1.2 ■ Sistemas por lotes sencillos

Los primeros computadores eran máquinas enormes (físicamente) que se controlaban desde una consola. Los dispositivos de entrada comunes eran lectores de tarjetas y unidades de cinta. Los dispositivos de salida comunes eran impresoras de líneas, unidades de cinta y perforadoras de tarjetas. Los usuarios de tales sistemas no interactuaban directamente con los computadores; más bien, el usuario preparaba un trabajo —que consistía en el programa, los datos, y cierta información de control acerca de la naturaleza del trabajo (tarjetas de control)— y lo entregaba al operador del computador. El trabajo casi siempre se presentaba en forma de tarjetas perforadas. En algún momento posterior (minutos, horas o días después), aparecía la salida, que consistía en el resultado del programa junto con un vaciado (o vuelco) de la memoria y los registros en caso de haber un error de programa.

El sistema operativo en estos primeros computadores era sencillo. Su principal obligación era transferir el control automáticamente de un trabajo al siguiente. El sistema operativo siempre estaba (residente) en la memoria (Fig. 1.2).

A fin de agilizar el procesamiento, los programas con necesidades similares se agrupaban en *lotes* y se introducían en el computador como un grupo. Así, los programadores dejaban sus programas con el operador; éste acomodaba los programas para formar lotes con necesidades similares y, cuando el computador quedaba disponible, ejecutaba cada lote. La salida de cada trabajo se devolvía al programador apropiado.

Así pues, un sistema operativo por lotes normalmente lee un flujo de trabajos individuales (de un lector de tarjetas, por ejemplo), cada uno con sus propias tarjetas de control que predefinen lo que el trabajo hace. Una vez terminado el trabajo, su salida generalmente se imprime (en una impresora de líneas, por ejemplo). La característica definitiva de un sistema por lotes es la *falta* de interacción entre el usuario y el trabajo mientras éste se ejecuta. El trabajo se prepara y se entrega, y cierto tiempo después aparece la salida. El retardo entre la presentación de un trabajo y su terminación (llamado tiempo de *retorno*) puede ser el resultado del

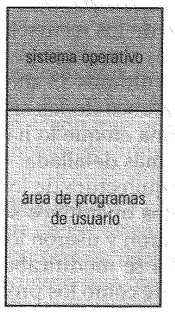

Figura 1.2 Organización de la memoria para un sistema por lotes sencillo.

volumen de cómputo requerido, o de retardos antes de que el sistema operativo comience a procesar el trabajo.

En este entorno de ejecución, la CPU con frecuencia está ociosa. Esta inactividad ocurre porque los dispositivos de E/S mecánicos son intrínsecamente más lentos que los dispositivos electrónicos. Incluso una CPU lenta realiza sus operaciones en tiempos del orden de microsegundos, y ejecuta miles de instrucciones cada segundo. Un lector de tarjetas rápido, en cambio, podría leer 1200 tarjetas por minuto (20 tarjetas por segundo). Así, la diferencia de velocidad entre la CPU y sus dispositivos de E/S puede ser de tres órdenes de magnitud o más. Con el paso del tiempo, claro, las mejoras a la tecnología dieron pie a dispositivos de E/S más rápidos. Desafortunadamente, las velocidades de CPU aumentaron a un ritmo aún mayor, de modo que el problema no sólo no se resolvió, sino que se exacerbó.

La introducción de la tecnología de discos ha sido útil en este sentido. En vez de leer las tarjetas directamente del lector a la memoria, para después procesar el trabajo, las tarjetas pueden leerse del lector al disco. La ubicación de las imágenes de tarjeta se registra en una tabla mantenida por el sistema operativo. Cuando se ejecuta un trabajo, el sistema operativo satisface sus solicitudes de entradas del lector de tarjetas leyendo del disco. Así mismo, cuando el trabajo solicita la impresión de una línea, esa línea se copia en un *buffer* del sistema y se escribe en el disco. Una vez que el trabajo termina, la salida se imprime realmente. Esta forma de procesamiento, se llama *spooling* (Fig. 1.3); el nombre es un acrónimo de operación periférica simultánea en línea (en inglés, *simultaneous peripheral operation on-line*). En esencia, el disco se utiliza como *buffer* de gran tamaño, para leer por adelantado hasta donde sea posible de los dispositivos de entrada, y para guardar los archivos de salida hasta que los dispositivos de salida puedan aceptarlos.

El *spooling* también sirve para procesar datos en sitios remotos. La CPU envía los datos por líneas de comunicación a una impresora remota (o acepta todo un trabajo de entrada de un lector de tarjetas remoto). El procesamiento remoto se realiza a

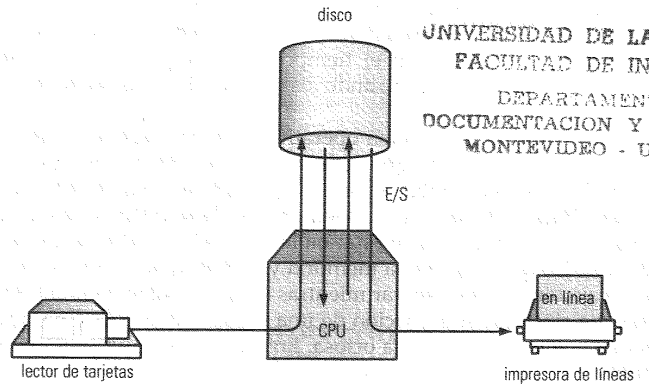

Figura 1.3 Spooling.

su propio ritmo, sin intervención de la CPU. Sólo es necesario notificar a la CPU cuando el procesamiento llega a su término, para que pueda poner en *spool* el siguiente lote de datos.

El *spooling* superpone la E/S de un trabajo al cómputo de otros trabajos. Incluso en un sistema sencillo, el *spooler* podría estar leyendo la entrada de un trabajo mientras imprime la salida de otro. Durante este tiempo, podrían estarse ejecutando uno o más trabajos distintos, leyendo sus "tarjetas" del disco e "imprimiendo" sus líneas de salida en el disco.

El *spooling* tiene un efecto benéfico directo sobre el desempeño del sistema. Por el costo de un poco de espacio en disco y de unas cuantas tablas, el cómputo de un trabajo se puede superponer a la E/S de otros trabajos. Así, el uso de *spool* puede mantener tanto la CPU como los dispositivos de E/S trabajando con un rendimiento mucho mayor.

1.3 ■ Sistemas por lotes multiprogramados

El *spooling* da origen a una importante estructura de datos: la *reserva de trabajos*. Como resultado del *spooling*, puede haber varios trabajos ya leídos esperando en el disco, listos para ejecutarse. Al tener reserva de trabajos en disco, el sistema operativo puede escoger cuál trabajo ejecutará a continuación, a fin de mejorar el aprovechamiento de la CPU. Si los trabajos llegan directamente en tarjetas o incluso en cinta magnética, no es posible ejecutarlos en un orden distinto. Los trabajos se deben ejecutar secuencialmente, bajo un régimen de servicio por orden de llegada. En cambio, si varios trabajos están en un dispositivo de acceso directo, como un disco, es posible la *planificación de trabajos*. Estudiaremos la planificación de trabajos y de la CPU con detalle en el capítulo 5; aquí sólo mencionaremos unos cuantos aspectos importantes.

El aspecto más importante de la planificación de trabajos es la capacidad de *multiprogramar*. La operación fuera de línea y el *spooling* para traslapar la E/S tienen sus limitaciones. En general, un solo usuario no puede mantener ni la CPU ni los dispositivos de E/S ocupados todo el tiempo. La multiprogramación aumenta el aprovechamiento de la CPU organizando los trabajos de tal forma que la CPU siempre tenga uno que ejecutar.

La idea es la siguiente. El sistema operativo mantiene varios trabajos en la memoria a la vez (Fig. 1.4). Este conjunto de trabajos es un subconjunto de los que se mantienen en la reserva de trabajos (puesto que el número de trabajos que se pueden mantener simultáneamente en la memoria generalmente es mucho menor que el número de trabajos que pueden estar en la reserva). El sistema operativo escoge uno de los trabajos que están en la memoria y comienza a ejecutarlo. Tarde o temprano, el trabajo tendrá que esperar mientras se lleva a cabo alguna tarea, como el montaje de una cinta o la terminación de una operación de E/S. En un sistema sin multiprogramación, la CPU estaría ociosa. En un sistema multiprogramado, el sistema operativo simplemente selecciona otro trabajo y lo ejecuta. Cuando *ese* trabajo necesita esperar, la CPU se conmuta a otro trabajo, y así sucesivamente. En algún

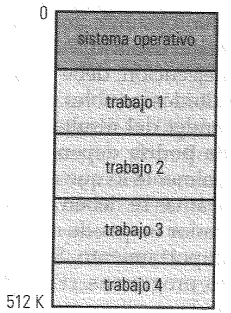

Figura 1.4 Organización de la memoria en un sistema multiprogramado.

momento, el primer trabajo terminará su espera y recuperará la CPU. En tanto haya algún trabajo que ejecutar, la CPU nunca estará ociosa.

Esta idea es común a otras situaciones de la vida real. Un abogado no tiene un solo cliente a la vez. Más bien, varios clientes pueden estar siendo atendidos al mismo tiempo. Mientras un caso está esperando ir a juicio o la preparación de documentos, el abogado puede trabajar en otro caso. Si tiene suficientes clientes, el abogado nunca tendrá que estar ocioso. (Los abogados ociosos tienden a convertirse en políticos, por lo que mantener a los abogados ocupados tiene cierto valor social.)

La multiprogramación es el primer caso en que el sistema operativo debe tomar decisiones por los usuarios. Por ello, los sistemas operativos multiprogramados son relativamente complejos. Todos los trabajos que ingresan en el sistema se mantienen en un área auxiliar de trabajos. Esta área auxiliar consiste en todos los procesos que residen en el almacenamiento masivo y que esperan la asignación de espacio en la memoria principal. Si varios trabajos están listos para colocarse en la memoria, y no hay espacio suficiente para todos, el sistema deberá escoger algunos de entre ellos. La toma de esta decisión es la *planificación de trabajos*, que se estudiará en el capítulo 5. Cuando el sistema operativo escoge un trabajo del área auxiliar, lo carga en la memoria para ejecutarlo. Tener varios programas en la memoria al mismo tiempo requiere alguna forma de gestión de memoria, tema que se trata en los capítulos 8 y 9. Además, si varios trabajos están listos para ejecutarse al mismo tiempo, el sistema debe escoger entre ellos. La toma de esta decisión es la *planificación de CPU*, que se verá en el capítulo 5. Por último, la ejecución concurrente de múltiples trabajos requiere limitar su capacidad para afectarse mutuamente en todas las fases del sistema operativo, incluidos la planificación de procesos, el almacenamiento en disco y la gestión de memoria. Analizaremos estas consideraciones a todo lo largo del texto.

1.4 ■ Sistemas de tiempo compartido

Los sistemas por lotes multiprogramados proporcionan un entorno en el que los distintos recursos del sistema (por ejemplo, CPU, memoria, dispositivos periféricos) se

aprovechan de manera efectiva. No obstante, los sistemas por lotes tienen ciertos problemas desde el punto de vista del usuario. Puesto que el usuario no puede interactuar con el trabajo durante su ejecución, debe preparar las tarjetas de control de modo que manejen todos los resultados posibles. En un trabajo de varios pasos, los pasos subsecuentes podrían depender del resultado de pasos anteriores. Por ejemplo, la ejecución de un programa podría depender de que su compilación tenga éxito. Puede ser difícil definir cabalmente lo que debe hacerse en todos los casos.

Otro problema es que los programas se deben depurar estáticamente, a partir de vuelcos instantáneos. El programador no puede modificar un programa durante su ejecución a fin de estudiar su comportamiento. Si el tiempo de retorno es largo, se inhibe la experimentación con los programas. (Por otro lado, esta situación puede obligar a tener cierta disciplina en la escritura y prueba de los programas.)

El tiempo compartido, o *multitareas*, es una extensión lógica de la multiprogramación. Se ejecutan múltiples trabajos mientras la CPU se conmuta entre ellos, pero la conmutación es tan frecuente que los usuarios pueden interactuar con cada programa durante su ejecución.

Un sistema de computador *interactivo*, o *manual*, permite la comunicación en línea entre el usuario y el sistema. El usuario da instrucciones al sistema operativo o a un programa directamente, y recibe una respuesta inmediata. Por lo regular, se utiliza el teclado para proporcionar entradas, y las salidas se envían a una pantalla —como un tubo de rayos catódicos (CRT, *cathode ray tube*) o monitor—. Cuando el sistema operativo termina de ejecutar una orden, busca el siguiente "enunciado de control" no de un lector de tarjetas, sino del teclado del usuario. El usuario introduce una orden, espera la respuesta, y decide cuál será la siguiente orden, con base en el resultado de la anterior. Para el usuario es fácil experimentar, y puede ver los resultados de inmediato. La mayor parte de los sistemas cuenta con un editor de textos interactivo para introducir programas, y un depurador interactivo para ayudar a corregir los errores de los programas.

Para que los usuarios puedan acceder con comodidad tanto a los datos como al código, deben contar con un *sistema de archivos en línea*. Un *archivo* es una colección de información relacionada definida por su creador. Por lo regular, los archivos representan programas (en sus formas tanto fuente como objeto) y datos. Los archivos de datos pueden ser numéricos, alfabéticos o alfanuméricos. Los archivos pueden ser de forma libre, como los de texto, o tener un formato rígido. En general, un archivo es una secuencia de bits, bytes, líneas o registros cuyo significado ha sido definido por su creador y usuario. El sistema operativo implementa el concepto abstracto de archivo administrando los dispositivos de almacenamiento masivo, como cintas y discos. Los archivos normalmente se organizan en grupos lógicos, o *directorios*, que facilitan su localización y acceso. Puesto que varios usuarios tienen acceso a archivos, es deseable controlar quién puede acceder a los archivos y de qué formas puede hacerlo.

Los sistemas por lote son apropiados para ejecutar trabajos grandes que casi no necesitan interacción. El usuario puede presentar trabajos y regresar después por los resultados; no es necesario que el usuario espere mientras se procesa el trabajo. Los trabajos interactivos suelen consistir en muchas acciones cortas, y los resultados de la siguiente orden podrían ser impredecibles. El usuario introduce la orden y espera los resultados. Por consiguiente, el tiempo de *respuesta* debe ser corto; del orden de segundos cuando más. Se emplea un sistema interactivo cuando se requiere un tiempo de respuesta corto.

Los primeros computadores de un solo usuario fueron sistemas interactivos. Esto es, todo el sistema estaba a la inmediata disposición del programador/operador. Esta situación brindaba al programador gran flexibilidad y libertad para probar y desarrollar programas. No obstante, como hemos visto, este sistema hacía que la CPU permaneciera mucho tiempo ociosa mientras esperaba que el programador/operador hiciera algo. En vista del elevado costo de esos computadores, la ociosidad de la CPU era muy indeseable. Se desarrollaron los sistemas operativos por lotes para evitar este problema. Los sistemas por lotes mejoraron el aprovechamiento de los sistemas por parte de sus dueños.

Los sistemas de *tiempo compartido* se crearon para brindar el uso interactivo de un sistema de computador a un costo razonable. Un sistema operativo de tiempo compartido utiliza planificación de la CPU y multiprogramación para ofrecer a cada usuario una pequeña porción del tiempo de un computador. Cada usuario tiene por lo menos un programa individual en la memoria. Un programa que está cargado en la memoria y se está ejecutando se conoce como *proceso*. Cuando un proceso se ejecuta, generalmente lo hace sólo durante un tiempo corto antes de que termine o necesite realizar operaciones de E/S. La E/S puede ser interactiva: las salidas podrían enviarse a una pantalla para que el usuario las vea, y las entradas podrían recibirse del teclado del usuario. Puesto que la E/S interactiva casi siempre se efectúa a un ritmo humano, puede tardar mucho tiempo en llevarse a cabo. Las entradas, por ejemplo, pueden estar limitadas por la velocidad con que el usuario teclea; consideramos que una persona que teclea a razón de cinco caracteres por segundo es rápida, pero es increíblemente lenta para los computadores. En lugar de dejar que la CPU esté ociosa mientras ocurre tal entrada interactiva, el sistema operativo conmuta rápidamente la CPU al programa de algún otro usuario.

Un sistema operativo de tiempo compartido permite a los múltiples usuarios *compartir* el computador simultáneamente. Puesto que cada acción u orden en un sistema de tiempo compartido tiende a ser corta, cada usuario necesita sólo un poco de tiempo de CPU. Como el computador cambia con gran rapidez de un usuario al siguiente, cada uno recibe la impresión de que tiene su propio computador, aunque en realidad muchos usuarios lo estén compartiendo.

La idea del tiempo compartido se demostró ya en 1960, pero dada la dificultad y el costo de construir tales sistemas, apenas se hicieron comunes a principios de la década de 1970. A medida que ha crecido la popularidad del tiempo compartido, los investigadores han intentado fusionar los sistemas por lotes y de tiempo compartido. Muchos sistemas de computador que se diseñaron primordialmente como sistemas por lotes se han modificado para crear un subsistema de tiempo compartido. Por ejemplo, el OS/360 de IBM, un sistema por lotes, se modificó para apoyar la opción de tiempo compartido (TSO, *Time-Sharing Option*). Al mismo tiempo, son varios los casos de sistemas de tiempo compartido a los que se ha añadido un subsistema por lotes. Hoy día, la mayor parte de los sistemas ofrecen tanto procesamiento por lotes como tiempo compartido, aunque su diseño y uso básico tiende a ser de un tipo o del otro.

Los sistemas operativos de tiempo compartido son todavía más complejos que los multiprogramados. Al igual que en la multiprogramación, es preciso mantener varios trabajos simultáneamente en la memoria, lo que requiere alguna forma de gestión y protección de memoria (Cap. 8). Para poder lograr un tiempo de respuesta razonable, podría ser necesario intercambiar trabajos entre la memoria principal y el disco que ahora funciona como almacén de respaldo de la memoria principal. Un método común para lograr este objetivo es la *memoria virtual*, una técnica que permite ejecutar un trabajo que tal vez no está en su totalidad en la memoria principal (Cap. 9). La principal ventaja obvia de este esquema es que los programas pueden ser más grandes que la memoria física. Además, la memoria principal se abstrae como una matriz grande y uniforme de almacenamiento, separando la memoria lógica que el usuario ve, de la memoria física. Esta organización permite a los programadores olvidarse de las limitaciones de almacenamiento en memoria. Los sistemas de tiempo compartido también necesitan un sistema de archivos en línea (Caps. 10 y 11). El sistema de archivos reside en una colección de discos, así que también debe incluirse la gestión de discos (Cap. 13). Además, los sistemas de tiempo compartido cuentan con un mecanismo de ejecución concurrente, lo que requiere esquemas de planificación de la CPU avanzados (Cap. 5). Para asegurar una ejecución ordenada, el sistema debe incluir mecanismos para la sincronización y comunicación de los trabajos (Cap. 6), y debe asegurar que los trabajos no se atasquen en un bloqueo mutuo, esperando cada uno eternamente a que el otro termine (Cap. 7).

La multiprogramación y el tiempo compartido son los temas centrales de los sistemas operativos modernos, y por tanto de este libro.

1.5 ■ Sistemas de computador personal

Con la caída en los costos del hardware, ha vuelto a ser factible tener un sistema de computación dedicado a un solo usuario. Estos tipos de sistemas se conocen como *computadores personales* (PC, *personal computer*). No hay duda de que los dispositivos de E/S han cambiado: los tableros de interruptores y los lectores de tarjetas han sido reemplazados por teclados tipo máquina de escribir y ratones. Las impresoras de líneas y perforadoras de tarjetas han cedido el paso a pantallas e impresoras pequeñas y rápidas.

Los computadores personales aparecieron en la década de 1970. Se trata de microcomputadores mucho más pequeños y económicos que los sistemas de macrocomputador (*mainframes*). Durante su primera década, las CPU de los PC carecían de las funciones necesarias para proteger el sistema operativo de los programas de usuario. Por ello, los sistemas operativos de PC no eran ni multiusuario ni multitareas. No obstante, los objetivos de estos sistemas operativos han cambiado con el tiempo; en lugar de maximizar el aprovechamiento de la CPU y los periféricos, los sistemas optan por maximizar la comodidad del usuario y la rapidez con que responden a sus necesidades. Estos sistemas incluyen los PC que ejecutan Microsoft Windows, y el Apple Macintosh. El sistema operativo MS-DOS de Microsoft ha sido supeditado por los diversos sabores de Microsoft Windows, e IBM ha modernizado el MS-DOS para crear el sistema multitareas OS/2. El sistema

operativo de Apple Macintosh ha sido transportado a un hardware más avanzado, y ahora incluye características nuevas como memoria virtual.

Los sistemas operativos para estos computadores se han beneficiado con el desarrollo de sistemas operativos para macrocomputadores de varias maneras. Los microcomputadores pudieron adoptar de inmediato la tecnología desarrollada para sistemas operativos más grandes. Por un lado, los costos del hardware de microcomputador son lo bastante bajos como para que individuos utilicen exclusivamente el computador, y el aprovechamiento de la CPU ha dejado de ser una preocupación central. Por ello, algunas de las decisiones de diseño que se toman en los sistemas operativos para macrocomputadores podrían no ser apropiadas para sistemas más pequeños. Por ejemplo, la protección de archivos podría no ser necesaria en una máquina personal.

Algunas personas han argumentado que la aparición de microprocesadores y memoria de bajo costo harán a los sistemas operativos (y a los cursos que los tratan) obsoletos. Los autores no creen que tal predicción llegue a cumplirse. Más bien, la reducción en los costos del hardware permitirá implementar conceptos de sistemas operativos relativamente avanzados (como el tiempo compartido y la memoria virtual) en un número aún mayor de sistemas. Así, la disminución en el costo del hardware de computador, como los microprocesadores, hará que sea más necesario entender los conceptos de sistemas operativos.

Por ejemplo, aunque aparentemente la protección de archivos no es necesaria en computadores personales aislados, es común que estos computadores se vinculen con otros a través de líneas telefónicas o redes de área local. Si otros computadores y otros usuarios pueden acceder a los archivos de un computador personal, la protección de archivos vuelve a ser una función necesaria de un sistema operativo. La carencia de tal protección ha facilitado la destrucción de datos en sistemas como MS-DOS y el sistema operativo Macintosh por parte de programas mal intencionados. Estos programas pueden ser del tipo que se reproduce a sí mismo, y difundirse rápidamente por mecanismos de *gusano* o *virus* para dañar compañías enteras o incluso redes mundiales. La protección y la seguridad se considerarán en los capítulos 19 y 20.

De hecho, si examinamos los sistemas operativos para macro y microcomputadores veremos que funciones que en otras épocas eran exclusivas de los macrocomputadores han sido adoptadas por los microcomputadores. Los mismos conceptos son apropiados para las diferentes clases de computadores: macro, mini y micro (Fig. 1.5).

Un buen ejemplo de esta tendencia ocurrió con el sistema operativo MULTICS. MULTICS se desarrolló entre 1965 y 1970 en el Massachusetts Institute of Technology (MIT) como *utilería* de cómputo; se ejecutaba un macrocomputador grande y complejo (el GE 645). Muchas de las ideas que se desarrollaron para MULTICS se utilizaron posteriormente en los Bell Laboratories (uno de los socios originales en el desarrollo de MULTICS) en el diseño de UNIX. El sistema operativo UNIX se diseñó alrededor de 1970 para un minicomputador PDP-11. Para 1980, las características de UNIX se convirtieron en la base para los sistemas operativos tipo UNIX de sistemas de microcomputador, y se están incluyendo en sistemas operativos más recientes como Microsoft Windows NT, IBM OS/2, y el Macintosh Operating System. Así, con el tiempo, las funciones creadas para un sistema de macrocomputador grande han pasado a los microcomputadores.

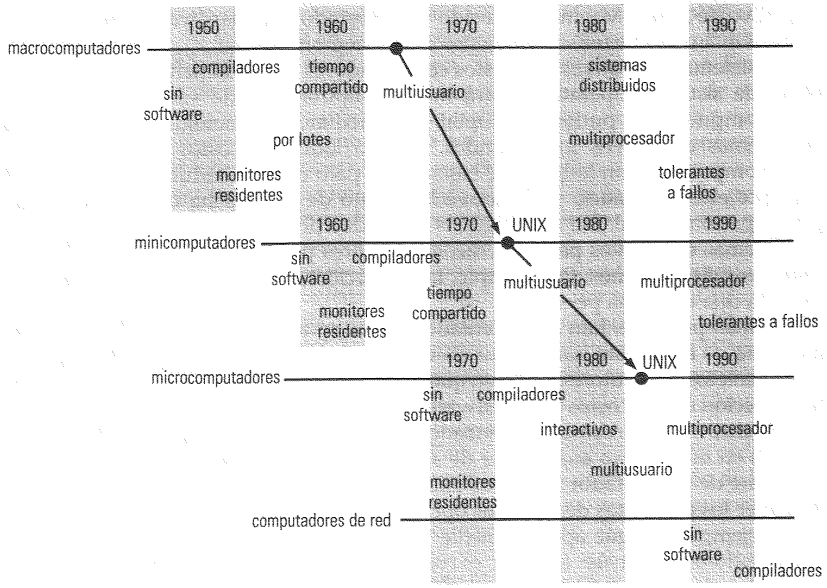

Figura 1.5 Migración de conceptos y funciones de sistemas operativos.

Al mismo tiempo que la escala de las características de los sistemas operativos grandes se estaba reduciendo para ajustarlas a los computadores personales, se estaban desarrollando sistemas de hardware más potentes, rápidos y complejos. La *estación de trabajo personal* es un computador personal grande, como el Sun, el HP/Apollo o el IBM RS/6000. Muchas universidades y empresas tienen un gran número de estaciones de trabajo vinculadas por medio de redes de área local. A medida que el hardware y el software de los PC se hace más complejo, la línea que divide las dos categorías se está desvaneciendo.

1.6 ■ Sistemas paralelos

La mayor parte de los sistemas hasta ahora han sido sistemas monoprocesador; es decir, han tenido una sola CPU. Sin embargo, la tendencia actual es hacia los *sistemas multiprocesador*. Tales sistemas tienen más de un procesador en comunicación íntima, los cuales comparten el *bus* del computador, el reloj y a veces la memoria y los dispositivos periféricos. Decimos que estos sistemas están *fuertemente acoplados*.

Hay varias razones para construir este tipo de sistemas. Una ventaja es el aumento en el *rendimiento*. Al incrementar el número de procesadores, esperamos realizar más trabajo en un tiempo más corto. Sin embargo, la proporción de aumento de la velocidad con n procesadores no es n, sino más bien menor que n. Cuando varios

procesadores cooperan para llevar a cabo una tarea, se incurre en cierto "gasto extra" para mantener todos los componentes funcionando correctamente. Este gasto extra, aunado a la contención por los recursos compartidos, reduce la ganancia que cabría esperar de los procesadores adicionales. De forma análoga, un grupo de n programadores que trabajan en íntima colaboración no produce n veces la cantidad de trabajo que realiza uno solo.

Los multiprocesadores también pueden ahorrar dinero en comparación con varios sistemas monoprocesador porque los procesadores pueden compartir periféricos, gabinetes y fuentes de potencia. Si varios programas deben operar con el mismo conjunto de datos, es más económico guardar esos datos en un disco y hacer que todos los procesadores los compartan, en vez de tener muchos computadores con discos locales y hacer copias de los datos.

Otra razón para tener sistemas multiprocesador es que mejoran la confiabilidad. Si es posible distribuir las funciones correctamente entre varios procesadores, el fallo de un procesador no detendrá el sistema, sólo lo hará más lento. Si tenemos 10 procesadores y uno falla, cada uno de los nueve procesadores restantes deberá asumir una porción del trabajo del que falló. Así, el sistema operará al 90% de su velocidad normal, en lugar de pararse. Esta capacidad para seguir dando un servicio proporcional al nivel de hardware que sobrevive se denomina *degradación gradual*. Los sistemas diseñados para *degradarse gradualmente* también se conocen como *tolerantes a fallos* (*fault-tolerant*).

El funcionamiento continuado en presencia de fallos requiere un mecanismo para detectar, diagnosticar y corregir (si es posible) el fallo. El sistema Tandem utiliza duplicación tanto de hardware como de software para asegurar una operación continuada a pesar de los fallos. El sistema consiste en dos procesadores idénticos, cada uno con su propia memoria local. Los procesadores se conectan con un *bus*. Un procesador es el primario, y el otro es el de respaldo. Se mantienen dos copias de cada proceso; una en la máquina primaria y otra en el respaldo. En puntos de verificación (*checkpoints*) fijos durante la ejecución del sistema, la información de estado de cada trabajo (incluida una copia de la imagen de memoria) se copia de la máquina primaria a la de respaldo. Si se detecta un fallo, se activa la copia de respaldo, y se reinicia a partir del punto de verificación más reciente. Esta solución obviamente es costosa, ya que hay mucha duplicación de hardware.

Los sistemas de múltiple procesador más comunes en la actualidad siguen el modelo de *multiprocesamiento simétrico*, en el que cada procesador ejecuta una copia idéntica del sistema operativo, y estas copias se comunican entre sí cuando es necesario. Algunos sistemas utilizan *multiprocesamiento asimétrico*, en el que a cada procesador se asigna una tarea específica. Un procesador maestro controla el sistema; los demás procesadores obtienen sus instrucciones del maestro o bien tienen tareas predefinidas. Este esquema define una relación maestro-esclavo. El procesador maestro planifica y asigna trabajo a los procesadores esclavos.

Un ejemplo de sistema con multiprocesamiento simétrico es la versión Encore de UNIX para el computador Multimax. Este computador puede configurarse de modo que utilice docenas de procesadores, todos los cuales ejecutan una copia de UNIX. La ventaja de este modelo es que muchos procesos pueden ejecutarse simultáneamente (N procesos si hay N CPU) sin que haya deterioro del desempeño. Sin

embargo, es preciso controlar cuidadosamente la E/S para asegurar que los datos lleguen al procesador apropiado. Además, dado que las CPU son independientes, una podría estar ociosa mientras otra está sobrecargada, con la consiguiente ineficiencia. A fin de evitar esto, los procesadores pueden compartir ciertas estructuras de datos. Un sistema multiprocesador de este tipo permite compartir dinámicamente trabajos y recursos entre los distintos procesadores, y puede reducir la varianza entre los sistemas. Por otro lado, tales sistemas deben escribirse con mucho cuidado, como veremos en el capítulo 6.

El multiprocesamiento asimétrico es más común en los sistemas extremadamente grandes, donde una de las actividades que más tiempo consume es el procesamiento de E/S. En los sistemas por lotes más antiguos, se utilizaban procesadores pequeños, situados a cierta distancia de la CPU principal, para operar los lectores de tarjetas e impresoras de líneas y transferir estos trabajos a y de el computador principal. Estos puntos se denominan sitios de *entrada remota de trabajos* (*RJE, remote-job-entry*). En un sistema de tiempo compartido, una de las principales actividades de E/S es procesar la E/S de caracteres entre las terminales y el computador. Si es preciso interrumpir la CPU principal cada vez que se transfiere un carácter a o de una terminal, la CPU podría pasar todo su tiempo procesando caracteres. Para evitar esta situación, la mayor parte de los sistemas cuenta con un procesador de extremo frontal (*front-end*) aparte que se encarga de toda la E/S de terminales. Por ejemplo, un sistema IBM grande podría usar un minicomputador IBM Series/1 como *front-end*. El *front-end* actúa como *buffer* entre las terminales y la CPU principal, y permite a esta última manejar líneas y bloques de caracteres, en lugar de caracteres individuales. La confiabilidad de tales sistemas se reduce a causa de su mayor especialización.

Es importante darse cuenta de que la diferencia entre multiprocesamiento simétrico y asimétrico puede deberse al hardware o al software. Podría existir hardware especial para diferenciar los múltiples procesadores, o podría escribirse software que permita un solo maestro y varios esclavos. Por ejemplo, el sistema operativo de Sun, SunOS versión 4, ofrece multiprocesamiento asimétrico, mientras que la versión 5 (Solaris 2) es simétrica.

A medida que baja el precio y aumenta la potencia de los microprocesadores, más y más funciones del sistema operativo se delegan a procesadores esclavos, también llamados *back-ends* o *procesadores de servicio*. Por ejemplo, es fácil añadir un procesador con su propia memoria para administrar un sistema de disco. El microprocesador podría recibir una secuencia de solicitudes de la CPU principal e implementar su propia cola de disco y algoritmo de planificación. Esta organización evita a la CPU principal el trabajo extra que implica la planificación de disco. Los PC contienen un microprocesador en el teclado que convierte las digitaciones en códigos que se envían a la CPU. De hecho, este uso de los microprocesadores se ha vuelto tan común que ya no se considera como multiprocesamiento.

1.7 ■ Sistemas distribuidos

Una tendencia reciente en los sistemas de computador es distribuir el cómputo entre varios procesadores. En contraste con los sistemas fuertemente acoplados que

vimos en la sección 1.6, los procesadores no comparten la memoria ni el reloj. En vez de ello, cada procesador tiene su propia memoria local. Los procesadores se comunican entre sí a través de diversas líneas de comunicación, como *buses* de alta velocidad o líneas telefónicas. Solemos decir que tales sistemas están *débilmente acoplados* o *distribuidos*.

Los procesadores de un sistema distribuido pueden tener diferentes tamaños y funciones; pueden incluir microprocesadores pequeños, estaciones de trabajo, minicomputadores y sistemas de computador de propósito general grandes. Tales procesadores reciben varios nombres distintos, como *sitios*, *nodos*, *computadores*, etc., dependiendo del contexto en el que se mencionan.

Hay diversas razones para construir sistemas distribuidos, siendo las principales:

- **Recursos Compartidos**. Si varios sitios distintos (con diferentes capacidades) se conectan entre sí, un usuario de un sitio podría aprovechar los recursos disponibles en otro. Por ejemplo, un usuario del sitio A podría estar usando una impresora láser que sólo está disponible en el sitio B. Mientras tanto, un usuario del sitio B podría estar accediendo a un archivo que reside en A. En general, el uso de recursos compartidos en un sistema distribuido ofrece mecanismos para compartir archivos en sitios remotos, procesar información de una base de datos distribuida, imprimir archivos en sitios remotos, usar equipos especializados remotos (como un procesador de matriz de alta velocidad) y realizar otras operaciones.
- **Computación más rápida**. Si un cálculo dado se puede dividir en varios subcálculos susceptibles de ejecución concurrente, un sistema distribuido podría permitirnos distribuir el cálculo entre los distintos sitios, y ejecutarlo de forma concurrente. Además, si un sitio en particular actualmente está sobrecargado de trabajos, algunos de ellos podrían transferirse a otros sitios cuya carga sea más ligera. Esta transferencia de trabajos se llama *carga compartida*.
- **Confiabilidad**. Si un sitio de un sistema distribuido falla, los sitios restantes podrían seguir funcionando. Si el sistema se compone de varias instalaciones autónomas grandes (es decir, computadores de propósito general), el fallo de uno no deberá afectar a los demás. Por otro lado, si el sistema consiste en varias máquinas pequeñas, cada una de las cuales se encarga de alguna función crucial del sistema (como la E/S de caracteres de las terminales, o el sistema de archivos), un solo fallo podría detener todo el sistema. En general, si existe suficiente redundancia en el sistema (tanto de hardware como de datos), éste puede continuar con su operación, incluso si algunos de sus sitios han fallado.
- **Comunicación**. Hay muchos casos en los que los programas necesitan intercambiar datos con otros programas del mismo sistema. Los sistemas de ventanas son un ejemplo, ya que a menudo comparten datos o transfieren datos entre presentaciones. Si muchos sitios están conectados a través de una red de comunicaciones, los procesos de diferentes sitios tienen la oportunidad de intercambiar información. Los usuarios podrían iniciar transferencias de archivos o comunicarse entre sí por *correo electrónico*. Un usuario puede enviar correo a otro en el mismo sitio o en uno distinto.

En el capítulo 15 presentaremos los sistemas distribuidos y estudiaremos la estructura general de las redes que los interconectan. En el capítulo 16 veremos la estructura general de los sistemas distribuidos. En el capítulo 17 analizaremos las distintas formas de diseñar e implementar un sistema de archivos distribuido. Por último, en el capítulo 18 trataremos la coordinación distribuida.

1.8 ■ Sistemas de tiempo real

Otra forma de sistema operativo de propósito especial es el sistema de *tiempo real*. Se usa un sistema de tiempo real cuando los requisitos de tiempo de la operación de un procesador o del flujo de datos son estrictos; por ello, a menudo se utilizan como dispositivos de control en aplicaciones dedicadas. Los sensores envían datos al computador, el cual debe analizar estos datos y posiblemente ajustar controles a fin de modificar las entradas de los sensores. Los sistemas que controlan experimentos científicos, los que producen imágenes médicas, los de control industrial y algunos sistemas de exhibición son sistemas de tiempo real. Esta clasificación también incluye algunos sistemas de inyección de combustible para motores de automóviles, controladores de aparatos domésticos y sistemas de armamentos. Un sistema operativo de tiempo real tiene restricciones de tiempo fijas bien definidas. El procesamiento *debe* efectuarse dentro de los intervalos definidos, o el sistema fallará. Por ejemplo, no es conveniente ordenar a un brazo robot que se detenga *después* de haber chocado con el automóvil que está construyendo. Se considera que un sistema de tiempo real está funcionando correctamente sólo si produce el resultado correcto dentro de los intervalos de tiempo estipulados. Podemos contrastar este requisito con un sistema de tiempo compartido, en el que es deseable (pero no obligatorio) responder rápidamente, o con un sistema por lotes, en el que tal vez no haya restricciones de tiempo.

Hay dos tipos de sistemas de tiempo real. Un sistema de *tiempo real duro* garantiza que las tareas críticas se terminarán a tiempo. Este objetivo requiere que todos los retardos del sistema estén limitados, desde la obtención de datos almacenados hasta el tiempo que el sistema operativo tarda en atender cada solicitud que se le presenta. Tales restricciones de tiempo determinan los recursos que están disponibles en este tipo de sistemas. El almacenamiento secundario de cualquier índole suele estar limitado o ausente, y los datos se almacenan de preferencia en memoria de corto plazo o en memoria sólo de lectura (ROM, *read-only memory*). La ROM se encuentra en dispositivos de almacenamiento no volátil que conservan su contenido aun en caso de fallar el suministro de electricidad; casi todos los demás tipos de memoria son volátiles. También está ausente la mayor parte de las funciones avanzadas de los sistemas operativos, ya que tienden a separar al usuario aún más del hardware, y tal separación causa incertidumbre acerca del tiempo que una operación tarda. Por ejemplo, los sistemas de tiempo real casi nunca tienen memoria virtual (que estudiaremos en el capítulo 9). Por ello, los sistemas de tiempo real duros son incompatibles con el funcionamiento de los sistemas de tiempo compartido, y no pueden combinarse con ellos. Puesto que ninguno de los sistemas operativos de propósito general existentes apoya la funcionalidad de tiempo real dura, no nos ocuparemos de este tipo de sistemas en el presente texto.

Un tipo menos restrictivo de sistema de tiempo real es el de *tiempo real blando*, en el que una tarea de tiempo real crítica goza de prioridad respecto a otras tareas, y conserva esa prioridad hasta que se lleva a cabo. Al igual que en los sistemas de tiempo real duros, es preciso limitar los retardos del núcleo: no es posible mantener a una tarea de tiempo real esperando indefinidamente a que el núcleo la ejecute. El tiempo real blando es una meta alcanzable que puede combinarse con otros tipos de sistemas. No obstante, los sistemas de tiempo real blandos tienen una utilidad más limitada que los duros. En vista de que no apoyan el cumplimento estricto de plazos, es riesgoso utilizarlos en control industrial y robótica, aunque hay varias áreas en las que pueden ser útiles, como multimedia, realidad virtual y proyectos científicos avanzados como la exploración submarina y planetaria. Estos sistemas requieren características avanzadas de los sistemas operativos que no pueden incluirse en los sistemas de tiempo real duro. La proliferación del uso de funciones de tiempo real blando ha hecho que se incluyan en la mayor parte de los sistemas operativos actuales, incluidas versiones importantes de UNIX.

En el capítulo 5 consideraremos el recurso de planificación necesario para implementar las funciones de tiempo real blando en un sistema operativo. En el capítulo 9 describiremos el diseño de la gestión de memoria para computación en tiempo real. Por último, en el capítulo 23, describiremos los componentes de tiempo real del sistema operativo Windows NT.

1.9 ■ Resumen

Durante los últimos 40 años se han desarrollado sistemas operativos con dos fines principales. En primer lugar, los sistemas operativos intentan planificar las actividades de cómputo con objeto de asegurar el buen desempeño del sistema de computador. Segundo, los sistemas operativos ofrecen un entorno cómodo para la creación y ejecución de programas.

En un principio, los sistemas de computación se operaban desde la consola delantera. Programas como los ensambladores, cargadores, enlazadores y compiladores hicieron más cómoda la programación del sistema, pero también requerían un tiempo de preparación considerable. A fin de reducir este tiempo, las instalaciones contrataron operadores y formaron lotes de trabajos similares.

En los sistemas por lotes se hizo posible la ejecución secuencial automática de trabajos mediante un sistema operativo residente, y se mejoró enormemente el aprovechamiento global del computador. El computador ya no tenía que esperar a que una persona lo operara. No obstante, el aprovechamiento de la CPU seguía siendo bajo a causa de la relativa lentitud de los dispositivos de E/S en comparación con la CPU. La operación fuera de línea de los dispositivos lentos hace posible usar múltiples sistemas de lector a cinta y de cinta a impresora con una misma CPU. El *spooling* permite a la CPU superponer las operaciones de entrada de un trabajo con los cálculos y salidas de otros trabajos.