1

INTRODUCCIÓN

Una computadora moderna consta de uno o más procesadores, una memoria principal, discos, impresoras, un teclado, un ratón, una pantalla o monitor, interfaces de red y otros dispositivos de entrada/salida. En general es un sistema complejo. Si todos los programadores de aplicaciones tuvieran que comprender el funcionamiento de todas estas partes, no escribirían código alguno. Es más: el trabajo de administrar todos estos componentes y utilizarlos de manera óptima es una tarea muy desafiante. Por esta razón, las computadoras están equipadas con una capa de software llamada **sistema operativo**, cuyo trabajo es proporcionar a los programas de usuario un modelo de computadora mejor, más simple y pulcro, así como encargarse de la administración de todos los recursos antes mencionados. Los sistemas operativos son el tema de este libro.

La mayoría de los lectores habrán tenido cierta experiencia con un sistema operativo como Windows, Linux, FreeBSD o Mac OS X, pero las apariencias pueden ser engañosas. El programa con el que los usuarios generalmente interactúan se denomina **shell**, cuando está basado en texto, y **GUI** (*Graphical User Interface*; Interfaz gráfica de usuario**)** cuando utiliza elementos gráficos o iconos. En realidad no forma parte del sistema operativo, aunque lo utiliza para llevar a cabo su trabajo.

La figura 1-1 presenta un esquema general de los componentes principales que aquí se analizan. En la parte inferior se muestra el hardware, que consiste en circuitos integrados (chips), tarjetas, discos, un teclado, un monitor y objetos físicos similares. Por encima del hardware se encuentra el software. La mayoría de las computadoras tienen dos modos de operación: modo kernel y modo usuario. El sistema operativo es la pieza fundamental del software y se ejecuta en **modo kernel** (también conocido como **modo supervisor**). En este modo, el sistema operativo tiene acceso

completo a todo el hardware y puede ejecutar cualquier instrucción que la máquina sea capaz de ejecutar. El resto del software se ejecuta en **modo usuario**, en el cual sólo un subconjunto de las instrucciones de máquina es permitido. En particular, las instrucciones que afectan el control de la máquina o que se encargan de la E/S (entrada/salida) están prohibidas para los programas en modo usuario. Volveremos a tratar las diferencias entre el modo kernel y el modo usuario repetidamente a lo largo de este libro.

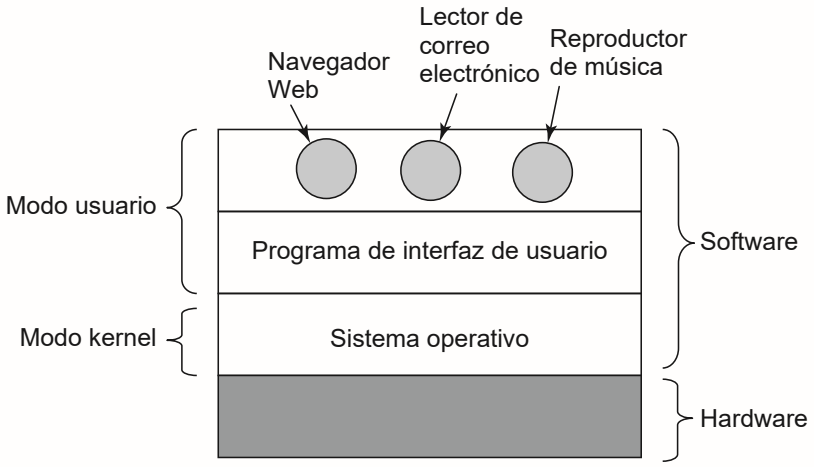

Figura 1-1. Ubicación del sistema operativo.

El programa de interfaz de usuario, shell o GUI, es el nivel más bajo del software en modo usuario y permite la ejecución de otros programas, como un navegador Web, lector de correo electrónico o reproductor de música. Estos programas también utilizan en forma intensiva el sistema operativo.

La ubicación del sistema operativo se muestra en la figura 1-1. Se ejecuta directamente sobre el hardware y proporciona la base para las demás aplicaciones de software.

Una distinción importante entre el sistema operativo y el software que se ejecuta en modo usuario es que, si a un usuario no le gusta, por ejemplo, su lector de correo electrónico, es libre de conseguir otro o incluso escribir el propio si así lo desea; sin embargo, no es libre de escribir su propio manejador de interrupciones de reloj, que forma parte del sistema operativo y está protegido por el hardware contra cualquier intento de modificación por parte de los usuarios.

Algunas veces esta distinción no es clara en los sistemas integrados (a los que también se conoce como integrados o incrustados, y que podrían no tener modo kernel) o en los sistemas interpretados (como los sistemas operativos basados en Java que para separar los componentes utilizan interpretación y no el hardware).

Además, en muchos sistemas hay programas que se ejecutan en modo de usuario, pero ayudan al sistema operativo o realizan funciones privilegiadas. Por ejemplo, a menudo hay un programa que permite a los usuarios cambiar su contraseña. Este programa no forma parte del sistema operativo y no se ejecuta en modo kernel, pero sin duda lleva a cabo una función delicada y tiene que proteger-

se de una manera especial. En ciertos sistemas, la idea se lleva hasta el extremo y partes de lo que tradicionalmente se considera el sistema operativo (por ejemplo, el sistema de archivos) se ejecutan en el espacio del usuario. En dichos sistemas es difícil trazar un límite claro. Todo lo que se ejecuta en modo kernel forma, sin duda, parte del sistema operativo, pero podría decirse que algunos programas que se ejecutan fuera de este modo también forman parte del mismo sistema, o por lo menos están estrechamente asociados a él.

Los sistemas operativos difieren de los programas de usuario (es decir, de aplicación) en varias cuestiones además del lugar en el que residen. En particular, son enormes, complejos y de larga duración. El código fuente de un sistema operativo como Linux o Windows contiene cerca de cinco millones de líneas de código. Para tener una idea de lo que esto significa, considere el trabajo de imprimir cinco millones de líneas en un formato de libro: con 50 líneas por página y 1000 páginas por volumen, se requerirían 100 volúmenes para listar un sistema operativo de este tamaño; es decir, todo un librero. Imagine el lector que tiene un trabajo como encargado de dar mantenimiento a un sistema operativo y que en su primer día su jefe le presenta un librero con el código y le dice: "Apréndase todo esto". Y ésta sólo sería la parte que se ejecuta en el kernel. Los programas de usuario como la interfaz gráfica, las bibliotecas y el software de aplicación básico (como el Explorador de Windows) pueden abarcar fácilmente de 10 a 20 veces esa cantidad.

En este punto, el lector debe tener una idea clara de por qué los sistemas operativos tienen una larga vida: es muy difícil escribir uno y, por lo tanto, el propietario se resiste a tirarlo y empezar de nuevo. En vez de ello, evolucionan durante periodos extensos. Windows 95/98/Me es, esencialmente, un sistema operativo distinto de Windows NT/2000/XP/Vista, su sucesor. Tienen una apariencia similar para los usuarios, ya que Microsoft se aseguró bien de ello, sin embargo, tuvo muy buenas razones para deshacerse de Windows 98, las cuales describiremos cuando estudiemos Windows con detalle en el capítulo 11.

El otro ejemplo principal que utilizaremos a lo largo de este libro (además de Windows) es UNIX, con sus variantes y clones. También ha evolucionado a través de los años con versiones tales como System V, Solaris y FreeBSD que se derivan del sistema original, mientras que Linux tiene una base de código nueva, modelada estrechamente de acuerdo con UNIX y altamente compatible con él. Utilizaremos ejemplos de UNIX a lo largo de este libro y analizaremos Linux con detalle en el capítulo 10.

En este capítulo hablaremos brevemente sobre varios aspectos clave de los sistemas operativos, incluyendo en síntesis qué son, cuál es su historia, cuáles son los tipos que existen, algunos de los conceptos básicos y su estructura. En capítulos posteriores volveremos a hablar sobre muchos de estos tópicos importantes con más detalle.

1.1 ¿QUÉ ES UN SISTEMA OPERATIVO?

Es difícil definir qué es un sistema operativo aparte de decir que es el software que se ejecuta en modo kernel (además de que esto no siempre es cierto). Parte del problema es que los sistemas operativos realizan dos funciones básicas que no están relacionadas: proporcionar a los programadores de aplicaciones (y a los programas de aplicaciones, naturalmente) un conjunto abstracto de recursos simples, en vez de los complejos conjuntos de hardware; y administrar estos recursos de hard-

ware. Dependiendo de quién se esté hablando, el lector podría escuchar más acerca de una función o de la otra. Ahora analizaremos ambas.

1.1.1 El sistema operativo como una máquina extendida

La **arquitectura** (conjunto de instrucciones, organización de memoria, E/S y estructura de bus) de la mayoría de las computadoras a nivel de lenguaje máquina es primitiva y compleja de programar, en especial para la entrada/salida. Para hacer este punto más concreto, considere la forma en que se lleva a cabo la E/S de disco flexible mediante los dispositivos controladores (*device controllers*) compatibles NEC PD765 que se utilizan en la mayoría de las computadoras personales basadas en Intel (a lo largo de este libro utilizaremos los términos "disco flexible" y "diskette" indistintamente). Utilizamos el disco flexible como un ejemplo debido a que, aunque obsoleto, es mucho más simple que un disco duro moderno. El PD765 tiene 16 comandos, cada uno de los cuales se especifica mediante la carga de 1 a 9 bytes en un registro de dispositivo. Estos comandos son para leer y escribir datos, desplazar el brazo del disco y dar formato a las pistas, así como para inicializar, detectar, restablecer y recalibrar el dispositivo controlador y las unidades.

Los comandos más básicos son `read` y `write` (lectura y escritura), cada uno de los cuales requiere 13 parámetros, empaquetados en 9 bytes. Estos parámetros especifican elementos tales como la dirección del bloque de disco a leer, el número de sectores por pista, el modo de grabación utilizado en el medio físico, el espacio de separación entre sectores y lo que se debe hacer con una marca de dirección de datos eliminados. Si el lector no comprende estos tecnicismos, no se preocupe: ése es precisamente el punto, pues se trata de algo bastante oscuro. Cuando la operación se completa, el chip del dispositivo controlador devuelve 23 campos de estado y error, empaquetados en 7 bytes. Como si esto no fuera suficiente, el programador del disco flexible también debe estar constantemente al tanto de si el motor está encendido o apagado. Si el motor está apagado, debe encenderse (con un retraso largo de arranque) para que los datos puedan ser leídos o escritos. El motor no se debe dejar demasiado tiempo encendido porque se desgastará. Por lo tanto, el programador se ve obligado a lidiar con el problema de elegir entre tener retrasos largos de arranque o desgastar los discos flexibles (y llegar a perder los datos).

Sin entrar en los detalles *reales*, debe quedar claro que el programador promedio tal vez no desee involucrarse demasiado con la programación de los discos flexibles (o de los discos duros, que son aún más complejos). En vez de ello, lo que desea es una abstracción simple de alto nivel que se encargue de lidiar con el disco. En el caso de los discos, una abstracción común sería que el disco contiene una colección de archivos con nombre. Cada archivo puede ser abierto para lectura o escritura, después puede ser leído o escrito y, por último, cerrado. Los detalles, tales como si la grabación debe utilizar o no la modulación de frecuencia y cuál es el estado del motor en un momento dado, no deben aparecer en la abstracción que se presenta al programador de aplicaciones.

La abstracción es la clave para lidiar con la complejidad. Las buenas abstracciones convierten una tarea casi imposible en dos tareas manejables. La primera de éstas es definir e implementar las abstracciones; la segunda, utilizarlas para resolver el problema en cuestión. Una abstracción que casi cualquier usuario de computadora comprende es el archivo: es una pieza útil de información, como una fotografía digital, un mensaje de correo electrónico almacenado o una página Web. Es más fácil lidiar con fotografías, correos electrónicos y páginas Web que con los detalles de los discos,

como en el caso del disco flexible descrito. El trabajo del sistema operativo es crear buenas abstracciones para después implementar y administrar los objetos abstractos entonces creados. En este libro hablaremos mucho acerca de las abstracciones, dado que son claves para comprender los sistemas operativos.

Este punto es tan importante que vale la pena repetirlo en distintas palabras. Con el debido respeto a los ingenieros industriales que diseñaron la Macintosh, el hardware es feo. Los procesadores, memorias, discos y otros dispositivos reales son muy complicados y presentan interfaces difíciles, enredadas, muy peculiares e inconsistentes para las personas que tienen que escribir software para utilizarlos. Algunas veces esto se debe a la necesidad de tener compatibilidad con el hardware anterior; otras, a un deseo de ahorrar dinero, y otras más, a que los diseñadores de hardware no tienen idea (o no les importa) qué tan grave es el problema que están ocasionando para el software. Una de las principales tareas del sistema operativo es ocultar el hardware y presentar a los programas (y a sus programadores) abstracciones agradables, elegantes, simples y consistentes con las que puedan trabajar. Los sistemas operativos ocultan la parte fea con la parte hermosa, como se muestra en la figura 1-2.

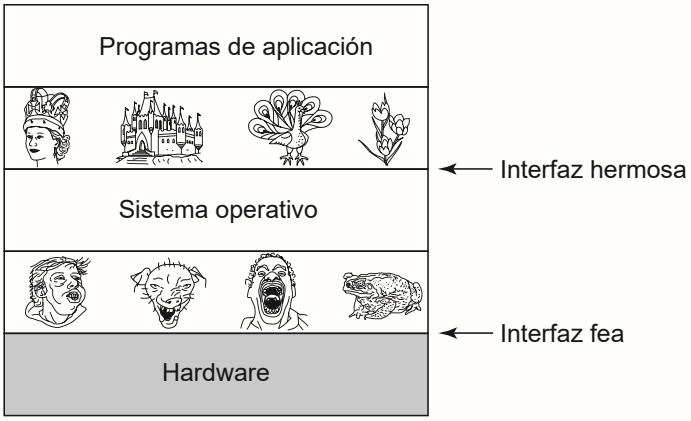

Figura 1-2. Los sistemas operativos ocultan el hardware feo con abstracciones hermosas.

Hay que recalcar que los verdaderos clientes del sistema operativo son los programas de aplicación (a través de los programadores de aplicaciones, desde luego). Son los que tratan directamente con el sistema operativo y sus abstracciones. En contraste, los usuarios finales tienen que lidiar con las abstracciones que proporciona la interfaz de usuario, ya sea un shell de línea de comandos o una interfaz gráfica. Aunque las abstracciones en la interfaz de usuario pueden ser similares a las que proporciona el sistema operativo, éste no siempre es el caso. Para aclarar este punto, considere el escritorio normal de Windows y el indicador para comandos orientado a texto. Ambos son programas que se ejecutan en el sistema operativo Windows y utilizan las abstracciones que este sistema proporciona, pero ofrecen interfaces de usuario muy distintas. De manera similar, un usuario de Linux que ejecuta Gnome o KDE ve una interfaz muy distinta a la que ve un usuario de Linux que trabaja directamente encima del Sistema X Window subyacente (orientado a texto), pero las abstracciones del sistema operativo subyacente son las mismas en ambos casos.

En este libro estudiaremos detalladamente las abstracciones que se proporcionan a los programas de aplicación, pero trataremos muy poco acerca de las interfaces de usuario, que es un tema bastante extenso e importante, pero que sólo está relacionado con la periferia de los sistemas operativos.

1.1.2 El sistema operativo como administrador de recursos

El concepto de un sistema operativo cuya función principal es proporcionar abstracciones a los programas de aplicación responde a una perspectiva de arriba hacia abajo. La perspectiva alterna, de abajo hacia arriba, sostiene que el sistema operativo está presente para administrar todas las piezas de un sistema complejo. Las computadoras modernas constan de procesadores, memorias, temporizadores, discos, ratones, interfaces de red, impresoras y una amplia variedad de otros dispositivos. En la perspectiva alterna, el trabajo del sistema operativo es proporcionar una asignación ordenada y controlada de los procesadores, memorias y dispositivos de E/S, entre los diversos programas que compiten por estos recursos.

Los sistemas operativos modernos permiten la ejecución simultánea de varios programas. Imagine lo que ocurriría si tres programas que se ejecutan en cierta computadora trataran de imprimir sus resultados en forma simultánea en la misma impresora. Las primeras líneas de impresión podrían provenir del programa 1, las siguientes del programa 2, después algunas del programa 3, y así en lo sucesivo: el resultado sería un caos. El sistema operativo puede imponer orden al caos potencial, guardando en búferes en disco toda la salida destinada para la impresora. Cuando termina un programa, el sistema operativo puede entonces copiar su salida, previamente almacenada, del archivo en disco a la impresora, mientras que al mismo tiempo el otro programa puede continuar generando más salida, ajeno al hecho de que la salida en realidad no se está enviando a la impresora todavía.

Cuando una computadora (o red) tiene varios usuarios, la necesidad de administrar y proteger la memoria, los dispositivos de E/S y otros recursos es cada vez mayor; de lo contrario, los usuarios podrían interferir unos con otros. Además, los usuarios necesitan con frecuencia compartir no sólo el hardware, sino también la información (archivos o bases de datos, por ejemplo). En resumen, esta visión del sistema operativo sostiene que su tarea principal es llevar un registro de qué programa está utilizando qué recursos, de otorgar las peticiones de recursos, de contabilizar su uso y de mediar las peticiones en conflicto provenientes de distintos programas y usuarios.

La administración de recursos incluye el **multiplexaje** (compartir) de recursos en dos formas distintas: en el tiempo y en el espacio. Cuando un recurso se multiplexa en el tiempo, los distintos programas o usuarios toman turnos para utilizarlo: uno de ellos obtiene acceso al recurso, después otro, y así en lo sucesivo. Por ejemplo, con sólo una CPU y varios programas que desean ejecutarse en ella, el sistema operativo primero asigna la CPU a un programa y luego, una vez que se ha ejecutado por el tiempo suficiente, otro programa obtiene acceso a la CPU, después otro, y en un momento dado el primer programa vuelve a obtener acceso al recurso. La tarea de determinar cómo se multiplexa el recurso en el tiempo (quién sigue y durante cuánto tiempo) es responsabilidad del sistema operativo. Otro ejemplo de multiplexaje en el tiempo es la compartición de la impresora. Cuando hay varios trabajos en una cola de impresión, para imprimirlos en una sola impresora, se debe tomar una decisión en cuanto a cuál trabajo debe imprimirse a continuación.

El otro tipo de multiplexaje es en el espacio. En vez de que los clientes tomen turnos, cada uno obtiene una parte del recurso. Por ejemplo, normalmente la memoria principal se divide entre varios programas en ejecución para que cada uno pueda estar residente al mismo tiempo (por ejemplo, para poder tomar turnos al utilizar la CPU). Suponiendo que hay suficiente memoria como para contener varios programas, es más eficiente contener varios programas en memoria a la vez, en vez de proporcionar a un solo programa toda la memoria, en especial si sólo necesita una pequeña fracción. Desde luego que esto genera problemas de equidad y protección, por ejemplo, y corresponde al sistema operativo resolverlos. Otro recurso que se multiplexa en espacio es el disco duro. En muchos sistemas, un solo disco puede contener archivos de muchos usuarios al mismo tiempo. Asignar espacio en disco y llevar el registro de quién está utilizando cuáles bloques de disco es una tarea típica de administración de recursos común del sistema operativo.

1.2 HISTORIA DE LOS SISTEMAS OPERATIVOS

Los sistemas operativos han ido evolucionando a través de los años. En las siguientes secciones analizaremos brevemente algunos de los hitos más importantes. Como los sistemas operativos han estado estrechamente relacionados a través de la historia con la arquitectura de las computadoras en las que se ejecutan, analizaremos generaciones sucesivas de computadoras para ver cómo eran sus sistemas operativos. Esta vinculación de generaciones de sistemas operativos con generaciones de computadoras es un poco burda, pero proporciona cierta estructura donde de cualquier otra forma no habría.

La progresión que se muestra a continuación es en gran parte cronológica, aunque el desarrollo ha sido un tanto accidentado. Cada fase surgió sin esperar a que la anterior terminara completamente. Hubo muchos traslapes, sin mencionar muchos falsos inicios y callejones sin salida. El lector debe tomar esto como guía, no como la última palabra.

La primera computadora digital verdadera fue diseñada por el matemático inglés Charles Babbage (de 1792 a 1871). Aunque Babbage gastó la mayor parte de su vida y fortuna tratando de construir su “máquina analítica”, nunca logró hacer que funcionara de manera apropiada, debido a que era puramente mecánica y la tecnología de su era no podía producir las ruedas, engranes y dientes con la alta precisión que requería. Por supuesto, la máquina analítica no tenía un sistema operativo.

Como nota histórica interesante, Babbage se dio cuenta de que necesitaba software para su máquina analítica, por lo cual contrató a una joven llamada Ada Lovelace, hija del afamado poeta británico Lord Byron, como la primera programadora del mundo. El lenguaje de programación Ada® lleva su nombre.

1.2.1 La primera generación (1945 a 1955): tubos al vacío

Después de los esfuerzos infructuosos de Babbage, no hubo muchos progresos en la construcción de computadoras digitales sino hasta la Segunda Guerra Mundial, que estimuló una explosión de esta actividad. El profesor John Atanasoff y su estudiante graduado Clifford Berry construyeron lo que ahora se conoce como la primera computadora digital funcional en Iowa State University. Utilizaba 300 tubos de vacío (bulbos). Aproximadamente al mismo tiempo, Konrad Zuse en Berlín

construyó la computadora Z3 a partir de relevadores. En 1944, la máquina Colossus fue construida por un equipo de trabajo en Bletchley Park, Inglaterra; la Mark I, por Howard Aiken en Harvard, y la ENIAC, por William Mauchley y su estudiante graduado J. Presper Eckert en la Universidad de Pennsylvania. Algunas fueron binarias, otras utilizaron bulbos, algunas eran programables, pero todas eran muy primitivas y tardaban segundos en realizar incluso hasta el cálculo más simple.

En estos primeros días, un solo grupo de personas (generalmente ingenieros) diseñaban, construían, programaban, operaban y daban mantenimiento a cada máquina. Toda la programación se realizaba exclusivamente en lenguaje máquina o, peor aún, creando circuitos eléctricos mediante la conexión de miles de cables a tableros de conexiones (*plugboards*) para controlar las funciones básicas de la máquina. Los lenguajes de programación eran desconocidos (incluso se desconocía el lenguaje ensamblador). Los sistemas operativos también se desconocían. El modo usual de operación consistía en que el programador trabajaba un periodo dado, registrándose en una hoja de firmas, y después entraba al cuarto de máquinas, insertaba su tablero de conexiones en la computadora e invertía varias horas esperando que ninguno de los cerca de 20,000 bulbos se quemara durante la ejecución. Prácticamente todos los problemas eran cálculos numéricos bastante simples, como obtener tablas de senos, cosenos y logaritmos.

A principios de la década de 1950, la rutina había mejorado un poco con la introducción de las tarjetas perforadas. Entonces fue posible escribir programas en tarjetas y leerlas en vez de usar tableros de conexiones; aparte de esto, el procedimiento era el mismo.

1.2.2 La segunda generación (1955 a 1965): transistores y sistemas de procesamiento por lotes

La introducción del transistor a mediados de la década de 1950 cambió radicalmente el panorama. Las computadoras se volvieron lo bastante confiables como para poder fabricarlas y venderlas a clientes dispuestos a pagar por ellas, con la expectativa de que seguirían funcionando el tiempo suficiente como para poder llevar a cabo una cantidad útil de trabajo. Por primera vez había una clara separación entre los diseñadores, constructores, operadores, programadores y el personal de mantenimiento.

Estas máquinas, ahora conocidas como **mainframes**, estaban encerradas en cuartos especiales con aire acondicionado y grupos de operadores profesionales para manejarlas. Sólo las empresas grandes, universidades o agencias gubernamentales importantes podían financiar el costo multimillonario de operar estas máquinas. Para ejecutar un **trabajo** (es decir, un programa o conjunto de programas), el programador primero escribía el programa en papel (en FORTRAN o en ensamblador) y después lo pasaba a tarjetas perforadas. Luego llevaba el conjunto de tarjetas al cuarto de entrada de datos y lo entregaba a uno de los operadores; después se iba a tomar un café a esperar a que los resultados estuvieran listos.

Cuando la computadora terminaba el trabajo que estaba ejecutando en un momento dado, un operador iba a la impresora y arrancaba las hojas de resultados para llevarlas al cuarto de salida de datos, para que el programador pudiera recogerlas posteriormente. Entonces, el operador tomaba uno de los conjuntos de tarjetas que se habían traído del cuarto de entrada y las introducía en la máquina. Si se necesitaba el compilador FORTRAN, el operador tenía que obtenerlo de un gabinete

de archivos e introducirlo a la máquina. Se desperdiciaba mucho tiempo de la computadora mientras los operadores caminaban de un lado a otro del cuarto de la máquina.

Dado el alto costo del equipo, no es sorprendente que las personas buscaran rápidamente formas de reducir el tiempo desperdiciado. La solución que se adoptó en forma general fue el **sistema de procesamiento por lotes**. La idea detrás de este concepto era recolectar una bandeja llena de trabajos en el cuarto de entrada de datos y luego pasarlos a una cinta magnética mediante el uso de una pequeña computadora relativamente económica, tal como la IBM 1401, que era muy adecuada para leer las tarjetas, copiar cintas e imprimir los resultados, pero no tan buena para los cálculos numéricos. Para llevar a cabo los cálculos numéricos se utilizaron otras máquinas mucho más costosas, como la IBM 7094. Este procedimiento se ilustra en la figura 1-3.

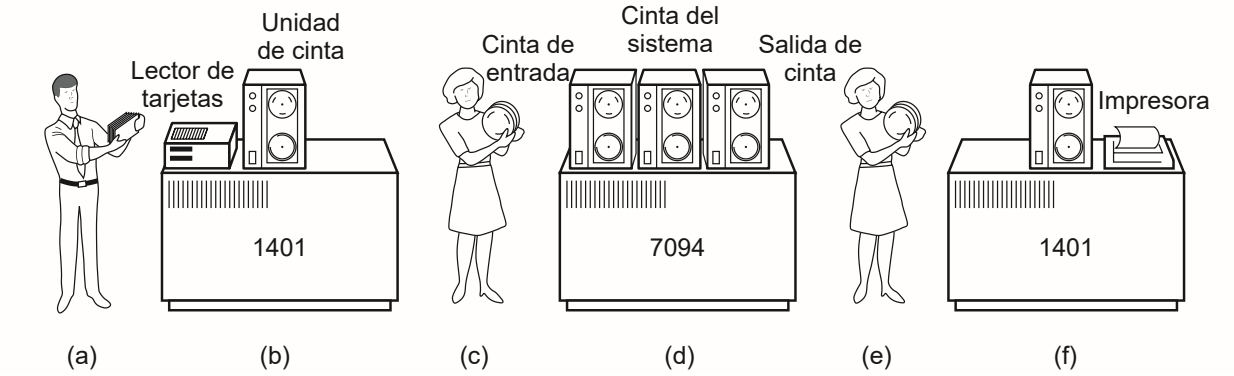

Figura 1-3. Uno de los primeros sistemas de procesamiento por lotes. a) Los programadores llevan las tarjetas a la 1401. b) La 1401 lee los lotes de trabajos y los coloca en cinta. c) El operador lleva la cinta de entrada a la 7094. d) La 7094 realiza los cálculos. e) El operador lleva la cinta de salida a la 1401. f) La 1401 imprime los resultados.

Después de aproximadamente una hora de recolectar un lote de trabajos, las tarjetas se leían y se colocaban en una cinta magnética, la cual se llevaba al cuarto de máquinas, en donde se montaba en una unidad de cinta. Después, el operador cargaba un programa especial (el ancestro del sistema operativo de hoy en día), el cual leía el primer trabajo de la cinta y lo ejecutaba. Los resultados se escribían en una segunda cinta, en vez de imprimirlos. Después de que terminaba cada trabajo, el sistema operativo leía de manera automática el siguiente trabajo de la cinta y empezaba a ejecutarlo. Cuando se terminaba de ejecutar todo el lote, el operador quitaba las cintas de entrada y de salida, reemplazaba la cinta de entrada con el siguiente lote y llevaba la cinta de salida a una 1401 para imprimir **fuera de línea** (es decir, sin conexión con la computadora principal).

En la figura 1-4 se muestra la estructura típica de un trabajo de entrada ordinario. Empieza con una tarjeta $JOB, especificando el tiempo máximo de ejecución en minutos, el número de cuenta al que se va a cargar y el nombre del programador. Después se utiliza una tarjeta $FORTRAN, indicando al sistema operativo que debe cargar el compilador FORTRAN de la cinta del sistema. Después le sigue inmediatamente el programa que se va a compilar y luego una tarjeta $LOAD, que indica al sistema operativo que debe cargar el programa objeto que acaba de compilar (a menudo, los programas compilados se escribían en cintas reutilizables y tenían que cargarse en forma explícita). Después se utiliza la tarjeta $RUN, la cual indica al sistema operativo que debe ejecutar el

programa con los datos que le suceden. Por último, la tarjeta $END marca el final del trabajo. Estas tarjetas de control primitivas fueron las precursoras de los shells e intérpretes de línea de comandos modernos.

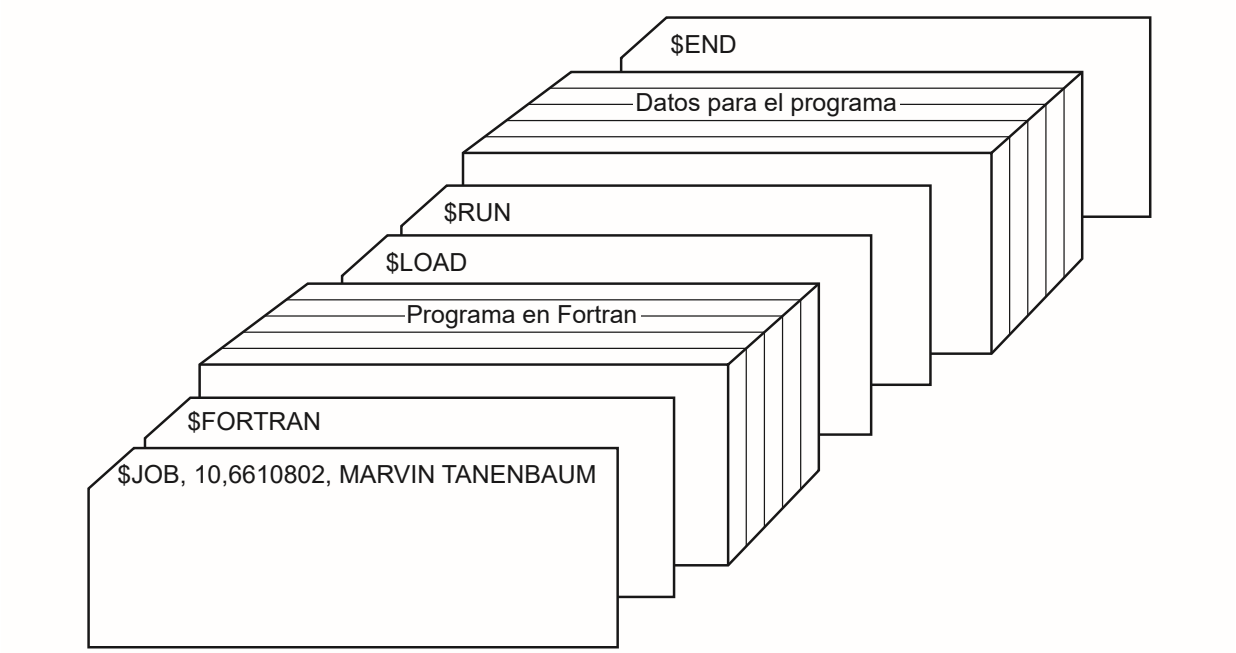

Figura 1-4. Estructura de un trabajo típico de FMS.

Las computadoras grandes de segunda generación se utilizaron principalmente para cálculos científicos y de ingeniería, tales como resolver ecuaciones diferenciales parciales que surgen a menudo en física e ingeniería. En gran parte se programaron en FORTRAN y lenguaje ensamblador. Los sistemas operativos típicos eran FMS (*Fortran Monitor System*) e IBSYS, el sistema operativo de IBM para la 7094.

1.2.3 La tercera generación (1965 a 1980): circuitos integrados y multiprogramación

A principio de la década de 1960, la mayoría de los fabricantes de computadoras tenían dos líneas de productos distintas e incompatibles. Por una parte estaban las computadoras científicas a gran escala orientadas a palabras, como la 7094, que se utilizaban para cálculos numéricos en ciencia e ingeniería. Por otro lado, estaban las computadoras comerciales orientadas a caracteres, como la 1401, que se utilizaban ampliamente para ordenar cintas e imprimir datos en los bancos y las compañías de seguros.

Desarrollar y dar mantenimiento a dos líneas de productos completamente distintos era una propuesta costosa para los fabricantes. Además, muchos nuevos clientes de computadoras necesitaban al principio un equipo pequeño, pero más adelante ya no era suficiente y deseaban una máquina más grande que pudiera ejecutar todos sus programas anteriores, pero con mayor rapidez.

IBM intentó resolver ambos problemas de un solo golpe con la introducción de la línea de computadoras System/360. La 360 era una serie de máquinas compatibles con el software, que variaban desde un tamaño similar a la 1401 hasta algunas que eran más potentes que la 7094. Las máquinas sólo diferían en el precio y rendimiento (máxima memoria, velocidad del procesador, número de dispositivos de E/S permitidos, etcétera). Como todas las máquinas tenían la misma arquitectura y el mismo conjunto de instrucciones, los programas escritos para una máquina podían ejecutarse en todas las demás, por lo menos en teoría. Lo que es más, la 360 se diseñó para manejar tanto la computación científica (es decir, numérica) como comercial. Por ende, una sola familia de máquinas podía satisfacer las necesidades de todos los clientes. En los años siguientes, mediante el uso de tecnología más moderna, IBM ha desarrollado sucesores compatibles con la línea 360, a los cuales se les conoce como modelos 370, 4300, 3080 y 3090. La serie zSeries es el descendiente más reciente de esta línea, aunque diverge considerablemente del original.

La IBM 360 fue la primera línea importante de computadoras en utilizar **circuitos integrados** (**ICs**) (a pequeña escala), con lo cual se pudo ofrecer una mayor ventaja de precio/rendimiento en comparación con las máquinas de segunda generación, las cuales fueron construidas a partir de transistores individuales. Su éxito fue inmediato y la idea de una familia de computadoras compatibles pronto fue adoptada por todos los demás fabricantes importantes. Los descendientes de estas máquinas se siguen utilizando hoy día en centros de cómputo. En la actualidad se utilizan con frecuencia para manejar bases de datos enormes (por ejemplo, para sistemas de reservaciones de aerolíneas) o como servidores para sitios de World Wide Web que deben procesar miles de solicitudes por segundo.

La mayor fortaleza de la idea de "una sola familia" fue al mismo tiempo su mayor debilidad. La intención era que todo el software, incluyendo al sistema operativo **OS/360**, funcionara en todos los modelos. Debía ejecutarse en los sistemas pequeños, que por lo general sólo reemplazaban a la 1401s, que copiaba tarjetas a cinta, y en los sistemas muy grandes, que a menudo reemplazaban a la 7094s, que realizaba predicciones del clima y otros cálculos pesados. Tenía que ser bueno en sistemas con pocos dispositivos periféricos y en sistemas con muchos. Tenía que funcionar en ambos entornos comerciales y científicos. Por encima de todo, tenía que ser eficiente para todos estos usos distintos.

No había forma en que IBM (o cualquier otra) pudiera escribir una pieza de software que cumpliera con todos estos requerimientos en conflicto. El resultado fue un enorme y extraordinariamente complejo sistema operativo, tal vez de dos a tres órdenes de magnitud más grande que el FMS. Consistía en millones de líneas de lenguaje ensamblador escrito por miles de programadores, con miles de errores, los cuales requerían un flujo continuo de nuevas versiones en un intento por corregirlos. Cada nueva versión corregía algunos errores e introducía otros, por lo que probablemente el número de errores permanecía constante en el tiempo.

Fred Brooks, uno de los diseñadores del OS/360, escribió posteriormente un libro ingenioso e incisivo (Brooks, 1996) que describía sus experiencias con el OS/360. Aunque sería imposible resumir este libro aquí, basta con decir que la portada muestra una manada de bestias prehistóricas atrapadas en un pozo de brea. La portada de Silberschatz y coautores (2005) muestra un punto de vista similar acerca de que los sistemas operativos son como dinosaurios.

A pesar de su enorme tamaño y sus problemas, el OS/360 y los sistemas operativos similares de tercera generación producidos por otros fabricantes de computadoras en realidad dejaban razo-

nablemente satisfechos a la mayoría de sus clientes. También popularizaron varias técnicas clave ausentes en los sistemas operativos de segunda generación. Quizá la más importante de éstas fue la **multiprogramación**. En la 7094, cuando el trabajo actual se detenía para esperar a que se completara una operación con cinta u otro dispositivo de E/S, la CPU simplemente permanecía inactiva hasta terminar la operación de E/S. Con los cálculos científicos que requieren un uso intensivo de la CPU, la E/S no es frecuente, por lo que este tiempo desperdiciado no es considerable. Con el procesamiento de datos comerciales, el tiempo de espera de las operaciones de E/S puede ser a menudo de 80 a 90 por ciento del tiempo total, por lo que debía hacerse algo para evitar que la (costosa) CPU esté inactiva por mucho tiempo.

La solución que surgió fue particionar la memoria en varias piezas, con un trabajo distinto en cada partición, como se muestra en la figura 1-5. Mientras que un trabajo esperaba a que se completara una operación de E/S, otro podía estar usando la CPU. Si pudieran contenerse suficientes trabajos en memoria principal al mismo tiempo, la CPU podía estar ocupada casi 100 por ciento del tiempo. Para tener varios trabajos de forma segura en memoria a la vez, se requiere hardware especial para proteger cada trabajo y evitar que los otros se entrometan y lo malogren; el 360 y los demás sistemas de tercera generación estaban equipados con este hardware.

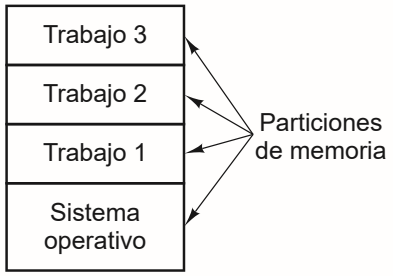

Figura 1-5. Un sistema de multiprogramación con tres trabajos en memoria.

Otra característica importante de los sistemas operativos de tercera generación fue la capacidad para leer trabajos en tarjetas y colocarlos en el disco tan pronto como se llevaban al cuarto de computadoras. Así, cada vez que terminaba un trabajo en ejecución, el sistema operativo podía cargar un nuevo trabajo del disco en la partición que entonces estaba vacía y lo ejecutaba. A esta técnica se le conoce como **spooling** (de *Simultaneous Peripheral Operation On Line*, operación periférica simultánea en línea) y también se utilizó para las operaciones de salida. Con el spooling, las máquinas 1401 no eran ya necesarias y desapareció la mayor parte del trabajo de transportar las cintas.

Aunque los sistemas operativos de tercera generación eran apropiados para los cálculos científicos extensos y las ejecuciones de procesamiento de datos comerciales masivos, seguían siendo en esencia sistemas de procesamiento por lotes. Muchos programadores añoraban los días de la primera generación en los que tenían toda la máquina para ellos durante unas cuantas horas, por lo que podían depurar sus programas con rapidez. Con los sistemas de tercera generación, el tiempo que transcurría entre enviar un trabajo y recibir de vuelta la salida era comúnmente de varias horas, por lo que una sola coma mal colocada podía ocasionar que fallara la compilación, y el programador desperdiciara la mitad del día.

Este deseo de obtener un tiempo rápido de respuesta allanó el camino para el **tiempo compartido** (*timesharing*), una variante de la multiprogramación donde cada usuario tenía una terminal en

línea. En un sistema de tiempo compartido, si 20 usuarios están conectados y 17 de ellos están pensando en dar un paseo o tomar café, la CPU se puede asignar por turno a los tres trabajos que desean ser atendidos. Como las personas que depuran programas generalmente envían comandos cortos (por ejemplo, compilar un procedimiento de cinco hojas[†]) en vez de largos (por ejemplo, ordenar un archivo con un millón de registros), la computadora puede proporcionar un servicio rápido e interactivo a varios usuarios y, tal vez, también ocuparse en trabajos grandes por lotes en segundo plano, cuando la CPU estaría inactiva de otra manera. El primer sistema de tiempo compartido de propósito general, conocido como **CTSS** (*Compatible Time Sharing System*, Sistema compatible de tiempo compartido), se desarrolló en el M.I.T. en una 7094 modificada en forma especial (Corbató y colaboradores, 1962). Sin embargo, en realidad el tiempo compartido no se popularizó sino hasta que el hardware de protección necesario se empezó a utilizar ampliamente durante la tercera generación.

Después del éxito del sistema CTSS, el M.I.T., Bell Labs y General Electric (que en ese entonces era un importante fabricante de computadoras) decidieron emprender el desarrollo de una "utilería para computadora", una máquina capaz de servir a varios cientos de usuarios simultáneos de tiempo compartido. Su modelo fue el sistema de electricidad: cuando se necesita energía, sólo hay que conectar un contacto a la pared y, dentro de lo razonable, toda la energía que se requiera estará ahí. Los diseñadores del sistema conocido como **MULTICS** (*MULTiplexed Information and Computing Service*; Servicio de Información y Cómputo MULTiplexado), imaginaron una enorme máquina que proporcionaba poder de cómputo a todos los usuarios en el área de Boston. La idea de que, sólo 40 años después, se vendieran por millones máquinas 10,000 veces más rápidas que su mainframe GE-645 (a un precio muy por debajo de los 1000 dólares) era pura ciencia ficción. Algo así como la idea de que en estos días existiera un transatlántico supersónico por debajo del agua.

MULTICS fue un éxito parcial. Se diseñó para dar soporte a cientos de usuarios en una máquina que era sólo un poco más potente que una PC basada en el Intel 386, aunque tenía mucho más capacidad de E/S. Esto no es tan disparatado como parece, ya que las personas sabían cómo escribir programas pequeños y eficientes en esos días, una habilidad que se ha perdido con el tiempo. Hubo muchas razones por las que MULTICS no acaparó la atención mundial; una de ellas fue el que estaba escrito en PL/I y el compilador de PL/I se demoró por años, además de que apenas funcionaba cuando por fin llegó. Aparte de eso, MULTICS era un sistema demasiado ambicioso para su época, algo muy parecido a la máquina analítica de Charles Babbage en el siglo diecinueve.

Para resumir esta larga historia, MULTICS introdujo muchas ideas seminales en la literatura de las computadoras, pero convertirlas en un producto serio y con éxito comercial importante era algo mucho más difícil de lo que cualquiera hubiera esperado. Bell Labs se retiró del proyecto y General Electric dejó el negocio de las computadoras por completo. Sin embargo, el M.I.T. persistió y logró hacer en un momento dado que MULTICS funcionara. Al final, la compañía que compró el negocio de computadoras de GE (Honeywell) lo vendió como un producto comercial y fue instalado por cerca de 80 compañías y universidades importantes a nivel mundial. Aunque en número pequeño, los usuarios de MULTICS eran muy leales. Por ejemplo, General Motors, Ford y la Agencia de Seguridad Nacional de los Estados Unidos desconectaron sus sistemas MULTICS sólo hasta

[†] En este libro utilizaremos los términos "procedimiento", "subrutina" y "función" de manera indistinta.

finales de la década de 1990, 30 años después de su presentación en el mercado y de tratar durante años de hacer que Honeywell actualizara el hardware.

Por ahora, el concepto de una "utilería para computadora" se ha disipado, pero tal vez regrese en forma de servidores masivos de Internet centralizados a los que se conecten máquinas de usuario relativamente "tontas", donde la mayoría del trabajo se realice en los servidores grandes. Es probable que la motivación en este caso sea que la mayoría de las personas no desean administrar un sistema de cómputo cada vez más complejo y melindroso, y prefieren delegar esa tarea a un equipo de profesionales que trabajen para la compañía que opera el servidor. El comercio electrónico ya está evolucionando en esta dirección, en donde varias compañías operan centros comerciales electrónicos en servidores multiprocesador a los que se conectan las máquinas cliente simples, algo muy parecido al diseño de MULTICS.

A pesar de la carencia de éxito comercial, MULTICS tuvo una enorme influencia en los sistemas operativos subsecuentes. Se describe en varios artículos y en un libro (Corbató y colaboradores, 1972; Corbató y Vyssotsky, 1965; Daley y Dennis, 1968; Organick, 1972; y Staltzer, 1974). También tuvo (y aún tiene) un sitio Web activo, ubicado en *www.multicians.org*, con mucha información acerca del sistema, sus diseñadores y sus usuarios.

Otro desarrollo importante durante la tercera generación fue el increíble crecimiento de las minicomputadoras, empezando con la DEC PDP-1 en 1961. La PDP-1 tenía sólo 4K de palabras de 18 bits, pero a $120,000 por máquina (menos de 5 por ciento del precio de una 7094) se vendió como pan caliente. Para cierta clase de trabajo no numérico, era casi tan rápida como la 7094 y dio origen a una nueva industria. A esta minicomputadora le siguió rápidamente una serie de otras PDP (a diferencia de la familia de IBM, todas eran incompatibles), culminando con la PDP-11.

Posteriormente, Ken Thompson, uno de los científicos de cómputo en Bell Labs que trabajó en el proyecto MULTICS, encontró una pequeña minicomputadora PDP-7 que nadie estaba usando y se dispuso a escribir una versión simple de MULTICS para un solo usuario. Más adelante, este trabajo se convirtió en el sistema operativo **UNIX®**, que se hizo popular en el mundo académico, las agencias gubernamentales y muchas compañías.

La historia de UNIX ya ha sido contada en muchos otros libros (por ejemplo, Salus, 1994). En el capítulo 10 hablaremos sobre parte de esa historia. Por ahora baste con decir que, debido a que el código fuente estaba disponible ampliamente, varias organizaciones desarrollaron sus propias versiones (incompatibles entre sí), lo cual produjo un caos. Se desarrollaron dos versiones principales: **System V** de AT&T y **BSD** (*Berkeley Software Distribution*, Distribución de Software de Berkeley) de la Universidad de California en Berkeley. Estas versiones tenían también variantes menores. Para que fuera posible escribir programas que pudieran ejecutarse en cualquier sistema UNIX, el IEEE desarrolló un estándar para UNIX conocido como **POSIX**, con el que la mayoría de las versiones de UNIX actuales cumplen. POSIX define una interfaz mínima de llamadas al sistema a la que los sistemas UNIX deben conformarse. De hecho, algunos de los otros sistemas operativos también admiten ahora la interfaz POSIX.

Como agregado, vale la pena mencionar que en 1987 el autor liberó un pequeño clon de UNIX conocido como **MINIX**, con fines educativos. En cuanto a su funcionalidad, MINIX es muy similar a UNIX, incluyendo el soporte para POSIX. Desde esa época, la versión original ha evolucionado en MINIX 3, que es altamente modular y está enfocada a presentar una muy alta confiabilidad. Tiene la habilidad de detectar y reemplazar módulos con fallas o incluso inutilizables (como los dis-

positivos controladores de dispositivos de E/S) al instante, sin necesidad de reiniciar y sin perturbar a los programas en ejecución. También hay disponible un libro que describe su operación interna y contiene un listado del código fuente en un apéndice (Tanenbaum y Woodhull, 2006). El sistema MINIX 3 está disponible en forma gratuita (incluyendo todo el código fuente) a través de Internet, en *www.minix3.org*.

El deseo de una versión de producción (en vez de educativa) gratuita de MINIX llevó a un estudiante finlandés, llamado Linus Torvalds, a escribir **Linux**. Este sistema estaba inspirado por MINIX, además de que fue desarrollado en este sistema y originalmente ofrecía soporte para varias características de MINIX (por ejemplo, el sistema de archivos de MINIX). Desde entonces se ha extendido en muchas formas, pero todavía retiene cierta parte de su estructura subyacente común para MINIX y UNIX. Los lectores interesados en una historia detallada sobre Linux y el movimiento de código fuente abierto tal vez deseen leer el libro de Glyn Moody (2001). La mayor parte de lo que se haya dicho acerca de UNIX en este libro se aplica también a System V, MINIX, Linux y otras versiones y clones de UNIX.

1.2.4 La cuarta generación (1980 a la fecha): las computadoras personales

Con el desarrollo de los circuitos LSI (*Large Scale Integration*, Integración a gran escala), que contienen miles de transistores en un centímetro cuadrado de silicio (chip), nació la era de la computadora personal. En términos de arquitectura, las computadoras personales (que al principio eran conocidas como **microcomputadoras**) no eran del todo distintas de las minicomputadoras de la clase PDP-11, pero en términos de precio sin duda eran distintas. Mientras que la minicomputadora hizo posible que un departamento en una compañía o universidad tuviera su propia computadora, el chip microprocesador logró que un individuo tuviera su propia computadora personal.

Cuando Intel presentó el microprocesador 8080 en 1974 (la primera CPU de 8 bits de propósito general), deseaba un sistema operativo, en parte para poder probarlo. Intel pidió a uno de sus consultores, Gary Kildall, que escribiera uno. Kildall y un amigo construyeron primero un dispositivo controlador para el disco flexible de 8 pulgadas de Shugart Associates que recién había sido sacado al mercado, y conectaron el disco flexible con el 8080, con lo cual produjeron la primera microcomputadora con un disco. Después Kildall escribió un sistema operativo basado en disco conocido como **CP/M** (*Control Program for Microcomputers*; Programa de Control para Microcomputadoras) para esta CPU. Como Intel no pensó que las microcomputadoras basadas en disco tuvieran mucho futuro, cuando Kildall pidió los derechos para CP/M, Intel le concedió su petición. Después Kildall formó una compañía llamada Digital Research para desarrollar y vender el CP/M.

En 1977, Digital Research rediseñó el CP/M para adaptarlo de manera que se pudiera ejecutar en todas las microcomputadoras que utilizaban los chips 8080, Zilog Z80 y otros. Se escribieron muchos programas de aplicación para ejecutarse en CP/M, lo cual le permitió dominar por completo el mundo de la microcomputación durante un tiempo aproximado de 5 años.

A principios de la década de 1980, IBM diseñó la IBM PC y buscó software para ejecutarlo en ella. La gente de IBM se puso en contacto con Bill Gates para obtener una licencia de uso de su intérprete de BASIC. También le preguntaron si sabía de un sistema operativo que se ejecutara en la PC. Gates sugirió a IBM que se pusiera en contacto con Digital Research, que en ese entonces era la compañía con dominio mundial en el área de sistemas operativos. Kildall rehusó a reunirse con IBM y envió a uno de sus subordinados, a lo cual se le considera sin duda la peor decisión de negocios de la historia. Para empeorar más aún las cosas, su abogado se rehusó a firmar el contrato de no divulgación de IBM sobre la PC, que no se había anunciado todavía. IBM regresó con Gates para ver si podía proveerles un sistema operativo.

Cuando IBM regresó, Gates se había enterado de que un fabricante local de computadoras, Seattle Computer Products, tenía un sistema operativo adecuado conocido como **DOS** (*Disk Operating System*; Sistema Operativo en Disco). Se acercó a ellos y les ofreció comprarlo (supuestamente por 75,000 dólares), a lo cual ellos accedieron de buena manera. Después Gates ofreció a IBM un paquete con DOS/BASIC, el cual aceptó. IBM quería ciertas modificaciones, por lo que Gates contrató a la persona que escribió el DOS, Tim Paterson, como empleado de su recién creada empresa de nombre Microsoft, para que las llevara a cabo. El sistema rediseñado cambió su nombre a **MS-DOS** (*Microsoft Disk Operating System;* Sistema Operativo en Disco de MicroSoft) y rápidamente llegó a dominar el mercado de la IBM PC. Un factor clave aquí fue la decisión de Gates (que en retrospectiva, fue en extremo inteligente) de vender MS-DOS a las empresas de computadoras para que lo incluyeran con su hardware, en comparación con el intento de Kildall por vender CP/M a los usuarios finales, uno a la vez (por lo menos al principio). Después de que se supo todo esto, Kildall murió en forma repentina e inesperada debido a causas que aún no han sido reveladas por completo.

Para cuando salió al mercado en 1983 la IBM PC/AT, sucesora de la IBM PC, con la CPU Intel 80286, MS-DOS estaba muy afianzado y CP/M daba sus últimos suspiros. Más adelante, MS-DOS se utilizó ampliamente en el 80386 y 80486. Aunque la versión inicial de MS-DOS era bastante primitiva, las versiones siguientes tenían características más avanzadas, incluyendo muchas que se tomaron de UNIX. (Microsoft estaba muy al tanto de UNIX e inclusive vendía una versión de este sistema para microcomputadora, conocida como XENIX, durante los primeros años de la compañía).

CP/M, MS-DOS y otros sistemas operativos para las primeras microcomputadoras se basaban en que los usuarios escribieran los comandos mediante el teclado. Con el tiempo esto cambió debido a la investigación realizada por Doug Engelbart en el Stanford Research Institute en la década de 1960. Engelbart inventó la **Interfaz Gráfica de Usuario GUI**, completa con ventanas, iconos, menús y ratón. Los investigadores en Xerox PARC adoptaron estas ideas y las incorporaron en las máquinas que construyeron.

Un día, Steve Jobs, que fue co-inventor de la computadora Apple en su cochera, visitó PARC, vio una GUI y de inmediato se dio cuenta de su valor potencial, algo que la administración de Xerox no hizo. Esta equivocación estratégica de gigantescas proporciones condujo a un libro titulado *Fumbling the Future* (Smith y Alexander, 1988). Posteriormente, Jobs emprendió el proyecto de construir una Apple con una GUI. Este proyecto culminó en Lisa, que era demasiado costosa y fracasó comercialmente. El segundo intento de Jobs, la Apple Macintosh, fue un enorme éxito, no sólo debido a que era mucho más económica que Lisa, sino también porque era **amigable para el**

usuario (*user friendly*), lo cual significaba que estaba diseñada para los usuarios que no sólo no sabían nada acerca de las computadoras, sino que además no tenían ninguna intención de aprender. En el mundo creativo del diseño gráfico, la fotografía digital profesional y la producción de video digital profesional, las Macintosh son ampliamente utilizadas y sus usuarios son muy entusiastas sobre ellas.

Cuando Microsoft decidió crear un sucesor para el MS-DOS estaba fuertemente influenciado por el éxito de la Macintosh. Produjo un sistema basado en GUI llamado Windows, el cual en un principio se ejecutaba encima del MS-DOS (es decir, era más como un shell que un verdadero sistema operativo). Durante cerca de 10 años, de 1985 a 1995, Windows fue sólo un entorno gráfico encima de MS-DOS. Sin embargo, a partir de 1995 se liberó una versión independiente de Windows, conocida como Windows 95, que incorporaba muchas características de los sistemas operativos y utilizaba el sistema MS-DOS subyacente sólo para iniciar y ejecutar programas de MS-DOS antiguos. En 1998, se liberó una versión ligeramente modificada de este sistema, conocida como Windows 98. Sin embargo, tanto Windows 95 como Windows 98 aún contenían una gran cantidad de lenguaje ensamblador para los procesadores Intel de 16 bits.

Otro de los sistemas operativos de Microsoft es **Windows NT** (NT significa Nueva Tecnología), que es compatible con Windows 95 en cierto nivel, pero fue completamente rediseñado en su interior. Es un sistema completo de 32 bits. El diseñador en jefe de Windows NT fue David Cutler, quien también fue uno de los diseñadores del sistema operativo VMS de VAX, por lo que hay algunas ideas de VMS presentes en NT. De hecho, había tantas ideas de VMS presentes que el propietario de VMS (DEC) demandó a Microsoft. El caso se resolvió en la corte por una cantidad de muchos dígitos. Microsoft esperaba que la primera versión de NT acabara con MS-DOS y todas las demás versiones de Windows, ya que era un sistema muy superior, pero fracasó. No fue sino hasta Windows NT 4.0 que finalmente empezó a tener éxito, en especial en las redes corporativas. La versión 5 de Windows NT cambió su nombre a Windows 2000 a principios de 1999. Estaba destinada a ser el sucesor de Windows 98 y de Windows NT 4.0.

Esto tampoco funcionó como se esperaba, por lo que Microsoft preparó otra versión de Windows 98 conocida como **Windows Me** (*Millennium edition*). En el 2001 se liberó una versión ligeramente actualizada de Windows 2000, conocida como Windows XP. Esa versión duró mucho más en el mercado (6 años), reemplazando a casi todas las versiones anteriores de Windows. Después, en enero del 2007 Microsoft liberó el sucesor para Windows XP, conocido como Windows Vista. Tenía una interfaz gráfica nueva, Aero, y muchos programas de usuario nuevos o actualizados. Microsoft espera que sustituya a Windows XP por completo, pero este proceso podría durar casi toda una década.

El otro competidor importante en el mundo de las computadoras personales es UNIX (y todas sus variantes). UNIX es más fuerte en los servidores tanto de redes como empresariales, pero también está cada vez más presente en las computadoras de escritorio, en especial en los países que se desarrollan con rapidez, como India y China. En las computadoras basadas en Pentium, Linux se está convirtiendo en una alternativa popular para Windows entre los estudiantes y cada vez más usuarios corporativos. Como agregado, a lo largo de este libro utilizaremos el término "Pentium" para denotar al Pentium I, II, III y 4, así como sus sucesores tales como el Core 2 Duo. El término **x86** también se utiliza algunas veces para indicar el rango completo de CPU Intel partiendo desde el 8086, mientras que utilizaremos "Pentium" para indicar todas las CPU desde el

Pentium I. Admitimos que este término no es perfecto, pero no hay disponible uno mejor. Uno se pregunta qué genio de mercadotecnia en Intel desperdició una marca comercial (Pentium) que la mitad del mundo conocía bien y respetaba, sustituyéndola con términos como "Core 2 duo" que muy pocas personas comprenden; ¿qué significan "2" y "duo"? Tal vez "Pentium 5" (o "Pentium 5 dual core", etc.) eran demasiado difíciles de recordar. **FreeBSD** es también un derivado popular de UNIX, que se originó del proyecto BSD en Berkeley. Todas la computadoras modernas Macintosh utilizan una versión modificada de FreeBSD. UNIX también es estándar en las estaciones de trabajo operadas por chips RISC de alto rendimiento, como los que venden Hewlett-Packard y Sun Microsystems.

Muchos usuarios de UNIX, en especial los programadores experimentados, prefieren una interfaz de línea de comandos a una GUI, por lo que casi todos los sistemas UNIX presentan un sistema de ventanas llamado **X Window System** (también conocido como **X11**), producido en el M.I.T. Este sistema se encarga de la administración básica de las ventanas y permite a los usuarios crear, eliminar, desplazar y cambiar el tamaño de las ventanas mediante el uso de un ratón. Con frecuencia hay disponible una GUI completa, como **Gnome** o **KDE**, para ejecutarse encima de X11, lo cual proporciona a UNIX una apariencia parecida a la Macintosh o a Microsoft Windows, para aquellos usuarios de UNIX que desean algo así.

Un interesante desarrollo que empezó a surgir a mediados de la década de 1980 es el crecimiento de las redes de computadoras personales que ejecutan **sistemas operativos en red** y **sistemas operativos distribuidos** (Tanenbaum y Van Steen, 2007). En un sistema operativo en red, los usuarios están conscientes de la existencia de varias computadoras, y pueden iniciar sesión en equipos remotos y copiar archivos de un equipo a otro. Cada equipo ejecuta su propio sistema operativo local y tiene su propio usuario (o usuarios) local.

Los sistemas operativos en red no son fundamentalmente distintos de los sistemas operativos con un solo procesador. Es obvio que necesitan un dispositivo controlador de interfaz de red y cierto software de bajo nivel para controlarlo, así como programas para lograr el inicio de una sesión remota y el acceso remoto a los archivos, pero estas adiciones no cambian la estructura esencial del sistema operativo.

En contraste, un sistema operativo distribuido se presenta a sus usuarios en forma de un sistema tradicional con un procesador, aun cuando en realidad está compuesto de varios procesadores. Los usuarios no tienen que saber en dónde se están ejecutando sus programas o en dónde se encuentran sus archivos; el sistema operativo se encarga de todo esto de manera automática y eficiente.

Los verdaderos sistemas operativos distribuidos requieren algo más que sólo agregar un poco de código a un sistema operativo con un solo procesador, ya que los sistemas distribuidos y los centralizados difieren en varios puntos críticos. Por ejemplo, los sistemas distribuidos permiten con frecuencia que las aplicaciones se ejecuten en varios procesadores al mismo tiempo, lo que requiere algoritmos de planificación del procesador más complejos para poder optimizar la cantidad de paralelismo.

Los retrasos de comunicación dentro de la red implican a menudo que estos (y otros) algoritmos deban ejecutarse con información incompleta, obsoleta o incluso incorrecta. Esta situación es muy distinta a la de un sistema con un solo procesador, donde el sistema operativo tiene información completa acerca del estado del sistema.

1.3 REVISIÓN DEL HARDWARE DE COMPUTADORA

Un sistema operativo está íntimamente relacionado con el hardware de la computadora sobre la que se ejecuta. Extiende el conjunto de instrucciones de la computadora y administra sus recursos. Para trabajar debe conocer muy bien el hardware, por lo menos en lo que respecta a cómo aparece para el programador. Por esta razón, revisaremos brevemente el hardware de computadora como se encuentra en las computadoras personales modernas. Después de eso, podemos empezar a entrar en los detalles acerca de qué hacen los sistemas operativos y cómo funcionan.

Conceptualmente, una computadora personal simple se puede abstraer mediante un modelo como el de la figura 1-6. La CPU, la memoria y los dispositivos de E/S están conectados mediante un bus del sistema y se comunican entre sí a través de este bus. Las computadoras personales modernas tienen una estructura más complicada en la que intervienen varios buses, los cuales analizaremos más adelante; por ahora basta con este modelo. En las siguientes secciones analizaremos brevemente estos componentes y examinaremos algunas de las cuestiones de hardware que son de relevancia para los diseñadores de sistemas operativos; sobra decir que será un resumen muy compacto. Se han escrito muchos libros acerca del tema del hardware de computadora y su organización. Dos libros muy conocidos acerca de este tema son el de Tanenbaum (2006) y el de Patterson y Hennessy (2004).

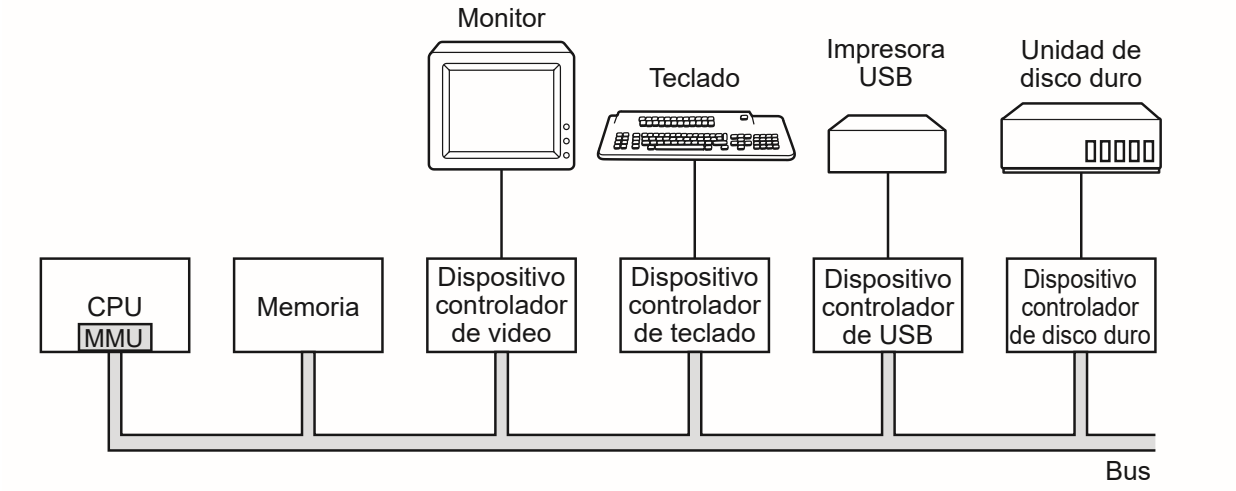

Figura 1-6. Algunos de los componentes de una computadora personal simple.

1.3.1 Procesadores

El “cerebro” de la computadora es la CPU, que obtiene las instrucciones de la memoria y las ejecuta. El ciclo básico de toda CPU es obtener la primera instrucción de memoria, decodificarla para determinar su tipo y operandos, ejecutarla y después obtener, decodificar y ejecutar las instrucciones subsiguientes. El ciclo se repite hasta que el programa termina. De esta forma se ejecutan los programas.

Cada CPU tiene un conjunto específico de instrucciones que puede ejecutar. Así, un Pentium no puede ejecutar programas de SPARC y un SPARC no puede ejecutar programas de Pentium. Como el acceso a la memoria para obtener una instrucción o palabra de datos requiere mucho más tiempo que ejecutar una instrucción, todas las CPU contienen ciertos registros en su interior para contener las variables clave y los resultados temporales. Debido a esto, el conjunto de instrucciones generalmente contiene instrucciones para cargar una palabra de memoria en un registro y almacenar una palabra de un registro en la memoria. Otras instrucciones combinan dos operandos de los registros, la memoria o ambos en un solo resultado, como la operación de sumar dos palabras y almacenar el resultado en un registro o la memoria.

Además de los registros generales utilizados para contener variables y resultados temporales, la mayoría de las computadoras tienen varios registros especiales que están visibles para el programador. Uno de ellos es el **contador de programa** (*program counter*), el cual contiene la dirección de memoria de la siguiente instrucción a obtener. Una vez que se obtiene esa instrucción, el contador de programa se actualiza para apuntar a la siguiente.

Otro registro es el **apuntador de pila** (*stack pointer*), el cual apunta a la parte superior de la pila (*stack*) actual en la memoria. La pila contiene un conjunto de valores por cada procedimiento al que se ha entrado pero del que todavía no se ha salido. El conjunto de valores en la pila por procedimiento contiene los parámetros de entrada, las variables locales y las variables temporales que no se mantienen en los registros.

Otro de los registros es **PSW** (*Program Status Word*; Palabra de estado del programa). Este registro contiene los bits de código de condición, que se asignan cada vez que se ejecutan las instrucciones de comparación, la prioridad de la CPU, el modo (usuario o kernel) y varios otros bits de control. Los programas de usuario pueden leer normalmente todo el PSW pero por lo general sólo pueden escribir en algunos de sus campos. El PSW juega un papel importante en las llamadas al sistema y en las operaciones de E/S.

El sistema operativo debe estar al tanto de todos los registros. Cuando la CPU se multiplexa en el tiempo, el sistema operativo detiene con frecuencia el programa en ejecución para (re)iniciar otro. Cada vez que detiene un programa en ejecución, el sistema operativo debe guardar todos los registros para poder restaurarlos cuando el programa continúe su ejecución.

Para mejorar el rendimiento, los diseñadores de CPUs abandonaron desde hace mucho tiempo el modelo de obtener, decodificar y ejecutar una instrucción a la vez. Muchas CPUs modernas cuentan con medios para ejecutar más de una instrucción al mismo tiempo. Por ejemplo, una CPU podría tener unidades separadas de obtención, decodificación y ejecución, de manera que mientras se encuentra ejecutando la instrucción n, también podría estar decodificando la instrucción $n + 1$ y obteniendo la instrucción $n + 2$. A dicha organización se le conoce como **canalización** (*pipeline*); la figura 1-7(a) ilustra una canalización de tres etapas. El uso de canalizaciones más grandes es común. En la mayoría de los diseños de canalizaciones, una vez que se ha obtenido una instrucción y se coloca en la canalización, se debe ejecutar aún si la instrucción anterior era una bifurcación condicional que se tomó. Las canalizaciones producen grandes dolores de cabeza a los programadores de compiladores y de sistemas operativos, ya que quedan al descubierto las complejidades de la máquina subyacente.

Aún más avanzada que el diseño de una canalización es la CPU **superescalar**, que se muestra en la figura 1-7(b). En este diseño hay varias unidades de ejecución; por ejemplo, una para la arit-

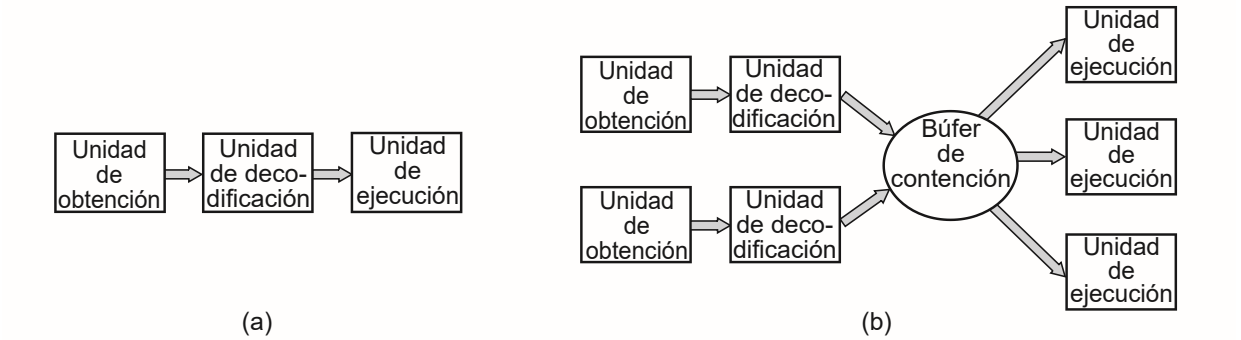

Figura 1-7. (a) Canalización de tres etapas; (b) CPU superescalar.

mética de enteros, una para la aritmética de punto flotante y otra para las operaciones Booleanas. Dos o más instrucciones se obtienen a la vez, se decodifican y se vacían en un búfer de contención hasta que puedan ejecutarse. Tan pronto como una unidad de ejecución se encuentre libre, busca en el búfer de contención para ver si hay una instrucción que pueda manejar; de ser así, saca la instrucción del búfer y la ejecuta. Una consecuencia de este diseño es que con frecuencia las instrucciones del programa se ejecutan en forma desordenada. En gran parte, es responsabilidad del hardware asegurarse de que el resultado producido sea el mismo que hubiera producido una implementación secuencial, pero una cantidad molesta de complejidad es impuesta al sistema operativo, como veremos más adelante.

La mayoría de las CPU, con excepción de las extremadamente simples que se utilizan en los sistemas integrados, tienen dos modos: modo kernel y modo usuario, como dijimos antes. Por lo general, un bit en el PSW controla el modo. Al operar en modo kernel, la CPU puede ejecutar cualquier instrucción de su conjunto de instrucciones y utilizar todas las características del hardware. El sistema operativo opera en modo kernel, lo cual le da acceso al hardware completo.

En contraste, los programas de usuario operan en modo de usuario, el cual les permite ejecutar sólo un subconjunto de las instrucciones y les da acceso sólo a un subconjunto de las características. En general, no se permiten las instrucciones que implican operaciones de E/S y protección de la memoria en el modo usuario. Desde luego que también está prohibido asignar el bit de modo del PSW para entrar al modo kernel.

Para obtener servicios del sistema operativo, un programa usuario debe lanzar una **llamada al sistema** (*system call*), la cual se atrapa en el kernel e invoca al sistema operativo. La instrucción `TRAP` cambia del modo usuario al modo kernel e inicia el sistema operativo. Cuando se ha completado el trabajo, el control se devuelve al programa de usuario en la instrucción que va después de la llamada al sistema. Más adelante en este capítulo explicaremos los detalles acerca del mecanismo de llamadas al sistema, pero por ahora piense en él como un tipo especial de instrucción de llamada a procedimiento que tiene la propiedad adicional de cambiar del modo usuario al modo kernel. Como indicación sobre la tipografía, utilizaremos el tipo de letra Helvetica en minúsculas para indicar las llamadas al sistema en el texto del libro, como se muestra a continuación: `read`.

Vale la pena indicar que las computadoras tienen otros traps aparte de la instrucción para ejecutar una llamada al sistema. La mayoría de los demás traps son producidos por el hardware para advertir acerca de una situación excepcional, tal como un intento de dividir entre 0 o un subdesbor-

damiento de punto flotante. En cualquier caso, el sistema operativo obtiene el control y debe decidir qué hacer. Algunas veces el programa debe terminarse con un error. Otras veces el error se puede ignorar (un número que provoque un subdesbordamiento puede establecerse en 0). Por último, cuando el programa anuncia por adelantado que desea manejar ciertos tipos de condiciones, puede devolvérsele el control para dejarlo resolver el problema.

Chips con multihilamiento y multinúcleo

La ley de Moore establece que el número de transistores en un chip se duplica cada 18 meses. Esta "ley" no es ningún tipo de ley de física, como la de la conservación del momento, sino una observación hecha por Gordon Moore, cofundador de Intel, acerca de la velocidad con la que los ingenieros de procesos en las compañías de semiconductores pueden reducir sus transistores. La ley de Moore ha estado vigente durante tres décadas hasta hoy y se espera que siga así durante al menos una década más.

La abundancia de transistores está ocasionando un problema: ¿qué se debe hacer con todos ellos? En párrafos anteriores vimos una solución: las arquitecturas superescalares, con múltiples unidades funcionales. Pero a medida que se incrementa el número de transistores, se puede hacer todavía más. Algo obvio por hacer es colocar cachés más grandes en el chip de la CPU y eso está ocurriendo, pero en cierto momento se llega al punto de rendimiento decreciente.

El siguiente paso obvio es multiplicar no sólo las unidades funcionales, sino también parte de la lógica de control. El Pentium 4 y algunos otros chips de CPU tienen esta propiedad, conocida como **multihilamiento** (*multithreading*) o **hiperhilamiento** (*hyperthreading*) (el nombre que puso Intel al multihilamiento). Para una primera aproximación, lo que hace es permitir que la CPU contenga el estado de dos hilos de ejecución (*threads*) distintos y luego alterne entre uno y otro con una escala de tiempo en nanosegundos (un hilo de ejecución es algo así como un proceso ligero, que a su vez es un programa en ejecución; veremos los detalles sobre esto en el capítulo 2). Por ejemplo, si uno de los procesos necesita leer una palabra de memoria (que requiere muchos ciclos de reloj), una CPU con multihilamiento puede cambiar a otro hilo. El multihilamiento no ofrece un verdadero paralelismo. Sólo hay un proceso en ejecución a la vez, pero el tiempo de cambio entre un hilo y otro se reduce al orden de un nanosegundo.

El multihilamiento tiene consecuencias para el sistema operativo, debido a que cada hilo aparece para el sistema operativo como una CPU separada. Considere un sistema con dos CPU reales, cada una con dos hilos. El sistema operativo verá esto como si hubiera cuatro CPU. Si hay suficiente trabajo sólo para mantener ocupadas dos CPU en cierto punto en el tiempo, podría planificar de manera inadvertida dos hilos en la misma CPU, mientras que la otra CPU estaría completamente inactiva. Esta elección es mucho menos eficiente que utilizar un hilo en cada CPU. El sucesor del Pentium 4, conocido como arquitectura Core (y también Core 2), no tiene hiperhilamiento, pero Intel ha anunciado que el sucesor del Core lo tendrá nuevamente.

Más allá del multihilamiento, tenemos chips de CPU con dos, cuatro o más procesadores completos, o **núcleos** (*cores*) en su interior. Los chips de multinúcleo (*multicore*) de la figura 1-8 contienen efectivamente cuatro minichips en su interior, cada uno con su propia CPU independiente (más adelante hablaremos sobre las cachés). Para hacer uso de dicho chip multinúcleo se requiere en definitiva un sistema operativo multiprocesador.

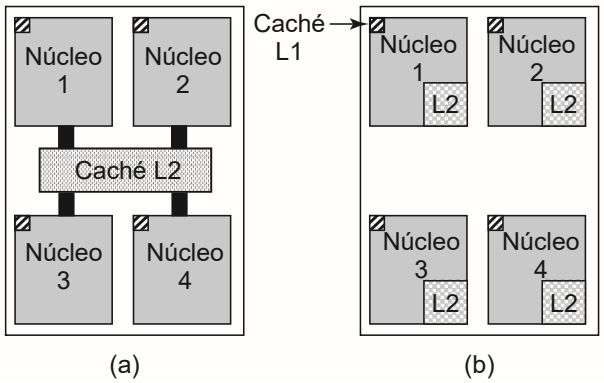

Figura 1-8. (a) Un chip de cuatro núcleos (*quad-core*) con una caché L2 compartida. (b) Un chip de cuatro núcleos con cachés L2 separadas.

1.3.2 Memoria

El segundo componente importante en cualquier computadora es la memoria. En teoría, una memoria debe ser en extremo rápida (más rápida que la velocidad de ejecución de una instrucción, de manera que la memoria no detenga a la CPU), de gran tamaño y muy económica. Ninguna tecnología en la actualidad cumple con todos estos objetivos, por lo que se adopta una solución distinta. El sistema de memoria está construido como una jerarquía de capas, como se muestra en la figura 1-9. Las capas superiores tienen mayor velocidad, menor capacidad y mayor costo por bit que las capas inferiores, a menudo por factores de mil millones o más.

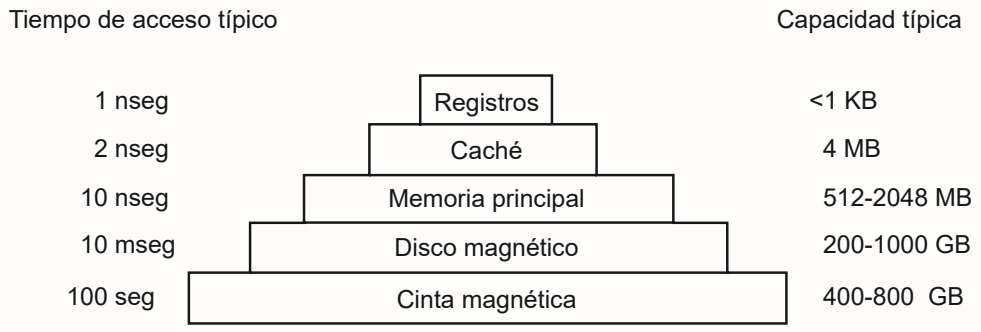

Figura 1-9. Una común jerarquía de memoria. Los números son sólo aproximaciones.

La capa superior consiste en los registros internos de la CPU. Están compuestos del mismo material que la CPU y, por ende, tienen la misma rapidez. En consecuencia no hay retraso a la hora de utilizarlos. La capacidad de almacenamiento disponible en estos registros es generalmente de 32 × 32 bits en una CPU de 32 bits y de 64 × 64 bits en una CPU de 64 bits. Menos de 1 KB en ambos casos. Los programas deben administrar los registros (es decir, decidir qué deben guardar en ellos) por su cuenta, en el software.

El siguiente nivel es la memoria caché, que el hardware controla de manera parcial. La memoria principal se divide en **líneas de caché**, que por lo general son de 64 bytes, con direcciones de 0 a 63 en la línea de caché 0, direcciones de 64 a 127 en la línea de caché 1 y así sucesivamente. Las líneas de caché que se utilizan con más frecuencia se mantienen en una caché de alta velocidad, ubicada dentro o muy cerca de la CPU. Cuando el programa necesita leer una palabra de memoria, el hardware de la caché comprueba si la línea que se requiere se encuentra en la caché. Si es así (a lo cual se le conoce como **acierto de caché**), la petición de la caché se cumple y no se envía una petición de memoria a través del bus hacia la memoria principal. Los aciertos de caché por lo general requieren un tiempo aproximado de dos ciclos de reloj. Los fallos de caché tienen que ir a memoria, con un castigo considerable de tiempo. La memoria caché está limitada en tamaño debido a su alto costo. Algunas máquinas tienen dos o incluso tres niveles de caché, cada una más lenta y más grande que la anterior.

El uso de cachés juega un papel importante en muchas áreas de las ciencias computacionales, no sólo en la caché de las líneas de RAM. Cada vez que hay un recurso extenso que se puede dividir en piezas, algunas de las cuales se utilizan con mucho más frecuencia que otras, a menudo se invoca a la caché para mejorar el rendimiento. Los sistemas operativos la utilizan todo el tiempo. Por ejemplo, la mayoría de los sistemas operativos mantienen (piezas de) los archivos que se utilizan con frecuencia en la memoria principal para evitar tener que obtenerlos del disco en forma repetida. De manera similar, los resultados de convertir nombres de rutas extensas tales como

/home/ast/proyectos/minix3/src/kernel/reloj.c

a la dirección en disco donde se encuentra el archivo, se pueden colocar en la caché para evitar búsquedas repetidas. Por último, cuando una dirección de una página Web (URL) se convierte en una dirección de red (dirección IP), el resultado se puede poner en la caché para usarlo a futuro (existen muchos otros usos).

En cualquier sistema de caché surgen con rapidez varias preguntas, incluyendo:

1. Cuándo se debe poner un nuevo elemento en la caché.
2. En qué línea de caché se debe poner el nuevo elemento.
3. Qué elemento se debe eliminar de la caché cuando se necesita una posición.
4. Dónde se debe poner un elemento recién desalojado en la memoria de mayor tamaño.

No todas las preguntas son relevantes para cada situación de uso de la caché. Para poner líneas de la memoria principal en la caché de la CPU, por lo general se introduce un nuevo elemento en cada fallo de caché. La línea de caché a utilizar se calcula generalmente mediante el uso de algunos de los bits de mayor orden de la dirección de memoria a la que se hace referencia. Por ejemplo, con 4096 líneas de caché de 64 bytes y direcciones de 32 bits, los bits del 6 al 17 podrían utilizarse para especificar la línea de caché, siendo los bits del 0 al 5 el byte dentro de la línea de la caché. En este caso, el elemento a quitar es el mismo en el que se colocan los nuevos datos, pero en otros sistemas podría ser otro. Por último, cuando se vuelve a escribir una línea de caché en la memoria principal (si se ha modificado desde la última vez que se puso en caché), la posición en memoria donde se debe volver a escribir se determina únicamente por la dirección en cuestión.

Las cachés son una idea tan útil que las CPUs modernas tienen dos de ellas. La **caché L1** o de primer nivel está siempre dentro de la CPU, y por lo general alimenta las instrucciones decodificadas al motor de ejecución de la CPU. La mayoría de los chips tienen una segunda caché L1 para las

palabras de datos que se utilizan con frecuencia. Por lo general, las cachés L1 son de 16 KB cada una. Además, a menudo hay una segunda caché, conocida como **caché L2**, que contiene varios megabytes de palabras de memoria utilizadas recientemente. La diferencia entre las cachés L1 y L2 está en la velocidad. El acceso a la caché L1 se realiza sin ningún retraso, mientras que el acceso a la caché L2 requiere un retraso de uno o dos ciclos de reloj.

En los chips multinúcleo, los diseñadores deben decidir dónde deben colocar las cachés. En la figura 1-8(a) hay una sola caché L2 compartida por todos los núcleos; esta metodología se utiliza en los chips multinúcleo de Intel. En contraste, en la figura 1-8(b) cada núcleo tiene su propia caché L2; AMD utiliza esta metodología. Cada estrategia tiene sus pros y sus contras. Por ejemplo, la caché L2 compartida de Intel requiere un dispositivo controlador de caché más complicado, pero la manera en que AMD utiliza la caché hace más difícil la labor de mantener las cachés L2 consistentes.

La memoria principal viene a continuación en la jerarquía de la figura 1-9. Es el "caballo de batalla" del sistema de memoria. Por lo general a la memoria principal se le conoce como **RAM** (*Random Access Memory*, Memoria de Acceso Aleatorio). Los usuarios de computadora antiguos algunas veces la llaman **memoria de núcleo** debido a que las computadoras en las décadas de 1950 y 1960 utilizaban pequeños núcleos de ferrita magnetizables para la memoria principal. En la actualidad, las memorias contienen desde cientos de megabytes hasta varios gigabytes y su tamaño aumenta con rapidez. Todas las peticiones de la CPU que no se puedan satisfacer desde la caché pasan a la memoria principal.

Además de la memoria principal, muchas computadoras tienen una pequeña cantidad de memoria de acceso aleatorio no volátil. A diferencia de la RAM, la memoria no volátil no pierde su contenido cuando se desconecta la energía. La **ROM** (*Read Only Memory*, Memoria de sólo lectura) se programa en la fábrica y no puede modificarse después. Es rápida y económica. En algunas computadoras, el cargador de arranque (*bootstrap loader*) que se utiliza para iniciar la computadora está contenido en la ROM. Además, algunas tarjetas de E/S vienen con ROM para manejar el control de los dispositivos de bajo nivel.

La **EEPROM** (*Electrically Erasable PROM*, PROM eléctricamente borrable) y la **memoria flash** son también no volátiles, pero en contraste con la ROM se pueden borrar y volver a escribir datos en ellas. Sin embargo, para escribir en este tipo de memorias se requiere mucho más tiempo que para escribir en la RAM, por lo cual se utilizan en la misma forma que la ROM, sólo con la característica adicional de que ahora es posible corregir los errores en los programas que contienen, mediante la acción de volver a escribir datos en ellas en el campo de trabajo.

La memoria flash también se utiliza comúnmente como el medio de almacenamiento en los dispositivos electrónicos portátiles. Sirve como película en las cámaras digitales y como el disco en los reproductores de música portátiles, para nombrar sólo dos usos. La memoria flash se encuentra entre la RAM y el disco en cuanto a su velocidad. Además, a diferencia de la memoria en disco, si se borra demasiadas veces, se desgasta.

Otro tipo más de memoria es CMOS, la cual es volátil. Muchas computadoras utilizan memoria CMOS para guardar la fecha y hora actuales. La memoria CMOS y el circuito de reloj que incrementa la hora en esta memoria están energizados por una pequeña batería, por lo que la hora se actualiza en forma correcta aun cuando la computadora se encuentre desconectada. La memoria CMOS también puede contener los parámetros de configuración, como el disco del cual se debe iniciar el sistema. Se utiliza CMOS debido a que consume tan poca energía que la batería instalada en

la fábrica dura varios años. Sin embargo, cuando empieza a fallar puede parecer como si la computadora tuviera la enfermedad de Alzheimer, olvidando cosas que ha sabido durante años, como desde cuál disco duro se debe iniciar el sistema.

1.3.3 Discos

El siguiente lugar en la jerarquía corresponde al disco magnético (disco duro). El almacenamiento en disco es dos órdenes de magnitud más económico que la RAM por cada bit, y a menudo es dos órdenes de magnitud más grande en tamaño también. El único problema es que el tiempo para acceder en forma aleatoria a los datos en ella es de cerca de tres órdenes de magnitud más lento. Esta baja velocidad se debe al hecho de que un disco es un dispositivo mecánico, como se muestra en la figura 1-10.

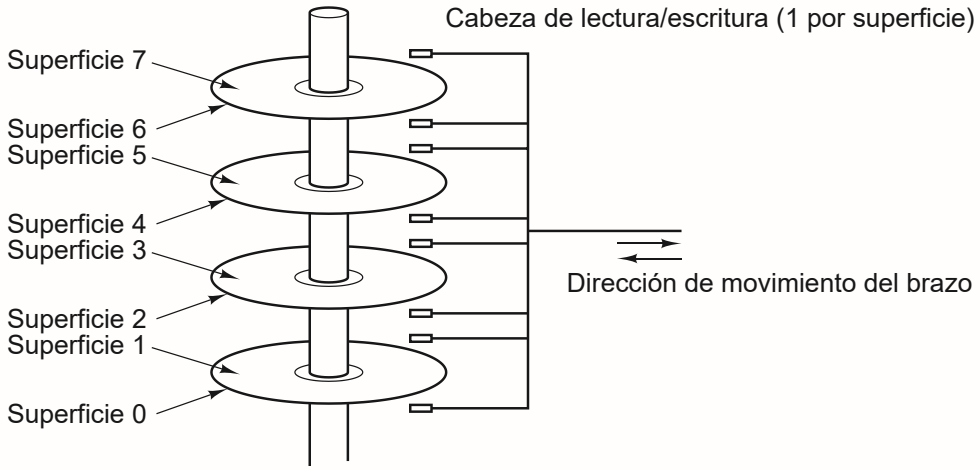

Figura 1-10. Estructura de una unidad de disco.

Un disco consiste en uno o más platos que giran a 5400, 7200 o 10,800 rpm. Un brazo mecánico, con un punto de giro colocado en una esquina, se mueve sobre los platos de manera similar al brazo de la aguja en un viejo tocadiscos. La información se escribe en el disco en una serie de círculos concéntricos. En cualquier posición dada del brazo, cada una de las cabezas puede leer una región anular conocida como **pista** (*track*). En conjunto, todas las pistas para una posición dada del brazo forman un **cilindro** (*cylinder*).

Cada pista se divide en cierto número de sectores, por lo general de 512 bytes por sector. En los discos modernos, los cilindros exteriores contienen más sectores que los interiores. Para desplazar el brazo de un cilindro al siguiente se requiere aproximadamente 1 milisegundo. Para desplazar el brazo a un cilindro aleatoriamente se requieren por lo general de 5 a 10 milisegundos, dependiendo de la unidad. Una vez que el brazo se encuentra en la pista correcta, la unidad debe esperar a que el sector necesario gire hacia abajo de la cabeza, con un retraso adicional de 5 a 10 milisegundos, dependiendo de las rpm de la unidad. Una vez que el sector está bajo la cabeza, la lectura o escritura ocurre a una velocidad de 50 MB/seg en los discos de bajo rendimiento hasta de 160 MB/seg en los discos más rápidos.

Muchas computadoras presentan un esquema conocido como **memoria virtual** (*virtual memory*), el cual describiremos hasta cierto punto en el capítulo 3. Este esquema hace posible la

ejecución de programas más grandes que la memoria física al colocarlos en el disco y utilizar la memoria principal como un tipo de caché para las partes que se ejecutan con más frecuencia. Este esquema requiere la reasignación de direcciones de memoria al instante, para convertir la dirección que el programa generó en la dirección física en la RAM en donde se encuentra la palabra. Esta asignación se realiza mediante una parte de la CPU conocida como **MMU** (*Memory Management Unit*, Unidad de Administración de Memoria), como se muestra en la figura 1-6.

La presencia de la caché y la MMU pueden tener un gran impacto en el rendimiento. En un sistema de multiprogramación, al cambiar de un programa a otro (lo que se conoce comúnmente como **cambio de contexto** o *context switch*), puede ser necesario vaciar todos los bloques modificados de la caché y modificar los registros de asignación en la MMU. Ambas operaciones son costosas y los programadores se esfuerzan bastante por evitarlas. Más adelante veremos algunas de las consecuencias de sus tácticas.

1.3.4 Cintas

La última capa de la jerarquía en la memoria es la cinta magnética. Este medio se utiliza con frecuencia como respaldo para el almacenamiento en disco y para contener conjuntos de datos muy extensos. Para acceder a una cinta, primero debe colocarse en un lector de cinta, ya sea que lo haga una persona o un robot (el manejo automatizado de las cintas es común en las instalaciones con bases de datos enormes). Después la cinta tal vez tenga que embobinarse hacia delante para llegar al bloque solicitado. En general, este proceso podría tardar varios minutos. La gran ventaja de la cinta es que es en extremo económica por bit y removible, lo cual es importante para las cintas de respaldo que se deben almacenar fuera del sitio de trabajo para que puedan sobrevivir a los incendios, inundaciones, terremotos y otros desastres.

La jerarquía de memoria que hemos descrito es la común, pero algunas instalaciones no tienen todas las capas o tienen unas cuantas capas distintas (como el disco óptico). Aún así, a medida que se desciende por todos las capas en la jerarquía, el tiempo de acceso aleatorio se incrementa en forma dramática, la capacidad aumenta de igual forma y el costo por bit baja considerablemente. En consecuencia, es probable que las jerarquías de memoria se utilicen por varios años más.

1.3.5 Dispositivos de E/S

La CPU y la memoria no son los únicos recursos que el sistema operativo debe administrar. Los dispositivos de E/S también interactúan mucho con el sistema operativo. Como vimos en la figura 1-6, los dispositivos de E/S generalmente constan de dos partes: un dispositivo controlador y el dispositivo en sí. El dispositivo controlador es un chip o conjunto de chips que controla físicamente el dispositivo. Por ejemplo, acepta los comandos del sistema operativo para leer datos del dispositivo y los lleva a cabo.

En muchos casos, el control del dispositivo es muy complicado y detallado, por lo que el trabajo del chip o los chips del dispositivo controlador es presentar una interfaz más simple al sistema operativo (pero de todas formas sigue siendo muy complejo). Por ejemplo, un controlador de disco podría aceptar un comando para leer el sector 11,206 del disco 2; después, tiene que convertir este número de

sector lineal en un cilindro, sector y cabeza. Esta conversión se puede complicar por el hecho de que los cilindros exteriores tienen más sectores que los interiores y que algunos sectores defectuosos se han reasignado a otros. Posteriormente, el dispositivo controlador tiene que determinar en cuál cilindro se encuentra el brazo y darle una secuencia de pulsos para desplazarse hacia dentro o hacia fuera el número requerido de cilindros; tiene que esperar hasta que el sector apropiado haya girado bajo la cabeza, y después empieza a leer y almacenar los bits a medida que van saliendo de la unidad, eliminando el preámbulo y calculando la suma de verificación. Por último, tiene que ensamblar los bits entrantes en palabras y almacenarlos en la memoria. Para hacer todo este trabajo, a menudo los dispositivos controladores consisten en pequeñas computadoras incrustadas que se programan para realizar su trabajo.

La otra pieza es el dispositivo en sí. Los dispositivos tienen interfaces bastante simples, debido a que no pueden hacer mucho y también para estandarizarlas. Esto último es necesario de manera que cualquier dispositivo controlador de disco IDE pueda manejar cualquier disco IDE, por ejemplo. **IDE** (*Integrated Drive Electronics*) significa **Electrónica de unidades integradas** y es el tipo estándar de disco en muchas computadoras. Como la interfaz real del dispositivo está oculta detrás del dispositivo controlador, todo lo que el sistema operativo ve es la interfaz para el dispositivo controlador, que puede ser bastante distinta de la interfaz para el dispositivo.

Como cada tipo de dispositivo controlador es distinto, se requiere software diferente para controlar cada uno de ellos. El software que se comunica con un dispositivo controlador, que le proporciona comandos y acepta respuestas, se conoce como **driver** (controlador). Cada fabricante de dispositivos controladores tiene que suministrar un driver específico para cada sistema operativo en que pueda funcionar. Así, un escáner puede venir, por ejemplo, con drivers para Windows 2000, Windows XP, Vista y Linux.

Para utilizar el driver, se tiene que colocar en el sistema operativo de manera que pueda ejecutarse en modo kernel. En realidad, los drivers se pueden ejecutar fuera del kernel, pero sólo unos cuantos sistemas actuales admiten esta posibilidad debido a que se requiere la capacidad para permitir que un driver en espacio de usuario pueda acceder al dispositivo de una manera controlada, una característica que raras veces se admite. Hay tres formas en que el driver se pueda colocar en el kernel: la primera es volver a enlazar el kernel con el nuevo driver y después reiniciar el sistema (muchos sistemas UNIX antiguos trabajan de esta manera); la segunda es crear una entrada en un archivo del sistema operativo que le indique que necesita el driver y después reinicie el sistema, para que en el momento del arranque, el sistema operativo busque los drivers necesarios y los cargue (Windows funciona de esta manera); la tercera forma es que el sistema operativo acepte nuevos drivers mientras los ejecuta e instala al instante, sin necesidad de reiniciar. Esta última forma solía ser rara, pero ahora se está volviendo mucho más común. Los dispositivos conectables en caliente *(hotpluggable*), como los dispositivos USB e IEEE 1394 (que se describen a continuación) siempre necesitan drivers que se cargan en forma dinámica.

Todo dispositivo controlador tiene un número pequeño de registros que sirven para comunicarse con él. Por ejemplo, un dispositivo controlador de disco con las mínimas características podría tener registros para especificar la dirección de disco, dirección de memoria, número de sectores e instrucción (lectura o escritura). Para activar el dispositivo controlador, el driver recibe un comando del sistema operativo y después lo traduce en los valores apropiados para escribirlos en los registros del dispositivo. La colección de todos los registros del dispositivo forma el **espacio de puertos de E/S**, un tema al que regresaremos en el capítulo 5.

En ciertas computadoras, los registros de dispositivo tienen una correspondencia con el espacio de direcciones del sistema operativo (las direcciones que puede utilizar), de modo que se puedan leer y escribir en ellas como si fuera en palabras de memoria ordinarias. En dichas computadoras no se requieren instrucciones de E/S especiales y los programas de usuario pueden aislarse del hardware al no colocar estas direcciones de memoria dentro de su alcance (por ejemplo, mediante el uso de registros base y límite). En otras computadoras, los registros de dispositivo se colocan en un espacio de puertos de E/S especial, donde cada registro tiene una dirección de puerto. En estas máquinas hay instrucciones `IN` y `OUT` especiales disponibles en modo kernel que permiten a los drivers leer y escribir en los registros. El primer esquema elimina la necesidad de instrucciones de E/S especiales, pero utiliza parte del espacio de direcciones. El segundo esquema no utiliza espacio de direcciones, pero requiere instrucciones especiales. Ambos sistemas se utilizan ampliamente.

Las operaciones de entrada y salida se pueden realizar de tres maneras distintas. En el método más simple, un programa de usuario emite una llamada al sistema, que el kernel posteriormente traduce en una llamada al procedimiento para el driver apropiado. Después el driver inicia la E/S y permanece en un ciclo estrecho, sondeando en forma continua al dispositivo para ver si ha terminado (por lo general hay un bit que indica si el dispositivo sigue ocupado). Una vez terminada la E/S, el driver coloca los datos (si los hay) en donde se necesitan y regresa. Después el sistema operativo devuelve el control al llamador. A este método se le conoce como **espera ocupada** y tiene la desventaja de que mantiene ocupada la CPU sondeando al dispositivo hasta que éste termina.

El segundo método consiste en que el driver inicie el dispositivo y le pida generar una interrupción cuando termine. En este punto el driver regresa. Luego, el sistema operativo bloquea el programa llamador si es necesario y busca otro trabajo por hacer. Cuando el dispositivo controlador detecta el final de la transferencia, genera una **interrupción** para indicar que la operación se ha completado.

Las interrupciones son muy importantes en los sistemas operativos, por lo cual vamos a examinar la idea con más detalle. En la figura 1-11(a) podemos ver un proceso de tres pasos para la E/S. En el paso 1, el driver indica al dispositivo controlador de disco lo que debe hacer, al escribir datos en sus registros de dispositivo. Después el dispositivo controlador inicia el dispositivo; cuando ha terminado de leer o escribir el número de bytes que debe transferir, alerta al chip controlador de interrupciones mediante el uso de ciertas líneas de bus en el paso 2. Si el controlador de interrupciones está preparado para aceptar la interrupción (lo cual podría no ser cierto si está ocupado con una de mayor prioridad), utiliza un pin en el chip de CPU para informarlo, en el paso 3. En el paso 4, el controlador de interrupciones coloca el número del dispositivo en el bus, para que la CPU pueda leerlo y sepa cuál dispositivo acaba de terminar (puede haber muchos dispositivos funcionando al mismo tiempo).

Una vez que la CPU ha decidido tomar la interrupción, el contador de programa y el PSW son típicamente agregados (*pushed*) en la pila actual y la CPU cambia al modo kernel. El número de dispositivo se puede utilizar como un índice en parte de la memoria para encontrar la dirección del manejador (*handler*) de interrupciones para este dispositivo. Esta parte de la memoria se conoce como **vector de interrupción**. Una vez que el manejador de interrupciones (parte del driver para el dispositivo que está realizando la interrupción) ha iniciado, quita el contador de programa y el PSW de la pila y los guarda, para después consultar al dispositivo y conocer su estado. Cuando el manejador de interrupciones termina, regresa al programa de usuario que se estaba ejecutando previamente a la primera instrucción que no se había ejecutado todavía. Estos pasos se muestran en la figura 1-11(b).

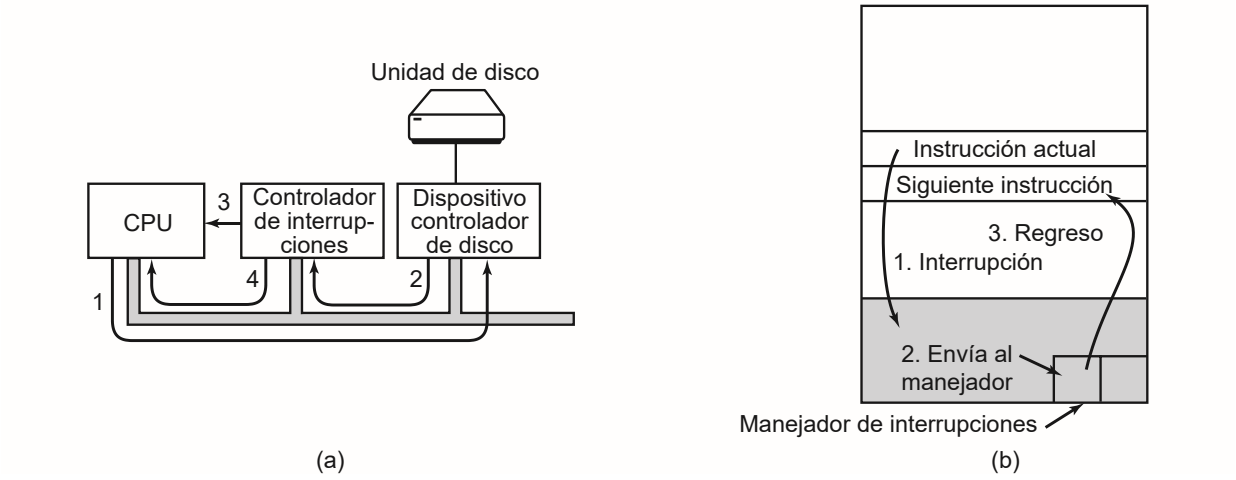

Figura 1-11. (a) Los pasos para iniciar un dispositivo de E/S y obtener una interrupción. (b) El procesamiento de interrupciones involucra tomar la interrupción, ejecutar el manejador de interrupciones y regresar al programa de usuario.

El tercer método para realizar operaciones de E/S hace uso de un chip especial llamado **DMA** (*Direct Memory Access;* Acceso directo a memoria) que puede controlar el flujo de bits entre la memoria y un dispositivo controlador sin la intervención constante de la CPU. La CPU configura el chip DMA, le indica cuántos bytes debe transferir, las direcciones de dispositivo y de memoria involucradas, la instrucción y deja que haga su trabajo. Cuando el chip DMA termina genera una interrupción, la cual se maneja de la manera antes descrita. En el capítulo 5 discutiremos con más detalle sobre el hardware de DMA y de E/S, en general.

A menudo, las interrupciones pueden ocurrir en momentos muy inconvenientes, por ejemplo mientras otro manejador de interrupciones se está ejecutando. Por esta razón, la CPU tiene una forma para deshabilitar las interrupciones y rehabilitarlas después. Mientras las interrupciones están deshabilitadas, cualquier dispositivo que termine continúa utilizando sus señales de interrupción, pero la CPU no se interrumpe sino hasta que se vuelven a habilitar las interrupciones. Si varios dispositivos terminan mientras las interrupciones están habilitadas, el controlador de interrupciones decide cuál debe dejar pasar primero, lo cual se basa generalmente en prioridades estáticas asignadas a cada dispositivo. El dispositivo de mayor prioridad gana.

1.3.6 Buses

La organización de la figura 1-6 se utilizó en las minicomputadoras durante años y también en la IBM PC original. Sin embargo, a medida que los procesadores y las memorias se hicieron más veloces, la habilidad de un solo bus (y sin duda, del bus de la IBM PC) de manejar todo el tráfico se forzaba hasta el punto de quiebre. Algo tenía que ceder. Como resultado se agregaron más buses, tanto para dispositivos de E/S más rápidos como para el tráfico entre la CPU y la memoria. Como consecuencia de esta evolución, un sistema Pentium extenso tiene actualmente una apariencia similar a la figura 1-12.

El sistema tiene ocho buses (caché, local, memoria, PCI, SCSI, USB, IDE e ISA), cada uno con una velocidad de transferencia y función distintas. El sistema operativo debe estar al tanto de todos

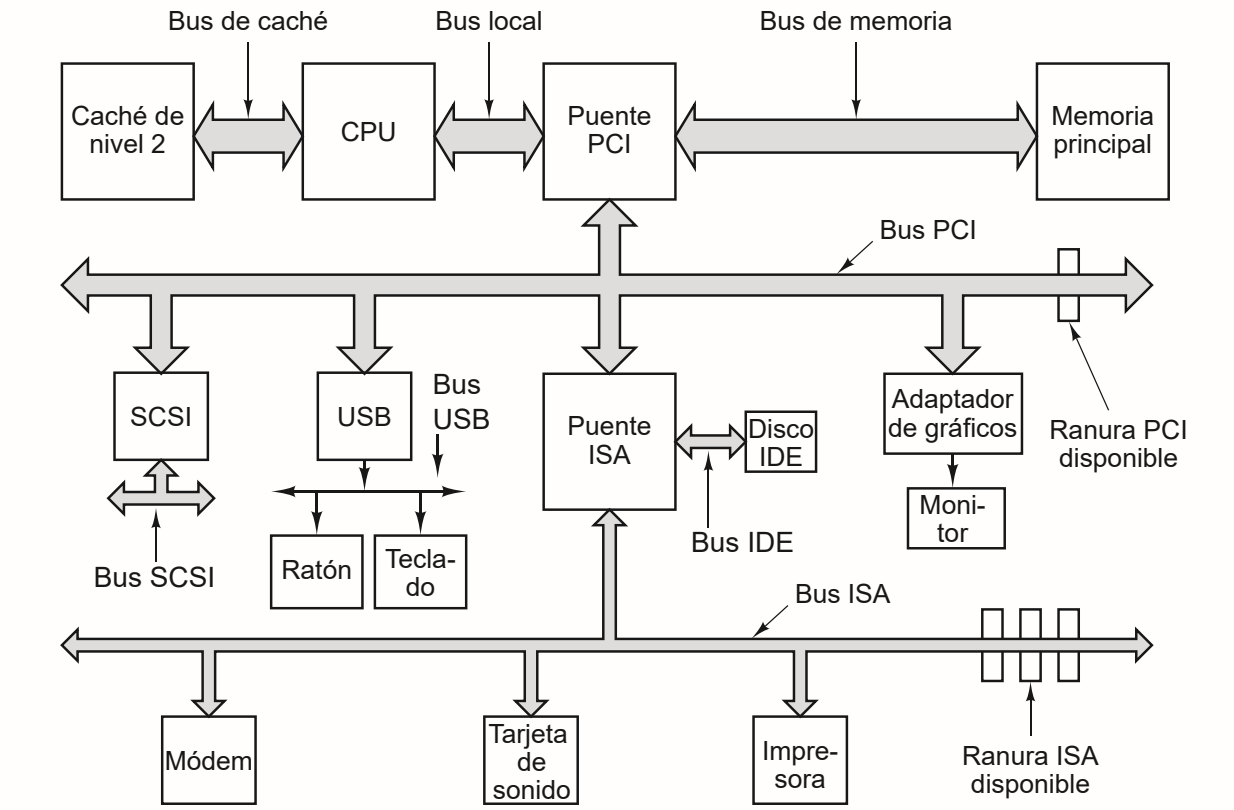

Figura 1-12. La estructura de un sistema Pentium extenso.

estos buses para su configuración y administración. Los dos buses principales son el bus **ISA** (*Industry Standard Architecture*, Arquitectura estándar de la industria) de la IBM PC original y su sucesor, el bus **PCI** (*Peripheral Component Interconnect*, Interconexión de componentes periféricos). El bus ISA (el bus original de la IBM PC/AT) opera a 8.33 MHz y puede transferir 2 bytes a la vez, para una velocidad máxima de 16.67 MB/seg. Se incluye para mantener compatibilidad hacia atrás con las tarjetas de E/S antiguas y lentas. Los sistemas modernos lo omiten con frecuencia, pues ya es obsoleto. El bus PCI fue inventado por Intel como sucesor para el bus ISA. Puede operar a 66 MHz y transferir 8 bytes a la vez, para lograr una velocidad de transferencia de datos de 528 MB/seg. La mayoría de los dispositivos de E/S de alta velocidad utilizan el bus PCI en la actualidad. Incluso algunas computadoras que no emplean procesadores Intel usan el bus PCI, debido al extenso número de tarjetas de E/S disponibles para este bus. Las nuevas computadoras están saliendo al mercado con una versión actualizad del bus PCI, conocida como **PCI Express**.

En esta configuración, la CPU se comunica con el chip puente PCI a través del bus local y el chip puente PCI se comunica con la memoria a través de un bus de memoria dedicado, que normalmente opera a 100 MHz. Los sistemas Pentium tienen una caché de nivel 1 en el chip y una caché de nivel 2 mucho mayor fuera del chip, conectada a la CPU mediante el bus de caché.

Además, este sistema contiene tres buses especializados: IDE, USB y SCSI. El bus IDE sirve para conectar dispositivos periféricos tales como discos y CD-ROM al sistema. El bus IDE es

fruto de la interfaz controladora de disco en la PC/AT y ahora es estándar en casi todos los sistemas basados en Pentium para el disco duro y a menudo para el CD-ROM.

El **USB** (*Universal Serial Bus*; Bus serial universal) se inventó para conectar a la computadora todos los dispositivos de E/S lentos, como el teclado y el ratón. Utiliza un pequeño conector con cuatro cables, dos de los cuales suministran energía eléctrica a los dispositivos USB. El USB es un bus centralizado en el que un dispositivo raíz sondea los dispositivos de E/S cada 1 milisegundo para ver si tienen tráfico. USB 1.0 podía manejar una carga agregada de 1.5 MB/seg, pero el más reciente USB 2.0 puede manejar 60 MB/seg. Todos los dispositivos USB comparten un solo dispositivo controlador USB, lo que hace innecesario instalar un nuevo controlador para cada nuevo dispositivo USB. En consecuencia, pueden agregarse dispositivos USB a la computadora sin necesidad de reiniciar.

El bus **SCSI** (*Small Computer System Interface*, Interfaz para sistemas de cómputo pequeños) es un bus de alto rendimiento, diseñado para discos, escáneres y otros dispositivos veloces que necesitan de un ancho de banda considerable. Puede operar a una velocidad de transferencia de hasta 160 MB/seg. Ha estado presente en los sistemas Macintosh desde que se inventaron y también es popular en UNIX y en ciertos sistemas basados en Intel.

Hay otro bus (que no se muestra en la figura 1-12) conocido como **IEEE 1394**. Algunas veces se le conoce como FireWire, aunque hablando en sentido estricto, FireWire es el nombre que utiliza Apple para su implementación del 1394. Al igual que el USB, el IEEE 1394 es un bus de bits en serie, pero está diseñado para transferencias empaquetadas de hasta 100 MB/seg., lo que lo hace conveniente para conectar a una computadora cámaras de video digitales y dispositivos multimedia similares. A diferencia del USB, el IEE 1394 no tiene un dispositivo controlador central.

Para trabajar en un entorno tal como el de la figura 1-12, el sistema operativo tiene que saber qué dispositivos periféricos están conectados a la computadora y cómo configurarlos. Este requerimiento condujo a Intel y Microsoft a diseñar un sistema de PC conocido como **plug and play** basado en un concepto similar que se implementó por primera vez en la Apple Macintosh. Antes de plug and play, cada tarjeta de E/S tenía un nivel de petición de interrupción fijo y direcciones fijas para sus registros de E/S. Por ejemplo, el teclado tenía la interrupción 1 y utilizaba las direcciones de E/S 0x60 a 0x64, el dispositivo controlador de disco flexible tenía la interrupción 6 y utilizaba las direcciones de E/S 0x3F0 a 0x3F7, la impresora tenía la interrupción 7 y utilizaba las direcciones de E/S 0x378 a 0x37A, y así sucesivamente.

Hasta aquí todo está bien. El problema llegó cuando el usuario compraba una tarjeta de sonido y una tarjeta de módem que utilizaban la misma interrupción, por ejemplo, la 4. Instaladas juntas serían incapaces de funcionar. La solución fue incluir interruptores DIP o puentes (*jumpers*) en cada tarjeta de E/S e indicar al usuario que por favor los configurara para seleccionar un nivel de interrupción y direcciones de dispositivos de E/S que no estuvieran en conflicto con las demás tarjetas en el sistema del usuario. Los adolescentes que dedicaron sus vidas a las complejidades del hardware de la PC podían algunas veces hacer esto sin cometer errores. Por desgracia nadie más podía hacerlo, lo cual provocó un caos.

La función de plug and play es permitir que el sistema recolecte automáticamente la información acerca de los dispositivos de E/S, asigne los niveles de interrupción y las direcciones de E/S de manera central, para que después indique a cada tarjeta cuáles son sus números. Este trabajo está íntimamente relacionado con el proceso de arranque de la computadora, por lo que a continuación analizaremos este proceso nada trivial.

1.3.7 Arranque de la computadora

En forma muy breve, el proceso de arranque del Pentium es el siguiente. Cada Pentium contiene una tarjeta madre (*motherboard*). En la tarjeta madre o padre hay un programa conocido como **BIOS** (*Basic Input Output System*, Sistema básico de entrada y salida) del sistema. El BIOS contiene software de E/S de bajo nivel, incluyendo procedimientos para leer el teclado, escribir en la pantalla y realizar operaciones de E/S de disco, entre otras cosas. Hoy en día está contenido en una RAM tipo flash que es no volátil pero el sistema operativo puede actualizarla cuando se encuentran errores en el BIOS.

Cuando se arranca la computadora, el BIOS inicia su ejecución. Primero hace pruebas para ver cuánta RAM hay instalada y si el teclado junto con otros dispositivos básicos están instalados y responden en forma correcta. Empieza explorando los buses ISA y PCI para detectar todos los dispositivos conectados a ellos. Comúnmente, algunos de estos dispositivos son **heredados** (es decir, se diseñaron antes de inventar la tecnología plug and play), además de tener valores fijos para los niveles de interrupciones y las direcciones de E/S (que posiblemente se establecen mediante interruptores o puentes en la tarjeta de E/S, pero que el sistema operativo no puede modificar). Estos dispositivos se registran; y los dispositivos plug and play también. Si los dispositivos presentes son distintos de los que había cuando el sistema se inició por última vez, se configuran los nuevos dispositivos.

Después, el BIOS determina el dispositivo de arranque, para lo cual prueba una lista de dispositivos almacenada en la memoria CMOS. El usuario puede cambiar esta lista si entra a un programa de configuración del BIOS, justo después de iniciar el sistema. Por lo general, se hace un intento por arrancar del disco flexible, si hay uno presente. Si eso falla, se hace una consulta a la unidad de CD-ROM para ver si contiene un CD-ROM que se pueda arrancar. Si no hay disco flexible ni CD-ROM que puedan iniciarse, el sistema se arranca desde el disco duro. El primer sector del dispositivo de arranque se lee y se coloca en la memoria, para luego ejecutarse. Este sector contiene un programa que por lo general examina la tabla de particiones al final del sector de arranque, para determinar qué partición está activa. Después se lee un cargador de arranque secundario de esa partición. Este cargador lee el sistema operativo de la partición activa y lo inicia.

Luego, el sistema operativo consulta al BIOS para obtener la información de configuración. Para cada dispositivo, comprueba si tiene el driver correspondiente. De no ser así, pide al usuario que inserte un CD-ROM que contenga el driver (suministrado por el fabricante del dispositivo). Una vez que tiene los drivers de todos los dispositivos, el sistema operativo los carga en el kernel. Después inicializa sus tablas, crea los procesos de segundo plano que se requieran, y arranca un programa de inicio de sesión o GUI.

1.4 LOS TIPOS DE SISTEMAS OPERATIVOS

Los sistemas operativos han estado en funcionamiento durante más de medio siglo. Durante este tiempo se ha desarrollado una variedad bastante extensa de ellos, no todos se conocen ampliamente. En esta sección describiremos de manera breve nueve. Más adelante en el libro regresaremos a ver algunos de estos distintos tipos de sistemas.

1.4.1 Sistemas operativos de mainframe

En el extremo superior están los sistemas operativos para las mainframes, las computadoras del tamaño de un cuarto completo que aún se encuentran en los principales centros de datos corporativos. La diferencia entre estas computadoras y las personales está en su capacidad de E/S. Una mainframe con 1000 discos y millones de gigabytes de datos no es poco común; una computadora personal con estas especificaciones sería la envidia de los amigos del propietario. Las mainframes también están volviendo a figurar en el ámbito computacional como servidores Web de alto rendimiento, servidores para sitios de comercio electrónico a gran escala y servidores para transacciones de negocio a negocio.

Los sistemas operativos para las mainframes están profundamente orientados hacia el procesamiento de muchos trabajos a la vez, de los cuales la mayor parte requiere muchas operaciones de E/S. Por lo general ofrecen tres tipos de servicios: procesamiento por lotes, procesamiento de transacciones y tiempo compartido. Un sistema de procesamiento por lotes procesa los trabajos de rutina sin que haya un usuario interactivo presente. El procesamiento de reclamaciones en una compañía de seguros o el reporte de ventas para una cadena de tiendas son actividades que se realizan comúnmente en modo de procesamiento por lotes. Los sistemas de procesamiento de transacciones manejan grandes cantidades de pequeñas peticiones, por ejemplo: el procesamiento de cheques en un banco o las reservaciones en una aerolínea. Cada unidad de trabajo es pequeña, pero el sistema debe manejar cientos o miles por segundo. Los sistemas de tiempo compartido permiten que varios usuarios remotos ejecuten trabajos en la computadora al mismo tiempo, como consultar una gran base de datos. Estas funciones están íntimamente relacionadas; a menudo los sistemas operativos de las mainframes las realizan todas. Un ejemplo de sistema operativo de mainframe es el OS/390, un descendiente del OS/360. Sin embargo, los sistemas operativos de mainframes están siendo reemplazados gradualmente por variantes de UNIX, como Linux.

1.4.2 Sistemas operativos de servidores

En el siguiente nivel hacia abajo se encuentran los sistemas operativos de servidores. Se ejecutan en servidores, que son computadoras personales muy grandes, estaciones de trabajo o incluso mainframes. Dan servicio a varios usuarios a la vez a través de una red y les permiten compartir los recursos de hardware y de software. Los servidores pueden proporcionar servicio de impresión, de archivos o Web. Los proveedores de Internet operan muchos equipos servidores para dar soporte a sus clientes y los sitios Web utilizan servidores para almacenar las páginas Web y hacerse cargo de las peticiones entrantes. Algunos sistemas operativos de servidores comunes son Solaris, FreeBSD, Linux y Windows Server 200x.

1.4.3 Sistemas operativos de multiprocesadores

Una manera cada vez más común de obtener poder de cómputo de las grandes ligas es conectar varias CPU en un solo sistema. Dependiendo de la exactitud con la que se conecten y de lo que se comparta, estos sistemas se conocen como computadoras en paralelo, multicomputadoras o multiprocesadores. Necesitan sistemas operativos especiales, pero a menudo son variaciones de los sistemas operativos de servidores con características especiales para la comunicación, conectividad y consistencia.

Con la reciente llegada de los chips multinúcleo para las computadoras personales, hasta los sistemas operativos de equipos de escritorio y portátiles convencionales están empezando a lidiar con multiprocesadores de al menos pequeña escala y es probable que el número de núcleos aumente con el tiempo. Por fortuna, se conoce mucho acerca de los sistemas operativos de multiprocesadores gracias a los años de investigación previa, por lo que el uso de este conocimiento en los sistemas multinúcleo no debe presentar dificultades. La parte difícil será hacer que las aplicaciones hagan uso de todo este poder de cómputo. Muchos sistemas operativos populares (incluyendo Windows y Linux) se ejecutan en multiprocesadores.

1.4.4 Sistemas operativos de computadoras personales

La siguiente categoría es el sistema operativo de computadora personal. Todos los sistemas operativos modernos soportan la multiprogramación, con frecuencia se inician docenas de programas al momento de arrancar el sistema. Su trabajo es proporcionar buen soporte para un solo usuario. Se utilizan ampliamente para el procesamiento de texto, las hojas de cálculo y el acceso a Internet. Algunos ejemplos comunes son Linux, FreeBSD, Windows Vista y el sistema operativo Macintosh. Los sistemas operativos de computadora personal son tan conocidos que tal vez no sea necesario presentarlos con mucho detalle. De hecho, muchas personas ni siquiera están conscientes de que existen otros tipos de sistemas operativos.

1.4.5 Sistemas operativos de computadoras de bolsillo

Continuando con los sistemas cada vez más pequeños, llegamos a las computadoras de bolsillo (*handheld*). Una computadora de bolsillo o **PDA** (*Personal Digital Assitant*, Asistente personal digital) es una computadora que cabe en los bolsillos y realiza una pequeña variedad de funciones, como libreta de direcciones electrónica y bloc de notas. Además, hay muchos teléfonos celulares muy similares a los PDAs, con la excepción de su teclado y pantalla. En efecto, los PDAs y los teléfonos celulares se han fusionado en esencia y sus principales diferencias se observan en el tamaño, el peso y la interfaz de usuario. Casi todos ellos se basan en CPUs de 32 bits con el modo protegido y ejecutan un sofisticado sistema operativo.

Los sistemas operativos que operan en estos dispositivos de bolsillo son cada vez más sofisticados, con la habilidad de proporcionar telefonía, fotografía digital y otras funciones. Muchos de ellos también ejecutan aplicaciones desarrolladas por terceros. De hecho, algunos están comenzando a asemejarse a los sistemas operativos de computadoras personales de hace una década. Una de las principales diferencias entre los dispositivos de bolsillo y las PCs es que los primeros no tienen discos duros de varios cientos de gigabytes, lo cual cambia rápidamente. Dos de los sistemas operativos más populares para los dispositivos de bolsillo son Symbian OS y Palm OS.

1.4.6 Sistemas operativos integrados

Los sistemas integrados (*embedded*), que también se conocen como incrustados o embebidos, operan en las computadoras que controlan dispositivos que no se consideran generalmente como computadoras, ya que no aceptan software instalado por el usuario. Algunos ejemplos comunes son los hornos

de microondas, las televisiones, los autos, los grabadores de DVDs, los teléfonos celulares y los reproductores de MP3. La propiedad principal que diferencia a los sistemas integrados de los dispositivos de bolsillo es la certeza de que nunca se podrá ejecutar software que no sea confiable. No se pueden descargar nuevas aplicaciones en el horno de microondas; todo el software se encuentra en ROM. Esto significa que no hay necesidad de protección en las aplicaciones, lo cual conlleva a cierta simplificación. Los sistemas como QNX y VxWorks son populares en este dominio.

1.4.7 Sistemas operativos de nodos sensores

Las redes de pequeños nodos sensores se están implementando para varios fines. Estos nodos son pequeñas computadoras que se comunican entre sí con una estación base, mediante el uso de comunicación inalámbrica. Estas redes de sensores se utilizan para proteger los perímetros de los edificios, resguardar las fronteras nacionales, detectar incendios en bosques, medir la temperatura y la precipitación para el pronóstico del tiempo, deducir información acerca del movimiento de los enemigos en los campos de batalla y mucho más.

Los sensores son pequeñas computadoras con radios integrados y alimentadas con baterías. Tienen energía limitada y deben trabajar durante largos periodos al exterior y desatendidas, con frecuencia en condiciones ambientales rudas. La red debe ser lo bastante robusta como para tolerar fallas en los nodos individuales, que ocurren con mayor frecuencia a medida que las baterías empiezan a agotarse.

Cada nodo sensor es una verdadera computadora, con una CPU, RAM, ROM y uno o más sensores ambientales. Ejecuta un sistema operativo pequeño pero real, por lo general manejador de eventos, que responde a los eventos externos o realiza mediciones en forma periódica con base en un reloj interno. El sistema operativo tiene que ser pequeño y simple debido a que los nodos tienen poca RAM y el tiempo de vida de las baterías es una cuestión importante. Además, al igual que con los sistemas integrados, todos los programas se cargan por adelantado; los usuarios no inician repentinamente programas que descargaron de Internet, lo cual simplifica el diseño en forma considerable. TinyOS es un sistema operativo bien conocido para un nodo sensor.

1.4.8 Sistemas operativos en tiempo real

Otro tipo de sistema operativo es el sistema en tiempo real. Estos sistemas se caracterizan por tener el tiempo como un parámetro clave. Por ejemplo, en los sistemas de control de procesos industriales, las computadoras en tiempo real tienen que recolectar datos acerca del proceso de producción y utilizarlos para controlar las máquinas en la fábrica. A menudo hay tiempos de entrega estrictos que se deben cumplir. Por ejemplo, si un auto se desplaza sobre una línea de ensamblaje, deben llevarse a cabo ciertas acciones en determinados instantes. Si un robot soldador realiza su trabajo de soldadura antes o después de tiempo, el auto se arruinará. Si la acción *debe* ocurrir sin excepción en cierto momento (o dentro de cierto rango), tenemos un **sistema en tiempo real duro**. Muchos de estos sistemas se encuentran en el control de procesos industriales, en aeronáutica, en la milicia y en áreas de aplicación similares. Estos sistemas deben proveer garantías absolutas de que cierta acción ocurrirá en un instante determinado.

Otro tipo de sistema en tiempo real es el **sistema en tiempo real suave**, en el cual es aceptable que muy ocasionalmente se pueda fallar a un tiempo predeterminado. Los sistemas de audio digital o de multimedia están en esta categoría. Los teléfonos digitales también son ejemplos de sistema en tiempo real suave.

Como en los sistemas en tiempo real es crucial cumplir con tiempos predeterminados para realizar una acción, algunas veces el sistema operativo es simplemente una biblioteca enlazada con los programas de aplicación, en donde todo está acoplado en forma estrecha y no hay protección entre cada una de las partes del sistema. Un ejemplo de este tipo de sistema en tiempo real es e-Cos.

Las categorías de sistemas para computadoras de bolsillo, sistemas integrados y sistemas en tiempo real se traslapan en forma considerable. Casi todos ellos tienen por lo menos ciertos aspectos de tiempo real suave. Los sistemas integrados y de tiempo real sólo ejecutan software que colocan los diseñadores del sistema; los usuarios no pueden agregar su propio software, lo cual facilita la protección. Los sistemas de computadoras de bolsillo y los sistemas integrados están diseñados para los consumidores, mientras que los sistemas en tiempo real son más adecuados para el uso industrial. Sin embargo, tienen ciertas características en común.

1.4.9 Sistemas operativos de tarjetas inteligentes

Los sistemas operativos más pequeños operan en las tarjetas inteligentes, que son dispositivos del tamaño de una tarjeta de crédito que contienen un chip de CPU. Tienen varias severas restricciones de poder de procesamiento y memoria. Algunas se energizan mediante contactos en el lector en el que se insertan, pero las tarjetas inteligentes sin contactos se energizan mediante inducción, lo cual limita en forma considerable las cosas que pueden hacer. Algunos sistemas de este tipo pueden realizar una sola función, como pagos electrónicos; otros pueden llevar a cabo varias funciones en la misma tarjeta inteligente. A menudo éstos son sistemas propietarios.

Algunas tarjetas inteligentes funcionan con Java. Lo que esto significa es que la ROM en la tarjeta inteligente contiene un intérprete para la Máquina virtual de Java (JVM). Los applets de Java (pequeños programas) se descargan en la tarjeta y son interpretados por el intérprete de la JVM. Algunas de estas tarjetas pueden manejar varias applets de Java al mismo tiempo, lo cual conlleva a la multiprogramación y a la necesidad de planificarlos. La administración de los recursos y su protección también se convierten en un problema cuando hay dos o más applets presentes al mismo tiempo. El sistema operativo (que por lo general es en extremo primitivo) presente en la tarjeta es el encargado de manejar estas cuestiones.

1.5 CONCEPTOS DE LOS SISTEMAS OPERATIVOS

La mayoría de los sistemas operativos proporcionan ciertos conceptos básicos y abstracciones tales como procesos, espacios de direcciones y archivos, que son la base para comprender su funcionamiento. En las siguientes secciones analizaremos algunos de estos conceptos básicos en forma breve, como una introducción. Más adelante en el libro volveremos a analizar cada uno de ellos con mayor detalle. Para ilustrar estos conceptos, de vez en cuando utilizaremos ejemplos que por lo general se basan en UNIX. No obstante, por lo general existen también ejemplos similares en otros sistemas, además de que en el capítulo 11 estudiaremos Windows Vista con detalle.

1.5.1 Procesos

Un concepto clave en todos los sistemas operativos es el **proceso**. Un proceso es en esencia un programa en ejecución. Cada proceso tiene asociado un **espacio de direcciones**, una lista de ubicaciones de memoria que va desde algún mínimo (generalmente 0) hasta cierto valor máximo, donde el proceso puede leer y escribir información. El espacio de direcciones contiene el programa ejecutable, los datos del programa y su pila. También hay asociado a cada proceso un conjunto de recursos, que comúnmente incluye registros (el contador de programa y el apuntador de pila, entre ellos), una lista de archivos abiertos, alarmas pendientes, listas de procesos relacionados y toda la demás información necesaria para ejecutar el programa. En esencia, un proceso es un recipiente que guarda toda la información necesaria para ejecutar un programa.

En el capítulo 2 volveremos a analizar el concepto de proceso con más detalle, pero por ahora la manera más fácil de que el lector se dé una buena idea de lo que es un proceso es pensar en un sistema de multiprogramación. El usuario puede haber iniciado un programa de edición de video para convertir un video de una hora a un formato específico (algo que puede tardar horas) y después irse a navegar en la Web. Mientras tanto, un proceso en segundo plano que despierta en forma periódica para comprobar los mensajes entrantes puede haber empezado a ejecutarse. Así tenemos (cuando menos) tres procesos activos: el editor de video, el navegador Web y el lector de correo electrónico. Cada cierto tiempo, el sistema operativo decide detener la ejecución de un proceso y empezar a ejecutar otro; por ejemplo, debido a que el primero ha utilizado más tiempo del que le correspondía de la CPU en el último segundo.

Cuando un proceso se suspende en forma temporal como en el ejemplo anterior, debe reiniciarse después exactamente en el mismo estado que tenía cuando se detuvo. Esto significa que toda la información acerca del proceso debe guardarse en forma explícita en alguna parte durante la suspensión. Por ejemplo, el proceso puede tener varios archivos abiertos para leerlos al mismo tiempo. Con cada uno de estos archivos hay un apuntador asociado que proporciona la posición actual (es decir, el número del byte o registro que se va a leer a continuación). Cuando un proceso se suspende en forma temporal, todos estos apuntadores deben guardarse de manera que una llamada a `read` que se ejecute después de reiniciar el proceso lea los datos apropiados. En muchos sistemas operativos, toda la información acerca de cada proceso (además del contenido de su propio espacio de direcciones) se almacena en una tabla del sistema operativo, conocida como la **tabla de procesos**, la cual es un arreglo (o lista enlazada) de estructuras, una para cada proceso que se encuentre actualmente en existencia.

Así, un proceso (suspendido) consiste en su espacio de direcciones, que se conoce comúnmente como **imagen de núcleo** (en honor de las memorias de núcleo magnético utilizadas antaño) y su entrada en la tabla de procesos, que guarda el contenido de sus registros y muchos otros elementos necesarios para reiniciar el proceso más adelante.

Las llamadas al sistema de administración de procesos clave son las que se encargan de la creación y la terminación de los procesos. Considere un ejemplo común. Un proceso llamado **intérprete de comandos** o **shell** lee comandos de una terminal. El usuario acaba de escribir un comando, solicitando la compilación de un programa. El shell debe entonces crear un proceso para ejecutar el compilador. Cuando ese proceso ha terminado la compilación, ejecuta una llamada al sistema para terminarse a sí mismo.

Si un proceso puede crear uno o más procesos aparte (conocidos como **procesos hijos**) y estos procesos a su vez pueden crear procesos hijos, llegamos rápidamente la estructura de árbol de procesos de la figura 1-13. Los procesos relacionados que cooperan para realizar un cierto trabajo a menudo necesitan comunicarse entre sí y sincronizar sus actividades. A esta comunicación se le conoce como **comunicación entre procesos**, que veremos con detalle en el capítulo 2.

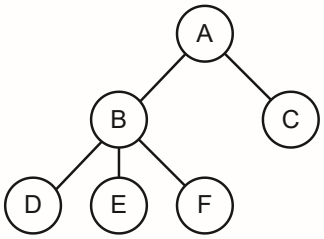

Figura 1-13. Un árbol de proceso. El proceso *A* creó dos proceso hijos, *B* y *C*. El proceso *B* creó tres procesos hijos, *D*, *E* y *F*.

Hay otras llamadas al sistema de procesos disponibles para solicitar más memoria (o liberar la memoria sin utilizar), esperar a que termine un proceso hijo y superponer su programa con uno distinto.

En algunas ocasiones se tiene la necesidad de transmitir información a un proceso en ejecución que no está esperando esta información. Por ejemplo, un proceso que se comunica con otro, en una computadora distinta, envía los mensajes al proceso remoto a través de una red de computadoras. Para protegerse contra la posibilidad de que se pierda un mensaje o su contestación, el emisor puede solicitar que su propio sistema operativo le notifique después de cierto número de segundos para que pueda retransmitir el mensaje, si no se ha recibido aún la señal de aceptación. Después de asignar este temporizador, el programa puede continuar realizando otro trabajo.

Cuando ha transcurrido el número especificado de segundos, el sistema operativo envía una **señal de alarma** al proceso. La señal provoca que el proceso suspenda en forma temporal lo que esté haciendo, almacene sus registros en la pila y empiece a ejecutar un procedimiento manejador de señales especial, por ejemplo, para retransmitir un mensaje que se considera perdido. Cuando termina el manejador de señales, el proceso en ejecución se reinicia en el estado en el que se encontraba justo antes de la señal. Las señales son la analogía en software de las interrupciones de hardware y se pueden generar mediante una variedad de causas además de la expiración de los temporizadores. Muchas traps detectadas por el hardware, como la ejecución de una instrucción ilegal o el uso de una dirección inválida, también se convierten en señales que se envían al proceso culpable.

Cada persona autorizada para utilizar un sistema recibe una **UID** (*User Identification*, Identificación de usuario) que el administrador del sistema le asigna. Cada proceso iniciado tiene el UID de la persona que lo inició. Un proceso hijo tiene el mismo UID que su padre. Los usuarios pueden ser miembros de grupos, cada uno de los cuales tiene una **GID** (*Group Identification*, Identificación de grupo).

Una UID conocida como **superusuario** (*superuser* en UNIX) tiene poder especial y puede violar muchas de las reglas de protección. En instalaciones extensas, sólo el administrador del sistema conoce la contraseña requerida para convertirse en superusuario, pero muchos de los usuarios ordi-

narios (en especial los estudiantes) dedican un esfuerzo considerable para tratar de encontrar fallas en el sistema que les permitan convertirse en superusuario sin la contraseña.

En el capítulo 2 estudiaremos los procesos, la comunicación entre procesos y las cuestiones relacionadas.

1.5.2 Espacios de direcciones

Cada computadora tiene cierta memoria principal que utiliza para mantener los programas en ejecución. En un sistema operativo muy simple sólo hay un programa a la vez en la memoria. Para ejecutar un segundo programa se tiene que quitar el primero y colocar el segundo en la memoria.

Los sistemas operativos más sofisticados permiten colocar varios programas en memoria al mismo tiempo. Para evitar que interfieran unos con otros (y con el sistema operativo), se necesita cierto mecanismo de protección. Aunque este mecanismo tiene que estar en el hardware, es controlado por el sistema operativo.

El anterior punto de vista se relaciona con la administración y protección de la memoria principal de la computadora. Aunque diferente, dado que la administración del espacio de direcciones de los procesos está relacionada con la memoria, es una actividad de igual importancia. Por lo general, cada proceso tiene cierto conjunto de direcciones que puede utilizar, que generalmente van desde 0 hasta cierto valor máximo. En el caso más simple, la máxima cantidad de espacio de direcciones que tiene un proceso es menor que la memoria principal. De esta forma, un proceso puede llenar su espacio de direcciones y aún así habrá suficiente espacio en la memoria principal para contener todo lo necesario.

Sin embargo, en muchas computadoras las direcciones son de 32 o 64 bits, con lo cual se obtiene un espacio de direcciones de 2^{32} o 2^{64} bytes, respectivamente. ¿Qué ocurre si un proceso tiene más espacio de direcciones que la memoria principal de la computadora, y desea usarlo todo? En las primeras computadoras, dicho proceso simplemente no podía hacer esto. Hoy en día existe una técnica llamada memoria virtual, como se mencionó antes, en la cual el sistema operativo mantiene una parte del espacio de direcciones en memoria principal y otra parte en el disco, moviendo pedazos de un lugar a otro según sea necesario. En esencia, el sistema operativo crea la abstracción de un espacio de direcciones como el conjunto de direcciones al que puede hacer referencia un proceso. El espacio de direcciones se desacopla de la memoria física de la máquina, pudiendo ser mayor o menor que la memoria física. La administración de los espacios de direcciones y la memoria física forman una parte importante de lo que hace un sistema operativo, por lo cual el capítulo 3 se dedica a este tema.

1.5.3 Archivos

Otro concepto clave de casi todos los sistemas operativos es el sistema de archivos. Como se dijo antes, una de las funciones principales del sistema operativo es ocultar las peculiaridades de los discos y demás dispositivos de E/S, presentando al programador un modelo abstracto limpio y agradable de archivos independientes del dispositivo. Sin duda se requieren las llamadas al sistema para crear los archivos, eliminarlos, leer y escribir en ellos. Antes de poder leer un archivo, debe locali-

zarse en el disco para abrirse y una vez que se ha leído información del archivo debe cerrarse, por lo que se proporcionan llamadas para hacer estas cosas.

Para proveer un lugar en donde se puedan mantener los archivos, la mayoría de los sistemas operativos tienen el concepto de un **directorio** como una manera de agrupar archivos. Por ejemplo, un estudiante podría tener un directorio para cada curso que esté tomando (para los programas necesarios para ese curso), otro directorio para su correo electrónico y otro más para su página de inicio en World Wide Web. Así, se necesitan llamadas al sistema para crear y eliminar directorios. También se proporcionan llamadas para poner un archivo existente en un directorio y para eliminar un archivo de un directorio. Las entradas de directorio pueden ser archivos u otros directorios. Este modelo también da surgimiento a una jerarquía (el sistema de archivos) como se muestra en la figura 1-14.

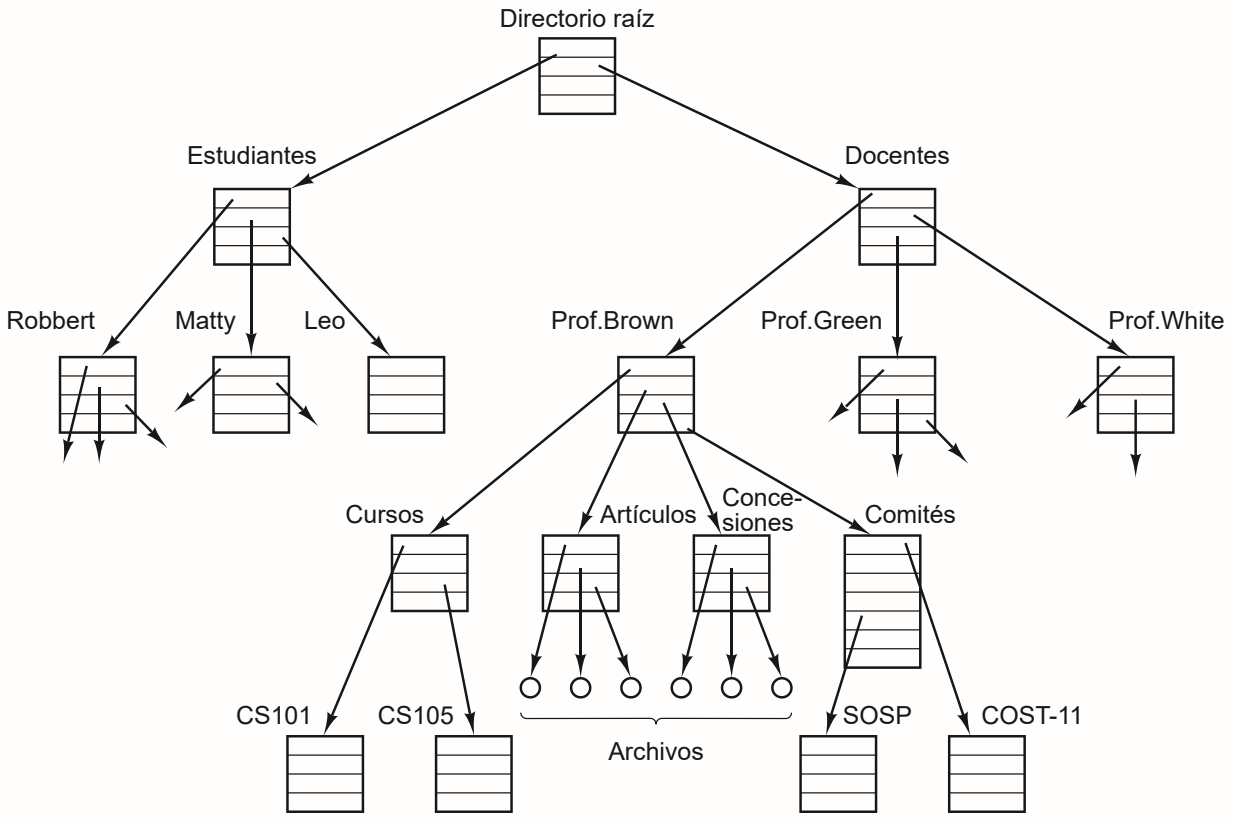

Figura 1-14. Un sistema de archivos para un departamento universitario.

Las jerarquías de procesos y de archivos están organizadas en forma de árboles, pero la similitud se detiene ahí. Por lo general, las jerarquías de procesos no son muy profundas (más de tres niveles es algo inusual), mientras que las jerarquías de archivos son comúnmente de cuatro, cinco o incluso más niveles de profundidad. Es común que las jerarquías de procesos tengan un tiempo de vida corto, por lo general de minutos a lo más, mientras que la jerarquía de directorios puede existir por años. La propiedad y la protección también difieren para los procesos y los archivos. Por lo común, sólo un proceso padre puede controlar o incluso acceder a un proceso hijo, pero casi siem-

pre existen mecanismos para permitir que los archivos y directorios sean leídos por un grupo aparte del propietario.

Para especificar cada archivo dentro de la jerarquía de directorio, se proporciona su **nombre de ruta** de la parte superior de la jerarquía de directorios, el **directorio raíz**. Dichos nombres de ruta absolutos consisten de la lista de directorios que deben recorrerse desde el directorio raíz para llegar al archivo, y se utilizan barras diagonales para separar los componentes. En la figura 1-14, la ruta para el archivo *CS101* es */Docentes/Prof.Brown/Cursos/CS101*. La primera barra diagonal indica que la ruta es absoluta, es decir, que empieza en el directorio raíz. Como una observación adicional, en MS-DOS y Windows se utiliza el carácter de barra diagonal inversa (\) como separador en vez del carácter de barra diagonal (/), por lo que la ruta del archivo antes mostrado podría escribirse como *\Docentes\Prof.Brown\Cursos\CS101*. A lo largo de este libro utilizaremos generalmente la convención de UNIX para las rutas.

En cada instante, cada proceso tiene un **directorio de trabajo** actual, en el que se buscan los nombres de ruta que no empiecen con una barra diagonal. Como ejemplo, en la figura 1-14 si */Docentes/Prof.Brown* fuera el directorio de trabajo, entonces el uso del nombre de ruta *Cursos/CS101* produciría el mismo archivo que el nombre de ruta absoluto antes proporcionado. Los procesos pueden modificar su directorio de trabajo mediante una llamada al sistema que especifique el nuevo directorio de trabajo.

Antes de poder leer o escribir en un archivo se debe abrir y en ese momento se comprueban los permisos. Si está permitido el acceso, el sistema devuelve un pequeño entero conocido como **descriptor de archivo** para usarlo en las siguientes operaciones. Si el acceso está prohibido, se devuelve un código de error.

Otro concepto importante en UNIX es el sistema de archivos montado. Casi todas las computadoras personales tienen una o más unidades ópticas en las que se pueden insertar los CD-ROMs y los DVDs. Casi siempre tienen puertos USB, a los que se pueden conectar memorias USB (en realidad son unidades de estado sólido), y algunas computadoras tienen discos flexibles o discos duros externos. Para ofrecer una manera elegante de lidiar con estos medios removibles, UNIX permite adjuntar el sistema de archivos en un CD-ROM o DVD al árbol principal. Considere la situación de la figura 1-15(a). Antes de la llamada `mount` (montar), el **sistema de archivos raíz** en el disco duro y un segundo sistema de archivos en un CD-ROM están separados y no tienen relación alguna.

Sin embargo, el sistema de archivo en el CD-ROM no se puede utilizar, debido a que no hay forma de especificar los nombres de las rutas en él. UNIX no permite colocar prefijos a los nombres de rutas basados en un nombre de unidad o un número; ese sería precisamente el tipo de dependencia de dispositivos que los sistemas operativos deben eliminar. En vez de ello, la llamada al sistema `mount` permite adjuntar el sistema de archivos en CD-ROM al sistema de archivos raíz en donde el programa desea que esté. En la figura 1-15(b) el sistema de archivos en el CD-ROM se ha montado en el directorio *b*, con lo cual se permite el acceso a los archivos */b/x* y */b/y*. Si el directorio *b* tuviera archivos, éstos no estarían accesibles mientras el CD-ROM estuviera montado, debido a que */b* haría referencia al directorio raíz del CD-ROM (el hecho de no poder acceder a estos archivos no es tan grave como parece: los sistemas de archivos casi siempre se montan en directorios vacíos). Si un sistema contiene varios discos duros, todos se pueden montar en un solo árbol también.

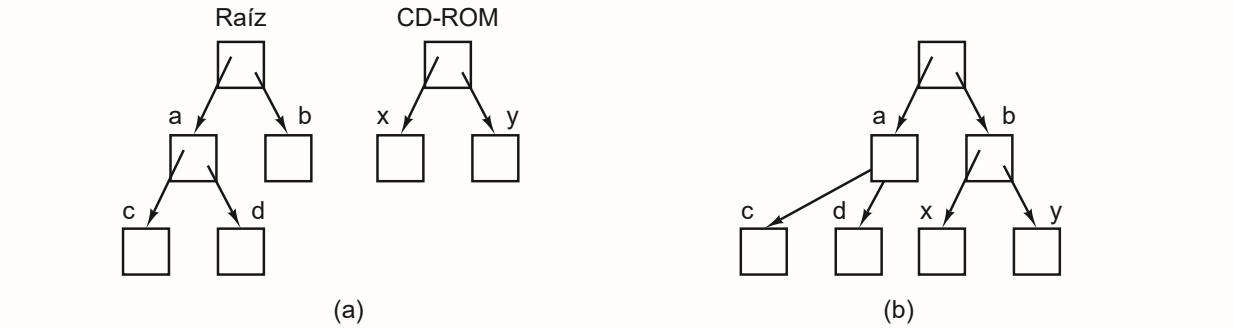

Figura 1.15. (a) Antes de montarse, los archivos en el CD-ROM no están accesibles. (b) Después de montarse, forman parte de la jerarquía de archivos.

Otro concepto importante en UNIX es el **archivo especial**. Los archivos especiales se proporcionan para poder hacer que los dispositivos de E/S se vean como archivos. De esta forma se puede leer y escribir en ellos utilizando las mismas llamadas al sistema que se utilizan para leer y escribir en archivos. Existen dos tipos de archivos especiales: **archivos especiales de bloque** y **archivos especiales de carácter**. Los archivos especiales de bloque se utilizan para modelar dispositivos que consisten en una colección de bloques direccionables al azar, tales como los discos. Al abrir un archivo especial de bloque y leer, por decir, el bloque 4, un programa puede acceder de manera directa al cuarto bloque en el dispositivo sin importar la estructura del sistema de archivos que contenga. De manera similar, los archivos especiales de carácter se utilizan para modelar impresoras, módems y otros dispositivos que aceptan o producen como salida un flujo de caracteres. Por convención, los archivos especiales se mantienen en el directorio */dev*. Por ejemplo, */dev/lp* podría ser la impresora (a la que alguna vez se le llamó impresora de línea).

La última característica que veremos en esta descripción general está relacionada con los procesos y los archivos: los canales. Un **canal (**pipe**)** es un tipo de pseudoarchivo que puede utilizarse para conectar dos procesos, como se muestra en la figura 1-16. Si los procesos *A* y *B* desean comunicarse mediante el uso de un canal, deben establecerlo por adelantado. Cuando el proceso *A* desea enviar datos al proceso *B*, escribe en el canal como si fuera un archivo de salida. De hecho, la implementación de un canal es muy parecida a la de un archivo. El proceso *B* puede leer los datos a través del canal, como si fuera un archivo de entrada. Por ende, la comunicación entre procesos en UNIX tiene una apariencia muy similar a las operaciones comunes de lectura y escritura en los archivos. Y por si fuera poco, la única manera en que un proceso puede descubrir que el archivo de salida en el que está escribiendo no es en realidad un archivo sino un canal, es mediante una llamada al sistema especial. Los sistemas de archivos son muy importantes. En los capítulos 4, 10 y 11 hablaremos mucho más sobre ellos.

1.5.4 Entrada/salida

Todas las computadoras tienen dispositivos físicos para adquirir entrada y producir salida. Después de todo, ¿qué tendría de bueno una computadora si los usuarios no pudieran indicarle qué debe hacer y no pudieran obtener los resultados una vez que realizara el trabajo solicitado? Existen muchos

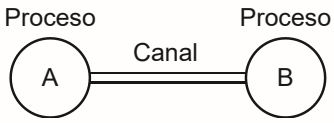

Figura 1-16. Dos procesos conectados mediante un canal.

tipos de dispositivos de entrada y de salida, incluyendo teclados, monitores, impresoras, etcétera. Es responsabilidad del sistema operativo administrar estos dispositivos.

En consecuencia, cada sistema operativo tiene un subsistema de E/S para administrar sus dispositivos de E/S. Parte del software de E/S es independiente de los dispositivos, es decir, se aplica a muchos o a todos los dispositivos de E/S por igual. Otras partes del software, como los drivers de dispositivos, son específicas para ciertos dispositivos de E/S. En el capítulo 5 analizaremos el software de E/S.

1.5.5 Protección

Las computadoras contienen grandes cantidades de información que los usuarios comúnmente desean proteger y mantener de manera confidencial. Esta información puede incluir mensajes de correo electrónico, planes de negocios, declaraciones fiscales y mucho más. Es responsabilidad del sistema operativo administrar la seguridad del sistema de manera que los archivos, por ejemplo, sólo sean accesibles para los usuarios autorizados.

Como un ejemplo simple, sólo para tener una idea de cómo puede funcionar la seguridad, considere el sistema operativo UNIX. Los archivos en UNIX están protegidos debido a que cada uno recibe un código de protección binario de 9 bits. El código de protección consiste en tres campos de 3 bits, uno para el propietario, uno para los demás miembros del grupo del propietario (el administrador del sistema divide a los usuarios en grupos) y uno para todos los demás. Cada campo tiene un bit para el acceso de lectura, un bit para el acceso de escritura y un bit para el acceso de ejecución. Estos 3 bits se conocen como los **bits rwx**. Por ejemplo, el código de protección *rwxr-x--x* indica que el propietario puede leer (**r**), escribir (**w**) o ejecutar (**x**) el archivo, otros miembros del grupo pueden leer o ejecutar (pero no escribir) el archivo y todos los demás pueden ejecutarlo (pero no leer ni escribir). Para un directorio, *x* indica el permiso de búsqueda. Un guión corto indica que no se tiene el permiso correspondiente.

Además de la protección de archivos, existen muchas otras cuestiones de seguridad. Una de ellas es proteger el sistema de los intrusos no deseados, tanto humanos como no humanos (por ejemplo, virus). En el capítulo 9 analizaremos varias cuestiones de seguridad.

1.5.6 El shell

El sistema operativo es el código que lleva a cabo las llamadas al sistema. Los editores, compiladores, ensambladores, enlazadores e intérpretes de comandos en definitiva no forman parte del sistema operativo, aun cuando son importantes y útiles. Con el riesgo de confundir un poco las cosas, en esta sección describiremos brevemente el intérprete de comandos de UNIX, conocido como

shell. Aunque no forma parte del sistema operativo, utiliza con frecuencia muchas características del mismo y, por ende, sirve como un buen ejemplo de la forma en que se pueden utilizar las llamadas al sistema. También es la interfaz principal entre un usuario sentado en su terminal y el sistema operativo, a menos que el usuario esté usando una interfaz gráfica de usuario. Existen muchos shells, incluyendo *sh*, *csh*, *ksh* y *bash*. Todos ellos soportan la funcionalidad antes descrita, que se deriva del shell original (*sh*).

Cuando cualquier usuario inicia sesión, se inicia un shell. El shell tiene la terminal como entrada estándar y salida estándar. Empieza por escribir el **indicador de comandos (prompt)**, un carácter tal como un signo de dólar, que indica al usuario que el shell está esperando aceptar un comando. Por ejemplo, si el usuario escribe

```
date
```

el shell crea un proceso hijo y ejecuta el programa *date* como el hijo. Mientras se ejecuta el proceso hijo, el shell espera a que termine. Cuando el hijo termina, el shell escribe de nuevo el indicador y trata de leer la siguiente línea de entrada.

El usuario puede especificar que la salida estándar sea redirigida a un archivo, por ejemplo:

```
date >archivo
```

De manera similar, la entrada estándar se puede redirigir, como en:

```
sort <archivo1 >archivo2
```

con lo cual se invoca el programa sort con la entrada que se recibe del *archivo1* y la salida se envía al *archivo2*.

La salida de un programa se puede utilizar como entrada para otro, si se conectan mediante un canal. Así:

```
cat archivo1 archivo2 archivo3 | sort >/dev/lp
```

invoca al programa *cat* para con*cat*enar tres archivos y enviar la salida a *sort* para ordenar todas las líneas en orden alfabético. La salida de *sort* se redirige al archivo */dev/lp*, que por lo general es la impresora.

Si un usuario coloca el signo & después de un comando, el shell no espera a que se complete. En vez de ello, proporciona un indicador de comandos de inmediato. En consecuencia:

```
cat archivo1  archivo2  archivo3  |  sort >/dev/lp &
```

inicia el comando sort como un trabajo en segundo plano y permite al usuario continuar su trabajo de manera normal mientras el ordenamiento se lleva a cabo. El shell tiene otras características interesantes, que no describiremos aquí por falta de espacio. La mayoría de los libros en UNIX describen el shell hasta cierto grado (por ejemplo, Kernighan y Pike, 1984; Kochan y Wood, 1990; Medinets, 1999; Newham y Rosenblatt, 1998; y Robbins, 1999).

Actualmente, muchas computadoras personales utilizan una GUI. De hecho, la GUI es sólo un programa que se ejecuta encima del sistema operativo, como un shell. En los sistemas Linux, este hecho se hace obvio debido a que el usuario tiene una selección de (por lo menos) dos GUIs: Gnome y KDE o ninguna (se utiliza una ventana de terminal en X11). En Windows también es posible

reemplazar el escritorio estándar de la GUI (*Windows Explorer*) con un programa distinto, para lo cual se modifican ciertos valores en el registro, aunque pocas personas hacen esto.

1.5.7 La ontogenia recapitula la filogenia

Después de que se publicó el libro de Charles Darwin titulado *El origen de las especies*, el zoólogo alemán Ernst Haeckel declaró que "la ontogenia recapitula la filogenia". Lo que quiso decir fue que el desarrollo de un embrión (ontogenia) repite (es decir, recapitula) la evolución de las especies (filogenia). En otras palabras, después de la fertilización un óvulo humano pasa a través de las etapas de ser un pez, un cerdo y así en lo sucesivo, hasta convertirse en un bebé humano. Los biólogos modernos consideran esto como una simplificación burda, pero aún así tiene cierto grado de verdad.

Algo análogo ha ocurrido en la industria de las computadoras. Cada nueva especie (mainframe, minicomputadora, computadora personal, computadora de bolsillo, computadora de sistema integrado, tarjeta inteligente, etc.) parece pasar a través del desarrollo que hicieron sus ancestros, tanto en hardware como en software. A menudo olvidamos que la mayor parte de lo que ocurre en el negocio de las computadoras y en muchos otros campos está controlado por la tecnología. La razón por la que los antiguos romanos no tenían autos no es que les gustara caminar mucho; se debió a que no sabían construirlos. Las computadoras personales existen *no* debido a que millones de personas tienen un deseo reprimido durante siglos de poseer una computadora, sino a que ahora es posible fabricarlas a un costo económico. A menudo olvidamos qué tanto afecta la tecnología a nuestra visión de los sistemas y vale la pena reflexionar sobre este punto de vez en cuando.

En especial, con frecuencia ocurre que un cambio en la tecnología hace que una idea se vuelva obsoleta y desaparece con rapidez. Sin embargo, otro cambio en la tecnología podría revivirla de nuevo. Esto es en especial verdadero cuando el cambio tiene que ver con el rendimiento relativo de distintas partes del sistema. Por ejemplo, cuando las CPUs se volvieron mucho más rápidas que las memorias, las cachés tomaron importancia para agilizar la memoria "lenta". Si algún día la nueva tecnología de memoria hace que las memorias sean mucho más rápidas que las CPUs, las cachés desaparecerán. Y si una nueva tecnología de CPUs las hace más rápidas que las memorias de nuevo, las cachés volverán a aparecer. En biología, la extinción es para siempre, pero en la ciencia computacional, algunas veces sólo es durante unos cuantos años.

Como consecuencia de esta transitoriedad, en este libro analizaremos de vez en cuando conceptos "obsoletos", es decir, ideas que no son óptimas con la tecnología actual. Sin embargo, los cambios en la tecnología pueden traer de nuevo algunos de los denominados "conceptos obsoletos". Por esta razón, es importante comprender por qué un concepto es obsoleto y qué cambios en el entorno pueden hacer que vuelva de nuevo.

Para aclarar aún más este punto consideremos un ejemplo simple. Las primeras computadoras tenían conjuntos de instrucciones cableados de forma fija. Las instrucciones se ejecutaban directamente por el hardware y no se podían modificar. Después llegó la microprogramación (primero se introdujo en gran escala con la IBM 360), en la que un intérprete subyacente ejecutaba las "instrucciones de hardware" en el software. La ejecución de instrucciones fijas se volvió obsoleta. Pero esto no era lo bastante flexible. Después se inventaron las computadoras RISC y la microprogramación

(es decir, la ejecución interpretada) se volvió obsoleta, debido a que la ejecución directa era más veloz. Ahora estamos viendo el resurgimiento de la interpretación en forma de applets de Java que se envían a través de Internet y se interpretan al momento de su llegada. La velocidad de ejecución no siempre es crucial, debido a que los retrasos en la red son tan grandes que tienden a dominar. Por ende, el péndulo ha oscilado varias veces entre la ejecución directa y la interpretación y puede volver a oscilar de nuevo en el futuro.

Memorias extensas

Ahora vamos a examinar algunos desarrollos históricos en el hardware y la forma en que han afectado al software repetidas veces. Las primeras mainframes tenían memoria limitada. Una IBM 7090 o 7094 completamente equipada, que fungió como rey de la montaña desde finales de 1959 hasta 1964, tenía cerca de 128 KB de memoria. En su mayor parte se programaba en lenguaje ensamblador y su sistema operativo estaba escrito en lenguaje ensamblador también para ahorrar la valiosa memoria.

A medida que pasaba el tiempo, los compiladores para lenguajes como FORTRAN y COBOL se hicieron lo bastante buenos como para que el lenguaje ensamblador se hiciera obsoleto. Pero cuando se liberó al mercado la primera minicomputadora comercial (PDP-1), sólo tenía 4096 palabras de 18 bits de memoria y el lenguaje ensamblador tuvo un regreso sorpresivo. Con el tiempo, las microcomputadoras adquirieron más memoria y los lenguajes de alto nivel prevalecieron.

Cuando las microcomputadoras llegaron a principios de 1980, las primeras tenían memorias de 4 KB y la programación en lenguaje ensamblador surgió de entre los muertos. A menudo, las computadoras embebidas utilizaban los mismos chips de CPU que las microcomputadoras (8080, Z80 y posteriormente 8086) y también se programaban en ensamblador al principio. Ahora sus descendientes, las computadoras personales, tienen mucha memoria y se programan en C, C++ y Java, además de otros lenguajes de alto nivel. Las tarjetas inteligentes están pasando por un desarrollo similar, aunque más allá de un cierto tamaño, a menudo tienen un intérprete de Java y ejecutan los programas de Java en forma interpretativa, en vez de que se compile Java al lenguaje máquina de la tarjeta inteligente.

Hardware de protección

Las primeras mainframes (como la IBM 7090/7094) no tenían hardware de protección, por lo que sólo ejecutaban un programa a la vez. Un programa con muchos errores podía acabar con el sistema operativo y hacer que la máquina fallara con facilidad. Con la introducción de la IBM 360, se hizo disponible una forma primitiva de protección de hardware y estas máquinas podían de esta forma contener varios programas en memoria al mismo tiempo, y dejarlos que tomaran turnos para ejecutarse (multiprogramación). La monoprogramación se declaró obsoleta.

Por lo menos hasta que apareció la primera minicomputadora (sin hardware de protección) la multiprogramación no fue posible. Aunque la PDP-1 y la PDP-8 no tenían hardware de protección, en cierto momento se agregó a la PDP-11 dando entrada a la multiprogramación y con el tiempo a UNIX.

Cuando se construyeron las primeras microcomputadoras, utilizaban el chip de CPU 8080 de Intel, que no tenía protección de hardware, por lo que regresamos de vuelta a la monoprogramación. No fue sino hasta el Intel 80286 que se agregó hardware de protección y se hizo posible la multiprogramación. Hasta la fecha, muchos sistemas integrados no tienen hardware de protección y ejecutan un solo programa.

Ahora veamos los sistemas operativos. Al principio, las primeras mainframes no tenían hardware de protección ni soporte para la multiprogramación, por lo que ejecutaban sistemas operativos simples que se encargaban de un programa cargado en forma manual a la vez. Más adelante adquirieron el soporte de hardware y del sistema operativo para manejar varios programas a la vez, después capacidades completas de tiempo compartido.

Cuando aparecieron las minicomputadoras por primera vez, tampoco tenían hardware de protección y ejecutaban un programa cargado en forma manual a la vez, aun cuando la multiprogramación estaba bien establecida en el mundo de las mainframes para ese entonces. Gradualmente adquirieron hardware de protección y la habilidad de ejecutar dos o más programas a la vez. Las primeras microcomputadoras también fueron capaces de ejecutar sólo un programa a la vez, pero más adelante adquirieron la habilidad de la multiprogramación. Las computadoras de bolsillo y las tarjetas inteligentes se fueron por la misma ruta.

En todos los casos, el desarrollo de software se rigió en base a la tecnología. Por ejemplo, las primeras microcomputadoras tenían cerca de 4 KB de memoria y no tenían hardware de protección. Los lenguajes de alto nivel y la multiprogramación eran demasiado para que un sistema tan pequeño pudiera hacerse cargo. A medida que las microcomputadoras evolucionaron en las computadoras personales modernas, adquirieron el hardware necesario y después el software necesario para manejar características más avanzadas. Es probable que este desarrollo continúe por varios años más. Otros campos también pueden tener esta rueda de reencarnaciones, pero en la industria de las computadoras parece girar con más velocidad.

Discos

Las primeras mainframes estaban en su mayor parte basadas en cinta magnética. Leían un programa de la cinta, lo compilaban, lo ejecutaban y escribían los resultados de vuelta en otra cinta. No había discos ni un concepto sobre el sistema de archivos. Eso empezó a cambiar cuando IBM introdujo el primer disco duro: el RAMAC (*RAndoM Access*, Acceso aleatorio) en 1956. Ocupaba cerca de 4 metros cuadrados de espacio de piso y podía almacenar 5 millones de caracteres de 7 bits, lo suficiente como para una fotografía digital de mediana resolución. Pero con una renta anual de 35,000 dólares, ensamblar suficientes discos como para poder almacenar el equivalente de un rollo de película era en extremo costoso. Con el tiempo los precios disminuyeron y se desarrollaron los sistemas de archivos primitivos.

Uno de los nuevos desarrollos típicos de esa época fue la CDC 6600, introducida en 1964 y considerada durante años como la computadora más rápida del mundo. Los usuarios podían crear "archivos permanentes" al darles un nombre y esperar que ningún otro usuario hubiera decidido también que, por decir, "datos" fuera un nombre adecuado para un archivo. Éste era un directorio de un solo nivel. Con el tiempo las mainframes desarrollaron sistemas de archivos jerárquicos complejos, lo cual probablemente culminó con el sistema de archivos MULTICS.

Cuando las minicomputadoras empezaron a usarse, con el tiempo también tuvieron discos duros. El disco estándar en la PDP-11 cuando se introdujo en 1970 era el disco RK05, con una capacidad de 2.5 MB, aproximadamente la mitad del RAMAC de IBM, pero sólo tenía cerca de 40 cm de diámetro y 5 cm de altura. Pero también tenía al principio un directorio de un solo nivel. Cuando llegaron las microcomputadoras, CP/M fue al principio el sistema operativo dominante y también soportaba un solo directorio en el disco (flexible).

Memoria virtual

La memoria virtual (que se describe en el capítulo 3) proporciona la habilidad de ejecutar programas más extensos que la memoria física de la computadora, llevando y trayendo pedazos entre la RAM y el disco. Pasó por un desarrollo similar, ya que apareció primero en las mainframes, después avanzó a las minis y a las micros. La memoria virtual también permitió la capacidad de ligar dinámicamente un programa a una biblioteca en tiempo de ejecución, en vez de compilarlo. MULTICS fue el primer sistema operativo en tener esta capacidad. Con el tiempo, la idea se propagó descendiendo por toda la línea y ahora se utiliza ampliamente en la mayoría de los sistemas UNIX y Windows.

En todos estos desarrollos vemos ideas que se inventaron en un contexto y más adelante se descartaron cuando cambió el contexto (la programación en lenguaje ensamblador, la monoprogramación, los directorios de un solo nivel, etc.) sólo para reaparecer en un contexto distinto, a menudo una década más tarde. Por esta razón, en este libro algunas veces vemos ideas y algoritmos que pueden parecer atrasados en comparación con las PC de hoy en día con capacidades de gigabytes, pero que pronto pueden volver en las computadoras incrustadas y las tarjetas inteligentes.

1.6 LLAMADAS AL SISTEMA

Hemos visto que los sistemas operativos tienen dos funciones principales: proveer abstracciones a los programas de usuario y administrar los recursos de la computadora. En su mayor parte, la interacción entre los programas de usuario y el sistema operativo se relaciona con la primera función: por ejemplo, crear, escribir, leer y eliminar archivos. La parte de la administración de los recursos es en gran parte transparente para los usuarios y se realiza de manera automática. Por ende, la interfaz entre los programas de usuario y el sistema operativo trata principalmente acerca de cómo lidiar con las abstracciones. Para comprender realmente qué hacen los sistemas operativos, debemos examinar esta interfaz con detalle. Las llamadas al sistema disponibles en la interfaz varían de un sistema operativo a otro (aunque los conceptos subyacentes tienden a ser similares).

Por lo tanto, nos vemos obligados a elegir una opción entre 1) generalidades imprecisas (“los sistemas operativos tienen llamadas al sistema para leer archivos”) y 2) cierto sistema específico (“UNIX tiene una llamada al sistema conocida como `read` con tres parámetros: uno para especificar el archivo, uno para decir en dónde se van a colocar los datos y uno para indicar cuantos bytes se deben leer”).

Hemos optado por la última opción; implica más trabajo, pero proporciona una visión más detallada en cuanto a lo que realmente hacen los sistemas operativos. Aunque este análisis se refiere

en forma específica a POSIX (Estándar internacional 9945-1) y por ende también a UNIX, System V, BSD, Linux, MINIX 3, etc., la mayoría de los demás sistemas operativos modernos tienen llamadas al sistema que realizan las mismas funciones, incluso si difieren los detalles. Como la verdadera mecánica relacionada con la acción de emitir una llamada al sistema es altamente dependiente de la máquina y a menudo debe expresarse en código ensamblador, se proporciona una biblioteca de procedimientos para hacer que sea posible realizar llamadas al sistema desde programas en C y por lo general desde otros lenguajes también.

Es conveniente tener en cuenta lo siguiente. Cualquier computadora con una sola CPU puede ejecutar sólo una instrucción a la vez. Si un proceso está ejecutando un programa de usuario en modo usuario y necesita un servicio del sistema, como leer datos de un archivo, tiene que ejecutar una instrucción de trap para transferir el control al sistema operativo. Después, el sistema operativo averigua qué es lo que quiere el proceso llamador, para lo cual inspecciona los parámetros. Luego lleva a cabo la llamada al sistema y devuelve el control a la instrucción que va después de la llamada al sistema. En cierto sentido, realizar una llamada al sistema es como realizar un tipo especial de llamada a un procedimiento, sólo que las llamadas al sistema entran al kernel y las llamadas a procedimientos no.

Para hacer más entendible el mecanismo de llamadas al sistema, vamos a dar un vistazo rápido a la llamada al sistema `read`. Como dijimos antes, tiene tres parámetros: el primero especifica el archivo, el segundo apunta al búfer y el tercero proporciona el número de bytes a leer. Al igual que casi todas las llamadas al sistema, se invoca desde programas en C mediante una llamada a un procedimiento de la biblioteca con el mismo nombre que la llamada al sistema: *read*. Una llamada desde un programa en C podría tener la siguiente apariencia:

```
cuenta=read(fd, bufer, nbytes);
```

La llamada al sistema (y el procedimiento de biblioteca) devuelve el número de bytes que se leen en *cuenta*. Por lo general este valor es el mismo que *nbytes* pero puede ser más pequeño si, por ejemplo, se encuentra el fin de archivo al estar leyendo.

Si la llamada al sistema no se puede llevar a cabo, ya sea debido a un parámetro inválido o a un error del disco, *cuenta* se establece a −1 y el número de error se coloca en una variable global llamada *errno*. Los programas siempre deben comprobar los resultados de una llamada al sistema para ver si ocurrió un error.

Las llamadas al sistema se llevan a cabo en una serie de pasos. Para que este concepto quede más claro, vamos a examinar la llamada `read` antes descrita. En su preparación para llamar al procedimiento de biblioteca *read*, que es quien realmente hace la llamada al sistema `read`, el programa llamador primero mete los parámetros en la pila, como se muestra en los pasos 1 a 3 de la figura 1-17.

Los compiladores de C y C++ meten los parámetros en la pila en orden inverso por razones históricas (esto tiene que ver con hacer que el primer parámetro para *printf*, la cadena del formato, aparezca en la parte superior de la pila). Los parámetros primero y tercero se pasan por valor, pero el segundo parámetro se pasa por referencia, lo cual significa que se pasa la dirección del búfer (lo cual se indica mediante &), no el contenido del mismo. Después viene la llamada al procedimiento de biblioteca (paso 4). Esta instrucción es la instrucción de llamada a procedimiento normal utilizada para llamar a todos los procedimientos.

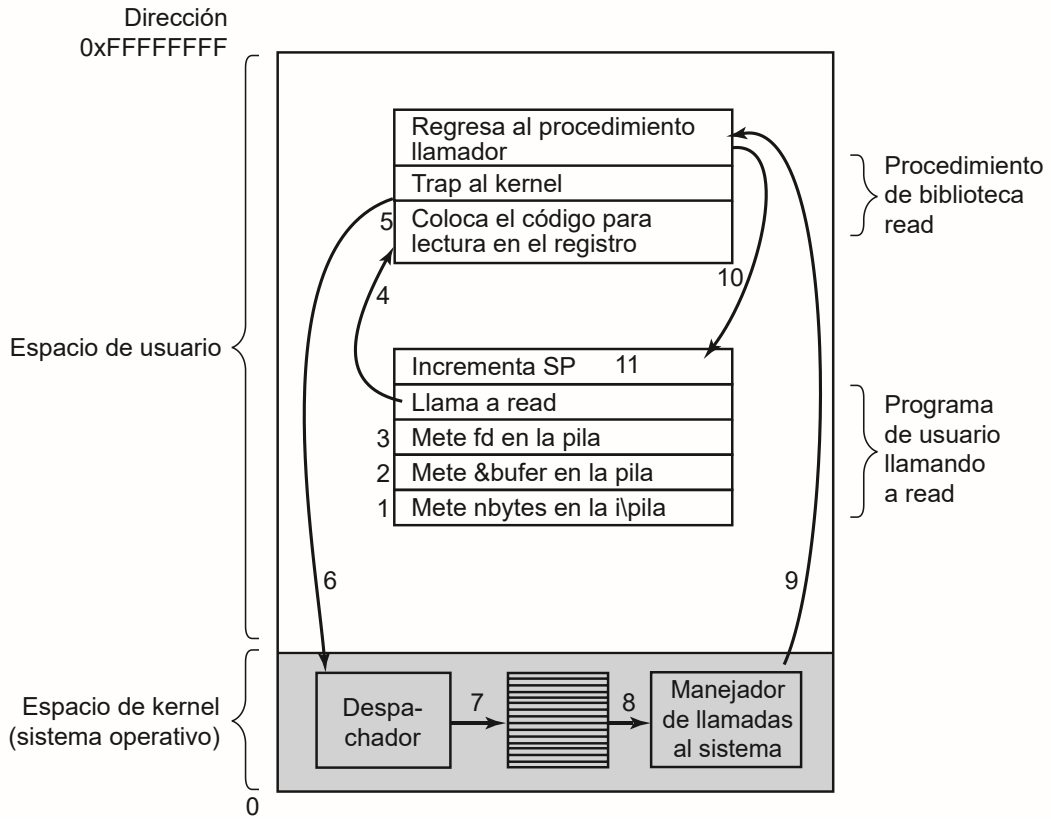

Figura 1-17. Los 11 pasos para realizar la llamada al sistema `read(fd, bufer, nbytes)`.

El procedimiento de biblioteca (probablemente escrito en lenguaje ensamblador) coloca por lo general el número de la llamada al sistema en un lugar en el que el sistema operativo lo espera, como en un registro (paso 5). Después ejecuta una instrucción `TRAP` para cambiar del modo usuario al modo kernel y empezar la ejecución en una dirección fija dentro del núcleo (paso 6). La instrucción `TRAP` en realidad es muy similar a la instrucción de llamada a procedimiento en el sentido en que la instrucción que le sigue se toma de una ubicación distante y la dirección de retorno se guarda en la pila para un uso posterior.

Sin embargo, la instrucción `TRAP` también difiere de la instrucción de llamada a un procedimiento en dos formas básicas. En primer lugar, como efecto secundario, cambia a modo kernel. La instrucción de llamada al procedimiento no cambia el modo. En segundo lugar, en vez de dar una dirección relativa o absoluta en donde se encuentra el procedimiento, la instrucción `TRAP` no puede saltar a una dirección arbitraria. Dependiendo de la arquitectura, salta a una ubicación fija, hay un campo de 8 bits en la instrucción que proporciona el índice a una tabla en memoria que contiene direcciones de salto, o su equivalente.

El código de kernel que empieza después de la instrucción `TRAP` examina el número de llamada al sistema y después la pasa al manejador correspondiente de llamadas al sistema, por lo general a través de una tabla de apuntadores a manejadores de llamadas al sistema, indexados en base al

número de llamada al sistema (paso 7). En ese momento se ejecuta el manejador de llamadas al sistema (paso 8). Una vez que el manejador ha terminado su trabajo, el control se puede regresar al procedimiento de biblioteca que está en espacio de usuario, en la instrucción que va después de la instrucción `TRAP` (paso 9). Luego este procedimiento regresa al programa de usuario en la forma usual en que regresan las llamadas a procedimientos (paso 10).

Para terminar el trabajo, el programa de usuario tiene que limpiar la pila, como lo hace después de cualquier llamada a un procedimiento (paso 11). Suponiendo que la pila crece hacia abajo, como es comúnmente el caso, el código compilado incrementa el apuntador de la pila lo suficiente como para eliminar los parámetros que se metieron antes de la llamada a *read*. Ahora el programa es libre de hacer lo que quiera a continuación.

En el paso 9 anterior, dijimos que "se puede regresar al procedimiento de biblioteca que está en espacio de usuario …" por una buena razón. La llamada al sistema puede bloquear al procedimiento llamador, evitando que continúe. Por ejemplo, si trata de leer del teclado y no se ha escrito nada aún, el procedimiento llamador tiene que ser bloqueado. En este caso, el sistema operativo buscará a su alrededor para ver si se puede ejecutar algún otro proceso a continuación. Más adelante, cuando esté disponible la entrada deseada, este proceso recibirá la atención del sistema y se llevarán a cabo los pasos 9 a 11.

En las siguientes secciones examinaremos algunas de las llamadas al sistema POSIX de uso más frecuente, o dicho en forma más específica, los procedimientos de biblioteca que realizan esas llamadas al sistema. POSIX tiene aproximadamente 100 llamadas a procedimientos. Algunas de las más importantes se listan en la figura 1-18, agrupadas por conveniencia en cuatro categorías. En el texto examinaremos brevemente cada llamada para ver cuál es su función.

En mayor grado, los servicios ofrecidos por estas llamadas determinan la mayor parte de la labor del sistema operativo, ya que la administración de recursos en las computadoras personales es una actividad mínima (por lo menos si se le compara con los equipos grandes que tienen muchos usuarios). Los servicios incluyen acciones tales como crear y terminar procesos, crear, eliminar, leer y escribir en archivos, administrar directorios y realizar operaciones de entrada y salida.

Al margen, vale la pena mencionar que la asignación de las llamadas a procedimientos POSIX a llamadas al sistema no es de uno a uno. El estándar POSIX especifica varios procedimientos que debe suministrar un sistema que se conforme a este estándar, pero no especifica si deben ser llamadas al sistema, llamadas a una biblioteca, o algo más. Si un procedimiento puede llevarse a cabo sin necesidad de invocar una llamada al sistema (es decir, sin atrapar en el kernel), por lo general se realizará en espacio de usuario por cuestión de rendimiento. Sin embargo, la mayoría de los procedimientos POSIX invocan llamadas al sistema, en donde por lo general un procedimiento se asigna directamente a una llamada al sistema. En unos cuantos casos, en especial en donde los procedimientos requeridos son sólo pequeñas variaciones de algún otro procedimiento, una llamada al sistema maneja más de una llamada a la biblioteca.

1.6.1 Llamadas al sistema para la administración de procesos

El primer grupo de llamadas en la figura 1-18 se encarga de la administración de los procesos. `fork` es un buen lugar para empezar este análisis. `fork` es la única manera de crear un nuevo proceso en POSIX. Crea un duplicado exacto del proceso original, incluyendo todos los descriptores de archivos,

Administración de procesos

Llamada	Descripción
pid = fork()	Crea un proceso hijo, idéntico al padre
pid = waitpid(pid, &statloc, opciones)	Espera a que un hijo termine
s = execve(nombre, argv, entornp)	Reemplaza la imagen del núcleo de un proceso
exit(estado)	Termina la ejecución de un proceso y devuelve el estado

Administración de archivos

Llamada	Descripción
fd = open(archivo, como, …)	Abre un archivo para lectura, escritura o ambas
s = close(fd)	Cierra un archivo abierto
n = read(fd, bufer, nbytes)	Lee datos de un archivo y los coloca en un búfer
n = write(fd, bufer, nbytes)	Escribe datos de un búfer a un archivo
posicion = lseek(fd, desplazamiento, dedonde)	Desplaza el apuntador del archivo
s = stat(nombre, &buf)	Obtiene la información de estado de un archivo

Administración del sistema de directorios y archivos

Llamada	Descripción
s = mkdir(nombre, modo)	Crea un nuevo directorio
s = rmdir(nombre)	Elimina un directorio vacío
s = link(nombre1, nombre2)	Crea una nueva entrada llamada nombre2, que apunta a nombre1
s = unlink(nombre)	Elimina una entrada de directorio
s = mount(especial, nombre, bandera)	Monta un sistema de archivos
s = umount(especial)	Desmonta un sistema de archivos

Llamadas varias

Llamada	Descripción
s = chdir(nombredir)	Cambia el directorio de trabajo
s = chmod(nombre, modo)	Cambia los bits de protección de un archivo
s = kill(pid, senial)	Envía una señal a un proceso
segundos = tiempo(&segundos)	Obtiene el tiempo transcurrido desde Ene 1, 1970

Figura 1-18. Algunas de las principales llamadas al sistema POSIX. El código de retorno *s* es −1 si ocurrió un error. Los códigos de retorno son: *pid* es un id de proceso, *fd* es un descriptor de archivo, *n* es una cuenta de bytes, *posicion* es un desplazamiento dentro del archivo y *segundos* es el tiempo transcurrido. Los parámetros se explican en el texto.

registros y todo lo demás. Después de `fork`, el proceso original y la copia (el padre y el hijo) se van por caminos separados. Todas las variables tienen valores idénticos al momento de la llamada a fork, pero como los datos del padre se copian para crear al hijo, los posteriores cambios en uno de ellos no afectarán al otro (el texto del programa, que no se puede modificar, se comparte entre el padre y el hijo). La llamada a `fork` devuelve un valor, que es cero en el hijo e igual al identificador del proceso (`PID`) hijo en el padre. Mediante el uso del PID devuelto, los dos procesos pueden ver cuál es el proceso padre y cuál es el proceso hijo.

En la mayoría de los casos, después de una llamada a `fork` el hijo tendrá que ejecutar código distinto al del padre. Considere el caso del shell: lee un comando de la terminal, llama a fork para crear un proceso hijo, espera a que el hijo ejecute el comando y después lee el siguiente comando cuando el hijo termina. Para esperar a que el hijo termine, el padre ejecuta una llamada al sistema `waitpid`, la cual sólo espera hasta que el hijo termine (cualquier hijo, si existe más de uno). `Waitpid` puede esperar a un hijo específico o a cualquier hijo anterior si establece el primer parámetro a −1. Cuando `waitpid` se completa, la dirección a la que apunta el segundo parámetro (*statloc*) se establece al estado de salida del hijo (terminación normal o anormal, con el valor de `exit`). También se proporcionan varias opciones, especificadas por el tercer parámetro.

Ahora considere la forma en que el shell utiliza a `fork`. Cuando se escribe un comando, el shell crea un nuevo proceso usando `fork`. Este proceso hijo debe ejecutar el comando de usuario. Para ello utiliza la llamada al sistema `execve`, la cual hace que toda su imagen de núcleo completa se sustituya por el archivo nombrado en su primer parámetro. (En realidad, la llamada al sistema en sí es `exec`, pero varios procedimientos de biblioteca lo llaman con distintos parámetros y nombres ligeramente diferentes. Aquí trataremos a estas llamadas como si fueran llamadas al sistema.) En la figura 1-19 se muestra un shell muy simplificado que ilustra el uso de `fork`, `waitpid` y `execve`.

```
#define TRUE 1

while (TRUE) {                                  /* se repite en forma indefinida */
        type_prompt();                          /* muestra el indicador de comando en la pantalla */
        read_command(command, parameters);      /* lee la entrada de la terminal */

        if (fork() !=0) {                       /* usa fork para el proceso hijo */
                /* Codigo del padre. */
                waitpid(-1, &status, 0);        /* espera a que el hijo termine */
        } else {
                /* Codigo del hijo. */
                execve(command, parameters, 0); /* ejecuta el comando */
        }
}
```

Figura 1-19. Una versión simplificada del shell. En este libro supondremos que *TRUE* se define como 1.

En el caso más general, `execve` tiene tres parámetros: el nombre del archivo que se va a ejecutar, un apuntador al arreglo de argumentos y un apuntador al arreglo del entorno. En breve describiremos estos parámetros. Se proporcionan varias rutinas de biblioteca incluyendo a *execl*, *execv*,

execle y *execve*, para permitir la omisión de los parámetros o para especificarlos de varias formas. En este libro utilizaremos el nombre `exec` para representar la llamada al sistema que se invoca mediante cada una de estas rutinas.

Consideremos el caso de un comando tal como

```
cp archivo1 archivo2
```

que se utiliza para copiar *archivo1* a *archivo2*. Una vez que el shell se ha bifurcado mediante `fork`, el proceso hijo localiza y ejecuta el archivo *cp* y le pasa los nombres de los archivos de origen y destino.

El programa principal de *cp* (y el programa principal de la mayoría de los otros programas en C) contienen la siguiente declaración:

```
main(argc, argv, envp)
```

en donde *argc* es una cuenta del número de elementos en la línea de comandos, incluyendo el nombre del programa. Para el ejemplo anterior, *argc* es 3.

El segundo parámetro, *argv*, es un apuntador a un arreglo. El elemento *i* de ese arreglo es un apuntador a la *i*-ésima cadena en la línea de comandos. En nuestro ejemplo, *argv*[0] apuntaría a la cadena "cp", *argv*[1] apuntaría a la cadena "archivo1" y *argv*[2] apuntaría a la cadena "archivo2".

El tercer parámetro de *main*, *envp*, es un apuntador al entorno, un arreglo de cadenas que contiene asignaciones de la forma *nombre = valor* que se utilizan para pasar información, tal como el tipo de terminal y el nombre del directorio de inicio, a los programas. Hay procedimientos de biblioteca que los programas pueden llamar para obtener las variables de entorno, que a menudo se utilizan para personalizar la forma en que un usuario desea realizar ciertas tareas (por ejemplo, la impresora predeterminada que desea utilizar). En la figura 1-19 no se pasa un entorno al hijo, por lo que el tercer parámetro de *execve* es un cero.

Si `exec` parece complicado, no se desanime; es (en sentido semántico) la más compleja de todas las llamadas al sistema POSIX. Todas las demás son mucho más simples. Como ejemplo de una llamada simple considere a `exit`, que los procesos deben utilizar cuando terminan su ejecución. Tiene un parámetro, el estado de `exit` (0 a 255), que se devuelve al padre mediante *statloc* en la llamada al sistema `waitpid`.

En UNIX los procesos tienen su memoria dividida en tres segmentos: el **segmento de texto** (es decir, el código del programa), el **segmento de datos** (es decir, las variables) y el **segmento de pila**. El segmento de datos crece hacia arriba y la pila crece hacia abajo, como se muestra en la figura 1-20. Entre ellos hay un espacio libre de direcciones sin utilizar. La pila crece hacia ese espacio de manera automática, según sea necesario, pero la expansión del segmento de datos se realiza de manera explícita mediante una llamada al sistema (`brk`), la cual especifica la nueva dirección en donde debe terminar el segmento de datos. Sin embargo, esta llamada no está definida en el estándar de POSIX, ya que se recomienda a los programadores utilizar el procedimiento de biblioteca *malloc* para asignar espacio de almacenamiento en forma dinámica y la implementación subyacente de *malloc* no se consideró como un tema adecuado para su estandarización, ya que pocos programadores lo utilizan directamente y es improbable que alguien se dé cuenta siquiera que brk no está en POSIX.

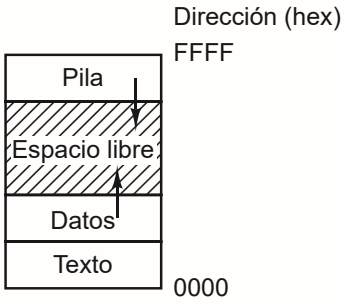

Figura 1-20. Los procesos tienen tres segmentos: de texto, de datos y de pila.

1.6.2 Llamadas al sistema para la administración de archivos

Muchas llamadas al sistema se relacionan con el sistema de archivos. En esta sección analizaremos las llamadas que operan con archivos individuales; en la siguiente sección examinaremos las llamadas que implican el uso de directorios o el sistema de archivos como un todo.

Para leer o escribir en un archivo, éste debe primero abrirse mediante open. Esta llamada especifica el nombre del archivo que se va a abrir, ya sea como un nombre de ruta absoluto o relativo al directorio de trabajo, y un código de *O_RDONLY*, *O_WRONLY* o *O_RDWR*, que significa abrir para lectura, escritura o ambos. Para crear un nuevo archivo se utiliza el parámetro *O_CREAT*. Después se puede utilizar el descriptor de archivo devuelto para leer o escribir. Al terminar, el archivo se puede cerrar mediante `close`, que hace que el descriptor de archivo esté disponible para reutilizarlo en una llamada a `open` posterior.

Las llamadas de uso más frecuente son sin duda `read` y `write`. Anteriormente vimos a `read`. `Write` tiene los mismos parámetros.

Aunque la mayoría de los programas leen y escriben archivos en forma secuencial, para ciertas aplicaciones los programas necesitan la capacidad de acceder a cualquier parte del archivo en forma aleatoria. Con cada archivo hay un apuntador asociado, el cual indica la posición actual en el archivo. Al leer (escribir) en forma secuencial, por lo general apunta al siguiente byte que se va a leer (escribir). La llamada a `lseek` cambia el valor del apuntador de posición, de manera que las siguientes llamadas a `read` o `write` puedan empezar en cualquier parte del archivo.

`Lseek` tiene tres parámetros: el primero es el descriptor del archivo, el segundo es una posición en el archivo y el tercero indica si la posición en el archivo es relativa al inicio del mismo, a la posición actual o al final del archivo. El valor devuelto por `lseek` es la posición absoluta en el archivo (en bytes) después de modificar el apuntador.

Para cada archivo, UNIX lleva el registro del modo del archivo (archivo regular, especial, directorio, etcétera), su tamaño, la hora de la última modificación y demás información. Para ver esta información, los programas pueden utilizar la llamada al sistema `stat`. El primer parámetro especifica el archivo que se va a inspeccionar; el segundo es un apuntador a una estructura en donde se va a colocar la información. Las llamadas al sistema fstat hacen lo mismo para un archivo abierto.

1.6.3 Llamadas al sistema para la administración de directorios

En esta sección analizaremos algunas llamadas al sistema que se relacionan más con los directorios o con el sistema de archivos como un todo, en vez de relacionarse sólo con un archivo específico, como en la sección anterior. Las primeras dos llamadas, `mkdir` y `rmdir`, crean y eliminan directorios vacíos, respectivamente. La siguiente llamada es `link`. Su propósito es permitir que aparezca el mismo archivo bajo dos o más nombres, a menudo en distintos directorios. Un uso común es para permitir que varios miembros del mismo equipo de programación compartan un archivo común, en donde cada uno de ellos puede ver el archivo en su propio directorio, posiblemente bajo distintos nombres. No es lo mismo compartir un archivo que proporcionar a cada miembro del equipo una copia privada; tener un archivo compartido significa que los cambios que realice cualquier miembro del equipo serán visibles de manera instantánea para los otros miembros; sólo hay un archivo. Cuando se realizan copias de un archivo, los cambios subsiguientes que se realizan en una copia no afectan a las demás.

Para ver cómo funciona `link`, considere la situación de la figura 1-21(a). Aquí hay dos usuarios, *ast* y *jim*, y cada uno tiene su propio directorio con algunos archivos. Si *ast* ejecuta ahora un programa que contenga la llamada al sistema

```
link("/usr/jim/memo", "/usr/ast/nota");
```

el archivo *memo* en el directorio de *jim* se introduce en el directorio de *ast* bajo el nombre *nota*. De aquí en adelante, */usr/jim/memo* y */usr/ast/nota* harán referencia al mismo archivo. Para complementar, hay que tener en cuenta que el lugar en el que se mantengan los directorios de los usuarios, ya sea en */usr*, */user*, */home* o en alguna otra parte es simplemente una decisión que realiza el administrador del sistema local.

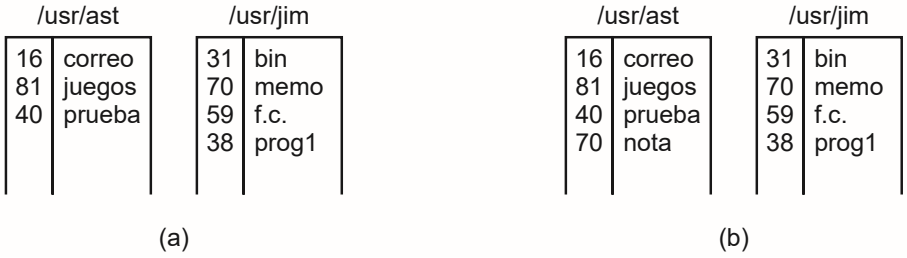

Figura 1-21. a) Dos directorios antes de enlazar */usr/jim/memo* al directorio de *ast*. b) Los mismos directorios después del enlace.

Entendiendo cómo funciona `link` probablemente nos aclare lo que hace. Todo archivo en UNIX tiene un número único, su número-i, que lo identifica. Este número-i es un índice en una tabla de **nodos-i**, uno por archivo, que indican quién es propietario del archivo, en dónde están sus bloques de disco, etcétera. Un directorio es simplemente un archivo que contiene un conjunto de pares (número-i, nombre ASCII). En las primeras versiones de UNIX, cada entrada de directorio era de 16 bytes: 2 bytes para el número-i y 14 bytes para el nombre. Ahora se necesita una estructura más complicada para soportar los nombres de archivo extensos, pero en concepto un directorio sigue siendo un conjunto de pares (número-i, nombre ASCII). En la figura 1-21, *correo* tiene el número-i 16, y así sucesivamente. Lo que `link` hace es tan sólo crear una nueva entrada de directorio con un nombre

(posiblemente nuevo), usando el número-i de un archivo existente. En la figura 1-21(b), dos entradas tienen el mismo número-i (70) y por ende se refieren al mismo archivo. Si más adelante se elimina una de las dos mediante la llamada al sistema `unlink`, la otra sigue vigente. Si se eliminan ambas, UNIX ve que no existen entradas para el archivo (un campo en el nodo-i lleva la cuenta del número de entradas de directorio que apuntan al archivo), por lo que el archivo se elimina del disco.

Como dijimos antes, la llamada al sistema `mount` permite combinar dos sistemas de archivos en uno. Una situación común es hacer que el sistema de archivos raíz contenga las versiones binarias (ejecutables) de los comandos comunes y otros archivos de uso frecuente, en un disco duro. Así, el usuario puede insertar un disco de CD-ROM con los archivos que se van a leer en la unidad de CD-ROM.

Al ejecutar la llamada al sistema `mount`, el sistema de archivos de CD-ROM se puede adjuntar al sistema de archivos raíz, como se muestra en la figura 1-22. Una instrucción común en C para realizar el montaje es

```
mount("/dev/fd0", "/mnt", 0);
```

donde el primer parámetro es el nombre de un archivo especial de bloque para la unidad 0, el segundo parámetro es la posición en el árbol en donde se va a montar y el tercer parámetro indica que el sistema de archivo se va a montar en modo de lectura-escritura o de sólo escritura.

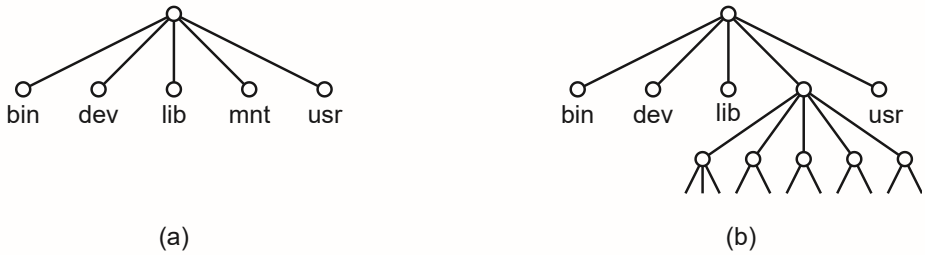

Figura 1-22. (a) Sistema de archivos antes del montaje. (b) Sistema de archivos después del montaje.

Después de la llamada a `mount`, se puede tener acceso a un archivo en la unidad 0 con sólo utilizar su ruta del directorio raíz o del directorio de trabajo, sin importar en cuál unidad se encuentre. De hecho, las unidades segunda, tercera y cuarta también se pueden montar en cualquier parte del árbol. La llamada a `mount` hace posible integrar los medios removibles en una sola jerarquía de archivos integrada, sin importar en qué dispositivo se encuentra un archivo. Aunque este ejemplo involucra el uso de CD-ROMs, también se pueden montar porciones de los discos duros (que a menudo se les conoce como **particiones** o **dispositivos menores**) de esta forma, así como los discos duros y memorias USB externos. Cuando ya no se necesita un sistema de archivos, se puede desmontar mediante la llamada al sistema `umount`.

1.6.4 Miscelánea de llamadas al sistema

También existe una variedad de otras llamadas al sistema. Sólo analizaremos cuatro de ellas aquí. La llamada a `chdir` cambia el directorio de trabajo actual. Después de la llamada

```
chdir("/usr/ast/prueba");
```

una instrucción para abrir el archivo *xyz* abrirá */usr/ast/prueba/xyz*. El concepto de un directorio de trabajo elimina la necesidad de escribir nombres de ruta absolutos (extensos) todo el tiempo.

En UNIX, cada archivo tiene un modo que se utiliza por protección. El modo incluye los bits leer-escribir-ejecutar para el propietario, grupo y los demás. La llamada al sistema chmod hace posible modificar el modo de un archivo. Por ejemplo, para que un archivo sea de sólo lectura para todos excepto el propietario, podríamos ejecutar

```
chmod("archivo", 0644);
```

La llamada al sistema kill es la forma en que los usuarios y los procesos de usuario envían señales. Si un proceso está preparado para atrapar una señal específica, y luego ésta llega, se ejecuta un manejador de señales. Si el proceso no está preparado para manejar una señal, entonces su llegada mata el proceso (de aquí que se utilice ese nombre para la llamada).

POSIX define varios procedimientos para tratar con el tiempo. Por ejemplo, time sólo devuelve la hora actual en segundos, en donde 0 corresponde a Enero 1, 1970 a medianoche (justo cuando el día empezaba y no terminaba). En las computadoras que utilizan palabras de 32 bits, el valor máximo que puede devolver time es de $2^{32} - 1$ segundos (suponiendo que se utiliza un entero sin signo). Este valor corresponde a un poco más de 136 años. Así, en el año 2106 los sistemas UNIX de 32 bits se volverán locos, algo parecido al famoso problema del año 2000 (Y2K) que hubiera causado estragos con las computadoras del mundo en el 2000, si no fuera por el masivo esfuerzo que puso la industria de IT para corregir el problema. Si actualmente usted tiene un sistema UNIX de 32 bits, se le recomienda que lo intercambie por uno de 64 bits antes del año 2106.

1.6.5 La API Win32 de Windows

Hasta ahora nos hemos enfocado principalmente en UNIX. Es tiempo de dar un vistazo breve a Windows. Windows y UNIX difieren de una manera fundamental en sus respectivos modelos de programación. Un programa de UNIX consiste en código que realiza una cosa u otra, haciendo llamadas al sistema para realizar ciertos servicios. En contraste, un programa de Windows es por lo general manejado por eventos. El programa principal espera a que ocurra cierto evento y después llama a un procedimiento para manejarlo. Los eventos comunes son las teclas que se oprimen, el ratón que se desplaza, un botón de ratón que se oprime o un CD-ROM que se inserta. Después, los manejadores se llaman para procesar el evento, actualizar la pantalla y actualizar el estado interno del programa. En todo, esto produce un estilo algo distinto de programación que con UNIX, pero debido a que el enfoque de este libro es acerca de la función y la estructura del sistema operativo, estos distintos modelos de programación no nos preocuparán por mucho.

Desde luego que Windows también tiene llamadas al sistema. Con UNIX, hay casi una relación de uno a uno entre las llamadas al sistema (por ejemplo, `read`) y los procedimientos de biblioteca (por ejemplo, *read*) que se utilizan para invocar las llamadas al sistema. En otras palabras, para cada llamada al sistema, es raro que haya un procedimiento de biblioteca que sea llamado para invocarlo, como se indica en la figura 1-17. Lo que es más, POSIX tiene aproximadamente 100 llamadas a procedimientos.

Con Windows, la situación es bastante distinta. Para empezar, las llamadas a la biblioteca y las llamadas al sistema están muy desacopladas. Microsoft ha definido un conjunto de procedimientos conocidos como **API Win32** (*Application Program Interface*, Interfaz de programación de aplicaciones) que los programadores deben utilizar para obtener los servicios del sistema operativo. Esta interfaz se proporciona (parcialmente) en todas las versiones de Windows, desde Windows 95. Al desacoplar la interfaz de las llamadas actuales al sistema, Microsoft retiene la habilidad de modificar las llamadas al sistema en el tiempo (incluso de versión en versión) sin invalidar los programas existentes. Lo que constituye realmente a Win32 es también ligeramente ambiguo, debido a que Windows 2000, Windows XP y Windows Vista tienen muchas nuevas llamadas que antes no estaban disponibles. En esta sección Win32 significa la interfaz que soportan todas las versiones de Windows.

El número de llamadas a la API Win32 es muy grande, alrededor de los miles. Lo que es más, aunque muchas de ellas invocan llamadas al sistema, un número considerable de las mismas se lleva a cabo completamente en espacio de usuario. Como consecuencia, con Windows es imposible ver lo que es una llamada al sistema (el kernel se encarga de ello) y qué es simplemente una llamada a la biblioteca en espacio de usuario. De hecho, lo que es una llamada al sistema en una versión de Windows se puede realizar en espacio de usuario en una versión diferente y viceversa. Cuando hablemos sobre las llamadas al sistema Windows en este libro, utilizaremos los procedimientos de Win32 (en donde sea apropiado), ya que Microsoft garantiza que éstos serán estables a través del tiempo. Pero vale la pena recordar que no todos ellos son verdaderas llamadas al sistema (es decir, traps al kernel).

La API Win32 tiene un gran número de llamadas para administrar ventanas, figuras geométricas, texto, tipos de letras, barras de desplazamiento, cuadros de diálogo, menús y otras características de la GUI. Hasta el grado de en que el subsistema de gráficos se ejecute en el kernel (lo cual es cierto en algunas versiones de Windows, pero no en todas), éstas son llamadas al sistema; en caso contrario, sólo son llamadas a la biblioteca. ¿Debemos hablar sobre estas llamadas en este libro o no? Como en realidad no se relacionan con la función de un sistema operativo, hemos decidido que no, aun cuando el kernel puede llevarlas a cabo. Los lectores interesados en la API Win32 pueden consultar uno de los muchos libros acerca del tema (por ejemplo, Hart, 1997; Rector y Newcomer, 1997; y Simon, 1997).

Incluso está fuera de cuestión introducir todas las llamadas a la API Win32 aquí, por lo que nos restringiremos a las llamadas que apenas si corresponden a la funcionalidad de las llamadas de Unix que se listan en la figura 1-18. Éstas se listan en la figura 1-23.

Ahora daremos un breve repaso a la lista de la figura 1-23. `CreateProcess` crea un proceso. Realiza el trabajo combinado de `fork` y `execve` en UNIX. Tiene muchos parámetros que especifican las propiedades del proceso recién creado. Windows no tiene una jerarquía de procesos como UNIX, por lo que no hay un concepto de un proceso padre y un proceso hijo. Una vez que se crea un proceso, el creador y el creado son iguales. `WaitForSingleObject` se utiliza para esperar un evento. Se pueden esperar muchos eventos posibles. Si el parámetro especifica un proceso, entonces el proceso llamador espera a que el proceso especificado termine, lo cual se hace mediante `ExitProcess`.

Las siguientes seis llamadas operan con archivo y tienen una funcionalidad similar a sus contrapartes de UNIX, aunque difieren en cuanto a los parámetros y los detalles. Aún así, los archivos

UNIX	Win32	Descripción
fork	CreateProcess	Crea un nuevo proceso
waitpid	WaitForSingleObject	Puede esperar a que un proceso termine
execve	(ninguno)	CreateProces = fork + execve
exit	ExitProcess	Termina la ejecución
open	CreateFile	Crea un archivo o abre uno existente
close	CloseHandle	Cierra un archivo
read	ReadFile	Lee datos de un archivo
write	WriteFile	Escribe datos en un archivo
lseek	SetFilePointer	Desplaza el apuntador del archivo
stat	GetFileAttributesEx	Obtiene varios atributos de un archivo
mkdir	CreateDirectory	Crea un nuevo directorio
rmdir	RemoveDirectory	Elimina un directorio vacío
link	(ninguno)	Win32 no soporta los enlaces
unlink	DeleteFile	Destruye un archivo existente
mount	(ninguno)	Win32 no soporta el montaje
umount	(ninguno)	Win32 no soporta el montaje
chdir	SetCurrentDirectory	Cambia el directorio de trabajo actual
chmod	(ninguno)	Win32 no soporta la seguridad (aunque NT sí)
kill	(ninguno)	Win32 no soporta las señales
time	GetLocalTime	Obtiene la hora actual

Figura 1-23. Las llamadas a la API Win32 que apenas corresponden a las llamadas de UNIX de la figura 1-18.

se pueden abrir, cerrar, leer y escribir en ellos en forma muy parecida a UNIX. Las llamadas `SetFilePointer` y `GetFileAttributesEx` establecen la posición de un archivo y obtienen algunos de sus atributos.

Windows tiene directorios, que se crean y eliminan con las llamadas a la API `CreateDirectory` y `RemoveDirectory`, respectivamente. También hay una noción de un directorio actual, el cual se establece mediante `SetCurrentDirectory`. La hora actual del día se adquiere mediante el uso de `GetLocalTime`.

La interfaz de Win32 no tiene enlaces a archivos, sistemas de archivos montados, seguridad ni señales, por lo que las llamadas correspondientes a las de UNIX no existen. Desde luego que Win32 tiene un gran número de otras llamadas que UNIX no tiene, en especial para administrar la GUI. Y Windows Vista tiene un elaborado sistema de seguridad, además de que también soporta los enlaces de archivos.

Tal vez vale la pena hacer una última observación acerca de Win32: no es una interfaz tan uniforme o consistente. La principal culpabilidad de esto fue la necesidad de tener compatibilidad hacia atrás con la interfaz de 16 bits anterior utilizada en Windows 3.x.

1.7 ESTRUCTURA DE UN SISTEMA OPERATIVO

Ahora que hemos visto la apariencia exterior de los sistemas operativos (es decir, la interfaz del programador), es tiempo de dar un vistazo a su interior. En las siguientes secciones analizaremos seis estructuras distintas que se han probado, para poder darnos una idea del espectro de posibilidades. De ninguna manera quiere esto decir que sean exhaustivas, pero nos dan una idea de algunos diseños que se han probado en la práctica. Los seis diseños son: sistemas monolíticos, sistemas de capas, microkernels, sistemas cliente-servidor, máquinas virtuales y exokernels.

1.7.1 Sistemas monolíticos

En este diseño, que hasta ahora se considera como la organización más común, todo el sistema operativo se ejecuta como un solo programa en modo kernel. El sistema operativo se escribe como una colección de procedimientos, enlazados entre sí en un solo programa binario ejecutable extenso. Cuando se utiliza esta técnica, cada procedimiento en el sistema tiene la libertad de llamar a cualquier otro, si éste proporciona cierto cómputo útil que el primero necesita. Al tener miles de procedimientos que se pueden llamar entre sí sin restricción, con frecuencia se produce un sistema poco manejable y difícil de comprender.

Para construir el programa objeto actual del sistema operativo cuando se utiliza este diseño, primero se compilan todos los procedimientos individuales (o los archivos que contienen los procedimientos) y luego se vinculan en conjunto para formar un solo archivo ejecutable, usando el enlazador del sistema. En términos de ocultamiento de información, en esencia no hay nada: todos los procedimientos son visibles para cualquier otro procedimiento (en contraste a una estructura que contenga módulos o paquetes, en donde la mayor parte de la información se oculta dentro de módulos y sólo los puntos de entrada designados de manera oficial se pueden llamar desde el exterior del módulo).

Sin embargo, hasta en los sistemas monolíticos es posible tener cierta estructura. Para solicitar los servicios (llamadas al sistema) que proporciona el sistema operativo, los parámetros se colocan en un lugar bien definido (por ejemplo, en la pila) y luego se ejecuta una instrucción de trap. Esta instrucción cambia la máquina del modo usuario al modo kernel y transfiere el control al sistema operativo, lo cual se muestra como el paso 6 en la figura 1-17. Después el sistema operativo obtiene los parámetros y determina cuál es la llamada al sistema que se va a llevar a cabo. Después la indiza en una tabla que contiene en la ranura *k* un apuntador al procedimiento que lleva a cabo la llamada al sistema *k* (paso 7 en la figura 1-17).

Esta organización sugiere una estructura básica para el sistema operativo:

1. Un programa principal que invoca el procedimiento de servicio solicitado.
2. Un conjunto de procedimientos de servicio que llevan a cabo las llamadas al sistema.
3. Un conjunto de procedimientos utilitarios que ayudan a los procedimientos de servicio.

En este modelo, para cada llamada al sistema hay un procedimiento de servicio que se encarga de la llamada y la ejecuta. Los procedimientos utilitarios hacen cosas que necesitan varios procedi-

mientos de servicio, como obtener datos de los programas de usuario. Esta división de los procedimientos en tres niveles se muestra en la figura 1-24.

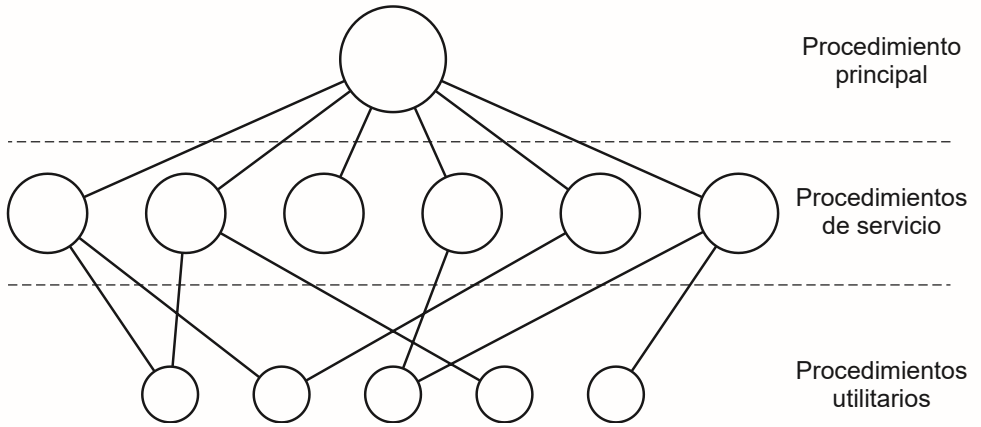

Figura 1-24. Un modelo de estructuración simple para un sistema monolítico.

Además del núcleo del sistema operativo que se carga al arrancar la computadora, muchos sistemas operativos soportan extensiones que se pueden cargar, como los drivers de dispositivos de E/S y sistemas de archivos. Estos componentes se cargan por demanda.

1.7.2 Sistemas de capas

Una generalización del diseño de la figura 1-24 es organizar el sistema operativo como una jerarquía de capas, cada una construida encima de la que tiene abajo. El primer sistema construido de esta forma fue el sistema THE, construido en Technische Hogeschool Eindhoven en Holanda por E. W. Dijkstra (1968) y sus estudiantes. El sistema THE era un sistema simple de procesamiento por lotes para una computadora holandesa, la Electrologica X8, que tenía 32K de palabras de 27 bits (los bits eran costosos en aquel entonces).

El sistema tenía seis capas, como se muestra en la figura 1-25. El nivel 0 se encargaba de la asignación del procesador, de cambiar entre un proceso y otro cuando ocurrían interrupciones o expiraban los temporizadores. Por encima del nivel 0, el sistema consistía en procesos secuenciales, cada uno de los cuales e podía programar sin necesidad de preocuparse por el hecho de que había varios procesos en ejecución en un solo procesador. En otras palabras, el nivel 0 proporcionaba la multiprogramación básica de la CPU.

La capa 1 se encargaba de la administración de la memoria. Asignaba espacio para los procesos en la memoria principal y en un tambor de palabras de 512 K que se utilizaba para contener partes de procesos (páginas), para los que no había espacio en la memoria principal. Por encima de la capa 1, los procesos no tenían que preocuparse acerca de si estaban en memoria o en el tambor; el software de la capa 1 se encargaba de asegurar que las páginas se llevaran a memoria cuando se requerían.

Capa	Función
5	El operador
4	Programas de usuario
3	Administración de la entrada/salida
2	Comunicación operador-proceso
1	Administración de memoria y tambor
0	Asignación del procesador y multiprogramación

Figura 1-25. Estructura del sistema operativo THE.

La capa 2 se encargaba de la comunicación entre cada proceso y la consola del operador (es decir, el usuario). Encima de esta capa, cada proceso tenía en efecto su propia consola de operador. La capa 3 se encargaba de administrar los dispositivos de E/S y de guardar en búferes los flujos de información dirigidos para y desde ellos. Encima de la capa 3, cada proceso podía trabajar con los dispositivos abstractos de E/S con excelentes propiedades, en vez de los dispositivos reales con muchas peculiaridades. La capa 4 era en donde se encontraban los programas de usuario. No tenían que preocuparse por la administración de los procesos, la memoria, la consola o la E/S. El proceso operador del sistema se encontraba en el nivel 5.

Una mayor generalización del concepto de capas estaba presente en el sistema MULTICS. En vez de capa, MULTICS se describió como una serie de anillos concéntricos, en donde los interiores tenían más privilegios que los exteriores (que en efecto viene siendo lo mismo). Cuando un procedimiento en un anillo exterior quería llamar a un procedimiento en un anillo interior, tenía que hacer el equivalente de una llamada al sistema; es decir, una instrucción `TRAP` cuyos parámetros se comprobara cuidadosamente que fueran válidos antes de permitir que continuara la llamada. Aunque todo el sistema operativo era parte del espacio de direcciones de cada proceso de usuario en MULTICS, el hardware hizo posible que se designaran procedimientos individuales (en realidad, segmentos de memoria) como protegidos contra lectura, escritura o ejecución.

Mientras que en realidad el esquema de capas de THE era sólo una ayuda de diseño, debido a que todas las partes del sistema estaban enlazadas entre sí en un solo programa ejecutable, en MULTICS el mecanismo de los anillos estaba muy presente en tiempo de ejecución y el hardware se encargaba de implementarlo. La ventaja del mecanismo de los anillos es que se puede extender fácilmente para estructurar los subsistemas de usuario. Por ejemplo, un profesor podría escribir un programa para evaluar y calificar los programas de los estudiantes, ejecutando este programa en el anillo n, mientras que los programas de los estudiantes se ejecutaban en el anillo $n + 1$ y por ende no podían cambiar sus calificaciones.

1.7.3 Microkernels

Con el diseño de capas, los diseñadores podían elegir en dónde dibujar el límite entre kernel y usuario. Tradicionalmente todos las capas iban al kernel, pero eso no es necesario. De hecho, puede tener mucho sentido poner lo menos que sea posible en modo kernel, debido a que los errores en el

kernel pueden paralizar el sistema de inmediato. En contraste, los procesos de usuario se pueden configurar para que tengan menos poder, por lo que un error en ellos tal vez no sería fatal.

Varios investigadores han estudiado el número de errores por cada 1000 líneas de código (por ejemplo, Basilli y Perricone, 1984; y Ostrand y Weyuker, 2002). La densidad de los errores depende del tamaño del módulo, su tiempo de vida y más, pero una cifra aproximada para los sistemas industriales formales es de diez errores por cada mil líneas de código. Esto significa que es probable que un sistema operativo monolítico de cinco millones de líneas de código contenga cerca de 50,000 errores en el kernel. Desde luego que no todos estos son fatales, ya que algunos errores pueden ser cosas tales como emitir un mensaje de error incorrecto en una situación que ocurre raras veces. Sin embargo, los sistemas operativos tienen tantos errores que los fabricantes de computadoras colocan botones de reinicio en ellas (a menudo en el panel frontal), algo que los fabricantes de televisiones, estéreos y autos no hacen, a pesar de la gran cantidad de software en estos dispositivos.

La idea básica detrás del diseño de microkernel es lograr una alta confiabilidad al dividir el sistema operativo en módulos pequeños y bien definidos, sólo uno de los cuales (el microkernel) se ejecuta en modo kernel y el resto se ejecuta como procesos de usuario ordinarios, sin poder relativamente. En especial, al ejecutar cada driver de dispositivo y sistema de archivos como un proceso de usuario separado, un error en alguno de estos procesos puede hacer que falle ese componente, pero no puede hacer que falle todo el sistema. Así, un error en el driver del dispositivo de audio hará que el sonido sea confuso o se detenga, pero la computadora no fallará. En contraste, en un sistema monolítico con todos los drivers en el kernel, un driver de audio con errores puede hacer fácilmente referencia a una dirección de memoria inválida y llevar a todo el sistema a un alto rotundo en un instante.

Se han implementado y desplegado muchos microkernels (Accetta y colaboradores, 1986; Haertig y colaboradores, 1997; Heiser y colaboradores, 2006; Herder y colaboradores, 2006; Hildebrand, 1992; Kirsch y colaboradores, 2005; Liedtke, 1993, 1995, 1996; Pike y colaboradores, 1992; y Zuberi y colaboradores, 1999). Son en especial comunes en las aplicaciones en tiempo real, industriales, aeronáuticas y militares que son de misión crítica y tienen requerimientos de confiabilidad muy altos. Algunos de los microkernels mejor conocidos son Integrity, K42, L4, PikeOS, QNX, Symbian y MINIX 3. Ahora veremos en forma breve las generalidades acerca de MINIX 3, que ha llevado la idea de la modularidad hasta el límite, dividiendo la mayor parte del sistema operativo en varios procesos independientes en modo usuario. MINIX 3 es un sistema de código fuente abierto en conformidad con POSIX, disponible sin costo en *www.minix3.org* (Herder y colaboradores, 2006a; Herder y colaboradores, 2006b).

El microkernel MINIX 3 sólo tiene cerca de 3200 líneas de C y 800 líneas de ensamblador para las funciones de muy bajo nivel, como las que se usan para atrapar interrupciones y conmutar proceso. El código de C administra y planifica los procesos, se encarga de la comunicación entre procesos (al pasar mensajes entre procesos) y ofrece un conjunto de aproximadamente 35 llamadas al kernel para permitir que el resto del sistema operativo realice su trabajo. Estas llamadas realizan funciones tales como asociar los drivers a las interrupciones, desplazar datos entre espacios de direcciones e instalar nuevos mapas de memoria para los procesos recién creados. La estructura de procesos de MINIX 3 se muestra en la figura 1-26, en donde los manejadores de las llamadas al kernel se etiquetan como *Sys*. El manejador de dispositivo para el reloj también está

en el kernel, debido a que el planificador interactúa de cerca con él. Todos los demás dispositivos controladores se ejecutan como procesos de usuario separados.

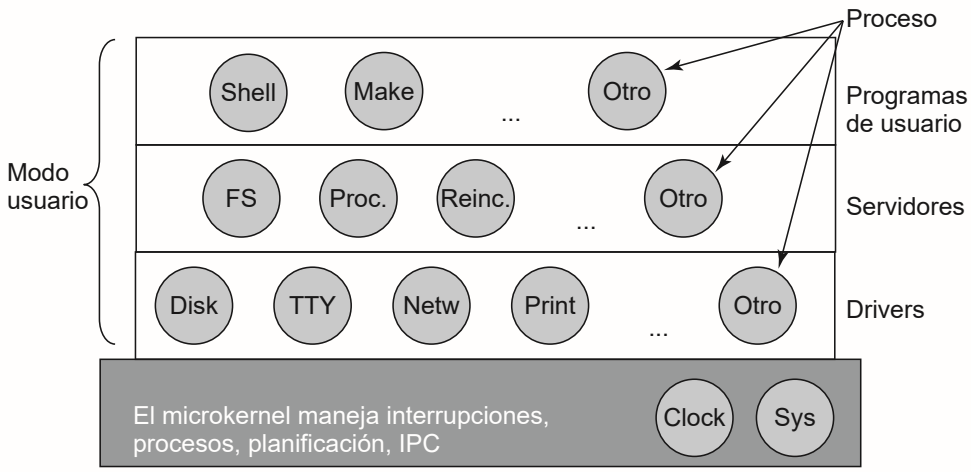

Figura 1-26. Estructura del sistema MINIX 3.

Fuera del kernel, el sistema se estructura como tres capas de procesos, todos se ejecutan en modo usuario. La capa más inferior contiene los drivers de dispositivos. Como todos se ejecutan en modo usuario, no tienen acceso físico al espacio de puertos de E/S y no pueden emitir comandos de E/S directamente. En vez de ello, para programar un dispositivo de E/S el driver crea una estructura para indicarle qué valores debe escribir en cuáles puertos de E/S y realiza una llamada al kernel para indicarle que realice la escritura. Esta metodología permite que el kernel compruebe que el driver esté escribiendo (o leyendo) de la E/S que está autorizado a utilizar. En consecuencia (y a diferencia de un diseño monolítico), un driver de audio defectuoso no puede escribir accidentalmente en el disco.

Encima de los drivers hay otra capa en modo usuario que contiene los servidores, que realizan la mayor parte del trabajo del sistema operativo. Uno o más servidores de archivos administran el (los) sistema(s) de archivos, el administrador de procesos crea, destruye y administra los procesos y así sucesivamente. Los programas de usuario obtienen servicios del sistema operativo mediante el envío de mensajes cortos a los servidores, pidiéndoles las llamadas al sistema POSIX. Por ejemplo, un proceso que necesite realizar una llamada `read` envía un mensaje a uno de los servidores de archivos para indicarle qué debe leer.

Un servidor interesante es el **servidor de reencarnación**, cuyo trabajo es comprobar si otros servidores y drivers están funcionando en forma correcta. En caso de que se detecte uno defectuoso, se reemplaza automáticamente sin intervención del usuario. De esta forma, el sistema es autocorregible y puede lograr una alta confiabilidad.

El sistema tiene muchas restricciones que limitan el poder de cada proceso. Como dijimos antes, los drivers sólo pueden utilizar los puertos de E/S autorizados, pero el acceso a las llamadas al kernel también está controlado dependiendo del proceso, al igual que la habilidad de enviar mensajes a otros procesos. Además, los procesos pueden otorgar un permiso limitado a otros procesos para hacer que el kernel acceda a sus espacios de direcciones. Como ejemplo, un sistema de archivos

puede otorgar permiso al dispositivo controlador de disco para dejar que el kernel coloque un bloque de disco recién leído en una dirección específica dentro del espacio de direcciones del sistema de archivos. El resultado de todas estas restricciones es que cada driver y servidor tiene el poder exacto para realizar su trabajo y no más, con lo cual se limita en forma considerable el daño que puede ocasionar un componente defectuoso.

Una idea que está en parte relacionada con tener un kernel mínimo es colocar el **mecanismo** para hacer algo en el kernel, pero no la **directiva**. Para aclarar mejor este punto, considere la planificación de los proceso. Un algoritmo de planificación relativamente simple sería asignar una prioridad a cada proceso y después hacer que el kernel ejecute el proceso de mayor prioridad que sea ejecutable. El mecanismo, en el kernel, es buscar el proceso de mayor prioridad y ejecutarlo. La directiva, asignar prioridades a los procesos, puede realizarse mediante los procesos en modo usuario. De esta forma, la directiva y el mecanismo se pueden desacoplar y el kernel puede reducir su tamaño.

1.7.4 Modelo cliente-servidor

Una ligera variación de la idea del microkernel es diferenciar dos clases de procesos: los **servidores**, cada uno de los cuales proporciona cierto servicio, y los **clientes**, que utilizan estos servicios. Este modelo se conoce como **cliente-servidor**. A menudo la capa inferior es un microkernel, pero eso no es requerido. La esencia es la presencia de procesos cliente y procesos servidor.

La comunicación entre clientes y servidores se lleva a cabo comúnmente mediante el paso de mensajes. Para obtener un servicio, un proceso cliente construye un mensaje indicando lo que desea y lo envía al servicio apropiado. Después el servicio hace el trabajo y envía de vuelta la respuesta. Si el cliente y el servidor se ejecutan en el mismo equipo se pueden hacer ciertas optimizaciones, pero en concepto estamos hablando sobre el paso de mensajes.

Una generalización obvia de esta idea es hacer que los clientes y los servidores se ejecuten en distintas computadoras, conectadas mediante una red de área local o amplia, como se describe en la figura 1-27. Como los clientes se comunican con los servidores mediante el envío de mensajes, no necesitan saber si los mensajes se manejan en forma local en sus propios equipos o si se envían a través de una red a servidores en un equipo remoto. En cuanto a lo que al cliente concierne, lo mismo ocurre en ambos casos: se envían las peticiones y se regresan las respuestas. Por ende, el modelo cliente-servidor es una abstracción que se puede utilizar para un solo equipo o para una red de equipos.

Cada vez hay más sistemas que involucran a los usuarios en sus PCs domésticas como clientes y equipos más grandes que operan en algún otro lado como servidores. De hecho, la mayor parte de la Web opera de esta forma. Una PC envía una petición de una página Web al servidor y la página Web se envía de vuelta. Éste es un uso común del modelo cliente-servidor en una red.

1.7.5 Máquinas virtuales

Las versiones iniciales del OS/360 eran, en sentido estricto, sistemas de procesamiento por lotes. Sin embargo, muchos usuarios del 360 querían la capacidad de trabajar de manera interactiva en una terminal, por lo que varios grupos, tanto dentro como fuera de IBM, decidieron escribir siste-

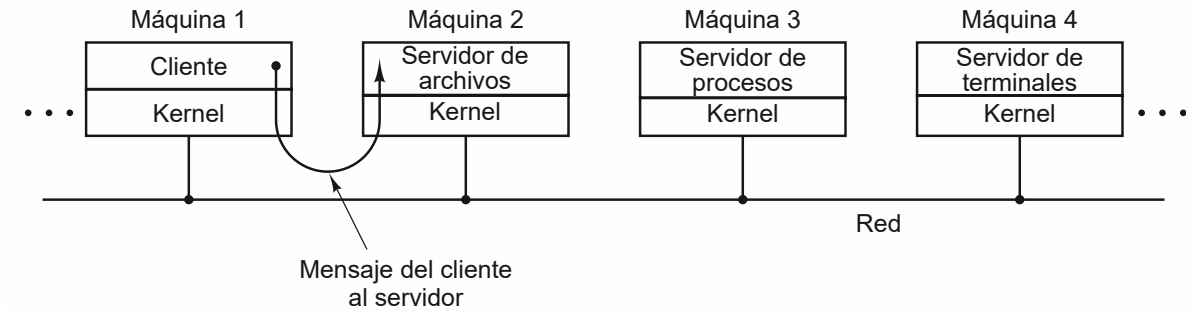

Figura 1-27. El modelo cliente-servidor sobre una red.

mas de tiempo compartido para este sistema. El sistema de tiempo compartido oficial de IBM, conocido como TSS/360, se liberó después de tiempo y cuando por fin llegó era tan grande y lento que pocos sitios cambiaron a este sistema. En cierto momento fue abandonado, una vez que su desarrollo había consumido cerca de 50 millones de dólares (Graham, 1970). Pero un grupo en el Scientific Center de IBM en Cambridge, Massachusetts, produjo un sistema radicalmente distinto que IBM aceptó eventualmente como producto. Un descendiente lineal de este sistema, conocido como **z/VM**, se utiliza ampliamente en la actualidad, en las mainframes de IBM (zSeries) que se utilizan mucho en centros de datos corporativos extensos, por ejemplo, como servidores de comercio electrónico que manejan cientos o miles de transacciones por segundo y utilizan bases de datos cuyos tamaños llegan a ser hasta de varios millones de gigabytes.

VM/370

Este sistema, que en un principio se llamó CP/CMS y posteriormente cambió su nombre a VM/370 (Seawright y MacKinnon, 1979), estaba basado en una astuta observación: un sistema de tiempo compartido proporciona (1) multiprogramación y (2) una máquina extendida con una interfaz más conveniente que el hardware por sí solo. La esencia de VM/370 es separar por completo estas dos funciones.

El corazón del sistema, que se conoce como **monitor de máquina virtual**, se ejecuta en el hardware solamente y realiza la multiprogramación, proporcionando no una, sino varias máquinas virtuales a la siguiente capa hacia arriba, como se muestra en la figura 1-28. Sin embargo, a diferencia de otros sistemas operativos, estas máquinas virtuales no son máquinas extendidas, con archivos y otras características adecuadas. En vez de ello, son copias *exactas* del hardware, incluyendo el modo kernel/ usuario, la E/S, las interrupciones y todo lo demás que tiene la máquina real.

Como cada máquina virtual es idéntica al verdadero hardware, cada una puede ejecutar cualquier sistema operativo que se ejecute directamente sólo en el hardware. Distintas máquinas virtuales pueden (y con frecuencia lo hacen) ejecutar distintos sistemas operativos. En el sistema VM/370 original, algunas ejecutaban OS/360 o uno de los otros sistemas operativos extensos de procesamiento por lotes o de procesamiento de transacciones, mientras que otros ejecutaban un sistema interactivo de un solo usuario llamado **CMS** (Conversational Monitor System; **Sistema monitor conversacional**) para los usuarios interactivos de tiempo compartido. Este último fue popular entre los programadores.

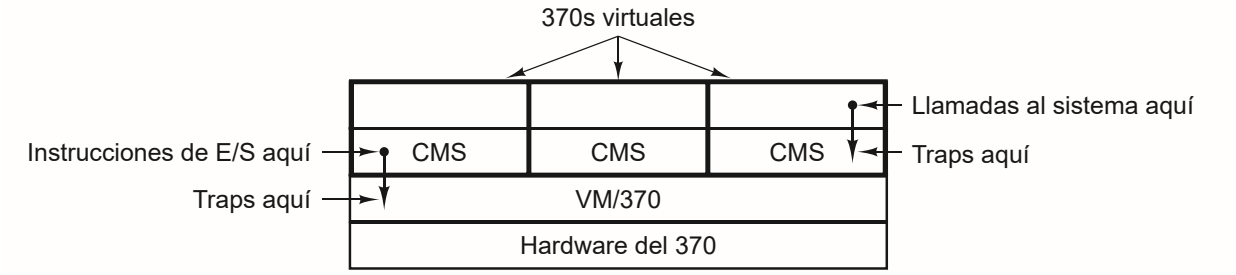

Figura 1-28. La estructura de VM/370 con CMS.

Cuando un programa de CMS ejecutaba una llamada al sistema, ésta quedaba atrapada para el sistema operativo en su propia máquina virtual, no para VM/370, de igual forma que si se ejecutara en una máquina real, en vez de una virtual. Después, CMS emitía las instrucciones normales de E/S de hardware para leer su disco virtual o lo que fuera necesario para llevar a cabo la llamada. Estas instrucciones de E/S eran atrapadas por la VM/370, que a su vez las ejecutaba como parte de su simulación del hardware real. Al separar por completo las funciones de multiprogramación y proporcionar una máquina extendida, cada una de las piezas podían ser más simples, más flexibles y mucho más fáciles de mantener.

En su encarnación moderna, z/VM se utiliza por lo general para ejecutar varios sistemas operativos completos, en vez de sistemas simplificados de un solo usuario como CMS. Por ejemplo, la serie zSeries es capaz de ejecutar una o más máquinas virtuales de Linux junto con los sistemas operativos tradicionales de IBM.

Redescubrimiento de las máquinas virtuales

Mientras que IBM ha tenido un producto de máquina virtual disponible durante cuatro décadas, y unas cuantas compañías más como Sun Microsystems y Hewlett-Packard han agregado recientemente el soporte de máquinas virtuales a sus servidores empresariales de alto rendimiento, la idea de la virtualización se había ignorado por mucho tiempo en el mundo de la PC, hasta hace poco. Pero en los últimos años, se han combinado nuevas necesidades, nuevo software y nuevas tecnologías para convertirla en un tema de moda.

Primero hablaremos sobre las necesidades. Muchas compañías han ejecutado tradicionalmente sus servidores de correo, servidores Web, servidores FTP y otros servidores en computadoras separadas, algunas veces con distintos sistemas operativos. Consideran la virtualización como una forma de ejecutarlos todos en la misma máquina, sin que una falla de un servidor haga que falle el resto.

La virtualización también es popular en el mundo del hospedaje Web. Sin ella, los clientes de hospedaje Web se ven obligados a elegir entre el **hospedaje compartido** (que les ofrece sólo una cuenta de inicio de sesión en un servidor Web, pero ningún control sobre el software de servidor) y hospedaje dedicado (que les ofrece su propia máquina, lo cual es muy flexible pero no es costeable para los sitios Web de pequeños a medianos). Cuando una compañía de hospedaje Web ofrece la renta de máquinas virtuales, una sola máquina física puede ejecutar muchas máquinas virtuales,

cada una de las cuales parece ser una máquina completa. Los clientes que rentan una máquina virtual pueden ejecutar cualesquier sistema operativo y software que deseen, pero a una fracción del costo de un servidor dedicado (debido a que la misma máquina física soporta muchas máquinas virtuales al mismo tiempo).

Otro uso de la virtualización es para los usuarios finales que desean poder ejecutar dos o más sistemas operativos al mismo tiempo, por decir Windows y Linux, debido a que algunos de sus paquetes de aplicaciones favoritos se ejecutan en el primero y algunos otros en el segundo. Esta situación se ilustra en la figura 1-29(a), en donde el término "monitor de máquinas virtuales" ha cambiado su nombre por el de **hipervisor** de tipo 1 en años recientes.

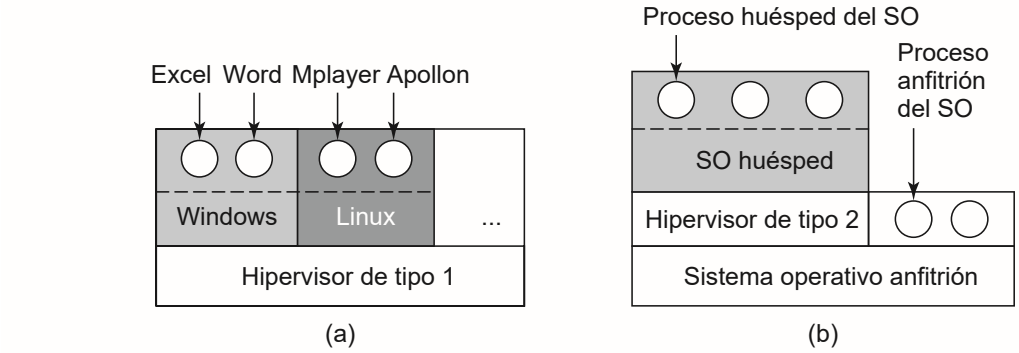

Figura 1-29. (a) Un hipervisor de tipo 1. (b) Un hipervisor de tipo 2.

Ahora pasemos al software. Aunque nadie disputa lo atractivo de las máquinas virtuales, el problema está en su implementación. Para poder ejecutar software de máquina virtual en una computadora, su CPU debe ser virtualizable (Popek y Goldberg, 1974). En síntesis, he aquí el problema. Cuando un sistema operativo que opera en una máquina virtual (en modo usuario) ejecuta una instrucción privilegiada, tal como para modificar el PSW o realizar una operación de E/S, es esencial que el hardware la atrape para el monitor de la máquina virtual, de manera que la instrucción se pueda emular en el software. En algunas CPUs (como el Pentium, sus predecesores y sus clones) los intentos de ejecutar instrucciones privilegiadas en modo de usuario simplemente se ignoran. Esta propiedad hacía que fuera imposible tener máquinas virtuales en este hardware, lo cual explica la falta de interés en el mundo de la PC. Desde luego que había intérpretes para el Pentium que se ejecutaban en el Pentium, pero con una pérdida de rendimiento de por lo general 5x a 10x, no eran útiles para un trabajo serio.

Esta situación cambió como resultado de varios proyectos de investigación académicos en la década de 1990, como Disco en Stanford (Bugnion y colaboradores, 1997), que ocasionaron el surgimiento de productos comerciales (por ejemplo, VMware Workstation) y que reviviera el interés en las máquinas virtuales. VMware Workstation es un hipervisor de tipo 2, el cual se muestra en la figura 1-29(b). En contraste con los hipervisores de tipo 1, que se ejecutaban en el hardware directo, los hipervisores de tipo 2 se ejecutan como programas de aplicación encima de Windows, Linux o algún otro sistema operativo, conocido como **sistema operativo anfitrión**. Una vez que se inicia un hipervisor de tipo 2, lee el CD-ROM de instalación para el **sistema operativo huésped** elegido

y lo instala en un disco virtual, que es tan sólo un gran archivo en el sistema de archivos del sistema operativo anfitrión.

Cuando se arrancar el sistema operativo huésped, realiza lo mismo que en el hardware real; por lo general inicia algunos procesos en segundo plano y después una GUI. Algunos hipervisores traducen los programas binarios del sistema operativo huésped bloque por bloque, reemplazando ciertas instrucciones de control con llamadas al hipervisor. Después, los bloques traducidos se ejecutan y se colocan en caché para su uso posterior.

Un enfoque distinto en cuanto al manejo de las instrucciones de control es el de modificar el sistema operativo para eliminarlas. Este enfoque no es una verdadera virtualización, sino **paravirtualización**. En el capítulo 8 hablaremos sobre la virtualización con más detalle.

La máquina virtual de Java

Otra área en donde se utilizan máquinas virtuales, pero de una manera algo distinta, es para ejecutar programas de Java. Cuando Sun Microsystems inventó el lenguaje de programación Java, también inventó una máquina virtual (es decir, una arquitectura de computadora) llamada **JVM** (*Java Virtual Machine*, Máquina virtual de Java). El compilador de Java produce código para la JVM, que a su vez típicamente se ejecuta mediante un software intérprete de JVM. La ventaja de este método es que el código de la JVM se puede enviar a través de Internet, a cualquier computadora que tenga un intérprete de JVM y se ejecuta allí. Por ejemplo, si el compilador hubiera producido programas binarios para SPARC o Pentium, no se podrían haber enviado y ejecutado en cualquier parte con la misma facilidad. (Desde luego que Sun podría haber producido un compilador que produjera binarios de SPARC y después distribuir un intérprete de SPARC, pero JVM es una arquitectura mucho más simple de interpretar.) Otra ventaja del uso de la JVM es que, si el intérprete se implementa de manera apropiada, que no es algo completamente trivial, se puede comprobar que los programas de JVM entrantes sean seguros para después ejecutarlos en un entorno protegido, de manera que no puedan robar datos ni realizar algún otro daño.

1.7.6 Exokernels

En vez de clonar la máquina actual, como se hace con las máquinas virtuales, otra estrategia es particionarla; en otras palabras, a cada usuario se le proporciona un subconjunto de los recursos. Así, una máquina virtual podría obtener los bloques de disco del 0 al 1023, la siguiente podría obtener los bloques de disco del 1024 al 2047 y así sucesivamente.

En la capa inferior, que se ejecuta en el modo kernel, hay un programa llamado **exokernel** (Engler y colaboradores, 1995). Su trabajo es asignar recursos a las máquinas virtuales y después comprobar los intentos de utilizarlos, para asegurar que ninguna máquina trate de usar los recursos de otra. Cada máquina virtual de nivel de usuario puede ejecutar su propio sistema operativo, al igual que en la VM/370 y las Pentium 8086 virtuales, con la excepción de que cada una está restringida a utilizar sólo los recursos que ha pedido y que le han sido asignados.

La ventaja del esquema del exokernel es que ahorra una capa de asignación. En los otros diseños, cada máquina virtual piensa que tiene su propio disco, con bloques que van desde 0 hasta cier-

to valor máximo, por lo que el monitor de la máquina virtual debe mantener tablas para reasignar las direcciones del disco (y todos los demás recursos). Con el exokernel, esta reasignación no es necesaria. El exokernel sólo necesita llevar el registro para saber a cuál máquina virtual se le ha asignado cierto recurso. Este método sigue teniendo la ventaja de separar la multiprogramación (en el exokernel) del código del sistema operativo del usuario (en espacio de usuario), pero con menos sobrecarga, ya que todo lo que tiene que hacer el exokernel es mantener las máquinas virtuales separadas unas de las otras.

1.8 EL MUNDO SEGÚN C

Por lo general los sistemas operativos son extensos programas en C (o algunas veces C++) que consisten de muchas piezas escritas por muchos programadores. El entorno que se utiliza para desarrollar sistemas operativos es muy distinto a lo que están acostumbrados los individuos (como los estudiantes) al escribir pequeños programas en Java. Esta sección es un intento de proporcionar una muy breve introducción al mundo de la escritura de sistemas operativos para los programadores inexpertos de Java.

1.8.1 El lenguaje C

Ésta no es una guía para el uso de C, sino un breve resumen de algunas de las diferencias clave entre C y Java. Java está basado en C, por lo que existen muchas similitudes entre los dos. Ambos son lenguajes imperativos con tipos de datos, variables e instrucciones de control, por ejemplo. Los tipos de datos primitivos en C son enteros (incluyendo cortos y largos), caracteres y números de punto flotante. Los tipos de datos compuestos se pueden construir mediante el uso de arreglos, estructuras y uniones. Las instrucciones de control en C son similares a las de Java, incluyendo las instrucciones if, switch, for y while. Las funciones y los parámetros son aproximadamente iguales en ambos lenguajes.

Una característica de C que Java no tiene son los apuntadores explícitos. Un **apuntador** es una variable que apunta a (es decir, contiene la dirección de) una variable o estructura de datos. Considere las siguientes instrucciones:

```
char c1, c2, *p;
c1 = 'x';
p = &c1;
c2 = *p;
```

que declaran a *c1* y *c2* como variables de tipo carácter y a *p* como una variable que apunta a (es decir, contiene la dirección de) un carácter. La primera asignación almacena el código ASCII para el carácter 'c' en la variable *c1*. La segunda asigna la dirección de *c1* a la variable apuntador *p*. La tercera asigna el contenido de la variable a la que apunta *p*, a la variable *c2*, por lo que después de ejecutar estas instrucciones, *c2* también contiene el código ASCII para 'c'. En teoría, los apuntadores están tipificados, por lo que no se debe asignar la dirección de un número de punto flotante a un

apuntador tipo carácter, pero en la práctica los compiladores aceptan dichas asignaciones, aunque algunas veces con una advertencia. Los apuntadores son una construcción muy potente, pero también una gran fuente de errores cuando se utilizan sin precaución.

Algunas cosas que C no tiene son: cadenas integradas, hilos, paquetes, clases, objetos, seguridad de tipos y recolección de basura. La última es un gran obstáculo para los sistemas operativos. Todo el almacenamiento en C es estático o el programador lo asigna y libera de manera explícita, por lo general con las funciones de biblioteca *malloc* y *free*. Esta última propiedad (control total del programador sobre la memoria) junto con los apuntadores explícitos que hacen de C un lenguaje atractivo para escribir sistemas operativos. En esencia, los sistemas operativos son sistemas en tiempo real hasta cierto punto, incluso los de propósito general. Cuando ocurre una interrupción, el sistema operativo sólo puede tener unos cuantos microsegundos para realizar cierta acción o de lo contrario, puede perder información crítica. Es intolerable que el recolector de basura entre en acción en un momento arbitrario.

1.8.2 Archivos de encabezado

Por lo general, un proyecto de sistema operativo consiste en cierto número de directorios, cada uno de los cuales contiene muchos archivos *.c* que contienen el código para cierta parte del sistema, junto con varios archivos de encabezado *.h* que contienen declaraciones y definiciones utilizadas por uno o más archivos de código. Los archivos de encabezado también pueden incluir **macros** simples, tales como:

```
#define TAM_BUFER 4096
```

las cuales permiten que el programador nombre constantes, de manera que cuando *TAM_BUFER* se utilice en el código, se reemplace durante la compilación por el número 4096. Una buena práctica de programación de C es nombrar cada constante excepto 0, 1 y −1, y algunas veces también éstas se nombran. Las macros pueden tener parámetros, tales como:

```
#define max(a, b) (a > b ? a : b)
```

con lo cual el programador puede escribir:

```
i = max(j, k+1)
```

y obtener:

```
i = (j > k+1 ? j : k + 1)
```

para almacenar el mayor de *j* y *k + 1* en *i*. Los encabezados también pueden contener compilación condicional, por ejemplo:

```
#ifdef PENTIUM
intel_int_ack();
#endif
```

lo cual se compila en una llamada a la función *intel_int_ack* si la macro *PENTIUM* está definida y no hace nada en caso contrario. La compilación condicional se utiliza con mucha frecuencia para

aislar el código dependiente de la arquitectura, de manera que cierto código se inserte sólo cuando se compile el sistema en el Pentium, que se inserte otro código sólo cuando el sistema se compile en una SPARC y así sucesivamente. Un archivo *.c* puede incluir físicamente cero o más archivos de encabezado mediante la directiva *#include*. También hay muchos archivos de encabezado comunes para casi cualquier archivo *.c*, y se almacenan en un directorio central.

1.8.3 Proyectos de programación extensos

Para generar el sistema operativo, el compilador de C compila cada archivo *.c* en un **archivo de código objeto**. Estos archivos, que tienen el sufijo *.o*, contienen instrucciones binarias para el equipo de destino. Posteriormente la CPU los ejecutará directamente. No hay nada como el código byte de Java en el mundo de C.

La primera pasada del compilador de C se conoce como **preprocesador de C**. A medida que lee cada archivo *.c*, cada vez que llega a una directiva *#include* va y obtiene el archivo de encabezado cuyo nombre contiene y lo procesa, expandiendo las macros, manejando la compilación condicional (y varias otras cosas) y pasando los resultados a la siguiente pasada del compilador, como si se incluyeran físicamente.

Como los sistemas operativos son muy extensos (es común que tengan cerca de cinco millones de líneas de código), sería imposible tener que recompilar todo cada vez que se modifica un archivo. Por otro lado, para cambiar un archivo de encabezado clave que se incluye en miles de otros archivos, hay que volver a compilar esos archivos. Llevar la cuenta de qué archivos de código objeto depende de cuáles archivos de encabezado es algo que no se puede manejar sin ayuda.

Por fortuna, las computadoras son muy buenas precisamente en este tipo de cosas. En los sistemas UNIX hay un programa llamado *make* (con numerosas variantes tales como *gmake*, *pmake*, etcétera) que lee el *Makefile*, un archivo que indica qué archivos son dependientes de cuáles otros archivos. Lo que hace *make* es ver qué archivos de código objeto se necesitan para generar el archivo binario del sistema operativo que se necesita en un momento dado y para cada uno comprueba si alguno de los archivos de los que depende (el código y los encabezados) se ha modificado después de la última vez que se creó el archivo de código objeto. De ser así, ese archivo de código objeto se tiene que volver a compilar. Cuando *make* ha determinado qué archivos *.c* se tienen que volver a compilar, invoca al compilador de C para que los recompile, con lo cual se reduce el número de compilaciones al mínimo. En los proyectos extensos, la creación del *Makefile* está propensa a errores, por lo que hay herramientas que lo hacen de manera automática.

Una vez que están listos todos los archivos *.o*, se pasan a un programa conocido como **enlazador** para combinarlos en un solo archivo binario ejecutable. Cualquier función de biblioteca que sea llamada también se incluye en este punto, se resuelven las referencias dentro de las funciones y se reasignan las direcciones de la máquina según sea necesario. Cuando el enlazador termina, el resultado es un programa ejecutable, que por tradición se llama *a.out* en los sistemas UNIX. Los diversos componentes de este proceso se ilustran en la figura 1-30 para un programa con tres archivos de C y dos archivos de encabezado. Aunque hemos hablado aquí sobre el desarrollo de sistemas operativos, todo esto se aplica al desarrollo de cualquier programa extenso.

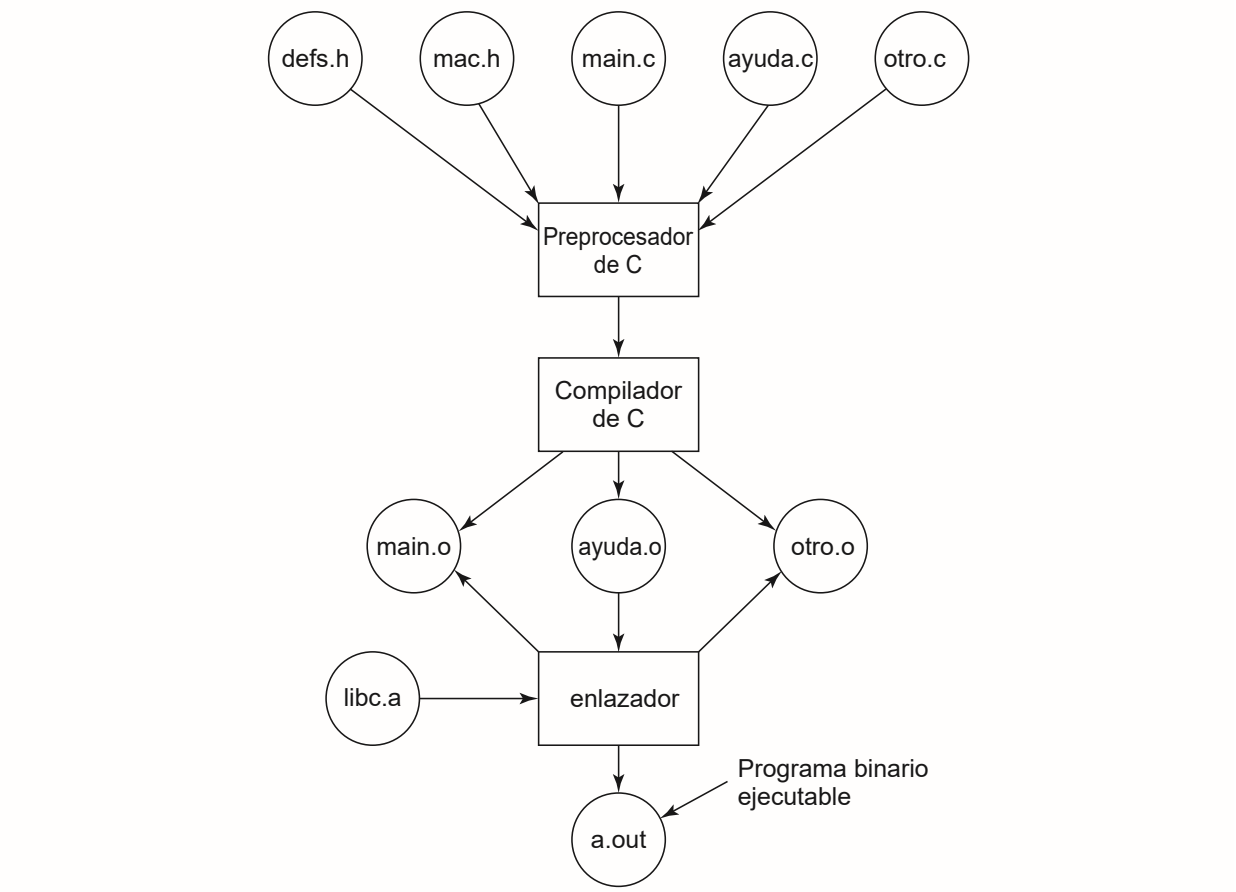

Figura 1-30. El proceso de compilar archivos de C y de encabezado para crear un archivo ejecutable.

1.8.4 El modelo del tiempo de ejecución

Una vez que se ha enlazado el archivo binario del sistema operativo, la computadora puede reiniciarse con el nuevo sistema operativo. Una vez en ejecución, puede cargar piezas en forma dinámica que no se hayan incluido de manera estática en el binario, como los drivers de dispositivos y los sistemas de archivos. En tiempo de ejecución, el sistema operativo puede consistir de varios segmentos, para el texto (el código del programa), los datos y la pila. Por lo general, el segmento de texto es inmutable y no cambia durante la ejecución. El segmento de datos empieza con cierto tamaño y se inicializa con ciertos valores, pero puede cambiar y crecer según lo necesite. Al principio la pila está vacía, pero crece y se reduce a medida que se hacen llamadas a funciones y se regresa de ellas. A menudo el segmento de texto se coloca cerca de la parte final de la memoria, el segmento de datos justo encima de éste, con la habilidad de crecer hacia arriba y el segmento de pila en una dirección virtual superior, con la habilidad de crecer hacia abajo, pero los distintos sistemas trabajan de manera diferente.

En todos los casos, el código del sistema operativo es ejecutado directamente por el hardware, sin intérprete y sin compilación justo-a-tiempo, como se da el caso con Java.

1.9 INVESTIGACIÓN ACERCA DE LOS SISTEMAS OPERATIVOS

La ciencia computacional es un campo que avanza con rapidez y es difícil predecir hacia dónde va. Los investigadores en universidades y los laboratorios de investigación industrial están desarrollando constantemente nuevas ideas, algunas de las cuales no van a ningún lado, pero otras se convierten en la piedra angular de futuros productos y tienen un impacto masivo en la industria y en los usuarios. Saber qué ideas tendrán éxito es más fácil en retrospección que en tiempo real. Separar el trigo de la paja es en especial difícil, debido a que a menudo se requieren de 20 a 30 años para que una idea haga impacto.

Por ejemplo, cuando el presidente Eisenhower estableció la Agencia de Proyectos Avanzados de Investigación (ARPA) del Departamento de Defensa en 1958, trataba de evitar que el Ejército predominara sobre la Marina y la Fuerza Aérea en relación con el presupuesto de investigación. No estaba tratando de inventar Internet. Pero una de las cosas que hizo ARPA fue patrocinar cierta investigación universitaria sobre el entonces oscuro tema de la conmutación de paquetes, lo cual condujo a la primera red experimental de conmutación de paquetes, ARPANET. Entró en funcionamiento en 1969. Poco tiempo después, otras redes de investigación patrocinadas por ARPA estaban conectadas a ARPANET,y nació Internet, que en ese entonces era utilizada felizmente por los investigadores académicos para enviarse correo electrónico unos con otros durante 20 años. A principios de la década de 1990, Tim Berners-Lee inventó la World Wide Web en el laboratorio de investigación CERN en Ginebra y Marc Andreesen escribió el código de un navegador gráfico para esta red en la Universidad de Illinois. De repente, Internet estaba llena de adolescentes conversando entre sí. Probablemente el presidente Eisenhower esté revolcándose en su tumba.

La investigación en los sistemas operativos también ha producido cambios dramáticos en los sistemas prácticos. Como vimos antes, los primeros sistemas computacionales comerciales eran todos sistemas de procesamiento por lotes, hasta que el M.I.T. inventó el tiempo compartido interactivo a principios de la década de 1960. Todas las computadoras eran basadas en texto, hasta que Doug Engelbart inventó el ratón y la interfaz gráfica de usuario en el Stanford Research Institute a finales de la década de 1960. ¿Quién sabe lo que vendrá a continuación?

En esta sección y en otras secciones equivalentes en el resto de este libro, daremos un breve vistazo a una parte de la investigación sobre los sistemas operativos que se ha llevado a cabo durante los últimos 5 a 10 años, sólo para tener una idea de lo que podría haber en el horizonte. En definitiva, esta introducción no es exhaustiva y se basa en gran parte en los artículos que se han publicado en los principales diarios de investigación y conferencias, debido a que estas ideas han sobrevivido por lo menos a un proceso de revisión personal riguroso para poder publicarse. La mayoría de los artículos citados en las secciones de investigación fueron publicados por la ACM, la IEEE Computer Society o USENIX, que están disponibles por Internet para los miembros (estudiantes) de estas organizaciones. Para obtener más información acerca de estas organizaciones y sus bibliotecas digitales, consulte los siguientes sitios Web:

ACM	http://www.acm.org
IEEE Computer Society	http://www.computer.org
USENIX	http://www.usenix.org

Casi todos los investigadores de sistemas operativos consideran que los sistemas operativos actuales son masivos, inflexibles, no muy confiables, inseguros y están cargados de errores, evidentemente algunos más que otros (*no divulgamos los nombres para no dañar a los aludidos*). En consecuencia, hay mucha investigación acerca de cómo construir mejores sistemas operativos. En fechas recientes se ha publicado trabajo acerca de los nuevos sistemas operativos (Krieger y colaboradores, 2006), la estructura del sistema operativo (Fassino y colaboradores, 2002), la rectitud del sistema operativo (Elphinstone y colaboradores, 2007; Kumar y Li, 2002; y Yang y colaboradores, 2006), la confiabilidad del sistema operativo (Swift y colaboradores, 2006; y LeVasseur y colaboradores, 2004), las máquinas virtuales (Barham y colaboradores, 2003; Garfinkel y colaboradores, 2003; King y colaboradores, 2003; y Whitaker y colaboradores, 2002), los virus y gusanos (Costa y colaboradores, 2005; Portokalidis y colaboradores, 2006; Tucek y colaboradores, 2007; y Vrable y colaboradores, 2005), los errores y la depuración (Chou y colaboradores, 2001; y King y colaboradores, 2005), el hiperhilamiento y el multihilamiento (Fedorova, 2005; y Bulpin y Pratt, 2005), y el comportamiento de los usuarios (Yu y colaboradores, 2006), entre muchos otros temas.

1.10 DESCRIPCIÓN GENERAL SOBRE EL RESTO DE ESTE LIBRO

Ahora hemos completado nuestra introducción y un breve visión del sistema operativo. Es tiempo de entrar en detalles. Como dijimos antes, desde el punto de vista del programador, el propósito principal de un sistema operativo es proveer ciertas abstracciones clave, siendo las más importantes los procesos y hilos, los espacios de direcciones y los archivos. De manera acorde, los siguientes tres capítulos están dedicados a estos temas cruciales.

El capítulo 2 trata acerca de los procesos y hilos. Describe sus propiedades y la manera en que se comunican entre sí. También proporciona varios ejemplos detallados acerca de la forma en que funciona la comunicación entre procesos y cómo se pueden evitar algunos obstáculos.

En el capítulo 3 estudiaremos los espacios de direcciones y su administración adjunta de memoria con detalle. Examinaremos el importante tema de la memoria virtual, junto con los conceptos estrechamente relacionados, como la paginación y la segmentación.

Después, en el capítulo 4 llegaremos al importantísimo tema de los sistemas de archivos. Hasta cierto punto, lo que el usuario ve es en gran parte el sistema de archivos. Analizaremos la interfaz del sistema de archivos y su implementación.

En el capítulo 5 se cubre el tema de las operaciones de entrada/salida. Hablaremos sobre los conceptos de independencia de dispositivos y dependencia de dispositivos. Utilizaremos como ejemplos varios dispositivos importantes, como los discos, teclados y pantallas.

El capítulo 6 trata acerca de los interbloqueos. Anteriormente en este capítulo mostramos en forma breve qué son los interbloqueos, pero hay mucho más por decir. Hablaremos también sobre las formas de prevenirlos o evitarlos.

En este punto habremos completado nuestro estudio acerca de los principios básicos de los sistemas operativos con una sola CPU. Sin embargo, hay más por decir, en especial acerca de los temas avanzados. En el capítulo 7 examinaremos los sistemas de multimedia, que tienen varias

propiedades y requerimientos distintos de los sistemas operativos convencionales. Entre otros elementos, la planificación de tareas y el sistema de archivos se ven afectados por la naturaleza de la multimedia. Otro tema avanzado es el de los sistemas con múltiples procesadores, incluyendo los multiprocesadores, las computadoras en paralelo y los sistemas distribuidos. Cubriremos estos temas en el capítulo 8.

Un tema de enorme importancia es la seguridad de los sistemas operativos, que se cubre en el capítulo 9. Entre los temas descritos en este capítulo están las amenazas (es decir, los virus y gusanos), los mecanismos de protección y los modelos de seguridad.

A continuación veremos algunos ejemplos prácticos de sistemas operativos reales. Éstos son Linux (capítulo 10), Windows Vista (capítulo 11) y Symbian (capítulo 12). El libro concluye con ciertos consejos y pensamientos acerca del diseño de sistemas operativos en el capítulo 13.

1.11 UNIDADES MÉTRICAS

Para evitar cualquier confusión, vale la pena declarar en forma explícita que en este libro, como en la ciencia computacional en general, se utilizan unidades métricas en vez de las tradicionales unidades del sistema inglés. Los principales prefijos métricos se listan en la figura 1-31. Por lo general, estos prefijos se abrevian con sus primeras letras y las unidades mayores de 1 están en mayúsculas. Así, una base de datos de 1 TB ocupa 10^{12} bytes de almacenamiento, y un reloj de 100 pseg (o 100 ps) emite un pulso cada 10^{-10} segundos. Como mili y micro empiezan con la letra "m", se tuvo que hacer una elección. Por lo general, "m" es para mili y "μ" (la letra griega mu) es para micro.

Exp.	Explícito	Prefijo	Exp.	Explícito	Prefijo
10^{-3}	0.001	mili	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

Figura 1-31. Los principales prefijos métricos.

También vale la pena recalcar que para medir los tamaños de las memorias, en la práctica común de la industria, las unidades tienen significados ligeramente diferentes. Ahí, Kilo significa 2^{10} (1024) en vez de 10^{3} (1000), debido a que las memorias siempre utilizan una potencia de dos. Por ende, una memoria de 1 KB contiene 1024 bytes, no 1000 bytes. De manera similar, una memoria de 1 MB contiene 2^{20} (1,048,576) bytes y una memoria de 1 GB contiene 2^{30} (1,073,741,824) bytes. Sin embargo, una línea de comunicación de 1 Kbps transmite 1000 bits por segundo y una LAN de 10 Mbps opera a 10,000,000 bits/seg, debido a que estas velocidades no son potencias de dos. Por desgracia, muchas personas tienden a mezclar estos dos sistemas, en especial para los tamaños

de los discos. Para evitar ambigüedades, en este libro utilizaremos los símbolos KB, MB y GB para 2^{10}, 2^{20} y 2^{30} bytes, respectivamente y los símbolos Kbps, Mbps y Gbps para 10^3, 10^6 y 10^9 bits/seg, respectivamente.

1.12 RESUMEN

Los sistemas operativos se pueden ver desde dos puntos de vista: como administradores de recursos y como máquinas extendidas. En el punto de vista correspondiente al administrador de recursos, la función del sistema operativo es administrar las distintas partes del sistema en forma eficiente. En el punto de vista correspondiente a la máquina extendida, la función del sistema operativo es proveer a los usuarios abstracciones que sean más convenientes de usar que la máquina actual. Estas abstracciones incluyen los procesos, espacios de direcciones y archivos.

Los sistemas operativos tienen una larga historia, empezando desde los días en que reemplazaron al operador, hasta los sistemas modernos de multiprogramación. Entre los puntos importantes tenemos a los primeros sistemas de procesamiento por lotes, los sistemas de multiprogramación y los sistemas de computadora personal.

Como los sistemas operativos interactúan de cerca con el hardware, es útil tener cierto conocimiento del hardware de computadora parea comprenderlos. Las computadoras están compuestas de procesadores, memorias y dispositivos de E/S. Estas partes se conectan mediante buses.

Los conceptos básicos en los que se basan todos los sistemas operativos son los procesos, la administración de memoria, la administración de E/S, el sistema de archivos y la seguridad. Cada uno de estos conceptos se tratará con detalle en un capítulo posterior.

El corazón de cualquier sistema operativo es el conjunto de llamadas al sistema que puede manejar. Estas llamadas indican lo que realmente hace el sistema operativo. Para UNIX, hemos visto cuatro grupos de llamadas al sistema. El primer grupo de llamadas se relaciona con la creación y terminación de procesos. El segundo grupo es para leer y escribir en archivos. El tercer grupo es para administrar directorios. El cuarto grupo contiene una miscelánea de llamadas.

Los sistemas operativos se pueden estructurar en varias formas. Las más comunes son como un sistema monolítico, una jerarquía de capas, microkernel, cliente-servidor, máquina virtual o exokernel.

PROBLEMAS

1. ¿Qué es la multiprogramación?
2. ¿Qué es spooling? ¿Cree usted que las computadoras personales avanzadas tendrán spooling como característica estándar en el futuro?
3. En las primeras computadoras, cada byte de datos leídos o escritos se manejaba mediante la CPU (es decir, no había DMA). ¿Qué implicaciones tiene esto para la multiprogramación?

4. La idea de una familia de computadoras fue introducida en la década de 1960 con las mainframes IBM System/360. ¿Está muerta ahora esta idea o sigue en pie?

5. Una razón por la cual las GUI no se adoptaron con rapidez en un principio fue el costo del hardware necesario para darles soporte. ¿Cuánta RAM de video se necesita para dar soporte a una pantalla de texto monocromático de 25 líneas x 80 caracteres? ¿Cuánta se necesita para un mapa de bits de 1024 × 768 píxeles y colores 24 bits? ¿Cuál fue el costo de esta RAM con precios de 1980 (5 dólares/KB)? ¿Cuánto vale ahora?

6. Hay varias metas de diseño a la hora de crear un sistema operativo, por ejemplo: la utilización de recursos, puntualidad, que sea robusto, etcétera. De un ejemplo de dos metas de diseño que puedan contradecirse entre sí.

7. ¿Cuál de las siguientes instrucciones debe permitirse sólo en modo kernel?

a) Deshabilitar todas las interrupciones.
b) Leer el reloj de la hora del día.
c) Establecer el reloj de la hora del día.
d) Cambiar el mapa de memoria.

8. Considere un sistema con dos CPUs y que cada CPU tiene dos hilos (hiperhilamiento). Suponga que se inician tres programas *P0*, *P1* y *P2* con tiempos de ejecución de 5, 10 y 20 mseg, respectivamente. ¿Cuánto se tardará en completar la ejecución de estos programas? Suponga que los tres programas están 100% ligados a la CPU, que no se bloquean durante la ejecución y no cambian de CPU una vez que se les asigna.

9. Una computadora tiene una canalización con cuatro etapas. Cada etapa requiere el mismo tiempo para hacer su trabajo, a saber, 1 nseg. ¿Cuántas instrucciones por segundo puede ejecutar esta máquina?

10. Considere un sistema de cómputo con memoria caché, memoria principal (RAM) y disco, y que el sistema operativo utiliza memoria virtual. Se requieren 2 nseg para acceder a una palabra desde la caché, 10 nseg para acceder a una palabra desde la RAM y 10 ms para acceder a una palabra desde el disco. Si la proporción de aciertos de caché es de 95% y la proporción de aciertos de memoria (después de un fallo de caché) es de 99%, ¿cuál es el tiempo promedio para acceder a una palabra?

11. Un revisor alerta observa un error de ortografía consistente en el manuscrito del libro de texto de sistemas operativos que está a punto de ser impreso. El libro tiene cerca de 700 páginas, cada una con 50 líneas de 80 caracteres. ¿Cuánto tiempo se requerirá para digitalizar en forma electrónica el texto, para el caso en que la copia maestra se encuentre en cada uno de los niveles de memoria de la figura 1-9? Para los métodos de almacenamiento interno, considere que el tiempo de acceso dado es por carácter, para los discos suponga que el tiempo es por bloque de 1024 caracteres y para la cinta suponga que el tiempo dado es para el inicio de los datos, con un acceso posterior a la misma velocidad que el acceso al disco.

12. Cuando un programa de usuario realiza una llamada al sistema para leer o escribir en un archivo en disco, proporciona una indicación de qué archivo desea, un apuntador al búfer de datos y la cuenta. Después, el control se transfiere al sistema operativo, el cual llama al driver apropiado. Suponga que el driver inicia el disco y termina hasta que ocurre una interrupción. En el caso de leer del disco, es obvio que el procedimiento que hizo la llamada tiene que ser bloqueado (debido a que no hay datos para leer). ¿Qué hay sobre el caso de escribir en el disco? ¿Necesita ser bloqueado el procedimiento llamador, para esperar a que se complete la transferencia del disco?

13. ¿Qué es una instrucción de trap? Explique su uso en los sistemas operativos.

14. ¿Cuál es la diferencia clave entre un trap y una interrupción?

15. ¿Por qué se necesita la tabla de procesos en un sistema de tiempo compartido? ¿Se necesita también en los sistemas de computadora personal en los que sólo existe un proceso, y ese proceso ocupa toda la máquina hasta que termina?

16. ¿Existe alguna razón por la que sería conveniente montar un sistema de archivos en un directorio no vacío? De ser así, ¿cuál es?

17. ¿Cuál es el propósito de una llamada al sistema en un sistema operativo?

18. Para cada una de las siguientes llamadas al sistema, proporcione una condición que haga que falle: `fork`, `exec` y `unlink`.

19. ¿Podría la llamada

```
cuenta = write(fd, bufer, nbytes);
```

devolver algún valor en *cuenta* distinto de *nbytes*? Si es así, ¿por qué?

20. Un archivo cuyo descriptor es *fd* contiene la siguiente secuencia de bytes: 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5. Se realizan las siguientes llamadas al sistema:

```
lseek(fd, 3, SEEK_SET);
read(fd, &bufer, 4);
```

en donde la llamada `lseek` realiza una búsqueda en el byte 3 del archivo. ¿Qué contiene *bufer* después de completar la operación de lectura?

21. Suponga que un archivo de 10 MB se almacena en un disco, en la misma pista (pista #: 50) en sectores consecutivos. El brazo del disco se encuentra actualmente situado en la pista número 100. ¿Cuánto tardará en recuperar este archivo del disco? Suponga que para desplazar el brazo de un cilindro al siguiente se requiere aproximadamente 1 ms y se requieren aproximadamente 5 ms para que el sector en el que está almacenado el inicio del archivo gire bajo la cabeza. Suponga además que la lectura ocurre a una velocidad de 100 MB/s.

22. ¿Cuál es la diferencia esencial entre un archivo especial de bloque y un archivo especial de carácter?

23. En el ejemplo que se da en la figura 1-17, el procedimiento de biblioteca se llama *read* y la misma llamada al sistema se llama `read`. ¿Es esencial que ambos tengan el mismo nombre? Si no es así, ¿cuál es más importante?

24. El modelo cliente-servidor es popular en los sistemas distribuidos. ¿Puede utilizarse también en un sistema de una sola computadora?

25. Para un programador, una llamada al sistema se ve igual que cualquier otra llamada a un procedimiento de biblioteca. ¿Es importante que un programador sepa cuáles procedimientos de biblioteca resultan en llamadas al sistema? ¿Bajo qué circunstancias y por qué?

26. La figura 1-23 muestra que varias llamadas al sistema de UNIX no tienen equivalentes en la API Win32. Para cada una de las llamadas que se listan y no tienen un equivalente en Win32, ¿cuáles son las consecuencias de que un programador convierta un programa de UNIX para que se ejecute en Windows?

27. Un sistema operativo portátil se puede portar de la arquitectura de un sistema a otro, sin ninguna modificación. Explique por qué no es factible construir un sistema operativo que sea completamente portátil. Describa dos capas de alto nivel que tendrá al diseñar un sistema operativo que sea altamente portátil.

28. Explique cómo la separación de la directiva y el mecanismo ayuda a construir sistemas operativos basados en microkernel.

29. He aquí algunas preguntas para practicar las conversiones de unidades:

(a) ¿A cuántos segundos equivale un microaño?
(b) A los micrómetros se les conoce comúnmente como micrones. ¿Qué tan largo es un gigamicron?
(c) ¿Cuántos bytes hay en una memoria de 1 TB?
(d) La masa de la Tierra es de 6000 yottagramos. ¿Cuánto es eso en kilogramos?

30. Escriba un shell que sea similar a la figura 1-19, pero que contenga suficiente código como para que pueda funcionar y lo pueda probar. También podría agregar algunas características como la redirección de la entrada y la salida, los canales y los trabajos en segundo plano.

31. Si tiene un sistema personal parecido a UNIX (Linux, MINIX, FreeBSD, etcétera) disponible que pueda hacer fallar con seguridad y reiniciar, escriba una secuencia de comandos de shell que intente crear un número ilimitado de procesos hijos y observe lo que ocurre. Antes de ejecutar el experimento, escriba `sync` en el shell para vaciar los búferes del sistema de archivos al disco y evitar arruinar el sistema de archivos. **Nota:** No intente esto en un sistema compartido sin obtener primero permiso del administrador del sistema. Las consecuencias serán obvias de inmediato, por lo que es probable que lo atrapen y sancionen.

32. Examine y trate de interpretar el contenido de un directorio tipo UNIX o Windows con una herramienta como el programa *od* de UNIX o el programa *DEBUG* de MS-DOS. *Sugerencia:* La forma en que haga esto dependerá de lo que permita el SO. Un truco que puede funcionar es crear un directorio en un disco flexible con un sistema operativo y después leer los datos puros del disco usando un sistema operativo distinto que permita dicho acceso.