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

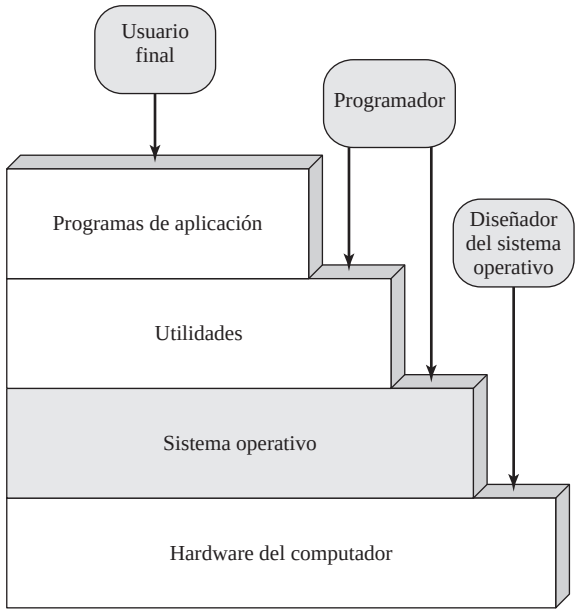


Figura 2.1. Capas y vistas de un sistema de computación.

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.

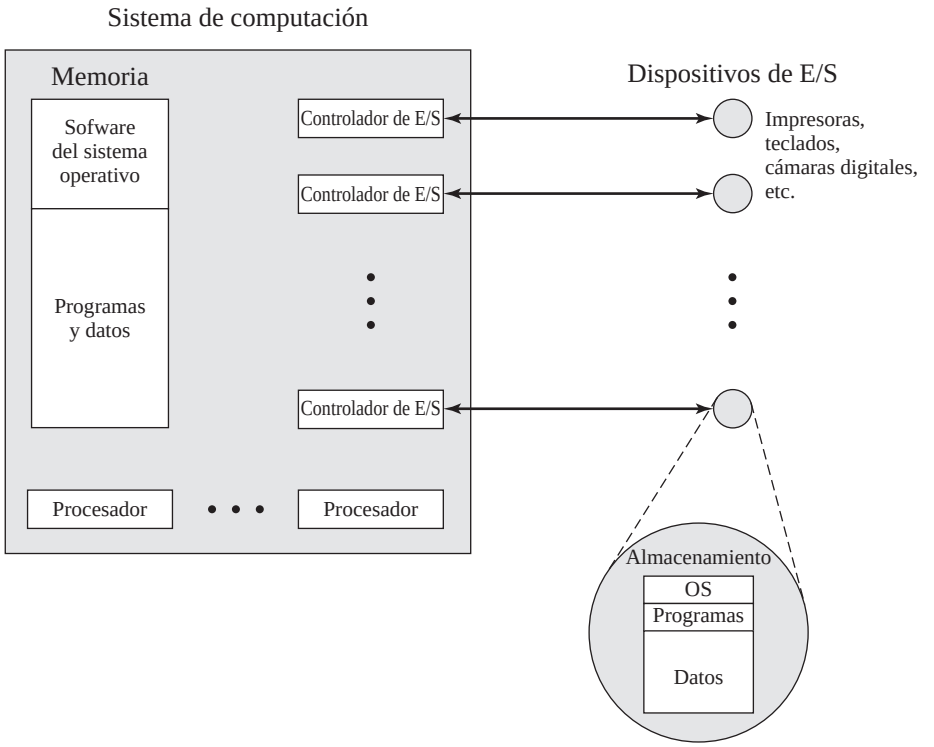


Figura 2.2. El sistema operativo como gestor de recursos.

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.

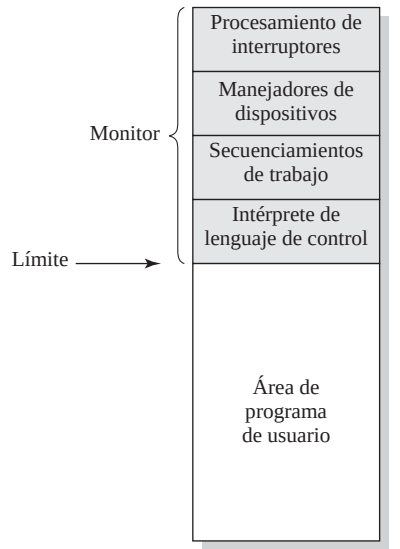


Figura 2.3. Disposición de memoria de un monitor residente.

- **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.

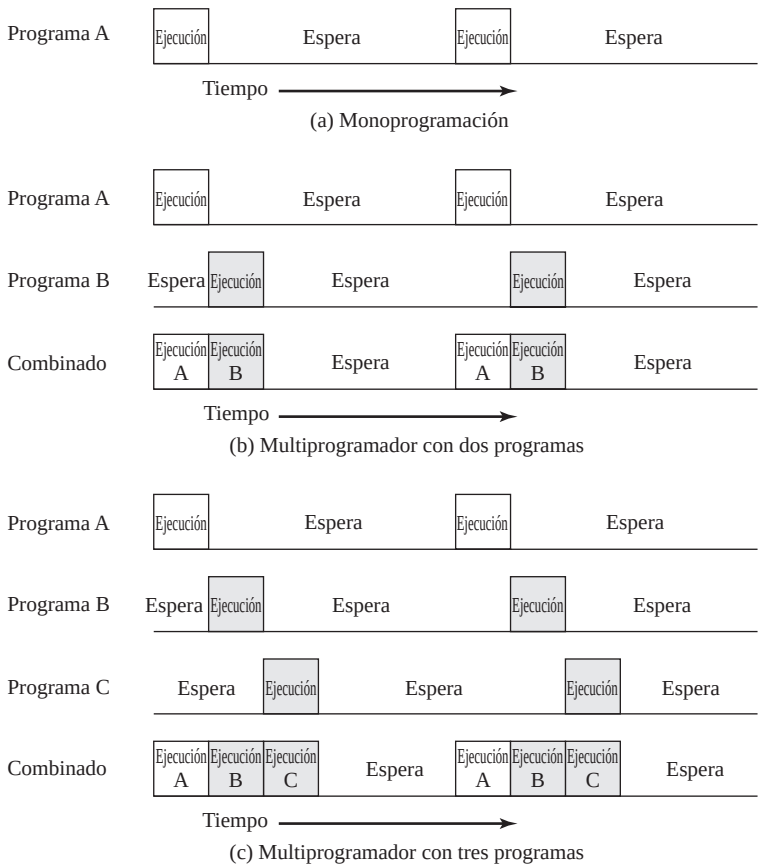


Figura 2.5. Ejemplo de multiprogramación.

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.

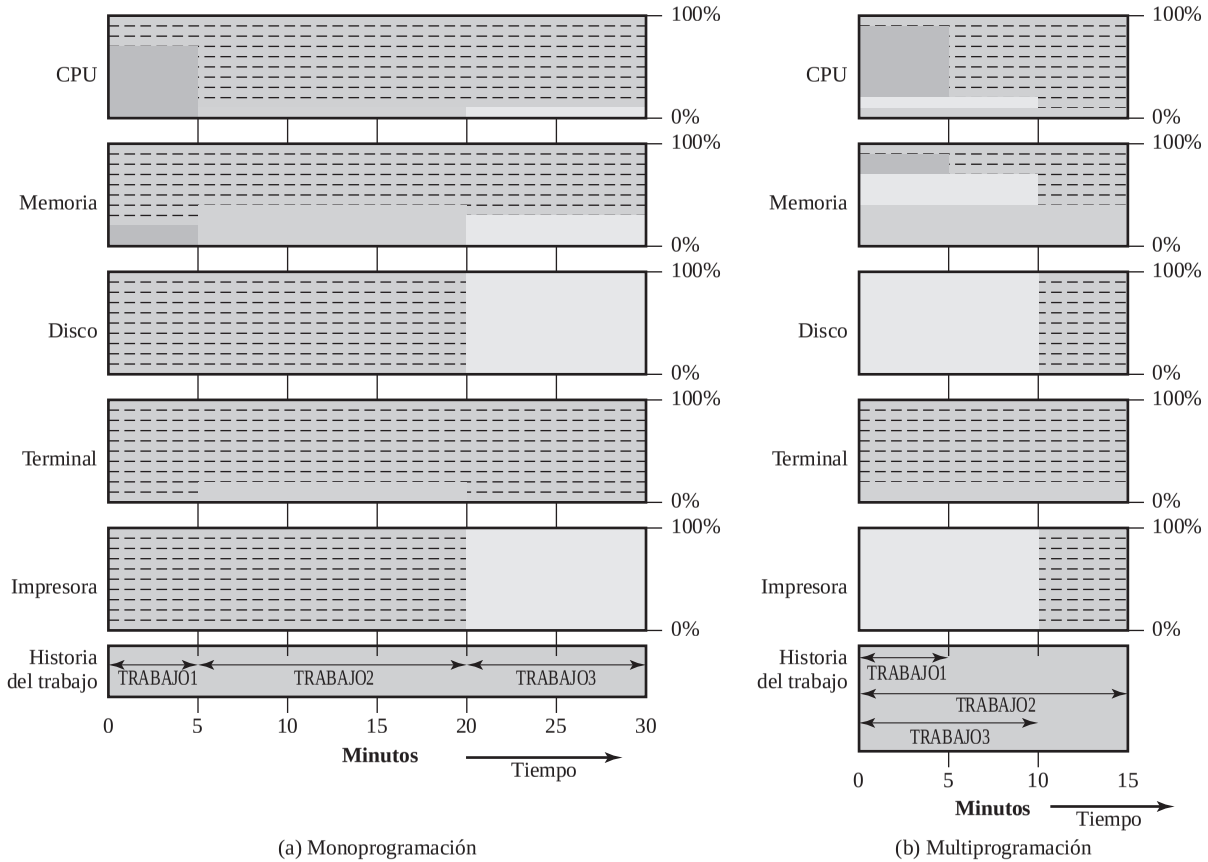


Figura 2.6. Histogramas de utilización.

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).

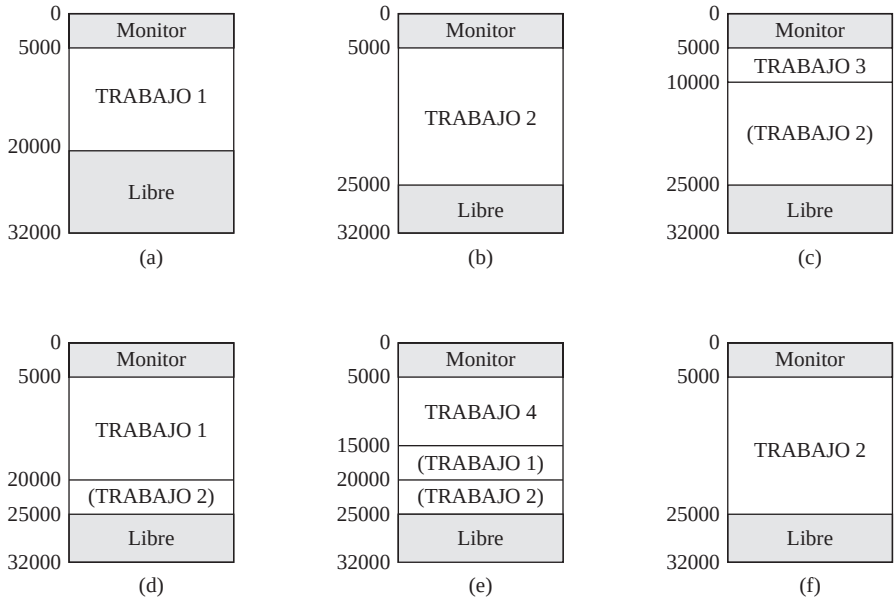


Figura 2.7. Operación del CTSS.

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-

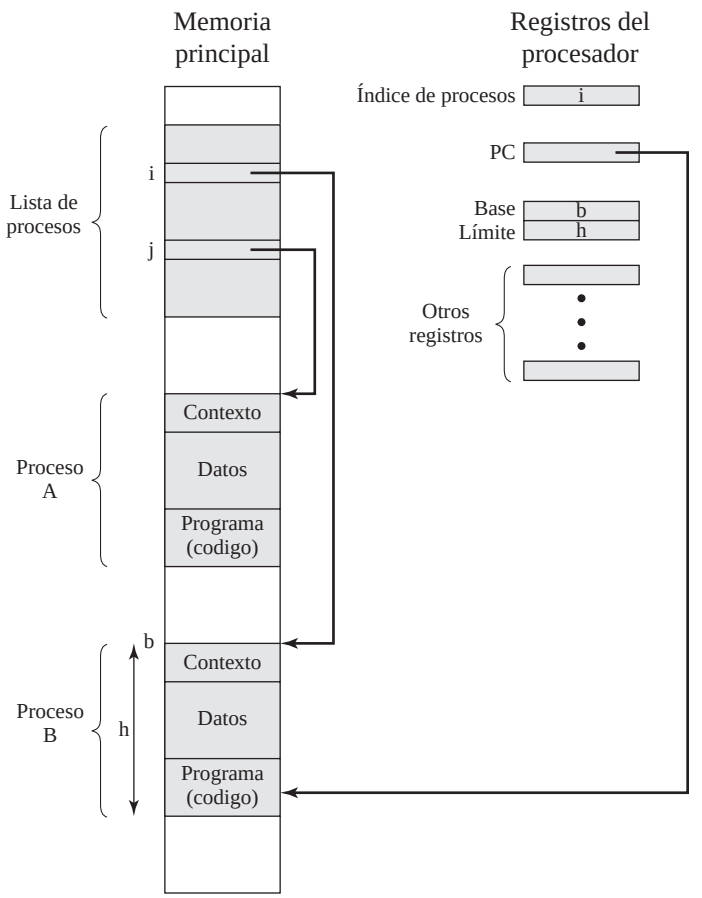


Figura 2.8. Implementación de procesos típica.

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

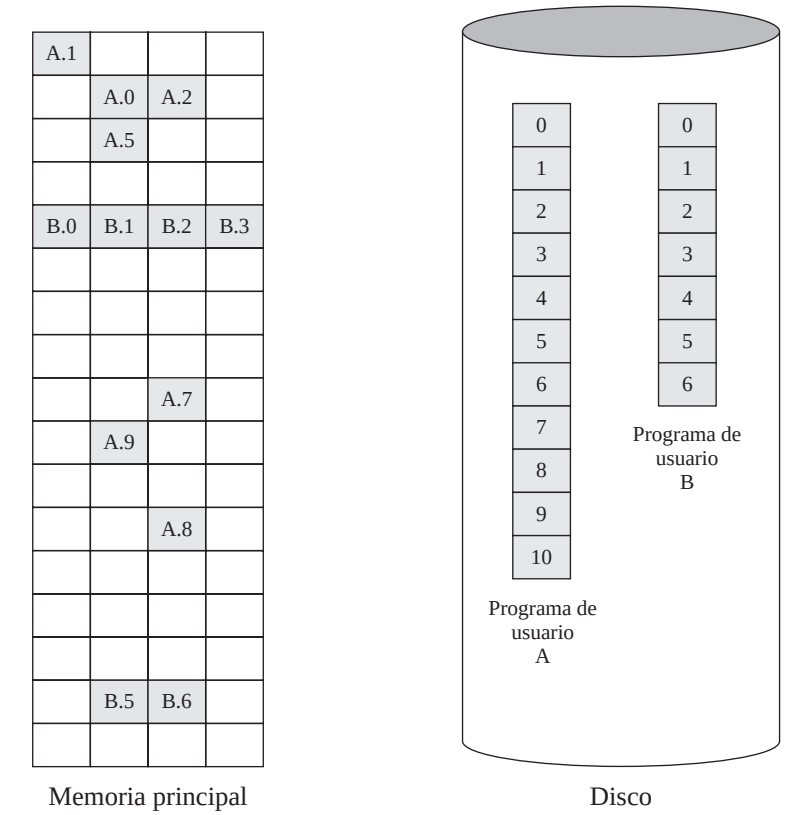


Figura 2.9. Conceptos de memoria virtual.

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.

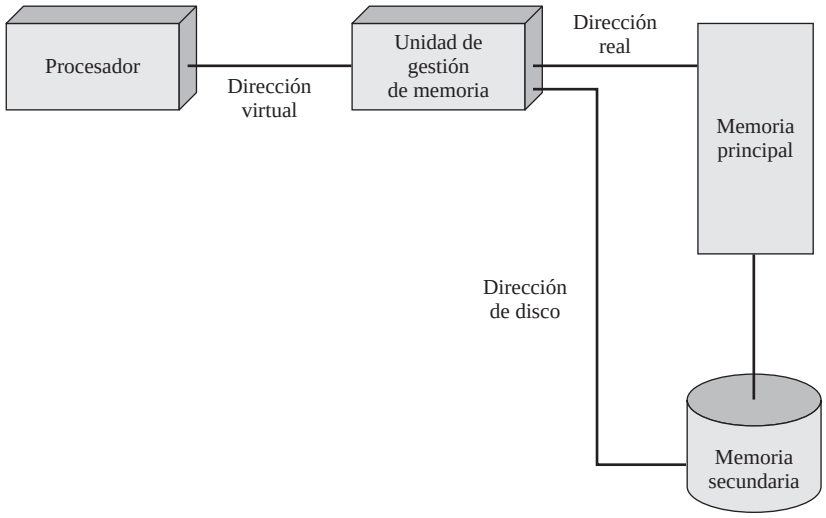


Figura 2.10. Direccionamiento de memoria virtual.

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.

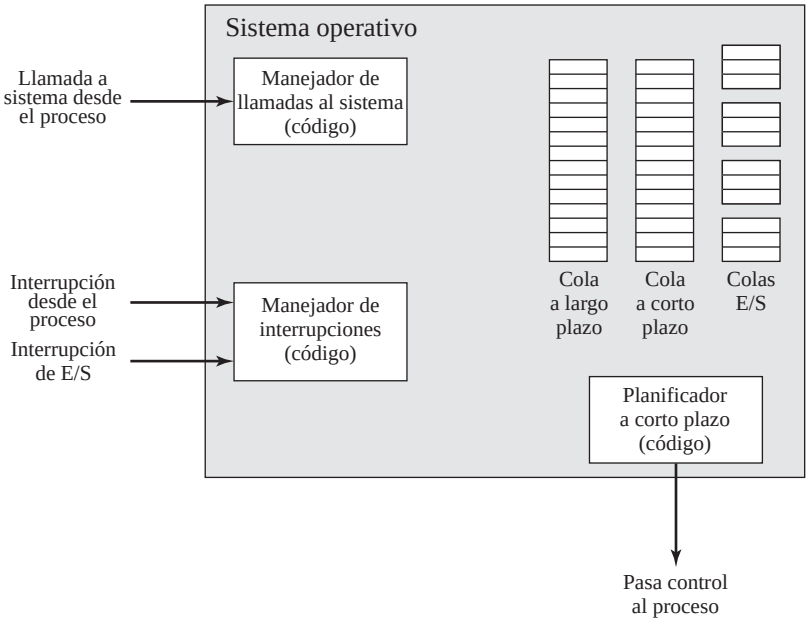


Figura 2.11. Elementos clave de un sistema operativo para la multiprogramación.

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.

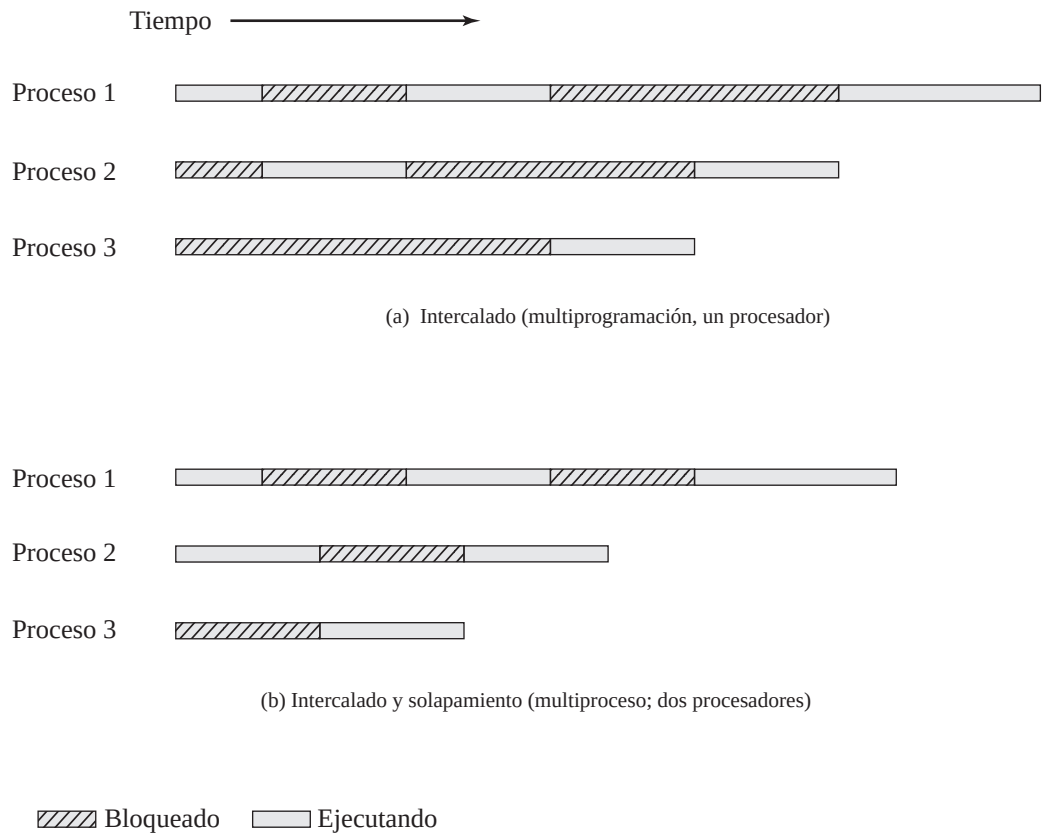


Figura 2.12. Multiprogramación y multiproceso.

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.

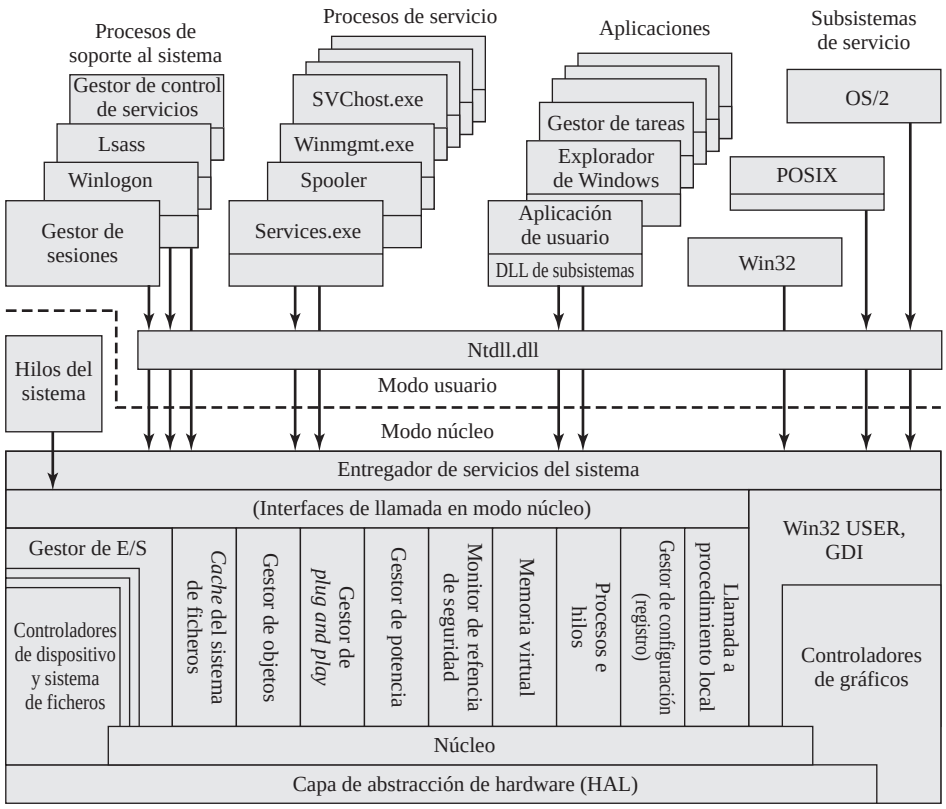
Una segunda motivación para la multitarea es el aumento de la computación cliente/servidor. En este paradigma, un computador personal o estación de trabajo (cliente) y un sistema *host* (servidor) se utilizan de forma conjunta para llevar a cabo una aplicación particular. Los dos están enlazados, y a 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-

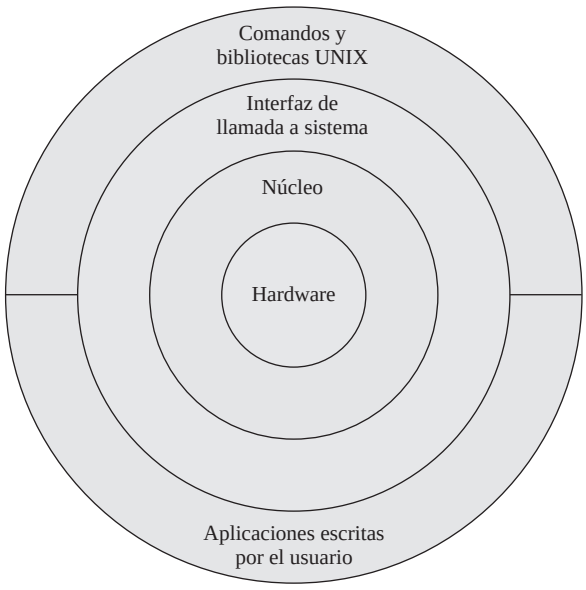


Figura 2.14. Arquitectura general de UNIX.

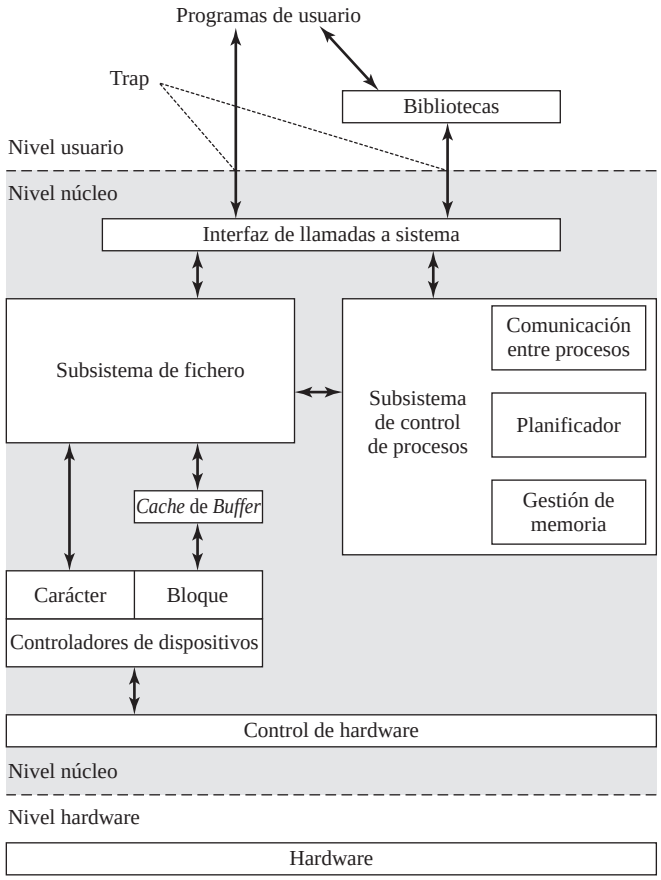


Figura 2.15. Núcleo tradicional de UNIX [BACH86].

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

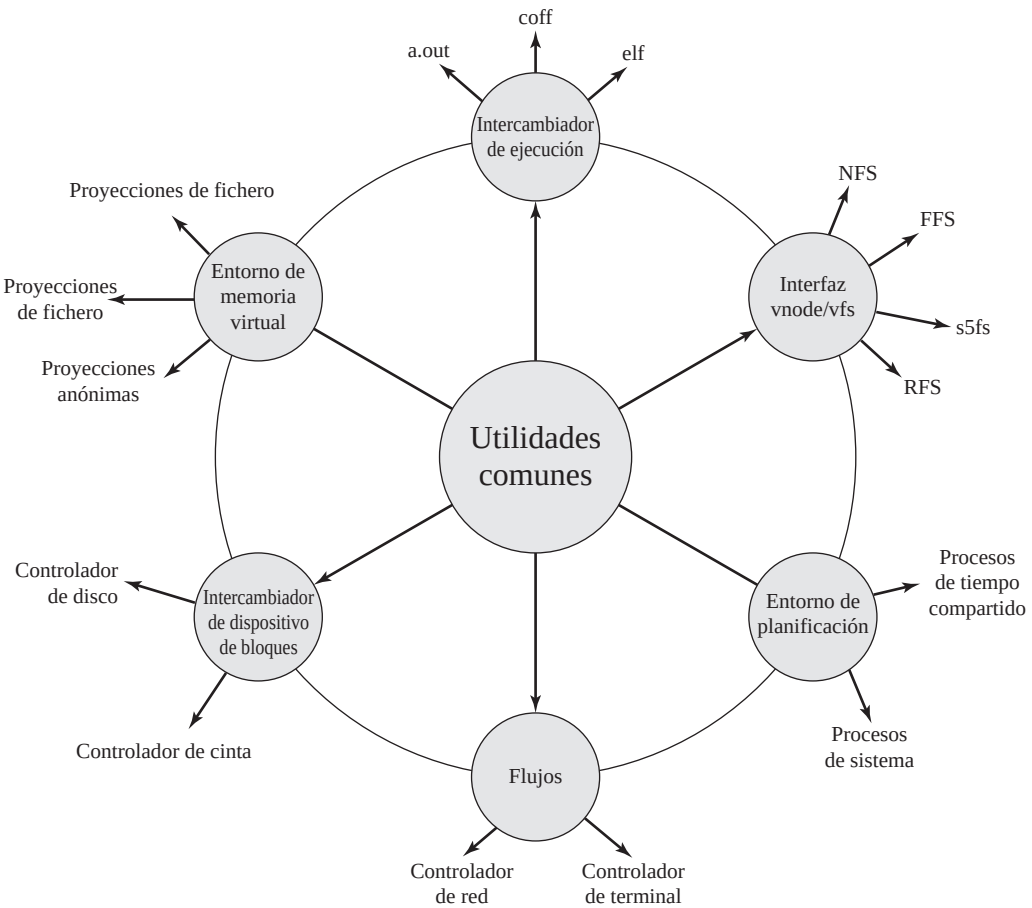


Figura 2.16. Núcleo UNIX moderno [VAHA96].

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

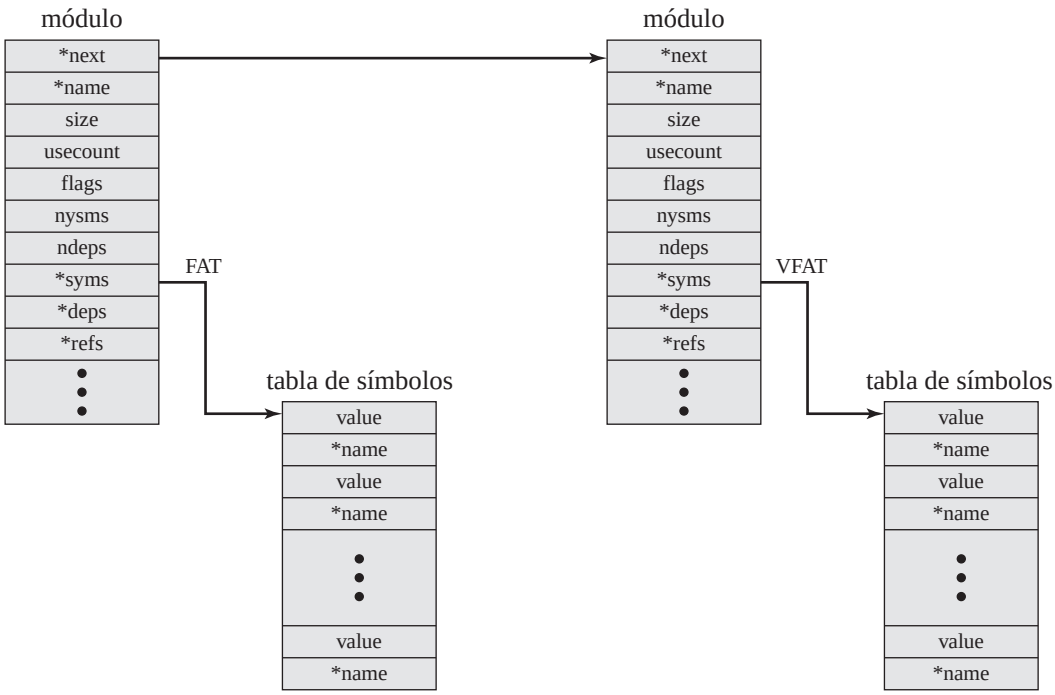


Figura 2.17. Lista ejemplo de módulos de núcleo de Linux.

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.

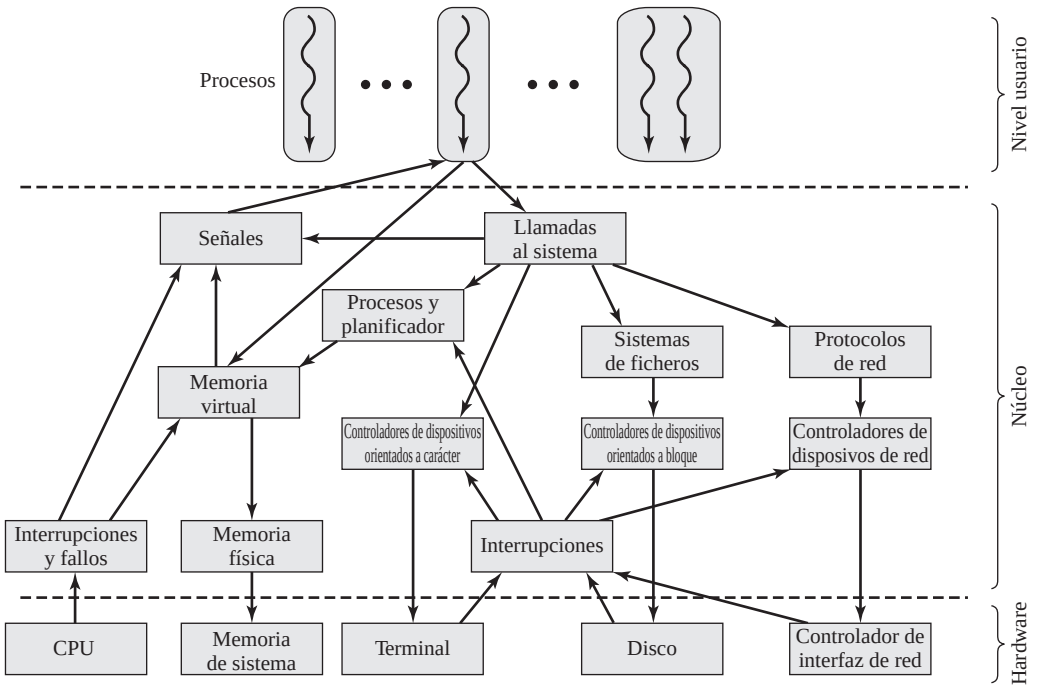


Figura 2.18. Componentes del núcleo de Linux.

- **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-

miento de los aspectos internos de Solaris. Dos buenos tratamientos de los aspectos internos de Linux se recogen en [BOVE03] y [BAR00].

Aunque hay incontables libros sobre varias versiones de Windows, hay curiosamente poco material disponible sobre los aspectos internos de Windows. [SOLO00] proporciona un tratamiento excelente de los aspectos internos de Windows 2000 y gran parte de este material es válido para posteriores versiones. [BOSW03] cubre parte de los aspectos internos de Windows 2003.

BAR00 Bar, M. *Linux Internals*. New York, McGraw-Hill, 2000.

BOSW03 Boswell, W. *Inside Windows Server 2003*. Reading, MA:Addison-Wesley, 2003.

BOVE03 Bovet, D., y Cesati, M. *Understanding the Linux Kernel*. Sebastopol, CA: O'Reilly, 2003.

BRIN01 Brinch Hansen, P. *Classic Operating Systems: From Batch Processing to Distributed Systems*. New York: Springer-Verlag, 2001.

GOOD94 Goodheart, B., y Cox, J. *The Magic Garden Explained: The Internals of UNIX System V Release 4*. Englewood Cliffs, NJ: Prentice Hall, 1994.

MAUR01 Mauro, J., y McDougall, R. *Solaris Internals: Core Kernel Architecture*. Palo Alto, CA: Sun Microsystems Press, 2001.

MCKU96 McKusick, M.; Bostic, K.; Karels, M.; y Quartermain, J. *The Design and Implementation of the 4.4BSD UNIX Operating System*. Reading, MA: Addison-Wesley, 1996.

NUTT04 Nutt,G. *Operating System*. Reading, MA: Addison-Wesley, 2004.

SILB04 Silberschatz, A.; Galvin, P.; y Gagne,G. *Operating System Concepts with Java*. Reading, MA: Addison-Wesley, 2004.

SOLO00 Solomon, D. *Inside Microsoft Windows 2000*. Redmond, WA: Microsoft Press, 2000.

TANE01 Tanenbaum, A. *Modern Operating Systems*. Upper Saddle River, NJ: Prentice Hall, 2001.

VAHA96 Vahalia, U. *UNIX Internals: The New Frontiers*. Upper Saddle River, NJ: Prentice Hall, 1996.

SITIOS WEB RECOMENDADOS

- **El centro de recursos del sistema operativo.** Una colección útil de documentos y artículos sobre un amplio rango de temas de sistemas operativos.
- **Revisión de los sistemas operativos.** Una extensa revisión de sistemas operativos comerciales, libres, de investigación y para aficionados.
- **Comparación técnica de los sistemas operativos.** Incluye una cantidad sustancial de información sobre una variedad de sistemas operativos.
- **Grupo especial de interés ACM sobre los sistemas operativos.** Información sobre publicaciones y conferencias SIGOPS.
- **Comité Técnico IEEE sobre los sistemas operativos y los entornos de aplicación.** Incluye un boletín en línea y enlaces a otros sitios.
- **La FAQ comp.os.research.** FAQ amplio y válido que cubre aspectos de diseño de los sistemas operativos.

- **Universo Guru UNIX.** Fuente excelente de información sobre UNIX.
- **Proyecto de documentación Linux.** El nombre describe el sitio.

2.10. TÉRMINOS CLAVE, CUESTIONES DE REPASO Y PROBLEMAS

TÉRMINOS CLAVE

Contexto de ejecución	Monitor	Procesamiento serie
Dirección física	Monitor residente	Proceso
Dirección real	Monoprogramación	*Round-robin* o turno rotatorio
Dirección virtual	Multihilo	Sistema *batch* o en lotes
Estado del proceso	Multiprocesamiento simétrico	Sistema *batch* o en lotes multiprogramado
Gestión de memoria	Multiprogramación	Sistema de tiempo compartido
Hilo	Multitarea	Sistema operativo
Interrupción	Núcleo	Trabajo
Instrucción privilegiada	Núcleo monolítico	Tarea
Lenguaje de control de trabajos	Planificación	Tiempo compartido
Micronúcleo	Procesamiento *batch* o en lotes	

CUESTIONES DE REPASO

2.1. ¿Cuáles son los tres objetivos de diseño de un sistema operativo?

2.2. ¿Qué es el núcleo de un sistema operativo?

2.3. ¿Qué es multiprogramación?

2.4. ¿Qué es un proceso?

2.5. ¿Cómo utiliza el sistema operativo el contexto de ejecución de un proceso?

2.6. Liste y explique brevemente cinco responsabilidades relacionadas con la gestión de almacenamiento de un sistema operativo típico.

2.7. Explique la distinción entre una dirección real y una dirección virtual.

2.8. Describa la técnica de planificación *round-robin* o turno rotatorio.

2.9. Explique la diferencia entre un núcleo monolítico y un micronúcleo.

2.10. ¿En qué consiste el uso de multihilos o *multithreading*?

PROBLEMAS

2.1. Supóngase que se tiene un computador multiprogramado en el cual cada trabajo tiene características idénticas. En un periodo de computación *T*, un trabajo gasta la mitad del tiempo en E/S y la otra mitad en actividad del procesador. Cada trabajo ejecuta un total de *N*

periodos. Asúmase que se utiliza una planificación *round-robin* simple, y que las operaciones de E/S se pueden solapar con las operaciones del procesador. Defina las siguientes cantidades:

- Tiempo de servicio = Tiempo real para completar un trabajo.
- Productividad = Número medio de trabajos completados en el periodo *T*.
- Utilización del procesador = Porcentaje de tiempo que el procesador está activo (no esperando).

Calcule estas cantidades para uno, dos y cuatro trabajos simultáneos, asumiendo que el periodo *T* se distribuye de las siguientes formas:

a) Primera mitad E/S, segunda mitad procesador.

b) Primero y cuarto cuartos E/S, segundo y tercer cuartos procesador.

2.2. Un programa limitado por la E/S es aquel que, si ejecuta solo, gasta más tiempo esperando operaciones de E/S que utilizando el procesador. Un programa limitado por el procesador es lo contrario. Supóngase un algoritmo de planificación a corto plazo que favorece a aquellos programas que han utilizado poco tiempo de procesador en el pasado reciente. Explique por qué este algoritmo favorece a los programas limitados por la E/S y no niega permanentemente el tiempo de procesador a los programas limitados por el procesador.

2.3. Contraste las políticas de planificación que se podrían utilizar cuando se intenta optimizar un sistema de tiempo compartido y aquéllas que se utilizan para optimizar un sistema en lotes multiprogramado.

2.4. ¿Cuál es el propósito de las llamadas al sistema y cómo se relacionan las llamadas al sistema con el sistema operativo y el concepto de modo dual (modo núcleo y modo usuario)?

2.5. En el sistema operativo de IBM, OS/390, uno de los módulos principales en el núcleo es el gestor de recursos del sistema (SRM: *System Resource Manager*). Este módulo es responsable de la asignación de recursos entre los espacios de direcciones (procesos). El SRM da a OS/390 un grado de sofisticación único entre los sistemas operativos. Ningún otro sistema operativo de *mainframe*, y ciertamente ningún otro tipo de sistema operativo, puede realizar las funciones llevadas a cabo por SRM. El concepto de recurso incluye al procesador, memoria real y canales de E/S. SRM acumula estadísticas pertenecientes a la utilización del procesador, canales y varias estructuras de datos clave. Su propósito es proporcionar un rendimiento óptimo basado en monitorización y análisis del rendimiento. La instalación proporciona varios objetivos de rendimiento y éstas sirven como guía al SRM, que modifica dinámicamente la instalación y las características de rendimiento de los trabajos basándose en la utilización del sistema. Adicionalmente, SRM proporciona informes que permiten al operador entrenado refinar las configuraciones y el establecimiento de parámetros para mejorar el servicio de usuario.

Este problema es un ejemplo de actividad SRM. La memoria real se divide en bloques de igual tamaño llamados marcos, de los cuales podría haber muchos miles. Cada marco puede contener un bloque de memoria virtual, conocido como página. SRM recibe el control aproximadamente 20 veces cada segundo, inspeccionando cada marco de página. Si la página no se ha referenciado o cambiado, un contador se incrementa por 1. A lo largo del tiempo, SRM hace la media de estos números para determinar el número de segundos medio que un marco de página en el sistema queda inalterable. ¿Cuál podría ser el propósito de esto y qué acción podría realizar SRM?

PARTE II

PROCESOS

La tarea fundamental de cualquier sistema operativo moderno es la gestión de procesos. El sistema operativo debe reservar recursos para los procesos, permitir a los mismos compartir e intercambiar información, proteger los recursos de cada uno de ellos del resto, y permitir la sincronización entre procesos. Para conseguir alcanzar estos requisitos, el sistema operativo debe mantener una estructura determinada para cada proceso que describa el estado y la propiedad de los recursos y que permite al sistema operativo establecer el control sobre los procesos.

En un monoprocesador multiprogramado, la ejecución de varios procesos se puede intercalar en el tiempo. En un multiprocesador, no sólo se intercala la ejecución de procesos, también es posible que haya múltiples procesos que se ejecuten de forma simultánea. Tanto la ejecución intercalada como de forma simultánea son tipos de concurrencia y llevan a que el sistema se enfrente a diferentes problemas, tanto en el ámbito de las aplicaciones de programador como en el de los sistemas operativos.

En muchos sistemas operativos actuales, la problemática de la gestión de procesos se encuentra ampliada por la introducción del concepto de hilo (*thread*). En un sistema multihilo, el concepto de proceso mantiene los atributos de la propiedad de recursos, mientras que los aspectos de la ejecución de múltiples flujos de instrucciones se encuentra relacionada con los hilos que ejecutan dentro de ese proceso.

ÍNDICE PARA LA PARTE DOS

CAPÍTULO 4. HILOS, SMP, Y MICRONÚCLEOS

El Capítulo 4 cubre tres áreas características de los sistemas operativos contemporáneos que representan unos avances sobre el diseño de los sistemas operativos tradicionales. En muchos sistemas operativos, el concepto tradicional de proceso se ha dividido en dos partes: una de ellas que trata de la propiedad de los recursos (proceso) y otra que trata de la ejecución del flujo de instrucciones (hilo o *thread*). Un único proceso puede contener múltiples hilos. La organización multihilo proporciona ventajas en la estructuración de las aplicaciones y en su rendimiento. El Capítulo 4 también examina los multiprocesadores simétricos (SMP), que son sistemas que tienen múltiples procesadores, cada uno de los cuales es capaz de ejecutar cualquier aplicación o el código de sistema. La organización SMP mejora el rendimiento y la fiabilidad. SMP a menudo se usa en conjunto con la programación multihilo, pero aún sin ella proporciona unas importantes mejoras a nivel de rendimiento. Para finalizar el Capítulo 4 examina el concepto de micronúcleo, que es un estilo de diseño de sistemas operativos que minimiza la cantidad de código de sistema que se ejecuta en modo núcleo. Las ventajas de esta estrategia también se analizan.

CAPÍTULO 5. CONCURRENCIA. EXCLUSIÓN MUTUA Y SINCRONIZACIÓN

Dos temas centrales en los sistemas operativos modernos son las multiprogramación y el procesamiento distribuido. El concepto de concurrencia es fundamental para ambos, y fundamental también para la tecnología de diseño de los sistemas operativos. El Capítulo 5 repasa dos aspectos del control de la concurrencia: la exclusión mutua y la sincronización. La exclusión mutua se refiere a la posibilidad de que múltiples procesos (o hilos) compartan código, recursos, o datos de forma de que sólo uno de ellos tenga acceso al objeto compartido en cada momento. La sincronización se encuentra relacionada con la exclusión mutua: es la posibilidad de que múltiples procesos coordinen sus actividades para intercambiar información. El Capítulo 5 proporciona un amplio tratamiento de los aspectos relativos a la concurrencia, comenzando por un repaso de las consideraciones de diseño implicadas. El capítulo proporciona una revisión del soporte hardware para la concurrencia presentando los mecanismos más importantes para darle soporte: semáforos, monitores, y paso de mensajes.

CAPÍTULO 6. CONCURRENCIA. INTERBLOQUEO E INANICIÓN

El Capítulo 6 muestra dos aspectos más de la concurrencia. Un *interbloqueo* es una situación en la cual dos o más procesos están esperando a que otros miembros del conjunto completen una operación para poder continuar, pero ninguno de los miembros es capaz de hacerlo. Los interbloqueos son un fenómeno difícil de anticipar, y no hay soluciones generales fáciles para éstos. El Capítulo 6 muestra las tres estrategias principales para manejar un interbloqueo: prevenirlo, evitarlo, y detectarlo. La *inanición* se refiere una situación en la cual un proceso se encuentra listo para ejecutar pero se le deniega el acceso al procesador de forma continuada en deferencia a otros procesos. En su mayor parte, la inanición se trata como una cuestión de planificación y por tanto la trataremos en la Parte Cuatro. Aunque el Capítulo 6 se centra en los interbloqueos, la inanición se trata dentro del contexto de las soluciones a los mismos necesarias para evitar el problema de la inanición.

CAPÍTULO 3

Descripción y control de procesos

El diseño de un sistema operativo debe reflejar ciertos requisitos generales. Todos los sistemas operativos multiprogramados, desde los sistemas operativos monousuario como Windows 98 hasta sistemas *mainframes* como IBM z/OS, que son capaces de dar soporte a miles de usuarios, se construyen en torno al concepto de proceso. La mayoría de los requisitos que un sistema operativo debe cumplir se pueden expresar con referencia a los procesos:

- El sistema operativo debe intercalar la ejecución de múltiples procesos, para maximizar la utilización del procesador mientras se proporciona un tiempo de respuesta razonable.
- El sistema operativo debe reservar recursos para los procesos conforme a una política específica (por ejemplo, ciertas funciones o aplicaciones son de mayor prioridad) mientras que al mismo tiempo evita interbloqueos[1].
- Un sistema operativo puede requerir dar soporte a la comunicación entre procesos y la creación de procesos, mediante las cuales ayuda a la estructuración de las aplicaciones.

Se comienza el estudio detallado de los sistemas operativos examinando la forma en la que éstos representan y controlan los procesos. Después de una introducción al concepto de proceso, el capítulo presentará los estados de los procesos, que caracterizan el comportamiento de los mismos. Seguidamente, se presentarán las estructuras de datos que el sistema operativo usa para gestionar los procesos. Éstas incluyen las estructuras para representar el estado de cada proceso así como para registrar características de los mismos que el sistema operativo necesita para alcanzar sus objetivos. Posteriormente, se verá cómo el sistema operativo utiliza estas estructuras para controlar la ejecución de los procesos. Por último, se discute la gestión de procesos en UNIX SVR4. El Capítulo 4 proporciona ejemplos más modernos de gestión de procesos, tales como Solaris, Windows, y Linux.

Nota: en este capítulo hay referencias puntuales a la memoria virtual. La mayoría de las veces podemos ignorar este concepto en relación con los procesos, pero en ciertos puntos de la discusión, las consideraciones sobre memoria virtual se hacen pertinentes. La memoria virtual no se discutirá en detalle hasta el Capítulo 8; ya se ha proporcionado una somera visión general en el Capítulo 2.

3.1. ¿QUÉ ES UN PROCESO?

CONCEPTOS PREVIOS

Antes de definir el término proceso, es útil recapitular algunos de los conceptos ya presentados en los Capítulos 1 y 2:

1. Una plataforma de computación consiste en una colección de recursos hardware, como procesador, memoria, módulos de E/S, relojes, unidades de disco y similares.
2. Las aplicaciones para computadores se desarrollan para realizar determinadas tareas. Suelen aceptar entradas del mundo exterior, realizar algún procesamiento y generar salidas.
3. No es eficiente que las aplicaciones estén escritas directamente para una plataforma hardware específica. Las principales razones son las siguientes:

---

[1] Los interbloqueos se examinarán en el Capítulo 6. Como ejemplo sencillo, un interbloqueo ocurre si dos procesos necesitan dos recursos iguales para continuar y cada uno de los procesos tiene la posesión de uno de los recursos. A menos que se realice alguna acción, cada proceso esperará indefinidamente por conseguir el otro recurso.

a) Numerosas aplicaciones pueden desarrollarse para la misma plataforma, de forma que tiene sentido desarrollar rutinas comunes para acceder a los recursos del computador.

b) El procesador por sí mismo proporciona únicamente soporte muy limitado para la multiprogramación. Es necesario disponer de software para gestionar la compartición del procesador así como otros recursos por parte de múltiples aplicaciones al mismo tiempo.

c) Cuando múltiples aplicaciones están activas al mismo tiempo es necesario proteger los datos, el uso de la E/S y los recursos propios de cada aplicación con respecto a las demás.

4. El sistema operativo se desarrolló para proporcionar una interfaz apropiada para las aplicaciones, rica en funcionalidades, segura y consistente. El sistema operativo es una capa de software entre las aplicaciones y el hardware del computador (Figura 2.1) que da soporte a aplicaciones y utilidades.
5. Se puede considerar que el sistema operativo proporciona una representación uniforme y abstracta de los recursos, que las aplicaciones pueden solicitar y acceder. Los recursos incluyen la memoria principal, las interfaces de red, los sistemas de ficheros, etc. Una vez que el sistema operativo ha creado estas abstracciones de los recursos para que las aplicaciones las usen, debe también controlar su uso. Por ejemplo, un sistema operativo podría permitir compartición y protección de recursos.

Ahora que se conocen los conceptos de aplicaciones, software de sistema y de recursos se está en disposición de hablar sobre cómo un sistema operativo puede, de forma ordenada, gestionar la ejecución de aplicaciones de forma que:

- Los recursos estén disponibles para múltiples aplicaciones.
- El procesador físico se conmute entre múltiples aplicaciones, de forma que todas lleguen a procesarse.
- El procesador y los dispositivos de E/S se puedan usar de forma eficiente.

El enfoque adoptado por todos los sistemas operativos modernos recae en un modelo bajo el cual la ejecución de una aplicación se corresponde con la existencia de uno o más procesos.

PROCESOS Y BLOQUES DE CONTROL DE PROCESOS

Se debe recordar que en el Capítulo 2 se sugirieron diversas definiciones del término *proceso*, incluyendo:

- Un programa en ejecución.
- Una instancia de un programa ejecutado en un computador.
- La entidad que se puede asignar y ejecutar en un procesador.
- Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.

También se puede pensar en un proceso como en una entidad que consiste en un número de elementos. Los dos elementos esenciales serían el **código de programa** (que puede compartirse con otros procesos que estén ejecutando el mismo programa) y un **conjunto de datos** asociados a dicho

código. Supongamos que el procesador comienza a ejecutar este código de programa, y que nos referiremos a esta entidad en ejecución como un proceso. En cualquier instante puntual del tiempo, *mientras el proceso está en ejecución*, este proceso se puede caracterizar por una serie de elementos, incluyendo los siguientes:

- **Identificador.** Un identificador único asociado a este proceso, para distinguirlo del resto de procesos.
- **Estado.** Si el proceso está actualmente corriendo, está en el estado *en ejecución*.
- **Prioridad**: Nivel de prioridad relativo al resto de procesos.
- **Contador de programa.** La dirección de la siguiente instrucción del programa que se ejecutará.
- **Punteros a memoria.** Incluye los punteros al código de programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otros procesos.
- **Datos de contexto.** Estos son datos que están presenten en los registros del procesador cuando el proceso está corriendo.
- **Información de estado de E/S.** Incluye las peticiones de E/S pendientes, dispositivos de E/S (por ejemplo, una unidad de cinta) asignados a dicho proceso, una lista de los ficheros en uso por el mismo, etc.
- **Información de auditoría.** Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.

La información de la lista anterior se almacena en una estructura de datos, que se suele llamar bloque de control de proceso (*process control block*) (Figura 3.1), que el sistema operativo crea y gestiona. El punto más significativo en relación al bloque de control de proceso, o BCP, es que contiene suficiente información de forma que es posible interrumpir el proceso cuando está corriendo y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna. El BCP es la herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación. Cuando un proceso se interrumpe, los valores actuales del contador de programa y los registros del procesador (datos de contexto) se guardan en los campos correspondientes del BCP y el estado del proceso se cambia a cualquier otro valor, como *bloqueado* o *listo* (descritos a continuación). El sistema operativo es libre ahora para poner otro proceso en estado de ejecución. El contador de programa y los datos de contexto se recuperan y cargan en los registros del procesador y este proceso comienza a correr.

De esta forma, se puede decir que un proceso está compuesto del código de programa y los datos asociados, además del bloque de control de proceso o BCP. Para un computador monoprocesador, en un instante determinado, como máximo un único proceso puede estar corriendo y dicho proceso estará en el estado *en ejecución*.

3.2. ESTADOS DE LOS PROCESOS

Como se acaba de comentar, para que un programa se ejecute, se debe crear un proceso o tarea para dicho programa. Desde el punto de vista del procesador, él ejecuta instrucciones de su repertorio de instrucciones en una secuencia dictada por el cambio de los valores del registro contador de programa. A lo largo del tiempo, el contador de programa puede apuntar al código de diferentes programas que son parte de diferentes procesos. Desde el punto de vista de un programa individual, su ejecución implica una secuencia de instrucciones dentro de dicho programa.

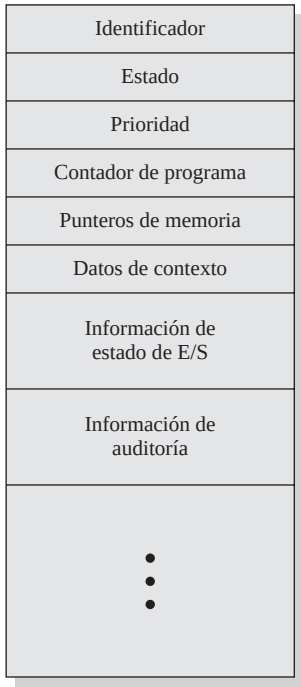


Figura 3.1. Bloque de control de programa (BCP) simplificado.

Se puede caracterizar el comportamiento de un determinado proceso, listando la secuencia de instrucciones que se ejecutan para dicho proceso. A esta lista se la denomina **traza** del proceso. Se puede caracterizar el comportamiento de un procesador mostrando cómo las trazas de varios procesos se entrelazan.

Considere un ejemplo. La Figura 3.2 muestra el despliegue en memoria de tres procesos. Para simplificar la exposición, se asume que dichos procesos no usan memoria virtual; por tanto, los tres procesos están representados por programas que residen en memoria principal. De manera adicional, existe un pequeño programa **activador** (*dispatcher*) que intercambia el procesador de un proceso a otro. La Figura 3.3 muestra las trazas de cada uno de los procesos en los primeros instantes de ejecución. Se muestran las 12 primeras instrucciones ejecutadas por los procesos A y C. El proceso B ejecuta 4 instrucciones y se asume que la cuarta instrucción invoca una operación de E/S, a la cual el proceso debe esperar.

Ahora vea estas trazas desde el punto de vista del procesador. La Figura 3.4 muestra las trazas entrelazadas resultante de los 52 primeros ciclos de ejecución (por conveniencia los ciclos de instrucciones han sido numerados). En este ejemplo, se asume que el sistema operativo sólo deja que un proceso continúe durante seis ciclos de instrucción, después de los cuales se interrumpe; lo cual previene que un solo proceso monopolice el uso del tiempo del procesador. Como muestra la Figura 3.4, las primeras seis instrucciones del proceso A se ejecutan seguidas de una alarma de temporización (*time-out*) y de la ejecución de cierto código del activador, que ejecuta seis instrucciones antes de devolver el control al proceso B[2]. Después de que se ejecuten cuatro instrucciones, el proceso B solicita una acción de

[2] El reducido número de instrucciones ejecutadas por los procesos y por el planificador es irreal; se ha utilizado para simplificar el ejemplo y clarificar las explicaciones.

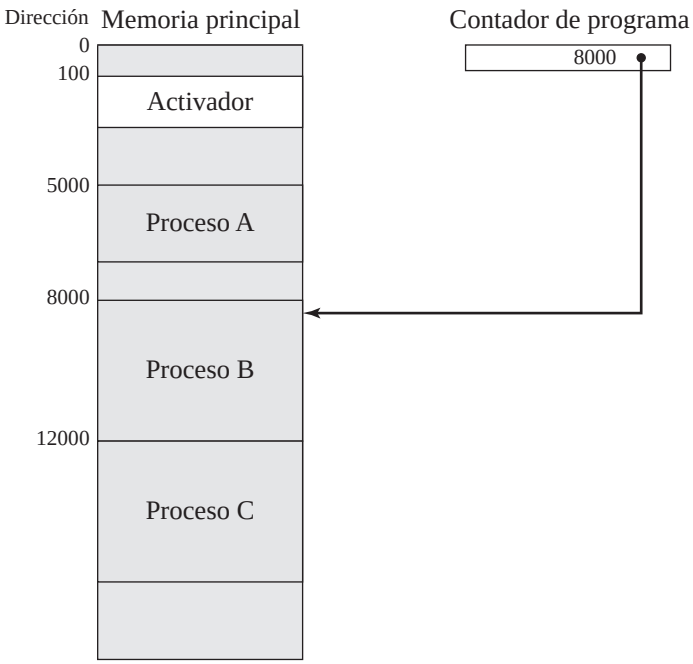


Figura 3.2. Instantánea de un ejemplo de ejecución (Figura 3.4) en el ciclo de instrucción 13.

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011
(a) Traza del Proceso A	**(b) Traza del Proceso B**	**(c) Traza del Proceso C**

5000 = Dirección de comienzo del programa del Proceso A.
8000 = Dirección de comienzo del programa del Proceso B.
12000 = Dirección de comienzo del programa del Proceso C.

Figura 3.3. Traza de los procesos de la Figura 3.2.

E/S, para la cual debe esperar. Por tanto, el procesador deja de ejecutar el proceso B y pasa a ejecutar el proceso C, por medio del activador. Después de otra alarma de temporización, el procesador vuelve al proceso A. Cuando este proceso llega a su temporización, el proceso B aún estará esperando que se complete su operación de E/S, por lo que el activador pasa de nuevo al proceso C.

UN MODELO DE PROCESO DE DOS ESTADOS

La responsabilidad principal del sistema operativo es controlar la ejecución de los procesos; esto incluye determinar el patrón de entrelazado para la ejecución y asignar recursos a los procesos. El primer paso en el diseño de un sistema operativo para el control de procesos es describir el comportamiento que se desea que tengan los procesos.

Se puede construir el modelo más simple posible observando que, en un instante dado, un proceso está siendo ejecutando por el procesador o no. En este modelo, un proceso puede estar en dos estados: Ejecutando o No Ejecutando, como se muestra en la Figura 3.5a. Cuando el sistema operativo crea un nuevo proceso, crea el bloque de control de proceso (BCP) para el nuevo proceso e inserta dicho proceso en el sistema en estado No Ejecutando. El proceso existe, es conocido por el sistema operativo, y está esperando su oportunidad de ejecutar. De cuando en cuando, el proceso actualmente en ejecución se interrumpirá y una parte del sistema operativo, el activador, seleccionará otro proceso

1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
		Temporización
7	100	
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	
		Petición de E/S
17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	
27	12004	
28	12005	
		Temporización
29	100	
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
		Temporización
41	100	
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
		Temporización

100 = Dirección de comienzo del programa activador.
Las zonas sombreadas indican la ejecución del proceso de activación;
la primera y la tercera columna cuentan ciclos de instrucciones;
la segunda y la cuarta columna las direcciones de las instrucciones que se ejecutan

Figura 3.4. Traza combinada de los procesos de la Figura 3.2.

a ejecutar. El proceso saliente pasará del estado Ejecutando a No Ejecutando y pasará a Ejecutando un nuevo proceso.

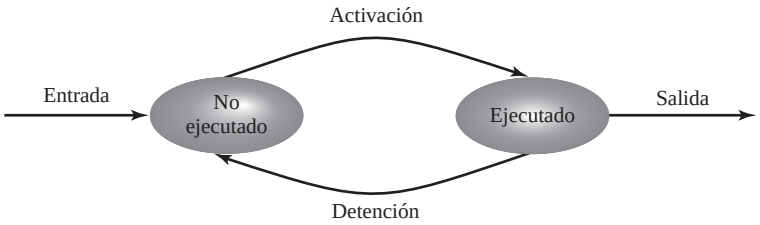
De este modelo simple, ya se puede apreciar algo del diseño de los elementos del sistema operativo. Cada proceso debe representarse de tal manera que el sistema operativo pueda seguirle la pista. Es decir, debe haber información correspondiente a cada proceso, incluyendo el estado actual y su localización en memoria; esto es el bloque de control de programa. Los procesos que no están ejecutando deben estar en una especie de cola, esperando su turno de ejecución. La Figura 3.5b sugiere esta estructura. Existe una sola cola cuyas entradas son punteros al BCP de un proceso en particular. Alternativamente, la cola debe consistir en una lista enlazada de bloques de datos, en la cual cada bloque representa un proceso; exploraremos posteriormente esta última implementación.

Podemos describir el comportamiento del activador en términos de este diagrama de colas. Un proceso que se interrumpe se transfiere a la cola de procesos en espera. Alternativamente, si el proceso ha finalizado o ha sido abortado, se descarta (sale del sistema). En cualquier caso, el activador selecciona un proceso de la cola para ejecutar.

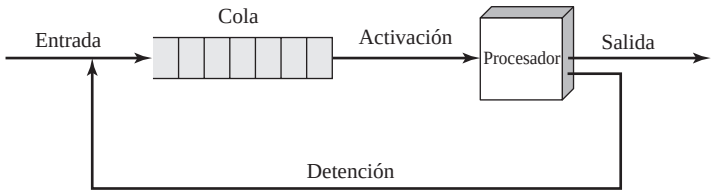
CREACIÓN Y TERMINACIÓN DE PROCESOS

Antes de intentar refinar nuestro sencillo modelo de dos estados, resultará útil hablar de la creación y terminación de los procesos; por último, y de forma independiente del modelo de comportamiento de procesos que se use, la vida de un proceso está acotada entre su creación y su terminación.

Creación de un proceso. Cuando se va a añadir un nuevo proceso a aquellos que se están gestionando en un determinado momento, el sistema operativo construye las estructuras de datos que se usan para manejar el proceso (como se describió en la Sección 3.3) y reserva el espacio de direcciones en memoria principal para el proceso. Estas acciones constituyen la creación de un nuevo proceso.



(a) Diagrama de transiciones de estados



(b) Modelos de colas

Figura 3.5. Modelo de proceso de dos estados.

Existen cuatro eventos comunes que llevan a la creación de un proceso, como se indica en la Tabla 3.1. En un entorno por lotes, un proceso se crea como respuesta a una solicitud de trabajo. En un entorno interactivo, un proceso se crea cuando un nuevo usuario entra en el sistema. En ambos casos el sistema operativo es responsable de la creación de nuevos procesos. Un sistema operativo puede, a petición de una aplicación, crear procesos. Por ejemplo, si un usuario solicita que se imprima un fichero, el sistema operativo puede crear un proceso que gestione la impresión. El proceso solicitado puede, de esta manera, operar independientemente del tiempo requerido para completar la tarea de impresión.

TABLA 3.1. Razones para la creación de un proceso.

Nuevo proceso de lotes	El sistema operativo dispone de un flujo de control de lotes de trabajos, habitualmente una cinta un disco. Cuando el sistema operativo está listo para procesar un nuevo trabajo, leerá la siguiente secuencia de mandatos de control de trabajos.
Sesión interactiva	Un usuario desde un terminal entra en el sistema.
Creado por el sistema operativo para proporcionar un servicio	El sistema operativo puede crear un proceso para realizar una función en representación de un programa de usuario, sin que el usuario tenga que esperar (por ejemplo, un proceso para controlar la impresión).
Creado por un proceso existente	Por motivos de modularidad o para explotar el paralelismo, un programa de usuario puede ordenar la creación de un número de procesos.

Tradicionalmente, un sistema operativo creaba todos los procesos de una forma que era transparente para usuarios y programas. Esto aún es bastante común en muchos sistemas operativos contemporáneos. Sin embargo, puede ser muy útil permitir a un proceso la creación de otro. Por ejemplo, un proceso de aplicación puede generar otro proceso para recibir los datos que la aplicación está generando y para organizar esos datos de una forma apropiada para su posterior análisis. El nuevo proceso ejecuta en paralelo con el proceso original y se activa cuando los nuevos datos están disponibles. Esta organización puede ser muy útil en la estructuración de la aplicación. Otro ejemplo, un proceso servidor (por ejemplo, un servidor de impresoras, servidor de ficheros) puede generar un proceso por cada solicitud que esté manejando. Cuando un sistema operativo crea un proceso a petición explícita de otro proceso, dicha acción se denomina ***creación del proceso***.

Cuando un proceso lanza otro, al primero se le denomina **proceso padre**, y al proceso creado se le denomina **proceso hijo**. Habitualmente, la relación entre procesos necesita comunicación y cooperación entre ellos. Alcanzar esta cooperación es una tarea complicada para un programador; este aspecto se verá en el Capítulo 5.

Terminación de procesos. La Tabla 3.2 resume las razones típicas para la terminación de un proceso. Todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización, o que ha completado su tarea. Un trabajo por lotes debe incluir una instrucción HALT o una llamada a un servicio de sistema operativo especifica para su terminación. En el caso anterior, la instrucción HALT generará una interrupción para indicar al sistema operativo que dicho proceso ha finalizado. Para una aplicación interactiva, las acciones del usuario indicarán cuando el proceso ha terminado. Por ejemplo, en un sistema de tiempo compartido, el proceso de un usuario en particular puede terminar cuando el usuario sale del sistema o apaga su terminal. En un ordenador personal o una estación de trabajo, el usuario puede salir de una aplicación (por ejemplo, un procesador de texto o una

hoja de cálculo). Todas estas acciones tienen como resultado final la solicitud de un servicio al sistema operativo para terminar con el proceso solicitante.

Adicionalmente, un número de error o una condición de fallo puede llevar a la finalización de un proceso. La Tabla 3.2 muestra una lista de las condiciones más habituales de finalización por esos motivos[3].

Tabla 3.2. Razones para la terminación de un proceso.

Finalización normal	El proceso ejecuta una llamada al sistema operativo para indicar que ha completado su ejecución
Límite de tiempo excedido	El proceso ha ejecutado más tiempo del especificado en un límite máximo. Existen varias posibilidades para medir dicho tiempo. Estas incluyen el tiempo total utilizado, el tiempo utilizado únicamente en ejecución, y, en el caso de procesos interactivos, la cantidad de tiempo desde que el usuario realizó la última entrada.
Memoria no disponible	El proceso requiere más memoria de la que el sistema puede proporcionar.
Violaciones de frontera	El proceso trata de acceder a una posición de memoria a la cual no tiene acceso permitido.
Error de protección	El proceso trata de usar un recurso, por ejemplo un fichero, al que no tiene permitido acceder, o trata de utilizarlo de una forma no apropia, por ejemplo, escribiendo en un fichero de sólo lectura.
Error aritmético	El proceso trata de realizar una operación de cálculo no permitida, tal como una división por 0, o trata de almacenar números mayores de los que la representación hardware puede codificar.
Límite de tiempo	El proceso ha esperado más tiempo que el especificado en un valor máximo para que se cumpla un determinado evento.
Fallo de E/S	Se ha producido un error durante una operación de entrada o salida, por ejemplo la imposibilidad de encontrar un fichero, fallo en la lectura o escritura después de un límite máximo de intentos (cuando, por ejemplo, se encuentra un área defectuosa en una cinta), o una operación inválida (la lectura de una impresora en línea).
Instrucción no válida	El proceso intenta ejecutar una instrucción inexistente (habitualmente el resultado de un salto a un área de datos y el intento de ejecutar dichos datos).
Instrucción privilegiada	El proceso intenta utilizar una instrucción reservada al sistema operativo.
Uso inapropiado de datos	Una porción de datos es de tipo erróneo o no se encuentra inicializada.
Intervención del operador por el sistema operativo	Por alguna razón, el operador o el sistema operativo ha finalizado el proceso (por ejemplo, se ha dado una condición de interbloqueo).
Terminación del proceso padre	Cuando un proceso padre termina, el sistema operativo puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre.
Solicitud del proceso padre	Un proceso padre habitualmente tiene autoridad para finalizar sus propios procesos descendientes.

[3] Un sistema operativo compasivo podría en algunos casos permitir al usuario recuperarse de un fallo sin terminar el proceso. Por ejemplo, si un usuario solicita acceder a un fichero y se deniega ese acceso, el sistema operativo podría simplemente informar al usuario de ese hecho y permitir que el proceso continúe.

Por último, en ciertos sistemas operativos, un proceso puede terminarse por parte del proceso que lo creó o cuando dicho proceso padre a su vez ha terminado.

MODELO DE PROCESO DE CINCO ESTADOS

Si todos los procesos estuviesen siempre preparados para ejecutar, la gestión de colas proporcionada en la Figura 3.5b sería efectiva. La cola es una lista de tipo FIFO y el procesador opera siguiendo una estrategia cíclica (*round-robin* o turno rotatorio) sobre todos los procesos disponibles (cada proceso de la cola tiene cierta cantidad de tiempo, por turnos, para ejecutar y regresar de nuevo a la cola, a menos que se bloquee). Sin embargo, hasta con el sencillo ejemplo que vimos antes, esta implementación es inadecuada: algunos procesos que están en el estado de No Ejecutando están listos para ejecutar, mientras que otros están bloqueados, esperando a que se complete una operación de E/S. Por tanto, utilizando una única cola, el activador no puede seleccionar únicamente los procesos que lleven más tiempo en la cola. En su lugar, debería recorrer la lista buscando los procesos que no estén bloqueados y que lleven en la cola más tiempo.

Una forma más natural para manejar esta situación es dividir el estado de No Ejecutando en dos estados, Listo y Bloqueado. Esto se muestra la Figura 3.6. Para gestionarlo correctamente, se han añadido dos estados adicionales que resultarán muy útiles. Estos cinco estados en el nuevo diagrama son los siguientes:

- **Ejecutando.** El proceso está actualmente en ejecución. Para este capítulo asumimos que el computador tiene un único procesador, de forma que sólo un proceso puede estar en este estado en un instante determinado.
- **Listo.** Un proceso que se prepara para ejecutar cuando tenga oportunidad.
- **Bloqueado.** Un proceso que no puede ejecutar hasta que se cumpla un evento determinado o se complete una operación E/S.
- **Nuevo.** Un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo. Típicamente, se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su bloque de control de proceso (BCP) si ha sido creado.
- **Saliente.** Un proceso que ha sido liberado del grupo de procesos ejecutables por el sistema operativo, debido a que ha sido detenido o que ha sido abortado por alguna razón.

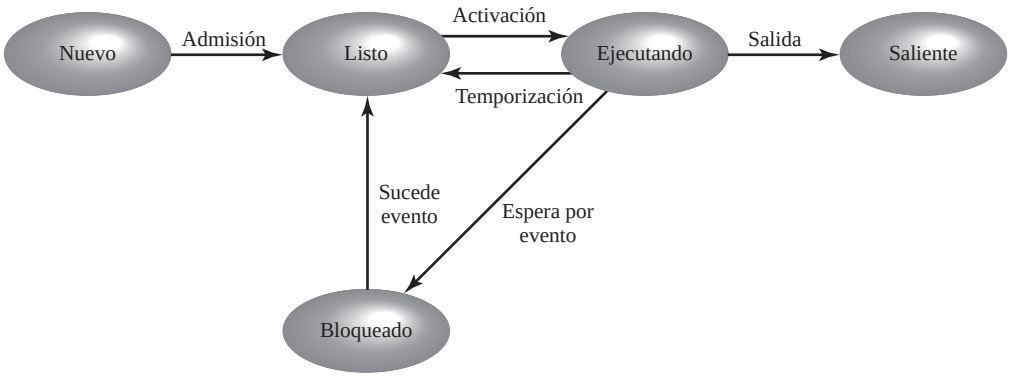


Figura 3.6. Modelo de proceso de cinco estados.

Los estados Nuevo y Saliente son útiles para construir la gestión de procesos. El estado Nuevo se corresponde con un proceso que acaba de ser definido. Por ejemplo, si un nuevo usuario intenta entrar dentro de un sistema de tiempo compartido o cuando se solicita un nuevo trabajo a un sistema de proceso por lotes, el sistema operativo puede definir un nuevo proceso en dos etapas. Primero, el sistema operativo realiza todas las tareas internas que correspondan. Se asocia un identificador a dicho proceso. Se reservan y construyen todas aquellas tablas que se necesiten para gestionar al proceso. En este punto, el proceso se encuentra el estado Nuevo. Esto significa que el sistema operativo ha realizado todas las tareas necesarias para crear el proceso pero el proceso en sí, aún no se ha puesto en ejecución. Por ejemplo, un sistema operativo puede limitar el número de procesos que puede haber en el sistema por razones de rendimiento o limitaciones de memoria principal. Mientras un proceso está en el estado Nuevo, la información relativa al proceso que se necesite por parte del sistema operativo se mantiene en tablas de control de memoria principal. Sin embargo, el proceso en sí mismo no se encuentra en memoria principal. Esto es, el código de programa a ejecutar no se encuentra en memoria principal, y no se ha reservado ningún espacio para los datos asociados al programa. Cuando un proceso se encuentra en el estado Nuevo, el programa permanece en almacenamiento secundario, normalmente en disco[4].

De forma similar, un proceso sale del sistema en dos fases. Primero, el proceso termina cuando alcanza su punto de finalización natural, cuando es abortado debido a un error no recuperable, o cuando otro proceso con autoridad apropiada causa que el proceso se aborte. La terminación mueve el proceso al estado Saliente. En este punto, el proceso no es elegible de nuevo para su ejecución. Las tablas y otra información asociada con el trabajo se encuentran temporalmente preservadas por el sistema operativo, el cual proporciona tiempo para que programas auxiliares o de soporte extraigan la información necesaria. Por ejemplo, un programa de auditoría puede requerir registrar el tiempo de proceso y otros recursos utilizados por este proceso saliente con objeto de realizar una contabilidad de los recursos del sistema. Un programa de utilidad puede requerir extraer información sobre el histórico de los procesos por temas relativos con el rendimiento o análisis de la utilización. Una vez que estos programas han extraído la información necesaria, el sistema operativo no necesita mantener ningún dato relativo al proceso y el proceso se borra del sistema.

La Figura 3.6 indica que tipos de eventos llevan a cada transición de estado para cada proceso; las posibles transiciones son las siguientes:

- **Null → Nuevo.** Se crea un nuevo proceso para ejecutar un programa. Este evento ocurre por cualquiera de las relaciones indicadas en la tabla 3.1.
- **Nuevo → Listo.** El sistema operativo mueve a un proceso del estado nuevo al estado listo cuando éste se encuentre preparado para ejecutar un nuevo proceso. La mayoría de sistemas fijan un límite basado en el número de procesos existentes o la cantidad de memoria virtual que se podrá utilizar por parte de los procesos existentes. Este límite asegura que no haya demasiados procesos activos y que se degrade el rendimiento sistema.
- **Listo → Ejecutando.** Cuando llega el momento de seleccionar un nuevo proceso para ejecutar, el sistema operativo selecciona uno los procesos que se encuentre en el estado Listo. Esta es una tarea la lleva acabo el planificador. El planificador se estudiará más adelante en la Parte Cuatro.
- **Ejecutando → Saliente.** el proceso actual en ejecución se finaliza por parte del sistema operativo tanto si el proceso indica que ha completado su ejecución como si éste se aborta. *Véase* tabla 3.2.

---

[4] En las explicaciones de este párrafo, hemos ignorado el concepto de memoria virtual. En sistemas que soporten la memoria virtual, cuando un proceso se mueve de Nuevo a Listo, su código de programa y sus datos se cargan en memoria virtual. La memoria virtual se ha explicado brevemente en el Capítulo 2 y se examinará con más detalle en el Capítulo 8.

- **Ejecutando → Listo.** La razón más habitual para esta transición es que el proceso en ejecución haya alcanzado el máximo tiempo posible de ejecución de forma ininterrumpida; prácticamente todos los sistemas operativos multiprogramados imponen este tipo de restricción de tiempo. Existen otras posibles causas alternativas para esta transición, que no están incluidas en todos los sistemas operativos. Es de particular importancia el caso en el cual el sistema operativo asigna diferentes niveles de prioridad a diferentes procesos. Supóngase, por ejemplo, que el proceso A está ejecutando a un determinado nivel de prioridad, y el proceso B, a un nivel de prioridad mayor, y que se encuentra bloqueado. Si el sistema operativo se da cuenta de que se produce un evento al cual el proceso B está esperando, moverá el proceso B al estado de Listo. Esto puede interrumpir al proceso A y poner en ejecución al proceso B. Decimos, en este caso, que el sistema operativo ha expulsado al proceso A[5]. Adicionalmente, un proceso puede voluntariamente dejar de utilizar el procesador. Un ejemplo son los procesos que realiza alguna función de auditoría o de mantenimiento de forma periódica.
- **Ejecutando → Bloqueado.** Un proceso se pone en el estado Bloqueado si solicita algo por lo cual debe esperar. Una solicitud al sistema operativo se realiza habitualmente por medio de una llamada al sistema; esto es, una llamada del proceso en ejecución a un procedimiento que es parte del código del sistema operativo. Por ejemplo, un proceso ha solicitado un servicio que el sistema operativo no puede realizar en ese momento. Puede solicitar un recurso, como por ejemplo un fichero o una sección compartida de memoria virtual, que no está inmediatamente disponible. Cuando un proceso quiere iniciar una acción, tal como una operación de E/S, que debe completarse antes de que el proceso continúe. Cuando un proceso se comunica con otro, un proceso puede bloquearse mientras está esperando a que otro proceso le proporcione datos o esperando un mensaje de ese otro proceso.
- **Bloqueado → Listo.** Un proceso en estado Bloqueado se mueve al estado Listo cuando sucede el evento por el cual estaba esperando.
- **Listo → Saliente.** Por claridad, esta transición no se muestra en el diagrama de estados. En algunos sistemas, un padre puede terminar la ejecución de un proceso hijo en cualquier momento. También, si el padre termina, todos los procesos hijos asociados con dicho padre pueden finalizarse.
- **Bloqueado → Saliente.** Se aplican los comentarios indicados en el caso anterior.

Si regresamos a nuestro sencillo ejemplo, Figura 3.7, ahí se muestra la transición entre cada uno de los estados de proceso. La figura 3.8a sugiere la forma de aplicar un esquema de dos colas: la colas de Listos y la cola de Bloqueados. Cada proceso admitido por el sistema, se coloca en la cola de Listos. Cuando llega el momento de que el sistema operativo seleccione otro proceso a ejecutar, selecciona uno de la cola de Listos. En ausencia de un esquema de prioridad, esta cola puede ser una lista de tipo FIFO (*first-in-first-out*). Cuando el proceso en ejecución termina de utilizar el procesador, o bien finaliza o bien se coloca en la cola de Listos o de Bloqueados, dependiendo de las circunstancias. Por último, cuando sucede un evento, cualquier proceso en la cola de Bloqueados que únicamente esté esperando a dicho evento, se mueve a la cola de Listos.

Esta última transición significa que, cuando sucede un evento, el sistema operativo debe recorrer la cola entera de Bloqueados, buscando aquellos procesos que estén esperando por dicho evento. En

---

[5] En general, el término **expulsión** (*preemption*) se define como la reclamación de un recurso por parte de un proceso antes de que el proceso que lo poseía finalice su uso. En este caso, el recurso es el procesador. El proceso está ejecutando y puede continuar su ejecución pero es expulsado por otro proceso que va a entrar a ejecutar.

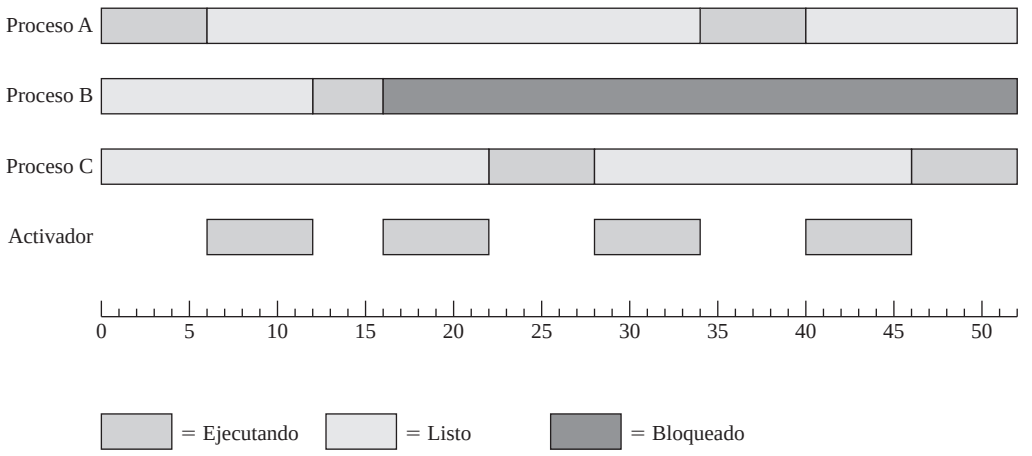


Figura 3.7. Estado de los procesos de la traza de la Figura 3.4.

los sistemas operativos con muchos procesos, esto puede significar cientos o incluso miles de procesos en esta lista, por lo que sería mucho más eficiente tener una cola por cada evento. De esta forma, cuando sucede un evento, la lista entera de procesos de la cola correspondiente se movería al estado de Listo (Figura 3. 8b).

Un refinamiento final sería: si la activación de procesos está dictada por un esquema de prioridades, sería conveniente tener varias colas de procesos listos, una por cada nivel de prioridad. El sistema operativo podría determinar cual es el proceso listo de mayor prioridad simplemente seleccionando éstas en orden.

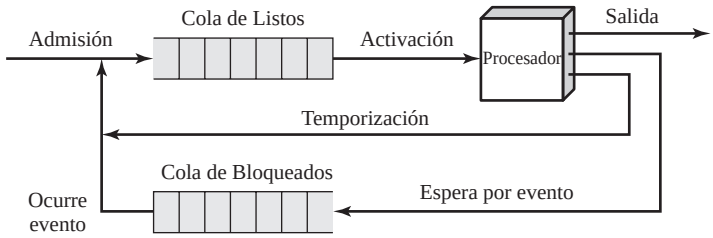
PROCESOS SUSPENDIDOS

La necesidad de intercambio o *swapping*. Los tres principales estados descritos (Listo, Ejecutando, Bloqueado) proporcionan una forma sistemática de modelizar el comportamiento de los procesos y diseñar la implementación del sistema operativo. Se han construido algunos sistemas operativos utilizando únicamente estos tres estados.

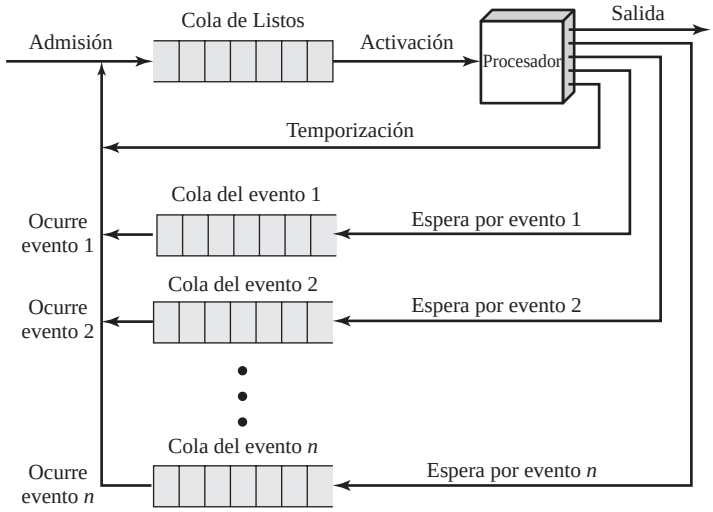
Sin embargo, existe una buena justificación para añadir otros estados al modelo. Para ver este beneficio de nuevos estados, vamos a suponer un sistema que no utiliza memoria virtual. Cada proceso que se ejecuta debe cargarse completamente en memoria principal. Por ejemplo, en la Figura 3.8b, todos los procesos en todas las colas deben residir en memoria principal.

Recuérdese que la razón de toda esta compleja maquinaria es que las operaciones de E/S son mucho más lentas que los procesos de cómputo y, por tanto, el procesador en un sistema monoprogramado estaría ocioso la mayor parte del tiempo. Pero los ajustes de la Figura 3.8b no resuelven completamente el problema. Es verdad que, en este caso, la memoria almacena múltiples procesos y el procesador puede asignarse a otro proceso si el que lo usa se queda bloqueado. La diferencia de velocidad entre el procesador y la E/S es tal que sería muy habitual que todos los procesos en memoria se encontrasen a esperas de dichas operaciones. Por tanto, incluso en un sistema multiprogramado, el procesador puede estar ocioso la mayor parte del tiempo.

¿Qué se puede hacer? La memoria principal puede expandirse para acomodar más procesos. Hay dos fallos en esta solución. Primero, existe un coste asociado a la memoria principal, que, desde un



(a) Cola simple de Bloqueados



(b) Múltiples colas de Bloqueados

Figura 3.8. Modelo de colas de la Figura 3.6.

coste reducido a nivel de bytes, comienza a incrementarse según nos acercamos a un almacenamiento de gigabytes. Segundo, el apetito de los programas a nivel de memoria ha crecido tan rápido como ha bajado el coste de las memorias. De forma que las grandes memorias actuales han llevado a ejecutar procesos de gran tamaño, no más procesos.

Otra solución es el *swapping* (memoria de intercambio), que implica mover parte o todo el proceso de memoria principal al disco. Cuando ninguno de los procesos en memoria principal se encuentra en estado Listo, el sistema operativo intercambia uno de los procesos bloqueados a disco, en la cola de Suspendidos. Esta es una lista de procesos existentes que han sido temporalmente expulsados de la memoria principal, o suspendidos. El sistema operativo trae otro proceso de la cola de Suspendidos o responde a una solicitud de un nuevo proceso. La ejecución continúa con los nuevos procesos que han llegado.

El *swapping*, sin embargo, es una operación de E/S, y por tanto existe el riesgo potencial de hacer que el problema empeore. Pero debido a que la E/S en disco es habitualmente más rápida que la E/S sobre otros sistemas (por ejemplo, comparado con cinta o impresora), el *swapping* habitualmente mejora el rendimiento del sistema.

Con el uso de *swapping* tal y como se ha escrito, debe añadirse un nuevo estado a nuestro modelo de comportamiento de procesos (Figura 3.9a): el estado Suspendido. Cuando todos los procesos en

memoria principal se encuentran en estado Bloqueado, el sistema operativo puede suspender un proceso poniéndolo en el estado Suspendido y transfiriéndolo a disco. El espacio que se libera en memoria principal puede usarse para traer a otro proceso.

Cuando el sistema operativo ha realizado la operación de *swap* (transferencia a disco de un proceso), tiene dos opciones para seleccionar un nuevo proceso para traerlo a memoria principal: puede admitir un nuevo proceso que se haya creado o puede traer un proceso que anteriormente se encontrase en estado de Suspendido. Parece que sería preferible traer un proceso que anteriormente estuviese suspendido, para proporcionar dicho servicio en lugar de incrementar la carga total del sistema.

Pero, esta línea de razonamiento presenta una dificultad: todos los procesos que fueron suspendidos se encontraban previamente en el estado de Bloqueado en el momento de su suspensión. Claramente no sería bueno traer un proceso bloqueado de nuevo a memoria porque podría no encontrarse todavía listo para la ejecución. Se debe reconocer, sin embargo, que todos los procesos en estado Suspendido estaban originalmente en estado Bloqueado, en espera de un evento en particular. Cuando el evento sucede, el proceso no está Bloqueado y está potencialmente disponible para su ejecución.

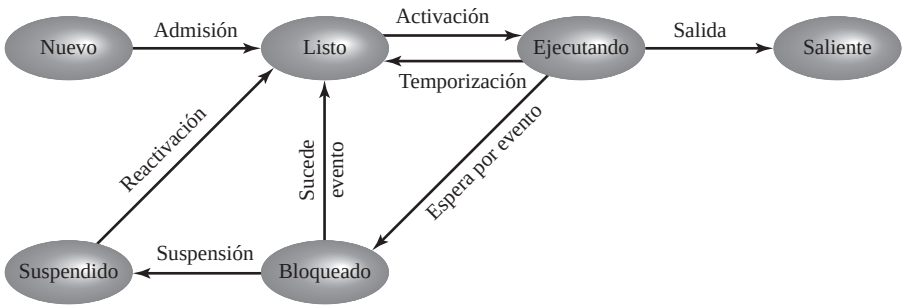
De esta forma, necesitamos replantear este aspecto del diseño. Hay dos conceptos independientes aquí: si un proceso está esperando a un evento (Bloqueado o no) y si un proceso está transferido de memoria a disco (suspendido o no). Para describir estas combinaciones de 2×2 necesitamos cuatro estados:

- **Listo.** El proceso está en memoria principal disponible para su ejecución.
- **Bloqueado.** El proceso está en memoria principal y esperando un evento.
- **Bloqueado/Suspendido.** El proceso está en almacenamiento secundario y esperando un evento.
- **Listo/Suspendido.** El proceso está en almacenamiento secundario pero está disponible para su ejecución tan pronto como sea cargado en memoria principal.

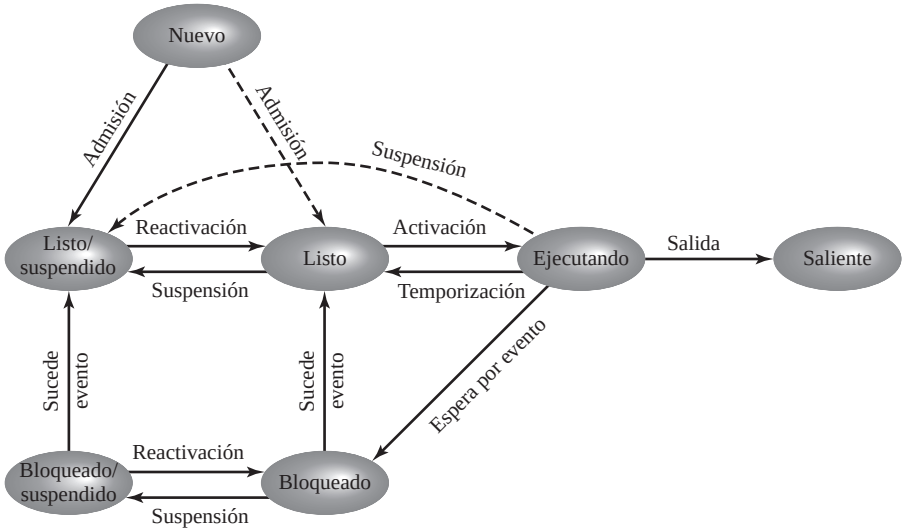
Antes de ir a ver un diagrama de transiciones de estado que incluya estos dos nuevos estados Suspendidos, se debe mencionar otro punto. La explicación ha asumido que no se utiliza memoria virtual y que un proceso está completamente en memoria principal, o completamente fuera de la misma. Con un esquema de memoria virtual, es posible ejecutar un proceso que está sólo parcialmente en memoria principal. Si se hace referencia a una dirección de proceso que no se encuentra en memoria principal, la porción correspondiente del proceso se trae a ella. El uso de la memoria principal podría parecer que elimina la necesidad del *swapping* explícito, porque cualquier dirección necesitada por un proceso que la demande puede traerse o transferirse de disco a memoria por el propio hardware de gestión de memoria del procesador. Sin embargo, como veremos en el Capítulo 8 el rendimiento de los sistemas de memoria virtual pueden colapsarse si hay un número suficientemente grande de procesos activos, todos ellos están parcialmente en memoria principal. De esta forma, incluso con un sistema de memoria virtual, el sistema operativo necesita hacer *swapping* de los procesos de forma explícita y completa, de vez en cuando, con la intención de mejorar el rendimiento.

Veamos ahora, en la Figura 3.9b, el modelo de transición de estados que ha sido diseñado. (Las líneas punteadas de la figura indican transiciones posibles pero no necesarias.) Las nuevas transiciones más importantes son las siguientes:

- **Bloqueado → Bloqueado/Suspendido.** Si no hay procesos listos, entonces al menos uno de los procesos bloqueados se transfiere al disco para hacer espacio para otro proceso que no se encuentra bloqueado. Esta transición puede realizarse incluso si hay procesos listos disponi-



(a) Con un único estado Suspendido



(b) Con dos estados Suspendido

Figura 3.9. Diagrama de transición de estados de procesos con estado suspendidos.

bles, si el sistema operativo determina que el proceso actualmente en ejecución o los procesos listos que desea ejecutar requieren más memoria principal para mantener un rendimiento adecuado.

- **Bloqueado/Suspendido → Listo/Suspendido.** Un proceso en el estado Bloqueado/Suspendido se mueve al estado Listo/Suspendido cuando sucede un evento al que estaba esperando. Nótese que esto requiere que la información de estado concerniente a un proceso suspendido sea accesible para el sistema operativo.
- **Listo/Suspendido → Listo.** Cuando no hay más procesos listos en memoria principal, el sistema operativo necesitará traer uno para continuar la ejecución. Adicionalmente, puede darse el caso de que un proceso en estado Listo/Suspendido tenga mayor prioridad que cualquiera de los procesos en estado Listo. En este caso, el sistema operativo puede haberse diseñado para determinar que es más importante traer un proceso de mayor prioridad que para minimizar el efecto del *swapping*.

- **Listo → Listo/Suspendido.** Normalmente, el sistema operativo preferirá suspender procesos bloqueados que un proceso Listo, porque un proceso Listo se puede ejecutar en ese momento, mientras que un proceso Bloqueado ocupa espacio de memoria y no se puede ejecutar. Sin embargo, puede ser necesario suspender un proceso Listo si con ello se consigue liberar un bloque suficientemente grande de memoria. También el sistema operativo puede decidir suspender un proceso Listo de baja prioridad antes que un proceso Bloqueado de alta prioridad, si se cree que el proceso Bloqueado estará pronto listo.

Otras transiciones interesantes de considerar son las siguientes:

- **Nuevo → Listo/Suspendido y Nuevo a Listo.** Cuando se crea un proceso nuevo, puede añadirse a la cola de Listos o a la cola de Listos/Suspendidos. En cualquier caso, el sistema operativo puede crear un bloque de control de proceso (BCP) y reservar el espacio de direcciones del proceso. Puede ser preferible que el sistema operativo realice estas tareas internas cuanto antes, de forma que pueda disponer de suficiente cantidad de procesos no bloqueados. Sin embargo, con esta estrategia, puede ocurrir que no haya espacio suficiente en memoria principal para el nuevo proceso; de ahí el uso de la transición (Nuevo→Listo/Suspendido). Por otro lado, deseamos hacer hincapié en que la filosofía de creación de procesos diferida, haciéndolo cuanto más tarde, hace posible reducir la sobrecarga del sistema operativo y le permite realizar las tareas de creación de procesos cuando el sistema está lleno de procesos bloqueados.
- **Bloqueado/Suspendido → Bloqueado.** La incursión de esta transición puede parecer propia de un diseño más bien pobre. Después de todo, si hay un proceso que no está listo para ejecutar y que no está en memoria principal, ¿qué sentido tiene traerlo? Pero considérese el siguiente escenario: un proceso termina, liberando alguna memoria principal. Hay un proceso en la cola de Bloqueados/Suspendidos con mayor prioridad que todos los procesos en la cola de Listos/Suspendidos y el sistema operativo tiene motivos para creer que el evento que lo bloquea va a ocurrir en breve. Bajo estas circunstancias, sería razonable traer el proceso Bloqueado a memoria por delante de los procesos Listos.
- **Ejecutando → Listo/Suspendido.** Normalmente, un proceso en ejecución se mueve a la cola de Listos cuando su tiempo de uso del procesador finaliza. Sin embargo, si el sistema operativo expulsa al proceso en ejecución porque un proceso de mayor prioridad en la cola de Bloqueado/Suspendido acaba de desbloquearse, el sistema operativo puede mover al proceso en ejecución directamente a la cola de Listos/Suspendidos y liberar parte de la memoria principal.
- **De cualquier estado → Saliente.** Habitualmente, un proceso termina cuando está ejecutando, bien porque ha completado su ejecución o debido a una condición de fallo. Sin embargo, en algunos sistemas operativos un proceso puede terminarse por el proceso que lo creó o cuando el proceso padre a su vez ha terminado. Si se permite esto, un proceso en cualquier estado puede moverse al estado Saliente.

Otros usos para la suspensión de procesos. Hace poco, hemos definido el concepto de procesos suspendidos como el proceso que no se encuentra en memoria principal. Un proceso que no está en memoria principal no está disponible de forma inmediata para su ejecución, independientemente de si está a la espera o no de un evento.

Podemos analizar el concepto de procesos suspendidos, definiendo un proceso suspendido como el que cumple las siguientes características:

1. El proceso no está inmediatamente disponible para su ejecución.

2. El proceso puede estar o no a la espera de un evento, si es así, la condición de bloqueo es independiente de la condición estar suspendido, y si sucede el evento que lo bloquea, eso no habilita al proceso para su ejecución inmediata.
3. El proceso fue puesto en estado suspendido por un agente: bien el proceso mismo, el proceso padre o el sistema operativo, con el propósito de prevenir su ejecución.
4. El proceso no puede ser recuperado de este estado hasta que el agente explícitamente así lo indique.

Tabla 3.3. Razones para la suspensión de un proceso.

Swapping	El sistema operativo necesita liberar suficiente memoria principal para traer un proceso en estado Listo de ejecución.
Otras razones del sistema operativo	El sistema operativo puede suspender un proceso en segundo plano o de utilidad o un proceso que se sospecha puede causar algún problema.
Solicitud interactiva del usuario	Un usuario puede desear suspender la ejecución de un programa con motivo de su depuración o porque está utilizando un recurso.
Temporización	Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso monitor de estadísticas sobre el sistema) y puede suspenderse mientras espera el siguiente intervalo de ejecución.
Solicitud del proceso padre	Un proceso padre puede querer suspender la ejecución de un descendiente para examinar o modificar dicho proceso suspendido, o para coordinar la actividad de varios procesos descendientes.

La Tabla 3.3 muestra una lista de las condiciones para la suspensión de un proceso. Una de las razones que hemos discutido anteriormente es proporcionar espacio de memoria para traer procesos en estado Listos/Suspendidos o para incrementar el espacio de memoria disponible para los procesos Listos. El sistema operativo puede tener otros motivos para suspender un proceso. Por ejemplo, se puede utilizar un proceso de traza o de auditoría para monitorizar una actividad del sistema; el proceso puede recoger datos a nivel de utilización de varios recursos (procesador, memoria, canales) y la tasa de progreso del uso de los procesos en el sistema. El sistema operativo, bajo el control del operador, puede activar y desactivar este proceso con determinada periodicidad. Si el sistema operativo detecta o sospecha que puede haber un problema, puede suspender procesos. Un ejemplo de esto es un interbloqueo, que ya se discutirá en el Capítulo 6. Otro ejemplo, se detectan problemas en una línea de comunicación, el operador solicita al sistema operativo que suspenda el proceso que utiliza dicha línea hasta realizar algunas pruebas.

Otro tipo de razones están relacionadas con el uso interactivo del sistema. Por ejemplo, si los usuarios sospechan de un programa con algún tipo de *bug*, se puede detener la ejecución de dicho programa, examinando y modificando el programa y sus datos y reanudándolo de nuevo. O puede haber un proceso en segundo plano que recolecta trazas y estadísticas, el cual puede ser activado por parte del usuario.

Algunas consideraciones de tiempo también pueden llevar a activar decisiones de *swapping*. Por ejemplo, si un proceso se activa periódicamente pero está ocioso la mayor parte del tiempo puede ser enviado a disco entre cada una de las ejecuciones que realiza. Un ejemplo es un programa que monitoriza la utilización del sistema y la actividad del usuario.

Por último, un proceso padre puede suspender un proceso descendiente. Por ejemplo, el proceso A lanza al proceso B para realizar una lectura sobre un fichero. Posteriormente, el proceso B encuentra un error en la lectura del fichero y se lo indica al proceso A. El proceso A suspende al proceso B e investiga la causa.

En todos estos casos, la activación de un proceso Suspendido se solicita por medio del agente que inicialmente puso al proceso en suspensión.

3.3. DESCRIPCIÓN DE PROCESOS

El sistema operativo controla los eventos dentro del computador, planifica y activa los procesos para su ejecución por el procesador, reserva recursos para los mismos y responde a las solicitudes de servicios básicos de los procesos de usuario. Fundamentalmente, se piensa en el sistema operativo como en la entidad que gestiona el uso de recursos del sistema por parte de los procesos.

Este concepto se ilustra en la Figura 3.10. En un entorno multiprogramado, hay numerosos procesos (P1,... Pn) creados y residentes en memoria virtual. Cada proceso, durante el transcurso de su ejecución, necesita acceder a ciertos recursos del sistema, incluido el procesador, los dispositivos de E/S y la memoria principal. En la figura, el proceso P1 está ejecutando; al menos parte del proceso está en memoria principal, y controla dos dispositivos de E/S. El proceso P2 está también totalmente en memoria principal pero está bloqueado esperando a disponer de un dispositivo de E/S que está asignado a P1. El proceso Pn se encuentra transferido a disco (*swapping*) y está por tanto suspendido.

Exploraremos los detalles de la gestión de estos recursos por parte del sistema operativo en los próximos capítulos. Aquí nos interesa una cuestión más fundamental: ¿qué información necesita el sistema operativo para controlar los procesos y gestionar los recursos de estos?

ESTRUCTURAS DE CONTROL DEL SISTEMA OPERATIVO

Si el sistema operativo se encarga de la gestión de procesos y recursos, debe disponer de información sobre el estado actual de cada proceso y cada recurso. El mecanismo universal para proporcionar esta información es el siguiente: el sistema operativo construye y mantiene tablas de información sobre cada entidad que gestiona. Se indica una idea general del alcance de esta tarea en la Figura 3.11, que muestra cuatro diferentes tipos de tablas mantenidas por el sistema operativo: memoria, E/S, ficheros y procesos. A pesar de que los detalles difieren de un sistema operativo a otro, fundamentalmente, todos los sistemas operativos mantienen información de estas cuatro categorías.

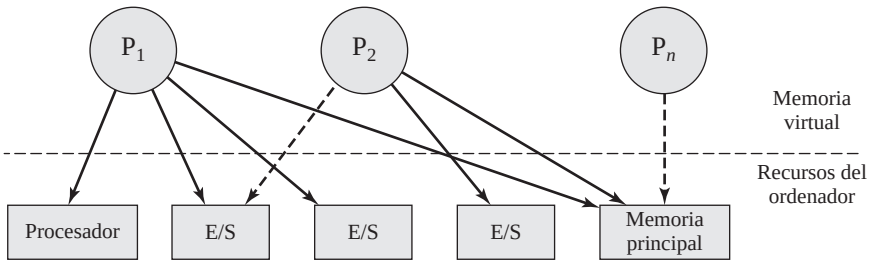


Figura 3.10. Procesos y recursos (reserva de recursos en una instantánea del sistema).

Las **tablas de memoria** se usan para mantener un registro tanto de la memoria principal (real) como de la secundaria (virtual). Parte de la memoria principal está reservada para el uso del sistema operativo; el resto está disponible para el uso de los procesos. Los procesos se mantienen en memoria secundaria utilizando algún tipo de memoria virtual o técnicas de *swapping*. Las tablas de memoria deben incluir la siguiente información:

- Las reservas de memoria principal por parte de los procesos.
- Las reservas de memoria secundaria por parte de los procesos.
- Todos los atributos de protección que restringe el uso de la memoria principal y virtual, de forma que los procesos puedan acceder a ciertas áreas de memoria compartida.
- La información necesaria para manejar la memoria virtual.

Veremos en detalle las estructuras de información para la gestión de memoria en la Parte Tres.

El sistema operativo debe utilizar las **tablas de E/S** para gestionar los dispositivos de E/S y los canales del computador. Pero, en un instante determinado, un dispositivo E/S puede estar disponible o asignado a un proceso en particular. Si la operación de E/S se está realizando, el sistema operativo necesita conocer el estado de la operación y la dirección de memoria principal del área usada como fuente o destino de la transferencia de E/S. La gestión de E/S se examina en detalle en el Capítulo 11.

El sistema operativo también puede mantener las **tablas de ficheros**. Estas tablas proporcionan información sobre la existencia de ficheros, su posición en almacenamiento secundario, su estado ac-

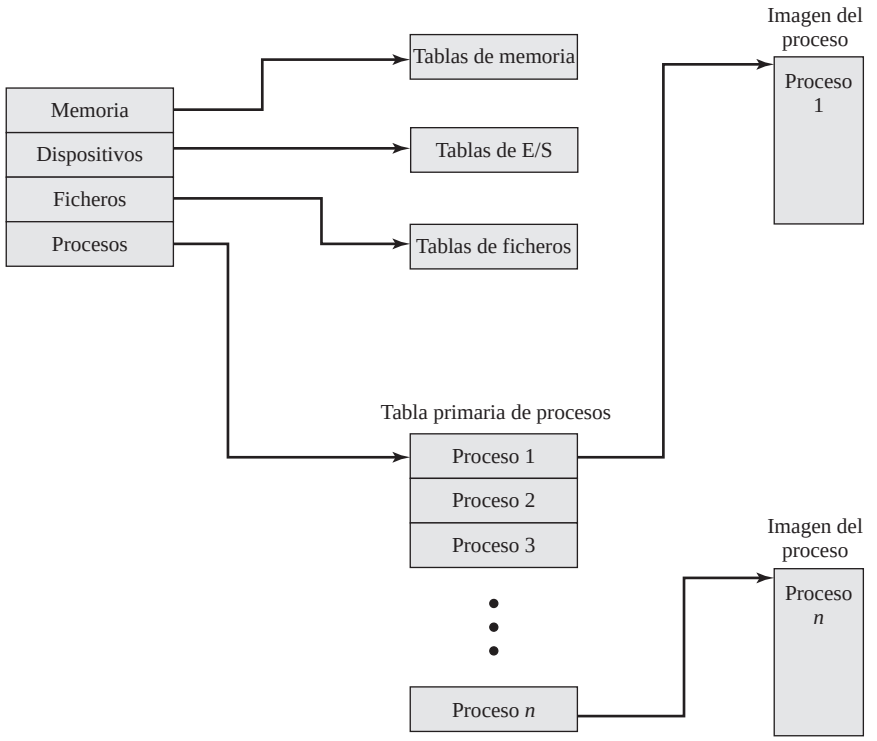


Figura 3.11. Estructura general de las tablas de control del sistema operativo.

tual, y otros atributos. La mayoría de, o prácticamente toda, esta información se puede gestionar por el sistema de ficheros, en cuyo caso el sistema operativo tiene poco o ningún conocimiento sobre los ficheros. En otros sistemas operativos, la gran mayoría del detalle de la gestión de ficheros sí que gestiona en el propio sistema operativo. Este aspecto se explorará en el Capítulo 12.

En último lugar, el sistema operativo debe mantener **tablas de procesos** para gestionar los procesos. La parte restante de esta sección se encuentra dedicada a examinar los requisitos que tienen las tablas de procesos. Antes de proceder con esta explicación, vamos a realizar dos puntualizaciones importantes. Primero, a pesar de que la Figura 3.11 muestra cuatro tipos diferentes de tablas debe quedar claro que estas tablas se encuentran entrelazadas y referenciadas entre sí de alguna manera. Memoria, E/S, y ficheros se gestionan por parte de los procesos, de forma que debe haber algunas referencias en estos recursos, directa o indirectamente, desde las tablas de procesos. Los ficheros indicados en las tablas de ficheros son accesibles mediante dispositivos E/S y estarán, en algún caso, residentes en memoria virtual o principal. Las tablas son también accesibles para el sistema operativo, y además están controladas por la gestión de memoria.

Segundo, ¿cómo puede el sistema operativo crear las tablas iniciales? Ciertamente, el sistema operativo debe tener alguna información sobre el entorno básico, tal como: cuánta memoria física existe, cuáles son los dispositivos de E/S y cuáles son sus identificadores, por ejemplo. Esto es una cuestión de configuración. Esto es, cuando el sistema operativo se inicializa, debe tener acceso a algunos datos de configuración que definen el entorno básico y estos datos se deben crear fuera del sistema operativo, con asistencia humana o por medio de algún software de autoconfiguración.

ESTRUCTURAS DE CONTROL DE PROCESO

Se considerará qué información debe conocer el sistema operativo si quiere manejar y controlar los procesos. Primero, debe conocer dónde están localizados los procesos, y segundo, debe conocer los atributos de los procesos que quiere gestionar (por ejemplo, identificador de proceso y estado del mismo).

Localización de los procesos. Antes de que podamos tratar la cuestión de dónde se encuentran los procesos o cuáles son sus atributos, debemos tratar una cuestión más fundamental: ¿qué es la representación física de un proceso? Como mínimo, un proceso debe incluir un programa o un conjunto de programas a ejecutar. Asociados con estos programas existen unas posiciones de memoria para los datos de variables locales y globales y de cualquier constante definida. Así, un proceso debe consistir en una cantidad suficiente de memoria para almacenar el programa y datos del mismo. Adicionalmente, la ejecución típica de un programa incluye una pila (*véase* Apéndice 1B) que se utiliza para registrar las llamadas a procedimientos y los parámetros pasados entre dichos procedimientos. Por último, cada proceso está asociado a un número de atributos que son utilizados por el sistema operativo para controlar el proceso. Normalmente, el conjunto de estos atributos se denomina **bloque de control del proceso** (BCP)[6]. Nos podemos referir al conjunto de programa, datos, pila, y atributos, como la **imagen del proceso** (Tabla 3.4).

La posición de la imagen del proceso dependerá del esquema de gestión de memoria que se utilice. En el caso más simple, la imagen del proceso se mantiene como un bloque de memoria contiguo, o continuo. Este bloque se mantiene en memoria secundaria, habitualmente disco. Para que el sistema operativo pueda gestionar el proceso, al menos una pequeña porción de su imagen se debe mantener

---

[6] Otros nombres habituales usados para esta estructura de datos son bloque de control de tarea, descriptor del proceso, y descriptor de tarea.

en memoria principal. Para ejecutar el proceso, la imagen del proceso completa se debe cargar en memoria principal o al menos en memoria virtual. Asimismo, el sistema operativo necesita conocer la posición en disco de cada proceso y, para cada proceso que se encuentre en memoria principal, su posición en dicha memoria. Se vio una ligera modificación un poco más compleja de este esquema con el sistema operativo CTSS, en el Capítulo 2. Con CTSS, cuando un proceso se transfiere a disco (*swapping*) parte de la imagen del proceso permanece en memoria principal. De esta forma, el sistema operativo debe mantener en registro qué partes de la imagen del proceso se encuentran todavía en memoria principal.

Tabla 3.4. Elementos típicos de la imagen de un proceso.

Datos del usuario	La parte modificable del espacio de usuario. Puede incluir datos de programa, área de pila de usuario, y programas que puedan ser modificados.
Programa de usuario	El programa a ejecutar
Pila de sistema	Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema.
Bloque de control de proceso	Datos necesarios para que el sistema operativo pueda controlar los procesos (*véase* tabla 3.5).

Los sistemas operativos modernos suponen la existencia de un hardware de paginación que permite el uso de la memoria física no contigua, para dar soporte a procesos parcialmente residentes en la memoria principal. En esos casos, una parte de la imagen del proceso se puede encontrar en dicha memoria principal, con las restantes en memoria secundaria[7]. De esta forma, las tablas mantenidas por sistema operativo deben mostrar la localización de cada página de la imagen del proceso.

La Figura 3.11 muestra la estructura de la información de localización de la siguiente forma. Hay una tabla primaria de procesos con una entrada por cada proceso. Cada entrada contiene, al menos, un puntero a la imagen del proceso. Si la imagen del proceso contiene múltiples bloques, esta información se mantiene directamente en la tabla principal del proceso o bien está disponible por referencias cruzadas entre las entradas de las tablas de memoria. Por supuesto, esta es una representación genérica; un sistema operativo en particular puede tener su propia forma de organizar esta información de localización.

Atributos de proceso. Cualquier sistema operativo multiprogramado de la actualidad requiere una gran cantidad de información para manejar cada proceso. Como quedó explicado, esta información se puede considerar que reside en el bloque de control del proceso (BCP). Los diferentes sistemas organizarán esta información de formas diferentes. Al final de este capítulo y en el próximo se muestran varios ejemplos. Por ahora, exploraremos simplemente el tipo de información que el sistema operativo puede utilizar, sin considerar cualquier detalle de cómo esta información está organizada.

---

[7] Esta breve explicación pasa por encima algunos detalles. En particular, en un sistema que utilice memoria virtual, toda imagen de un proceso activo se encuentra siempre en memoria secundaria. Sólo una parte de la imagen se carga en memoria principal, ésta se copia en lugar de moverse. De esta forma, la memoria secundaria mantiene una copia de todos los segmentos y/o páginas. Sin embargo, si la parte de imagen del proceso en memoria principal se ve modificada, la copia en memoria secundaria estará desactualizada hasta que la parte residente en memoria principal se copie de nuevo al disco.

La Tabla 3.5 muestra la lista de los tipos de categorías de información que el sistema operativo requiere para cada proceso. Resulta sorprendente la cantidad de información necesaria para esta gestión. Cuando se vaya apreciando con mayor exactitud las responsabilidades del sistema operativo, esta lista parecerá más razonable.

Podemos agrupar la información del bloque de control del proceso en tres categorías generales:

- Identificación del proceso
- Información de estado del procesador
- Información de control del proceso

Tabla 3.5. Elementos típicos de un bloque de control del proceso (BCP).

Identificación del proceso

Identificadores

Identificadores numéricos que se pueden guardar dentro del bloque de control del proceso:

- identificadores del proceso
- identificador del proceso que creó a este proceso (proceso padre)
- identificador del usuario

Información de estado del procesador

Registros visibles por el usuario

Un registro visible por el usuario es aquel al que se puede hacer referencia por medio del lenguaje máquina que ejecuta el procesador cuando está en modo usuario. Normalmente, existen de 8 a 32 de estos registros, aunque determinadas implementaciones RISC tienen más de 100.

Registros de estado y control

Hay una gran variedad de registros del procesador que se utilizan para el control de operaciones. Estos incluyen:

- *Contador de programa*: contiene la dirección de la siguiente instrucción a ejecutar.
- *Códigos de condición*: resultan de la operación lógica o aritmética más reciente (por ejemplo, signo, cero, acarreo, igual, desbordamiento).
- *Información de estado*: incluyen los *flags* de interrupciones habilitadas/deshabilitadas, modo ejecución.

Puntero de pila

Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema. Un puntero de pila apunta a la parte más alta de la pila.

Información de control de proceso

Información de estado y de planificación

Esta es la información que el sistema operativo necesita para analizar las funciones de planificación. Elementos típicos de esta información son:

- *Estado del proceso*: indica si está listo o no el proceso para ser planificado para su ejecución (por ejemplo, Ejecutando, Listo, Esperando, Detenido).
- *Prioridad*: uno o más campos que se pueden utilizar para escribir la prioridad de planificación del proceso. En algunos sistemas, se necesitan múltiples valores (por ejemplo, por-defecto, actual, mayor-disponible).
- *Información relativa a planificación*: esto dependerá del algoritmo de planificación utilizado. Por ejemplo, la cantidad de tiempo que el proceso estaba esperando y la cantidad de tiempo que el proceso ha ejecutado la última vez que estuvo corriendo.
- *Evento*: identificar el evento al cual el proceso está esperando para continuar su ejecución.

Estructuración de datos

Un proceso puede estar enlazado con otro proceso en una cola, anillo o con cualquier otra estructura. Por ejemplo, todos los procesos en estado de espera por un nivel de prioridad en particular pueden estar enlazados en una cola. Un proceso puede mostrar una relación padre-hijo (creador-creado) con otro proceso. El bloque de control del proceso puede contener punteros a otros procesos para dar soporte a estas estructuras.

Comunicación entre procesos

Se pueden asociar diferentes *flags*, señales, y mensajes relativos a la comunicación entre dos procesos independientes. Alguna o toda esta información se puede mantener en el bloque de control de proceso (BCP).

Privilegios de proceso

Los procesos adquieren privilegios de acuerdo con la memoria a la que van a acceder y los tipos de instrucciones que se pueden ejecutar. Adicionalmente, los privilegios se pueden utilizar para usar utilidades de sistema o servicios.

Gestión de memoria

Esta sección puede incluir punteros a tablas de segmentos y/o páginas que describen la memoria virtual asignada a este proceso.

Propia de recursos y utilización

Se deben indicar los recursos controlados por el proceso, por ejemplo, ficheros abiertos. También se puede incluir un histórico de utilización del procesador o de otros recursos; esta información puede ser necesaria para el planificador.

Con respecto al **identificador de proceso**, en prácticamente todos los sistemas operativos, a cada proceso se le asocia un identificador numérico único, que puede ser simplemente un índice en la tabla de procesos principal (Figura 3.11); de otra forma, debe existir una traducción que permita al sistema operativo localizar las tablas apropiadas basándose en dicho identificador de proceso. Este identificador es útil para diferentes cuestiones. Muchas de las otras tablas controladas por el sistema operativo incluyen información que referencia a otro proceso por medio del identificador de proceso. Por ejemplo, se puedan organizar las tablas de memoria para proporcionar un mapa de la memoria principal que indique a qué proceso se tiene asignada cada región. Pueden aparecer referencias similares en las tablas de E/S o de ficheros. Cuando un proceso se comunica con otro proceso, el identificador de proceso informa al sistema operativo del destino de la comunicación. Cuando los procesos pueden crear otros procesos, los identificadores indican el proceso padre y los descendientes en cada caso.

Junto con estos identificadores de proceso, a un proceso se le puede asignar un identificador de usuario que indica qué usuario es responsable del trabajo.

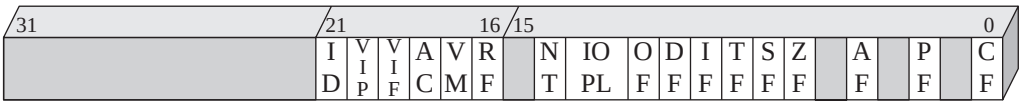
La **información de estado de proceso** indica los contenidos de los registros del procesador. Cuando un proceso está ejecutando, esta información está, por supuesto, en los registros. Cuando un proceso se interrumpe, toda la información de los registros debe salvaguardarse de forma que se pueda restaurar cuando el proceso continúe con su ejecución. La naturaleza y el número de estos registros depende del diseño del procesador. Normalmente, el conjunto de registros incluye registros visibles por usuario, registros de control y de estado, y punteros de pila. Todos estos se describieron en el Capítulo 1.

Se debe notar que, el diseño de todos los procesadores incluye un registro o conjuntos de registros, habitualmente conocidos como palabras de estado de programa (*program status word*, PSW) que contiene información de estado. La PSW típicamente contiene códigos de condición así como otra información de estado. Como ejemplo de una palabra de estado de un procesador se tiene lo que los procesadores Pentium definen como registros EFLAFS (*véase* Figura 3.12 y Tabla 3.6). Esta estructura es utilizada por los sistemas operativos (incluyendo UNIX y Windows) que se ejecuten sobre procesadores de tipo Pentium.

La tercera categoría principal de información del bloque de control de proceso se denomina, a falta de otro nombre mejor, **información de control de proceso**. Esta información adicional la necesita el sistema operativo para controlar y coordinar varios procesos activos. La última parte de la Tabla 3.5 indica el ámbito de esta información. Según examinemos los detalles de funcionamiento del sistema operativo en los sucesivos capítulos, la necesidad de algunos de los elementos de esta lista parecerá más clara.

El contenido de esta figura no debería modificarse. La traducción de los registros de control de una arquitectura en concreto no tiene mucho sentido, puesto que las siglas con las que se denotan están en inglés. El significado de dichas siglas y la traducción del término se encuentran en la siguiente tabla.

La Figura 3.13 sugiere la estructura de las imágenes de los procesos en memoria virtual. Cada imagen de proceso consiste en un bloque de control de proceso, una pila de usuario, un espacio de direcciones privadas del proceso, y cualquier otro espacio de direcciones que el proceso comparta con otros procesos. En la figura, cada imagen de proceso aparece como un rango de direcciones contiguo, pero en una implementación real, este no tiene por qué ser el caso; dependerá del esquema de gestión de memoria y de la forma en la cual se encuentren organizadas las estructuras de control por parte del sistema operativo.



ID = (Indicador de identificación)
VIP = (Interrupción virtual pendiente)
VIF = (Indicador de interrupción virtual)
AC = (Comprobación de alineamiento)
VM = (Modo virtual 8086)
RF = (Indicación de reanudación)
NT = (Indicador de tarea anidada)
IOPL = (Nivel de privilegio de E/S)
OF = (Indicador de desbordamiento)
DF = (Indicador de dirección)
IF = (Indicador de interrupciones habilitadas)
TF = (Indicador de *trap*)
SF = (Indicador de signo)
ZF = (Indicador de cero)
AF = Indicador auxiliar de acarreo)
PF = (Indicador de paridad)
CF = (Indicador de acarreo)

Figura 3.12. Registro EFLAGS de un Pentium II.

Tabla 3.6. Bits del registro EFLAGS de un Pentium.

Bits de control	
AC (*Alignment check*)	Fija si una palabra (o una doble-palabra) se encuentra direccionada en la frontera entre palabras (o dobles-palabras).
ID (*Identification flag*)	Si este bit puede ser puesto a 1 y a 0, este procesador soporta la instrucción CPUID. Esta instrucción proporciona información sobre el vendedor, familia, y modelo.
RF (*Resume flag*)	Permite al programador desactivar las extensiones de depuración de forma que la instrucción pueda volverse a ejecutar después de una excepción de depuración sin causar inmediatamente después otra excepción de depuración.
IOPL (*I/O privilege level*)	Cuando está activado, hace que el procesador genere una excepción para todos los accesos a dispositivo de E/S durante una operación en modo protegido.
DF (*Direction flag*)	Determina cuando una instrucción de procesamiento de cadenas de caracteres incrementa o decrementa los semi-registros de 16 bits SE y DI (para operaciones de 16 bits) o los registros de 32 bits ESE y EDI (para operaciones de 32 bits).
IF (*Interruption enable flag*)	Cuando está puesto a 1, el procesador reconocerá interrupciones externas.
TF (*Trap flag*)	Cuando está puesto a 1, causa una interrupción después de la ejecución de cada instrucción. Su uso está dirigido a la depuración de código.
Bits de modos de operación	
NT (*Nested task flag*)	Indica que la tarea actual está anidada dentro de otra tarea en modo de operación protegido.
VM (*Virtual 8086 mode*)	Permite al programador activar o desactivar el modo virtual 8086, que determina si el procesador funciona como si fuese una máquina 8086.
VIP (*Virtual interrupt pending*)	Usado en el modo virtual 8086 para indicar que una o más interrupciones están a la espera de servicio.
VIF (*Virtual interruption flag*)	Usado en modo virtual 8086 en lugar de IF.
Códigos de condición	
AF (*Auxiliar carry flag*)	Representa el acarreo o resto entre medios bytes de una operación aritmética o lógica de 8 bits usando el registro AL.
CF (*Carry flag*)	Indica el acarreo o el resto por medio del bit situado más a la izquierda después de una operación aritmética. También se modificaron por algunas operaciones de desplazamiento o rotación.
OF (*Overflow flag*)	Indica un desbordamiento (*overflow*) aritmético después de una suma o resta.
PF (*Parity flag*)	Paridad del resultado de una operación aritmética o lógica. 1 indica paridad par; 0 indica paridad impar.
SF (*Sign flag*)	Indica el signo del resultado de una operación aritmética o lógica.
ZF (*Zero flag*)	Indica que el resultado de una operación aritmética o lógica es 0.

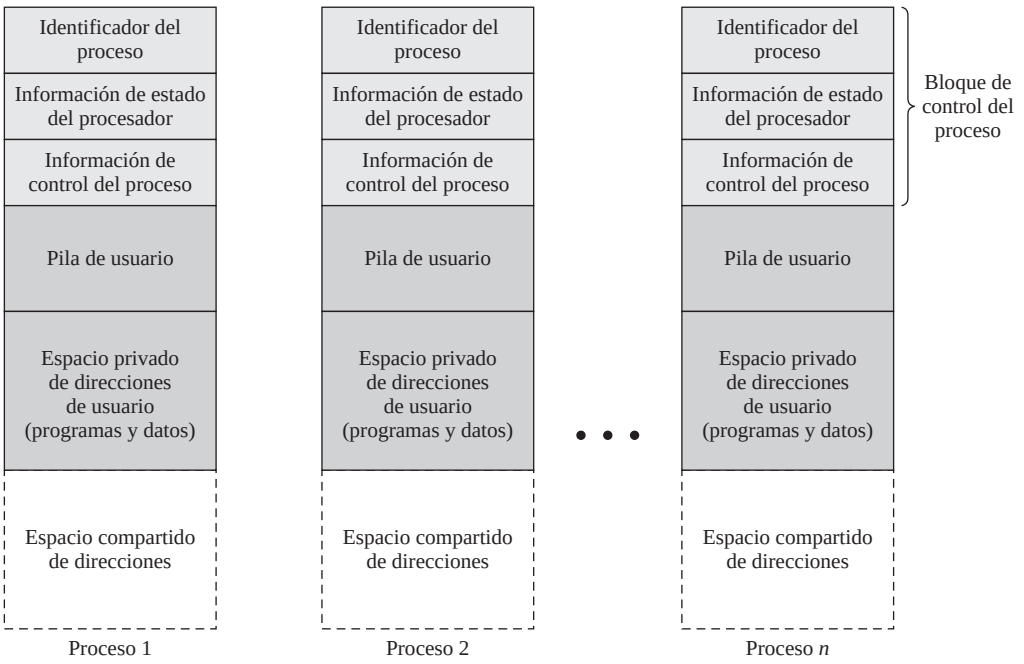


Figura 3.13. Procesos de usuario en memoria virtual.

Como indica la Tabla 3.5, un bloque de control de proceso puede contener información estructural, incluyendo punteros que permiten enlazar bloques de control de proceso entre sí. De esta forma, las colas que se describieron en la sección anterior pueden implementarse como listas enlazadas de bloques de control de proceso. Por ejemplo, la estructura de colas de la Figura 3.8a se puede implementar tal y como sugiere la Figura 3.14.

El papel del bloque de control de proceso. El bloque de control de proceso es la más importante de las estructuras de datos del sistema operativo. Cada bloque de control de proceso contiene toda la información sobre un proceso que necesita el sistema operativo. Los bloques se leen y/o se modifican por la práctica totalidad de los módulos del sistema operativo, incluyendo aquellos relacionados con la planificación, la reserva de recursos, el procesamiento entre regiones, y el análisis y monitorización del rendimiento. Se puede decir que el conjunto de los bloques de control de proceso definen el estado del sistema operativo.

Esto muestra un aspecto importante en el diseño. Un gran número de rutinas dentro del sistema operativo necesitarán acceder a información de los bloques de control de proceso. Proporcionar acceso directo a estas tablas no es difícil. Cada proceso lleva asociado un único identificador, que puede utilizarse para indexar dentro de una tabla de punteros a bloques de control de proceso. La dificultad no reside en el acceso sino en la protección. Dos posibles problemas se pueden presentar son:

- Un fallo en una simple rutina, como un manejador de interrupción, puede dañar los bloques de control de proceso, que puede destruir la capacidad del sistema para manejar los procesos afectados.
- Un cambio en la estructura o en la semántica de los bloques de control de procesos puede afectar a un gran número de módulos del sistema operativo.

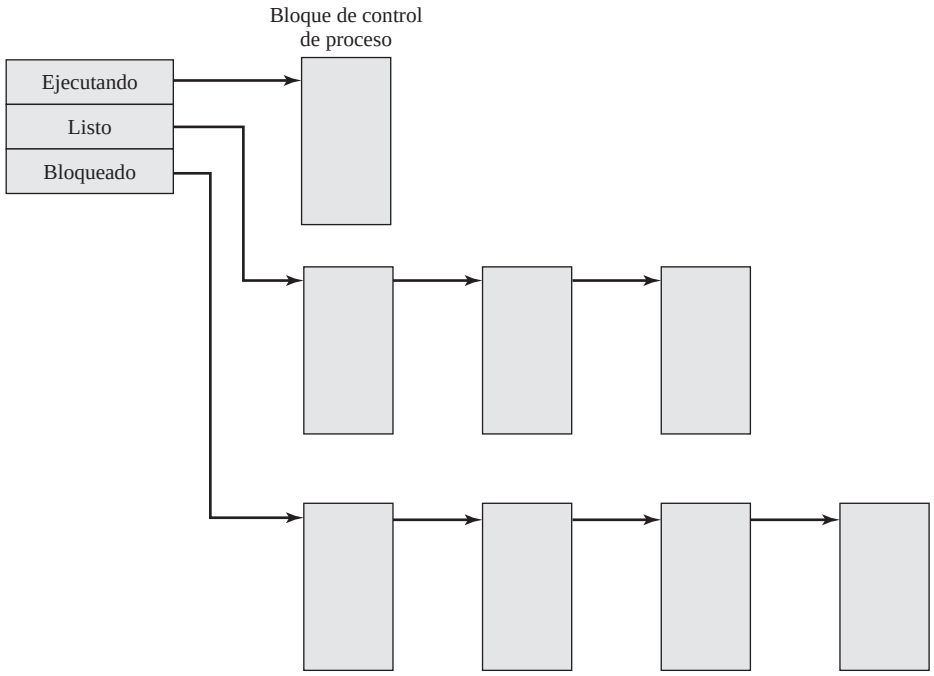


Figura 3.14. Estructuras de listas de procesos.

Estos problemas se pueden ser tratar obligando a que todas rutinas del sistema operativo pasen a través de una rutina de manejador, cuyo único trabajo es proteger a los bloques de control de proceso, y que es la única que realiza el arbitraje en las operaciones de lectura y escritura de dichos bloques. Los factores a equilibrar en el uso de está rutina son, por un lado los aspectos de rendimiento, y por el otro, el grado en el cual el resto del software del sistema puede verificarse para determinar su corrección.

3.4. CONTROL DE PROCESOS

MODOS DE EJECUCIÓN

Antes de continuar con nuestra explicación sobre cómo el sistema operativo gestiona los procesos, necesitamos distinguir entre los modos de ejecución del procesador, normalmente asociados con el sistema operativo y los asociados con los programas de usuario. Muchos procesadores proporcionan al menos dos modos de ejecución. Ciertas instrucciones se pueden ejecutar en modos privilegiados únicamente. Éstas incluirían lectura y modificación de los registros de control, por ejemplo la palabra de estado de programa; instrucciones de E/S primitivas; e instrucciones relacionadas con la gestión de memoria. Adicionalmente, ciertas regiones de memoria sólo se pueden acceder en los modos más privilegiados.

El modo menos privilegiado a menudo se denomina **modo usuario**, porque los programas de usuario típicamente se ejecutan en este modo. El modo más privilegiado se denomina **modo sistema**, **modo control** o **modo núcleo**. Este último término se refiere al núcleo del sistema operativo, que es la parte del sistema operativo que engloba las funciones más importantes del sistema. La Tabla 3.7 lista las funciones que normalmente se encuentran dentro del núcleo del sistema operativo.

Tabla 3.7. Funciones típicas de un núcleo de sistema operativo.

Gestión de procesos

- Creación y terminación de procesos
- Planificación y activación de procesos
- Intercambio de procesos
- Sincronización de procesos y soporte para comunicación entre procesos
- Gestión de los bloques de control de proceso

Gestión memoria

- Reserva de espacio direcciones para los procesos
- *Swapping*
- Gestión de páginas y segmentos

Gestión E/S

- Gestión de *buffers*
- Reserva de canales de E/S y de dispositivos para los procesos

Funciones de soporte

- Gestión de interrupciones
- Auditoría
- Monitorización

El motivo por el cual se usan los otros modos es claro. Se necesita proteger al sistema operativo y a las tablas clave del sistema, por ejemplo los bloques de control de proceso, de la interferencia con programas de usuario. En modo núcleo, el software tiene control completo del procesador y de sus instrucciones, registros, y memoria. Este nivel de control no es necesario y por seguridad no es recomendable para los programas de usuario.

Aparecen dos cuestiones: ¿cómo conoce el procesador en que modo está ejecutando y cómo este modo puede modificarse? En lo referente la primera cuestión, existe típicamente un bit en la palabra de estado de programa (PSW) que indica el modo de ejecución. Este bit se cambia como respuesta a determinados eventos. Habitualmente, cuando un usuario realiza una llamada a un servicio del sistema operativo o cuando una interrupción dispara la ejecución de una rutina del sistema operativo, este modo de ejecución se cambia a modo núcleo y; tras la finalización del servicio, el modo se fija de nuevo a modo usuario. Por ejemplo, si consideramos el procesador Intel Itanium, que implementa la arquitectura IA-64 de 64 bits, el procesador tiene un registro de estado (psr) que incluye un campo de los 2 bits llamado cpl (*current privilege level*). El Nivel 0 es en el nivel con más privilegios mientras que el Nivel 3 es el nivel con menos privilegios. La mayoría de los sistemas operativos, como Linux, utilizar el Nivel 0 para el núcleo y uno de los otros niveles como nivel de usuario. Cuando llega una interrupción, el procesador limpia la mayor parte de los bits en psr, incluyendo el campo cpl. Esto automáticamente pone en 0 el cpl. Al final de la rutina de manejo de interrupción, la última instrucción que se ejecuta es irt (interrupt return). Esta instrucción hace que el procesador restaure el psr del programa interrumpido, que restaura a su vez el nivel de privilegios de dicho programa. Una secuencia similar ocurre cuando la aplicación solicita una llamada al sistema. Para los Itanium, una aplicación solicita la llamada sistema indicando un identificador de llamada de sistema

y los argumentos de dicha llamada en unas áreas definidas y posteriormente ejecutando una instrucción especial que se puede invocar en modo usuario y que tiene como objeto transferir el control al núcleo.

CREACIÓN DE PROCESOS

En la Sección 3.1 hemos comentado los eventos que llevará creación de uno proceso. Una vez vistas las estructuras de datos asociadas a un proceso, ahora posiciones está en posición de describir brevemente los pasos que llevan a la verdadera creación de un proceso.

Una vez que el sistema operativo decide, por cualquier motivo (Tabla 3.1), crear un proceso procederá de la siguiente manera:

1. **Asignar un identificador de proceso único al proceso**. En este instante, se añade una nueva entrada a la tabla primaria de procesos, que contiene una entrada por proceso.
2. **Reservar espacio para proceso**. Esto incluye todos los elementos de la imagen del proceso. Para ello, el sistema operativo debe conocer cuánta memoria se requiere para el espacio de direcciones privado (programas y datos) y para la pila de usuario. Estos valores se pueden asignar por defecto basándonos en el tipo de proceso, o pueden fijarse en base a la solicitud de creación del trabajo remitido por el usuario. Si un proceso es creado por otro proceso, el proceso padre puede pasar los parámetros requeridos por el sistema operativo como parte de la solicitud de la creación de proceso. Si existe una parte del espacio direcciones compartido por este nuevo proceso, se fijan los enlaces apropiados. Por último, se debe reservar el espacio para el bloque de control de proceso (BCP).
3. **Inicialización del bloque de control de proceso**. La parte de identificación de proceso del BCP contiene el identificador del proceso así como otros posibles identificadores, tal como el indicador del proceso padre. En la información de estado de proceso del BCP, habitualmente se inicializa con la mayoría de entradas a 0, excepto el contador de programa (fijado en el punto entrada del programa) y los punteros de pila de sistema (fijados para definir los límites de la pila del proceso). La parte de información de control de procesos se inicializa en base a los valores por omisión, considerando también los atributos que han sido solicitados para este proceso. Por ejemplo, el estado del proceso se puede inicializar normalmente a Listo o Listo/Suspendido. La prioridad se puede fijar, por defecto, a la prioridad más baja, a menos que una solicitud explicita la eleve a una prioridad mayor. Inicialmente, el proceso no debe poseer ningún recurso (dispositivos de E/S, ficheros) a menos que exista una indicación explícita de ello o que haya sido heredados del padre.
4. **Establecer los enlaces apropiados**. Por ejemplo, si el sistema operativo mantiene cada cola del planificador como una lista enlazada, el nuevo proceso debe situarse en la cola de Listos o en la cola de Listos/Suspendidos.
5. **Creación o expansión de otras estructuras de datos**. Por ejemplo, el sistema operativo puede mantener un registro de auditoría por cada proceso que se puede utilizar posteriormente a efectos de facturación y/o de análisis de rendimiento del sistema.

CAMBIO DE PROCESO

A primera vista, la operación de cambio de proceso puede parecer sencilla. En algunos casos, un proceso en ejecución se interrumpe para que el sistema operativo asigne a otro proceso el estado de Ejecutando y de esta forma establecer el turno entre los procesos. Sin embargo, es necesario tener en

cuenta algunas consideraciones de diseño. Primero, ¿qué evento dispara los cambios de proceso? Otra consideración que se debe tener en cuenta es la distinción entre cambio de proceso (*process switching*) y cambio de modo (*mode switching*). Por último, ¿qué debe hacer el sistema operativo con las diferentes estructuras que gestiona para proporcionar un cambio de proceso?

Cuándo se realiza el cambio de proceso. Un cambio de proceso puede ocurrir en cualquier instante en el que el sistema operativo obtiene el control sobre el proceso actualmente en ejecución. La Tabla 3.8 indica los posibles momentos en los que el sistema operativo puede tomar dicho control.

Primero, consideremos las interrupciones del sistema. Realmente, podemos distinguir, como hacen muchos sistemas, dos tipos diferentes de interrupciones de sistema, unas simplemente denominadas interrupciones, y las otras denominadas *traps*. Las primeras se producen por causa de algún tipo de evento que es externo e independiente al proceso actualmente en ejecución, por ejemplo la finalización de la operación de E/S. Las otras están asociadas a una condición de error o excepción generada dentro del proceso que está ejecutando, como un intento de acceso no permitido a un fichero. Dentro de una **interrupción** ordinaria, el control se transfiere inicialmente al manejador de interrupción, que realiza determinadas tareas internas y que posteriormente salta a una rutina del sistema operativo, encargada de cada uno de los tipos de interrupciones en particular. Algunos ejemplos son:

- **Interrupción de reloj.** El sistema operativo determinar si el proceso en ejecución ha excedido o no la unidad máxima de tiempo de ejecución, denominada **rodaja de tiempo** (*time slice*). Esto es, una rodaja de tiempo es la máxima cantidad de tiempo que un proceso puede ejecutar antes de ser interrumpido. En dicho caso, este proceso se puede pasar al estado de Listo y se puede activar otro proceso.
- **Interrupción de E/S.** El sistema operativo determina qué acción de E/S ha ocurrido. Si la acción de E/S constituye un evento por el cual están esperando uno o más procesos, el sistema operativo mueve todos los procesos correspondientes al estado de Listos (y los procesos en estado Bloqueado/Suspendido al estado Listo/Suspendido). El sistema operativo puede decidir si reanuda la ejecución del proceso actualmente en estado Ejecutando o si lo expulsa para proceder con la ejecución de un proceso Listo de mayor prioridad.

Tabla 3.8. Mecanismos para la interrupción de la ejecución de un proceso.

Mecanismo	Causa	Uso
Interrupción	Externa a la ejecución del proceso actualmente en ejecución.	Reacción ante un evento externo asíncrono.
Trap	Asociada a la ejecución de la instrucción actual.	Manejo de una condición de error o de excepción.
Llamada al sistema	Solicitud explícita.	Llamada a una función del sistema operativo.

- **Fallo de memoria.** El procesador se encuentra con una referencia a una dirección de memoria virtual, a una palabra que no se encuentra en memoria principal. El sistema operativo debe traer el bloque (página o segmento) que contiene la referencia desde memoria secundaria a memoria principal. Después de que se solicita la operación de E/S para traer el bloque a memoria, el proceso que causó el fallo de memoria se pasa al estado de Bloqueado; el sistema operativo realiza un cambio de proceso y pone a ejecutar a otro proceso. Después de que el bloque de memoria solicitado se haya traído, el proceso pasará al estado Listo.

Con un ***trap***, el sistema operativo conoce si una condición de error o de excepción es irreversible. Si es así, el proceso en ejecución se pone en el estado Saliente y se hace un cambio de proceso. De no ser así, el sistema operativo actuará dependiendo de la naturaleza del error y su propio diseño. Se puede intentar un procedimiento de recuperación o simplemente la notificación al usuario, pudiendo implicar tanto un cambio de proceso como la continuación de la ejecución del proceso actual.

Por último, el sistema operativo se puede activar por medio de una **llamada al sistema** procedente del programa en ejecución. Por ejemplo, si se está ejecutando un proceso y se ejecuta una operación que implica una llamada de E/S, como abrir un archivo. Esta llamada implica un salto a una rutina que es parte del código del sistema operativo. La realización de una llamada al sistema puede implicar en algunos casos que el proceso que la realiza pase al estado de Bloqueado.

Cambio de modo. En el Capítulo 1, discutimos la inclusión de una fase de interrupción como parte del ciclo de una instrucción. Recordando esto, en la fase de interrupción, el procesador comprueba que no exista ninguna interrupción pendiente, indicada por la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador pasa a la fase de búsqueda de instrucción, siguiendo con el programa del proceso actual. Si hay una interrupción pendiente, el proceso actúa de la siguiente manera:

1. Coloca el contador de programa en la dirección de comienzo de la rutina del programa manejador de la interrupción.
2. Cambia de modo usuario a modo núcleo de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

El procesador, acto seguido, pasa a la fase de búsqueda de instrucción y busca la primera instrucción del programa de manejo de interrupción, que dará servicio a la misma. En este punto, habitualmente, el contexto del proceso que se ha interrumpido se salvaguarda en el bloque de control de proceso del programa interrumpido.

Una pregunta que se puede plantear es, ¿qué constituye el contexto que se debe salvaguardar? La respuesta es que se trata de toda la información que se puede ver alterada por la ejecución de la rutina de interrupción y que se necesitará para la continuación del proceso que ha sido interrumpido. De esta forma, se debe guardar la parte del bloque de control del proceso que hace referencia a la información de estado del procesador. Esto incluye el contador de programa, otros registros del procesador, y la información de la pila.

¿Se necesita hacer algo más? Eso depende de qué ocurra luego. El manejador de interrupción es habitualmente un pequeño programa que realiza unas pocas tareas básicas relativas a la interrupción. Por ejemplo, borra el *flag* o indicador que señala la presencia de interrupciones. Puede enviar una confirmación a la entidad que lanzó dicha interrupción, como por ejemplo el módulo de E/S. Y puede realizar algunas tareas internas variadas relativas a los efectos del evento que causó la interrupción. Por ejemplo, si la interrupción se refiere a un evento de E/S, el manejador de interrupción comprobará la existencia o no de una condición de error. Si ha ocurrido un error, el manejador mandará una señal al proceso que solicitó dicha operación de E/S. Si la interrupción proviene del reloj, el manejador la va a pasar el control al activador, el cual decidirá pasar a otro proceso debido a que la rodaja de tiempo asignada a ese proceso ha expirado.

¿Qué pasa con el resto de información del bloque de control de proceso? Si a esta interrupción le sigue un cambio de proceso a otro proceso, se necesita hacer algunas cosas más. Sin embargo, en muchos sistemas operativos, la existencia de una interrupción no implica necesariamente un cambio de proceso. Es posible, por tanto, que después de la ejecución de la rutina de interrupción, la ejecución se reanude con el mismo proceso. En esos casos sólo se necesita salvaguardar la información del estado del procesador cuando se produce la interrupción y restablecerlo cuando se reanude la

ejecución del proceso. Habitualmente, las operaciones de salvaguarda y recuperación se realizan por hardware.

Cambio del estado del proceso. Esta claro, por tanto, que el cambio de modo es un concepto diferente del cambio de proceso[8]. Un cambio de modo puede ocurrir sin que se cambie el estado del proceso actualmente en estado Ejecutando. En dicho caso, la salvaguarda del estado y su posterior restauración comportan sólo una ligera sobrecarga. Sin embargo, si el proceso actualmente en estado Ejecutando, se va a mover a cualquier otro estado (Listo, Bloqueado, etc.), entonces el sistema operativo debe realizar cambios sustanciales en su entorno. Los pasos que se realizan para un cambio de proceso completo son:

1. Salvar el estado del procesador, incluyendo el contador de programa y otros registros.
2. Actualizar el bloque de control del proceso que está actualmente en el estado Ejecutando. Esto incluye cambiar el estado del proceso a uno de los otros estados (Listo, Bloqueado, Listo/Suspendido, o Saliente). También se tienen que actualizar otros campos importantes, incluyendo la razón por la cual el proceso ha dejado el estado de Ejecutando y demás información de auditoría.
3. Mover el bloque de control de proceso a la cola apropiada (Listo, Bloqueado en el evento *i*, Listo/Suspendido).
4. Selección de un nuevo proceso a ejecutar; esta cuestión se analiza con más detalle en la Parte Cuatro.
5. Actualizar el bloque de control del proceso elegido. Esto incluye pasarlo al estado Ejecutando.
6. Actualizar las estructuras de datos de gestión de memoria. Esto se puede necesitar, dependiendo de cómo se haga la traducción de direcciones; estos aspectos se cubrirán en la Parte Tres.
7. Restaurar el estado del procesador al que tenía en el momento en el que el proceso seleccionado salió del estado Ejecutando por última vez, leyendo los valores anteriores de contador de programa y registros.

Por tanto, el cambio de proceso, que implica un cambio en el estado, requiere un mayor esfuerzo que un cambio de modo.

EJECUCIÓN DEL SISTEMA OPERATIVO

En el Capítulo 2, señalamos dos aspectos intrínsecos del sistema operativo:

- El sistema operativo funciona de la misma forma que cualquier software, en el sentido que un sistema operativo es un conjunto de programas ejecutados por un procesador.
- El sistema operativo, con frecuencia, cede el control y depende del procesador para recuperar dicho control.

---

[8] El término **cambio de contexto** (*context switch*) se usa en literatura sobre sistemas operativos y en libros de texto. Desafortunadamente, aunque la mayoría de estas referencias utilizan este término para lo que en este libro se ha denominado cambio de proceso, otras fuentes lo usan para referirse al cambio de modo o incluso al cambio de thread (definido en el siguiente capítulo). Para evitar ambigüedades, este término no se ha utilizado en este libro.

Si el sistema operativo es simplemente una colección de programas y si son ejecutados por el procesador, tal y como cualquier otro programa, ¿es el sistema operativo un proceso del sistema? y si es así ¿cómo se controla? Estas interesantes cuestiones han inspirado unas cuentas direcciones diferentes de diseño. La Figura 3.15 ilustra el rango de diferentes visiones que se encuentran en varios sistemas operativos contemporáneos.

Núcleo sin procesos. Una visión tradicional, que es común a muchos sistemas operativos antiguos, es la ejecución del sistema operativo fuera de todo proceso (Figura 3.15a). Con esta visión, cuando el proceso en ejecución se interrumpe o invoca una llamada al sistema, el contexto se guarda y el control pasa al núcleo. El sistema operativo tiene su propia región de memoria y su propia pila de sistema para controlar la llamada a procedimientos y sus retornos. El sistema operativo puede realizar todas las funciones que necesite y restaurar el contexto del proceso interrumpido, que hace que se retome la ejecución del proceso de usuario afectado. De forma alternativa, el sistema operativo puede realizar la salvaguarda del contexto y la activación de otro proceso diferente. Si esto ocurre o no depende de la causa de la interrupción y de las circunstancias puntuales en el momento.

En cualquier caso, el punto clave en este caso es que el concepto de proceso se aplica aquí únicamente a los programas de usuario. El código del sistema operativo se ejecuta como una entidad independiente que requiere un modo privilegiado de ejecución.

Ejecución dentro de los procesos de usuario. Una alternativa que es común en los sistemas operativos de máquinas pequeñas (PC, estaciones de trabajo) es ejecutar virtualmente todo el software de sistema operativo en el contexto de un proceso de usuario. Esta visión es tal que el sistema operativo se percibe como un conjunto de rutinas que el usuario invoca para realizar diferentes funciones, ejecutadas dentro del entorno del proceso de usuario. Esto se muestra en la Figura 3.15b. En un punto determinado, el sistema operativo maneja *n* imágenes de procesos. Cada imagen incluye no

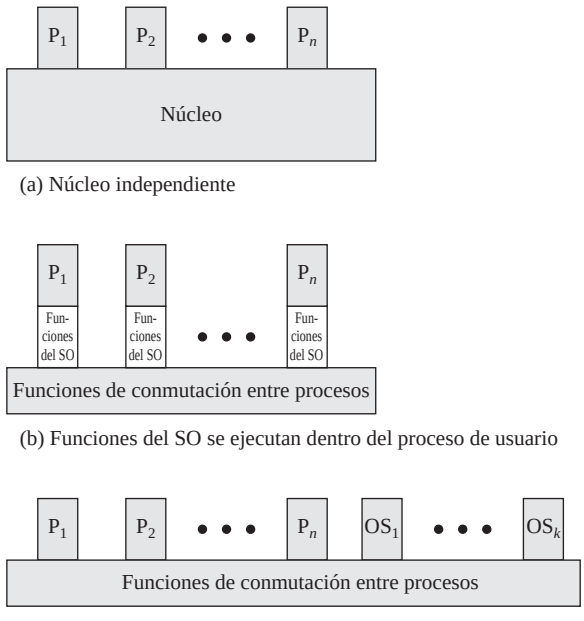


Figura 3.15. Relación entre el sistema operativo y los procesos de usuario.

sólo las regiones mostradas en la Figura 3.13, sino que además incluye áreas de programa, datos y pila para los programas del núcleo.

La Figura 3.16 sugiere la estructura de imagen de proceso típica para esta estrategia. Se usa una pila de núcleo separada para manejar llamadas/retornos cuando el proceso está en modo núcleo. El código de sistema operativo y sus datos están en el espacio de direcciones compartidas y se comparten entre todos los procesos.

Cuando ocurre una interrupción, *trap* o llamada al sistema, el procesador se pone en modo núcleo y el control se pasa al sistema operativo. Para este fin, el contexto se salva y se cambia de modo a una rutina del sistema operativo. Sin embargo, la ejecución continúa dentro del proceso de usuario actual. De esta forma, no se realiza un cambio de proceso, sino un cambio de modo dentro del mismo proceso.

Si el sistema operativo, después de haber realizado su trabajo, determina que el proceso actual debe continuar con su ejecución, entonces el cambio de modo continúa con el programa interrumpido, dentro del mismo proceso. Ésta es una de las principales ventajas de esta alternativa: se ha interrumpido un programa de usuario para utilizar alguna rutina del sistema operativo, y luego continúa, y todo esto se ha hecho sin incurrir en un doble cambio de proceso. Sin embargo, si se determina que se debe realizar un cambio de proceso en lugar de continuar con el proceso anterior, entonces el control pasa a la rutina de cambio de proceso. Esta rutina puede o no ejecutarse dentro del proceso actual, dependiendo del diseño del sistema. En algún momento, no obstante, el proceso actual se debe poner en un estado de no ejecución, designando a otro porceso como el siguiente a

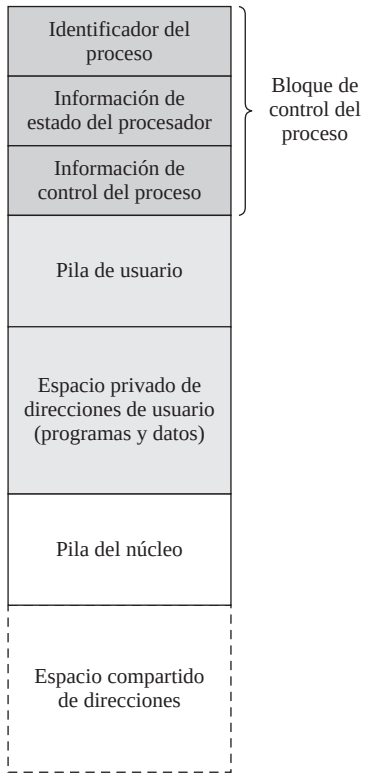


Figura 3.16. Imagen de proceso: el sistema operativo ejecuta dentro del espacio de usuario.

ejecutar. Durante esta fase, es más conveniente ver la ejecución como fuera de cualquiera de los procesos.

De alguna forma, esta visión de un sistema operativo es muy reseñable. De una forma más sencilla, en determinados instantes de tiempo, un proceso salva su estado, elige otro proceso a ejecutar entre aquellos que están listos, y delega el control a dicho proceso. La razón por la cual este esquema no se convierte en una situación arbitraria y caótica es que durante los instantes críticos el código que se está ejecutando es código de compartido de sistema operativo, no código de usuario. Debido al concepto de modo usuario y modo núcleo, el usuario no puede interferir con las rutinas del sistema operativo, incluso cuando se están ejecutando dentro del entorno del proceso del usuario. Esto nos recuerda que existe una clara distinción entre el concepto de proceso y programa y que la relación entre los dos no es uno a uno. Dentro de un proceso, pueden ejecutarse tanto un programas de usuario como programas del sistema operativo. Los programas de sistema operativo que se ejecutan en varios procesos de usuario son idénticos.

Sistemas operativos basados en procesos. Otra alternativa, mostrada en la Figura 3.15c, es implementar un sistema operativo como una colección de procesos de sistema. Como en las otras opciones, el software que es parte del núcleo se ejecuta en modo núcleo. En este caso, las principales funciones del núcleo se organizan como procesos independientes. De nuevo, debe haber una pequeña cantidad de código para intercambio de procesos que se ejecuta fuera de todos los procesos.

Esta visión tiene diferentes ventajas. Impone una disciplina de diseño de programas que refuerza el uso de sistemas operativos modulares con mínimas y claras interfaces entre los módulos. Adicionalmente, otras funciones del sistema operativo que no sean críticas están convenientemente separadas como otros procesos. Por ejemplo, hemos mencionado anteriormente un programa de monitorización que recoge niveles de utilización de diferentes recursos (procesador, memoria, canales) y el ratio de progreso de los procesos en el sistema. Debido a que este programa no realiza ningún servicio a ningún otro programa activo, sólo se puede invocar por el sistema operativo. Como proceso, esta función puede ejecutarse a un nivel de prioridad determinado, intercalándose con otros procesos bajo el control del activador. Por último, la implementación del sistema operativo como un grupo de procesos en entornos de multiprocesadores y multicomputadores, en los cuales determinados servicios del sistema operativo se pueden enviar a procesadores dedicados, incrementando el rendimiento.

3.5. UNIX SVR4 PROCESS MANAGEMENT

UNIX System V hace uso de una gestión de procesos simple pero muy potente que es fácilmente visible a nivel de usuario. UNIX sigue el modelo mostrado en la Figura 3.15b, en la cual la gran mayoría del sistema operativo se ejecuta dentro del entorno del proceso. De esta forma se necesitan dos modos, usuario y núcleo. UNIX utiliza también dos categorías de procesos, procesos de sistema y procesos de usuario. Los procesos del sistema ejecutan en modo núcleo y ejecuta código del sistema operativo para realizar tareas administrativas o funciones internas, como reserva de memoria o *swapping* de procesos. Los procesos de usuario operan en modo usuario para ejecutar programas y utilidades y en modo núcleo para ejecutar instrucciones que pertenecen al núcleo. Un proceso de usuario entra en modo núcleo por medio de la solicitud de la llamada al sistema, cuando se genera una de excepción (fallo) o cuando ocurre una interrupción.

ESTADOS DE LOS PROCESOS

En los sistemas operativos UNIX se utilizan un total de nueve estados de proceso; estos se encuentran recogidos en la Tabla 3.9 y el diagrama de transiciones se muestra en la Figura 3.17 (basada en figura

en [BACH86]). Esta figura es similar a la Figura 3.9b, con dos estados de procesos dormidos, propios de UNIX, correspondientes a dos estados de bloqueo. Las diferencias se pueden resumir en:

- UNIX utiliza dos estados Ejecutando que indican si el proceso está ejecutando en modo usuario o en modo núcleo.
- Se debe realizar la distinción entre dos estados: (Listo para Ejecutar, en Memoria) y (Expulsado). Estos son esencialmente el mismo estado, como indica la línea punteada que los une. La distinción se realiza para hacer énfasis en la forma mediante la cual se llega al estado de Expulsado. Cuando un proceso ejecuta en modo núcleo (como resultado de una llamada al sistema, interrupción de reloj o interrupción de E/S), requerirá un tiempo para que el sistema operativo complete su trabajo y esté listo para devolver el control al proceso de usuario. En este punto, el núcleo decide expulsar al proceso actual en favor de uno de los procesos listos de mayor prioridad. En este caso, el proceso actual se mueve al estado Expulsado. Sin embargo, por cuestiones de activación, estos procesos en el estado Expulsado y todos aquellos en los estados Listo para Ejecutar en Memoria, forman una única cola.

La expulsión sólo puede ocurrir cuando un proceso está a punto de moverse de modo núcleo a modo usuario. Mientras los procesos ejecutan al modo núcleo, no pueden ser expulsados. Esto hace que los sistemas UNIX no sean apropiados para procesamiento en tiempo real. Se proporcionará una explicación de los requisitos para procesamiento de tiempo-real en el Capítulo 10.

En UNIX existen dos procesos con un interés particular. El proceso 0 es un proceso especial que se crea cuando el sistema arranca; en realidad, es una estructura de datos predefinida que se carga en cuanto el ordenador arranca. Es el proceso *swapper*. Adicionalmente, el proceso 0 lanza al proceso 1, que se denomina proceso *init*; todos los procesos del sistema tienen al proceso 1 como antecesor. Cuando un nuevo usuario interactivo quiere entrar en el sistema, es el proceso 1 el que crea un proceso de usuario para él. Posteriormente, el proceso del usuario puede crear varios procesos hijos que conforman una estructura de árbol de procesos, de esta forma cualquier aplicación en particular puede consistir en un número de procesos relacionados entre sí.

Tabla 3.9. Estados de procesos en UNIX.

Ejecutando Usuario	Ejecutando en modo usuario.
Ejecutando Núcleo	Ejecutando en modo núcleo.
Listo para Ejecutar, en Memoria	Listo para ejecutar tan pronto como el núcleo lo planifique.
Dormido en Memoria	No puede ejecutar hasta que ocurra un evento; proceso en memoria principal (estado de bloqueo).
Listo para Ejecutar, en *Swap*	El proceso está listo para preguntar, pero el *swapper* debe cargar el proceso en memoria principal antes de que el núcleo pueda planificarlo para su ejecución.
Durmiendo, en *Swap*	El proceso está esperando un evento y ha sido expulsado al almacenamiento secundario (estado de bloqueo).
Expulsado	El proceso ha regresado de modo núcleo a modo usuario, pero el núcleo lo ha expulsado y ha realizado la activación de otro proceso.
Creado	El proceso ha sido creado recientemente y aún no está listo para ejecutar.
Zombie	El proceso ya no existe, pero deja un registro para que lo recoja su proceso padre.

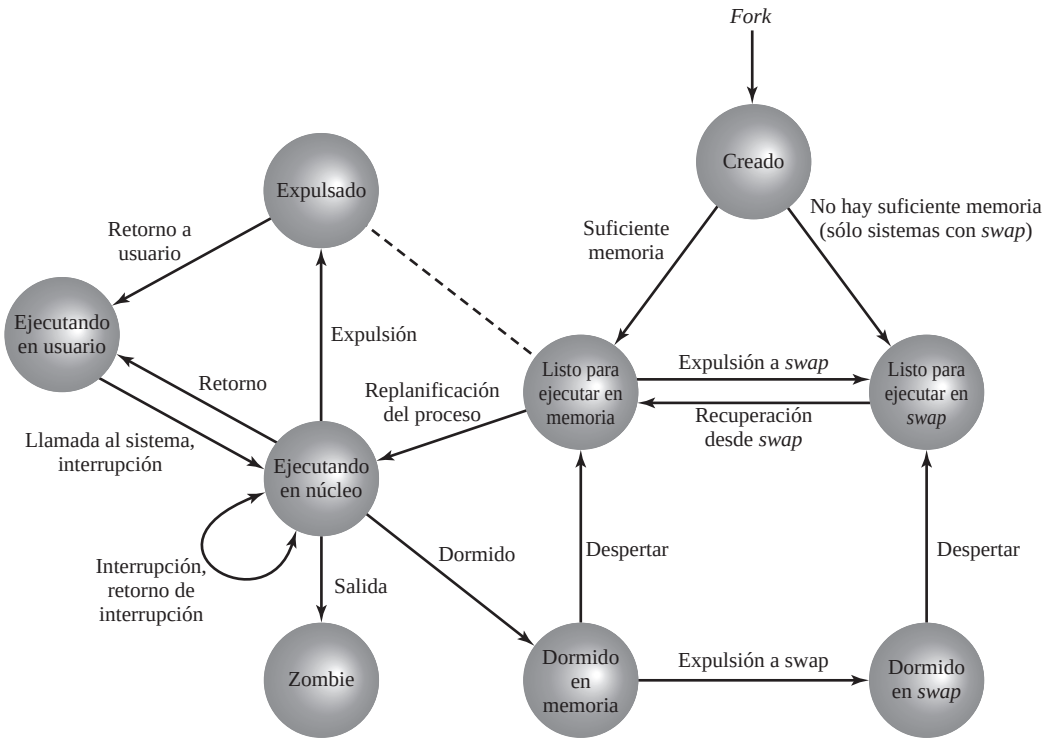


Figura 3.17. Diagrama de transiciones entre estados de procesos UNIX.

DESCRIPCIÓN DE PROCESOS

Un proceso UNIX es un conjunto de estructuras de datos, más bien complejas, que proporcionan al sistema operativo toda la información necesaria para manejar y activar los procesos. La Tabla 3.10 recoge los elementos de la imagen de proceso, que están organizados en tres partes: contexto a nivel de usuario, contexto de registros, y contexto a nivel de sistema.

El **contexto a nivel de usuario** contiene los elementos básicos de un programa de usuario que se pueden generar directamente desde un fichero objeto compilado. Un programa de usuario se divide en áreas de texto y datos; el área texto es de sólo-lectura y se crea con la intención de contener las instrucciones del programa. Cuando el proceso está en ejecución, el procesador utiliza el área de pila de usuario para gestionar las llamadas a procedimientos y sus retornos, así como los parámetros pasados. El área de memoria compartida es un área de datos que se comparte con otros procesos. Sólo existe una única copia física del área de memoria compartida, pero, por medio de la utilización de la memoria virtual, se presenta a cada uno de los procesos que comparten esta región de memoria común dentro de su espacio de dirección. Cuando un proceso está ejecutando, la información de estado de procesador se almacena en el área de **contexto de registros**.

El **contexto a nivel de sistema** contiene la información restante que necesita el sistema operativo para manejar el proceso. Consiste en una parte estática, de tamaño fijo y que permanece como parte del proceso a lo largo de todo su tiempo de vida, y una parte dinámica, que varía de tamaño a lo largo de la vida del proceso. La entrada de la tabla de procesos es un elemento de la parte estática y contiene información de control del proceso que es accesible por parte del núcleo

en todo momento; además, en un sistema de memoria virtual, todas las entradas en la tabla de procesos se mantienen en memoria principal. La Tabla 3.11 muestra los contenidos de la entrada de la tabla de procesos. El área de usuario, o área U, contiene información adicional de proceso que necesita el núcleo cuando está ejecutando en el contexto de este proceso; también se utiliza cuando el proceso se pagina desde/hacia memoria (*swapping*). La Tabla 3.12 muestra los contenidos de dicha tabla.

Tabla 3.10. Imagen de un proceso UNIX.

	Contexto a nivel de usuario
Texto	Instrucciones máquina ejecutables del programa.
Datos	Datos accesibles por parte del programa asociado a dicho proceso.
Pila de usuario	Contiene los argumentos, las variables locales, y los punteros a funciones ejecutadas en modo usuario.
Memoria compartida	Memoria compartida con otros procesos, usada para la comunicación entre procesos.
	Contexto de registros
Contador de programa	Dirección de la siguiente instrucción a ejecutar; puede tratarse del espacio de memoria del núcleo o de usuario de dicho proceso.
Registro de estado del procesador	Contiene el estado del hardware del procesador en el momento de la expulsión; los contenidos y el formato dependen específicamente del propio hardware.
Puntero de pila	Apunta a la cima de la pila de núcleo o usuario, dependiendo del modo de operación en el momento de la expulsión.
Registros de propósito general	Depende del hardware.
	Contexto nivel de sistema
Entrada en la tabla de procesos	Define el estado del proceso; esta información siempre está accesible por parte de sistema operativo.
Área U (de usuario)	Información de control del proceso que sólo se necesita acceder en el contexto del propio proceso.
Tabla de regiones por proceso	Define la traducción entre las direcciones virtuales y físicas; también contiene información sobre los permisos que indican el tipo de acceso permitido por parte del proceso; sólo-lectura, lectura-escritura, o lectura-ejecución.
Pila del núcleo	Contiene el marco de pila de los procedimientos del núcleo cuando el proceso ejecuta en modo núcleo.

Tabla 3.11. Entrada en la tabla de procesos UNIX.

Estado del proceso	Estado actual del proceso.
Punteros	Al área U y al área de memoria del proceso (texto, datos, pila).
Tamaño de proceso	Permite al sistema operativo conocer cuánto espacio está reservado para este proceso.
Identificadores de usuario	El **ID de usuario real** identifica el usuario que es responsable de la ejecución del proceso. El **ID de usuario efectivo** se puede utilizar para que el proceso gane, de forma temporal, los privilegios asociados a un programa en particular; mientras ese programa se ejecuta como parte del proceso, el proceso opera con el identificador de usuario efectivo.
Identificadores de proceso	Identificador de este proceso; identificador del proceso padre. Estos identificadores se fijan cuando el proceso entra en el estado Creado después de la llamaba al sistema *fork*.
Descriptor de evento	Válido cuando el proceso está en un estado dormido; cuando el evento ocurre, el proceso se mueve al estado Listo para Ejecutar.
Prioridad	Utilizado en la planificación del proceso.
Señal	Enumera las señales enviadas a este proceso pero que no han sido aún manejadas.
Temporizadores	Incluye el tiempo de ejecución del proceso, la utilización de recursos de núcleo, y el temporizador fijado por el usuario para enviar la señal de alarma al proceso.
P_link	Puntero al siguiente enlace en la cola de Listos (válido si proceso está Listo para Ejecutar).
Estado de memoria	Indica si la imagen del proceso se encuentra en memoria principal o secundaria. Si está en memoria, este campo indica si puede ser expulsado a *swap* o si está temporalmente fijado en memoria principal.

La distinción entre la entrada de la tabla de procesos y el área U refleja el hecho de que el núcleo de UNIX siempre ejecuta en el contexto de un proceso. La mayor parte del tiempo, el núcleo está realizando tareas relacionadas con dicho proceso. Sin embargo, otra parte del tiempo, como cuando el núcleo está realizando tareas de planificación preparatorias para la activación de otro proceso, se necesita la información sobre la totalidad de procesos del sistema. La información en la tabla de procesos es accesible cuando el proceso específico no es el que actualmente está en ejecución.

La tercera parte estática de contexto a nivel de sistema es una tabla de regiones por proceso, que se utiliza para el sistema de gestión de memoria. Por último, la pila del núcleo es una parte dinámica de contexto a nivel de sistema. Esta pila se usa cuando el proceso está ejecutando en modo núcleo y contiene la información que debe salvaguardarse y restaurarse en las llamadas a procedimientos o cuando ocurre una interrupción.

Tabla 3.12. Área U de UNIX.

Puntero a la tabla de proceso	Indica la entrada correspondiente a esta área U.
Identificador de usuario	Identificador de usuario real y efectivo. Utilizado para determinar los privilegios.
Temporizadores	Registro del tiempo que el proceso (y sus descendientes) han utilizado para ejecutar en modo usuario y modo núcleo.
Vector de manejadores de señales	Para cada tipo de señal definida en el sistema, se indica cómo el proceso va a reaccionar a la hora de recibirla (salir, ignorar, ejecutar una función específica definida por el usuario).
Terminal de control	Indica el terminal de acceso (*login*) para este proceso, si existe.
Campo de error	Recoge los errores encontrados durante una llamada al sistema.
Valor de retorno	Contiene los resultados de una llamada al sistema.
Parámetros de E/S	Indica la cantidad de datos transferidos, la dirección fuente (o destino) del vector de datos en el espacio de usuario, y los desplazamientos en fichero para la E/S.
Parámetros en fichero	Directorio actual y directorio raíz, dentro del sistema de ficheros, asociado al entorno de este proceso.
Tabla de descriptores de fichero de usuario	Recoge los ficheros del proceso abierto.
Campos límite	Restringe el tamaño del proceso y el tamaño máximo de fichero que puede crear.
Campos de los modos de permiso	Máscara de los modos de protección para la creación de ficheros por parte de este proceso.

CONTROL DE PROCESOS

La creación de procesos en UNIX se realiza por medio de la llamada al sistema *fork()*. Cuando con un proceso solicita una llamada *fork*, el sistema operativo realiza las siguientes funciones [BACH86]:

1. Solicita la entrada en la tabla de procesos para el nuevo proceso.
2. Asigna un identificador de proceso único al proceso hijo.
3. Hace una copia de la imagen del proceso padre, con excepción de las regiones de memoria compartidas.
4. Incrementa el contador de cualquier fichero en posesión del padre, para reflejar el proceso adicional que ahora también posee dichos ficheros.
5. Asigna al proceso hijo el estado Listo para Ejecutar.
6. Devuelve el identificador del proceso hijo al proceso padre, y un valor 0 al proceso hijo.

Todo este trabajo se realiza en modo núcleo, dentro del proceso padre. Cuando el núcleo ha completado estas funciones puede realizar cualquiera de las siguientes acciones, como parte de la rutina del activador:

1. Continuar con el proceso padre. El control vuelve a modo usuario en el punto en el que se realizó la llamada *fork* por parte del padre.
2. Transferir el control al proceso hijo. El proceso hijo comienza ejecutar en el mismo punto del código del padre, es decir en el punto de retorno de la llamada *fork*.
3. Transferir el control a otro proceso. Ambos procesos, padre e hijo, permanecen en el estado Listos para Ejecutar.

Puede resultar quizá un poco difícil visualizar este modo de creación de procesos debido a que ambos procesos, padre e hijo, están ejecutando el mismo segmento de código. La diferencia reside en que: cuando se retorna de la función *fork*, el parámetro de retorno se comprueba. Si el valor es 0, entonces este es el proceso hijo, y se puede realizar una bifurcación en la ejecución del programa para continuar con la ejecución de programa hijo. Si el valor no es 0, éste es el proceso padre, y puede continuar con la línea principal ejecución.

3.6. RESUMEN

El concepto fundamental dentro de los sistemas operativos modernos es el concepto de proceso. La función principal de un sistema operativo es crear, gestionar y finalizar los procesos. Cuando un proceso está activo, el sistema operativo debe ver cómo reservar tiempo para su ejecución por parte del procesador, coordinar sus actividades, gestionar las demandas que planteen conflictos, y reservar recursos del sistema para estos procesos.

Para realizar estas funciones de gestión de procesos, el sistema operativo mantiene una descripción de cada proceso o imagen de proceso, que incluye el espacio de direcciones dentro del cual el proceso está ejecutando, y el bloque de control de proceso. Este último contiene toda la información que el sistema operativo necesita para gestionar el proceso, incluyendo el estado actual, la reserva de recursos, la prioridad y otros datos de relevancia.

Durante su tiempo de vida, un proceso se mueve a lo largo de diferentes estados. Los más importantes de estos estados son Listo, Ejecutando, y Bloqueado. Un proceso Listo es un proceso que no está actualmente en ejecución pero que está listo para ser ejecutado tan pronto el sistema operativo lo active. Un proceso que está Ejecutando es aquel que está actualmente en ejecución por el procesador. En los sistemas multiprocesador, podrá haber varios procesos en este estado. Un proceso Bloqueado está esperando a que se complete un determinado evento, como una operación de E/S.

Un proceso en ejecución se interrumpe bien por una interrupción, que es un evento que ocurre fuera de proceso y que es recogido por el procesador, o por la ejecución de una llamada al sistema. En cualquier caso, el procesador realiza un cambio de modo, que es la transferencia de control a unas rutinas del sistema operativo. El sistema operativo, después de haber realizado el trabajo necesario, puede continuar con el proceso interrumpido o puede cambiar a otros procesos.

3.7. LECTURAS RECOMENDADAS

Todos los libros de texto listados de la Sección 2.9 cubren el material de este capítulo. Se pueden encontrar en [GOOD94] y [GRAY97] unas buenas descripciones de la gestión de procesos en UNIX. [NEHM75] es una descripción interesante de los estados de proceso del sistema operativo y las funciones necesarias para la activación de procesos.

GOOD94 Goodheart, B., y Cox, J. *The magic Garden Explained: The Internals of UNIX System V Release 4*. Englewood Cliffs, NJ: Prentice Hall, 1994.

GRAY97 Gray, J. *Interprocess Communication in Unix: The Nooks and Crannies.* Upper Saddler River, NJ: Prentice Hall, 1997.

NEHM75 Nehmer, J. «Dispatcher Primitives for the Construction of Operating System Kernels.» *Acta Informática*, vol 5, 1975.

3.8. TÉRMINOS CLAVE, CUESTIONES DE REPASO, Y PROBLEMAS

TÉRMINOS CLAVE

bloque de control de proceso	imagen de proceso	proceso hijo
cambio de modo	interrupción	proceso padre
cambio de proceso	modo núcleo	*round-robin*
estado bloqueado	modo privilegiado	*swapping*
estado ejecutando	modo sistema	tarea
estado listo	modo usuario	traza
estado saliente	nuevo estado	*trap*
estado suspendido	palabra de estado de programa	
expulsión	proceso	

CUESTIONES DE REPASO

3.1. ¿Qué es una traza de instrucciones?

3.2. ¿Cuáles son los eventos comunes que llevan a la creación de un proceso?

3.3. Para el modelo de procesamiento de la Figura 3.6, defina brevemente cada estado.

3.4. ¿Qué significa la expulsión de un proceso?

3.5. ¿Que es el *swapping* y cuál es su objetivo?

3.6. ¿Por qué la Figura 3.9 tiene dos estados bloqueados?

3.7. Indique cuatro características de un proceso suspendido.

3.8. ¿Para qué tipo de entidades el sistema operativo mantiene tablas de información por motivos de gestión?

3.9. Indique tres categorías generales de información que hay en el bloque de control de proceso.

3.10. ¿Por qué se necesitan dos modos (usuario y núcleo)?

3.11. ¿Cuáles son los pasos que realiza el sistema operativo para la creación de un proceso?

3.12. ¿Cuál es la diferencia entre interrupción y *trap*?

3.13. Dé tres ejemplos de interrupción.

3.14. ¿Cuál es la diferencia entre cambio de modo y cambio de proceso?

PROBLEMAS

3.1. Nombre cinco actividades principales del sistema operativo respecto a la gestión de procesos, y de forma breve describa por qué cada una es necesaria.

3.2. En [PINK89], se definen para los procesos los siguientes estados: ejecuta (ejecutando), activo (listo), bloqueado, y suspendido. Un proceso está bloqueado si está esperando el permiso para acceder a un recurso, y está bloqueado cuando está esperando a que se realice una operación sobre un recurso que ya ha adquirido. En muchos sistemas operativos, estos dos estados están agrupados en el estado de bloqueado, y el estado suspendido tiene el sentido usado en este capítulo. Compare las ventajas de ambas definiciones.

3.3. Para el modelo de siete estados de la Figura 3.9b, dibuje un diagrama de colas similar al de la Figura 3.8b.

3.4. Considerando el diagrama de transiciones de la Figura 3.9b. Suponga que le toca al sistema operativo activar un proceso y que hay procesos en el estado Listo y Listo/Suspendido, y que al menos uno de los procesos en estado Listo/Suspendido tiene mayor prioridad que todos los procesos Listos. Existen dos políticas extremas (1) siempre activar un proceso en estado Listo, para minimizar el efecto del swapping y (2) siempre dar preferencia a los procesos con mayor prioridad, incluso cuando eso implique hacer swapping, y dicho swapping no fuese necesario. Sugiera una política que encuentre un punto medio entre prioridad y rendimiento.

3.5. La Tabla 3.13 muestra los estados de proceso del sistema operativo VAX/VMS.

a) ¿Puede proporcionar una justificación para la existencia de tantos estados distintos de espera?

b) ¿Por qué los siguientes estados no tienen una versión residente y en *swap*: espera por fallo de página, espera por colisión de página, espera a un evento común, espera a liberación de página, y espera por recurso?

c) Dibuje un diagrama de transiciones de estado que indique la acción o suceso que causa dicha transición.

3.6. El sistema operativo VAX/VMS utiliza cuatro modos de acceso al procesador para facilitar la protección y compartición de recursos del sistema entre los procesos. El modo de acceso determina:

- **Privilegios de ejecución de instrucciones.** ¿Qué instrucciones puede ejecutar el procesador?
- **Privilegios de acceso a memoria.** ¿Qué posiciones de memoria pueden acceder las instrucciones actuales?

Los cuatros modos de acceso son:

- **Núcleo.** Ejecuta el núcleo del sistema operativo VMS, que incluye gestión de memoria, manejo de interrupciones, y operaciones de E/S.
- **Ejecutivo.** Ejecuta muchas de las llamadas al sistema, incluyendo las rutinas de gestión de ficheros y registros (disco o cinta).
- **Supervisor.** Ejecuta otros servicios del sistema operativo, tales como las respuestas a los mandatos del usuario.
- **Usuario.** Ejecuta los programas de usuario, además de las utilidades como compiladores, editores, enlazadores, y depuradores.

Tabla 3.13. VAX/VMS Estados del Proceso

Estado del Proceso	Condición del Proceso
Actualmente en ejecución	Proceso en ejecución.
Computable (residente)	Listo y residente en memoria principal.
Computable (en swap)	Listo, pero expulsado de memoria principal.
Espera por fallo de página	El proceso ha hecho referencia a una página que no está en memoria principal y debe esperar a que dicha página se lea.
Espera por colisión de página	El proceso ha hecho referencia a una página compartida que ha sido la causa de una espera por fallo de página en otro proceso, o una página privada que está en fase de ser leída en memoria o escrita a disco.
Espera a un evento común	Esperando a un *flag* de evento compartido (*flags* de evento son mecanismos de comunicación entre procesos por señales de un solo bit).
Espera por liberación de página	Esperando a que se libere una nueva página en memoria principal, que será asignada a un conjunto de páginas asignadas a dicho proceso (el conjunto de trabajo del proceso).
Espera hibernada (residente)	El proceso se ha puesto él mismo en estado de espera.
Espera hibernada (en swap)	Proceso hibernado que ha sido expulsado de memoria principal.
Espera a un evento local (residente)	El proceso está en memoria principal y a la espera de un *flag* de evento local (habitualmente a la espera de la finalización de una operación de E/S).
Espera a un evento local (en swap)	Proceso en espera de un evento local que ha sido expulsado de memoria principal.
Espera suspendida (residente)	El proceso se ha puesto en estado de espera por otro proceso.
Espera suspendida (en swap)	Proceso en espera suspendida que ha sido expulsado de memoria principal.
Espera de recurso	El proceso está esperando a disponer de un recurso de tipo general del sistema.

Un proceso que ejecuta en el modo con menos privilegios a menudo necesita llamar a un procedimiento que se ejecuta en un nivel de mayor privilegio; por ejemplo, un programa de usuario requiere una llamada al sistema. Esta llamada se realiza por medio de una instrucción de cambio de modo (CHM), que causa una interrupción que transfiere el control a una rutina en el nuevo modo de acceso. El retorno se realiza por medio de la instrucción REI (retorno de una excepción o interrupción).

a) Un gran número de sistemas operativos sólo tienen dos modos, núcleo y usuario. ¿Cuáles son las ventajas y los inconvenientes de tener cuatro modos en lugar de dos?

b) ¿Puede incluir un caso en el que haya más de cuatro modos?

3.7. El esquema de VMS comentado en el problema anterior se denomina habitualmente estructura de protección en anillo, tal y como se ilustra en la Figura 3.18. En realidad, el modelo sencillo núcleo/usuario, tal y como se describe en la Sección 3.3, es una estructura de dos anillos. [SILB04] aborda el problema con el siguiente enfoque.

La principal desventaja del modelo de estructura en anillo (jerárquico) es que no nos permite forzar el principio de necesidad-de-conocimiento. En particular, si un objeto debe ser

accesible en el dominio D_j pero no accesible en el dominio D_i, entonces debemos tener $j<i$. Pero esto implica que todo segmento accesible en D_i también es accesible en D_j.

a) Explique claramente cuál es el problema indicado en esta última cita.

b) Sugiera una forma mediante la cual un sistema operativo con estructura en anillo pueda resolver este problema.

3.8. La Figura 3.7b sugiere que un proceso sólo se puede encontrar en una cola de espera por un evento en el mismo instante.

a) ¿Es posible que un proceso esté esperando por más de un evento a la vez? Proporcione un ejemplo.

b) En dicho caso, ¿cómo modificaría la estructura de colas de la figura para dar soporte a esta nueva funcionalidad?

3.9. En gran número de los primeros ordenadores, una interrupción hacía que automáticamente los registros del procesador se guardasen en unas posiciones determinadas asociadas con esa señal de interrupción en particular, ¿bajo qué circunstancias esta es una buena técnica? Explique, por qué en general no es así.

3.10. En la Sección 3.4, se indica que UNIX no es apropiado para aplicaciones de tiempo real porque un proceso ejecutando en modo núcleo no puede ser expulsado. Elabore más el razonamiento.

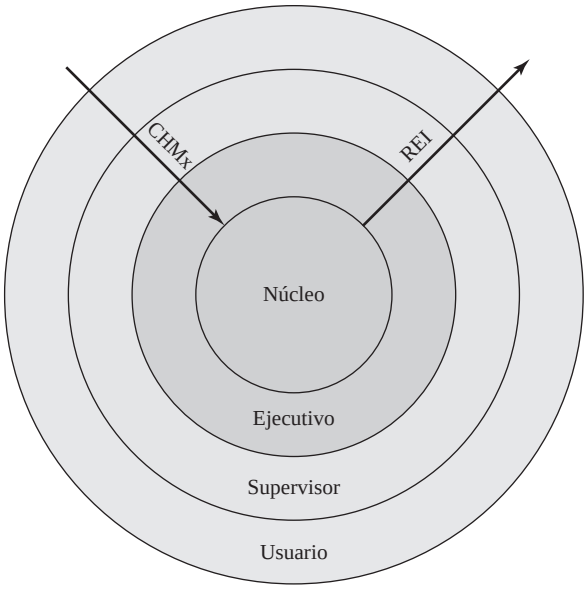


Figura 3.18. Modos de acceso de VAX/VMS.

PROYECTO DE PROGRAMACIÓN UNO. DESARROLLO DE UN INTÉRPRETE DE MANDATOS

El *shell* o intérprete de mandatos es la interfaz de usuario fundamental de los sistemas operativos. El primer proyecto es escribir un intérprete de mandatos sencillo – myshell – con las siguientes propiedades:

1. El intérprete de mandatos debe dar soporte a los siguientes mandatos internos:
   i. `cd <directorio>` – cambia el directorio actual a `<directorio>`. Si el argumento `<directorio>` no aparece, devuelve el directorio actual. Si el directorio no existe se debe proporcionar un mensaje de error apropiado. Este mandato debe modificar también la variable de entorno PWD.
   ii. `clr` – limpia la pantalla.
   iii. dir <directory> – lista el contenido de `<directorio>`.
   iv. `environ` – muestra todas las variables de entorno.
   v. `echo <comentario>` – muestra `<comentario>` en la pantalla seguido de una nueva línea (espacios múltiples o tabuladores se reducen a un especio sencillo).
   vi. `help` – muestra el manual de usuario usando el filtro more.
   vii. `pause` – detiene la ejecución del intérprete de mandatos hasta que se pulse 'Intro'.
   viii. `quit` – sale del intérprete de mandatos.
   ix. El entorno del intérprete de mandatos debe contener `shell=<ruta>/myshell` donde `<ruta>/myshell` es la ruta completa al ejecutable del intérprete de mandatos (no una ruta fijada al directorio inicial, sino la ruta real desde dónde se ha ejecutado).
2. Todo el resto de entradas por teclado se interpretan como la invocación de un programa, que deben realizarse por medio de un *fork* y la ejecución de dicho programa. Todo ello como un proceso hijo del intérprete de mandatos. Los programa deben ejecutarse en un entorno que incluya la entrada: `parent=<ruta>/myshell` donde `<ruta>/myshell` es la ruta descrita en el apartado 1.ix anterior.
3. El intérprete de mandatos debe ser capaz de leer su entrada de mandatos de un fichero. Por ejemplo, si se invoca al intérprete de mandatos con la línea:

   ```
   myshell fichero-lotes
   ```

   Donde `fichero-lotes` se supone que contiene las líneas de mandatos para el intérprete. Cuando se llegue al final del fichero, el intérprete de mandatos debe terminar. Obviamente, si el intérprete se invoca sin argumentos de entrada, solicitará los mandatos al usuario vía consola.
4. El intérprete de mandatos debe soportar redirección de E/S, sobre *stdin* y/o *stdout*. Por ejemplo, la línea de mandatos:

   ```
   nombreprograma arg1 arg2 < entrada > salida
   ```

   ejecutará el programa `nombreprograma` con los argumentos `arg1` y `arg2`, el flujo de entra *stdin* se alimentará del fichero `entrada` y el flujo de salida *stdout* se volcará en el fichero `salida`.

   La redirección de *stdout* debe de ser posible también para los mandatos internos: `dir`, `environ`, `echo`, y `help`.

   Para la redirección de salida, si el carácter de redirección es > se creará el fichero `salida` si no existe y si existe se truncará su contenido. Si el indicador de redirección es >> se creará el fichero `salida` si no existe y si existe se añadirá la salida al final de su contenido.
5. El intérprete de mandatos debe soportar la ejecución de mandatos en segundo plano (*background*). Un signo `&` al final de la línea de mandatos indica que el intérprete debe devolver un *prompt* al usuario, inmediatamente después de haber lanzado el programa.
6. El *prompt* debe indicar la ruta del directorio actual.

Nota: se puede asumir que todos los argumentos en la línea de mandatos (incluyendo los símbolos de redirección `>`, `<`, y `>>`; y background `&`) estarán separados de los otros argumentos en la línea de mandatos por espacios en blanco – uno o más espacios y/o tabuladores (obsérvese la línea de mandatos en 4).

Requisitos del proyecto

1. Diseñe un intérprete de mandatos sencillo que satisfaga los criterios antes mencionados e impleméntelo en la plataforma UNIX seleccionada.
2. Escriba un manual sencillo que describa cómo usar el intérprete. El manual debe contener suficiente información para que un usuario principiante en UNIX pueda usarlo. Por ejemplo, se debe explicar los conceptos de redirección de E/S, de entorno de programa, y de ejecución en segundo plano (*background*). El manual **DEBE** llamarse `readme` y debe ser un documento de texto plano que pueda leerse con un editor de texto estándar.

   Como ejemplo del tipo de profundidad que se pide, deberá inspeccionar los manuales en línea de `csh` y `tcsh`. (`man csh`, `man tcsh`). Estos intérpretes tienen mucha más funcionalidad que el que se pide, de forma que el manual que se requiere no deberá ser tan largo. No debe incluir bajo ningún concepto consideraciones de implementación, ficheros fuente o código. Esto se incluirá en otros ficheros del proyecto. Este manual debe ser un Manual de Usuario no un Manual de Desarrollo.
3. El código fuente **DEBE** estar extensamente comentado y apropiadamente estructurado, permitiendo a sus colegas comprenderlo y darle mantenimiento al código. ¡El código comentado con propiedad y bien alineado es mucho más fácil de interpretar e interesa que la persona que pueda evaluar su código pueda entenderlo con facilidad sin necesidad de hacer gimnasia mental!
4. Los detalles sobre el envío del proyecto se proporcionarán con antelación a la fecha límite.
5. El envío del proyecto debe contener sólo ficheros fuente, incluyendo ficheros de cabecera, `makefile` (en letras minúsculas, por favor) el fichero `readme` (en letras minúsculas, por favor). No se debe incluir ningún fichero ejecutable. El evaluador recompilará automáticamente su intérprete de mandatos a partir del código fuente. Si el código fuente no compila, no será calificado.
6. El `makefile` (en letras minúsculas, por favor) **DEBE** generar un fichero binario llamado `myshell` (en letras minúsculas, por favor). Un ejemplo de `makefile` sería:

```
# Pepe Potamo, s1234567 – Proyecto 1 de SO
# CompLab1/01 tutor: Chema Peña
myshell: myshell.c utility.c myshell.h
    gcc –Wall myshell.c utility.c –o myshell
```

   El programa `myshell` se generará simplemente tecleando `make` en la línea de mandatos.

Nota: la cuarta línea del `makefile` de ejemplo **DEBE** comenzar por un tabulador.

7. En el ejemplo mostrado arriba los ficheros incluidos en el directorio de envío eran:

```
makefile
myshell.c
utility.c
myshell.h
readme
```

Envío

Es necesario un `makefile`. Todos los ficheros incluidos en su envío se copiarán al mismo directorio, por tanto no incluya rutas en su `makefile`. El `makefile` debe incluir todas las dependencias para compilar el programa. Si se incluye una biblioteca, su `makefile` debe construir dicha biblioteca.

Para dejar esto claro: ***no construya a mano ningún binario o fichero objeto***. Todo lo que se requerirá serán sus ficheros fuente, un `makefile`, y el fichero `readme`. Verifique su proyecto, copiando dichos ficheros a un directorio vacío y compilándolo completamente por medio del mandato `make`.

Usaremos un *script* que copia sus ficheros a un directorio de prueba, borre el fichero `myshell` previo, así como todos los ficheros `*.a`, y/o los `*.o`, y ejecute un `make`, copie una serie de ficheros de pruebas al directorio, y compruebe su intérprete de mandatos por medio de una serie de ficheros por lotes pasados por la entrada estándar (*stdin*) y por línea de mandatos. Si esta batería de pruebas falla debido a nombres erróneos, diferencias entre mayúsculas y minúsculas, versiones erróneas de código que fallen al compilar o falta de ficheros, etc., la secuencia de evaluación se detendrá. En este caso, la calificación obtenida será la de las pruebas pasadas completamente, la puntuación sobre el código fuente y el manual.

Documentación solicitada

En primer lugar, su código fuente se comprobará y evaluará así como el manual `readme`. Los comentarios son completamente necesarios en su código fuente. El manual de usuario se podrá presentar en el formato que se desee (siempre que se pueda visualizar por medio de un editor de texto estándar). Además, el manual debe contener suficiente información para que un usuario principiante de UNIX pueda usar el intérprete. Por ejemplo, se debe explicar los conceptos de redirección de E/S, de entorno de programa, y de ejecución en segundo plano (*background*). El manual **DEBE** llamarse `readme` (todo en minúsculas, por favor, **SIN** extensión `.txt`)