TEMA 1 – INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS

Bibliografía

El contenido de este documento se ha elaborado, principalmente, a partir de las siguientes referencias bibliográficas, y con propósito meramente académico:

[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.
[STALLINGS] W. Stallings. Sistemas operativos. 5ª edicion, Prentice Hall, Madrid, 2005.
[TANENBAUM] A. S. Tanenbaum. Sistemas operativos modernos, 3a edicion. Prentice Hall, Madrid, 2009.
[CARRETERO] F. Pérez, J. Carretero, F. García. Problemas de sistemas operativos: de la base al diseño, 2ª edición. McGraw-Hill. 2003.

Contenido

- Introducción (Introducción, Sección 1-1 [Aranda]) ¿Qué es un sistema operativo?
- Planificación y gestión de los recursos (Capitulo 2, Sección 2.3 [Stallings]).
- Máquina virtual y multinivel. (Capítulo 1 [Tanenbaum], Capítulo 1 [Carretero]).
- Elementos básicos o componentes del computador (Capitulo 1, sección 1.1 [Stallings]).
- Registros del procesador (Capitulo 1, sección 1.2 [Stallings]).
- Ejecución de instrucciones (Capitulo 1, sección 1.3 [Stallings]).
- Interrupciones (Capítulo 1, sección 1.4 [Stallings]).
- Sistema de Entrada/Salida (Capítulo 1, sección 1.7 [Stallings]).
- Técnicas de comunicación de E/S (Capítulo 1, sección 1.7 [Stallings]).
- Control de procedimientos (Pila) (Capítulo 1, Apéndice 1B [Stallings].
- Multiprogramación (Capítulo 2, sección 2.2 [Stallings]).
- Desarrollos que han llevado a los sistemas operativos modernos (Capítulo 2, sección 2.4 [Stallings].
- Un poco de historia (Capítulo 2, sección 2.6 y 2.8 [Stallings]).

Tema 1 Introducción

Un computador sin el "software" es una máquina sin utilidad, necesita de programas que le permitan gestionar bases de datos, realizar complicados cálculos matemáticos, ayudar al diseño, procesar imágenes, comunicarse con computadores situados en cualquier parte del mundo y a todas las tareas que estamos acostumbrados que realicen. Pero un computador es un sistema complejo que se compone de procesadores, memoria central, discos, terminales, conexiones de red, dispositivos de *E/S*, etc. La gestión de todos estos elementos y su utilización correcta es una labor ardua y en extremo difícil. Si esta gestión la tienen que hacer los propios programas de aplicación, que además pueden estar ejecutándose simultáneamente, es muy probable que los programadores se vieran desbordados por la dificultad.

No obstante, hace tiempo que se vio la necesidad de distinguir dos tipos de programas:

a) Los programas del sistema

b) Los programas de aplicación

En un sentido amplio los programas del sistema se encargan de controlar las operaciones propias del computador, mientras que los programas de aplicación son los que resuelven problemas específicos a los usuarios. De los programas del sistema el más importante es el *sistema operativo* cuyo objetivo es que el computador se pueda utilizar de una manera cómoda y eficiente, proporcionando un caparazón a la máquina desnuda que permite dar la visión de una *máquina virtual,* con la que es factible comunicarse al crearse un interfaz entre el usuario y la máquina y gestionar los recursos de la misma.

La Figura 1.1 muestra un diagrama con esta situación. En la parte inferior se encuentra el hardware, en el que está incluido el procesador, la memoria y los demás periféricos, como la consola, los discos, etc, además, si el procesador está implementado mediante lógica microprogramada también hay que considerar la microprogramación (firmware) y el lenguaje máquina (ver Dormido et al., 2000).

El sistema operativo oculta toda esta complejidad del hardware y proporciona un interfaz más adecuado para usar el sistema. Actúa como un mediador, de forma que los programadores y los programas de aplicación puedan acceder y utilizar los recursos y servicios del sistema de una forma más fácil, sin preocuparse de la gestión de los mismos.

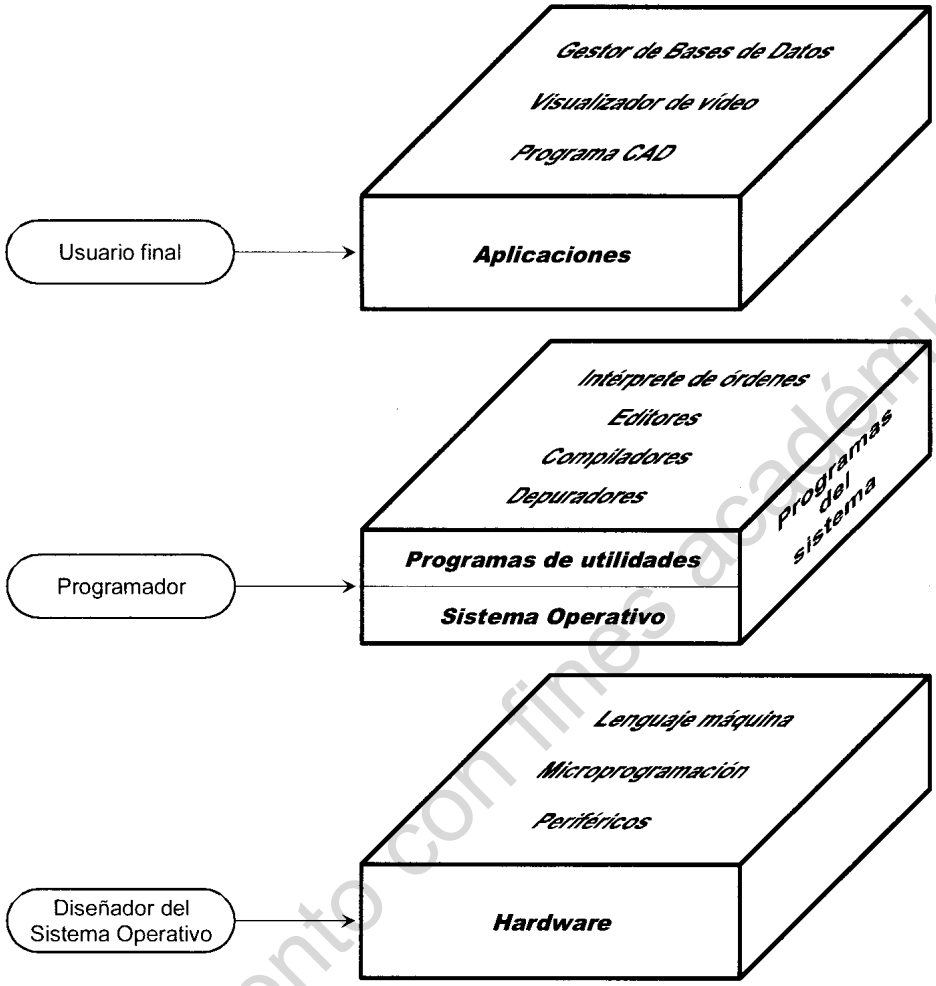


Figura 1.1: Capas en las que se pueden dividir los componentes de un computador

Los programadores emplean otros programas del sistema, como compiladores, editores o depuradores. Las aplicaciones que estos desarrollan también usan los programas del sistema que efectúan llamadas al sistema operativo, a las funciones de *E/S* y al sistema de archivos.

En el extremo superior de la Figura 1.1, por encima de los programas del sistema, se encuentran las aplicaciones. El usuario final, que no es consciente de la arquitectura del sistema, verá al computador a través de estas aplicaciones desarrolladas por el programador de aplicaciones.

Sección 1-1

¿Qué es un sistema operativo?

En términos generales no hay una definición de sistema operativo completamente adecuada y que sea aceptada universalmente. En los años sesenta se decía que era *el software que controlaba al hardware*, sin embargo esta definición ha quedado totalmente desfasada en la actualidad. Ahora quizás sea más fácil decir que hacen los sistemas operativos que decir que son. Por ello, más que dar una definición diciendo que es un sistema operativo, lo que se va a realizar es exponer cuales tienen que ser sus funciones u objetivos.

En la Figura 1.1 se ha situado al sistema operativo como una capa por encima del hardware, de forma que lo oculta al resto de los programas y a los usuarios. Desde esta perspectiva el sistema operativo no es más que un programa que controla la ejecución de los programas de aplicaciones y actúa como un interfaz entre los usuarios del computador y el hardware del mismo. Sus objetivos básicos son:

1) *Comodidad para los usuarios*. El SO hace que el computador sea más fácil de utilizar. En este caso hace que los usuarios vean una máquina virtual o extendida, que es más sencilla de programar y de utilizar que la máquina desnuda.

2) *Eficiencia*. El SO gestiona los recursos del sistema de forma más eficaz. Su función es en este caso la de un gestor de recursos.

A continuación se analizan cada uno de estos objetivos.

1.1.1 Los sistemas operativos como máquinas virtuales

La utilización directa del hardware es difícil, especialmente para la realización de las operaciones de *E/S*. Por ejemplo, para operar directamente con el controlador de un disquete hay que manejar del orden de 16 órdenes, y donde cada una puede tener más de una docena de parámetros, que hay que empaquetar en un registro del dispositivo de 8 bytes, devolviéndose después de la operación un registro de 7 bytes con los estados y los campos de errores. Para hacer una escritura en el disquete hay que verificar si el motor esta funcionando, si no lo está dar la orden de arranque, esperar a que gire a la velocidad adecuada, entonces dar las órdenes de posicionamiento del brazo en el sector y pista adecuada, ordenar la escritura, etc. El programador, y usuario en general, no desea enfrentarse a toda esta problemática, sino que desea una abstracción sencilla y fácil de entender. Por ejemplo, ver al disco como un conjunto de archivos, de forma que cuando se quiere escribir algo sólo hay que indicar el nombre del archivo.

Para ocultar toda esta problemática del hardware está el sistema operativo, de forma que el programador y el usuario ven una abstracción del mismo que se les presenta como una máquina virtual que entiende órdenes a un nivel superior. En este sentido, el sistema operativo tiene que proporcionar servicios para las funciones siguientes:

1) *Creación de programas*. Existen otros programas del sistema, como son los depuradores, los editores y los enlazadores, que no son parte del sistema operativo, pero que son accesibles a través de él.

2) *Ejecución de programas*. Para poder ejecutar un programa se tiene que realizar una serie de funciones previas, tales como cargar el código y los datos en la memoria principal, inicializar los dispositivos de *E/S* y preparar los recursos necesarios para la ejecución. Todo esto lo gestiona el sistema operativo.

3) *Operaciones de entrada/salida*. Un programa puede requerir una operación de *E/S* sobre un periférico. Pero cada uno tiene sus peculiaridades y un controlador específico con su conjunto de instrucciones. Como en el ejemplo del controlador de la disquetera anterior, es el propio sistema operativo el encargado de hacer todas esas funciones que permiten la lectura, escritura y comunicación con los periféricos.

4) *Manipulación y control del sistema de archivos*. Además de comunicarse con el controlador del periférico en donde está el sistema de archivos, el sistema operativo debe conocer la propia estructura (formato) de almacenamiento y proporcionar los mecanismos adecuados para su control y protección.

5) *Detección de errores*. Hay una gran cantidad de errores, tanto del hardware como del software, que pueden ocurrir. Por ejemplo: un mal funcionamiento de un periférico, fallos en la transmisión de los datos, errores de cálculo en un programa, divisiones por cero, rebose, fallos de la memoria, violaciones de permisos, etc. El sistema operativo debe ser capaz de detectarlos y solucionarlos o por lo menos hacer que tengan el menor impacto posible sobre el resto de las aplicaciones.

6) *Control del acceso al sistema*. En sistemas de acceso compartido o en sistemas públicos, el sistema operativo debe controlar el acceso al mismo, vigilando quién tiene acceso y a qué recursos. Por este motivo tiene que tener mecanismos de protección de los recursos e implementar una adecuada política de seguridad, de forma que no pueda acceder quién no esté autorizado. Debido a la gran conectividad que tienen hoy día los sistemas informáticos, este es un aspecto que cada vez está teniendo mayor interés.

7) *Elaboración de informes estadísticos*. Resulta muy conveniente conocer el grado de la utilización de los recursos y de los distintos parámetros del sistema, como el tiempo de respuesta. De esta forma se dispone de información que permite saber con antelación las necesidades futuras y configurar al sistema para dar el mejor rendimiento.

1.1.2 El sistema operativo como gestor de recursos

La visión anterior de un sistema operativo es una visión de abajo hacia arriba. Un punto de vista alternativo, considera al SO como el gestor de todos los elementos que componen el sistema. En este caso su función es proporcionar a los programas que compiten por ellos una asignación ordenada y controlada de los procesadores, la memoria y los periféricos.

El sistema operativo realiza este control de una forma muy peculiar, si se compara con los mecanismos que utilizan otros sistemas. Por ejemplo, el control de temperatura de un edificio se hace mediante un termostato que está totalmente diferenciado del sistema de distribución de calor (los radiadores) y del mecanismo de generación de calor (la caldera). Esta separación no existe en el caso del sistema operativo,

que funciona de la misma forma que otro programa (al cual tiene que controlar y dar servicio); es decir, el sistema operativo se ejecuta en el procesador lo mismo que el resto de los programas. Otra característica que lo hace peculiar es que con cierta frecuencia pierde el control sobre el procesador y debe esperar a disponer de él.

Desde este punto de vista, el sistema operativo no es más que otro programa del computador que el procesador no distingue del resto. La gran diferencia está en la finalidad, ya que la CPU dedicará parte de su tiempo a ejecutar otros programas según lo planifique el sistema operativo y también controla el acceso a los otros recursos del sistema. Sin embargo, esto requiere que el propio sistema operativo ceda el control del procesador a otra "tarea" y cuando posteriormente lo recupera otra vez, debe preparar la siguiente tarea que va a ejecutarse en el procesador.

Planificación y gestión de los recursos

Una responsabilidad clave de los sistemas operativos es la gestión de varios recursos disponibles para ellos (espacio de memoria principal, dispositivos de E/S, procesadores) y para planificar su uso por parte de los distintos procesos activos. Cualquier asignación de recursos y política de planificación debe tener en cuenta tres factores:

- **Equitatividad**. Normalmente, se desea que todos los procesos que compiten por un determinado recurso, se les conceda un acceso equitativo a dicho recurso. Esto es especialmente cierto para trabajos de la misma categoría, es decir, trabajos con demandas similares.
- **Respuesta diferencial**. Por otro lado, el sistema operativo puede necesitar discriminar entre diferentes clases de trabajos con diferentes requisitos de servicio. El sistema operativo debe tomar las decisiones de asignación y planificación con el objetivo de satisfacer el conjunto total de requisitos. Además, debe tomar las decisiones de forma dinámica. Por ejemplo, si un proceso está esperando por el uso de un dispositivo de E/S, el sistema operativo puede intentar planificar este proceso para su ejecución tan pronto como sea posible a fin de liberar el dispositivo para posteriores demandas de otros procesos.
- **Eficiencia**. El sistema operativo debe intentar maximizar la productividad, minimizar el tiempo de respuesta, y, en caso de sistemas de tiempo compartido, acomodar tantos usuarios como sea posible. Estos criterios entran en conflicto; encontrar un compromiso adecuado en una situación particular es un problema objeto de la investigación sobre sistemas operativos.

La planificación y la gestión de recursos son esencialmente problemas de investigación, y se pueden aplicar los resultados matemáticos de esta disciplina. Adicionalmente, medir la actividad del sistema es importante para ser capaz de monitorizar el rendimiento y realizar los ajustes correspondientes.

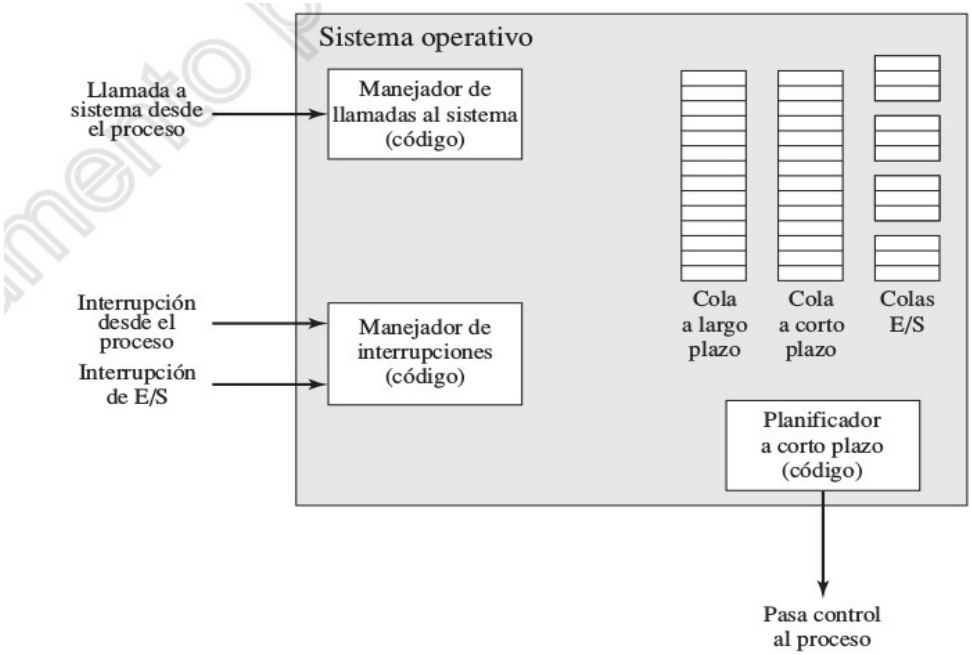


Figura 2.11. Elementos clave de un sistema operativo para la multiprogramación.

La Figura 2.11 sugiere los principales elementos del sistema operativo relacionados con la planificación de procesos y la asignación de recursos en un entorno de multiprogramación. El sistema operativo mantiene un número de colas, cada una de las cuales es simplemente una lista de procesos esperando por algunos recursos. La cola a corto plazo está compuesta por procesos que se

encuentran en memoria principal (o al menos una porción mínima esencial de cada uno de ellos está en memoria principal) y están listos para ejecutar, siempre que el procesador esté disponible. Cualquiera de estos procesos podría usar el procesador a continuación. Es responsabilidad del planificador a corto plazo, o **dispatcher**, elegir uno de ellos. Una estrategia común es asignar en orden a cada proceso de la cola un intervalo de tiempo; esta técnica se conoce como round-robin o turno rotatorio. En efecto, la técnica de turno rotatorio emplea una cola circular. Otra estrategia consiste en asignar niveles de prioridad a los distintos procesos, siendo el planificador el encargado de elegir los procesos en orden de prioridad.

La cola a largo plazo es una lista de nuevos trabajos esperando a utilizar el procesador. El sistema operativo añade trabajos al sistema transfiriendo un proceso desde la cola a largo plazo hasta la cola a corto plazo. En este punto, se debe asignar una porción de memoria principal al proceso entrante. Portanto, el sistema operativo debe estar seguro de que no sobrecarga la memoria o el tiempo de procesador admitiendo demasiados procesos en el sistema. Hay una cola de E/S por cada dispositivo de E/S. Más de un proceso puede solicitar el uso del mismo dispositivo de E/S. Todos los procesos que esperan utilizar dicho dispositivo, se encuentran alineados en la cola del dispositivo. De nuevo, el sistema operativo debe determinar a qué proceso le asigna un dispositivo de E/S disponible.

Si ocurre una **interrupción** (se estudiará a continuación), el sistema operativo recibe el control del procesador a través de un manejador de interrupciones. Un proceso puede invocar específicamente un servicio del sistema operativo, tal como un manejador de un dispositivo de E/S, mediante una llamada a sistema (se estudiará posteriormente). En este caso, un manejador de la llamada a sistema (**trap**) es el punto de entrada al sistema operativo. En cualquier caso, una vez que se maneja la interrupción o la llamada a sistema, se invoca al planificador a corto plazo para que seleccione un proceso para su ejecución.

Lo que se ha detallado hasta ahora es una descripción funcional; los detalles y el diseño modular de esta porción del sistema operativo difiere en los diferentes sistemas. Gran parte del esfuerzo en la investigación y desarrollo de los sistemas operativos ha sido dirigido a la creación de algoritmos de planificación y estructuras de datos que proporcionen equitatividad, respuesta diferencial y eficiencia.

Máquina multinivel

El sistema operativo separa al programador de trabajar con el hardware a bajo nivel (bits, transistores, puertas lógicas, etc) y presenta una interfaz sencilla ocultando asuntos desagradables como interrupciones, temporizadores, administración de memoria, etc. Desde esta perspectiva, una de las funciones del sistema operativo es presentar al usuario el equivalente de una máquina virtual que es más fácil de programa que el hardware subyacente. Por tanto, una máquina virtual es aquella que basada en una máquina más elemental presenta una mayor facilidad de uso incluyendo toda su funcionalidad. Las máquinas virtuales pueden estar dispuestas en varias capas, como se verá a continuación.

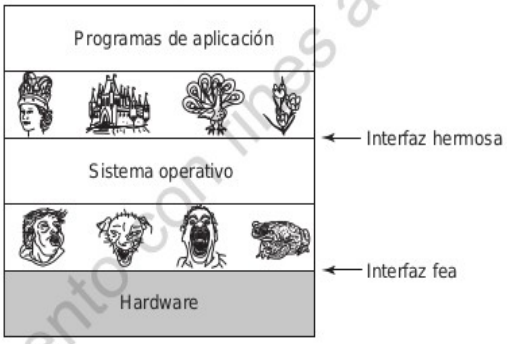
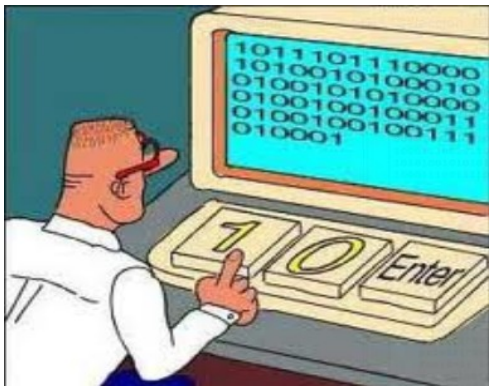


Figura 1-2. Los sistemas operativos ocultan el hardware feo con abstracciones hermosas.

Las máquinas interpretan más fácilmente las señales *on* y *off*, lo que equivale a interpretar la presencia o la ausencia de voltaje. Por lo tanto, el alfabeto de las máquinas está constituido por dos símbolos que son representados por el 1 y el 0 respectivamente. A cada una de estos símbolos se le conoce como dígito binario o bit (binary digit), y sobre este alfabeto se construyen los comandos o instrucciones con los cuales nos comunicamos. Las instrucciones que ejecutan los computadores son colecciones de bits que pueden ser vistos como números. Por ejemplo, el siguiente patrón de bits, le indica al computador que debe sumar dos números: 1000110010100000. El problema de esto es que escribir un programa a base de bits es algo tedioso y complicado para los programadores.



Surge entonces lo que llamamos **máquina multinivel**, que es una estructuración en capas bajo una serie de abstracciones, donde cada capa se apoya en la que está debajo de ella, y facilita el trabajo con el sistema operativo. A continuación se muestra una representación de una máquina multinivel.

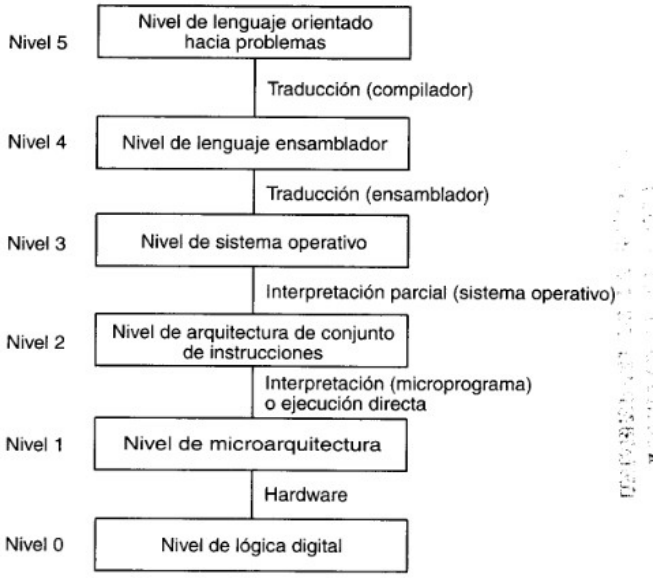
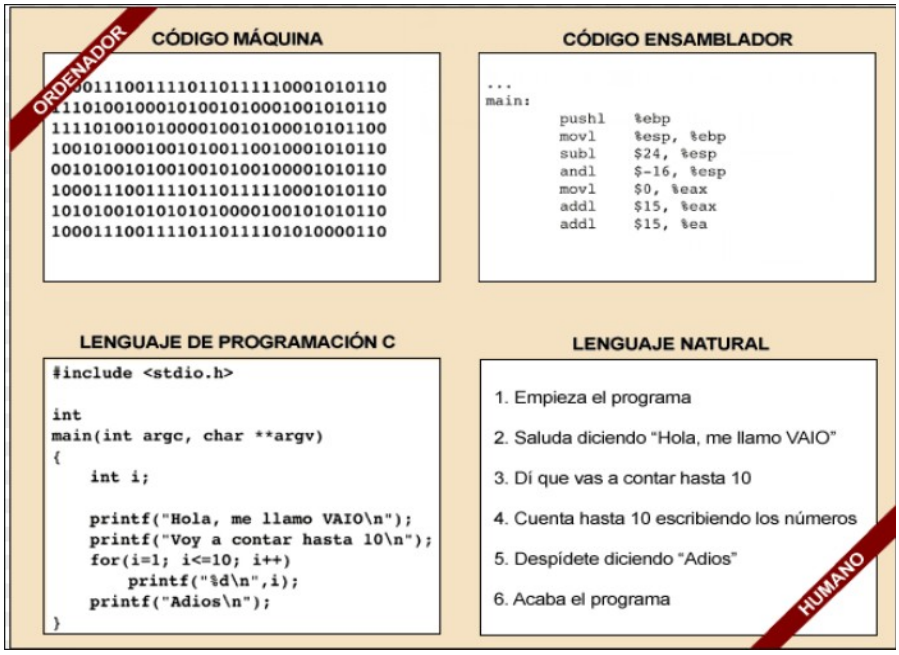


Figura 1-2. Computadora de seis niveles. El método de apoyo para cada nivel se indica inmediatamente abajo de él (junto con el nombre del programa de apoyo).

Lenguaje de alto nivel

Con el objetivo de expresar los programas en un lenguaje más cercano a la forma de pensamiento de los programadores surgen los lenguajes de alto nivel (nivel 5). Para que los computadores puedan procesar estos nuevos tipos de lenguajes se desarrollaron otros programas que traducen

programas escritos en un lenguaje de alto nivel a un lenguaje de más bajo nivel llamado ensamblador (nivel 4). Estamos hablando de los compiladores. En la figura se muestra un ejemplo de un programa escrito en un lenguaje de alto nivel como C, que es traducido a un programa en lenguaje ensamblador por medio del compilador.



Ensamblador

El lenguaje ensamblador todavía no es un lenguaje entendible por la máquina. Ensamblador es un tipo de programa informático que se encarga de traducir un fichero fuente escrito en lenguaje ensamblador (nivel 4) mediante instrucciones mnemónicas a un fichero objeto que contiene código máquina (niveles 3 y 2). El lenguaje máquina está formado por un conjunto de instrucciones que determinan una serie de acciones que debe realizar la máquina. Por ejemplo, el programador escribe "add A, B" (mnemónico) y el ensamblador traduce esta instrucción de más alto nivel en "1000110010100000" (lenguaje de máquina). Según la arquitectura del microprocesador y el número de registros de éste, los mnemónicos y la traducción a lenguaje máquina difieren de unas a otras (los fabricantes de microprocesadores publican el repertorio de instrucciones que sus máquinas pueden ejecutar). En los niveles 3 y 2 existen las mismas instrucciones máquina, la diferencia esta en que en el nivel 3 el sistema operativo debe hacer determinadas reorganizaciones de las instrucciones a nivel de memoria, posible ejecución en paralelo de programas, etc.

En el lenguaje ensamblador para un procesador x86:

La sentencia

- `MOV AL, 61h`

Asigna el valor hexadecimal `61` (97 decimal) al registro "`AL`".

El programa ensamblador lee la sentencia de arriba y produce su equivalente binario en lenguaje de máquina

- Binario: `10110000 01100001` (hexadecimal: `B61`)

El código de máquina generado por el ensamblador consiste de 2 bytes. El primer byte contiene

empaquetado la instrucción MOV y el código del registro hacia donde se va a mover el dato:

```
1011  0000  01100001
   |    |        |
   |    |        +---- Número 61h en binario
   |    |
   |    +--- Registro AL
   +-------- Instrucción MOV
```

En el segundo byte se especifica el número 61h, escrito en binario como `01100001`, que se asignará al registro `AL`, quedando la sentencia ejecutable como:

- `10110000 01100001`

En las máquinas en las que no existe el nivel de microprogramación (se explica a continuación), las instrucciones del nivel de máquina son realizadas directamente por los circuitos electrónicos (el hardware, el nivel 0), es decir, se pasa del nivel 2 al nivel 0, sin pasar por el 1.

Cabe decir aquí, que el funcionamiento del sistema operativo, es decir, el propio código del mismo, las funciones que lo rigen, el cómo trabaja, está codificado a nivel de instrucciones, es decir, en código máquina. El código máquina del sistema operativo se carga al arrancar nuestra computadora para así poder proporcionar servicios y gestionar recursos para el usuario. A este código codificado en instrucciones máquina no se puede acceder por parte de un usuario normal, el cual solo puede hacer cambios en el nivel 5 o como mucho en el nivel 4, siendo el diseñador del sistema operativo el que se encarga de hacer modificaciones a más bajo nivel. Esto da lugar a lo que se llama modo usuario y modo núcleo, de lo cual hablaremos posteriormente.

Microprogramación (no confundir con multiprogramación, que se estudiará más adelante)

El diseño de microprocesadores de propósito general conoce dos técnicas que conducen a una clasificación de éstos en dos grupos:

- Los microprocesadores "cableados": aquellos que tienen una unidad de control específicamente diseñada sobre el silicio para un juego de instrucciones concreto. En este caso el hardware se encarga directamente de ejecutar las microinstrucciones del nivel 2.
- Los microprocesadores "microprogramados": aquellos que tienen una unidad de control genérica o prediseñada y que implementan un juego de instrucciones u otro dependiendo de un microprograma software (paso de nivel 2 a nivel 1).

Hoy día la microprogramación ha desaparecido prácticamente por completo. Esto se debe a los siguientes factores:

- Ya existen herramientas avanzadas para diseñar complejas unidades de control con millones de transistores litografiados. Estas herramientas prácticamente garantizan la ausencia de errores de diseño.
- Las unidades de control cableadas tienen un rendimiento significativamente mayor que cualquier unidad microprogramada, resultando más competitivas.

En el caso de que un sistema necesite ser microprogramado, existe otro paso intermedio (del nivel 2 al nivel 1) antes de llegar al hardware (nivel 0). El lenguaje máquina no es el que se pondría directamente sobre la lógica digital (circuitos integrados, transistores, puertas lógicas, etc) o nivel 0, sino que se interpretaría del nivel 2 al nivel 1 por un programa llamado microprograma. El microprograma transforma el lenguaje máquina en microinstrucciones con un determinado formato de unos y ceros (una instrucción en lenguaje máquina se “parte” en tareas más sencillas que dar lugar a varias microinstrucciones). Estas microinstrucciones ya si se pueden ejecutar en el hardware interpretadas por el microprograma y siguiendo un determinado flujo o trayectoria de datos hasta la

unidad aritmético-lógica (ALU) del procesador. Por tanto, la microprogramación constituye un intérprete entre el lenguaje maquina del nivel 2 y el hardware del microprocesador (nivel 0) que dice qué camino deben seguir las microinstrucciones en el procesador y sus registros. En las máquinas que no están microprogramadas, esa trayectoria de datos la controla directamente el hardware. El cómo se hace de manera concreta es algo que queda fuera de está asignatura y se enfoca más a temas de arquitectura de computadores.

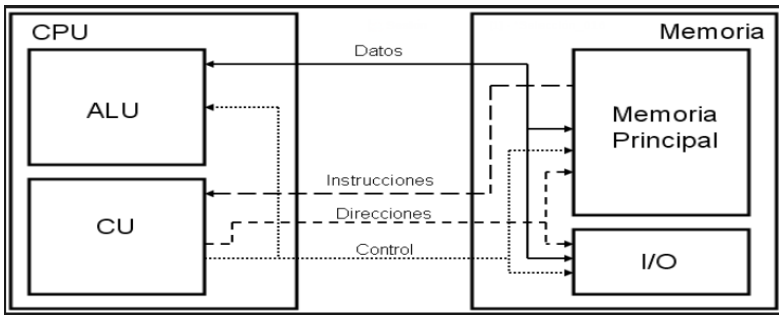
Elementos básicos o componentes del procesador

Al más alto nivel (arquitectura de Von Neumann), un computador consta del procesador, la memoria y los componentes de E/S, incluyendo uno o más módulos de cada tipo. Estos componentes se interconectan de manera que se pueda lograr la función principal del computador, que es ejecutar programas. Por tanto, hay cuatro elementos estructurales principales:

- **Procesador**. Controla el funcionamiento del computador y realiza sus funciones de procesamiento de datos. Se denomina usualmente unidad central de proceso (Central Processing Unit, CPU).
- **Memoria principal**. Almacena datos y programas. Esta memoria es habitualmente volátil; es decir, cuando se apaga el computador, se pierde su contenido. En contraste, el contenido de la memoria del disco se mantiene incluso cuando se apaga el computador. A la memoria principal se le denomina también memoria real o memoria primaria.
- **Módulos de E/S**. Transfieren los datos entre el computador y su entorno externo. El entorno externo está formado por diversos dispositivos, incluyendo dispositivos de memoria secundaria (por ejemplo, discos), equipos de comunicaciones y terminales.
- **Bus del sistema**. Proporciona comunicación entre los procesadores, la memoria principal y los módulos de E/S.

La Figura 1.1 muestra estos componentes de más alto nivel. Una de las funciones del procesador es el intercambio de datos con la memoria. Para este fin, se utilizan normalmente dos registros internos (al procesador): un registro de dirección de memoria (RDIM), que especifica la dirección de memoria de la siguiente lectura o escritura; y un registro de datos de memoria (RDAM), que contiene los datos que se van a escribir en la memoria o que recibe los datos leídos de la memoria. De manera similar, un registro de dirección de E/S (RDIE/S) especifica un determinado dispositivo de E/S, y un registro de datos de E/S (RDAE/S) permite el intercambio de datos entre un módulo de E/S y el procesador.

Un módulo de memoria consta de un conjunto de posiciones definidas mediante direcciones numeradas secuencialmente. Cada posición contiene un patrón de bits que se puede interpretar como una instrucción o como datos. Un módulo de E/S transfiere datos desde los dispositivos externos hacia el procesador y la memoria, y viceversa. Contiene buffers (es decir, zonas de almacenamiento internas) que mantienen temporalmente los datos hasta que se puedan enviar.



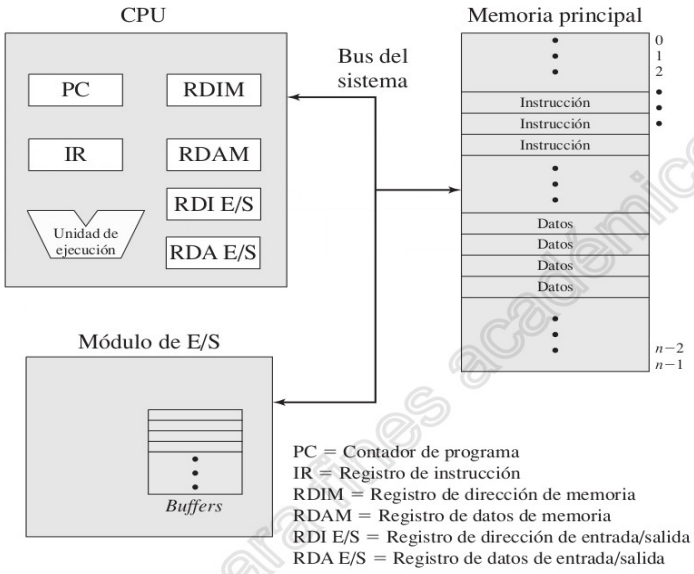


Figura 1.1. Componentes de un computador: visión al más alto nivel.

Registros del procesador

Un procesador incluye un conjunto de registros que proporcionan un tipo de memoria que es más rápida y de menor capacidad que la memoria principal. A continuación hablaremos de ellos muy brevemente, tratándose en profundidad en asignaturas de arquitectura de computadores:

- **Registros visibles para el usuario.** Permiten al programador en lenguaje máquina o en ensamblador minimizar las referencias a memoria principal optimizando el uso de registros.

  Para lenguajes de alto nivel, un compilador que realice optimización intentará tomar decisiones inteligentes sobre qué variables se asignan a registros y cuáles a posiciones de memoria principal. Algunos lenguajes de alto nivel, tales como C, permiten al programador sugerir al compilador qué variables deberían almacenarse en registros. Dependiendo de la arquitectura del procesador, estos registros pueden cambiar. Los tipos de registros que están normalmente disponibles son registros de datos (por ejemplo, una operación de suma) u de dirección (contienen direcciones de memoria principal de datos e instrucciones) .

- **Registros de control y estado (no visibles para el usuario)**. Usados por el procesador para controlar su operación y por rutinas privilegiadas del sistema operativo para controlar la ejecución de programas.

  Se emplean varios registros del procesador para controlar el funcionamiento del mismo. A algunos de ellos se puede acceder mediante instrucciones de máquina ejecutadas en lo que se denomina modo de control o de sistema operativo (modo kernel o núcleo). Por supuesto,

diferentes máquinas tendrán distintas organizaciones de registros y utilizarán diferente terminología. A continuación, se proporcionará una lista razonablemente completa de tipos de registros, con una breve descripción de cada uno de ellos. Además de los registros RDIRM, RDAM, RDIE/S y RDAE/S mencionados anteriormente (Figura 1.1), los siguientes son esenciales para la ejecución de instrucciones:

- Contador de programa (Program Counter, PC). Contiene la dirección de la próxima instrucción que se leerá de la memoria.

- Registro de instrucción (Instruction Register, IR). Contiene la última instrucción leída.

Todos los diseños de procesador incluyen también un registro, o conjunto de registros, conocido usualmente como la palabra de estado del programa (Program Status Word, PSW), que contiene información de estado. La PSW contiene normalmente códigos de condición, además de otra información de estado, tales como un bit para habilitar/inhabilitar las interrupciones y un bit de modo usuario/supervisor. Los códigos de condición (también llamados indicadores) son bits cuyo valor lo asigna normalmente el hardware de procesador teniendo en cuenta el resultado de las operaciones. Por ejemplo, una operación aritmética puede producir un resultado positivo, negativo, cero o desbordamiento. Además de almacenarse el resultado en sí mismo en un registro o en la memoria, se fija también un código de condición en concordancia con el resultado de la ejecución de la instrucción aritmética. Posteriormente, se puede comprobar el código de condición como parte de una operación de salto condicional.

debería estar en registros, más rápidos y más caros, y cuánta en la memoria principal, menos rápida y más económica.

1.3. EJECUCIÓN DE INSTRUCCIONES

Un programa que va a ejecutarse en un procesador consta de un conjunto de instrucciones almacenado en memoria. En su forma más simple, el procesamiento de una instrucción consta de dos pasos: el procesador lee (*busca*) instrucciones de la memoria, una cada vez, y ejecuta cada una de ellas. La ejecución del programa consiste en repetir el proceso de búsqueda y ejecución de instrucciones. La ejecución de la instrucción puede involucrar varias operaciones dependiendo de la naturaleza de la misma.

Se denomina *ciclo de instrucción* al procesamiento requerido por una única instrucción. En la Figura 1.2 se describe el ciclo de instrucción utilizando la descripción simplificada de dos pasos. A estos dos pasos se les denomina *fase de búsqueda* y de *ejecución*. La ejecución del programa se detiene sólo si se apaga la máquina, se produce algún tipo de error irrecuperable o se ejecuta una instrucción del programa que para el procesador.

BÚSQUEDA Y EJECUCIÓN DE UNA INSTRUCCIÓN

Al principio de cada ciclo de instrucción, el procesador lee una instrucción de la memoria. En un procesador típico, el contador del programa (PC) almacena la dirección de la siguiente instrucción que se va a leer. A menos que se le indique otra cosa, el procesador siempre incrementa el PC después de cada instrucción ejecutada, de manera que se leerá la siguiente instrucción en orden secuencial (es decir, la instrucción situada en la siguiente dirección de memoria más alta). Considere, por ejemplo, un computador simplificado en el que cada instrucción ocupa una palabra de memoria de 16 bits. Suponga que el contador del programa está situado en la posición 300. El procesador leerá la siguiente instrucción de la posición 300. En sucesivos ciclos de instrucción completados satisfactoriamente, se leerán instrucciones de las posiciones 301, 302, 303, y así sucesivamente. Esta secuencia se puede alterar, como se explicará posteriormente.

La instrucción leída se carga dentro de un registro del procesador conocido como registro de instrucción (IR). La instrucción contiene bits que especifican la acción que debe realizar el procesador. El procesador interpreta la instrucción y lleva a cabo la acción requerida. En general, estas acciones se dividen en cuatro categorías:

- **Procesador-memoria.** Se pueden transferir datos desde el procesador a la memoria o viceversa.
- **Procesador-E/S.** Se pueden enviar datos a un dispositivo periférico o recibirlos desde el mismo, transfiriéndolos entre el procesador y un módulo de E/S.

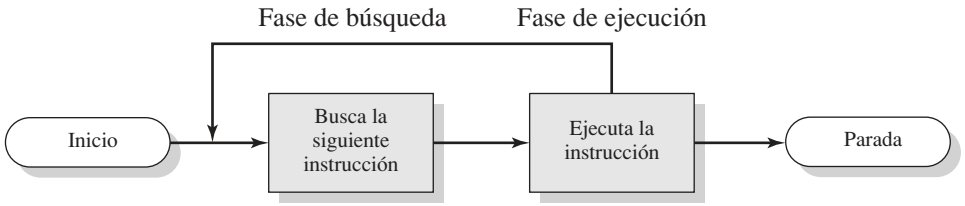


Figura 1.2. Ciclo de instrucción básico.

- **Procesamiento de datos.** El procesador puede realizar algunas operaciones aritméticas o lógicas sobre los datos.
- **Control.** Una instrucción puede especificar que se va a alterar la secuencia de ejecución. Por ejemplo, el procesador puede leer una instrucción de la posición 149, que especifica que la siguiente instrucción estará en la posición 182. El procesador almacenará en el contador del programa un valor de 182. Como consecuencia, en la siguiente fase de búsqueda, se leerá la instrucción de la posición 182 en vez de la 150.

Una ejecución de una instrucción puede involucrar una combinación de estas acciones.

Considere un ejemplo sencillo utilizando una máquina hipotética que incluye las características mostradas en la Figura 1.3. El procesador contiene un único registro de datos, llamado el acumulador (AC). Tanto las instrucciones como los datos tienen una longitud de 16 bits, estando la memoria organizada como una secuencia de palabras de 16 bits. El formato de la instrucción proporciona 4 bits para el código de operación, permitiendo hasta $2^4 = 16$ códigos de operación diferentes (representados por un único dígito hexadecimal[3]). Con los 12 bits restantes del formato de la instrucción, se pueden direccionar directamente hasta $2^{12} = 4.096$ (4K) palabras de memoria (denotadas por tres dígitos hexadecimales).

La Figura 1.4 ilustra una ejecución parcial de un programa, mostrando las partes relevantes de la memoria y de los registros del procesador. El fragmento de programa mostrado suma el contenido de la palabra de memoria en la dirección 940 al de la palabra de memoria en la dirección 941, almacenando el resultado en esta última posición. Se requieren tres instrucciones, que corresponden a tres fases de búsqueda y de ejecución, como se describe a continuación:

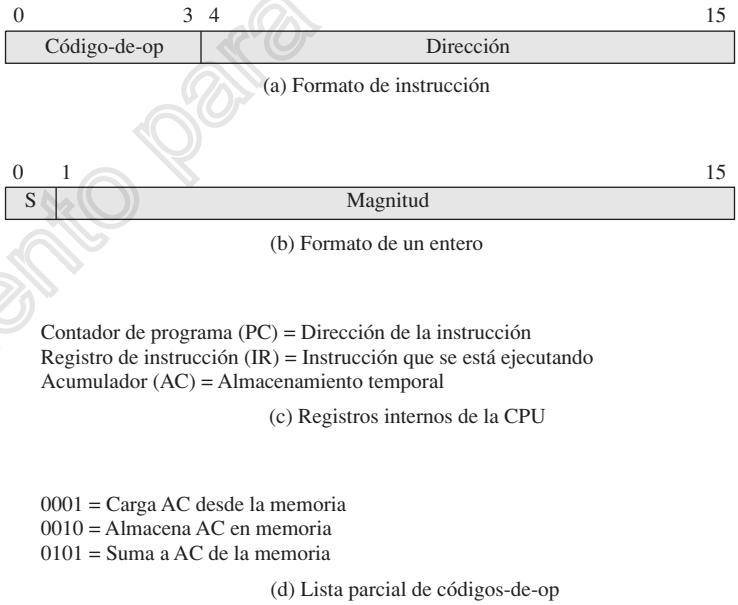


Figura 1.3. Características de una máquina hipotética.

[3] Puede encontrar un repaso básico de los sistema numéricos (decimal, binario y hexadecimal) en el *Computer Science Student Resource Site* en **WilliamStallings.com/StudentSupport.html**.

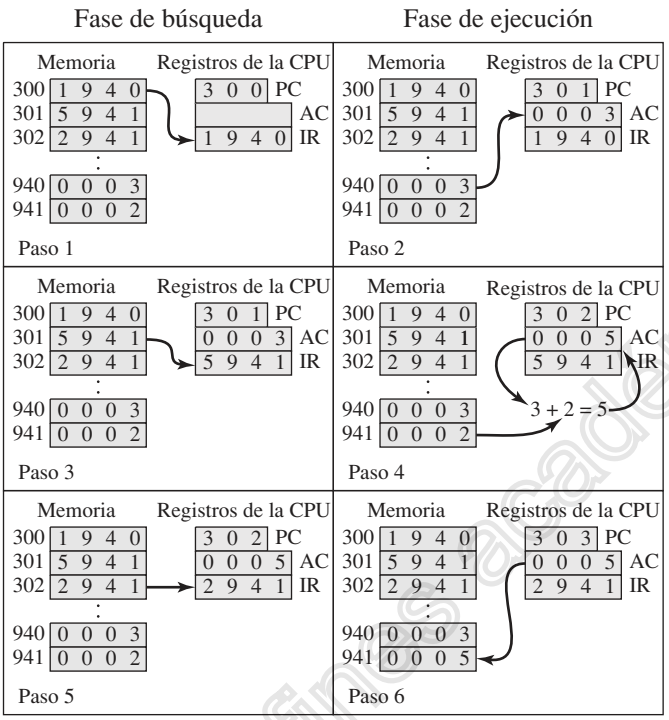


Figura 1.4. Ejemplo de ejecución de un programa (contenido de la memoria y los registros en hexadecimal).

1. El PC contiene el valor 300, la dirección de la primera instrucción. Esta instrucción (el valor 1940 en hexadecimal) se carga dentro del registro de instrucción IR y se incrementa el PC. Nótese que este proceso involucra el uso del registro de dirección de memoria (RDIM) y el registro de datos de memoria (RDAM). Para simplificar, no se muestran estos registros intermedios.
2. Los primeros 4 bits (primer dígito hexadecimal) en el IR indican que en el AC se va a cargar un valor leído de la memoria. Los restantes 12 bits (tres dígitos hexadecimales) especifican la dirección de memoria, que es 940.
3. Se lee la siguiente instrucción (5941) de la posición 301 y se incrementa el PC.
4. El contenido previo del AC y el contenido de la posición 941 se suman y el resultado se almacena en el AC.
5. Se lee la siguiente instrucción (2941) de la posición 302 y se incrementa el PC.
6. Se almacena el contenido del AC en la posición 941.

En este ejemplo, se necesitan tres ciclos de instrucción, de tal forma que cada uno consta de una fase de búsqueda y una fase de ejecución, para sumar el contenido de la posición 940 al contenido de la 941. Con un juego de instrucciones más complejo, se necesitarían menos ciclos de instrucción. La mayoría de los procesadores modernos incluyen instrucciones que contienen más de una dirección. Por tanto, la fase de ejecución de una determinada instrucción puede involucrar más de una referencia a memoria. Asimismo, en vez de referencias a memoria, una instrucción puede especificar una operación de E/S.

Interrupciones

Prácticamente todos los computadores proporcionan un mecanismo por el cual otros módulos (módulo de memoria y módulo de E/S – se estudiará posteriormente –) pueden interrumpir el secuenciamiento normal del procesador. La Tabla 1.1 detalla los tipos más comunes de interrupciones.

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.

Básicamente, las interrupciones constituyen una manera de mejorar la utilización del procesador. Por ejemplo, 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 utilizando el esquema de ciclo de instrucción de la Figura 1.2.

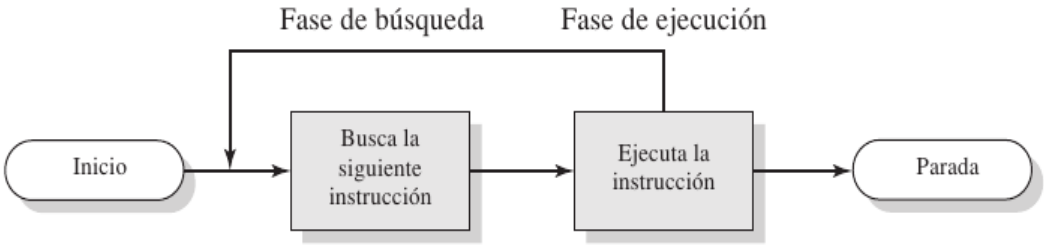
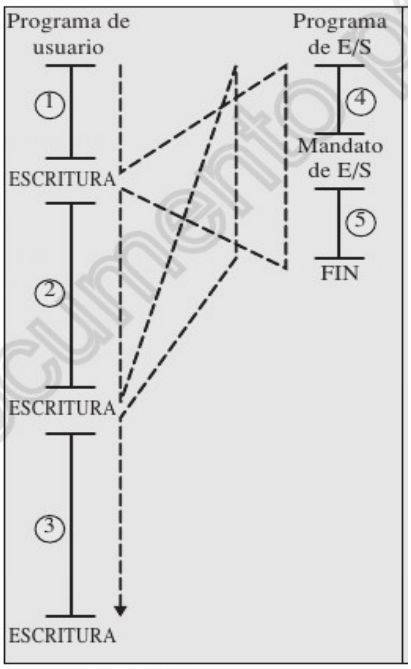


Figura 1.2. Ciclo de instrucción básico.

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 que debe hacer (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 rutina de E/S (se estudia a continuación) que es una utilidad del sistema que realizará la operación real de E/S. El programa de E/S consta de tres secciones:

- Una secuencia de instrucciones, etiquetada como 4 en la figura, para preparar la operación real de E/S. Esto puede incluir copiar los datos de salida en un buffer especial y preparar los parámetros de un mandato para el dispositivo.
- El mandato real de E/S. Sin el uso de interrupciones, una vez que se emite este mandato, el programa debe esperar a que el dispositivo de E/S realice la función solicitada (o comprobar periódicamente el estado, o muestrear, el dispositivo de E/S). El programa podría esperar simplemente realizando repetidamente una operación de comprobación para determinar si se ha realizado la operación de E/S.
- Una secuencia de instrucciones, etiquetada como 5 en la figura, para completar la operación. Esto puede incluir establecer un valor que indique el éxito o el fallo de la operación.

Debido a que la operación de E/S puede tardar un tiempo relativamente largo hasta que se completa, el programa de E/S se queda colgado esperando que se complete; por ello, el programa de usuario se detiene en el momento de la llamada de ESCRITURA durante un periodo de tiempo considerable. Aparece entonces el concepto de interrupción para poder aprovechar el procesador y que éste no quede ocioso cuando se produzca una de las causas de la Tabla 1.1 (clases de interrupciones).

Interrupciones y ciclo de instrucción

Gracias a las interrupciones, el procesador puede 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 de más abajo. Como anteriormente, el programa de usuario (Ej: la opción de imprimir de un editor de textos) alcanza un punto en el que hace una llamada al sistema que consiste en una llamada de ESCRITURA. El programa de E/S que se invoca en este caso consta sólo del código de preparación y el mandato real de E/S. Después de que se ejecuten estas pocas instrucciones, se devuelve el control al programa de usuario. Mientras tanto, el dispositivo externo está ocupado aceptando datos

de la memoria del computador e imprimiéndolos. La operación de E/S se lleva a cabo de forma concurrente con la ejecución de instrucciones en el programa de usuario.

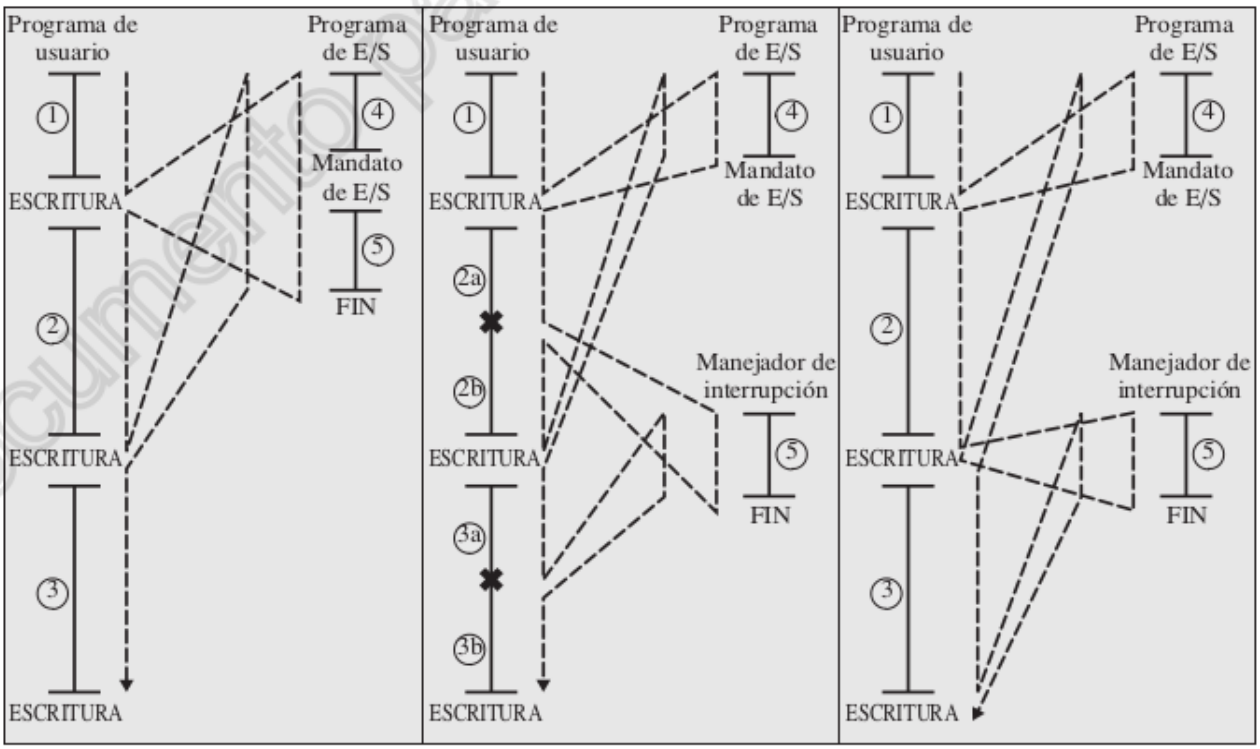


Figura 1.5. Flujo de programa del control sin interrupciones y con ellas.

Cuando el dispositivo externo está listo para ser atendido, es decir, cuando está preparado para aceptar más datos del procesador (en el ejemplo de la impresora, que su memoria local disponga de espacio para seguir recibiendo datos a almacenar y posteriormente imprimir), el módulo de E/S de este dispositivo externo manda una señal de petición de interrupción al procesador. El procesador responde suspendiendo la ejecución del programa actual, saltando a la rutina de servicio específica de este dispositivo de E/S (rutina o pequeño programa que maneja un determinado tipo de interrupciones), conocida como manejador de interrupción, y reanudando la ejecución original después de haber atendido al dispositivo. 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 determinada instrucción.

De cara al programa de usuario, una interrupción suspende la secuencia normal de ejecución. Cuando se completa el procesamiento de la interrupción, se reanuda la ejecución (Figura 1.6). 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.

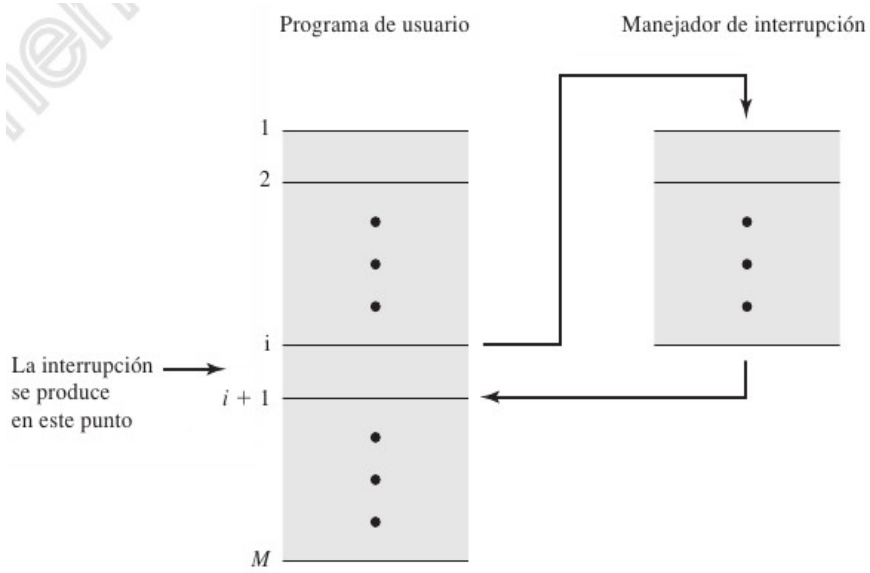


Figura 1.6. Transferencia de control mediante interrupciones.

Para tratar las interrupciones, se añade una fase de interrupción al ciclo de instrucción, como se muestra en la Figura 1.7 (compárese con la Figura 1.2). 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). 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 la rutina o programa del manejador de interrupción (*Interrupt Service Routine - ISR*). La rutina del manejador de interrupción es generalmente parte del sistema operativo y está cargada siempre en memoria principal. Normalmente, esta rutina determina la naturaleza de la interrupción y realiza las acciones que se requieran. En el ejemplo que se está usando, el manejador determina qué módulo de E/S generó la interrupción y puede dar paso a un programa que escriba más datos en ese módulo de E/S. Cuando se completa la rutina del manejador de interrupción, el procesador puede reanudar la ejecución del programa de usuario en el punto de la interrupción. En el siguiente tema veremos más concretamente qué datos son necesarios guardar del proceso que está actualmente en ejecución para dar paso al manejo de la operación de E/S.

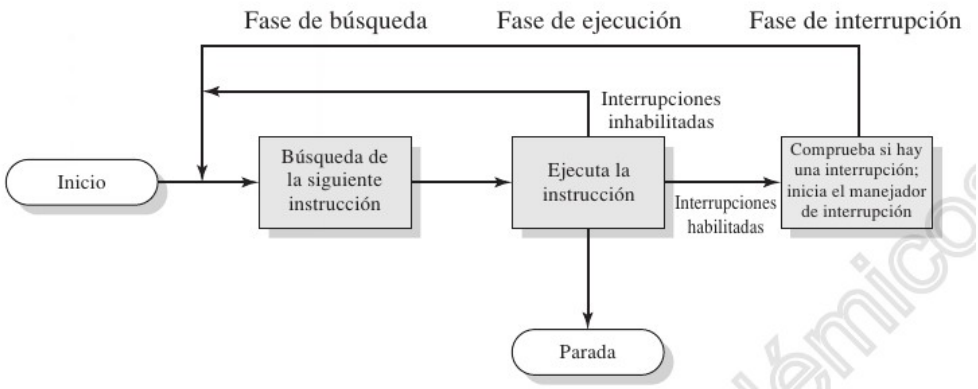


Figura 1.7. Ciclo de instrucción con interrupciones.

Es evidente que este proceso implica cierta sobrecarga. Deben ejecutarse instrucciones adicionales (en el manejador de interrupción) para determinar la naturaleza de la interrupción y decidir sobre la acción apropiada. Sin embargo, debido a la cantidad relativamente elevada de tiempo que se gastaría simplemente a la espera de una operación de E/S, el procesador se puede emplear mucho

más eficientemente con el uso de interrupciones.

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. Las Figuras 1.5b y 1.8 asumen que el tiempo requerido para la operación de E/S es relativamente corto: inferior al tiempo que tarda en completarse la ejecución de instrucciones entre las operaciones de escritura del programa de usuario. El caso más típico, especialmente para un dispositivo lento como una impresora, es que la operación de E/S tarde mucho más tiempo que la ejecución de una secuencia de instrucciones de usuario. La Figura 1.5c ilustra este tipo de situación. En este caso, el programa de usuario alcanza la segunda llamada de ESCRITURA antes de que se complete la operación de E/S generada por la primera llamada. El resultado es que el programa de usuario se queda colgado en ese punto. Cuando se completa la operación de E/S precedente, se puede procesar la nueva llamada de ESCRITURA y se puede empezar una nueva operación de E/S. La Figura 1.9 muestra la temporización de esta situación con el uso de interrupciones o sin ellas. Se puede observar que hay una ganancia en eficiencia debido a que parte del tiempo durante el que se realiza la operación de E/S se solapa con la ejecución de las instrucciones del usuario.

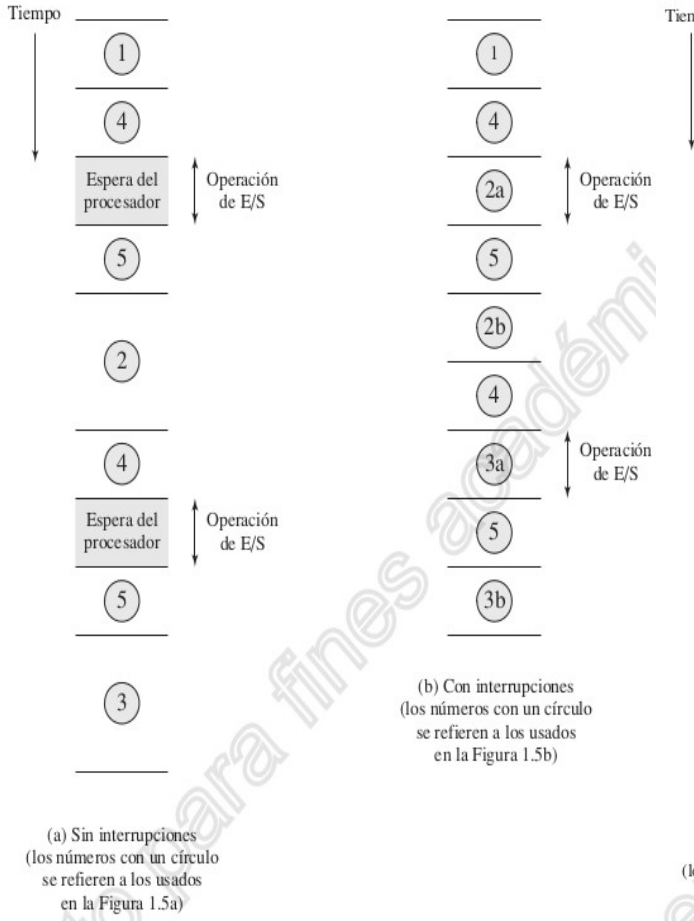


Figura 1.8. Temporización del programa: espera breve de E/S.

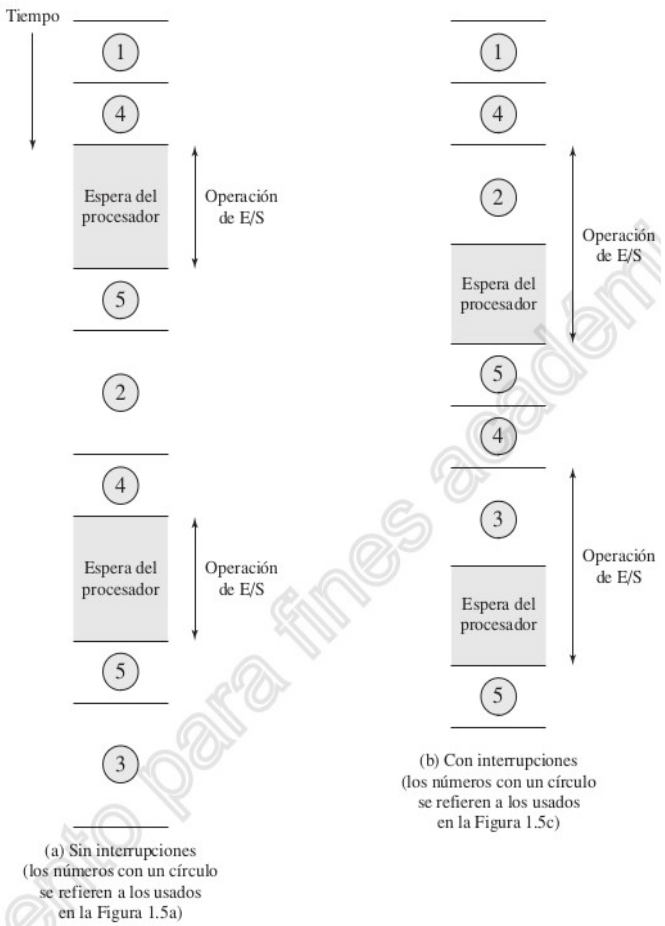


Figura 1.9. Temporización del programa: espera larga de E/S.

Procesamiento de interrupciones

La aparición de una interrupción dispara varios eventos, tanto en el hardware del procesador como en el software. La Figura 1.10 muestra una secuencia típica.

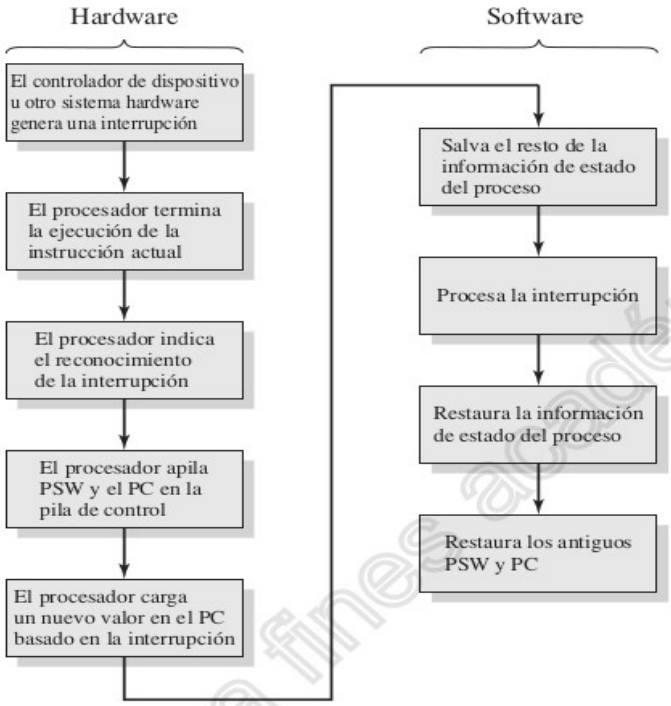


Figura 1.10. Procesamiento simple de interrupciones.

Cuando un dispositivo de E/S completa una operación de E/S, se produce la siguiente secuencia de eventos en el hardware:

1. El dispositivo genera una señal de interrupción hacia el procesador.

2. El procesador termina la ejecución de la instrucción actual antes de responder a la interrupción, como se indica en la Figura 1.7.

3. El procesador comprueba si hay una petición de interrupción pendiente, determina que hay una y manda una señal de reconocimiento al dispositivo que produjo la interrupción. Este reconocimiento permite que el dispositivo elimine su señal de interrupción.

4. En ese momento, el procesador necesita prepararse para transferir el control a la rutina de interrupción. 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 (veremos más sobre la información a guardar). Esta información se puede apilar en la pila de control de sistema (véase la sección “Control de procedimientos”).

5. A continuación, el procesador carga el contador del programa con la posición del punto de entrada de la rutina de manejo de interrupción que responderá a esta interrupción. Dependiendo de la arquitectura de computador y del diseño del sistema operativo, puede haber un único programa, uno por cada tipo de interrupción o uno por cada dispositivo y tipo de interrupción. Si hay más de una rutina de manejo de interrupción, el procesador debe determinar cuál invocar. Esta información puede estar incluida en la señal de interrupción original o el procesador puede tener que realizar una petición al dispositivo que generó la interrupción para obtener una respuesta que contiene la

información requerida.

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 conlleva las siguientes operaciones:

6. En este momento, el contador del programa 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, que se estudia en el siguiente tema). Generalmente, el manejador de interrupción comenzará salvando el contenido de todos los registros en la pila. La Figura 1.11a muestra un ejemplo sencillo. En este caso, un programa de usuario se interrumpe después de la instrucción en la posición N. El contenido de todos los registros, así como la dirección de la siguiente instrucción (N + 1), un total de M palabras, se apilan en la pila de control. El puntero de pila se actualiza para que señale a la nueva cima de la pila, mientras que el contador de programa quedará apuntando al principio de la rutina de servicio de interrupción.

7. 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 o con otro evento distinto que haya causado la interrupción. Asimismo, puede implicar el envío de mandatos adicionales o reconocimientos al dispositivo de E/S.

8. 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 (como ejemplo, véase la Figura 1.11b).

9. La última acción consiste en restituir de la pila los valores de la PSW y del contador del programa. 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. Esto se debe a que la interrupción no es una rutina llamada desde el programa (OJO, diferenciación importante). En su lugar, 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.

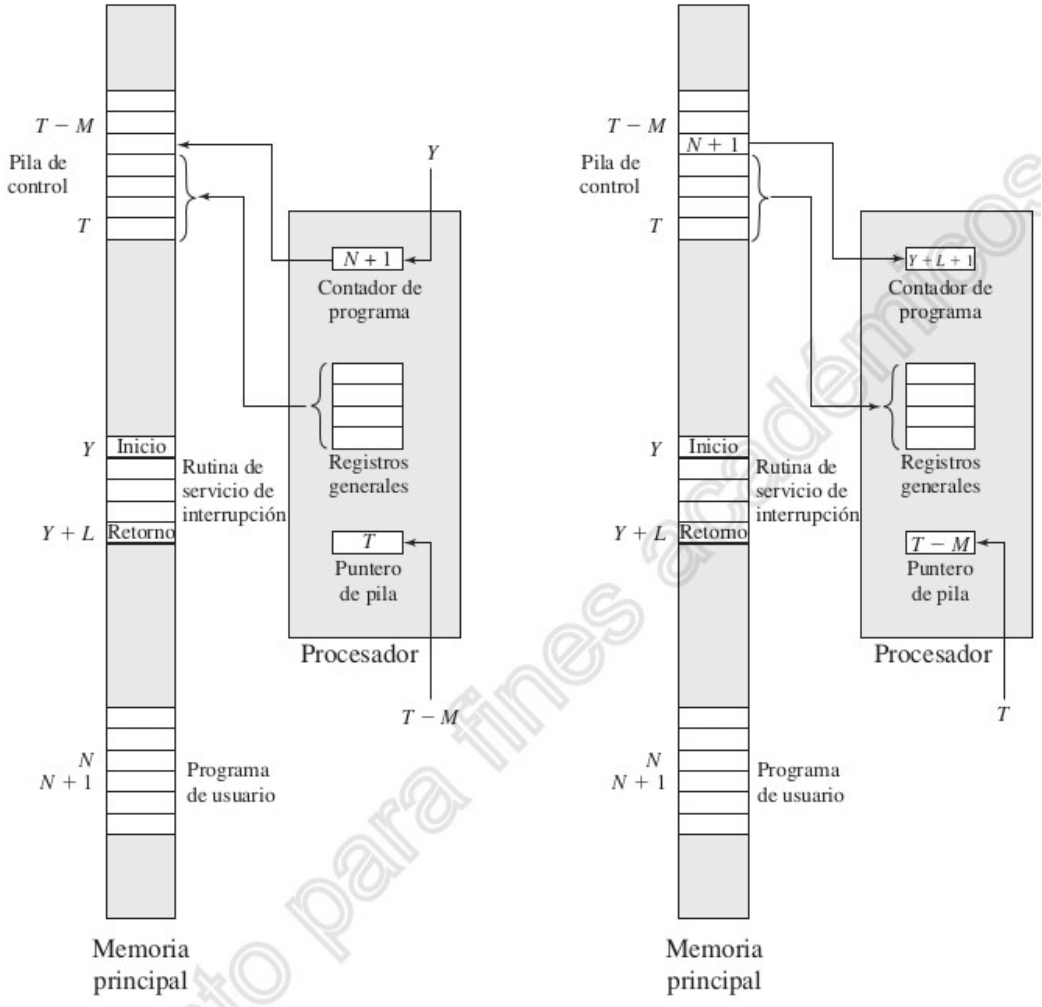


Figura 1.11. Cambios en la memoria y en los registros durante una interrupción.

Resumidamente, cuando ocurre una interrupción, se han de llevar a cabo las siguientes operaciones:

1. El control pasa al sistema operativo.

2. El sistema operativo almacena el estado completo del proceso interrumpido para poder continuar su ejecución cuando se sirva la interrupción. En algunos casos esto se hace mediante instrucciones especiales.

3. Se analiza la interrupción producida y se pasa el control a la rutina de servicio correspondiente (es otro pequeño programa encargado de tratar las interrupciones). En muchos sistemas esta acción la realiza el hardware automáticamente.

4. Se ejecuta la rutina de servicio.

5. Se ejecuta el siguiente proceso que seleccione el planificador (se estudiará más adelante), cargando su contexto salvado. En muchos casos éste puede ser un proceso distinto al interrumpido.

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. 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. 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, 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).

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.

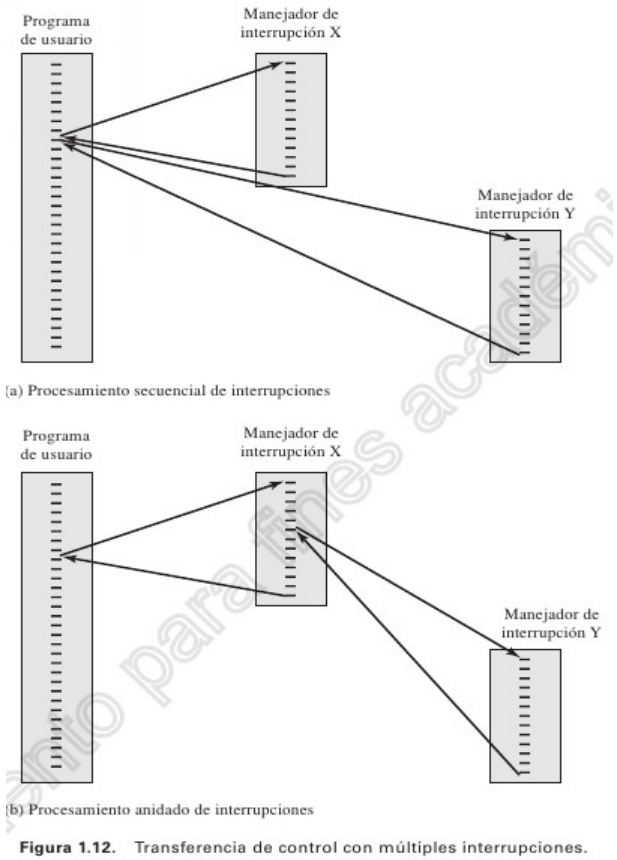


Figura 1.12. Transferencia de control con múltiples interrupciones.

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. 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.

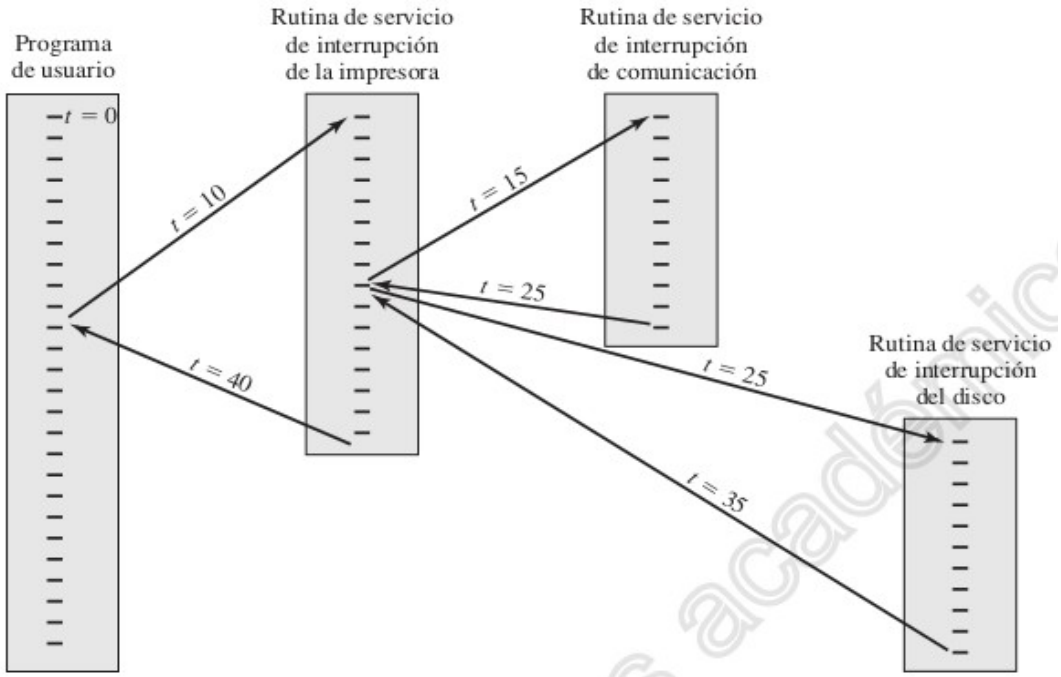


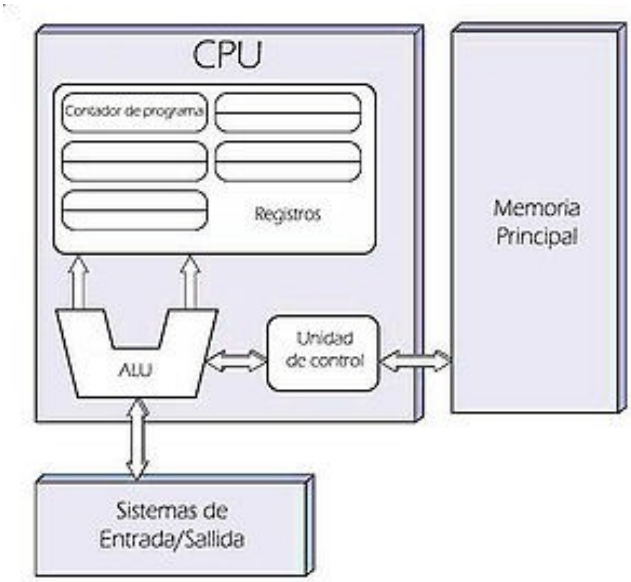
Figura 1.13. Ejemplo de secuencia de tiempo con múltiples interrupciones.

Sistema de Entrada-Salida (E/S)

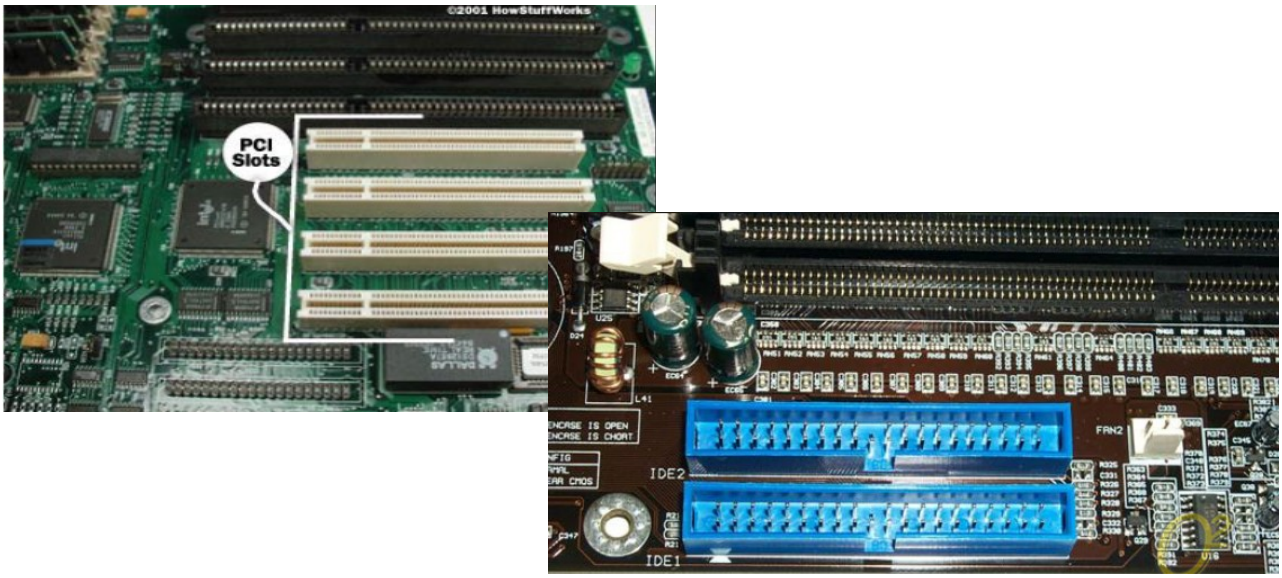
Los ordenadores actuales se basan en la arquitectura de Von Neumann. Los ordenadores con esta arquitectura constan de tres partes claramente diferencias: El procesador (unidad aritmético-lógica o ALU, la unidad de control y los registros), la memoria y uno o varios dispositivos de entrada/salida (teclado, impresora, disco duro, tarjeta de red, monitor, etc). 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.

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).



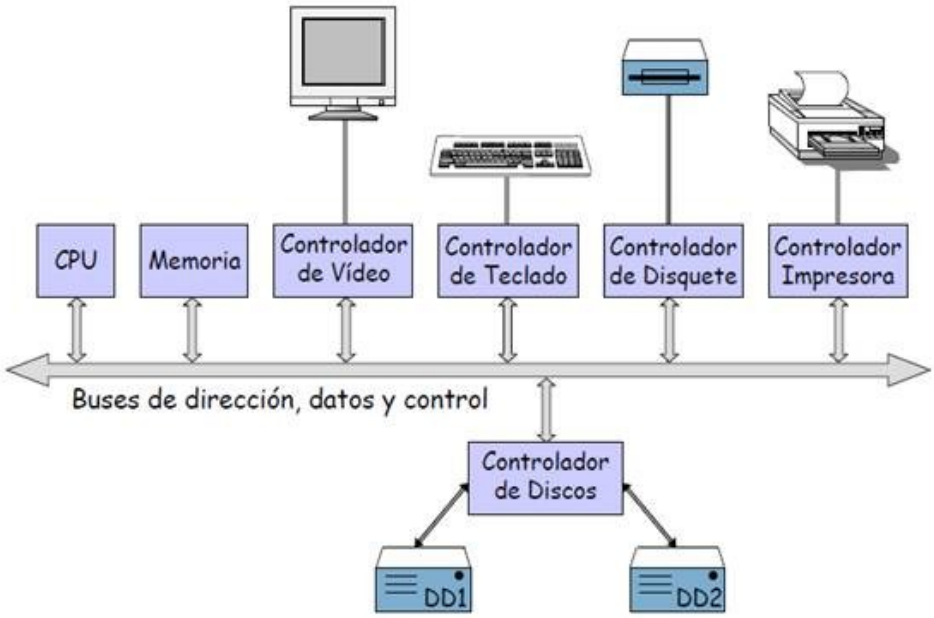
El sistema o módulo de E/S (también llamado **controlador**) es un circuito o chip integrado en la placa base, separado en muchos casos físicamente del procesador y la memoria pero unido a estos mediante buses en la misma placa. En muchos computadores este módulo es conocido como puente norte y puente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

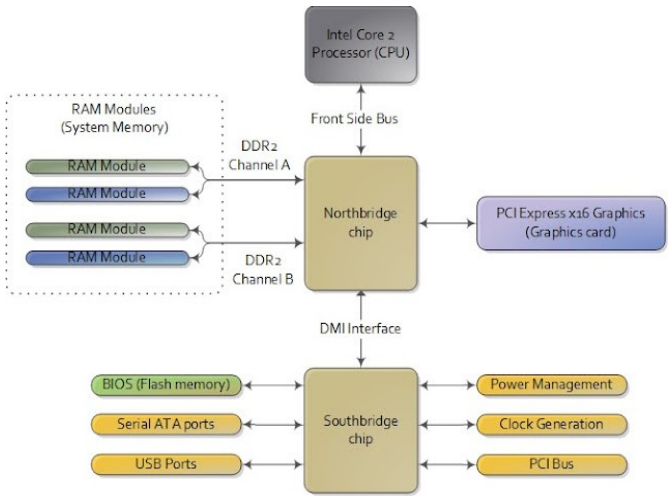


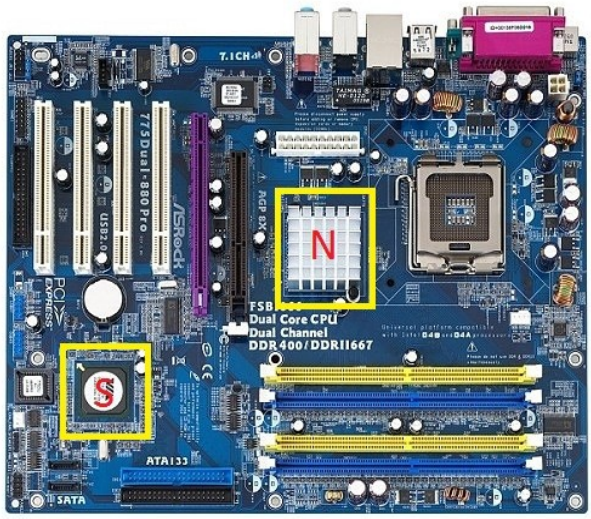
Un módulo de E/S puede controlar múltiples dispositivos (discos, impresoras, monitores, tarjetas de red, etc). El SO, normalmente, trata con el módulo de E/S, y no directamente con el dispositivo. Cada dispositivo de E/S está hecho por un fabricante diferente y puede tener diferentes registros y chips 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**. 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 (se estudia más adelante) 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.







De manera general (veremos que hay variaciones), la transferencia de datos entre un dispositivo externo y la CPU necesita la siguiente secuencia de acciones:

1. La CPU pide al módulo de E/S que compruebe el estado del dispositivo externo al que está conectado.

2. El módulo de E/S devuelve el estado del dispositivo externo.

3. Si el dispositivo está operativo y preparado para transmitir, la CPU solicita la transferencia del dato mediante una orden al módulo de E/S.

4. El módulo de E/S obtiene los datos del dispositivo externo.

5. El dato se transfiere desde el módulo de E/S a la CPU.

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

Cuando el procesador ejecuta un programa y encuentra una instrucción relacionada con la E/S, ejecuta esa instrucción generando un mandato 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 y fija los bits correspondientes en el registro de estado de E/S, pero no realiza ninguna acción para avisar al procesador. En concreto, no interrumpe 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.

Con esta técnica, el procesador es responsable de extraer los datos de la memoria principal en una operación de salida y de almacenarlos en ella en una operación de entrada. El software de E/S se escribe de manera que el procesador ejecuta instrucciones que le dan control directo de la operación de E/S, incluyendo comprobar el estado del dispositivo, enviar un mandato de lectura o de escritura, y transferir los datos. Por tanto, el juego de instrucciones incluye instrucciones de E/S de 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 y/o escribir datos entre los registros del procesador y los dispositivos externos.

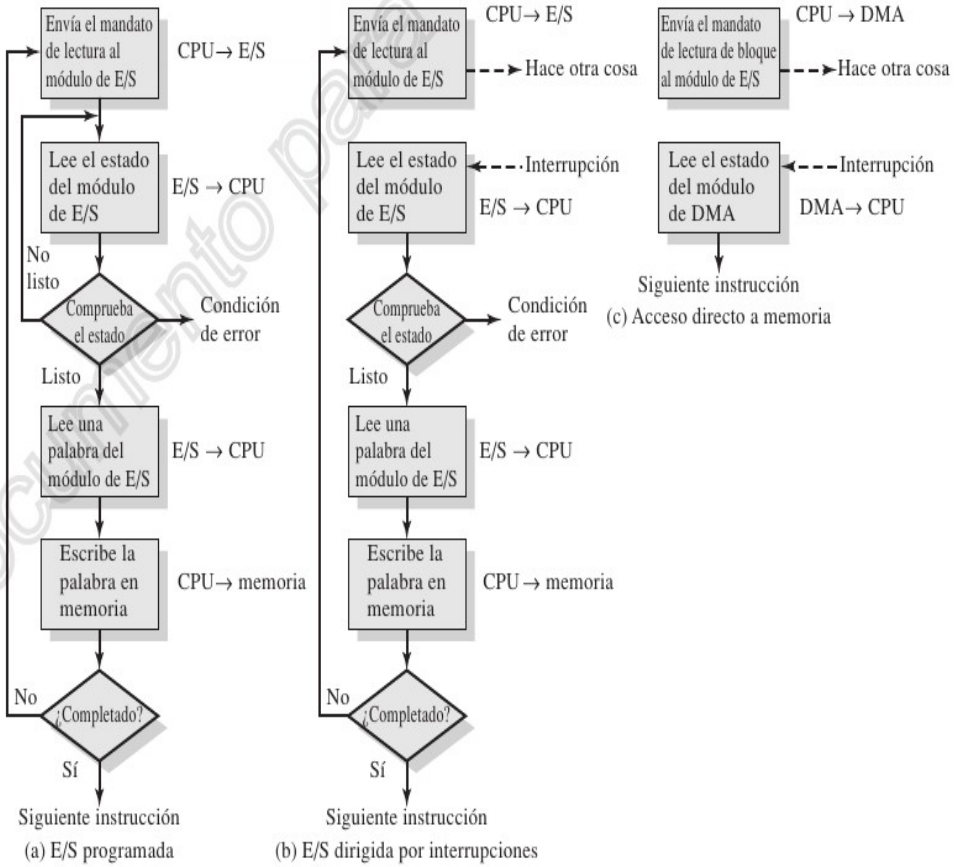


Figura 1.19. Tres técnicas para leer un bloque de datos.

La Figura 1.19a 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 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.

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. 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. Pare ello, el procesador guardaría de alguna forma de contexto del proceso que actualmente esté en ejecución (por ejemplo, el contador de programa y otros registros) y saltaría a una rutina de tratamiento de interrupciones,

que determinaría la naturaleza de la interrupción y la delegaría al módulo de E/S, para acto seguido ejecutar otro proceso diferente al que solicitó la E/S.

Una vez el módulo de E/S haya terminado con la operación que se le solicitó, interrumpirá al procesador para solicitar su servicio cuando esté listo para intercambiar datos con el mismo. El procesador ejecutará la transferencia de datos, y después reanudará el procesamiento del proceso actual o del proceso que estuviera esperando los datos que el módulo de E/S ha proporcionado.

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 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 en varios programas diferentes a la vez). 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 contexto del programa que se está ejecutando actualmente y comienza a ejecutar un programa de manejo de interrupción que procesa la interrupción. En este caso, el procesador lee la palabra de datos del módulo de E/S y la almacena en memoria. A continuación, restaura el contexto del programa que había realizado el mandato de E/S (o de algún otro programa) y reanuda su ejecución.

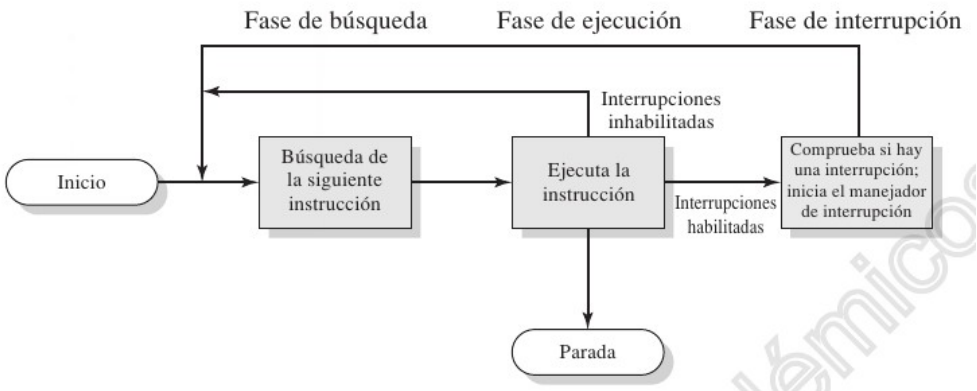


Figura 1.7. Ciclo de instrucción con interrupciones.

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. 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. Alternativamente, puede haber una única línea de interrupción, pero se utilizan líneas adicionales para guardar la dirección de un

dispositivo. De nuevo, se le asignan diferentes prioridades a los distintos dispositivos.

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, 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. En cualquier caso, la técnica funciona como se describe a continuación. 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, el procesador continúa con otro trabajo. Ha delegado esta operación de E/S al módulo de DMA, que 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.

Por tanto, el procesador solo está involucrado al principio y al final de la transferencia (Fig 1.19c). El módulo de DMA necesita tomar el control del bus para transferir datos hacia la memoria o desde ella. Debido a esta competencia en el uso del bus, puede haber veces en las que el procesador necesita el bus y debe esperar al módulo de DMA. Esto no es una interrupción; el procesador no salva un contexto y pasa a hacer otra cosa. En su lugar, el procesador se detiene durante un ciclo de bus (el tiempo que se tarda en transferir una palabra a través del bus). El efecto global es causar que el procesador ejecute más lentamente durante una transferencia de DMA en el caso de que el procesador requiera acceso al bus. Sin embargo, para una transferencia de E/S de múltiples palabras, el DMA es mucho más eficiente que la E/S dirigida por interrupciones o la programada.

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.

Implementación de la pila

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).

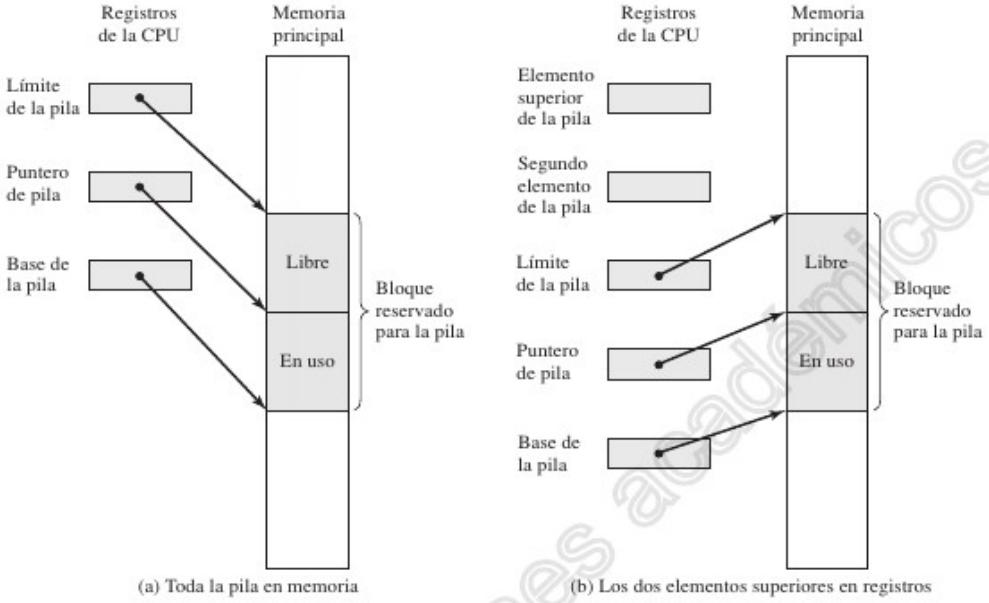


Figura 1.25. Organización habitual de la pila.

La implementación de una pila requiere que haya un conjunto de posiciones dedicado a almacenar los elementos de la pila. En la Figura 1.25 se muestra una técnica habitual. Se reserva en memoria principal (o memoria virtual) 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:

- 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.

Tradicionalmente, y en la mayoría de las máquinas actuales, la base de la pila está en el extremo con la dirección más alta del bloque de pila reservado, y el límite está en el extremo con la dirección más baja. Por tanto, la pila crece desde las direcciones más altas a las más bajas.

Llamadas y retornos de procedimientos

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. 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.

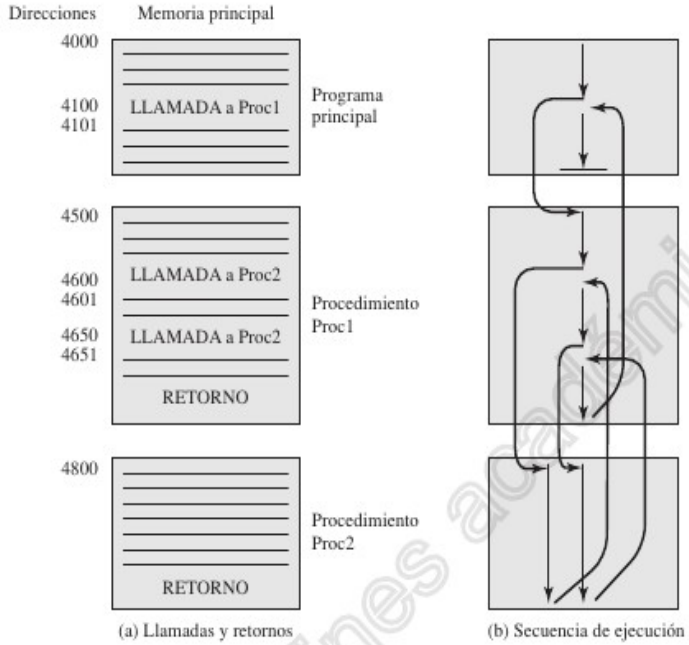


Figura 1.26. Procedimientos anidados.

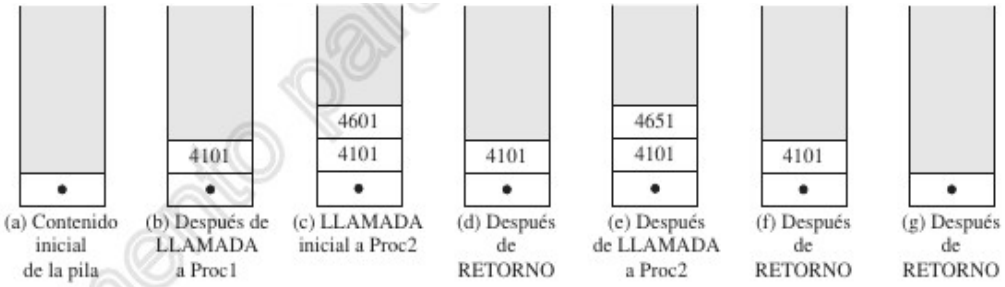


Figura 1.27. Uso de la pila para implementar los procedimientos anidados de la Figura 1.26.

Es también necesario con frecuencia pasar parámetros en una llamada a procedimiento. Estos se podrían pasar en los registros. Otra 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. Ambas técnicas tienen sus inconvenientes. Si se utilizan los registros, el programa llamado y el que realiza la llamada deben escribirse de manera que se asegure que los registros se utilizan apropiadamente. El almacenamiento de parámetros en memoria hace difícil intercambiar un número variable de parámetros.

Una estrategia más flexible para el paso de parámetros es la pila. 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.

En la Figura 1.28 se muestra un ejemplo. El ejemplo se refiere a un procedimiento P en el que se declaran las variables locales x1 y x2, y el procedimiento Q, al cual puede llamar P y en el que se declaran las variables y1 y y2. El primer elemento almacenado en cada marco de pila es un puntero al principio del marco previo. Esta técnica se necesita si el número o la longitud de los parámetros que se van a apilar es variable. A continuación, se almacena el punto de retorno del procedimiento que corresponde a este marco de pila. Finalmente, se reserva espacio en la cima del marco de pila

para las variables locales. Estas variables locales se pueden utilizar para el paso de parámetros. Por ejemplo, supóngase que cuando P llama a Q le pasa un valor como parámetro. Este valor se podría almacenar en la variable y1. Por tanto, en un lenguaje de alto nivel, habría una instrucción en la rutina P similar a la siguiente:

LLAMADA Q(y1)

Cuando se ejecuta esta llamada, se crea un nuevo marco de pila para Q (Figura 1.28b), que incluye un puntero al marco de pila para P, la dirección de retorno de P, y dos variables locales para Q, una de las cuales se inicia con el valor pasado como parámetro desde P. La otra variable local, y2, es simplemente una variable local utilizada por Q en sus cálculos. En el siguiente apartado se analiza la necesidad de incluir las variables locales en el marco de pila.

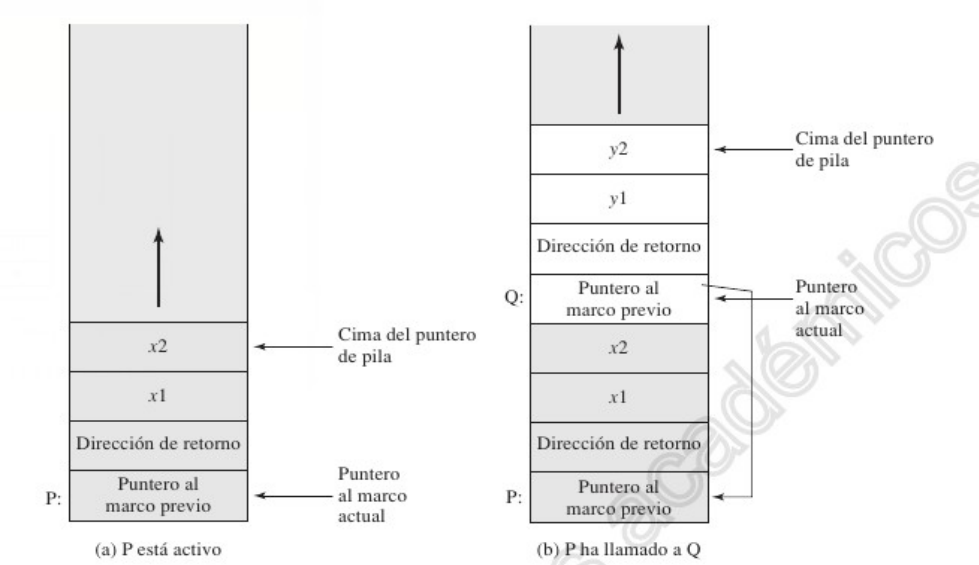


Figura 1.28. Crecimiento del marco de pila utilizando los procedimientos de ejemplo P y Q.

Multiprogramación

Aunque no hemos hablado de ello explicitamente, 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.

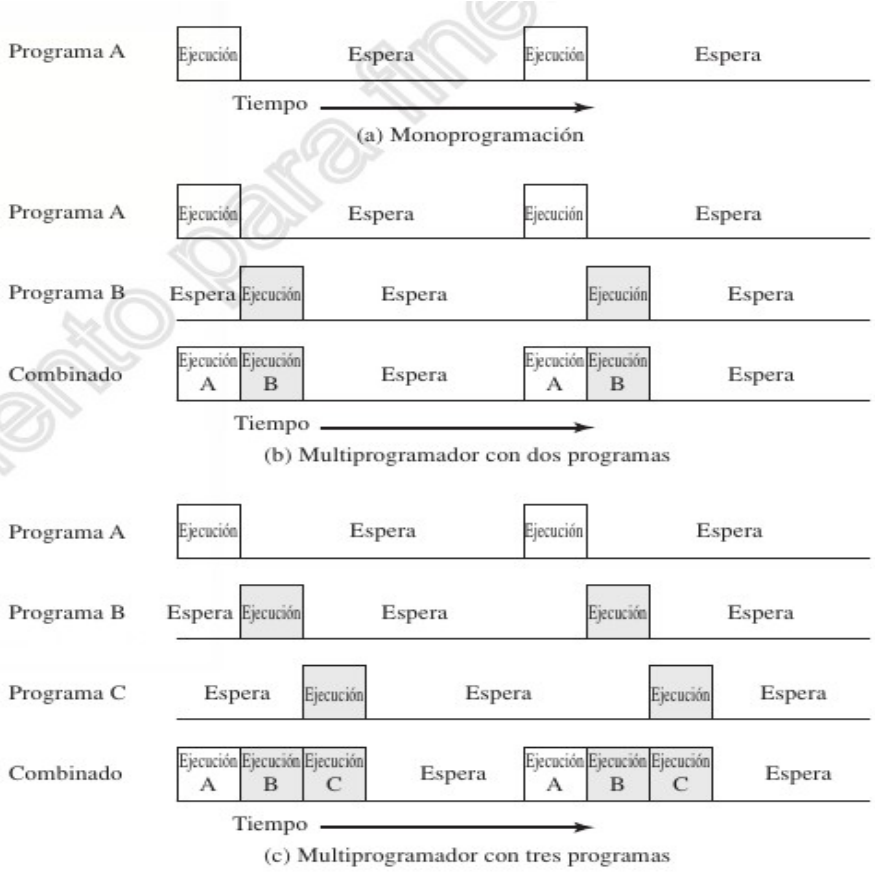


Figura 2.5. Ejemplo de multiprogramación.

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".

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 tanto en sistemas operativos nuevos como en nuevas versiones de sistemas operativos existentes que han creado un cambio fundamental en la naturaleza de los sistemas operativos. Estos sistemas operativos modernos responden a las 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 (e.j. Bit NX. El bit NX es una característica física del procesador que permite al sistema operativo prohibir la ejecución e inyección de código en área de datos, mejorando la seguridad. Esta características está disponible en los modos de 32 y 64 bits, y está soportada por Linux, Solaris, y a partir de Windows XP SP2, Windows Server 2003 SP1).

Multihilo

Multitheading es una técnica en la cual un proceso, ejecutando una aplicación, se divide en una serie de hilos o threads que pueden ejecutar concurrentemente. Se pueden hacer las siguientes distinciones:

- Thread o hilo. Se trata de una unidad de trabajo. Incluye el contexto del procesador (que con tiene el contador del programa y el puntero de pila) y su propia área de datos para una pila (para posibilitar el salto a subrutinas). Un hilo se ejecuta secuencialmente y se puede interrumpir de forma que el procesador pueda dar paso a otro hilo.
- Proceso. Es una colección de uno o más hilos y sus recursos de sistema asociados (como la memoria, conteniendo tanto código, como datos, ficheros abiertos y dispositivos). Esto corresponde al concepto de programa en ejecución. Dividiendo una sola aplicación en múltiples hilos, el programador tiene gran control sobre la modularidad de las aplicaciones y la temporización de los eventos relacionados con la aplicación.

La técnica multithreading es útil para las aplicaciones que llevan a cabo un número de tareas esencialmente independientes que no necesitan ser serializadas. Un ejemplo es un servidor de bases de datos que escucha y procesa numerosas peticiones de cliente. Con múltiples hilos ejecutándose dentro del mismo proceso, intercambiar la ejecución entre los hilos supone menos sobrecarga del procesador que intercambiar la ejecución entre diferentes procesos pesados. Los hilos son también útiles para estructurar procesos que son parte del núcleo del sistema operativo

Multiprocesamiento simétrico

Hasta hace poco tiempo, los computadores personales y estaciones de trabajo de un único usuario contenían un único procesador de propósito general. A medida que la demanda de rendimiento se incrementa y el coste de los microprocesadores continúa cayendo, los fabricantes han introducido en el mercado computadores con múltiples procesadores. Para lograr mayor eficiencia y fiabilidad, una técnica consiste en emplear **multiprocesamiento simétrico** (**SMP**: Symmetric MultiProcessing), un término que se refiere a la arquitectura hardware del computador y también al comportamiento del sistema operativo que explota dicha arquitectura. Se puede definir un multiprocesador simétrico como un sistema de computación aislado con las siguientes características:

1. Tiene múltiples procesadores.

2. Estos procesadores comparten las mismas utilidades de memoria principal y de E/S, interconectadas por un bus de comunicación u otro esquema de conexión interna.

3. Todos los procesadores pueden realizar las mismas funciones (de ahí el término simétrico).

El sistema operativo de un SMP planifica procesos o hilos a través de todos los procesadores. SMP tiene diversas ventajas potenciales sobre las arquitecturas monoprocesador, entre las que se incluyen:

- **Rendimiento**. Si el trabajo se puede organizar de tal forma que alguna porción del trabajo se pueda realizar en paralelo, entonces un sistema con múltiples procesadores alcanzará mayor rendimiento que uno con un solo procesador del mismo tipo. Esto se muestra en la Figura 2.12. Con la multiprogramación, sólo un proceso puede ejecutar a la vez; mientras tanto, el resto de los procesos esperan por el procesador. Con multiproceso, más de un proceso puede ejecutarse simultáneamente, cada uno de ellos en un procesador diferente.
- **Disponibilidad**. En un multiprocesador simétrico, debido a que todos los procesadores pueden llevar a cabo las mismas funciones, el fallo de un solo procesador no para la máquina. Por el contrario, el sistema puede continuar funcionando con un rendimiento reducido.
- **Crecimiento incremental**. Un usuario puede mejorar el rendimiento de un sistema añadiendo un procesador adicional.
- **Escalado**. Los fabricantes pueden ofrecer un rango de productos con diferente precio y características basadas en el número de procesadores configurado en el sistema.

Es importante notar que estas características son beneficios potenciales, no garantizados. El sistema operativo debe proporcionar herramientas y funciones para explotar el paralelismo en un sistema SMP.

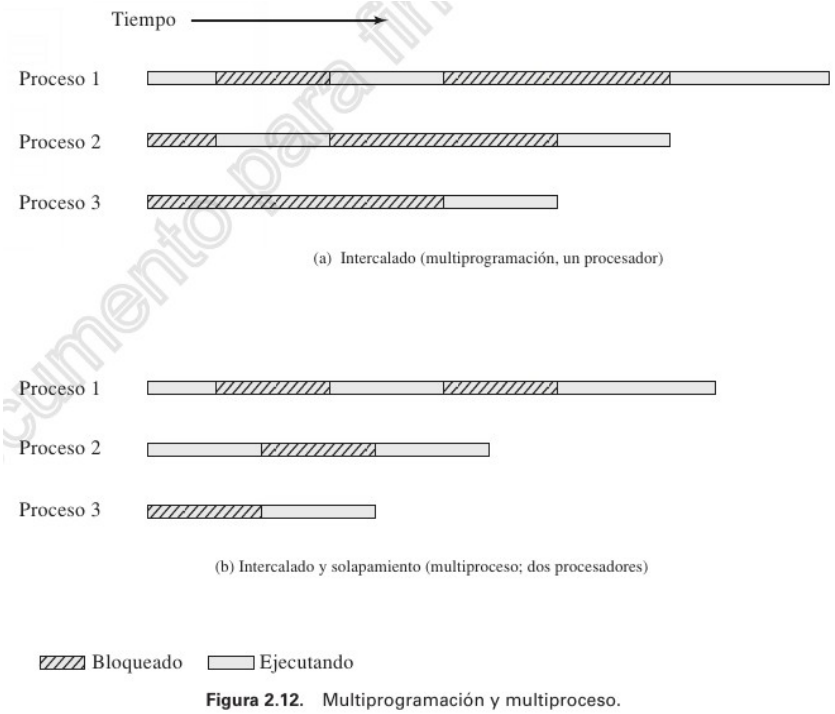
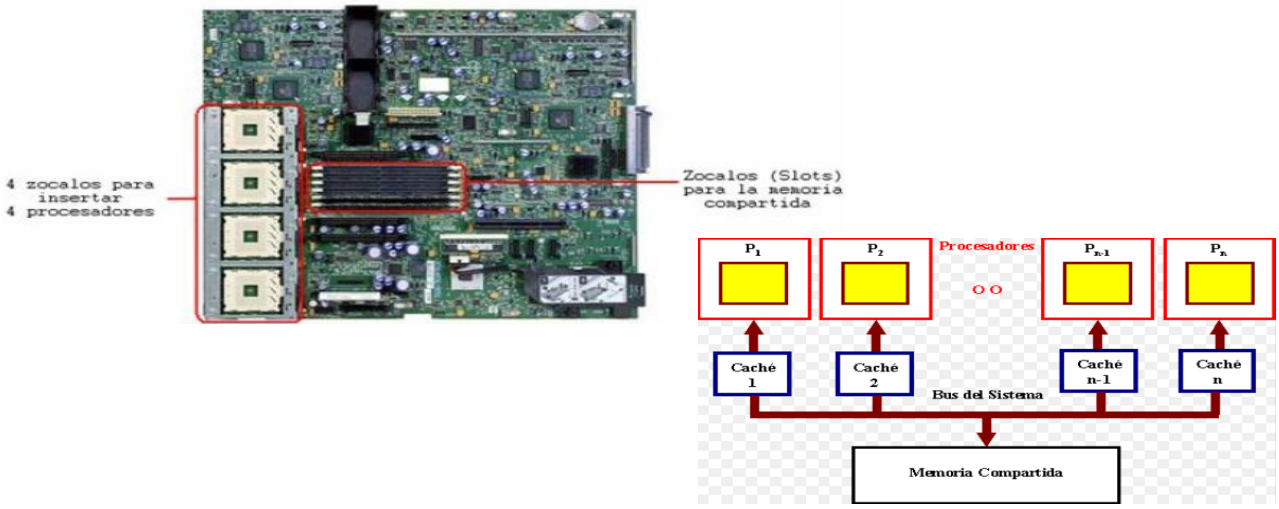
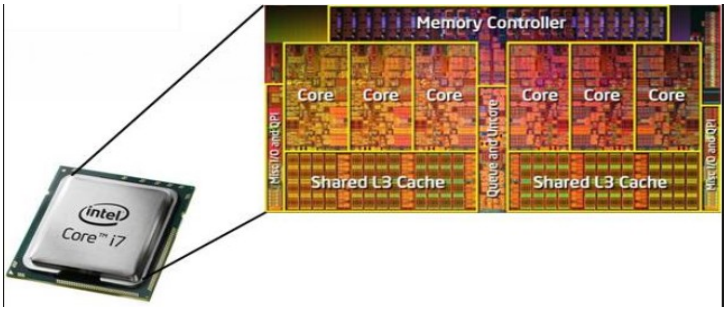


Figura 2.12. Multiprogramación y multiproceso.

Placa base con 4 slots para alojar un procesador por slot, usados principalmente para tratar grandes volúmenes de datos con mucho procesamiento:



Procesador con múltiples núcleos. Son los utilizados actualmente en sistemas personales:



La técnica multithreading y SMP son frecuentemente analizados juntos, pero son dos utilidades independientes. Incluso en un nodo monoprocesador, la técnica de multihreading es útil para estructurar aplicaciones y procesos de núcleo. Una máquina SMP es útil incluso para procesos que no contienen hilos, porque varios procesos pueden ejecutar en paralelo. Sin embargo, ambas utilidades se complementan y se pueden utilizar de forma conjunta efectivamente.

Una característica atractiva de un SMP es que la existencia de múltiples procesadores es transparente al usuario. El sistema operativo se encarga de planificar los hilos o procesos en procesadores individuales y de la sincronización entre los procesadores.

Un poco de historia

UNIX como sistema tradicional

La historia de UNIX es un relato narrado con frecuencia y no se repetirá con muchos detalles aquí. Por el contrario, aquí se realizará un breve resumen. UNIX se desarrolló inicialmente en los laboratorios Bell y se hizo operacional en 1970. Algunas de las personas relacionadas con los laboratorios Bell también habían participado en otro trabajo sobre sistemas de tiempo compartido desarrollado en el proyecto MAC del MIT. Este proyecto se encargó primero del desarrollo de CTSS y después de Multics (nombre de otros sistemas operativos en desarrollo). Aunque es habitual decir que el UNIX original fue una versión recortada de Multics, los desarrolladores de UNIX realmente dijeron estar más influenciados por CTSS. No obstante, UNIX incorporó muchas ideas de Multics.

El trabajo de UNIX en los laboratorios Bell, y después en otras instituciones, produjo un conjunto de versiones de UNIX. El primer hito más notable fue portar el sistema UNIX de la computadora PDP-7 (***P**rogrammed **D**ata **P**rocessor)* a la PDP-11 (computador de 18 bits a un computador más avanzado con arquitectura Von Neumann). Ésta fue la primera pista de que UNIX sería un sistema

operativo para todos los computadores. El siguiente hito importante fue la reescritura de UNIX en el lenguaje de programación C. Ésta era una estrategia sin precedentes en ese tiempo. Se pensaba que algo tan complejo como un sistema operativo, que debe tratar eventos críticos en el tiempo, debía escribirse exclusivamente en lenguaje ensamblador. La implementación C demostró las ventajas de utilizar un lenguaje de alto nivel para la mayoría o todo el código del sistema. Hoy, prácticamente todas las implementaciones UNIX están escritas en C.

Ordenador PDP-1.

Estas primeras versiones de UNIX fueron populares dentro de los laboratorios Bell. En 1974, el sistema UNIX se describió en una revista técnica por primera vez. Esto despertó un gran interés por el sistema. Se proporcionaron licencias de UNIX tanto a instituciones comerciales como a universidades. La primera versión completamente disponible fuera de los laboratorios Bell fue la Versión 6, en 1976. La siguiente versión, la Versión 7, aparecida en 1978, es la antecesora de los sistemas UNIX más modernos. Por otro lado, el sistema más importante no vinculado con AT&T (otro importante laboratorio) y basado en UNIX se desarrolló en la Universidad de California en Berkeley, y se llamó UNIX BSD (Berkeley Software Distribution), ejecutándose primero en PDP y después en máquinas VAX (computadora sucesora de PDP-11, trabaja con 32 bits). AT&T continuó desarrollando y refinando el sistema. En 1982, los laboratorios Bell combinaron diversas variantes AT&T de UNIX en un único sistema, denominado comercialmente como UNIX System III. Un gran número de características se añadieron posteriormente al sistema operativo para producir UNIX System V. Fueron liberadas cuatro versiones de System V, denominadas Releases 1, 2, 3 y 4.

La Figura 2.14 proporciona una descripción general de la arquitectura UNIX. El hardware subyacente es gestionado por el software del sistema operativo. El sistema operativo se denomina frecuentemente el núcleo del sistema, o simplemente núcleo, para destacar su aislamiento frente a los usuarios y a las aplicaciones. Sin embargo, UNIX viene equipado con un conjunto de servicios de usuario e interfaces que se consideran parte del sistema. Estos se pueden agrupar en el shell, otro software de interfaz, y los componentes del compilador C (compilador, ensamblador, cargador). La capa externa está formada por las aplicaciones de usuario y la interfaz de usuario al compilador C.

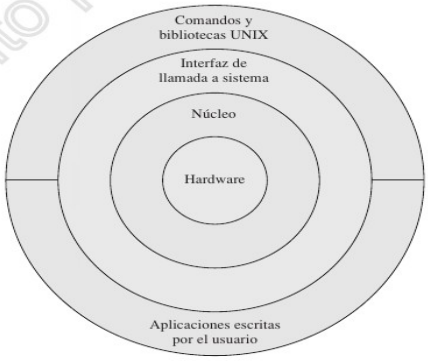


Figura 2.14. Arquitectura general de UNIX.

La Figura 2.15 muestra una vista más cercana del núcleo. Los programas de usuario pueden invocar los servicios del sistema operativo directamente o a través de programas de biblioteca. La interfaz de llamada a sistemas es la frontera con el usuario y permite que el software de alto nivel obtenga acceso a funciones específicas de núcleo. En el otro extremo, el sistema operativo contiene rutinas primitivas que interaccionan directamente con el hardware. Entre estas dos interfaces, el sistema se divide en dos partes principales, una encargada del control de procesos y la otra encargada de la gestión de ficheros y de la E/S. El subsistema de control de procesos se encarga de la gestión de memoria, la planificación y ejecución de los procesos, así como de la sincronización y la comunicación entre los procesos. El sistema de ficheros intercambia datos entre la memoria y los dispositivos externos tanto como flujos de caracteres como bloques. Para lograr esto, se utilizan una gran variedad de controladores de dispositivos. Para las transferencias orientadas a bloques, se utiliza una técnica de cache de discos: entre el espacio de direccionamiento del usuario y el dispositivo externo se interpone un buffer de sistema en memoria principal.

Hubo varias versiones Se diseñaron para ejecutar sobre un único procesador y carecen de la capacidad para proteger sus estructuras de datos del acceso concurrente por parte de múltiples procesadores (se estudiará en los próximos capítulos). Su núcleo no es muy versátil, soportando un único tipo de sistema de ficheros, una única política de planificación de procesos y un único formato de fichero ejecutable. El núcleo tradicional de UNIX no está diseñado para ser extensible y tiene pocas utilidades para la reutilización de código. El resultado es que, según se iban añadiendo nuevas características a varias versiones de UNIX, se tuvo que añadir mucho código, proporcionando un núcleo de gran tamaño y no modular.

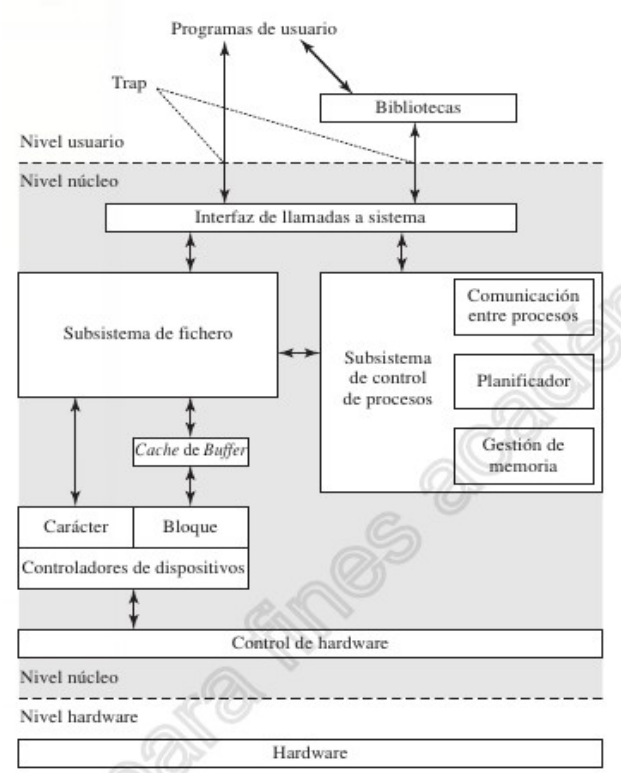


Figura 2.15. Núcleo tradicional de UNIX [BACH86].

Sistemas UNIX modernos

Cuando UNIX evolucionó, un gran número de diferentes implementaciones proliferó, cada una de las cuales proporcionó algunas características útiles. Fue necesaria la producción de una nueva implementación que unificara muchas de las importantes innovaciones, añadiera otras características de diseño de los sistemas operativos modernos, y produjera una arquitectura más modular. La arquitectura mostrada en la Figura 2.16 muestra los aspectos típicos de un núcleo

UNIX moderno. Existe un pequeño núcleo de utilidades, escritas de forma modular, que proporciona funciones y servicios necesarios para procesos del sistema operativo. Cada uno de los círculos externos representa funciones y una interfaz que podría implementarse de diferentes formas. Ahora se verán algunos ejemplos de sistemas UNIX modernos.

SYSTEM V RELEASE 4 (SVR4)

SVR4, desarrollado conjuntamente por AT&T y Sun Microsistemas, combina características de SVR3, 4.3BSD, Microsoft Xenix System y SunOS (sistemas derivados de UNIX). Fue casi una reescritura completa del núcleo del System V y produjo una implementación bien organizada, aunque compleja. Las nuevas características de esta versión incluyen soporte al procesamiento en tiempo real, clases de planificación de procesos, estructuras de datos dinámicamente asignadas, gestión de la memoria virtual, sistema de ficheros virtual y un núcleo expulsivo (en un núcleo expulsivo, cuando se desbloquea un proceso de mayor prioridad que el que está en ejecución, sólo debe esperar para ponerse a ejecutar a que concluya el tratamiento de interrupciones de los dispositivos, si los hubiera. En el caso de un núcleo no expulsivo, sin embargo, se debe esperar también a que se complete el tratamiento de interrupciones.

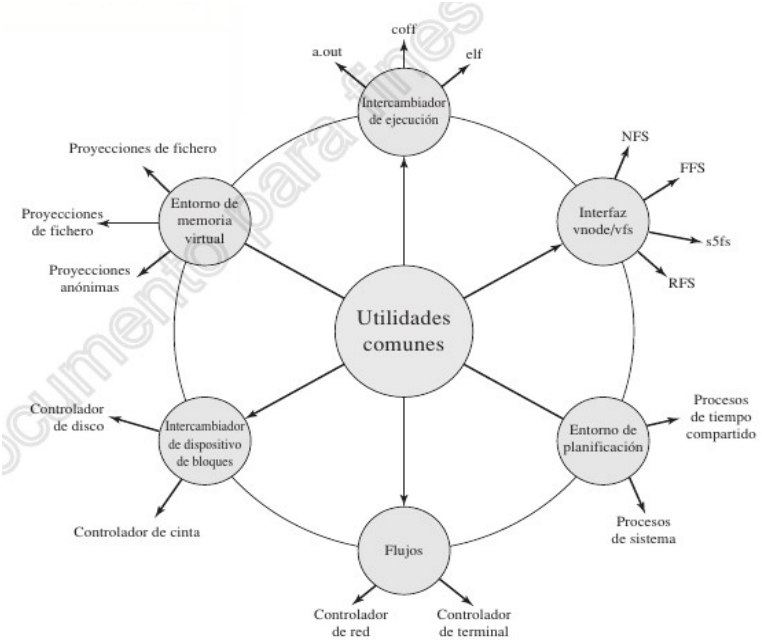


Figura 2.16. Núcleo UNIX moderno [VAHA96].

SVR4 mezcló los esfuerzos de los diseñadores comerciales y académicos y se desarrolló para proporcionar una plataforma uniforme que permitiera el despegue comercial de UNIX. Logró su objetivo y es quizá una de las variantes más importantes de UNIX. Incorpora la mayoría de las características importantes desarrolladas en cualquier sistema UNIX y lo hace de una forma integrada y comercialmente viable. SVR4 ejecuta en un gran rango de máquinas, desde los microprocesadores de 32 bits hasta los supercomputadores.

SOLARIS 9

Solaris es una versión UNIX de Sun basada en SVR4. La última versión es la 9, y proporciona todas las características de SVR4 más un conjunto de características avanzadas, como un núcleo multihilo, completamente expulsivo, con soporte completo para SMP, y una interfaz orientada a objetos para los sistemas de ficheros. Solaris es la implementación UNIX más utilizada y comercialmente más exitosa.

4.4BSD

Las series de UNIX BSD (Berkeley Software Distribution) han jugado un papel importante en el desarrollo de la teoría de diseño de los sistemas operativos. 4.xBSD se ha utilizado ampliamente en instalaciones académicas y ha servido como base de algunos productos comerciales UNIX. Es probable que BSD es seguramente responsable de gran parte de la popularidad de UNIX y que la mayoría de las mejoras de UNIX aparecieron en primer lugar en las versiones BSD.

4.4BSD fue la versión final de BSD que Berkeley produjo, disolviéndose posteriormente la organización encargada del diseño e implementación. Se trata de una actualización importante de 4.3BSD, que incluye un nuevo sistema de memoria virtual, cambios en la estructura del núcleo, y una larga lista de otras mejoras.

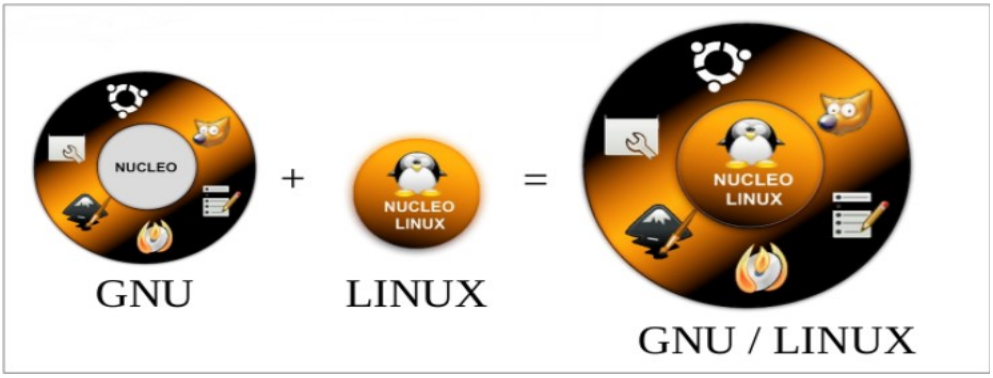
La versión del sistema operativo de Macintosh, Mac OS X, se basa en 4.4BSD.

Linux

Linux comenzó como una variante UNIX para la arquitectura del PC IBM (Intel 80386). Linus Torvalds, un estudiante finlandés de informática, escribió la versión inicial. Torvalds distribuyó por Internet una primera versión de Linux en 1991. Desde entonces, algunas personas, colaborando en Internet, han contribuido al desarrollo de Linux, todo bajo el control de Torvalds. Debido a que Linux es libre y el código fuente está disponible, se convirtió pronto en una alternativa para otras estaciones de trabajo UNIX, tal como las ofrecidas por Sun Microsystems e IBM. Hoy en día, Linux es un sistema UNIX completo que ejecuta en todas esas plataformas y algunas más, incluyendo Intel Pentium e Itanium, y el PowerPC de Motorota/IBM.

La clave del éxito de Linux ha sido:

- La disponibilidad de los paquetes de software libre bajo los auspicios de la Fundación de Software Libre (Free Software Foundation, FSF). Esta fundación se centra en un software estable, independiente de plataforma, con alta calidad, y soportado por la comunidad de usuarios.
- El proyecto de GNU (*GNU's Not Unix),* iniciado por Richard Stallman en 1983 con el objetivo de crear un sistema operativo completamente libre, el sistema GNU. Dicho proyecto desarrolló herramientas con vistas al futuro núcleo de GNU, GNU Hurd (actualmente todavía en versión beta). Torvald utilizó estas herramientas GNU para incorporárselas a su núcleo LINUX, creando el sistema GNU/LINUX. Realmente cuando en la literatura leemos LINUX, nos estamos refiriendo a GNU/LINUX, ya que Linus Torvald aportó el núcleo y Richard Stallman las aplicaciones.



Además de su uso por muchos programadores individuales, Linux ha hecho ahora una penetración significativa en el mundo corporativo. Esto no es sólo debido al software libre, sino también a la calidad del núcleo de Linux. Muchos programadores con talento han contribuido a la versión actual, dando lugar a un producto técnicamente impresionante. Más aún, Linux es muy modular y

fácilmente configurable. Resulta óptimo para incrementar el rendimiento de una variedad de plataformas hardware. Además, con el código fuente disponible, los distribuidores pueden adaptar las aplicaciones y facilidades para cumplir unos requisitos específicos.

ESTRUCTURA MODULAR

La mayoría de los núcleos Linux son monolíticos. Un núcleo monolítico es aquél que incluye prácticamente toda la funcionalidad del sistema operativo en un gran bloque de código que ejecuta como un único proceso con un único espacio de direccionamiento. Todos los componentes funcionales del núcleo tienen acceso a todas las estructuras internas de datos y rutinas. Si los cambios se hacen sobre cualquier porción de un sistema operativo monolítico, todos los módulos y rutinas deben volverse a enlazar y reinstalar, y el sistema debe ser reiniciado para que los cambios tengan efecto. Como resultado, cualquier modificación, como por ejemplo añadir un nuevo controlador de dispositivo o función del sistema de fichero, es difícil. Este problema es especialmente agudo para Linux, cuyo desarrollo es global y ha sido realizado por un grupo de programadores independientes asociados de forma difusa.

Aunque Linux no utiliza una técnica de micronúcleo, logra muchas de las ventajas potenciales de esta técnica por medio de su arquitectura modular particular. Linux está estructurado como una colección de módulos, algunos de los cuales pueden cargarse y descargarse automáticamente bajo demanda. Estos bloques relativamente independientes se denominan módulos cargables. Esencialmente, un módulo es un fichero cuyo código puede enlazarse y desenlazarse con el núcleo en tiempo real. Normalmente, un módulo implementa algunas funciones específicas, como un sistema de ficheros, un controlador de dispositivo o algunas características de la capa superior del núcleo. Un módulo no se ejecuta como su propio proceso o hilo, aunque puede crear los hilos del núcleo que necesite por varios propósitos. En su lugar, un módulo se ejecuta en modo núcleo en nombre del proceso actual.

Por tanto, aunque Linux se puede considerar monolítico, su estructura modular elimina algunas de las dificultades para desarrollar y evolucionar el núcleo. Los módulos cargables de Linux tienen dos características importantes:

- Enlace dinámico. Un módulo de núcleo puede cargarse y enlazarse al núcleo mientras el núcleo está en memoria y ejecutándose. Un módulo también puede desenlazarse y eliminarse de la memoria en cualquier momento.
- Módulos apilables. Los módulos se gestionan como una jerarquía. Los módulos individuales sirven como bibliotecas cuando los módulos cliente los referencian desde la parte superior de la jerarquía, y actúan como clientes cuando referencian a módulos de la parte inferior de la jerarquía.

El enlace dinámico facilita la configuración y reduce el uso de la memoria del núcleo. En Linux, un programa de usuario o un usuario puede cargar y descargar explícitamente módulos del núcleo utilizando los mandatos *insmod* y *rmmod*. El núcleo mismo detecta la necesidad de funciones particulares y puede cargar y descargar módulos cuando lo necesite. Con módulos apilables, se pueden definir dependencias entre los módulos. Esto tiene dos ventajas:

1. El código común para un conjunto de módulos similares (por ejemplo, controladores para hardware similar) se puede mover a un único módulo, reduciendo la replicación.
2. El núcleo puede asegurar que los módulos necesarios están presentes, impidiendo descargar un módulo del cual otros módulos que ejecutan dependen y cargando algunos módulos adicionalmente requeridos cuando se carga un nuevo módulo.

La Figura 2.17 es un ejemplo que ilustra las estructuras utilizadas por Linux para gestionar módulos. La figura muestra la lista de los módulos del núcleo que existen después de que sólo dos módulos han sido cargados: FAT y VFAT. Cada módulo se define mediante dos tablas, la tabla de

módulos y la tabla de símbolos. La tabla de módulos incluye los siguientes elementos:

- *next. Puntero al siguiente módulo. Todos los módulos se organizan en una lista enlazada. La lista comienza con un pseudomódulo (no mostrado en la Figura 2.17).
- *name. Puntero al nombre del módulo.
- size. Tamaño del módulo en páginas de memoria
- usecount. Contador del uso del módulo. El contador se incrementa cuando una operación relacionada con las funciones del módulo comienza y se decrementa cuando la operación finaliza.
- flags. Opciones del módulo.
- nsyms. Número de símbolos exportados.
- ndeps. Número de módulos referenciados.
- *syms. Puntero a la tabla de símbolos de este módulo. Esta tabla contiene punteros a nombres relacionados con direcciones de memoria que son funciones, valores escalares, estructuras, etc.
- *deps. Puntero a la lista de módulos referenciados por este módulo.
- *refs. Puntero a la lista de módulos que usa este módulo.

La tabla de símbolos define aquellos símbolos controlados por este módulo que se utilizan en otros sitios. La Figura 2.17 muestra que el módulo VFAT se carga después del módulo FAT y que el módulo VFAT es dependiente del módulo FAT.

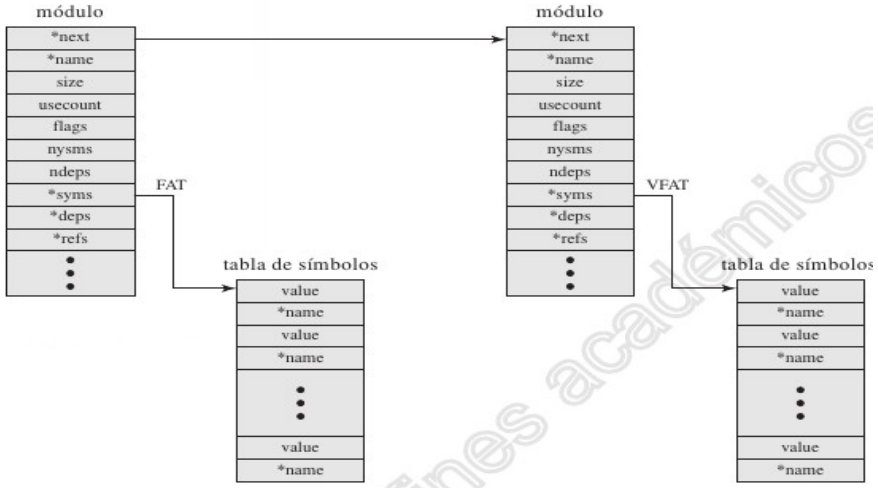


Figura 2.17. Lista ejemplo de módulos de núcleo de Linux.

COMPONENTES DEL NÚCLEO

La Figura 2.18, muestra los principales componentes del núcleo Linux. La figura muestra varios procesos ejecutando encima del núcleo. Cada caja indica un proceso separado, mientras que cada línea curvada con una cabeza de flecha representa un hilo de ejecución (en Linux, no hay distinción entre los conceptos de proceso e hilo. Sin embargo, múltiples hilos en Linux se pueden agrupar de tal forma que, efectivamente, pueda existir un único proceso compuesto por múltiples hilos).

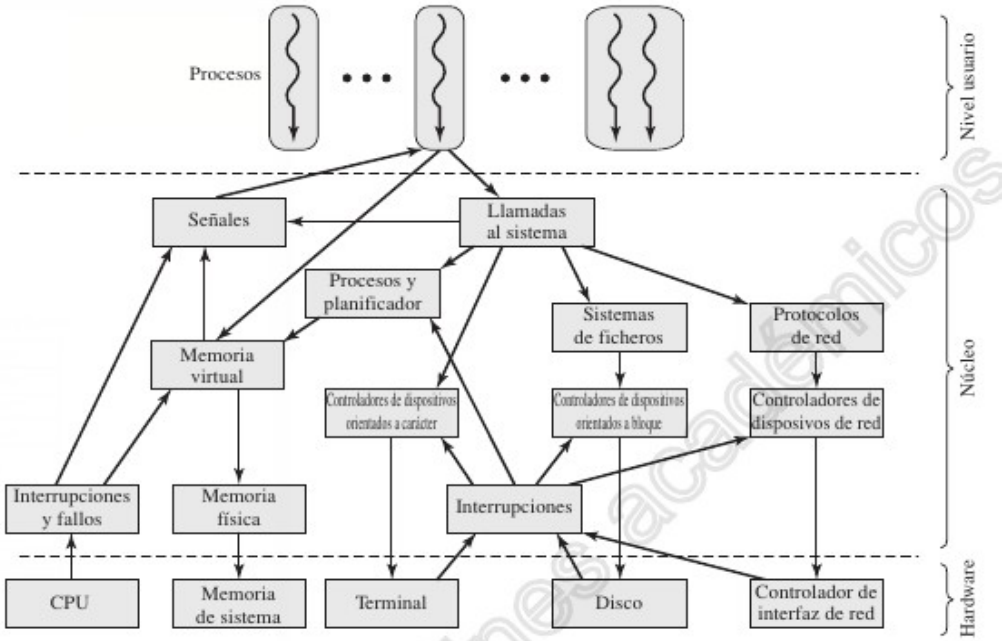


Figura 2.18. Componentes del núcleo de Linux.

El núcleo mismo está compuesto por una colección de componentes que interaccionan, usando flechas para indicar las principales interacciones. También se muestra el hardware subyacente como un conjunto de componentes utilizando flechas para indicar qué componentes del núcleo utilizan o controlan qué componentes del hardware. Todos los componentes del núcleo, por supuesto, ejecutan en la CPU, pero por simplicidad no se muestran estas relaciones.

Brevemente, los principales componentes del núcleo son los siguientes:

- **Señales**. El núcleo utiliza las señales para llamar a un proceso. Por ejemplo, las señales se utilizan para notificar ciertos fallos a un proceso como por ejemplo, la división por cero. La Tabla 2.6 da unos pocos ejemplos de señales.

Tabla 2.6. Algunas señales de Linux.

SIGHUP	Desconexión de un terminal	SIGCONT	Continuar
SIGQUIT	Finalización por teclado	SIGTSTP	Parada por teclado
SIGTRAP	Traza	SIGTTOU	Escritura de terminal
SIGBUS	Error de bus	SIGXCPU	Límite de CPU excedido
SIGKILL	Señal para matar	SIGVTALRM	Reloj de alarma virtual
SIGSEGV	Violación de segmentación	SIGWINCH	Cambio de tamaño de una ventana
SIGPIPE	Tubería rota	SIGPWR	Fallo de potencia
SIGTERM	Terminación	SIGRTMIN	Primera señal de tiempo real
SIGCHLD	Cambio en el estado del hijo	SIGRTMAX	Última señal de tiempo real

- **Llamadas al sistema**. La llamada al sistema es la forma en la cual un proceso requiere un servicio de núcleo específico. Hay varios cientos de llamadas al sistema, que pueden agruparse básicamente en seis categorías: sistema de ficheros, proceso, planificación, comunicación entre procesos, socket (red) y misceláneos. La Tabla 2.7 define unos pocos ejemplos de cada categoría.
- **Procesos y planificador.** Crea, gestiona y planifica procesos.
- **Memoria virtual**. Asigna y gestiona la memoria virtual para los procesos.
- **Sistemas de ficheros.** Proporciona un espacio de nombres global y jerárquico para los ficheros, directorios y otros objetos relacionados con los ficheros. Además, proporciona las funciones del sistema de ficheros.
- **Protocolos de red.** Da soporte a la interfaz Socket para los usuarios, utilizando la pila de

protocolos TCP/IP.

- **Controladores de dispositivo tipo carácter.** Gestiona los dispositivos que requiere el núcleo para enviar o recibir datos un byte cada vez, como los terminales, los módems y las impresoras.
- **Controladores de dispositivo tipo bloque.** Gestiona dispositivos que leen y escriben datos en bloques, tal como varias formas de memoria secundaria (discos magnéticos, CDROM, etc.).
- **Controladores de dispositivo de red.** Gestiona las tarjetas de interfaz de red y los puertos de comunicación que permiten las conexiones a la red, tal como los puentes y los encaminadores.
- **Traps y fallos.** Gestiona los traps y fallos generados por la CPU, como los fallos de memoria.
- **Memoria física**. Gestiona el conjunto de marcos de páginas de memoria real y asigna las páginas de memoria virtual.
- **Interrupciones.** Gestiona las interrupciones de los dispositivos periféricos.

Tabla 2.7. Algunas llamadas al sistema de Linux.

	Relacionadas con el sistema de ficheros
close	Cierra un descriptor de fichero.
link	Construye un nuevo nombre para un fichero.
open	Abre y posiblemente crea un fichero o dispositivo.
read	Lee un descriptor de fichero.
write	Escribe a través de un descriptor de fichero.
	Relacionadas con los procesos
execve	Ejecuta un programa.
exit	Termina el proceso que lo invoca.
getpid	Obtiene la identificación del proceso.
setuid	Establece la identidad del usuario del proceso actual.
prtrace	Proporciona una forma por la cual un proceso padre puede observar y controlar la ejecución de otro proceso, examinar y cambiar su imagen de memoria y los registros.
	Relacionadas con la planificación
sched_getparam	Establece los parámetros de planificación asociados con la política de planificación para el proceso identificado por su *pid*.
sched_get_priority_max	Devuelve el valor máximo de prioridad que se puede utilizar con el algoritmo de planificación identificado por la política.
sched_setscheduler	Establece tanto la política de planificación (por ejemplo, FIFO) como los parámetros asociados al *pid* del proceso.
sched_rr_get_interval	Escribe en la estructura *timespec* apuntada por el parámetro *tp* el cuanto de tiempo *round robin* para el proceso *pid*.
sched_yield	Un proceso puede abandonar el procesador voluntariamente sin necesidad de bloquearse a través de una llamada al sistema. El proceso entonces se moverá al final de la cola por su prioridad estática y un nuevo proceso se pondrá en ejecución.
	Relacionadas con la comunicación entre procesos (IPC)
msgrcv	Se asigna una estructura de *buffer* de mensajes para recibir un mensaje. Entonces, la llamada al sistema lee un mensaje de la cola de mensajes especificada por *msqid* en el *buffer* de mensajes nuevamente creado.
semctl	Lleva a cabo la operación de control especificada por *cmd* en el conjunto de semáforos *semid*.
semop	Lleva a cabo operaciones en determinados miembros del conjunto de semáforos *semid*.
shmat	Adjunta el segmento de memoria compartido identificado por *shmid* al segmento de datos del proceso que lo invoca.
shmctl	Permite al usuario recibir información sobre un segmento de memoria compartido, establecer el propietario, grupo y permisos de un segmento de memoria compartido o destruir un segmento.
	Relacionadas con los *sockets* (red)
bind	Asigna la dirección IP local y puerto para un *socket*. Devuelve 0 en caso de éxito y -1 en caso de error.
connect	Establece una conexión entre el *socket* dado y el *socket* asociado remoto con *sockaddr*.
gethostname	Devuelve el nombre de máquina local.
send	Envía los bytes que tiene el *buffer* apuntado por **msg* sobre el *socket* dado.
setsockopt	Envía las opciones sobre un *socket*.
	Misceláneos
create_module	Intenta crear una entrada del módulo cargable y reservar la memoria de núcleo que será necesario para contener el módulo.
fsync	Copia todas las partes en memoria de un fichero a un disco y espera hasta que el dispositivo informa que todas las partes están en almacenamiento estable.
query_module	Solicita información relacionada con los módulos cargables desde el núcleo.
time	Devuelve el tiempo en segundos desde 1 de enero de 1970.
vhangup	Simula la suspensión del terminal actual. Esta llamada sirve para que otros usuarios puedan tener un terminal «limpio» en tiempo de inicio.