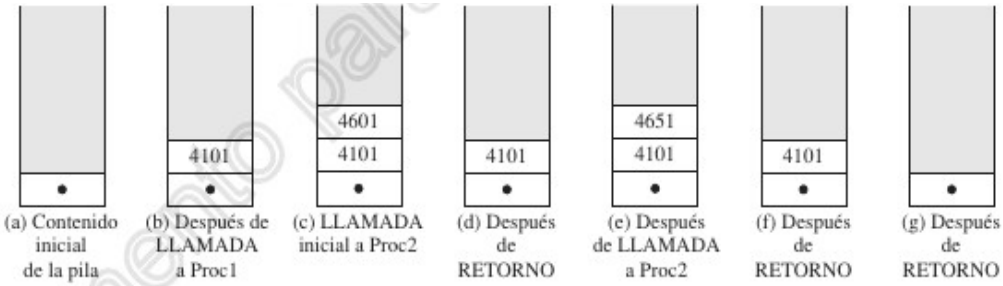


Figura 1.27. Uso de la pila para implementar los procedimientos anidados de la Figura 1.26.

Desarrollos que han llevado a los sistemas operativos modernos

A lo largo de los años, ha habido una evolución gradual de la estructura y las capacidades de los sistemas operativos. Sin embargo, en los últimos años se han introducido un gran número de nuevos elementos de diseño en sistemas operativos. Estos cambios responden a necesidades y gestión de:

- Nuevos desarrollos en hardware.
- Nuevas aplicaciones.
- Nuevas amenazas de seguridad.
- Incremento en las velocidades de cómputo (gestión de multiprocesadores).
- Nuevos dispositivos de conexión de alta velocidad a la red.
- Nuevos y variados dispositivos de almacenamiento.
- Accesos a Internet.
- Computación cliente-servidor.
- Nuevas amenazas de seguridad en el acceso a Internet por ataques sofisticados, tales como virus, gusanos, y técnicas de hacking, lo que ha supuesto un impacto profundo en el diseño de los sistemas operativos. Por ejemplo, el bit NX, que es una característica física del