CAPÍTULO 2

Introducción a los sistemas operativos

Comenzamos nuestro estudio de los sistemas operativos con una breve historia. La historia es interesante y también permite ofrecer una visión general de los principios de los sistemas operativos. La primera sección examina los objetivos y funciones de los sistemas operativos. A continuación, se muestra cómo han evolucionado los sistemas operativos, desde los primitivos sistemas en lotes hasta los sofisticados sistemas multitarea y multiusuario. El resto del capítulo describe la historia y características generales de los dos sistemas operativos que se utilizan como ejemplos a lo largo del libro. Todo el material de este capítulo se cubre en mayor profundidad en el resto de los capítulos.

2.1. OBJETIVOS Y FUNCIONES DE LOS SISTEMAS OPERATIVOS

Un sistema operativo es un programa que controla la ejecución de aplicaciones y programas y que actúa como interfaz entre las aplicaciones y el hardware del computador. Se puede considerar que un sistema operativo tiene los siguientes tres objetivos:

- **Facilidad de uso.** Un sistema operativo facilita el uso de un computador.
- **Eficiencia.** Un sistema operativo permite que los recursos de un sistema de computación se puedan utilizar de una manera eficiente.
- **Capacidad para evolucionar.** Un sistema operativo se debe construir de tal forma que se puedan desarrollar, probar e introducir nuevas funciones en el sistema sin interferir con su servicio.

Se van a examinar en orden estos tres aspectos de un sistema operativo.

EL SISTEMA OPERATIVO COMO UNA INTERFAZ DE USUARIO/COMPUTADOR

El hardware y software utilizados para proporcionar aplicaciones a los usuarios se pueden ver de forma jerárquica o en capas, tal y como se muestra en la Figura 2.1. El usuario de dichas aplicaciones, es decir, el usuario final, normalmente no se preocupa por los detalles del hardware del computador. Por tanto, el usuario final ve un sistema de computación en términos de un conjunto de aplicaciones. Una aplicación se puede expresar en un lenguaje de programación y normalmente es desarrollada por un programador de aplicaciones. Si un programador tuviera que desarrollar una aplicación como un conjunto de instrucciones en código máquina que se encargaran de controlar completamente el hardware del computador, se enfrentaría a una labor extremadamente compleja. Para facilitar esta tarea, se proporcionan un conjunto de programas de sistema. Algunos de estos programas se conocen como utilidades. Estos programas utilizan frecuentemente funciones que asisten al programador en las fases de creación de programas, gestión de ficheros y control de los dispositivos de E/S. Un programador hará uso de estas utilidades cuando desarrolle una aplicación, y las aplicaciones, invocarán a las utilidades durante su ejecución para llevar a cabo ciertas funciones. El programa de sistema más importante es el sistema operativo. El sistema operativo oculta los detalles del hardware al programador y le proporciona una interfaz apropiada para utilizar el sistema. Actúa como mediador, haciendo más fácil al programador y a la aplicación el acceso y uso de dichas utilidades y servicios.

De forma resumida, el sistema operativo proporciona normalmente servicios en las siguientes áreas:

- **Desarrollo de programas.** El sistema operativo proporciona una variedad de utilidades y servicios, tales como editores y depuradores, para asistir al programador en la creación de los

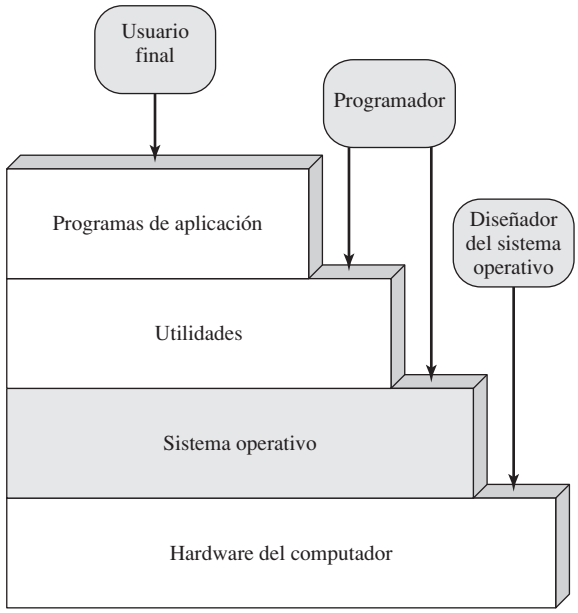


Figura 2.1. Capas y vistas de un sistema de computación.

programas. Normalmente, estos servicios se ofrecen en la forma de utilidades que, aunque no forman parte del núcleo del sistema operativo, se ofrecen con dicho sistema y se conocen como herramientas de desarrollo de programas de aplicación.

- **Ejecución de programas.** Se necesita realizar una serie de pasos para ejecutar un programa. Las instrucciones y los datos se deben cargar en memoria principal. Los dispositivos de E/S y los ficheros se deben inicializar, y otros recursos deben prepararse. Los sistemas operativos realizan estas labores de planificación en nombre del usuario.
- **Acceso a dispositivos de E/S.** Cada dispositivo de E/S requiere su propio conjunto peculiar de instrucciones o señales de control para cada operación. El sistema operativo proporciona una interfaz uniforme que esconde esos detalles de forma que los programadores puedan acceder a dichos dispositivos utilizando lecturas y escrituras sencillas.
- **Acceso controlado a los ficheros.** Para el acceso a los ficheros, el sistema operativo debe reflejar una comprensión detallada no sólo de la naturaleza del dispositivo de E/S (disco, cinta), sino también de la estructura de los datos contenidos en los ficheros del sistema de almacenamiento. Adicionalmente, en el caso de un sistema con múltiples usuarios, el sistema operativo puede proporcionar mecanismos de protección para controlar el acceso a los ficheros.
- **Acceso al sistema.** Para sistemas compartidos o públicos, el sistema operativo controla el acceso al sistema completo y a recursos del sistema específicos. La función de acceso debe proporcionar protección a los recursos y a los datos, evitando el uso no autorizado de los usuarios y resolviendo conflictos en el caso de conflicto de recursos.
- **Detección y respuesta a errores.** Se pueden dar gran variedad de errores durante la ejecución de un sistema de computación. Éstos incluyen errores de hardware internos y externos, tales como un error de memoria, o un fallo en un dispositivo; y diferentes errores software, tales como la división por cero, el intento de acceder a una posición de memoria prohibida o la incapacidad del sistema operativo para conceder la solicitud de una aplicación. En cada caso, el

sistema operativo debe proporcionar una respuesta que elimine la condición de error, suponiendo el menor impacto en las aplicaciones que están en ejecución. La respuesta puede oscilar entre finalizar el programa que causó el error hasta reintentar la operación o simplemente informar del error a la aplicación.

- **Contabilidad.** Un buen sistema operativo recogerá estadísticas de uso de los diferentes recursos y monitorizará parámetros de rendimiento tales como el tiempo de respuesta. En cualquier sistema, esta información es útil para anticipar las necesidades de mejoras futuras y para optimizar el sistema a fin de mejorar su rendimiento. En un sistema multiusuario, esta información se puede utilizar para facturar a los diferentes usuarios.

EL SISTEMA OPERATIVO COMO GESTOR DE RECURSOS

Un computador es un conjunto de recursos que se utilizan para el transporte, almacenamiento y procesamiento de los datos, así como para llevar a cabo el control de estas funciones. El sistema operativo se encarga de gestionar estos recursos.

¿Se puede decir que es el sistema operativo quien controla el transporte, almacenamiento y procesamiento de los datos? Desde un punto de vista, la respuesta es afirmativa: gestionando los recursos del computador, el sistema operativo tiene el control de las funciones básicas del mismo. Pero este control se realiza de una forma curiosa. Normalmente, se habla de un mecanismo de control como algo externo al dispositivo controlado, o al menos como algo que constituye una parte separada o distinta de dicho dispositivo. (Por ejemplo, un sistema de calefacción de una residencia se controla a través de un termostato, que está separado de los aparatos de generación y distribución de calor.) Este no es el caso del sistema operativo, que es un mecanismo de control inusual en dos aspectos:

- Las funciones del sistema operativo actúan de la misma forma que el resto del software; es decir, se trata de un programa o conjunto de programas ejecutados por el procesador.
- El sistema operativo frecuentemente cede el control y depende del procesador para volver a retomarlo.

De hecho, el sistema operativo es un conjunto de programas. Como otros programas, proporciona instrucciones para el procesador. La principal diferencia radica en el objetivo del programa. El sistema operativo dirige al procesador en el uso de los otros recursos del sistema y en la temporización de la ejecución de otros programas. No obstante, para que el procesador pueda realizar esto, el sistema operativo debe dejar paso a la ejecución de otros programas. Por tanto, el sistema operativo deja el control para que el procesador pueda realizar trabajo «útil» y de nuevo retoma el control para permitir al procesador que realice la siguiente pieza de trabajo. Los mecanismos que se utilizan para llevar a cabo esto quedarán más claros a lo largo del capítulo.

La Figura 2.2 muestra los principales recursos gestionados por el sistema operativo. Una porción del sistema operativo se encuentra en la memoria principal. Esto incluye el ***kernel***, o **núcleo**, que contiene las funciones del sistema operativo más frecuentemente utilizadas y, en cierto momento, otras porciones del sistema operativo actualmente en uso. El resto de la memoria principal contiene programas y datos de usuario. La asignación de este recurso (memoria principal) es controlada de forma conjunta por el sistema operativo y el hardware de gestión de memoria del procesador, como se verá. El sistema operativo decide cuándo un programa en ejecución puede utilizar un dispositivo de E/S y controla el acceso y uso de los ficheros. El procesador es también un recurso, y el sistema operativo debe determinar cuánto tiempo de procesador debe asignarse a la ejecución de un programa de usuario particular. En el caso de un sistema multiprocesador, esta decisión debe ser tomada por todos los procesadores.

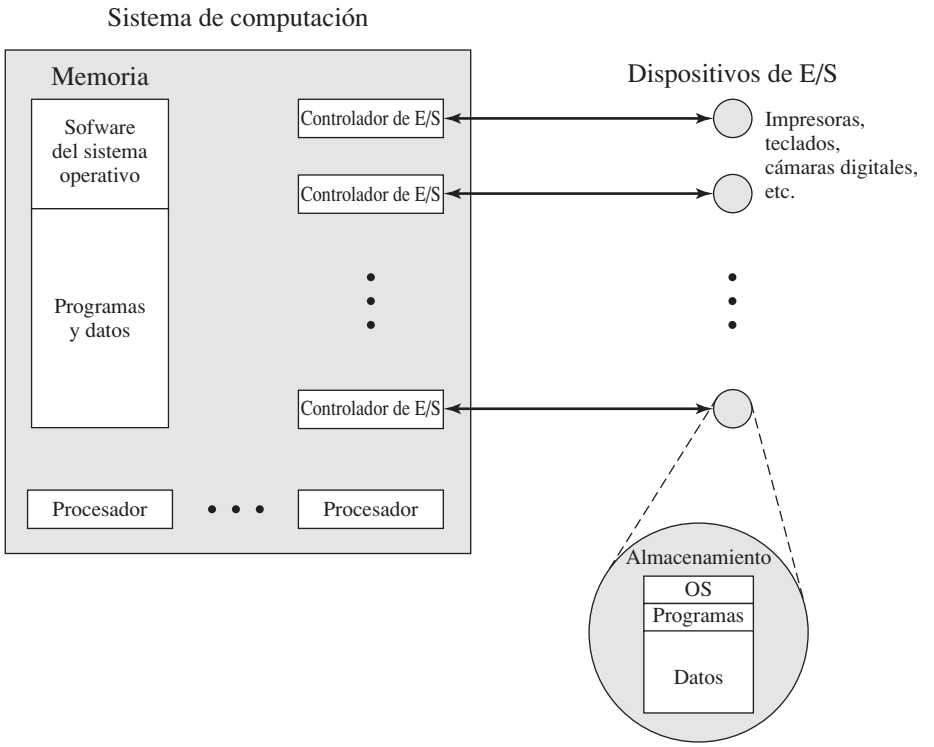


Figura 2.2. El sistema operativo como gestor de recursos.

FACILIDAD DE EVOLUCIÓN DE UN SISTEMA OPERATIVO

Un sistema operativo importante debe evolucionar en el tiempo por las siguientes razones:

- **Actualizaciones de hardware más nuevos tipos de hardware.** Por ejemplo, las primeras versiones de los sistemas operativos UNIX e IBM OS/2 no empleaban un mecanismo de paginado porque ejecutaban en máquinas sin hardware de paginación. La paginación se presenta brevemente, más adelante en este capítulo, y se discute detalladamente en el Capítulo 7. Versiones más recientes de estos sistemas operativos han cambiado esta faceta para explotar las capacidades de paginación. Además, el uso de terminales gráficos y en modo página en lugar de terminales de línea también afecta al diseño de los sistemas operativos. Por ejemplo, un terminal gráfico normalmente permite al usuario ver varias aplicaciones al mismo tiempo a través del uso de «ventanas» en la pantalla. Esto requiere una gestión más sofisticada por parte del sistema operativo.
- **Nuevos servicios.** En respuesta a la demanda del usuario o en respuesta a las necesidades de los gestores de sistema, el sistema operativo debe ofrecer nuevos servicios. Por ejemplo, si es difícil mantener un buen rendimiento con las herramientas existentes, se pueden añadir al sistema operativo nuevas herramientas de medida y control. Como segundo ejemplo, la mayoría de las aplicaciones requieren el uso de ventanas en la pantalla. Esta característica requiere actualizaciones importantes en el sistema operativo si éste no soporta ventanas.
- **Resolución de fallos.** Cualquier sistema operativo tiene fallos. Estos fallos se descubren con el transcurso del tiempo y se resuelven. Por supuesto, esto implica la introducción de nuevos fallos.

La necesidad de cambiar regularmente un sistema operativo introduce ciertos requisitos en su diseño. Un hecho obvio es que el sistema debe tener un diseño modular, con interfaces entre los módulos claramente definidas, y que debe estar bien documentado. Para programas grandes, tal como el típico sistema operativo contemporáneo, llevar a cabo una modularización sencilla no es adecuado [DENN80a]. Se detallará este tema más adelante en el capítulo.

2.2. LA EVOLUCIÓN DE LOS SISTEMAS OPERATIVOS

Para comprender los requisitos claves de un sistema operativo y el significado de las principales características de un sistema operativo contemporáneo, es útil considerar la evolución de los sistemas operativos a lo largo de los años.

PROCESAMIENTO SERIE

Con los primeros computadores, desde finales de los años 40 hasta mediados de los años 50, el programador interaccionaba directamente con el hardware del computador; no existía ningún sistema operativo. Estas máquinas eran utilizadas desde una consola que contenía luces, interruptores, algún dispositivo de entrada y una impresora. Los programas en código máquina se cargaban a través del dispositivo de entrada (por ejemplo, un lector de tarjetas). Si un error provocaba la parada del programa, las luces indicaban la condición de error. El programador podía entonces examinar los registros del procesador y la memoria principal para determinar la causa de error. Si el programa terminaba de forma normal, la salida aparecía en la impresora.

Estos sistemas iniciales presentaban dos problemas principales:

- **Planificación.** La mayoría de las instalaciones utilizaban una plantilla impresa para reservar tiempo de máquina. Típicamente, un usuario podía solicitar un bloque de tiempo en múltiplos de media hora aproximadamente. Un usuario podía obtener una hora y terminar en 45 minutos; esto implicaba malgastar tiempo de procesamiento del computador. Por otro lado, el usuario podía tener problemas, si no finalizaba en el tiempo asignado y era forzado a terminar antes de resolver el problema.
- **Tiempo de configuración.** Un único programa, denominado **trabajo**, podía implicar la carga en memoria del compilador y del programa en lenguaje de alto nivel (programa en código fuente) y a continuación la carga y el enlace del programa objeto y las funciones comunes. Cada uno de estos pasos podían suponer montar y desmontar cintas o configurar tarjetas. Si ocurría un error, el desgraciado usuario normalmente tenía que volver al comienzo de la secuencia de configuración. Por tanto, se utilizaba una cantidad considerable de tiempo en configurar el programa que se iba a ejecutar.

Este modo de operación puede denominarse procesamiento serie, para reflejar el hecho de que los usuarios acceden al computador en serie. A lo largo del tiempo, se han desarrollado varias herramientas de software de sistemas con el fin de realizar el procesamiento serie más eficiente. Estas herramientas incluyen bibliotecas de funciones comunes, enlazadores, cargadores, depuradores, y rutinas de gestión de E/S disponibles como software común para todos los usuarios.

SISTEMAS EN LOTES SENCILLOS

Las primeras máquinas eran muy caras, y por tanto, era importante maximizar su utilización. El tiempo malgastado en la planificación y configuración de los trabajos era inaceptable.

Para mejorar su utilización, se desarrolló el concepto de sistema operativo en lotes. Parece ser que el primer sistema operativo en lotes (y el primer sistema operativo de cualquier clase) fue desarrollado a mediados de los años 50 por General Motors para el uso de un IBM 701 [WEIZ81]. El concepto fue subsecuentemente refinado e implementado en el IBM 704 por un número de clientes de IBM. A principios de los años 60, un número de vendedores había desarrollado sistemas operativos en lote para sus sistemas de computación. IBSYS, el sistema operativo de IBM para los computadores 7090/7094, es particularmente notable por su gran influencia en otros sistemas.

La idea central bajo el esquema de procesamiento en lotes sencillo es el uso de una pieza de software denomina **monitor**. Con este tipo de sistema operativo, el usuario no tiene que acceder directamente a la máquina. En su lugar, el usuario envía un trabajo a través de una tarjeta o cinta al operador del computador, que crea un sistema por lotes con todos los trabajos enviados y coloca la secuencia de trabajos en el dispositivo de entrada, para que lo utilice el monitor. Cuando un programa finaliza su procesamiento, devuelve el control al monitor, punto en el cual dicho monitor comienza la carga del siguiente programa.

Para comprender cómo funciona este esquema, se puede analizar desde dos puntos de vista: el del monitor y el del procesador.

- **Punto de vista del monitor.** El monitor controla la secuencia de eventos. Para ello, una gran parte del monitor debe estar siempre en memoria principal y disponible para la ejecución (Figura 2.3). Esta porción del monitor se denomina **monitor residente**. El resto del monitor está formado por un conjunto de utilidades y funciones comunes que se cargan como subrutinas en el programa de usuario, al comienzo de cualquier trabajo que las requiera. El monitor lee de uno en uno los trabajos desde el dispositivo de entrada (normalmente un lector de tarjetas o dispositivo de cinta magnética). Una vez leído el dispositivo, el trabajo actual se coloca en el área de programa de usuario, y se le pasa el control. Cuando el trabajo se ha completado, devuelve el control al monitor, que inmediatamente lee el siguiente trabajo. Los resultados de cada trabajo se envían a un dispositivo de salida, (por ejemplo, una impresora), para entregárselo al usuario.

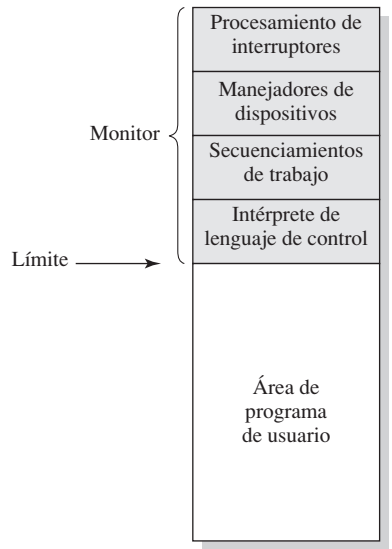


Figura 2.3. Disposición de memoria de un monitor residente.

- **Punto de vista del procesador.** En un cierto punto, el procesador ejecuta instrucciones de la zona de memoria principal que contiene el monitor. Estas instrucciones provocan que se lea el siguiente trabajo y se almacene en otra zona de memoria principal. Una vez que el trabajo se ha leído, el procesador encontrará una instrucción de salto en el monitor que le indica al procesador que continúe la ejecución al inicio del programa de usuario. El procesador entonces ejecutará las instrucciones del programa usuario hasta que encuentre una condición de finalización o de error. Cualquiera de estas condiciones hace que el procesador ejecute la siguiente instrucción del programa monitor. Por tanto, la frase «se pasa el control al trabajo» simplemente significa que el procesador leerá y ejecutará instrucciones del programa de usuario, y la frase «se devuelve el control al monitor» indica que el procesador leerá y ejecutará instrucciones del programa monitor.

El monitor realiza una función de planificación: en una cola se sitúa un lote de trabajos, y los trabajos se ejecutan lo más rápidamente posible, sin ninguna clase de tiempo ocioso entre medias. Además, el monitor mejora el tiempo de configuración de los trabajos. Con cada uno de los trabajos, se incluye un conjunto de instrucciones en algún formato primitivo de **lenguaje de control de trabajos** (*Job Control Language*, JCL). Se trata de un tipo especial de lenguaje de programación utilizado para dotar de instrucciones al monitor. Un ejemplo sencillo consiste en un usuario enviando un programa escrito en el lenguaje de programación FORTRAN más algunos datos que serán utilizados por el programa. Además del código en FORTRAN y las líneas de datos, el trabajo incluye instrucciones de control del trabajo, que se representan mediante líneas que comienzan mediante el símbolo `$´. El formato general del trabajo tiene el siguiente aspecto:

```
$JOB
$FTN
.
.   Instrucciones FORTRAN
.
$LOAD
$RUN
.
.   Datos
.
$END
```

Para ejecutar este trabajo, el monitor lee la línea $FTN y carga el compilador apropiado de su sistema de almacenamiento (normalmente una cinta). El compilador traduce el programa de usuario en código objeto, el cual se almacena en memoria en el sistema de almacenamiento. Si se almacena en memoria, la operación se denomina «compilar, cargar, y ejecutar». En el caso de que se almacene en una cinta, se necesita utilizar la instrucción $LOAD. El monitor lee esta instrucción y recupera el control después de la operación de compilación. El monitor invoca al cargador, que carga el programa objeto en memoria (en el lugar del compilador) y le transfiere el control. De esta forma, se puede compartir un gran segmento de memoria principal entre diferentes subsistemas, aunque sólo uno de ellos se puede ejecutar en un momento determinado.

Durante la ejecución del programa de usuario, cualquier instrucción de entrada implica la lectura de una línea de datos. La instrucción de entrada del programa de usuario supone la invocación de una rutina de entrada, que forma parte del sistema operativo. La rutina de entrada comprueba que el pro-

grama no lea accidentalmente una línea JCL. Si esto sucede, se genera un error y se transfiere el control al monitor. Al finalizar el trabajo de usuario, el monitor analizará todas las líneas de entrada hasta que encuentra la siguiente instrucción JCL. De esta forma, el sistema queda protegido frente a un programa con excesivas o escasas líneas de datos.

El monitor, o sistema operativo en lotes, es simplemente un programa. Éste confía en la habilidad del procesador para cargar instrucciones de diferentes porciones de la memoria principal que de forma alternativa le permiten tomar y abandonar el control. Otras características hardware son también deseables:

- **Protección de memoria.** Durante la ejecución del programa de usuario, éste no debe alterar el área de memoria que contiene el monitor. Si esto ocurriera, el hardware del procesador debe detectar un error y transferir el control al monitor. El monitor entonces abortará el trabajo, imprimirá un mensaje de error y cargará el siguiente trabajo.
- **Temporizador.** Se utiliza un temporizador para evitar que un único trabajo monopolice el sistema. Se activa el temporizador al comienzo de cada trabajo. Si el temporizador expira, se para el programa de usuario, y se devuelve el control al monitor.
- **Instrucciones privilegiadas.** Ciertas instrucciones a nivel de máquina se denominan privilegiadas y sólo las puede ejecutar el monitor. Si el procesador encuentra estas instrucciones mientras ejecuta un programa de usuario, se produce un error provocando que el control se transfiera al monitor. Entre las instrucciones privilegiadas se encuentran las instrucciones de E/S, que permiten que el monitor tome control de los dispositivos de E/S. Esto evita, por ejemplo, que un programa de usuario de forma accidental lea instrucciones de control de trabajos del siguiente trabajo. Si un programa de usuario desea realizar operaciones de E/S, debe solicitar al monitor que realice las operaciones por él.
- **Interrupciones.** Los modelos de computadores iniciales no tenían esta capacidad. Esta característica proporciona al sistema operativo más flexibilidad para dejar y retomar el control desde los programas de usuario.

Ciertas consideraciones sobre la protección de memoria y las instrucciones privilegiadas llevan al concepto de modos de operación. Un programa de usuario ejecuta en **modo usuario**, en el cual los usuarios no pueden acceder a ciertas áreas de memoria y no puede ejecutar ciertas instrucciones. El monitor ejecuta en modo sistema, o lo que se denomina **modo núcleo**, en el cual se pueden ejecutar instrucciones privilegiadas y se puede acceder a áreas de memoria protegidas.

Por supuesto, se puede construir un sistema operativo sin estas características. Pero los fabricantes de computadores rápidamente se dieron cuenta de que los resultados no eran buenos, y de este modo, se construyeron sistemas operativos en lotes primitivos con estas características hardware.

Con un sistema operativo en lotes, el tiempo de máquina alterna la ejecución de programas de usuario y la ejecución del monitor. Esto implica dos sacrificios: el monitor utiliza parte de la memoria principal y consume parte del tiempo de máquina. Ambas situaciones implican una sobrecarga. A pesar de esta sobrecarga, el sistema en lotes simple mejora la utilización del computador.

SISTEMAS EN LOTES MULTIPROGRAMADOS

El procesador se encuentra frecuentemente ocioso, incluso con el secuenciamiento de trabajos automático que proporciona un sistema operativo en lotes simple. El problema consiste en que los dispositivos de E/S son lentos comparados con el procesador. La Figura 2.4 detalla un cálculo representativo de este hecho, que corresponde a un programa que procesa un fichero con registros y

realiza de media 100 instrucciones máquina por registro. En este ejemplo, el computador malgasta aproximadamente el 96% de su tiempo esperando a que los dispositivos de E/S terminen de transferir datos a y desde el fichero. La Figura 2.5a muestra esta situación, donde existe un único programa, lo que se denomina monoprogramación. El procesador ejecuta durante cierto tiempo hasta que alcanza una instrucción de E/S. Entonces debe esperar que la instrucción de E/S concluya antes de continuar.

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

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

Figura 2.4. Ejemplo de utilización del sistema.

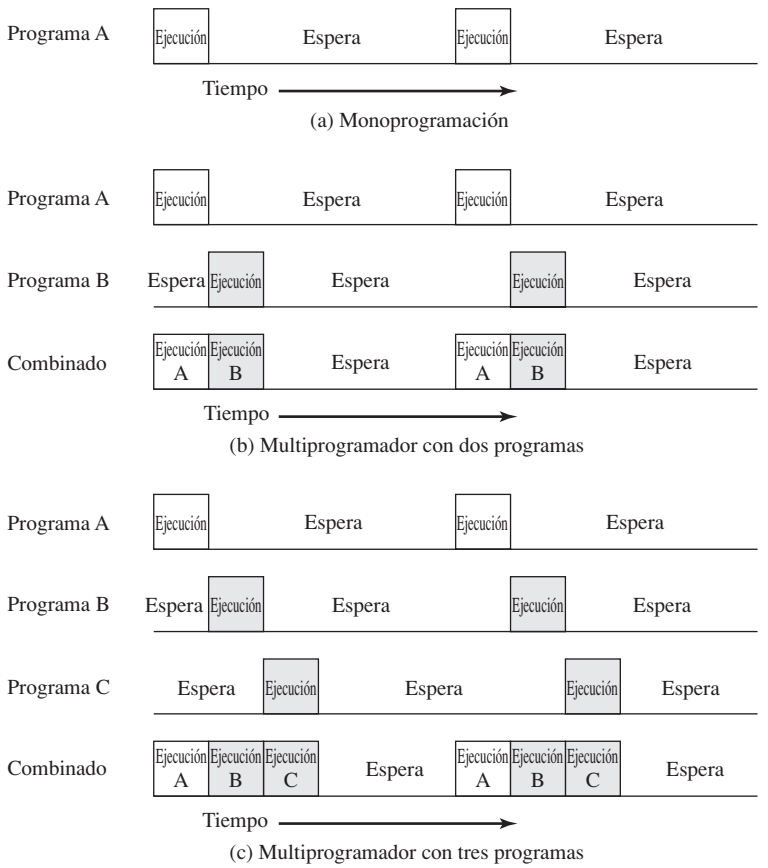


Figura 2.5. Ejemplo de multiprogramación.

Esta ineficiencia puede evitarse. Se sabe que existe suficiente memoria para contener al sistema operativo (monitor residente) y un programa de usuario. Supóngase que hay espacio para el sistema operativo y dos programas de usuario. Cuando un trabajo necesita esperar por la E/S, se puede asignar el procesador al otro trabajo, que probablemente no esté esperando por una operación de E/S (Figura 2.5b). Más aún, se puede expandir la memoria para que albergue tres, cuatro o más programas y pueda haber multiplexación entre todos ellos (Figura 2.5c). Este enfoque se conoce como **multiprogramación** o **multitarea**. Es el tema central de los sistemas operativos modernos.

Para mostrar los beneficios de la multiprogramación, se describe un ejemplo sencillo. Sea un computador con 250 Mbytes de memoria disponible (sin utilizar por el sistema operativo), un disco, un terminal y una impresora. Se envían simultáneamente a ejecución tres programas TRABAJO1, TRABAJO2 y TRABAJO3, con las características listadas en la Tabla 2.1. Se asumen requisitos mínimos de procesador para los trabajos TRABAJO1 y TRABAJO3, así como uso continuo de impresora y disco por parte del trabajo TRABAJO3. En un entorno por lotes sencillo, estos trabajos se ejecutarán en secuencia. Por tanto, el trabajo TRABAJO1 se completará en 5 minutos. El trabajo TRABAJO2 esperará estos 5 minutos y a continuación se ejecutará, terminando 15 minutos después. El trabajo TRABAJO3 esperará estos 20 minutos y se completará 30 minutos después de haberse enviado. La media de utilización de recursos, productividad y tiempos de respuestas se muestran en la columna de monoprogramación de la Tabla 2.2. La utilización de cada dispositivo se ilustra en la Figura 2.6a. Es evidente que existe una infrautilización de todos los recursos cuando se compara respecto al periodo de 30 minutos requerido.

Tabla 2.1. Atributos de ejecución de ejemplos de programas.

	TRABAJO 1	TRABAJO 2	TRABAJO 3
Tipo de trabajo	Computación pesada	Gran cantidad de E/S	Gran cantidad de E/S
Duración	5 minutos	15 minutos	10 minutos
Memoria requerida	50 M	100 M	75 M
¿Necesita disco?	No	No	Sí
¿Necesita terminal?	No	Sí	No
¿Necesita impresora?	No	No	Sí

Tabla 2.2. Efectos de la utilización de recursos sobre la multiprogramación.

	Monoprogramación	Multiprogramación
Uso de procesador	20%	40%
Uso de memoria	33%	67%
Uso de disco	33%	67%
Uso de impresora	33%	67%
Tiempo transcurrido	30 minutos	15 minutos
Productividad	6 trabajos/hora	12 trabajos/hora
Tiempo de respuesta medio	18 minutos	10 minutos

Ahora supóngase que los trabajos se ejecutan concurrentemente bajo un sistema operativo multiprogramado. Debido a que hay poco conflicto entre los trabajos, todos pueden ejecutar casi en el mínimo tiempo mientras coexisten con los otros en el computador (asumiendo que se asigne a los trabajos TRABAJO2 y TRABAJO3 suficiente tiempo de procesador para mantener sus operaciones de entrada y salida activas). El trabajo TRABAJO1 todavía requerirá 5 minutos para completarse, pero al final de este tiempo, TRABAJO2 habrá completado un tercio de su trabajo y TRABAJO3 la mitad. Los tres trabajos habrán finalizado en 15 minutos. La mejora es evidente al examinar la columna de multiprogramación de la Tabla 2.2, obtenido del histograma mostrado en la Figura 2.6b.

Del mismo modo que un sistema en lotes simple, un sistema en lotes multiprogramado también debe basarse en ciertas características hardware del computador. La característica adicional más notable que es útil para la multiprogramación es el hardware que soporta las interrupciones de E/S y DMA (*Direct Memory Access*: acceso directo a memoria). Con la E/S gestionada a través de interrupciones o DMA, el procesador puede solicitar un mandato de E/S para un trabajo y continuar con la ejecución de otro trabajo mientras el controlador del dispositivo gestiona dicha operación de E/S. Cuando esta última operación finaliza, el procesador es interrumpido y se pasa el control a un programa de tratamiento de interrupciones del sistema operativo. Entonces, el sistema operativo pasará el control a otro trabajo.

Los sistemas operativos multiprogramados son bastante sofisticados, comparados con los sistemas **monoprogramados**. Para tener varios trabajos listos para ejecutar, éstos deben guardarse en memoria principal, requiriendo alguna forma de **gestión de memoria**. Adicionalmente, si varios trabajos están listos para su ejecución, el procesador debe decidir cuál de ellos ejecutar; esta decisión requiere un algoritmo para planificación. Estos conceptos se discuten más adelante en este capítulo.

SISTEMAS DE TIEMPO COMPARTIDO

Con el uso de la multiprogramación, el procesamiento en lotes puede ser bastante eficiente. Sin embargo, para muchos trabajos, es deseable proporcionar un modo en el cual el usuario interaccione directamente con el computador. De hecho, para algunos trabajos, tal como el procesamiento de transacciones, un modo interactivo es esencial.

Hoy en día, los computadores personales dedicados o estaciones de trabajo pueden cumplir, y frecuentemente lo hacen, los requisitos que necesita una utilidad de computación interactiva. Esta opción no estuvo disponible hasta los años 60, cuando la mayoría de los computadores eran grandes y costosos. En su lugar, se desarrolló el concepto de tiempo compartido.

Del mismo modo que la multiprogramación permite al procesador gestionar múltiples trabajos en lotes en un determinado tiempo, la multiprogramación también se puede utilizar para gestionar múltiples trabajos interactivos. En este último caso, la técnica se denomina **tiempo compartido**, porque se comparte el tiempo de procesador entre múltiples usuarios. En un sistema de tiempo compartido, múltiples usuarios acceden simultáneamente al sistema a través de terminales, siendo el sistema operativo el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación. Por tanto, si hay n usuarios activos solicitando un servicio a la vez, cada usuario sólo verá en media $1/n$ de la capacidad de computación efectiva, sin contar la sobrecarga introducida por el sistema operativo. Sin embargo, dado el tiempo de reacción relativamente lento de los humanos, el tiempo de respuesta de un sistema diseñado adecuadamente debería ser similar al de un computador dedicado.

Ambos tipos de procesamiento, en lotes y tiempo compartido, utilizan multiprogramación. Las diferencias más importantes se listan en la Tabla 2.3.

Figura 2.6. Histogramas de utilización.

Tabla 2.3. Multiprogramación en lotes frente a tiempo compartido.

	Multiprogramación en lotes	**Tiempo compartido**
Objetivo principal	Maximizar el uso del procesador	Minimizar el tiempo de respuesta
Fuente de directivas al sistema operativo	Mandatos del lenguaje de control de trabajos proporcionados por el trabajo	Mandatos introducidos al terminal

Uno de los primeros sistemas operativos de tiempo compartido desarrollados fue el sistema CTSS (*Compatible Time-Sharing System*) [CORB62], desarrollado en el MIT por un grupo conocido como Proyecto MAC (*Machine-Aided Cognition*, o *Multiple-Access Computers*). El sistema fue inicialmente desarrollado para el IBM 709 en 1961 y más tarde transferido al IBM 7094.

Comparado con sistemas posteriores, CTSS es primitivo. El sistema ejecutó en una máquina con memoria principal con 32.000 palabras de 36 bits, con el monitor residente ocupando 5000 palabras. Cuando el control se asignaba a un usuario interactivo, el programa de usuario y los datos se cargaban en las restantes 27.000 palabras de memoria principal. Para arrancar, un programa siempre se cargaba al comienzo de la palabra 5000; esto simplificaba tanto el monitor como la gestión de memoria. Un reloj del sistema generaba una interrupción cada 0,2 segundos aproximadamente. En cada interrupción de reloj, el sistema operativo retomaba el control y podía asignar el procesador a otro usuario. Por tanto, a intervalos regulares de tiempo, el usuario actual podría ser desalojado y otro usuario puesto a ejecutar. Para preservar el estado del programa de usuario antiguo, los programas de usuario y los datos se escriben en el disco antes de que se lean los nuevos programas de usuario y nuevos datos. Posteriormente, el código y los datos del programa de usuario antiguo se restauran en memoria principal cuando dicho programa vuelve a ser planificado.

Para minimizar el tráfico de disco, la memoria de usuario sólo es escrita a disco cuando el programa entrante la sobreescribe. Este principio queda ilustrado en la Figura 2.7. Sean cuatro usuarios interactivos con los siguiente requisitos de memoria:

- TRABAJO1: 15.000
- TRABAJO2: 20.000
- TRABAJO3: 5000
- TRABAJO4: 10.000

Inicialmente, el monitor carga el trabajo TRABAJO1 y le transfiere control (a). Después, el monitor decide transferir el control al trabajo TRABAJO2. Debido a que el TRABAJO2 requiere más memoria que el TRABAJO1, se debe escribir primero el TRABAJO1 en disco, y a continuación debe cargarse el TRABAJO2 (b). A continuación, se debe cargar el TRABAJO3 para ejecutarse. Sin embargo, debido a que el TRABAJO3 es más pequeño que el TRABAJO2, una porción de este último queda en memoria, reduciendo el tiempo de escritura de disco (c). Posteriormente, el monitor decide transferir el control de nuevo al TRABAJO1. Una porción adicional de TRABAJO2 debe escribirse en disco cuando se carga de nuevo el TRABAJO1 en memoria (d). Cuando se carga el TRABAJO4, parte del trabajo TRABAJO1 y la porción de TRABAJO2 permanecen en memoria (e). En este punto, si cualquiera de estos trabajos (TRABAJO1 o TRABAJO2) son activados, sólo se requiere una carga parcial. En este ejemplo, es el TRABAJO2 el que ejecuta de nuevo. Esto requiere que el TRABAJO4 y la porción residente de el TRABAJO1 se escriban en el disco y la parte que falta de el TRABAJO2 se lea (f).

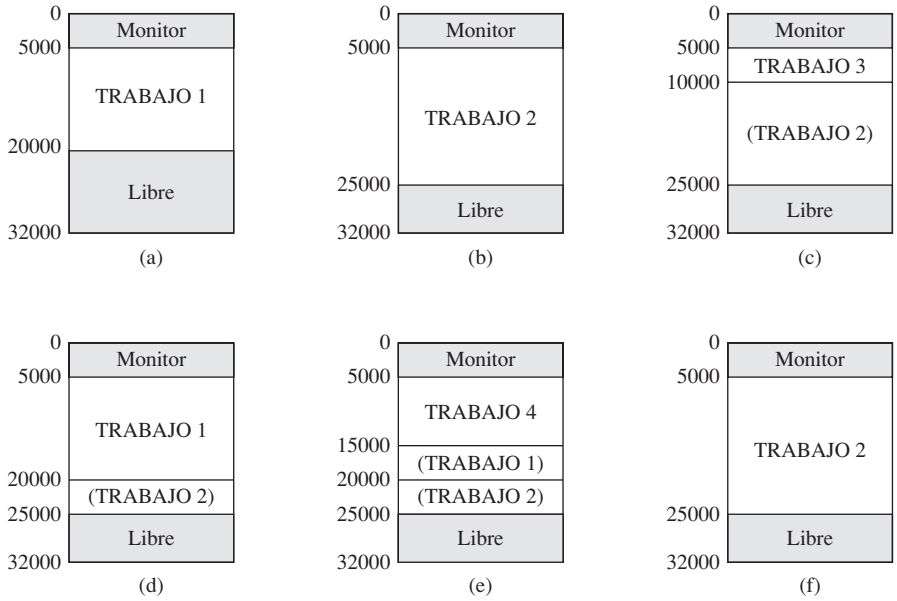


Figura 2.7. Operación del CTSS.

La técnica utilizada por CTSS es primitiva comparada con las técnicas de tiempo compartido actuales, pero funcionaba. Era extremadamente sencilla, lo que minimizaba el tamaño del monitor. Debido a que un trabajo siempre se cargaba en la misma dirección de memoria, no había necesidad de utilizar técnicas de reubicación en tiempo de carga (que se discutirán más adelante). El hecho de sólo escribir en disco cuando es necesario, minimiza la actividad del disco. Ejecutando sobre el 7094, CTSS permitía un número máximo de 32 usuarios.

La compartición de tiempo y la multiprogramación implican nuevos problemas para el sistema operativo. Si existen múltiples trabajos en memoria, éstos deben protegerse para evitar que interfieran entre sí, por ejemplo, a través de la modificación de los datos de los mismos. Con múltiples usuarios interactivos, el sistema de ficheros debe ser protegido, de forma que sólo usuarios autorizados tengan acceso a un fichero particular. También debe gestionarse los conflictos entre los recursos, tal como impresoras y dispositivos de almacenamiento masivo. Éstos y otros problemas, con sus posibles soluciones, se describirán a lo largo de este libro.

2.3. PRINCIPALES LOGROS

Los sistemas operativos se encuentran entre las piezas de software más complejas jamás desarrolladas. Esto refleja el reto de intentar resolver la dificultad de alcanzar determinados objetivos, algunas veces conflictivos, de conveniencia, eficiencia y capacidad de evolución. [DENN80a] propone cinco principales avances teóricos en el desarrollo de los sistemas operativos:

- Procesos.
- Gestión de memoria.
- Protección y seguridad de la información.
- Planificación y gestión de los recursos.
- Estructura del sistema.

Cada avance se caracteriza por principios, o abstracciones, que se han desarrollado para resolver problemas prácticos. Tomadas de forma conjunta, estas cinco áreas incluyen la mayoría de los aspectos clave de diseño e implementación de los sistemas operativos modernos. La breve revisión de estas cinco áreas en esta sección sirve como una visión global de gran parte del resto del libro.

PROCESOS

El concepto de proceso es fundamental en la estructura de los sistemas operativos. Este término fue utilizado por primera vez por los diseñadores del sistema Multics en los años 60 [DALE68]. Es un término un poco más general que el de trabajo. Se han dado muchas definiciones del término *proceso*, incluyendo:

- Un programa en ejecución.
- Una instancia de un programa ejecutándose en un computador.
- La entidad que se puede asignar o ejecutar en un procesador.
- Una unidad de actividad caracterizada por un solo hilo secuencial de ejecución, un estado actual, y un conjunto de recursos del sistema asociados.

Este concepto se aclarará a lo largo del texto.

Tres líneas principales de desarrollo del sistema de computación crearon problemas de temporización y sincronización que contribuyeron al desarrollo del concepto de proceso: operación en lotes multiprogramados, tiempo compartido, y sistemas de transacciones de tiempo real. Como ya se ha visto, la multiprogramación se diseñó para permitir el uso simultáneo del procesador y los dispositivos de E/S, incluyendo los dispositivos de almacenamiento, para alcanzar la máxima eficiencia. El mecanismo clave es éste: en respuesta a las señales que indican la finalización de las transacciones de E/S, el procesador es planificado para los diferentes programas que residen en memoria principal.

Una segunda línea de desarrollo fue el tiempo compartido de propósito general. En este caso, el objetivo clave de diseño es responder a las necesidades del usuario y, debido a razones económicas, ser capaz de soportar muchos usuarios simultáneamente. Estos objetivos son compatibles debido al tiempo de reacción relativamente lento del usuario. Por ejemplo, si un usuario típico necesita una media de 2 segundos de tiempo de procesamiento por minuto, entonces una cantidad de 30 usuarios aproximadamente podría compartir el mismo sistema sin interferencias notables. Por supuesto, la sobrecarga del sistema debe tenerse en cuenta para realizar estos cálculos.

Otra línea importante de desarrollo han sido los sistemas de procesamiento de transacciones de tiempo real. En este caso, un cierto número de usuarios realizan consultas o actualizaciones sobre una base de datos. Un ejemplo es un sistema de reserva para una compañía aérea. La principal diferencia entre el sistema de procesamiento de transacciones y el sistema de tiempo real es que el primero está limitado a una o unas pocas aplicaciones, mientras que los usuarios de un sistema de tiempo real pueden estar comprometidos en el desarrollo de programas, la ejecución de trabajos, y el uso de varias aplicaciones. En ambos casos, el tiempo de respuesta del sistema es impresionante.

La principal herramienta disponible para programadores de sistema para el desarrollo de la inicial multiprogramación y los sistemas interactivos multiusuario fue la interrupción. Cualquier trabajo podía suspender su actividad por la ocurrencia de un evento definido, tal como la finalización de una operación de E/S. El procesador guardaría alguna forma de contexto (por ejemplo, el contador de programa y otros registros) y saltaría a una rutina de tratamiento de interrupciones, que determinaría la naturaleza de la interrupción, procesaría la interrupción, y después continuaría el procesamiento de usuario con el trabajo interrumpido o algún otro trabajo.

El diseño del software del sistema para coordinar estas diversas actividades resultó ser notablemente difícil. Con la progresión simultánea de muchos trabajos, cada uno de los cuales suponía la realización de numerosos pasos para su ejecución secuencial, era imposible analizar todas las posibles combinaciones de secuencias de eventos. Con la ausencia de algún método sistemático de coordinación y cooperación entre las actividades, los programadores acudían a métodos *«ad hoc»* basados en la comprensión del entorno que el sistema operativo tenía que controlar. Estos esfuerzos eran vulnerables frente a errores de programación sutiles, cuyos efectos sólo podían observarse cuando ciertas extrañas secuencias de acciones ocurrían. Estos errores eran difíciles de diagnosticar, porque necesitaban distinguirse de los errores software y hardware de las aplicaciones. Incluso cuando se detectaba el error, era difícil determinar la causa, porque las condiciones precisas bajo las cuales el error aparecía, eran difíciles de reproducir. En términos generales, existen cuatro causas principales de dichos errores [DEBB80a]:

- **Inapropiada sincronización.** Es frecuente el hecho de que una rutina se suspenda esperando por algún evento en el sistema. Por ejemplo, un programa que inicia una lectura de E/S debe esperar hasta que los datos estén disponibles en un *buffer* antes de proceder. En este caso, se necesita una señal procedente de otra rutina. El diseño inapropiado del mecanismo de señalización puede provocar que las señales se pierdan o se reciban señales duplicadas.
- **Violación de la exclusión mutua.** Frecuentemente, más de un programa o usuario intentan hacer uso de recursos compartidos simultáneamente. Por ejemplo, dos usuarios podrían intentar editar el mismo fichero a la vez. Si estos accesos no se controlan, podría ocurrir un error. Debe existir algún tipo de mecanismo de exclusión mutua que permita que sólo una rutina en un momento determinado actualice un fichero. Es difícil verificar que la implementación de la exclusión mutua es correcta en todas las posibles secuencias de eventos.
- **Operación no determinista de un programa.** Los resultados de un programa particular normalmente dependen sólo de la entrada a dicho programa y no de las actividades de otro programa en un sistema compartido. Pero cuando los programas comparten memoria, y sus ejecuciones son entrelazadas por el procesador, podrían interferir entre ellos, sobreescribiendo zonas de memoria comunes de una forma impredecible. Por tanto, el orden en el que diversos programas se planifican puede afectar a la salida de cualquier programa particular.
- **Interbloqueos.** Es posible que dos o más programas se queden bloqueados esperándose entre sí. Por ejemplo, dos programas podrían requerir dos dispositivos de E/S para llevar a cabo una determinada operación (por ejemplo, una copia de un disco o una cinta). Uno de los programas ha tomado control de uno de los dispositivos y el otro programa tiene control del otro dispositivo. Cada uno de ellos está esperando a que el otro programa libere el recurso que no poseen. Dicho interbloqueo puede depender de la temporización de la asignación y liberación de recursos.

Lo que se necesita para enfrentarse a estos problemas es una forma sistemática de monitorizar y controlar la ejecución de varios programas en el procesador. El concepto de proceso proporciona los fundamentos. Se puede considerar que un proceso está formado por los siguientes tres componentes:

- Un programa ejecutable.
- Los datos asociados que necesita el programa (variables, espacio de trabajo, *buffers*, etc.).
- El contexto de ejecución del programa.

Este último elemento es esencial. El **contexto de ejecución**, o **estado del proceso**, es el conjunto de datos interno por el cual el sistema operativo es capaz de supervisar y controlar el proceso. Esta información interna está separada del proceso, porque el sistema operativo tiene información a la que

el proceso no puede acceder. El contexto incluye toda la información que el sistema operativo necesita para gestionar el proceso y que el procesador necesita para ejecutar el proceso apropiadamente. El contexto incluye el contenido de diversos registros del procesador, tales como el contador de programa y los registros de datos. También incluye información de uso del sistema operativo, como la prioridad del proceso y si un proceso está esperando por la finalización de un evento de E/S particular.

La Figura 2.8 indica una forma en la cual los procesos pueden gestionarse. Dos procesos, A y B, se encuentran en una porción de memoria principal. Es decir, se ha asignado un bloque de memoria a cada proceso, que contiene el programa, datos e información de contexto. Se incluye a cada proceso en una lista de procesos que construye y mantiene el sistema operativo. La lista de procesos contiene una entrada por cada proceso, e incluye un puntero a la ubicación del bloque de memoria que contiene el proceso. La entrada podría también incluir parte o todo el contexto de ejecución del proceso. El resto del contexto de ejecución es almacenado en otro lugar, tal vez junto al propio proceso (como queda reflejado en la Figura 2.8) o frecuentemente en una región de memoria separada. El registro índice del proceso contiene el índice del proceso que el procesador está actualmente controlando en la lista de procesos. El contador de programa apunta a la siguiente instrucción del proceso que se va a ejecutar. Los registros base y límite definen la región de memoria ocupada por el proceso: el registro base contiene la dirección inicial de la región de memoria y el registro límite el tamaño de la región (en bytes o palabras). El contador de programa y todas las referencias de datos se interpretan de for-

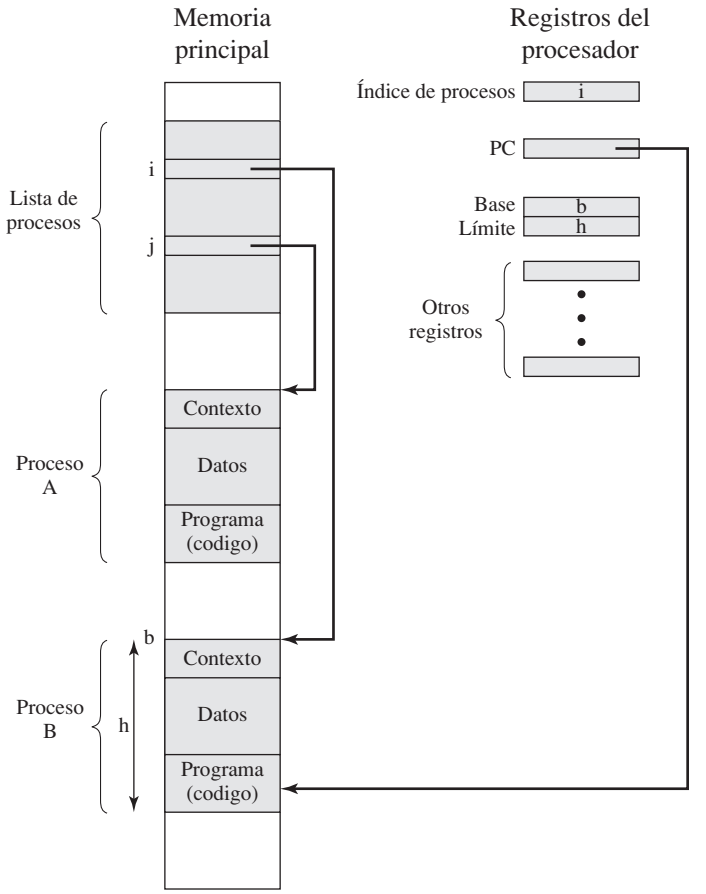


Figura 2.8. Implementación de procesos típica.

ma relativa al registro base y no deben exceder el valor almacenado en el registro límite. Esto previene la interferencia entre los procesos.

En la Figura 2.8, el registro índice del proceso indica que el proceso B está ejecutando. El proceso A estaba ejecutando previamente, pero fue interrumpido temporalmente. Los contenidos de todos los registros en el momento de la interrupción de A fueron guardados en su contexto de ejecución. Posteriormente, el sistema operativo puede cambiar el proceso en ejecución y continuar la ejecución del contexto de A. Cuando se carga el contador de programa con un valor que apunta al área de programa de A, el proceso A continuará la ejecución automáticamente.

Por tanto, el proceso puede verse como una estructura de datos. Un proceso puede estar en ejecución o esperando ejecutarse. El **estado** completo del proceso en un instante dado se contiene en su contexto. Esta estructura permite el desarrollo de técnicas potentes que aseguran la coordinación y la cooperación entre los procesos. Se pueden diseñar e incorporar nuevas características en el sistema operativo (por ejemplo, la prioridad), expandiendo el contexto para incluir cualquier información nueva que se utilice para dar soporte a dicha característica. A lo largo del libro, veremos un gran número de ejemplos donde se utiliza esta estructura de proceso para resolver los problemas provocados por la multiprogramación o la compartición de recursos.

GESTIÓN DE MEMORIA

Un entorno de computación que permita programación modular y el uso flexible de los datos puede ayudar a resolver mejor las necesidades de los usuarios. Los gestores de sistema necesitan un control eficiente y ordenado de la asignación de los recursos. Para satisfacer estos requisitos, el sistema operativo tiene cinco responsabilidades principales de gestión de almacenamiento:

- **Aislamiento de procesos.** El sistema operativo debe evitar que los procesos independientes interfieran en la memoria de otro proceso, tanto datos como instrucciones.
- **Asignación y gestión automática.** Los programas deben tener una asignación dinámica de memoria por demanda, en cualquier nivel de la jerarquía de memoria. La asignación debe ser transparente al programador. Por tanto, el programador no debe preocuparse de aspectos relacionados con limitaciones de memoria, y el sistema operativo puede lograr incrementar la eficiencia, asignando memoria a los trabajos sólo cuando se necesiten.
- **Soporte a la programación modular.** Los programadores deben ser capaces de definir módulos de programación y crear, destruir, y alterar el tamaño de los módulos dinámicamente.
- **Protección y control de acceso.** La compartición de memoria, en cualquier nivel de la jerarquía de memoria, permite que un programa direccione un espacio de memoria de otro proceso. Esto es deseable cuando se necesita la compartición por parte de determinadas aplicaciones. Otras veces, esta característica amenaza la integridad de los programas e incluso del propio sistema operativo. El sistema operativo debe permitir que varios usuarios puedan acceder de distintas formas a porciones de memoria.
- **Almacenamiento a largo plazo.** Muchas aplicaciones requieren formas de almacenar la información durante largos periodos de tiempo, después de que el computador se haya apagado.

Normalmente, los sistemas operativos alcanzan estos requisitos a través del uso de la memoria virtual y las utilidades de los sistemas operativos. El sistema operativo implementa un almacenamiento a largo plazo, con la información almacenada en objetos denominados ficheros. El fichero es un concepto lógico, conveniente para el programador y es una unidad útil de control de acceso y protección para los sistemas operativos.

La memoria virtual es una utilidad que permite a los programas direccionar la memoria desde un punto de vista lógico, sin importar la cantidad de memoria principal física disponible. La memoria virtual fue concebida como un método para tener múltiples trabajos de usuario residiendo en memoria principal de forma concurrente, de forma que no exista un intervalo de tiempo de espera entre la ejecución de procesos sucesivos, es decir, mientras un proceso se escribe en almacenamiento secundario y se lee el proceso sucesor. Debido a que los procesos varían de tamaño, si el procesador planifica un determinado número de procesos, es difícil almacenarlos compactamente en memoria principal. Se introdujeron los sistemas de paginación, que permiten que los procesos se compriman en un número determinado de bloques de tamaño fijo, denominados páginas. Un programa referencia una palabra por medio de una **dirección virtual**, que consiste en un número de página y un desplazamiento dentro de la página. Cada página de un proceso se puede localizar en cualquier sitio de memoria principal. El sistema de paginación proporciona una proyección dinámica entre las direcciones virtuales utilizadas en el programa y una **dirección real**, o dirección física, de memoria principal.

Con el hardware de proyección dinámica disponible, el siguiente paso era eliminar el requisito de que todas las páginas de un proceso residan en memoria principal simultáneamente. Todas las páginas de un proceso se mantienen en disco. Cuando un proceso está en ejecución, algunas de sus páginas se encuentran en memoria principal. Si se referencia una página que no está en memoria principal, el hardware de gestión de memoria lo detecta y permite que la página que falta se cargue. Dicho esquema se denomina área de memoria virtual y está representado en la Figura 2.9.

El hardware del procesador, junto con el sistema operativo, dota al usuario de un «procesador virtual» que tiene acceso a la memoria virtual. Este almacen podría ser un espacio de almacenamiento lineal o una colección de segmentos, que son bloques de longitud variable de direcciones contiguas. En ambos casos, las instrucciones del lenguaje de programación pueden referenciar al programa y a las ubicaciones de los datos en el área de memoria virtual. El aislamiento de los procesos se puede lograr dando a cada proceso una única área de memoria virtual, que no se solape con otras áreas. La compartición de memoria se puede lograr a través de porciones de dos espacios de memoria virtual que se solapan. Los ficheros se mantienen en un almacenamiento a largo plazo. Los ficheros o parte de los mismos se pueden copiar en la memoria virtual para que los programas los manipulen.

La Figura 2.10 destaca los aspectos de direccionamiento de un esquema de memoria virtual. El almacenamiento está compuesto por memoria principal directamente direccionable (a través de instrucciones máquina) y memoria auxiliar de baja velocidad a la que se accede de forma indirecta, cargando bloques de la misma en memoria principal. El hardware de traducción de direcciones (*memory management unit*: unidad de gestión de memoria) se interpone entre el procesador y la memoria. Los programas hacen referencia a direcciones virtuales, que son proyectadas sobre direcciones reales de memoria principal. Si una referencia a una dirección virtual no se encuentra en memoria física, entonces una porción de los contenidos de memoria real son llevados a la memoria auxiliar y los contenidos de la memoria real que se están buscando, son llevados a memoria principal. Durante esta tarea, el proceso que generó la dirección se suspende. El diseñador del sistema operativo necesita desarrollar un mecanismo de traducción de direcciones que genere poca sobrecarga y una política de asignación de almacenamiento que minimice el tráfico entre los diferentes niveles de la jerarquía de memoria.

PROTECCIÓN Y SEGURIDAD DE INFORMACIÓN

El crecimiento del uso de los sistemas de tiempo compartido y, más recientemente, las redes de computadores ha originado un incremento de la preocupación por la protección de la información. La naturaleza de las amenazas que conciernen a una organización variarán enormemente dependiendo de

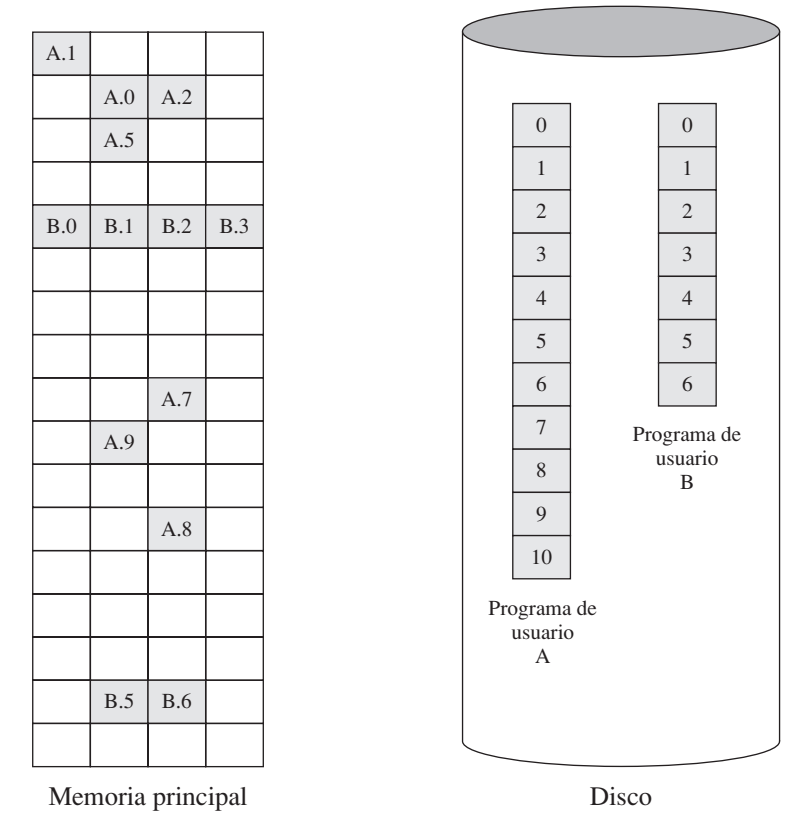


Figura 2.9. Conceptos de memoria virtual.

las circunstancias. Sin embargo, hay algunas herramientas de propósito general que se pueden utilizar en los computadores y sistemas operativos para soportar una gran variedad de mecanismos de protección y seguridad. En general, el principal problema es el control del acceso a los sistemas de computación y a la información almacenada en ellos.

La mayoría del trabajo en seguridad y protección relacionado con los sistemas operativos se puede agrupar de forma genérica en cuatro categorías:

- **Disponibilidad.** Relacionado con la protección del sistema frente a las interrupciones.
- **Confidencialidad.** Asegura que los usuarios no puedan leer los datos sobre los cuales no tienen autorización de acceso.
- **Integridad de los datos.** Protección de los datos frente a modificaciones no autorizadas.
- **Autenticidad.** Relacionado con la verificación apropiada de la identidad de los usuarios y la validez de los mensajes o los datos.

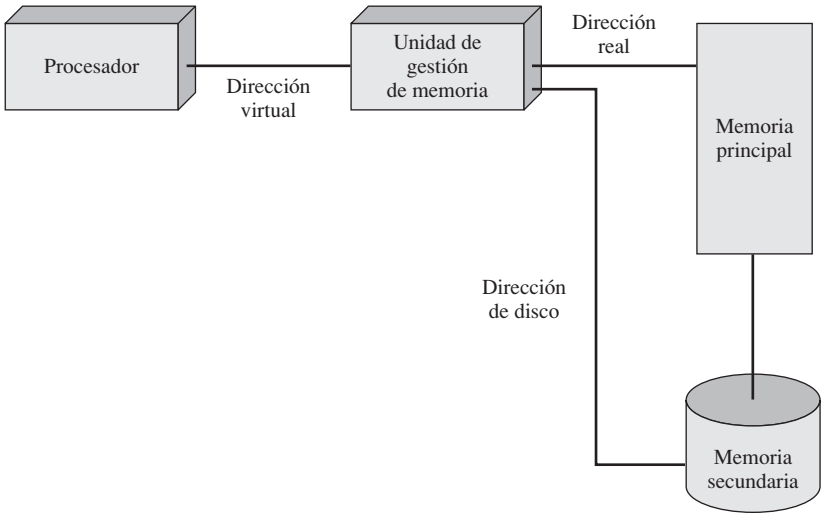


Figura 2.10. Direccionamiento de memoria virtual.

PLANIFICACIÓN Y GESTIÓN DE LOS RECURSOS

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

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

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

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

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

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

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

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

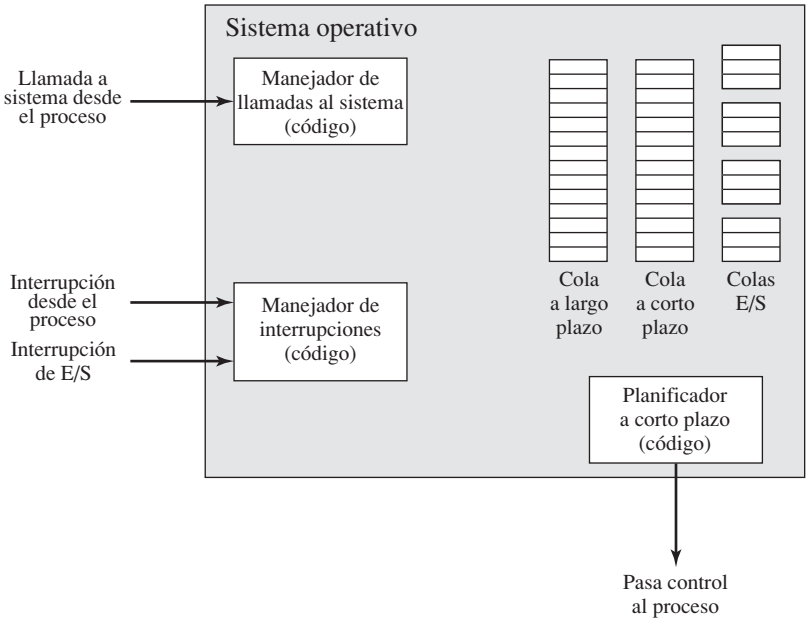


Figura 2.11. Elementos clave de un sistema operativo para la multiprogramación.

ESTRUCTURA DEL SISTEMA

A medida que se han añadido más características a los sistemas operativos y el hardware subyacente se ha vuelto más potente y versátil, han crecido el tamaño y la complejidad de los sistemas operativos. CTSS, que fue puesto en marcha en el MIT en 1963, estaba compuesto aproximadamente por 32.000 palabras de almacenamiento de 36 bits. OS/360, presentado por IBM un año después, tenía más de un millón de instrucciones de máquina. En 1975, el sistema Multics, desarrollado por MIT y los laboratorios Bell, había superado los 20 millones de instrucciones. Es cierto que más recientemente, se han introducido algunos sistemas operativos más sencillos para sistemas más pequeños, pero éstos inevitablemente se hacen más complejos a medida que el hardware subyacente y los requisitos de usuario se incrementan. Por tanto, el sistema UNIX de hoy es muchísimo más complejo que el sistema casi de juguete puesto en marcha por unos pocos programadores de gran talento a comienzo de los años 70, y el sencillo sistema MS-DOS supuso el comienzo de los ricos y complejos sistemas OS/2 y Windows. Por ejemplo, Windows NT 4.0 contiene 16 millones de líneas de código y Windows 2000 duplica este número.

El tamaño de un sistema operativo con un conjunto completo de características, y la dificultad del problema que afronta dicho sistema, ha llevado a esta disciplina a cuatro desafortunados, aunque demasiado comunes, problemas. En primer lugar, los sistemas operativos se entregan tarde de forma crónica. Esto implica la creación de nuevos sistemas operativos y frecuentes actualizaciones a viejos sistemas. En segundo lugar, los sistemas tienen fallos latentes que deben ser planteados y resueltos. En tercer lugar, el rendimiento no es frecuentemente el esperado. En último lugar, se ha comprobado que es imposible construir un sistema operativo complejo que no sea vulnerable a una gran cantidad de ataques de seguridad, incluyendo virus, gusanos y accesos no autorizados.

Para gestionar la complejidad de los sistemas operativos y eliminar estos problemas, se ha puesto mucho énfasis en la estructura software del sistema operativo a lo largo de los años. Ciertos puntos parecen obvios. El software debe ser modular. Esto ayudará a organizar el proceso de desarrollo de software y limitará el esfuerzo de diagnosticar y corregir errores. Los módulos deben tener interfaces bien definidas, y estas interfaces deben ser tan sencillas como sea posible. De nuevo, esto facilita la programación. También facilita la evolución del sistema. Con mínimas interfaces entre los módulos, se puede modificar un módulo con un mínimo impacto en otros módulos.

Para sistemas operativos grandes, que ejecutan desde millones a decenas de millones de líneas de código, no es suficiente la programación modular. De hecho, ha habido un incremento en el uso de los conceptos de capas jerárquicas y abstracción de información. La estructura jerárquica de un sistema operativo moderno separa sus funciones de acuerdo a las características de su escala de tiempo y su nivel de abstracción. Se puede ver el sistema como una serie de niveles. Cada nivel realiza un subconjunto relacionado de funciones requeridas por el sistema operativo. Dicho nivel confía en los niveles inmediatamente inferiores para realizar funciones más primitivas y ocultar los detalles de esas funciones. Cada nivel proporciona servicios a la capa inmediatamente superior. Idealmente, estos niveles deben definirse de tal forma que los cambios en un nivel no requieran cambios en otros niveles. Por tanto, de esta forma se ha descompuesto un problema en un número de subproblemas más manejables.

En general, las capas inferiores tratan con una escala de tiempo menor. Algunas partes del sistema operativo deben interaccionar directamente con el hardware del computador, donde los eventos pueden tener una escala de tiempo tan ínfima como unas pocas mil millonésimas de segundo. En el otro extremo del espectro, algunas partes del sistema operativo se comunican con el usuario, que invoca mandatos en una escala de tiempo mucho más larga, tal vez unos pocos segundos. El uso de un conjunto de niveles se adapta adecuadamente a este entorno.

La forma en que estos principios se aplican varía enormemente entre los sistemas operativos contemporáneos. Sin embargo, es útil en este punto, para el propósito de mostrar los sistemas operativos,

presentar un modelo de sistema operativo jerárquico. Uno propuesto en [BROW84] y [DENN84] es útil, aunque no corresponde a ningún sistema operativo particular. El modelo se define en la Tabla 2.4 y está compuesto por los siguientes niveles:

- **Nivel 1.** Está formado por circuitos electrónicos, donde los objetos tratados son registros, celdas de memoria, y puertas lógicas. Las operaciones definidas en estos objetos son acciones, como poner a cero un registro o leer una posición de memoria.
- **Nivel 2.** El conjunto de instrucciones del procesador. Las operaciones a este nivel son aquéllas permitidas en el conjunto de instrucciones de lenguaje máquina, como adición, resta, carga o almacenamiento.
- **Nivel 3.** Añade el concepto de procedimiento o subrutina, más las operaciones de llamada y retorno (*call*/*return*).
- **Nivel 4.** Introduce las interrupciones, que permiten al procesador guardar el contexto actual e invocar una rutina de tratamiento de interrupciones.

Tabla 2.4. Jerarquía de diseño del sistema operativo.

Nivel	Nombre	Objetos	Ejemplos de operaciones
13	Intérprete de mandatos	Entorno de programación de usuario	Sentencias en lenguaje del intérprete de mandatos
12	Procesos de usuario	Procesos de usuario	Salir, matar, suspender, continuar
11	Directorios	Directorios	Crear, destruir, insertar entrada, eliminar entrada, buscar, listar
10	Dispositivos	Dispositivos externos, como impresoras, pantallas y teclados	Abrir, cerrar, leer, escribir
9	Sistema de ficheros	Ficheros	Crear, destruir, abrir, cerrar, leer, escribir
8	Comunicaciones	Tuberías	Crear, destruir, abrir, cerrar, leer, escribir
7	Memoria virtual	Segmentos, páginas	Leer, escribir, cargar
6	Almacenamiento secundario local	Bloques de datos, canales de dispositivo	Leer, escribir, asignar, liberar
5	Procesos primitivos	Procesos primitivos, semáforos, lista de procesos listos	Suspender, continuar, esperar, señalizar
4	Interrupciones	Programas de gestión de interrupciones	Invocar, enmascarar, desenmascarar, reintentar
3	Procedimientos	Procedimientos, pila de llamadas, registro de activación	Marcar la pila, llamar, retornar
2	Conjunto de instrucciones	Pila de evaluación, intérprete de microprogramas, datos escalares y vectoriales	Cargar, almacenar, sumar, restar, saltar
1	Circuitos electrónicos	Registros, puertas, buses, etc.	Poner a 0, transferir, activar, complementar

El área sombreado en gris representa al hardware.

Estos cuatros primeros niveles no son parte del sistema operativo, sino que constituyen el hardware del procesador. Sin embargo, algunos elementos del sistema operativo se empiezan a mostrar en estos niveles, por ejemplo, las rutinas de tratamiento de interrupciones. Es el Nivel 5 el que corresponde con el sistema operativo propiamente dicho y en el que los conceptos asociados a la multiprogramación aparecen.

- **Nivel 5.** En este nivel se introduce la noción de un proceso como un programa en ejecución. Los requisitos fundamentales de los sistemas operativos para dar soporte a múltiples procesos incluyen la habilidad de suspender y continuar los procesos. Esto requiere guardar los registros de hardware de forma que se pueda interrumpir la ejecución de un proceso e iniciar la de otro. Adicionalmente, si los procesos necesitan cooperar, se necesitan algunos métodos de sincronización. Una de las técnicas más sencillas, y un concepto importante en el diseño de los sistemas operativos, es el semáforo, una técnica de señalización sencilla que se analiza en el Capítulo 5.
- **Nivel 6.** Trata los dispositivos de almacenamiento secundario del computador. En este nivel, se dan la funciones para posicionar las cabezas de lectura/escritura y la transferencia real de bloques. El Nivel 6 delega al Nivel 5 la planificación de la operación y la notificación al proceso solicitante de la finalización de la misma. Niveles más altos se preocupan de la dirección en el disco de los datos requeridos y proporcionan una petición del bloque de datos apropiado a un controlador de dispositivo del Nivel 5.
- **Nivel 7.** Crea un espacio de direcciones lógicas para los procesos. Este nivel organiza el espacio de direcciones virtuales en bloques que pueden moverse entre memoria principal y memoria secundaria. Tres esquemas son de uso común: aquéllos que utilizan páginas de tamaño fijo, aquéllos que utilizan segmentos de longitud variable y aquéllos que utilizan ambos. Cuando un bloque de memoria necesario no se encuentra en memoria principal, la lógica de este nivel requiere una transferencia desde el Nivel 6.

Hasta este punto, el sistema operativo trata con los recursos de un único procesador. Comenzando con el Nivel 8, el sistema operativo trata con objetos externos, como dispositivos periféricos y posiblemente redes y computadores conectados a la red. Los objetos de estos niveles superiores son objetos lógicos con nombre, que pueden compartirse entre procesos del mismo computador o entre múltiples computadores.

- **Nivel 8.** Trata con la comunicación de información y mensajes entre los procesos. Mientras que el Nivel 5 proporciona un mecanismo de señalización primitivo que permite la sincronización de procesos, este nivel trata con una compartición de información más rica. Una de las herramientas más potentes para este propósito es la tubería o *pipe*, que es un canal lógico para el flujo de datos entre los procesos. Una tubería se define por su salida de un proceso y su entrada en otro proceso. Se puede también utilizar para enlazar dispositivos externos o ficheros a procesos. El concepto se discute en el Capítulo 6.
- **Nivel 9.** Da soporte al almacenamiento a largo plazo en ficheros con nombre. En este nivel, los datos en el almacenamiento secundario se ven en términos de entidades abstractas y con longitud variable. Esto contrasta con la visión orientada a hardware del almacenamiento secundario en términos de pistas, sectores y bloques de tamaño fijo en el Nivel 6.
- **Nivel 10.** Proporciona acceso a los dispositivos externos utilizando interfaces estándar.
- **Nivel 11.** Es el nivel responsable para mantener la asociación entre los identificadores externos e internos de los recursos y objetos del sistema. El identificador externo es un nombre que puede utilizar una aplicación o usuario. El identificador interno es una dirección de otro identificador que puede utilizarse por parte de los niveles inferiores del sistema operativo para lo-

calizar y controlar un objeto. Estas asociaciones se mantienen en un directorio. Las entradas no sólo incluyen la asociación entre identificadores externos/internos, sino también características como los derechos de acceso.

- **Nivel 12.** Proporciona una utilidad completa para dar soporte a los procesos. Este nivel va más allá del proporcionado por el Nivel 5. En el Nivel 5, sólo se mantienen los contenidos de los registros del procesador asociados con un proceso, más la lógica de los procesos de planificación. En el Nivel 12, se da soporte a toda la información necesaria para la gestión ordenada de los procesos. Esto incluye el espacio de direcciones virtuales de los procesos, una lista de objetos y procesos con la cual puede interactuar y las restricciones de esta interacción, parámetros pasados al proceso en la creación. También se incluye cualquier otra característica que pudiera utilizar el sistema operativo para controlar el proceso.
- **Nivel 13.** Proporciona una interfaz del sistema operativo al usuario. Se denomina ***shell*** (caparazón), porque separa al usuario de los detalles de los sistemas operativos y presenta el sistema operativo simplemente como una colección de servicios. El *shell* acepta mandatos de usuario o sentencias de control de trabajos, los interpreta y crea y controla los procesos que necesita para su ejecución. Por ejemplo, a este nivel la interfaz podría implementarse de una manera gráfica, proporcionando al usuario mandatos a través de una lista presentada como un menú y mostrando los resultados utilizando una salida gráfica conectada a un dispositivo específico, tal y como una pantalla.

Este modelo hipotético de un sistema operativo proporciona una estructura descriptiva útil y sirve como una guía de implementación. El lector puede volver a estudiar esta estructura durante el desarrollo del libro, para situar cualquier aspecto particular de diseño bajo discusión en el contexto apropiado.

2.4. DESARROLLOS QUE HAN LLEVADO A LOS SISTEMAS OPERATIVOS MODERNOS

A lo largo de los años, ha habido una evolución gradual de la estructura y las capacidades de los sistemas operativos. Sin embargo, en los últimos años se han introducido un gran número de nuevos elementos de diseño tanto en sistemas operativos nuevos como en nuevas versiones de sistemas operativos existentes que han creado un cambio fundamental en la naturaleza de los sistemas operativos. Estos sistemas operativos modernos responden a nuevos desarrollos en hardware, nuevas aplicaciones y nuevas amenazas de seguridad. Entre las causas de hardware principales se encuentran las máquinas multiprocesador, que han logrado incrementar la velocidad de la máquina en mayor medida, los dispositivos de conexión de alta velocidad a la red, y el tamaño creciente y variedad de dispositivos de almacenamiento de memoria. En el campo de las aplicaciones, las aplicaciones multimedia, Internet y el acceso a la Web, y la computación cliente/servidor han influido en el diseño del sistema operativo. Respecto a la seguridad, el acceso a Internet de los computadores ha incrementado en gran medida la amenaza potencial y ataques sofisticados, tales como virus, gusanos, y técnicas de *hacking*, lo que ha supuesto un impacto profundo en el diseño de los sistemas operativos.

La velocidad de cambio en las demandas de los sistemas operativos requiere no sólo modificaciones o mejoras en arquitecturas existentes sino también nuevas formas de organizar el sistema operativo. Un amplio rango de diferentes técnicas y elementos de diseño se han probado tanto en sistemas operativos experimentales como comerciales, pero la mayoría de este trabajo encaja en las siguientes categorías:

- Arquitectura micronúcleo o *microkernel*.
- Multihilo.
- Multiprocesamiento simétrico.

- Sistemas operativos distribuidos.
- Diseño orientado a objetos.

Hasta hace relativamente poco tiempo, la mayoría de los sistemas operativos estaban formados por un gran **núcleo monolítico**. Estos grandes núcleos proporcionan la mayoría de las funcionalidades consideradas propias del sistema operativo, incluyendo la planificación, los sistemas de ficheros, las redes, los controladores de dispositivos, la gestión de memoria y otras funciones. Normalmente, un núcleo monolítico se implementa como un único proceso, con todos los elementos compartiendo el mismo espacio de direcciones. Una **arquitectura micronúcleo** asigna sólo unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos (IPC), y la planificación básica. Ciertos procesos proporcionan otros servicios del sistema operativo, algunas veces denominados servidores, que ejecutan en modo usuario y son tratados como cualquier otra aplicación por el micronúcleo. Esta técnica desacopla el núcleo y el desarrollo del servidor. Los servidores pueden configurarse para aplicaciones específicas o para determinados requisitos del entorno. La técnica micronúcleo simplifica la implementación, proporciona flexibilidad y se adapta perfectamente a un entorno distribuido. En esencia, un micronúcleo interactúa con procesos locales y remotos del servidor de la misma forma, facilitando la construcción de los sistemas distribuidos.

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

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

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

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

1. Tiene múltiples procesadores.
2. Estos procesadores comparten las mismas utilidades de memoria principal y de E/S, interconectadas por un bus de comunicación u otro esquema de conexión interna.
3. Todos los procesadores pueden realizar las mismas funciones (de ahí el término *simétrico*).

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

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

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

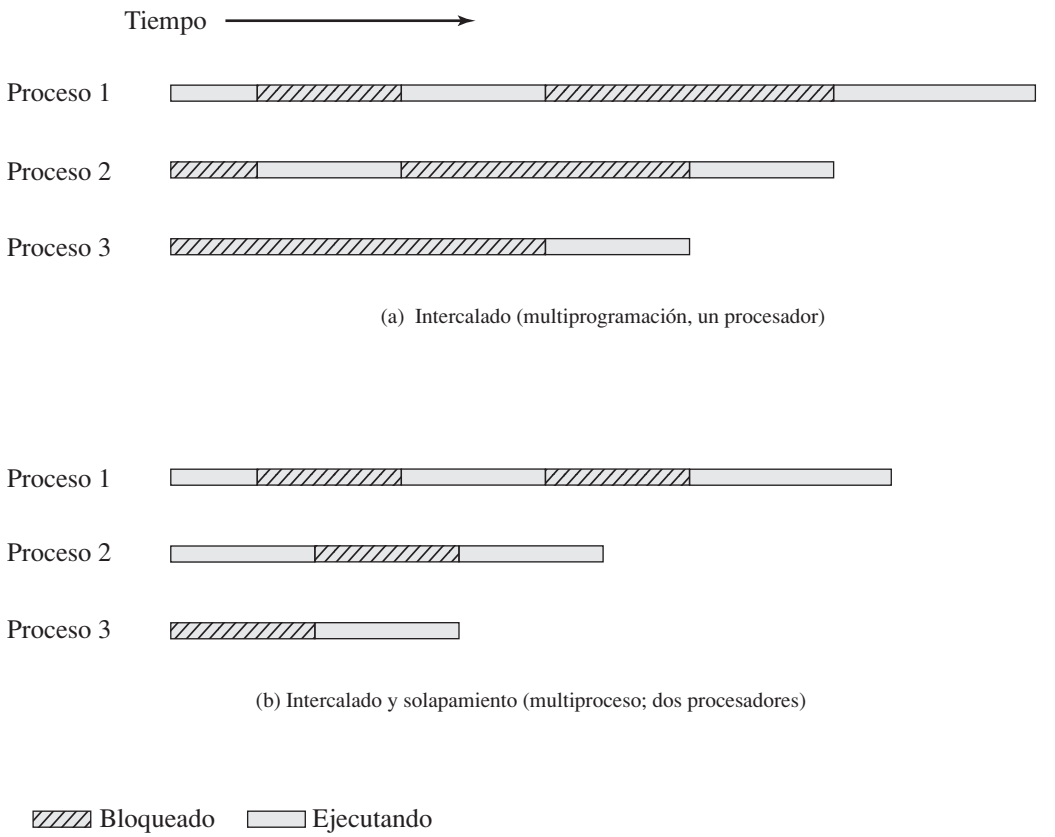


Figura 2.12. Multiprogramación y multiproceso.

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

Una característica atractiva de un SMP es que la existencia de múltiples procesadores es transparente al usuario. El sistema operativo se encarga de planificar los hilos o procesos en procesadores individuales y de la sincronización entre los procesadores. Este libro discute la planificación y los mecanismos de sincronización utilizados para proporcionar una apariencia de único sistema al usuario. Un problema diferente es proporcionar la apariencia de un solo sistema a un cluster de computadores separado-un sistema multicomputador. En este caso, se trata de una colección de entidades (computadores) cada uno con sus propios módulos de memoria principal, de memoria secundaria y otros módulos de E/S.

Un **sistema operativo distribuido** proporciona la ilusión de un solo espacio de memoria principal y un solo espacio de memoria secundario, más otras utilidades de acceso unificadas, como un sistema de ficheros distribuido. Aunque los clusters se están volviendo cada día más populares, y hay muchos productos para clusters en el mercado, el estado del arte de los sistemas distribuidos está retrasado con respecto a los monoprocesadores y sistemas operativos SMP. Se examinarán dichos sistemas en la Parte 6.

Otra innovación en el diseño de los sistemas operativos es el uso de tecnologías orientadas a objetos. El **diseño orientado a objetos** introduce una disciplina al proceso de añadir extensiones modulares a un pequeño núcleo. A nivel del sistema operativo, una estructura basada en objetos permite a los programadores personalizar un sistema operativo sin eliminar la integridad del sistema. La orientación a objetos también facilita el desarrollo de herramientas distribuidas y sistemas operativos distribuidos.

2.5. DESCRIPCIÓN GLOBAL DE MICROSOFT WINDOWS

En esta sección se va a realizar una descripción global de Microsoft Windows; se describirá UNIX en la siguiente sección.

HISTORIA

La historia de Windows comienza con un sistema operativo muy diferente, desarrollado por Microsoft para el primer computador personal IBM y conocido como MS-DOS o PC-DOS. La versión inicial, DOS 1.0 apareció en 1981. Estaba compuesto por 4000 líneas de código fuente en ensamblador y ejecutaba en 8 Kbytes de memoria, utilizando el microprocesador Intel 8086.

Cuando IBM desarrolló un computador personal basado en disco duro, el PC XT, Microsoft desarrolló DOS 2.0, que salió al mercado en 1983. Este sistema daba soporte al disco duro y proporcionaba jerarquía de directorios. Hasta ese momento, un disco podía contener sólo un directorio con ficheros, soportando un máximo de 64 ficheros. Mientras que esto era adecuado en la era de los disquetes, era demasiado limitado para los discos duros, y la restricción de un solo directorio era demasiado burda. Esta nueva versión permitía que los directorios contuvieran tantos subdirectorios como ficheros. La nueva versión también contenía un conjunto de mandatos más rico dentro del sistema operativo, que proporcionaban funciones que eran realizadas como programas externos con la versión 1. Entre las capacidades añadidas se encontraban algunas características de los sistemas UNIX, como la redirección de E/S, que consiste en la capacidad para modificar la entrada o salida de una determinada aplicación, y la impresión en segundo plano. La porción de memoria residente creció a 24 Kbytes.

Cuando IBM anunció el PC AT en 1984, Microsoft introdujo DOS 3.0. El sistema AT contenía el procesador Intel 80286, que proporcionaba características de direccionamiento extendido y protección de memoria. Estas no eran utilizadas por DOS. Para que fuera compatible con versiones anteriores, el sistema operativo simplemente utilizaba el 80286 como un «8086 rápido». El sistema operativo sí daba soporte a un nuevo teclado y periféricos de disco duro. Incluso así, los requisitos de memoria se incrementaron a 36 Kbytes. Hubo varias actualizaciones notables de la versión 3.0. DOS 3.1, que apareció en 1984, daba soporte a la conexión a través de la red para PC. El tamaño de la porción residente no cambió; esto se logró incrementando la cantidad de sistema operativo que podía ser intercambiado (*swapped*). DOS 3.3, que apareció en 1987, daba soporte a la nueva línea de máquinas IBM, las PS/2. De nuevo, esta versión no se beneficiaba de las capacidades del procesador del PS/2, proporcionadas por el 80286 y los *chips* de 32 bits 80386. En este punto, la porción residente había alcanzado un mínimo de 46 Kbytes, incrementándose esta cantidad si se seleccionaban ciertas extensiones opcionales.

En este momento, DOS estaba utilizando el entorno muy por debajo de sus posibilidades. La introducción del 80486 y el *chip* Intel Pentium proporcionaban características que simplemente no podía explotar un sencillo sistema DOS. Mientras tanto, al comienzo de los años 80, Microsoft comenzó a desarrollar una interfaz gráfica de usuario (GUI: *Graphical User Interface*) que sería interpuesta entre el usuario y el sistema operativo DOS. El objetivo de Microsoft era competir con Macintosh, cuyo sistema operativo era insuperable por facilidad de uso. En 1990, Microsoft tenía una versión de la GUI, conocida como Windows 3.0, que incorporaba algunas de las características más amigables de Macintosh. Sin embargo, estaba todavía limitada por la necesidad de ejecutar encima de DOS.

Microsoft intentó el desarrollo de un sistema operativo de nueva generación con IBM para explotar la potencia de los nuevos microprocesadores, el cual incorporaría las características de facilidad de uso de Windows, pero este proyecto fue finalmente abortado. Después de este intento fallido, Microsoft desarrolló un nuevo y propio sistema operativo desde cero, que denominó Windows NT. Windows NT explota las capacidades de los microprocesadores contemporáneos y proporciona multitarea en un entorno mono o multiusuario.

La primera versión de Windows NT (3.1) apareció en 1993, con la misma interfaz gráfica que Windows 3.1, otro sistema operativo de Microsoft (el sucesor de Windows 3.0). Sin embargo, NT 3.1 era un nuevo sistema operativo de 32 bits con la capacidad de dar soporte a las aplicaciones Windows, a las antiguas aplicaciones de DOS y a OS/2.

Después de varias versiones de NT 3.x, Microsoft produjo NT 4.0. NT 4.0 tiene esencialmente la misma arquitectura interna que 3.x. El cambio externo más notable es que NT 4.0 proporciona la misma interfaz de usuario que Windows 95. El cambio arquitectónico más importante es que varios componentes gráficos que ejecutaban en modo usuario como parte del subsistema Win32 en 3.x se mueven al sistema ejecutivo de Windows NT, que ejecuta en modo núcleo. El beneficio de este cambio consiste en la aceleración de estas funciones importantes. La desventaja potencial es que estas funciones gráficas ahora tienen acceso a servicios de bajo nivel del sistema, que pueden impactar en la fiabilidad del sistema operativo.

En 2000, Microsoft introdujo la siguiente principal actualización, que se materializó en el sistema Windows 2000. De nuevo, el sistema ejecutivo subyacente y la arquitectura del núcleo son fundamentalmente los mismos que NT 4.0, pero se han añadido nuevas características. El énfasis en Windows 2000 es la adición de servicios y funciones que dan soporte al procesamiento distribuido. El elemento central de las nuevas características de Windows 2000 es *Active Directory*, que es un servicio de directorios distribuido capaz de realizar una proyección entre nombres de objetos arbitrarios y cualquier información sobre dichos objetos.

Un punto final general sobre Windows 2000 es la distinción entre Windows 2000 Server y el escritorio de Windows 2000. En esencia, el núcleo, la arquitectura ejecutiva y los servicios son los mismos, pero Windows 2000 Server incluye algunos servicios requeridos para su uso como servidor de red.

En 2001, apareció la última versión de escritorio de Windows, conocida como Windows XP. Se ofrecieron versiones de XP tanto de PC de hogar como de estación de trabajo de negocio. También en 2001, apareció una versión de XP de 64 bits. En el año 2003, Microsoft presentó una nueva versión de servidor, conocido como Windows Server 2003. Existen versiones disponibles de 32 y 64 bits. Las versiones de 64 bits de XP y Server 2003 están diseñadas específicamente para el hardware del Intel Itanium de 64 bits.

MULTITAREA MONOUSUARIO

Windows (desde Windows 2000 en adelante) es un ejemplo significativo de lo que significa la nueva ola en los sistemas operativos de microcomputadores (otros ejemplos son OS/2 y MacOS). El desarrollo de Windows fue dirigido por la necesidad de explotar las capacidades de los microprocesadores actuales de 32 bits, que rivalizan con los *mainframes* y minicomputadores de hace unos poco años en velocidad, sofisticaciones de hardware y capacidad de memoria.

Una de las características más significativas de estos nuevos sistemas operativos es que, aunque están todavía pensados para dar soporte a un único usuario interactivo, se trata de sistemas operativos multitarea. Dos principales desarrollos han disparado la necesidad de multitarea en computadores personales, estaciones de trabajo y servidores. En primer lugar, con el incremento de la velocidad y la capacidad de memoria de los microprocesadores, junto al soporte para memoria virtual, las aplicaciones se han vuelto más complejas e interrelacionadas. Por ejemplo, un usuario podría desear utilizar un procesador de texto, un programa de dibujo, y una hoja de cálculo simultáneamente para producir un documento. Sin multitarea, si un usuario desea crear un dibujo y copiarlo en un documento, se requieren los siguientes pasos:

1. Abrir el programa de dibujo.
2. Crear el dibujo y guardarlo en un fichero o en el portapapeles.
3. Cerrar el programa de dibujo.
4. Abrir el procesador de texto.
5. Insertar el dibujo en el sitio adecuado.

Si se desea cualquier cambio, el usuario debe cerrar el procesador de texto, abrir el programa de dibujo, editar la imagen gráfica, guardarlo, cerrar el programa de dibujo, abrir el procesador de texto e insertar la imagen actualizada. Este proceso se vuelve tedioso muy rápidamente. A medida que los servicios y capacidades disponibles para los usuarios se vuelven más potentes y variados, el entorno monotarea se vuelve más burdo y menos amigable. En un entorno multitarea, el usuario abre cada aplicación cuando la necesita, y la deja abierta. La información se puede mover entre las aplicaciones fácilmente. Cada aplicación tiene una o más ventanas abiertas, y una interfaz gráfica con un dispositivo puntero tal como un ratón permite al usuario navegar fácilmente en este entorno.

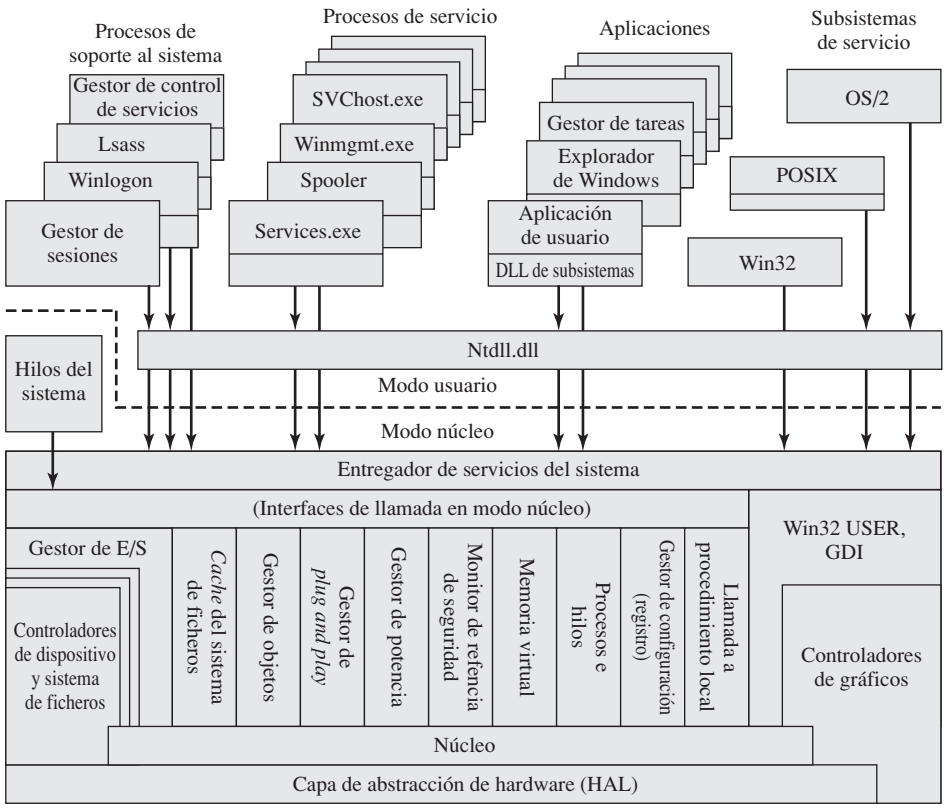
Una segunda motivación para la multitarea es el aumento de la computación cliente/servidor. En este paradigma, un computador personal o estación de trabajo (cliente) y un sistema *host* (servidor) se utilizan de forma conjunta para llevar a cabo una aplicación particular. Los dos están enlazados, y a cada uno se le asigna aquella parte del trabajo que se adapta a sus capacidades. El paradigma cliente/servidor se puede llevar a cabo en una red de área local formada por computadores personales y servidores o por medio de un enlace entre un sistema usuario y un gran *host*, como por ejemplo un *mainframe*. Una aplicación puede implicar a uno o más computadores personales y uno o más dispositivos de servidor. Para proporcionar la respuesta requerida, el sistema operativo necesita dar soporte al hardware de comunicación en tiempo real, los protocolos de comunicación asociados y las arquitecturas de transferencia de datos, y a la vez dar soporte a la interacción de los usuarios.

Las características que se describen a continuación se refieren a la versión Profesional de Windows. La versión Server (Servidor) es también multitarea pero debe permitir el uso de múltiples usuarios. Da soporte a múltiples conexiones locales de servidor y proporciona servicios compartidos que utilizan múltiples usuarios en la red. Como un servidor Internet, Windows puede permitir miles de conexiones web simultáneas.

ARQUITECTURA

La Figura 2.13 muestra la estructura global de Windows 2000; posteriores versiones de Windows tienen esencialmente la misma estructura a este nivel de detalle. Su estructura modular da a Windows una considerable flexibilidad. Se ha diseñado para ejecutar en una variedad de plataformas hardware y da soporte a aplicaciones escritas para una gran cantidad de sistemas operativos. En el momento de escritura de este libro, Windows está implementado solamente en las plataformas hardware Intel Pentium/x86 e Itanium.

Como virtualmente en todos los sistemas operativos, Windows separa el software orientado a aplicación del software del sistema operativo. La segunda parte, que incluye el sistema ejecutivo, el



Lsass = servidor de autenticación de seguridad local
POSIX = interfaz de sistema operativo portable
GDI = interfaz de dispositivo gráfico
DLL = bibliotecas de enlace dinámicas

Áreas coloreadas indican Sistema Ejecutivo

Figura 2.13. Arquitectura de Windows 2000 [SOLO00].

núcleo o *kernel* y la capa de abstracción del hardware, ejecuta en modo núcleo. El software que ejecuta en modo núcleo tiene acceso a los datos del sistema y al hardware. El resto del software, que ejecuta en modo usuario, tiene un acceso limitado a los datos del sistema.

ORGANIZACIÓN DEL SISTEMA OPERATIVO

Windows no tiene una arquitectura micronúcleo pura, sino lo que Microsoft denomina arquitectura micronúcleo modificada. Como en el caso de las arquitecturas micronúcleo puras, Windows es muy modular. Cada función del sistema se gestiona mediante un único componente del sistema operativo. El resto del sistema operativo y todas las aplicaciones acceden a dicha función a través del componente responsable y utilizando una interfaz estándar. Sólo se puede acceder a los datos del sistema claves mediante la función apropiada. En principio, se puede borrar, actualizar o reemplazar cualquier módulo sin reescribir el sistema completo o su interfaz de programa de aplicación (API). Sin embargo, a diferencia de un sistema micronúcleo puro, Windows se configura de forma que muchas de las funciones del sistema externas al micronúcleo ejecutan en modo núcleo. La razón reside en el rendimiento. Los desarrolladores de Windows descubrieron que utilizando la técnica micronúcleo pura, muchas funciones fuera del micronúcleo requerían varios intercambios entre procesos o hilos, cambios de modo y el uso de *buffers* de memoria extra. Los componentes en modo núcleo son los siguientes:

- **Sistema ejecutivo.** Contiene los servicios básicos del sistema operativo, como la gestión de memoria, la gestión de procesos e hilos, seguridad, E/S y comunicación entre procesos.
- **Núcleo.** Está formado por los componentes más fundamentales del sistema operativo. El núcleo gestiona la planificación de hilos, el intercambio de procesos, las excepciones, el manejo de interrupciones y la sincronización de multiprocesadores. A diferencia del resto del sistema ejecutivo y el nivel de usuario, el código del núcleo no se ejecuta en hilos. Por tanto, es la única parte del sistema operativo que no es expulsable o paginable.
- **Capa de abstracción de hardware (HAL: *Hardware Abstraction Layer*).** Realiza una proyección entre mandatos y respuestas hardware genéricos y aquéllos que son propios de una plataforma específica. Aísla el sistema operativo de las diferencias de hardware específicas de la plataforma. El HAL hace que el bus de sistema, el controlador de acceso a memoria directa (DMA), el controlador de interrupciones, los temporizadores de sistema y los módulos de memoria de cada máquina parezcan los mismos al núcleo. También entrega el soporte necesario para multiprocesamiento simétrico (SMP), explicado anteriormente.
- **Controladores de dispositivo.** Incluye tanto sistemas de ficheros como controladores de dispositivos hardware que traducen funciones de E/S de usuario en peticiones específicas a dispositivos hardware de E/S.
- **Gestión de ventanas y sistemas gráficos.** Implementa las funciones de la interfaz gráfica de usuario (GUI), tales como la gestión de ventanas, los controles de la interfaz de usuario y el dibujo.

El sistema ejecutivo de Windows incluye módulos para funciones del sistema específicas y proporciona un API para software en modo usuario. A continuación se describen cada uno de estos módulos del sistema ejecutivo:

- **Gestor de E/S.** Proporciona un entorno a través del cual las aplicaciones pueden acceder a los dispositivos de E/S. El gestor de E/S es responsable de enviar la petición al controlador del dispositivo apropiado para un procesamiento posterior. El gestor de E/S implementa todas las API de E/S de Windows y provee seguridad y nombrado para dispositivos y sistemas de ficheros (utilizando el gestor de objetos). La E/S de Windows se discute en el Capítulo 11.

- **Gestor de cache.** Mejora el rendimiento de la E/S basada en ficheros, provocando que los datos de disco referenciados recientemente residan en memoria principal para un acceso rápido, y retardando las escrituras en disco a través del mantenimiento de las actualizaciones en memoria durante un periodo corto de tiempo antes de enviarlas a disco.
- **Gestor de objetos.** Crea, gestiona y borra los objetos del sistema ejecutivo de Windows y los tipos de datos abstractos utilizados para representar recursos como procesos, hilos y objetos de sincronización. Provee reglas uniformes para mantener el control, el nombrado y la configuración de seguridad de los objetos. El gestor de objetos también crea los manejadores de objetos, que están formados por información de control de acceso y un puntero al objeto. Los objetos de Windows se discuten posteriormente en esta sección.
- **Gestor de *plug and play*.** Determina qué controladores se necesitan para un determinado dispositivo y carga dichos controladores.
- **Gestor de potencia.** Coordina la gestión de potencia entre varios dispositivos y se puede configurar para reducir el consumo de potencia hibernando el procesador.
- **Monitor de referencia de seguridad.** Asegura la validación de acceso y las reglas de generación de auditoría. El modelo orientado a objetos de Windows proporciona una visión de seguridad consistente y uniforme, especificando las entidades fundamentales que constituyen el sistema ejecutivo. Por tanto, Windows utiliza las mismas rutinas de validación de acceso y las comprobaciones de auditoria para todos los objetos protegidos, incluyendo ficheros, procesos, espacios de direcciones y dispositivos de E/S. La seguridad de Windows se discute en el Capítulo 15.
- **Gestor de memoria virtual.** Proyecta direcciones virtuales del espacio de direcciones del proceso a las páginas físicas de la memoria del computador. La gestión de memoria virtual de Windows se describe en el Capítulo 8.
- **Gestor de procesos e hilos.** Crea y borra los objetos y traza el comportamiento de los objetos proceso e hilo. La gestión de los procesos e hilos Windows se describe en el Capítulo 4.
- **Gestor de configuración.** Es responsable de implementar y gestionar el registro del sistema, que es el repositorio para la configuración de varios parámetros a nivel de sistema global y por usuario.
- **Utilidad de llamada a procedimiento local (LPC: *Local Procedure Call*).** Fuerza una relación cliente/servidor entre las aplicaciones y los subsistemas ejecutivos dentro de un único sistema, en un modo similar a una utilidad de llamada a procedimiento remoto (RPC: *Remote Procedure Call*) utilizada para procesamiento distribuido.

PROCESOS EN MODO USUARIO

Hay cuatro tipos básicos de procesos en modo usuario en Windows:

- **Procesos de sistema especiales.** Incluye servicios no proporcionados como parte del sistema operativo Windows, como el proceso de inicio y el gestor de sesiones.
- **Procesos de servicio.** Otros servicios de Windows como por ejemplo, el registro de eventos.
- **Subsistemas de entorno.** Expone los servicios nativos de Windows a las aplicaciones de usuario y por tanto, proporciona un entorno o personalidad de sistema operativo. Los subsistemas soportados son Win32, Posix y OS/2. Cada subsistema de entorno incluye bibliotecas de enlace dinámico (*Dynamic Link Libraries*, DLL), que convierten las llamadas de la aplicación de usuario a llamadas Windows.
- **Aplicaciones de usuario.** Pueden ser de cinco tipos: Win32, Posix, OS/2, Windows 3.1 o MS-DOS.

Windows está estructurado para soportar aplicaciones escritas para Windows 2000 y versiones posteriores. Windows 98 y varios sistemas operativos Windows proporcionan este soporte utilizando un solo sistema ejecutivo compacto a través de subsistemas de entorno protegidos. Los subsistemas protegidos son aquellas partes de Windows que interactúan con el usuario final. Cada subsistema es un proceso separado, y el sistema ejecutivo protege su espacio de direcciones del resto de subsistemas y aplicaciones. Un subsistema protegido proporciona una interfaz de usuario gráfica o de línea de mandatos que define el aspecto del sistema operativo para un usuario. Adicionalmente, cada subsistema protegido proporciona el API para dicho entorno operativo particular.

Esto significa que las aplicaciones creadas para un entorno operativo particular podrían ejecutarse sin ningún cambio en Windows, porque la interfaz del sistema operativo que ven es la misma que aquélla para la que se han escrito. De esta forma, por ejemplo, las aplicaciones basadas en OS/2 se pueden ejecutar en el sistema operativo Windows sin ninguna modificación. Más aún, debido a que el sistema Windows está diseñado para ser independiente de plataforma, mediante el uso de la capa de abstracción de hardware (HAL), debería ser relativamente fácil portar tanto los subsistemas protegidos como las aplicaciones soportadas de una plataforma hardware a otra. En muchos casos, sólo se requiere recompilar.

El subsistema más importante es Win32. Win32 es el API implementada tanto en Windows 2000 y versiones posteriores como en Windows 98. Algunas de las características de Win32 no están disponibles en Windows 98, pero las características implementadas en Windows 98 son idénticas a aquéllas de Windows 2000 y posteriores versiones.

MODELO CLIENTE/SERVIDOR

El sistema ejecutivo, los subsistemas protegidos y las aplicaciones se estructuran de acuerdo al modelo de computación cliente/servidor, que es un modelo común para la computación distribuida, que se discute en la sexta parte. Esta misma arquitectura se puede adoptar para el uso interno de un solo sistema, como es el caso de Windows.

Cada subsistema de entorno y subsistema del servicio ejecutivo se implementa como uno o más procesos. Cada proceso espera la solicitud de un cliente por uno de sus servicios (por ejemplo, servicios de memoria, servicios de creación de procesos o servicios de planificación de procesadores). Un cliente, que puede ser un programa u otro módulo del sistema operativo, solicita un servicio a través del envío de un mensaje. El mensaje se encamina a través del sistema ejecutivo al servidor apropiado. El servidor lleva a cabo la operación requerida y devuelve los resultados o la información de estado por medio de otro mensaje, que se encamina de vuelta al cliente mediante el servicio ejecutivo.

Las ventajas de la arquitectura cliente/servidor son las siguientes:

- Simplifica el sistema ejecutivo. Es posible construir diversos API sin conflictos o duplicaciones en el sistema ejecutivo. Se pueden añadir fácilmente nuevas interfaces.
- Mejora la fiabilidad. Cada módulo de los servicios ejecutivos se ejecuta como un proceso separado, con su propia partición de memoria, protegida de otros módulos. Además, los clientes no pueden acceder directamente al hardware o modificar la zona de memoria en la cual se almacena el sistema ejecutivo. Un único servidor puede fallar sin provocar el fallo o corromper el resto del sistema operativo.
- Proporciona a las aplicaciones maneras uniformes de comunicarse con el sistema ejecutivo a través de los LPC sin restringir la flexibilidad. Las aplicaciones cliente esconden el proceso de paso de mensajes a través de resguardos de funciones, que son contenedores no ejecutables almacenados en bibliotecas de enlace dinámicas (*Dynamic Link Libraries*, DLL). Cuando una

aplicación realiza una llamada a la interfaz del subsistema de entorno, el resguardo de la aplicación cliente empaqueta los parámetros de la llamada y los envía como un mensaje a un subsistema servidor que implementa la llamada.

- Proporciona una base adecuada para la computación distribuida. Normalmente, la computación distribuida utiliza el modelo cliente/servidor, con llamadas a procedimientos remotos implementadas utilizando módulos distribuidos cliente y servidor y el intercambio de mensajes entre clientes y servidores. Con Windows, un servidor local puede pasar un mensaje al servidor remoto para realizar su procesamiento en nombre de las aplicaciones locales cliente. Los clientes no necesitan saber si una petición es atendida local o remotamente. De hecho, si una petición se atiende de forma local o remota, puede cambiar dinámicamente, de acuerdo a las condiciones de carga actuales y a los cambios dinámicos de configuración.

HILOS Y SMP

Dos características importantes de Windows son el soporte que da a los hilos y a SMP, ambas características presentadas en la Sección 2.4. [CUST93] enumera las siguientes características de Windows que dan soporte a los hilos y a los SMP:

- Las rutinas del sistema operativo se pueden ejecutar en cualquier procesador disponible, y diferentes rutinas se pueden ejecutar simultáneamente en diferentes procesadores.
- Windows permite el uso de múltiple hilos de ejecución dentro de un único proceso. Múltiples hilos dentro del mismo proceso se pueden ejecutar en diferentes procesadores simultáneamente.
- Los procesos de servidor pueden utilizar múltiples hilos para procesar peticiones de más de un cliente simultáneamente.
- Windows proporciona mecanismos para compartir datos y recursos entre procesos y capacidades flexibles de comunicación entre procesos.

OBJETOS DE WINDOWS

Windows se apoya enormemente en los conceptos del diseño orientado a objetos. Este enfoque facilita la compartición de recursos y datos entre los procesos y la protección de recursos frente al acceso no autorizado. Entre los conceptos clave del diseño orientado a objetos utilizados por Windows se encuentran los siguientes:

- **Encapsulación.** Un objeto está compuesto por uno o más elementos de información, denominados atributos y uno o más procedimientos que se podrían llevar a cabo sobre esos datos, denominados servicios. La única forma de acceder a los datos en un objeto es invocando uno de los servicios del objeto. Por tanto, los datos de un objeto se pueden proteger fácilmente del uso no autorizado o incorrecto (por ejemplo, intentando ejecutar una pieza de datos no ejecutable).
- **Clases e instancias de objetos.** Una clase de objeto es una plantilla que lista los atributos y los servicios de un objeto y define ciertas características de los objetos. El sistema operativo puede crear instancias específicas de una clase de objetos cuando lo necesite. Por ejemplo, hay una única clase de objeto de proceso y un objeto de proceso por cada proceso actualmente activo. Este enfoque simplifica la creación y gestión de los objetos.
- **Herencia.** Esta característica no es soportada a nivel de usuario sino por alguna extensión dentro del sistema ejecutivo. Por ejemplo, los objetos directorio son ejemplos de objeto contenedor. Una propiedad de un objeto contenedor es que los objetos que contiene pueden heredar

propiedades del contenedor mismo. Como un ejemplo, supongamos que hay un directorio en el sistema de ficheros que está comprimido. Entonces, cualquier fichero que se cree dentro del contenedor directorio también estará comprimido.

- **Polimorfismo.** Internamente, Windows utiliza un conjunto común de funciones para manipular objetos de cualquier tipo; ésta es una característica de polimorfismo, tal y como se define en el Apéndice B. Sin embargo, Windows no es completamente polimórfico, porque hay muchas API que son específicas para tipos de objetos específicos.

El lector que no esté familiarizado con los conceptos orientados a objetos debe revisar el Apéndice B, que se encuentra al final del libro.

No todas las entidades de Windows son objetos. Los objetos se utilizan en casos donde los datos se usan en modo usuario y cuando el acceso a los datos es compartido o restringido. Entre las entidades representadas por los objetos se encuentran los ficheros, procesos, hilos, semáforos, temporizadores y ventanas. Windows crea y gestiona todos los tipos de objetos de una forma uniforme, a través del gestor de objetos. El gestor de objetos es responsable de crear y destruir objetos en nombre de las aplicaciones y de garantizar acceso a los servicios y datos de los objetos.

Cada objeto dentro del sistema ejecutivo, algunas veces denominado objeto del núcleo (para distinguirlo de los objetos a nivel de usuario, objetos que no son gestionados por el sistema ejecutivo), existe como un bloque de memoria gestionado por el núcleo y que es accesible solamente por el núcleo. Algunos elementos de la estructura de datos (por ejemplo, el nombre del objeto, parámetros de seguridad, contabilidad de uso) son comunes a todos los tipos de objetos, mientras que otros elementos son específicos de un tipo de objeto particular (por ejemplo, la prioridad del objeto hilo). Sólo el núcleo puede acceder a estas estructuras de datos de los objetos del núcleo; es imposible que una aplicación encuentre estas estructuras de datos y las lea o escriba directamente. En su lugar, las aplicaciones manipulan los objetos indirectamente a través del conjunto de funciones de manipulación de objetos soportado por el sistema ejecutivo. Cuando se crea un objeto, la aplicación que solicita la creación recibe un manejador del objeto. Esencialmente un manejador es un puntero al objeto referenciado. Cualquier hilo puede utilizar este manejador dentro del mismo proceso para invocar las funciones Win32 que trabajan con objetos.

Los objetos pueden tener información de seguridad asociada con ellos, en la forma de un descriptor de seguridad (*Security Descriptor*, SD). Esta información de seguridad se puede utilizar para restringir el acceso al objeto. Por ejemplo, un proceso puede crear un objeto semáforo con el objetivo de que sólo ciertos usuarios deben ser capaces de abrir y utilizar el semáforo. El SD de dicho objeto semáforo puede estar compuesto por la lista de aquellos usuarios que pueden (o no pueden) acceder al objeto semáforo junto con el conjunto de accesos permitidos (lectura, escritura, cambio, etc.).

En Windows, los objetos pueden tener nombre o no. Cuando un proceso crea un objeto sin nombre, el gestor de objetos devuelve un manejador para dicho objeto, y el manejador es la única forma de referirse a él. Los objetos con nombre son referenciados por otros procesos mediante dicho nombre. Por ejemplo, si un proceso A desea sincronizarse con el proceso B podría crear un objeto de tipo evento con nombre y pasar el nombre del evento a B. El proceso B podría entonces abrir y utilizar el objeto evento. Sin embargo, si A simplemente deseara utilizar el evento para sincronizar dos hilos dentro del proceso, crearía un objeto evento sin nombre, porque no necesita que otros procesos puedan utilizar dicho evento.

Como ejemplo de los objetos gestionados por Windows, a continuación se listan las dos categorías de objetos que gestiona el núcleo:

- **Objetos de control.** Utilizados para controlar las operaciones del núcleo en áreas que no corresponden a la planificación y la sincronización. La Tabla 2.5 lista los objetos de control del núcleo.

- **Objetos *dispatcher*.** Controla la activación y la sincronización de las operaciones del sistema. Estos objetos se describen en el Capítulo 6.

Tabla 2.5. Objetos de control del micronúcleo de Windows [MS96].

Llamada a procedimiento asíncrono	Utilizado para romper la ejecución de un hilo específico y provocar que se llame a un procedimiento en un modo de procesador especificado
Interrupción	Utilizado para conectar un origen de interrupción a una rutina de servicio de interrupciones por medio de una entrada en una tabla IDT (*Interrupt Dispatch Table*). Cada procesador tiene una tabla IDT que se utiliza para entregar interrupciones que ocurren en dicho procesador.
Proceso	Representa el espacio de direcciones virtuales e información de control necesaria para la ejecución de un conjunto de objetos hilo. Un proceso contiene un puntero a un mapa de direcciones, una lista de hilos listos para ejecutar que contiene objetos hilo, una lista de hilos que pertenecen al proceso, el tiempo total acumulado para todos los hilos que se ejecuten dentro del proceso y una prioridad base.
Perfil	Utilizado para medir la distribución de tiempo de ejecución dentro de un bloque de código. Se puede medir el perfil tanto de código de usuario como de sistema.

Windows no es un sistema operativo completamente orientado a objetos. No está implementado en un lenguaje orientado a objetos. Las estructuras de datos que residen completamente dentro de un componente del sistema ejecutivo no están representadas como objetos. Sin embargo, Windows muestra la potencia de la tecnología orientada a objetos y representa la tendencia cada vez más significativa del uso de esta tecnología en el diseño de los sistemas operativos.

2.6. SISTEMAS UNIX TRADICIONALES

HISTORIA

La historia de UNIX es un relato narrado con frecuencia y no se repetirá con muchos detalles aquí. Por el contrario, aquí se realizará un breve resumen.

UNIX se desarrolló inicialmente en los laboratorios Bell y se hizo operacional en un PDP-7 en 1970. Algunas de las personas relacionadas con los laboratorios Bell también habían participado en el trabajo de tiempo compartido desarrollado en el proyecto MAC del MIT. Este proyecto se encargó primero del desarrollo de CTSS y después de Multics. Aunque es habitual decir que el UNIX original fue una versión recortada de Multics, los desarrolladores de UNIX realmente dijeron estar más influenciados por CTSS [RITC78]. No obstante, UNIX incorporó muchas ideas de Multics.

El trabajo de UNIX en los laboratorios Bell, y después en otras instituciones, produjo un conjunto de versiones de UNIX. El primer hito más notable fue portar el sistema UNIX de PDP-7 al PDP-11. Ésta fue la primera pista de que UNIX sería un sistema operativo para todos los computadores. El siguiente hito importante fue la reescritura de UNIX en el lenguaje de programación C. Ésta era una estrategia sin precedentes en ese tiempo. Se pensaba que algo tan complejo como un sistema operativo, que debe tratar eventos críticos en el tiempo, debía escribirse exclusivamente en lenguaje ensamblador. La implementación C demostró las ventajas de utilizar un lenguaje de alto nivel para la mayoría o todo el código del sistema. Hoy, prácticamente todas las implementaciones UNIX están escritas en C.

Estas primeras versiones de UNIX fueron populares dentro de los laboratorios Bell. En 1974, el sistema UNIX se describió en una revista técnica por primera vez [RITC74]. Esto despertó un gran interés por el sistema. Se proporcionaron licencias de UNIX tanto a instituciones comerciales como a universidades. La primera versión completamente disponible fuera de los laboratorios Bell fue la Versión 6, en 1976. La siguiente versión, la Versión 7, aparecida en 1978, es la antecesora de los sistemas UNIX más modernos. El sistema más importante no vinculado con AT&T se desarrolló en la Universidad de California en Berkeley, y se llamó UNIX BSD (*Berkeley Software Distribution*), ejecutándose primero en PDP y después en máquinas VAX. AT&T continuó desarrollando y refinando el sistema. En 1982, los laboratorios Bell combinaron diversas variantes AT&T de UNIX en un único sistema, denominado comercialmente como UNIX System III. Un gran número de características se añadieron posteriormente al sistema operativo para producir UNIX System V.

DESCRIPCIÓN

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

La Figura 2.15 muestra una vista más cercana del núcleo. Los programas de usuario pueden invocar los servicios del sistema operativo directamente o a través de programas de biblioteca. La interfaz de llamada a sistemas es la frontera con el usuario y permite que el software de alto nivel obtenga acceso a funciones específicas de núcleo. En el otro extremo, el sistema operativo contiene rutinas primitivas que interaccionan directamente con el hardware. Entre estas dos interfaces, el sistema se divi-

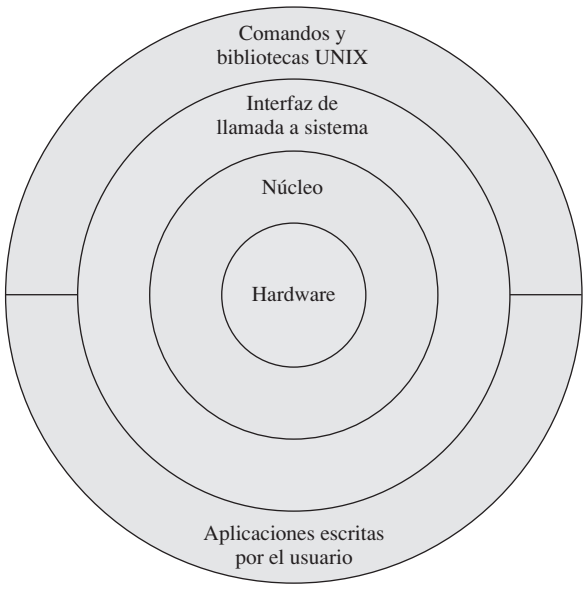


Figura 2.14. Arquitectura general de UNIX.

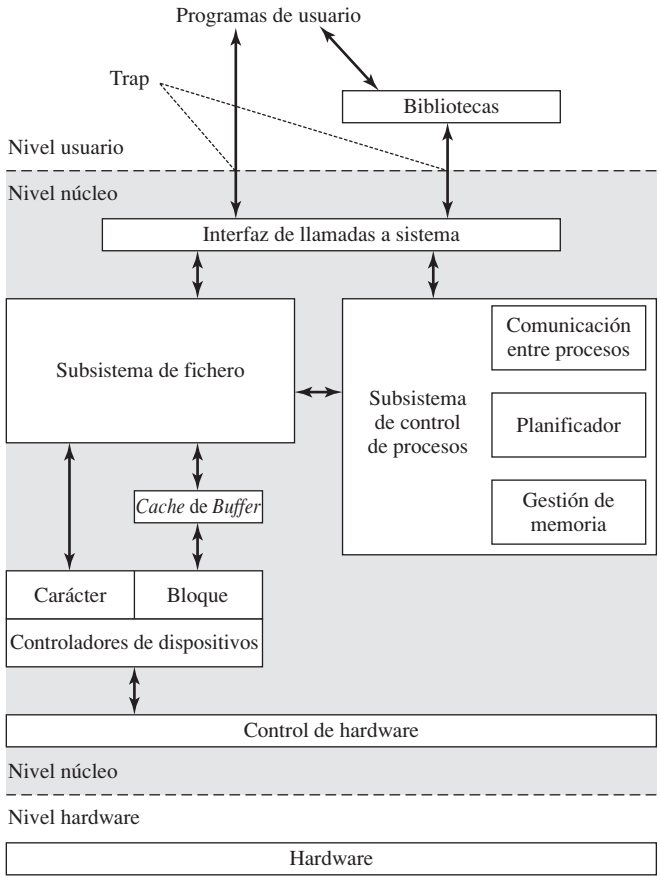


Figura 2.15. Núcleo tradicional de UNIX [BACH86].

de en dos partes principales, una encargada del control de procesos y la otra encargada de la gestión de ficheros y de la E/S. El subsistema de control de procesos se encarga de la gestión de memoria, la planificación y ejecución de los procesos, así como de la sincronización y la comunicación entre los procesos. El sistema de ficheros intercambia datos entre la memoria y los dispositivos externos tanto como flujos de caracteres como bloques. Para lograr esto, se utilizan una gran variedad de controladores de dispositivos. Para las transferencias orientadas a bloques, se utiliza una técnica de cache de discos: entre el espacio de direccionamiento del usuario y el dispositivo externo se interpone un *buffer* de sistema en memoria principal.

La descripción de esta subsección se refiere a lo que se han denominado sistemas UNIX tradicionales; [VAHA96] utiliza este término para referirse a System V Versión 3 (SVR3: *System V Release 3*), 4.3BSD y versiones anteriores. Las siguientes sentencias generales pueden afirmarse sobre un sistema UNIX tradicional. Se diseñaron para ejecutar sobre un único procesador y carecen de la capacidad para proteger sus estructuras de datos del acceso concurrente por parte de múltiples procesadores. Su núcleo no es muy versátil, soportando un único tipo de sistema de ficheros, una única política de planificación de procesos y un único formato de fichero ejecutable. El núcleo tradicional de UNIX no está diseñado para ser extensible y tiene pocas utilidades para la reutilización de código. El resultado es que, según se iban añadiendo nuevas características a varias versiones de UNIX, se tuvo que añadir mucho código, proporcionando un núcleo de gran tamaño y no modular.

2.7. SISTEMAS UNIX MODERNOS

Cuando UNIX evolucionó, un gran número de diferentes implementaciones proliferó, cada una de las cuales proporcionó algunas características útiles. Fue necesaria la producción de una nueva implementación que unificara muchas de las importantes innovaciones, añadiera otras características de diseño de los sistemas operativos modernos, y produjera una arquitectura más modular. La arquitectura mostrada en la Figura 2.16 muestra los aspectos típicos de un núcleo UNIX moderno. Existe un pequeño núcleo de utilidades, escritas de forma modular, que proporciona funciones y servicios necesarios para procesos del sistema operativo. Cada uno de los círculos externos representa funciones y una interfaz que podría implementarse de diferentes formas.

Ahora se verán algunos ejemplos de sistemas UNIX modernos.

SYSTEM V RELEASE 4 (SVR4)

SVR4, desarrollado conjuntamente por AT&T y Sun Microsistemas, combina características de SVR3, 4.3BSD, Microsoft Xenix System y SunOS. Fue casi una reescritura completa del núcleo del

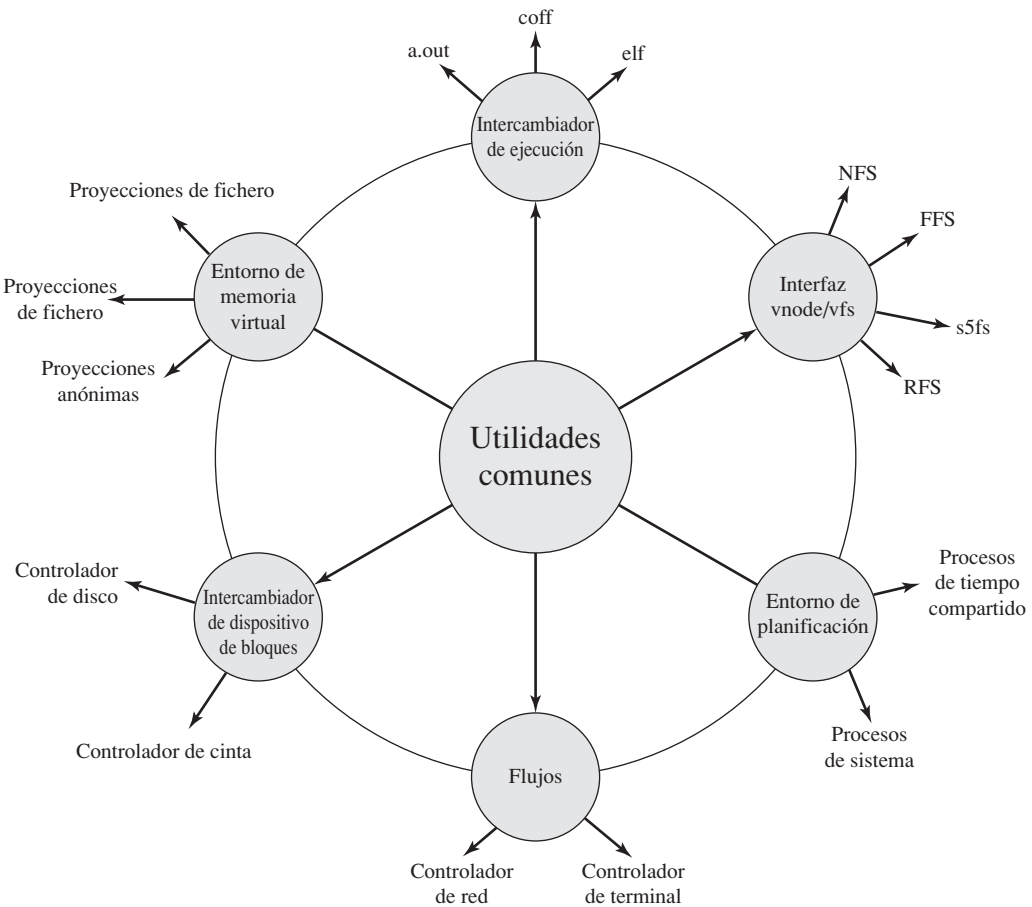


Figura 2.16. Núcleo UNIX moderno [VAHA96].

System V y produjo una implementación bien organizada, aunque compleja. Las nuevas características de esta versión incluyen soporte al procesamiento en tiempo real, clases de planificación de procesos, estructuras de datos dinámicamente asignadas, gestión de la memoria virtual, sistema de ficheros virtual y un núcleo expulsivo.

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

SOLARIS 9

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

4.4BSD

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

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

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

2.8. LINUX

HISTORIA

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

La clave del éxito de Linux ha sido la disponibilidad de los paquetes de software libre bajo los auspicios de la Fundación de Software Libre (*Free Software Foundation*, FSF). Esta fundación se centra en un software estable, independiente de plataforma, con alta calidad, y soportado por la comunidad de usuarios. El proyecto de GNU proporciona herramientas para desarrolladores de software y la licencia pública GNU (GPL: *GNU Public License*) es el sello de aprobación de FSF. Torvald utilizó herramientas GNU para el desarrollo del núcleo, que fue posteriormente distribuido bajo la licencia GPL. Por tanto, las distribuciones Linux que aparecen hoy en día son los productos del proyecto GNU de FSF, los esfuerzos individuales de Torvalds y muchos colaboradores a lo largo del mundo.

Además de su uso por muchos programadores individuales, Linux ha hecho ahora una penetración significativa en el mundo corporativo. Esto no es sólo debido al software libre, sino también a la calidad del núcleo de Linux. Muchos programadores con talento han contribuido a la versión actual, dando lugar a un producto técnicamente impresionante. Más aún, Linux es muy modular y fácilmente configurable. Resulta óptimo para incrementar el rendimiento de una variedad de plataformas hardware. Además, con el código fuente disponible, los distribuidores pueden adaptar las aplicaciones y facilidades para cumplir unos requisitos específicos. A lo largo de este libro, se proporcionarán detalles internos del núcleo de Linux.

ESTRUCTURA MODULAR

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

Aunque Linux no utiliza una técnica de micronúcleo, logra muchas de las ventajas potenciales de esta técnica por medio de su arquitectura modular particular. Linux está estructurado como una colección de módulos, algunos de los cuales pueden cargarse y descargarse automáticamente bajo demanda. Estos bloques relativamente independientes se denominan **módulos cargables** [GOYE99]. Esencialmente, un módulo es un fichero cuyo código puede enlazarse y desenlazarse con el núcleo en tiempo real. Normalmente, un módulo implementa algunas funciones específicas, como un sistema de ficheros, un controlador de dispositivo o algunas características de la capa superior del núcleo. Un módulo no se ejecuta como su propio proceso o hilo, aunque puede crear los hilos del núcleo que necesite por varios propósitos. En su lugar, un módulo se ejecuta en modo núcleo en nombre del proceso actual.

Por tanto, aunque Linux se puede considerar monolítico, su estructura modular elimina algunas de las dificultades para desarrollar y evolucionar el núcleo.

Los módulos cargables de Linux tienen dos características importantes:

- **Enlace dinámico.** Un módulo de núcleo puede cargarse y enlazarse al núcleo mientras el núcleo está en memoria y ejecutándose. Un módulo también puede desenlazarse y eliminarse de la memoria en cualquier momento.

- **Módulos apilables.** Los módulos se gestionan como una jerarquía. Los módulos individuales sirven como bibliotecas cuando los módulos cliente los referencian desde la parte superior de la jerarquía, y actúan como clientes cuando referencian a módulos de la parte inferior de la jerarquía.

El enlace dinámico [FRAN97] facilita la configuración y reduce el uso de la memoria del núcleo. En Linux, un programa de usuario o un usuario puede cargar y descargar explícitamente módulos del núcleo utilizando los mandatos *insmod* y *rmmod*. El núcleo mismo detecta la necesidad de funciones particulares y puede cargar y descargar módulos cuando lo necesite. Con módulos apilables, se pueden definir dependencias entre los módulos. Esto tiene dos ventajas:

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

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

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

La tabla de símbolos define aquellos símbolos controlados por este módulo que se utilizan en otros sitios.

La Figura 2.17 muestra que el módulo VFAT se carga después del módulo FAT y que el módulo VFAT es dependiente del módulo FAT.

COMPONENTES DEL NÚCLEO

La Figura 2.18, tomada de [MOSB02] muestra los principales componentes del núcleo Linux tal y como están implementados en una arquitectura IA-64 (por ejemplo, Intel Itanium). La figura muestra

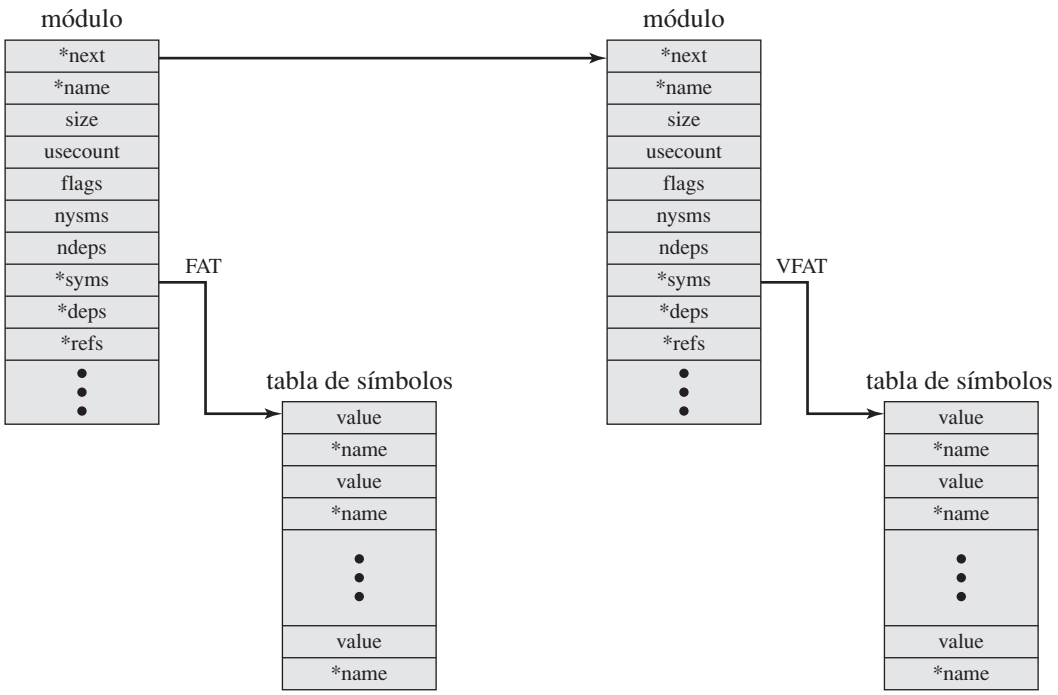


Figura 2.17. Lista ejemplo de módulos de núcleo de Linux.

varios procesos ejecutando encima del núcleo. Cada caja indica un proceso separado, mientras que cada línea curvada con una cabeza de flecha representa un hilo de ejecución*.

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

Brevemente, los principales componentes del núcleo son los siguientes:

- **Señales.** El núcleo utiliza las señales para llamar a un proceso. Por ejemplo, las señales se utilizan para notificar ciertos fallos a un proceso como por ejemplo, la división por cero. La Tabla 2.6 da unos pocos ejemplos de señales.
- **Llamadas al sistema.** La llamada al sistema es la forma en la cual un proceso requiere un servicio de núcleo específico. Hay varios cientos de llamadas al sistema, que pueden agruparse básicamente en seis categorías: sistema de ficheros, proceso, planificación, comunicación entre procesos, *socket* (red) y misceláneos. La Tabla 2.7 define unos pocos ejemplos de cada categoría.

En Linux, no hay distinción entre los conceptos de proceso e hilo. Sin embargo, múltiples hilos en Linux se pueden agrupar de tal forma que, efectivamente, pueda existir un único proceso compuesto por múltiples hilos. Estos aspectos se discuten en el Capítulo 4.

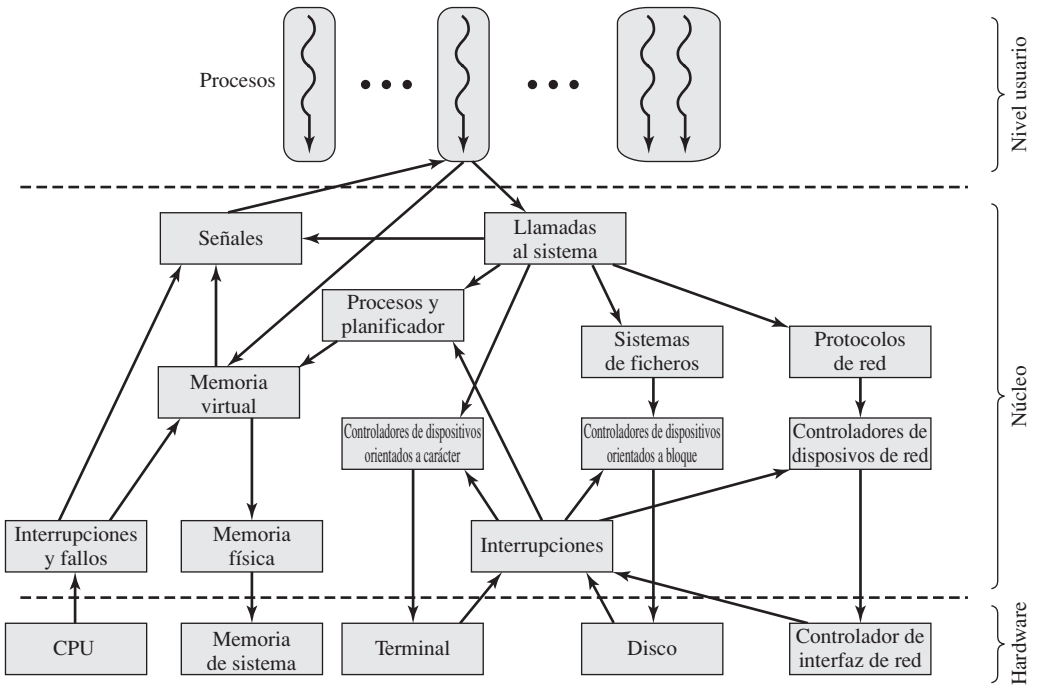


Figura 2.18. Componentes del núcleo de Linux.

- **Procesos y planificador.** Crea, gestiona y planifica procesos.
- **Memoria virtual.** Asigna y gestiona la memoria virtual para los procesos.
- **Sistemas de ficheros.** Proporciona un espacio de nombres global y jerárquico para los ficheros, directorios y otros objetos relacionados con los ficheros. Además, proporciona las funciones del sistema de ficheros.
- **Protocolos de red.** Da soporte a la interfaz *Socket* para los usuarios, utilizando la pila de protocolos TCP/IP.
- **Controladores de dispositivo tipo carácter.** Gestiona los dispositivos que requiere el núcleo para enviar o recibir datos un byte cada vez, como los terminales, los módems y las impresoras.
- **Controladores de dispositivo tipo bloque.** Gestiona dispositivos que leen y escriben datos en bloques, tal como varias formas de memoria secundaria (discos magnéticos, CDROM, etc.).
- **Controladores de dispositivo de red.** Gestiona las tarjetas de interfaz de red y los puertos de comunicación que permiten las conexiones a la red, tal como los puentes y los encaminadores.
- ***Traps* y fallos.** Gestiona los *traps* y fallos generados por la CPU, como los fallos de memoria.
- **Memoria física.** Gestiona el conjunto de marcos de páginas de memoria real y asigna las páginas de memoria virtual.
- **Interrupciones.** Gestiona las interrupciones de los dispositivos periféricos.

Tabla 2.6. Algunas señales de Linux.

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

Tabla 2.7. Algunas llamadas al sistema de Linux.

	Relacionadas con el sistema de ficheros
close	Cierra un descriptor de fichero.
link	Construye un nuevo nombre para un fichero.
open	Abre y posiblemente crea un fichero o dispositivo.
read	Lee un descriptor de fichero.
write	Escribe a través de un descriptor de fichero.
	Relacionadas con los procesos
execve	Ejecuta un programa.
exit	Termina el proceso que lo invoca.
getpid	Obtiene la identificación del proceso.
setuid	Establece la identidad del usuario del proceso actual.
prtrace	Proporciona una forma por la cual un proceso padre puede observar y controlar la ejecución de otro proceso, examinar y cambiar su imagen de memoria y los registros.
	Relacionadas con la planificación
sched_getparam	Establece los parámetros de planificación asociados con la política de planificación para el proceso identificado por su *pid*.
sched_get_priority_max	Devuelve el valor máximo de prioridad que se puede utilizar con el algoritmo de planificación identificado por la política.
sched_setscheduler	Establece tanto la política de planificación (por ejemplo, FIFO) como los parámetros asociados al *pid* del proceso.
sched_rr_get_interval	Escribe en la estructura *timespec* apuntada por el parámetro *tp* el cuanto de tiempo *round robin* para el proceso *pid*.
sched_yield	Un proceso puede abandonar el procesador voluntariamente sin necesidad de bloquearse a través de una llamada al sistema. El proceso entonces se moverá al final de la cola por su prioridad estática y un nuevo proceso se pondrá en ejecución.

	Relacionadas con la comunicación entre procesos (IPC)
msgrcv	Se asigna una estructura de *buffer* de mensajes para recibir un mensaje. Entonces, la llamada al sistema lee un mensaje de la cola de mensajes especificada por *msqid* en el *buffer* de mensajes nuevamente creado.
semctl	Lleva a cabo la operación de control especificada por *cmd* en el conjunto de semáforos *semid*.
semop	Lleva a cabo operaciones en determinados miembros del conjunto de semáforos *semid*.
shmat	Adjunta el segmento de memoria compartido identificado por *shmid* al segmento de datos del proceso que lo invoca.
shmctl	Permite al usuario recibir información sobre un segmento de memoria compartido, establecer el propietario, grupo y permisos de un segmento de memoria compartido o destruir un segmento.
	Relacionadas con los *sockets* (red)
bind	Asigna la dirección IP local y puerto para un *socket*. Devuelve 0 en caso de éxito y -1 en caso de error.
connect	Establece una conexión entre el *socket* dado y el *socket* asociado remoto con *sockaddr*.
gethostname	Devuelve el nombre de máquina local.
send	Envia los bytes que tiene el *buffer* apuntado por **msg* sobre el *socket* dado.
setsockopt	Envia las opciones sobre un *socket*.
	Misceláneos
create_module	Intenta crear una entrada del módulo cargable y reservar la memoria de núcleo que será necesario para contener el módulo.
fsync	Copia todas las partes en memoria de un fichero a un disco y espera hasta que el dispositivo informa que todas las partes están en almacenamiento estable.
query_module	Solicita información relacionada con los módulos cargables desde el núcleo.
time	Devuelve el tiempo en segundos desde 1 de enero de 1970.
vhangup	Simula la suspensión del terminal actual. Esta llamada sirve para que otros usuarios puedan tener un terminal «limpio» en tiempo de inicio.

2.9. LECTURAS Y SITIOS WEB RECOMENDADOS

Como en el área de arquitectura de computadores, existen muchos libros de sistemas operativos. [SILB04], [NUTT04] y [TANE01] cubren los principios básicos usando diversos sistemas operativos importantes como casos de estudio. [BRIN01] es una colección excelente de artículos que cubren los principales avances del diseño de los sistemas operativos a lo largo de los años.

Un tratamiento excelente de los aspectos internos de UNIX, que proporciona un análisis comparativo de un gran número de variantes, es [VAHA96]. Para UNIX SVR4, [GOOD94] proporciona un tratamiento definitivo, con amplios detalles técnicos. Para el sistema académicamente popular Berkeley UNIX 4.4BSD, [MCKU96] es altamente recomendado. [MAUR01] proporciona un buen trata-