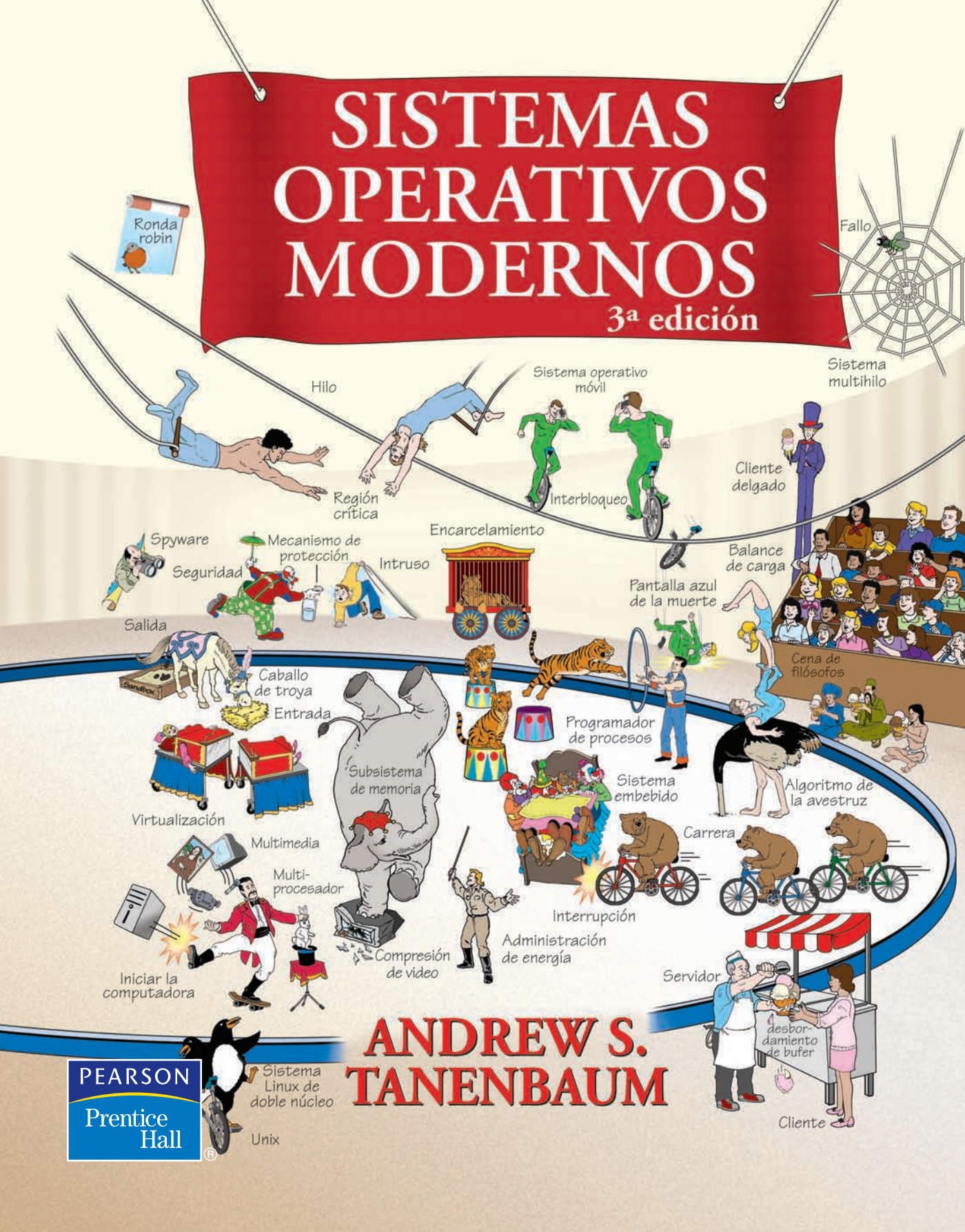
SISTEMAS OPERATIVOS MODERNOS

3ª edición



ANDREW S. TANENBAUM

PEARSON
Prentice Hall

SISTEMAS OPERATIVOS MODERNOS

TERCERA EDICIÓN

SISTEMAS OPERATIVOS MODERNOS

TERCERA EDICIÓN

ANDREW S. TANENBAUM
Vrije Universiteit
Amsterdam, Holanda

TRADUCCIÓN
Alfonso Vidal Romero Elizondo
Ingeniero en Sistemas Computacionales
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

REVISIÓN TÉCNICA
José Ramón Ríos Sánchez
Departamento Académico de Computación
Instituto Tecnológico Autónomo de México

Aarón Jiménez Govea
Catedrático del Departamento de Ciencias Computacionales
Universidad de Guadalajara, México

PEARSON Educación®

México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

Datos de catalogación bibliográfica

TANENBAUM, ANDREW S.

Sistemas operativos modernos. Tercera edición

PEARSON EDUCACIÓN, México, 2009

ISBN: 978-607-442-046-3
Área: Computación

Formato: 18.5 × 23.5 cm Páginas: 1104

ISBN 9780136006633

Edición en español

Editor: Luis Miguel Cruz Castillo
e-mail:luis.cruz@pearsoned.com
Editor de desarrollo: Bernardino Gutiérrez Hernández
Supervisor de producción: José D. Hernández Garduño

Edición en inglés

Editorial Director, Computer Science, Engineering, and Advanced Mathematics: *Marcia J. Horton*
Executive Editor: *Tracy Dunkelberger*
Editorial Assistant: *Melinda Haggerty*
Associtate Editor: *ReeAnne Davis*
Senior Managing Editor: *Scott Disanno*
Production Editor: *Irwin Zucker*
Cover Concept: *Andrews S. Tanenbaum and Tracy Dunkelberger*
Cover Design: *Tamara Newman*
Cover Illustrator: *Steve Lefkowitz*
Interior design: *Andrew S. Tanenbaum*
Typesetting: *Andrew S. Tanenbaum*
Art Director: *Kenny Beck*
Art Editor: *Gregory Dulles*
Media Editor: *David Alick*
Manufacturing Manager: *Alan Fischer*
Manufacturing Buyer: *Lisa McDowell*
Marketing Manager: *Mack Patterson*

TERCERA EDICIÓN, 2009

ISBN: 978-607-442-046-3

Impreso en México. *Printed in Mexico*.

1 2 3 4 5 6 7 8 9 0 09 10 11 12

Para Suzanne, Barbara, Marvin y a la memoria de Bram y Sweetie π

CONTENIDO

PREFACIO

La tercera edición de este libro difiere de la segunda en muchos aspectos. Para empezar, reordenamos los capítulos para colocar el material central al principio. También pusimos mayor énfasis en el sistema operativo como el creador de las abstracciones. El capítulo 1, se ha actualizado en forma considerable y ofrece una introducción de todos los conceptos; el capítulo 2 trata sobre la abstracción de la CPU en varios procesos; el 3 aborda la abstracción de la memoria física en espacios de direcciones (memoria virtual); el 4 versa sobre la abstracción del disco en archivos. En conjunto, los procesos, espacios de direcciones virtuales y archivos son los conceptos clave que proporcionan los sistemas operativos, y es la razón por la que hayamos colocado los capítulos correspondientes antes de lo establecido en la edición anterior.

El capítulo 1 se modificó y actualizó de manera considerable. Por ejemplo, ahora proporciona una introducción al lenguaje de programación C y al modelo de C en tiempo de ejecución para los lectores que están familiarizados sólo con Java.

En el capítulo 2, el análisis de los hilos (threads) se modificó y expandió para reflejar su nueva importancia. Entre otras cosas, ahora hay una sección acerca de los hilos Pthreads del estándar de IEEE.

El capítulo 3, sobre la administración de memoria, se reorganizó para poner énfasis en la idea de que una de las funciones clave de un sistema operativo es proporcionar la abstracción de un espacio de direcciones virtuales para cada proceso. Se eliminó el material que trata de la administración de memoria en los sistemas de procesamiento por lotes, y se actualizó el referente a la implementación de la paginación, con el fin de destacar la necesidad de administrar espacios de direcciones más extensos (ahora muy comunes) y de ofrecer una mayor velocidad.

Se actualizaron los capítulos 4 a 7: se eliminó cierto material anterior y se agregó nuevo. Las secciones sobre las actividades actuales de investigación en estos capítulos se reescribieron desde cero y se agregaron muchos nuevos problemas y ejercicios de programación.

Se actualizó el capítulo 8, incluyendo cierto material acerca de los sistemas multinúcleo. Se agregó una nueva sección sobre la tecnología de hipervirtualización, los hipervisores y las máquinas virtuales, donde se utilizó VMware como ejemplo.

El capítulo 9 se modificó y reorganizó de manera considerable, con mucho material actualizado referente a la explotación de los errores (bugs) en el código, el malware y las defensas contra éstos.

El capítulo 10, que trata acerca de Linux, es una readaptación del anterior (que trataba sobre UNIX y Linux). Sin duda, ahora el énfasis está en Linux y hay una gran cantidad de material nuevo.

El capítulo 11, sobre Windows Vista, es una importante revisión del capítulo 11 anterior, que trataba de Windows 2000. Este capítulo muestra un análisis completamente actualizado de Windows.

El capítulo 12 es nuevo. A mi parecer, los sistemas operativos embebidos, o incrustados (como los que se encuentran en los teléfonos celulares y los asistentes digitales personales, o PDAs) se ignoran en la mayoría de los libros de texto, a pesar de que hay más de estos dispositivos en el mercado que PCs y computadoras portátiles. Esta edición remedia este problema con un análisis extendido de Symbian OS, que se utiliza ampliamente en los teléfonos inteligentes (Smart Phones).

El capítulo 13, sobre el diseño de sistemas operativos, no presenta modificaciones importantes.

Este libro pone a su disposición una gran cantidad de material didáctico de apoyo. Los suplementos para el instructor se encuentran en el sitio web de este libro: **www.pearsoneducacion.net/tanembaum**, e incluyen diapositivas de PowerPoint, herramientas de software para estudiar sistemas operativos, experimentos de laboratorio para los estudiantes, simuladores y material adicional para utilizar en sus cursos. Los instructores que utilicen este libro como texto definitivamente deben darle un vistazo.

Varias personas me ayudaron con esta revisión. En primer lugar deseo agradecer a mi editora, Tracy Dunkelberger. Éste es mi libro número 18 y he desgastado a muchos editores en el proceso. Tracy fue más allá del cumplimiento de su deber con este libro, ya que hizo cosas tales como buscar colaboradores, organizar varias reseñas, ayudar con todos los suplementos, lidiar con los contratos, actuar como intermediario con PH, coordinar una gran cantidad de procesamiento en paralelo y, en general, asegurarse de que todo saliera a tiempo, además de otras cosas. También me ayudó a mantener un estricto itinerario de trabajo para poder lograr que este libro se imprimiera a tiempo, e hizo todo esto manteniéndose animosa y alegre, a pesar de tener muchas otras actividades que exigían gran parte de su atención. Gracias Tracy, en verdad te lo agradezco.

Ada Gavrilovska de Georgia Tech, experta en los aspectos internos sobre Linux, actualizó el capítulo 10 que estaba basado en UNIX (con énfasis en FreeBSD) para convertirlo en algo más enfocado en Linux, aunque gran parte del capítulo sigue siendo genérico para todos los sistemas UNIX. Linux es más popular entre los estudiantes que FreeBSD, por lo que éste es un cambio importante.

Dave Probert de Microsoft actualizó el capítulo 11, que estaba basado en Windows 2000, para concentrarlo en Windows Vista. Aunque tienen varias similitudes, también tienen diferencias considerables. Dave conoce mucho de Windows y tiene suficiente visión como para poder indicar la diferencia entre los puntos en los que Microsoft hizo lo correcto y en los que se equivocó. Como resultado de su trabajo este libro es mucho mejor.

Mike Jipping de Hope College escribió el capítulo acerca de Symbian OS. No abordar los sistemas embebidos en tiempo real era una grave omisión en el libro; gracias a Mike se resolvió ese problema. Los sistemas embebidos en tiempo real se están volviendo cada vez más importantes en el mundo, y este capítulo ofrece una excelente introducción al tema.

A diferencia de Ada, Dave y Mike, cada uno de los cuales se enfocó en un capítulo, Shivakant Mishra, de la University of Colorado en Boulder, fungió más como un sistema distribuido, leyendo y haciendo comentarios sobre muchos capítulos, además de proporcionar una gran cantidad de ejercicios y problemas de programación nuevos a lo largo del libro.

Hugh Lauer también merece una mención especial. Cuando le pedimos ideas sobre cómo revisar la segunda edición, no esperábamos un informe de 23 páginas con interlineado sencillo; y sin embargo eso fue lo que obtuvimos. Muchos de los cambios, como el nuevo énfasis sobre las abstracciones de los procesos, los espacios de direcciones y los archivos, se deben a su participación.

También quiero agradecer a otras personas que me ayudaron de muchas formas, incluyendo el sugerir nuevos temas a cubrir, leer el manuscrito con cuidado, hacer suplementos y contribuir con nuevos ejercicios. Entre ellos Steve Armstrong, Jeffrey Chastine, John Connelly, Mischa Geldermans, Paul Gray, James Griffioen, Jorrit Herder, Michael Howard, Suraj Kothari, Roger Kraft, Trudy Levine, John Masiyowski, Shivakant Mishra, Rudy Pait, Xiao Qin, Mark Russinovich, Krishna Sivalingam, Leendert van Doorn y Ken Wong.

El personal de Prentice Hall fue amistoso y cooperativo como siempre, en especial Irwin Zucker y Scott Disanno en producción, y David Alick, ReeAnne Davies y Melinda Haggerty en editorial.

Por último, pero no por ello menos importante, Barbara y Marvin siguen siendo maravillosos, como siempre, cada uno en una forma especial y única. Y desde luego, quiero agradecer a Suzanne por su amor y paciencia recientes.

Andrew S. Tanenbaum.

ACERCA DEL AUTOR

Andrew S. Tanenbaum tiene una licenciatura en ciencias del M.I.T. y un doctorado de la University of California, en Berkeley. En la actualidad es profesor de Ciencias computacionales en la Vrije Universiteit, en Amsterdam, Holanda, donde encabeza el Grupo de sistemas computacionales. Fue Decano de la Advanced School for Computing and Imaging, una escuela de graduados interuniversidades que realiza investigaciones acerca de los sistemas paralelos avanzados, distribuidos y de creación de imágenes. Ahora es Profesor académico de la Royal Netherlands Academy of Arts and Sciencies, lo cual le ha salvado de convertirse en un burócrata.

Tanenbaum ha realizado investigaciones sobre compiladores, sistemas operativos, redes, sistemas distribuidos de área local y sistemas distribuidos de área amplia que se escalan hasta mil millones de usuarios. Estos proyectos de investigación han producido más de 140 artículos evaluados por expertos en publicaciones especializadas y conferencias. También ha participado como autor o coautor en cinco libros, que a la fecha han aparecido en 18 ediciones. Estos libros se han traducido a 21 idiomas, desde vasco hasta tailandés, y se utilizan en universidades de todo el mundo; La combinación idioma+edición da 130 versiones.

El profesor Tanenbaum también ha producido una cantidad considerable de software. Fue el arquitecto principal del Amsterdam Compiler Kit, un kit de herramientas utilizado ampliamente para escribir compiladores portables. También fue uno de los diseñadores principales de Amoeba, uno de los primeros sistemas distribuidos utilizado en una colección de estaciones de trabajo conectadas mediante una LAN, y Globe, un sistema distribuido de área amplia.

También es autor de MINIX, un pequeño clon de UNIX cuyo propósito principal fue utilizarlo en los laboratorios de programación de los estudiantes. Fue la inspiración directa para Linux y la plataforma en la que se desarrolló inicialmente. La versión actual de MINIX, conocida como MINIX 3, se concentra en un sistema operativo extremadamente confiable y seguro. El profesor Tanenbaum considera que su labor habrá terminado el día que sea innecesario equipar cada computadora con un botón de reinicio. MIINIX 3 es un proyecto continuo de código fuente abierto, al cual todos están invitados a participar. El lector puede visitar *www.minix3.org* para descargar una copia gratuita y averiguar qué es lo que está ocurriendo en la actualidad.

Los estudiantes de doctorado del profesor Tanenbaum han obtenido mayor gloria después de graduarse y él está muy orgulloso de ellos.

Tanenbaum es miembro del ACM, del IEEE y de la Royal Netherlands Academy of Arts and Sciences. También ha recibido muchos premios por su labor científica, entre los que se incluyen:

- 2007 IEEE James H. Mulligan, Jr. Education Medal
- 2003 TAA McGuffey Award for Computer Science and Engineering
- 2002 TAA Texty Award for Computer Science and Engineering
- 1997 ACM/SIGCSE Award for Outstanding Contributions to Computer
- 1994 ACM Karl V. Karlstrom Outstanding Educator Award

También pertenece a la lista de *Quién es quién en el mundo* (*Who's Who in the World)*. Su página Web se encuentra en el URL *http://www.cs.vu.nl/~ast* /.

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.

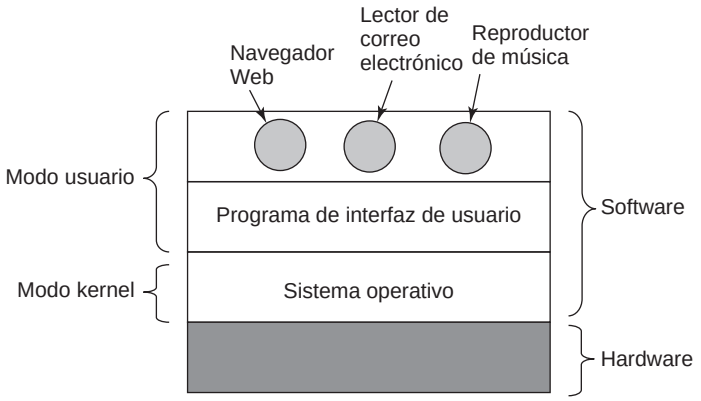


Figura 1-1. Ubicación del sistema operativo.

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.

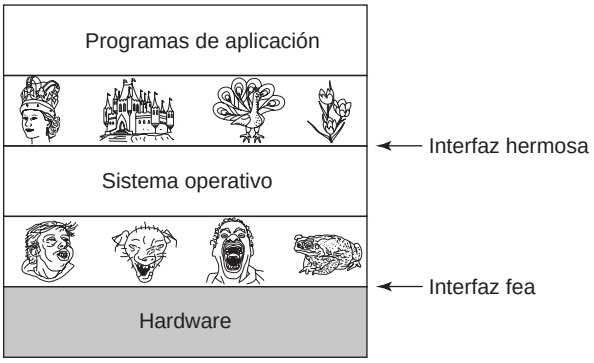


Figura 1-2. Los sistemas operativos ocultan el hardware feo con abstracciones hermosas.

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.

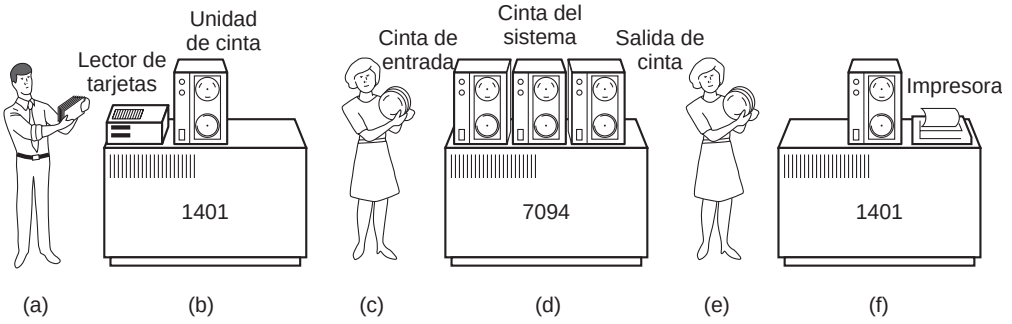


Figura 1-3. Uno de los primeros sistemas de procesamiento por lotes. a) Los programadores llevan las tarjetas a la 1401. b) La 1401 lee los lotes de trabajos y los coloca en cinta. c) El operador lleva la cinta de entrada a la 7094. d) La 7094 realiza los cálculos. e) El operador lleva la cinta de salida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

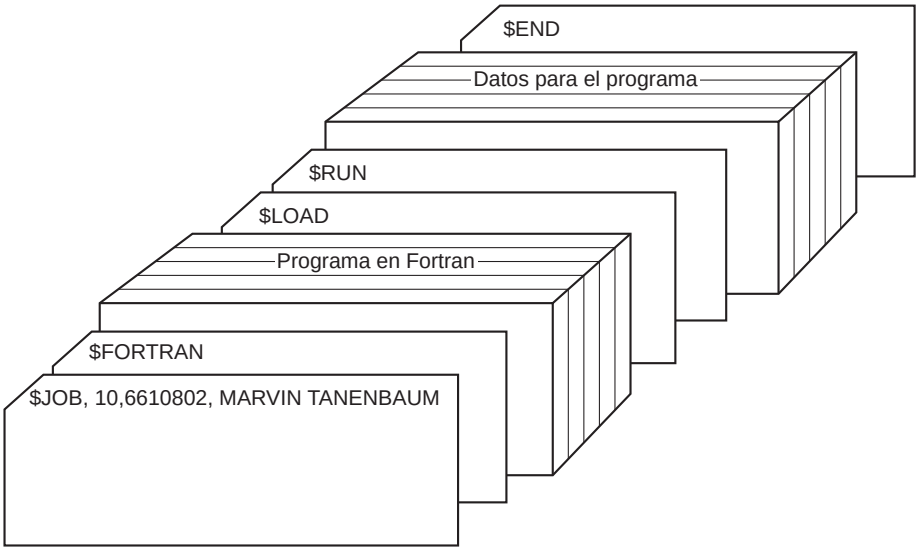


Figura 1-4. Estructura de un trabajo típico de FMS.

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.

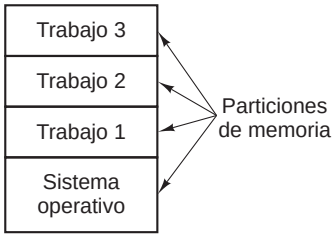


Figura 1-5. Un sistema de multiprogramación con tres trabajos en memoria.

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).

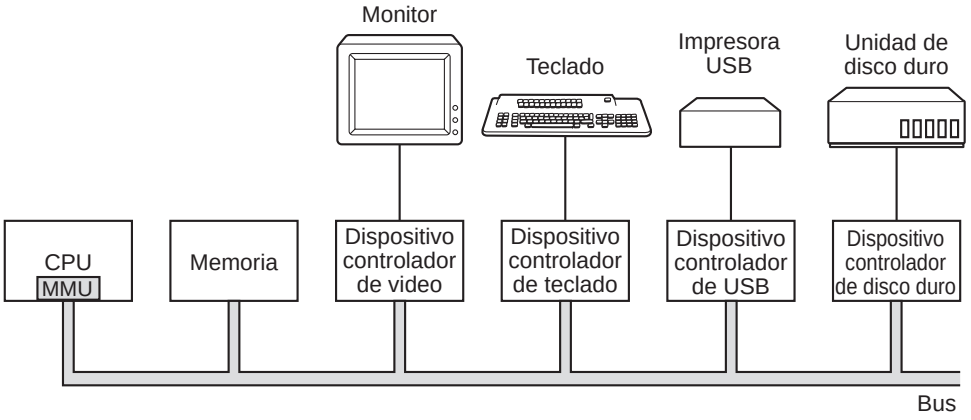


Figura 1-6. Algunos de los componentes de una computadora personal simple.

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-