UNIVERSIDAD ESTATAL A DISTANCIA
VICERRECTORA ACADÉMICA
ESCUELA CIENCIAS EXACTAS Y NATURALES
CARRERA INFORMÁTICA ADMINISTRATIVA
CÁTEDRA TECNOLOGÍA DE SISTEMAS
Email: 881@uned.ac.cr

GUÍA DE ESTUDIO PARA EL CURSO DE SISTEMAS OPERATIVOS (Código 881)

Elaborado por

Ing. Frank Mendoza Hernández

San José, Costa Rica

2006

Producción académica y
asesoría pedagógica:
Ana Láscaris-Comneno Slepuhin

Encargada de Cátedra:
Karol Castro Chaves

Especialista en contenidos:
Karol Castro Chaves

CONTENIDO

INTRODUCCIÓN

Las primeras computadoras eran máquinas inmensas operadas desde una consola. Solo se conocía el *hardware.* El programador escribía sus programas y, luego, desde la consola, la computadora era controlada.

Posteriormente, con el transcurrir de los años, se crearon *software* y *hardware* adicionales. Por ejemplo, se diseñaron ensambladores, cargadores y ligadotes, que vinieron a ayudar en las tareas de programación. También se diseñaron bibliotecas de funciones comunes. Como su nombre lo indica, estas bibliotecas servían para copiar una función en un nuevo programa; los programadores no tenían que crear esas funciones.

Además, cada dispositivo de entrada / salida traía consigo sus propias características. Por lo tanto, era necesario tener cuidado con su programación, que para cada dispositivo era diferente. Además, para cada dispositivo debía desarrollarse una subrutina especial, llamada **manejador de dispositivo**. Por ejemplo, una tarea sencilla como leer un carácter de un lector de cinta de papel, podría requerir complicadas secuencias de operaciones específicas para dicho dispositivo.

Un tiempo después, se crearon los compiladores y otros leguajes, que brindaron ayuda en las tareas de programación, pero con ello el funcionamiento del computador se volvió más complejo.

Un **sistema operativo** (SO) es un programa que actúa como intermediario entre el usuario y el *hardware* de un computador. En sí, es un programa de computadora, pero muy especial, quizá el más complejo e importante en una computadora.

El sistema operativo tiene como objetivo principal lograr que el sistema de computación se use de manera cómoda, y que el *hardware* del computador se emplee eficientemente.

El sistema operativo despierta a la computadora y hace que reconozca a la unidad de procesamiento central (CDU), la memoria, el teclado, el sistema de video y las unidades de disco. Además, facilita la comunicación de los usuarios con la computadora y sirve de plataforma para que se corran los programas de aplicación.

El sistema operativo posee cuatro tareas principales, que son proporcionar una interfaz de línea de comandos o una interfaz gráfica al usuario, administrar los dispositivos de *hardware* de la computadora, administrar y mantener los sistemas de archivos de disco y apoyar a otros programas.

Esta guía de estudio está orientada a cubrir los principales aspectos de cada tema, con miras a contribuir efectivamente en su aprendizaje. Por ello, se constituye en un material complementario que, en ningún caso, sustituye al libro de texto. Por lo anterior, es responsabilidad del estudiante estudiar a fondo los diferentes tópicos, y aclarar dudas o inquietudes en las tutorías presenciales que brinda este curso.

Comprende ocho temas de gran importancia para la formación académica del estudiante. El orden en que se presentan estos temas está de acuerdo con su nivel de dificultad. Por lo tanto, el estudiante debe seguir al detalle esta guía para lograr un buen entendimiento de la materia.

Los temas de estudio se describen a continuación:

- Introducción
- Procesos y subprocesos
- Bloqueos irreversibles
- Administración de memoria
- Entrada / salida
- Sistemas de archivos
- Seguridad
- Diseño de Sistemas Operativos

DESCRIPCIÓN DEL CURSO

Objetivo general

Brindar al estudiante los fundamentos básicos de diseño, programación, aplicación e implementación de los sistemas operativos.

Objetivos específicos

- Introducir al estudiante en los conceptos, problemática y técnicas de solución a los problemas de diseño en los sistemas operativos y procesos.
- Enfrentar al estudiante con la resolución de problemas de programación que requieran de la aplicación práctica de los conocimientos adquiridos sobre sistemas operativos.
- Propiciar la capacidad de análisis del estudiante, mediante el planteamiento de problemas específicos de los sistemas operativos, en los que aplique sus conocimientos al respecto.
- Describir el sistema de archivos en los sistemas operativos.
- Desarrollar los sistemas de administración de memoria en los sistemas operativos.
- Analizar los procesos de: entradas, salidas, bloqueos y sistemas operativos distribuidos.
- Desarrollar el problema del bloqueos en el sistema operativo.
- Relacionar el sistema operativo con el *hardware*.

Requisitos del curso

Este curso está diseñado para una carga académica asignada de tres créditos, Es parte del plan de Bachillerato de la carrera Informática Administrativa (código 30). En él se asume que usted ha aprobado, como mínimo, los cursos de: *Introducción a la Programación* (831), *Organización de Computadores* (823), *Matemática Discretas* (841) o, en su defecto, que posee conocimientos básicos de dichas áreas. El no tener los conocimientos previos que le entregan los cursos antes mencionados, le dificultará enormemente el éxito en esta asignatura. Por lo tanto, piénselo antes de seguir adelante.

Material de apoyo

Bibliografía oficial del curso:

- Libro de texto: Tanembaum, Andrew S. Sistemas operativos Modernos. 2ª Edición, Prentice-Hall Hispanoamericana, S.A., México. 2003.
- Castro, Karol (2006). La orientación del curso de Sistemas operativos. EUNED.
- Esta guía de estudio que usted está leyendo.

Bibliografía de apoyo:

- Milencovic, Milan. Sistemas operativos: Conceptos y Diseño. Segunda Edición, McGraw Hill, México, 1994.
- Silberschatz, Abraham y otros. Sistemas operativos. Sexta Edición, Editorial Limusa Wiley, México, 2002.
- Carretero Pérez, Jesús y otros. Sistemas operativos: una visión aplicada. Primera Edición, Editorial McGraw Hill, España, 2001

Desglose de temas

El presente curso de **Sistemas operativos** consta de nueve temas principales:

- Introducción
- Procesos y subprocesos
- Bloqueos irreversibles
- Administración de memoria
- Entrada/salida
- Sistema de archivos
- Sistemas con múltiples procesadores
- Seguridad
- Diseño de sistemas operativos

Para un adecuado aprovechamiento del curso, se escogió utilizar como unidad didáctica un libro de texto que resulta autodidáctico y que motiva al estudiante a continuar con el aprendizaje de los temas señalados.

El libro de texto consta de 9 capítulos. Cada uno de ellos trata sobre aspectos importantes relacionados con los sistemas operativos.

En la siguiente tabla se detallan los temas principales, los subtemas correspondientes, el número del capítulo del libro de texto y el número de página del libro donde podrán localizar cada un de ellos:

Tema	Capítulo del libro	Número de página del libro
Tema 1		
• Introducción	1	1-70
• Procesos y subprocesos	2	71-158
Tema 2		
• Bloqueos irreversibles	3	159-188
• Administración de memoria	4	189-268
• Entrada/salida	5	269-378
Tema 3		
• Sistema de archivos	6	379-452
• Sistemas con múltiples procesadores	8	503-582
Tema 4		
• Seguridad	9	583-670
• Diseño de sistemas operativos	12	855-900

GUÍAS DE LECTURA

En cada tema de esta guía de estudio usted encontrará una sección llamada *Guía de Lectura.* Esta tiene como finalidad indicarle las páginas respectivas que usted debe leer y estudiar de su libro de texto, para cada tema y subtema.

COMENTARIOS GENERALES

Los comentarios generales presentados para cada tema en esta guía de estudio brindan aspectos importantes de dicho tema, y su ubicación dentro de cada capítulo del libro de texto. Le servirán para sintetizar los conceptos transmitidos. De esta manera, usted podrá determinar si requiere repasar o aclarar alguno de los conceptos antes de desarrollar los ejercicios.

EJERCICIOS SUGERIDOS

Con el propósito de que usted realice una autoevaluación de su comprensión y aprendizaje del tema en estudio, esta guía incluye una sección llamada *Ejercicios sugeridos*, que selecciona algunos de todos los ejercicios incluidos al final de cada capítulo del libro de texto. Además, para algunos de los temas, se incluye un ejemplo adicional al presentado en el libro de texto, así como un ejercicio sugerido para desarrollar en la tutoría correspondiente al tema o temas de estudio.

RESOLUCIÓN DE EJERCICIOS SUGERIDOS

En esta área usted encontrará las soluciones a los ejercicios sugeridos.

CAPÍTULO 1

INTRODUCCIÓN

Sumario

- ¿Qué es un sistema operativo?
- Historia de los sistemas operativos
- La variedad de sistemas operativos
- Repaso de *hardware* de cómputo
- Concepto de los sistemas operativos
- Llamadas al sistema
- Estructura del sistema operativo
- Investigación sobre sistemas operativos
- Bosquejo del resto del libro
- Unidades métricas
- Resumen

Propósito del capítulo

Con el estudio de este tema se pretende brindar una noción general acerca de los sistemas operativos, su importancia, características, funciones y parte de su historia. Se conocerá en forma general la variedad de sistemas operativos, según la instalación que se posea.

Se realizará repaso de los conceptos de *hardware*, que incluye conceptos de memoria, dispositivos de entrada y salida, así como las diferentes configuraciones para el trasiego de datos.

También se incluirá una introducción general a los conceptos de procesos, memoria y archivos, que son básicos para entender el funcionamiento global del sistema operativo, y en general todo lo relativo al concepto de **llamadas al sistema**.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar los componentes básicos de un sistema de cómputo.
- Explicar la evolución que han tenido los sistemas operativos a través del desarrollo informático.
- Identificar los diferentes sistemas operativos que existen.
- Explicar los conceptos de *hardware*.
- Explicar los conceptos básicos de los sistemas operativos y las llamadas al sistema.
- Explicar la estructura de un sistema operativo.

Guía de lecturas

Para lograr los objetivos descritos anteriormente**,** es importante que usted realice las siguientes lecturas**:**

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

Un sistema de cómputo moderno consta de uno o más procesadores, una memoria principal, discos, impresoras, un teclado, una pantalla, interfaces de red y otros dispositivos de entrada/salida. Se trata de un sistema complejo. Escribir programas para administrar todos estos recursos para que operen de manera óptima es un trabajo difícil. Por esta razón, las computadoras vienen equipadas con una capa de *software* llamada **sistema operativo** (en lo sucesivo SO).

En resumen, la labor del SO es administrar todos los recursos de *hardware* y *software*, y proporcionar a los programas de los usuarios una interfaz sencilla para comunicarse con el *hardware*.

En la figura 1-1, página 2 del libro de texto, se puede ver de manera gráfica la posición que ocupa el SO en la secuencia de componentes de un sistema de cómputo.

El *hardware* está compuesto por dispositivos físicos, microarquitecura y el lenguaje de máquina.

Los dispositivos físicos son todos aquellos como: circuitos integrados, cables, fuentes de potencia, tubos de rayos catódicos y objetos similares.

La microarquitectura es el nivel en que los dispositivos físicos se agrupan para conformar unidades funcionales.

El lenguaje de máquina o arquitectura de conjunto de instrucciones (ISA) son una serie de instrucciones, en lenguaje ensamblador, que permiten controlar en forma directa todos los dispositivos antes mencionados.

Los **programas del sistema** están compuestos por el sistema operativo y otras aplicaciones que ayudan a los usuarios directamente.

El SO proporciona al programador una serie de instrucciones mediante las que puede trabajar de una manera cómoda con el computador, sin necesidad de llegar a un nivel bajo de programación para hacer uso de los recursos disponibles. El SO es (por lo regular) la porción del *software* que opera en **modo *kernel*** o **modo supervisor** y está protegida de la intervención del usuario.

Las otras aplicaciones mencionadas en la figura 1-1 corresponden a programas para hacer más eficiente los programas (tales como los compiladores que convierten el lenguaje fuente en lenguaje de máquina). El intérprete de comandos (SHELL) brinda a los usuarios una serie de comandos para interactuar de una manera ágil con el SO.

Los **programas de aplicación** son todos aquellos programas desarrollados por los programadores para las diferentes empresas e instituciones. Se desarrollan en lenguajes comerciales (COBOL, *Visual Basic*, C++, Pascal y muchos otros más en el mercado).

(Para ahondar más en el tema, refiérase a las páginas 1 a la 3 del libro de texto.)

¿Qué es un sistema operativo? Para comprender bien este concepto, diremos que el SO desempeña básicamente dos funciones independientes: extender la máquina y administración de recursos. Veamos cada uno de ellos:

El sistema operativo como máquina extendida:

En el lenguaje de máquina, la arquitectura de la mayoría de las computadoras es primitiva y engorrosa de programar, sobre todo en cuanto a entrada/salida. La función del sistema operativo es presentar al usuario el equivalente a una máquina extendida o máquina virtual, que es más fácil de programar que el *hardware* subyacente. El sistema operativo presta una serie de servicios que los programas pueden obtener empleando instrucciones especiales que se conocen como llamadas **al sistema.**

El sistema operativo como administrador de recursos:

El concepto de SO como algo que primordialmente presenta a sus usuarios una interfaz cómoda es una perspectiva descendente. Desde una perspectiva alterna, la ascendente, el sistema operativo tiene como misión administrar todos los elementos de un sistema complejo. Los computadores modernos constan de procesadores, memoria, temporizadores, discos, ratones, interfaces de red, impresoras y una amplia gama de otros dispositivos. En la perspectiva alterna, la tarea del SO consiste en efectuar un reparto ordenado y controlado de los procesadores, memorias y dispositivos de E/S, entre los diversos programas que compiten por obtenerlos.

(Para una mayor comprensión del tema, puede referirse a la página 5 y 6 del libro de texto.)

Dentro de la historia de los sistemas operativos, encontramos las siguientes generaciones:

- **La 1º generación (1945-1955): Bulbos y conexiones**

  Estas máquinas eran enormes y llenaban cuartos completos con decenas de miles de bulbos. Eran sumamente lentas. Toda la programación se llevaba a cabo en lenguaje de máquina obsoleto, y con frecuencia se utilizaban conexiones para controlar las funciones básicas de la máquina. Se desconocían los lenguajes de programación y no se oía de los SO. La inmensa mayoría de los problemas eran cálculos numéricos.

- **La 2º generación (1955-1965): Transistores y sistemas de procesamiento por lotes**

  Con la introducción del transistor a mediados de los 50 se modificó el panorama de la computación radicalmente.

  - Se habla de lenguajes de programación (por ejemplo: *Fortran*, Ensamblador).
  - Uso de tarjetas.
  - Separación de las funciones de diseño, operación, programación, mantenimiento, etc.
  - Uso de cintas para el procesamiento en lote.
  - Uso de *software*, precursor de los sistemas operativos.
  - Fueron utilizados principalmente para cálculos científicos y de ingeniería.

- **La 3º generación (1965-1980): Circuitos integrados y multiprogramación**

  Poseían ventaja en precio y desempeño que sus antecesoras.

  - Son utilizadas para cálculos científicos y comerciales
  - Usan un SO enorme e introducen la multiprogramación.
  - Uso de la técnica de *spooling* (operación simultánea y en línea de periféricos).
  - Inicia el uso del "tiempo compartido".
  - Se experimenta un crecimiento de las minicomputadoras.

- **La 4º generación (1980-1990): Computadoras personales (PC)**

  - Uso de los circuitos LSI (*Large Scale Integrate*) y chip's poderosos.
  - Genera una gran industria de producción de *software* para las PC.

- SO más populares MS-DOS y UNÍS.
- Se inicia el desarrollo de las redes de computadoras con SO de red y SO distribuidos.

Existe una gran variedad de sistemas operativos. A continuación se detallan los más importantes:

(Para ahondar más en el tema, refiérase a las páginas de la 6 a la 18 del libro de texto.)

Sistemas operativos de *Mainframe*: Están claramente orientados al procesamiento de varios trabajos a la vez. Casi todos necesitan cantidades enormes de E/S. Los servicios que ofrecen pueden ser de tres tipos: por lotes, procesamiento de transacciones y tiempo compartido.

- Un sistema por lotes procesa trabajos rutinarios sin que exista un usuario interactivo presente.
- Los sistemas procesadores de transacciones manejan numerosas solicitudes pequeñas, por ejemplo, como procesamiento de cheques o reservaciones de pasajes aéreos. Cada unidad de trabajo es pequeña, pero el sistema debe manejar cientos o miles de ellos por minuto.
- Los sistemas de tiempo compartido permiten a múltiples usuarios remotos ejecutar trabajos remotos de manera simultánea.

Sistemas operativos de servidor: Se encuentran un nivel más abajo. Estos se ejecutan en servidores, que son computadores personales muy grandes, estaciones de trabajo o incluso *mainframes* y, dan servicio a múltiples usuarios a través de una red, permitiéndoles compartir recursos de *hardware* y *software*.

Sistemas operativos multiprocesador: Son una forma cada más común de obtener potencia de computación; es conectar varios CPU en un sólo sistema. Dependiendo de la forma exacta de la conexión y de qué se comparte, a estos sistemas se les llaman computadoras paralelas, multicomputadoras o multiprocesadores. Necesitan sistemas operativos especiales, pero con frecuencia son variaciones de los sistemas operativos de servidor con funciones especiales de comunicación y conectividad.

Sistemas operativos de computador personal: Su misión consiste en presentar una buena interfaz a un solo usuario.

Sistemas operativos de tiempo real: Se caracterizan porque su parámetro clave es el tiempo. Se utilizan para controlar procesos de diferentes líneas de fabricación. Cuando se hace necesario que se ejecuten tareas en un tiempo determinado y por un intervalo deseado, se requiere de sistemas de tiempo real riguroso. Otro tipo de sistema de tiempo real es el no riguroso, en el que no se

requiere ser tan estricto en los tiempos, ni cumplir siempre con plazos. Entre estos últimos se encuentran los sistemas de audio digital o multimedia.

Sistemas operativos integrados: Se utilizan en computadoras de bolsillo (*palm-top*) y sistemas integrados. Los sistemas integrados operan en las computadoras que controlan dispositivos que, por lo general, no se consideran computadoras, tales como televisores y hornos de microondas. Tienen algunas características de los sistemas de tiempo real, pero también tienen limitaciones de tamaño, memoria y consumo de electricidad que los hacen especiales.

Los **sistemas operativos de tarjeta inteligente:** Son sistemas operativos más pequeños que se ejecutan en tarjetas inteligentes. Son dispositivos del tamaño de una tarjeta de crédito que contienen un chip de CPU.

Para una mayor comprensión del tema puede referirse a las páginas 18 a la 20 del libro de texto.

Un SO está íntimamente relacionado con el *hardware* de la computadora en la que opera, ya que extiende el conjunto de instrucciones de la computadora y administra sus recursos.

El cerebro de la computadora es la **CPU**. Esta toma instrucciones de la memoria y las ejecuta. El ciclo básico de toda CPU consiste en tomar la primera instrucción de la memoria, decodificarla para determinar su tipo y operandos, ejecutarla y luego tomar, decodificar y ejecutar instrucciones subsiguientes. Es así como se ejecutan los programas.

Casi todas las computadoras tienen varios registros especiales que puede ver el programador. Uno de ellos es el **contador de programa**, que contiene la dirección de memoria en la que está la siguiente instrucción que se tomará. Una vez obtenida esa instrucción, el contador de programa se actualizará de modo que apunte a su sucesora.

Otro registro es el **apuntador de pila**, que apunta a la parte superior de la pila actual en memoria. La pila contiene un marco procedimiento en el que se ha entrado, pero no se ha salido aún. El marco de pila de un procedimiento contiene parámetros de entrada, variables locales y variables temporales que no se guardan en registros.

Otro registro es la **palabra de estado de programa**. Este registro contiene los bits de código de condición, que se ajustan cuando se ejecutan instrucciones de comparación, junto con la prioridad de la CPU, el modo (de usuario o de Kernel) y otros bits de control.

El segundo componente importante de cualquier computadora es la **memoria**. El sistema de memoria se construye en una jerarquía de capas (tal como se muestra en la figura 1-7, página 24 del libro de texto).

La capa superior consiste en registros internos de la CPU. Estos se componen del mismo material que la CPU y por tanto, son tan rápidos como ella. El acceso a ellos no implica retrazo.

Luego viene la **memoria caché,** que en su mayor parte está bajo el control del *hardware*. La memoria principal se divide en líneas de caché. Las líneas de caché de uso más intenso se mantienen en un caché de alta velocidad situado dentro de la CPU o muy cerca de ella.

Luego viene la **memoria principal**. También se le conoce como **RAM** (memoria de acceso aleatoria). Todas las solicitudes de la CPU que no se pueden atender desde el caché se dirigen a la memoria principal.

En el siguiente escalón de la jerarquía se localiza el **disco magnético** (**disco duro**). Su acceso es más lento. Esto se debe al hecho de que un disco es un dispositivo mecánico, tal como se muestra en la figura 1-8 (página 25 del libro de texto).

La última capa de la jerarquía de memoria es la **cinta magnética**. Este medio suele utilizarse como respaldo de los discos y para guardar conjuntos de datos muy grandes.

Muchas computadoras tienen una pequeña cantidad de **memoria de acceso aleatorio no volátil** (**ROM**, memoria de solo lectura). Esta se programa en la fábrica y no puede modificase después. Es rápida y económica. En algunas computadoras, el cargador de auto arranque que sirve para iniciar la computadora está almacenado en la ROM.

Volviendo a la memoria principal, a menudo es conveniente tener varios programas en la memoria al mismo tiempo. Si un programa se bloquea mientras espera que termine una lectura de disco, este programa podrá utilizar la CPU para aprovecharla mejor.

Cuando existen dos o más programas a la vez en la memoria, deben resolverse dos problemas:

- ¿Cómo proteger los programas entre sí, cómo proteger al *kernel* de todos ellos?

- ¿Cómo manejar la relocalización?

El primer problema es obvio, pero el segundo es un poco más sutil. (La solución más sencilla se muestra en la figura 1-9ª, página 27 del libro de texto.) En esta figura se visualiza una computadora equipada con dos registros especiales, el registro base y el registro límite.

La memoria no es el único recurso que debe administrar el SO; los dispositivos de E/S, también interactúan intensamente con él. Los dispositivos de E/S, por lo regular, constan de dos partes: una controladora y el dispositivo en sí.

La controladora es un chip o conjunto de chips montados en una tarjeta insertable que controla físicamente el dispositivo. Dicha controladora acepta comandos del SO (leer datos del dispositivo) y los ejecuta.

El otro componente es el dispositivo en sí. Los dispositivos tienen interfaces relativamente simples, debido a que no tienen mucha capacidad para estandarizarlos.

Puesto que cada tipo de controladora es distinto, se requieren diferentes *softwares* para manejar cada una. El *software* que se comunica con una controladora, dándole comandos y aceptando respuestas, se denomina **controlador de dispositivo**.

La operaciones de entrada y salida pueden ejecutarse de tres maneras distintas. En el método más sencillo, un programa de usuario emite una llamada al sistema, que el *kernel* traduce en una llamada de procedimiento al controlador de dispositivo apropiado. Entonces, el controlador de dispositivo inicia E/S y entra en un círculo corto que pregunta continuamente al dispositivo para ver si ya terminó.

Una vez terminada la E/S, el controlador de dispositivo coloca los datos, si hay, donde se necesita y regresa. El SO devuelve entonces el control al invocador. Este método se denomina **espera activa** y tiene la desventaja de mantener ocupada a la CPU revisando el dispositivo hasta que éste termine.

El segundo método consiste en que el controlador de dispositivo pone en marcha el dispositivo y le pide generar una interrupción cuando haya terminado. En ese momento, el controlador de dispositivo regresará. El SO bloquea entonces al invocador si es necesario y busca otras cosas que hacer. Cuando la controladora detecta que la transferencia terminó, genera una interrupción para avisar.

El tercer método para efectuar E/S utiliza un chip especial de acceso directo a memoria **DMA** (*direct memory access*) que puede controlar el flujo de bits entre la memoria y alguna controladora, sin que la CPU tenga que intervenir en forma continua. La CPU prepara el chip DMA, indicándole cuántos *bytes* hay que transferir al dispositivo, las direcciones de memoria en cuestión y el sentido, y se desentiende de él. (Para una mayor comprensión del tema puede referirse a las páginas 28 a la 31 del libro de texto.)

A medida que aumentó la rapidez de los procesadores y las memorias, la capacidad de un solo **bus** para manejar todo el tráfico, se sometió a una demanda excesiva. Algo tenía que ceder. Por ello, se añadieron más buses, tanto para los dispositivos de E/S más rápidos como para el tráfico del CPU y la memoria.

A continuación, se describen varios **conceptos de los sistemas operativos:**

Un concepto clave en todos los SO es el **proceso**. Un proceso es básicamente un programa en ejecución. Cada proceso tiene asociado un **espacio de direcciones:** una lista de posiciones de memoria desde algún mínimo (normalmente cero) hasta algún máximo, que el proceso puede leer y puede escribir. El espacio de direcciones contiene el programa ejecutable, sus datos y su pila. Cada proceso está asociado, también, con algún conjunto de registros incluido el controlador de programa, el apuntador de pila y otros registros de *hardware*, así como toda la demás información necesaria para ejecutar el programa.

Cuando dos o más procesos están interactuando, a veces pueden meterse en una situación de estacionamiento de la que no pueden salir. Tal situación se denomina **bloque irreversible.**

Toda computadora tiene una memoria principal que se usa para contener los programas en ejecución. En un SO muy simple, sólo existe un programa a la vez en la memoria. Para ejecutar un segundo programa, es preciso desalojar el primero y colocar el segundo en la memoria.

Los sistemas operativos más avanzados permiten que existan varios programas en la memoria al mismo tiempo. Para evitar que se interfieran (e interfieran con el SO), se requiere algún tipo de mecanismo de protección. Aunque este mecanismo debe estar en el *hardware*, está bajo el control del SO.

Todas las computadoras tienen dispositivos físicos para obtener entradas y producir salidas.

Todo SO cuenta con un subsistema de E/S para administrar sus dispositivos de E/S. Una parte del *software* de E/S es independiente del dispositivo; es decir, es igual de válido para muchos dispositivos de E/S, otras partes, como los controladores de dispositivos, son específicos para determinados dispositivos de E/S.

Es obvio que se requieren llamadas al sistema para crear **archivos**, eliminarlos, leerlos y escribirlos. A fin de contar con un lugar dónde guardar los archivos, casi todos los sistemas operativos tienen el concepto de **directorio** como mecanismo para agruparlos.

Todo archivo dentro de la jerarquía de directorios puede especificarse mediante su **nombre de ruta** desde el tope de la jerarquía, que es el **directorio raíz.**

Las computadoras tienen grandes cantidades de información que los usuarios a menudo consideran confidencial. Corresponde al SO controlar la seguridad del

sistema para que únicamente los usuarios autorizados puedan tener acceso a los archivos.

El SO es código que ejecuta las llamadas al sistema. Los editores, compiladores, ensambladores, enlazadores e intérpretes de comandos (***shell***) en definitiva no forman parte del SO, aunque sean importantes y útiles.

Aunque no es parte del SO, el ***shell*** hace uso intensivo de muchas de sus funciones y por ello, es un buen ejemplo de cómo pueden usarse las llamadas al sistema. También constituye la interfaz primaria entre un usuario sentado frente a su terminal y el SO, a menos que el usuario esté utilizando una interfaz gráfica (GUI).

(Para una mayor comprensión del tema puede referirse a la sección 1.5, páginas 34 a la 43 del libro de texto.)

La interfaz entre el SO y los programas de usuario está definida por el conjunto de **llamadas al sistema** ofrecidas por el SO. Si queremos entender en realidad lo que hacen los sistemas operativos, debemos examinar de cerca esa interfaz. Las llamadas al sistema con que cuenta la interfaz varían de un SO a otro.

Existen diferentes tipos de llamadas al sistema:

1. Llamadas al sistema para administración de procesos
2. Llamadas al sistema para administración de archivos
3. Llamada al sistema para administración de directorios
4. Diversas llamadas al sistema.

(Para ampliar los conceptos anteriormente descritos, puede referirse a las secciones 1.6.1, 1.6.2, 1.6.3 y 1.6, páginas de la 44 a la 53 del libro de texto.)

Un programa UNIX consiste en código que hace esto o lo otro, emitiendo llamadas al sistema para solicitar ciertos servicios. En contraste, un programa Windows es controlado por eventos. El programa principal espera que ocurra algún evento y luego invoca un procedimiento para manejarlo. Un evento podría ser la pulsación de una tecla, un movimiento del ratón, la pulsación de un botón del ratón o la inserción de un diskette.

Dentro de la **estructura de un sistema operativo**, se encuentran:

El **sistema monolítico:** Esta organización, que por mucho es la más común. La estructura consiste en que no existe estructura. El SO se escribe como una colección de procedimientos, cada uno de los cuales puede invocar a cualquiera de los otros, cuando los necesite.

El **sistema en capas:** Es una generalización del enfoque de la figura 1-24 (página 57 del libro de texto). Consiste en organizar el SO en una jerarquía de capas,

cada una cimentada en la que está abajo. El primer sistema construido de esta manera fue THE.

Este sistema tenía seis capas (como se muestra en la figura 1-25, página 58 del libro de texto).

Una generalización adicional del concepto de las capas sucedió en el sistema MULTICS. En lugar de capas, el sistema MULTICS tenía una serie de anillos concéntricos, siendo los interiores más privilegiados que los exteriores.

El corazón del sistema, llamado monitor de la máquina virtual, se ejecuta en el *hardware* simple y realiza la multiprogramación, proporcionando varias **máquinas virtuales** a la siguiente capa superior (ver figura 1-26, página 59 del libro de texto). Sin embargo, a diferencia de los demás sistemas operativos, estas máquinas virtuales no son máquinas extendidas con archivos u otras características adecuadas. En su lugar, son copias exactas del *hardware* simple, con su modo núcleo/usuario, E/S, interrupciones y todo lo demás que posee la máquina real. Puesto que cada máquina virtual es idéntica al *hardware* real, cada uno puede ejecutar cualquier sistema operativo que se ejecute en forma directa sobre el *hardware*.

La labor del ***Exokernels*** consiste en asignar recursos a las máquinas virtuales y luego examinar cualquier intento de usarlos, para garantizar que ninguna máquina utilice los recursos de otra. Cada máquina virtual en el nivel de usuario puede ejecutar su propio SO, aunque cada una está limitada a los recursos que solicitó y le fueron asignados.

El **modelo cliente-servidor** es una tendencia de los sistemas operativos modernos. Consiste en llevar aún más lejos la idea de subir código a capas superiores y quitar lo más que se pueda del modo *kernel*, dejando un *microkernel* mínimo.

El enfoque acostumbrado es implementar casi todo el SO en procesos de usuario. Para solicitar un servicio, un proceso de usuario (denominado **proceso cliente**) envía una solicitud y un **proceso servidor**, realiza el trabajo y devuelve la respuesta.

En el modelo que se muestra en la figura 1-27 (página 62 del libro de texto), lo único que hace el *kernel* es manejar la comunicación entre clientes y servidores.

Otra ventaja del modelo cliente-servidor es su adaptabilidad para usarse en sistemas distribuidos (ver la figura 1-28, página 632 del libro de texto). Si un cliente se comunica con el servidor enviándole mensajes, no necesita saber si el mensaje se maneja en forma local en su propia máquina, o si se le envió por una red a un servidor en una máquina remota. En lo que al cliente concierne, en ambos casos sucede lo mismo: se envió una solicitud y se recibió una respuesta.

Ejercicios sugeridos

A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 1, realice los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 67, 68, 69 y 70: 1, 2, 3, 9, 14, 21, 23.

Resolución de ejercicios sugeridos

Ejercicio 1

Mencione las dos funciones principales de un sistema operativo.

Las dos funciones principales son:

a) Extender la máquina, para que sea más fácil de programar el *hardware* subyacente. El sistema operativo presta una serie de servicios que los programas pueden obtener empleando instrucciones especiales que se conocen como llamadas al sistema.

b) Administrador de recursos, que consiste en efectuar un reparto ordenado y controlado de los procesadores, memorias, dispositivos de E/S, entre los diversos programas que compiten por obtenerlos.

Ejercicio 2

¿Qué es multiprogramación?

La multiprogramación se dio a partir de la tercera generación, y consiste en dividir la memoria en varias partes, con un trabajo distinto en cada partición. Mientras un trabajo estaba esperando a que terminara la E/S, otro podría estar usando la CPU.

Ejercicio 3

¿Qué es el uso de spooling*? ¿Cree que las computadoras personales avanzadas lo usarán como función estándar en el futuro?*

El *spooling* consiste en cargar un trabajo nuevo del disco en la partición recién desocupada para ejecutarlo. Esta característica se utiliza tanto para entrada de datos como en las salidas.

El avance de la informática ha demostrado que el tiempo de CPU es cada vez más valioso, y la oportunidad con que se realicen los trabajos toma mayor relevancia. Las computadoras personales son muy potentes, y es importante aprovechar al máximo este recurso. Ya el *spooling* se utiliza en las salidas (lista de impresión), así que no sería nada raro utilizar un sistema similar para la entrada de datos.

Ejercicio 9

Enumere algunas diferencias entre los sistemas operativos de computadora personal y los de mainframes*.*

Las diferencias son:

- Básicamente, la diferencia principal con respecto a las PC es su capacidad de E/S.
- Están orientadas al procesamiento de varios trabajos a la vez, casi todos los cuales necesitan cantidades enormes de E/S.
- Ofrece tres tipos de servicios: para datos, procesamiento de transacciones y tiempo compartido.
- Algunos se utilizan como servidores de WEB avanzados.

Ejercicio 14

¿Cuál es la diferencia clave entre una interrupción de sistema (TRAP) y una interrupción normal?

La diferencia básica está en que una instrucción TRAP cambia del modo de usuario al modo *kernel* e inicia el sistema operativo. Cuando un sistema de usuario quiere obtener servicios del sistema operativo, se provoca una llamada al sistema.

Las interrupciones normales las hace el *hardware*, generalmente cuando se presenta una situación excepcional, como una división por cero, etc. En ambos casos, el control lo asume el sistema operativo.

Ejercicio 21

¿Qué diferencia fundamental existe entre un archivo especial de bloques y un archivo especial de caracteres?

Los primeros sirven para modelar dispositivos que constan de una colección de bloques direccionales en forma aleatoria, como los discos. Los segundos se usan para modelar impresoras, módems y otros dispositivos que aceptan o producen flujos de caracteres.

Ejercicio 23

El modelo cliente-servidor es muy utilizado en sistemas distribuidos. ¿Puede utilizarse en un sistema de una sola computadora?

El modelo cliente-servidor es eso, un modelo que puede ser adaptado a diferentes plataformas, entre ellas a los sistemas distribuidos, aunque no sea propio de ellos. El concepto básico es: un proceso usuario (cliente) envía la solicitud a un proceso servidor, que realiza el trabajo y devuelve la respuesta. En resumen, sí puede ser adaptado a una sola computadora, solamente que no es lógica esta instalación en una computadora que ya hace el servicio de manera natural.

CAPÍTULO 2

PROCESOS Y SUBPROCESOS

Sumario

- Procesos
- Subprocesos
- Comunicación entre procesos
- Problemas clásicos de comunicación entre procesos
- Calendarización

Propósito del capítulo

En este segundo capítulo trataremos el tema de los procesos y subprocesos, examinaremos sus propiedades y la forma en que se comunican entre sí. Se conocerán los diferentes algoritmos de comunicación que existen, sus escollos y limitaciones, y las soluciones que se han encontrado a cada uno de ellos.

El concepto central de cualquier SO es el proceso; una abstracción de un programa en ejecución. Todo lo demás gira en torno a este concepto, y es importante que el diseñador y el estudiante de sistemas operativos comprenda qué es un proceso.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar qué es un proceso y qué es un subproceso.
- Explicar la diferencia entre un proceso y un subproceso.
- Explicar cómo se realiza la comunicación entre los procesos y subprocesos.
- Identificar los problemas más comunes que se dan en la comunicación de los procesos.
- Identificar, analizar y explicar los diferentes algoritmos que se han diseñado para eliminar o disminuir los problemas de comunicación entre procesos.
- Explicar qué es la calendarización de los procesos.

Guía de lecturas

Para lograr los objetivos descritos anteriormente, es importante que realice las siguientes lecturas:

Subtema	Pp.	Subtema	Pp.
2.1 Procesos	71	2.3.3. Exclusión mutua con espera activa	103
2.1.1. El modelo de procesos	72	2.3.4. Activar y desactivar	108
2.1.2. Creación de procesos	73	2.3.5. Semáforos	110
2.1.3. Terminación de procesos	75	2.3.6. Mutexes	113
2.1.4. Jerarquía de procesos	76	2.3.7. Monitores	115
2.1.5. Estados de procesos	77	2.3.8. Transferencia de mensajes	119
2.1.6. Implantación de procesos	79	2.3.9. Barreras	123
2.2 Subprocesos	81	2.4. Problemas clásicos de comunicación entre procesos	124
2.2.1. El modelo de subprocesos	81	2.4.1. El problema de la cena de los filósofos	125
2.2.2. Uso de subprocesos	85	2.4.2. El problema de los lectores y escritores	128
2.2.3. Implementación de subprocesos en espacio de usuario	90	2.4.3. El problema del barbero dormilón	129
2.2.4. Implementación de subprocesos en el *kernel*	93	2.5. Calendarización	132
2.2.5. Implementaciones híbridas	93	2.5.1. Introducción a la calendarización	132
2.2.6. Activaciones del calendarizador	94	2.5.2. Calendarización en los sistemas por lotes	138
2.2.7. Subprocesos emergentes	96	2.5.3. Calendarización en sistemas interactivos	142
2.2.8. Cómo convertir en código son múltiples subprocesos el de un solo subproceso.	97	2.5.4. Calendarización en sistemas en tiempo real	148
2.3. Comunicación entre procesos	100	2.5.5. Política en comparación con mecanismo	149
2.3.1. Condiciones de competencia	100	2.5.6. Calendarización de subprocesos	150
2.3.2. Regiones críticas	102	2.6. Investigación sobre procesos	151
		2.7. Resumen	157

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

A fin de ocultar los efectos de las interrupciones, los sistemas operativos ofrecen un modelo conceptual, que consiste en procesos secuenciales que se ejecutan en paralelo. Los procesos pueden crearse y terminarse en forma dinámica. Cada proceso tiene su propio espacio de direcciones.

En algunas aplicaciones resulta útil tener múltiples subprocesos de control dentro de un mismo proceso. Estos procesos se calendarizan de manera independiente y cada uno tiene su propia pila, pero todos los subprocesos de un proceso comparten un espacio de direcciones común. Los subprocesos pueden implementarse en el espacio de usuario o en el *kernel*.

Todos los conceptos aquí mencionados se verán de una manera detallada en los siguientes párrafos.

En términos estrictos, la CPU solo puede ejecutar un programa en un instante dado. A lo largo de un segundo puede trabajar con varios programas, lo que da a los usuarios la ilusión de paralelismo. A veces se habla de pseudoparalelismo en este contexto, para contrastarlo con el verdadero paralelismo de *hardware* de los sistemas multiprocesador (tienen dos o más CPU que comparten la misma memoria física).

En el **modelo de procesos** todo el *software* ejecutable de la computadora, que a veces incluye al SO, se organiza en varios procesos secuenciales, o simplemente procesos. Un **proceso** no es más que un programa en ejecución, e incluye los valores que tienen el contador de programa, los registros y las variables. En lo conceptual, cada proceso tiene su propia CPU virtual.

Existen cuatro sucesos principales que causan la creación de procesos.

1. Inicialización del sistema.
2. Ejecución de una llamada al sistema para crear procesos por parte de un proceso en ejecución.
3. Solicitud de un usuario para crear un proceso.
4. Inicio de un trabajo por lotes.

Cuando se arranca un SO, por lo general se crean varios procesos. Algunos son de primer plano, es decir, procesos que interactúan con usuarios y trabajan para ellos. Otros son de segundo plano, que no están asociados con un usuario en particular, sino que tienen una función específica.

Los procesos que permanecen en segundo plano para encargarse de alguna actividad se llaman demonios, por ejemplo: impresión. (Para ampliar este tema, refiérase a las páginas de la 73 a la 75 del libro de texto.)

Tarde o temprano el proceso nuevo terminará, por lo general debido a una de las siguientes condiciones:

1. Terminación normal (voluntaria)
2. Terminación por error (voluntaria)
3. Error fatal (involuntaria)
4. Terminado por otro proceso (involuntario)

En algunos sistemas, cuando un proceso crea otro, el proceso padre y el hijo mantienen cierta asociación. El proceso hijo puede, a su vez, crear más procesos y formar una **jerarquía de procesos**.

En la figura 2-2 (página 78 del libro de texto) se visualiza un diagrama de estado que muestra los tres estados en los que puede estar un proceso:

1. En ejecución. Usando la CPU en ese instante.
2. Listo. Puede ejecutarse; detenido temporalmente para permitir que se ejecute otro proceso.
3. Bloqueado. No puede ejecutarse mientras no ocurra cierto suceso externo.

Para implementar el modelo de procesos, el SO mantiene una tabla (un arreglo de estructuras), llamada **tabla de procesos**, con una entrada por proceso. Esta entrada contiene información acerca del estado del proceso, su contador de programa, apuntador de pila, asignación de memoria, estado de sus archivos abiertos, información contable y de calendarización, y todas las demás cosas que deben guardarse cuando el proceso pasa del estado en ejecución al listo o bloqueado, para que se pueda volver a poner en marcha posteriormente, como si nunca se hubiera detenido.

(Para ahondar más en el tema refiérase a las páginas 71 a la 80 del libro de texto.)

En los sistemas operativos tradiciones cada proceso tiene un espacio de direcciones y un solo subproceso de control. De hecho, esta es la definición de proceso. No obstante, abundan las situaciones en las que es deseable tener varios subprocesos de control en el mismo espacio de direcciones, operando en forma seudoparalela, como si fueran procesos individuales.

Un proceso se puede considerar como una forma de agrupar recursos relacionados. Un proceso tiene un espacio de direcciones que contiene los datos y el texto del programa, así como otros recursos, que podrían incluir archivos abiertos, procesos hijos, alarmas pendientes, manejadores de señales, información contable, etc.

El otro concepto que tiene un proceso es un subproceso de ejecución, o simplemente **subproceso**. Ese tiene un contador de programa e indica cuál instrucción se ejecutará a continuación. Tiene registros, que contienen las variables de trabajo actuales, y tiene una pila que contiene el historial de ejecución, con un marco por cada procedimiento invocado del que todavía no se haya regresado.

Debido a que los subprocesos tienen alguna de las propiedades de los procesos, a veces se les llaman **procesos ligeros**. También, se emplea el término múltiples subprocesos para describir la situación en la que se permiten varios subprocesos en el mismo proceso. (Ver Figura 2-7, página 83 del libro de texto.)

El motivo principal para tener subprocesos es que en diversas aplicaciones se estén realizando varias actividades al mismo tiempo, por lo que algunas de ellas podrían bloquearse de vez en cuando. Al descomponer tal aplicación en múltiples subprocesos secuenciales que se ejecutan casi en paralelo, se simplifica el modelo de programación.

La **implementación de subprocesos en espacio de usuario** es un método que consiste en colocar por completo el sistema de subprocesos en espacio de usuario. El *kernel* no sabe nada de ellos. En lo que a él respecta, está administrando procesos ordinarios, de un solo subproceso. La primera ventaja, y la más obvia, es que puede implementarse un sistema de subprocesos en el nivel de usuario en un SO que no maneje subprocesos. Todos los sistemas operativos solían pertenecer a esta categoría, y todavía subsisten algunos.

Ahora consideremos la posibilidad de que el *kernel* esté enterado de la existencia de subprocesos y los administre. No se necesita un sistema de tiempo de ejecución en cada uno.

Además, no existe una tabla de subprocesos en cada proceso. En vez de eso, el *kernel* tiene una tabla de subprocesos que lleva el control de todos los subprocesos del sistema. Cuando un subproceso quiere crear o destruir otro subproceso, emite una llamada al *kernel*, que se encarga de crearlo o destruirlo actualizando su tabla de subprocesos.

Se han investigado varias formas de combinar las ventajas de los subprocesos en el nivel de usuario y en el nivel de *kernel*. Una de ellas es usar subprocesos en el nivel de *kernel*, y luego multiplexar subprocesos en el nivel de usuario en uno de los subprocesos de *kernel* o en todos, como se muestra en la figura 2-14 (página 94 del libro de texto. A esto se le llama **implementación híbrida**).

Lo que se busca con las **activaciones del calendarizador** es imitar la funcionalidad de los subprocesos de *kernel*, pero con el desempeño y la mayor flexibilidad que suelen tener los sistemas de subprocesos implementados en el espacio de usuario.

Los subprocesos suelen ser útiles en sistemas distribuidos. Una ventaja clave de los **subprocesos emergentes** es que, debido a que son totalmente nuevos, no tienen historial –registros, pila, etc.- que deba restaurarse. Cada uno inicia desde cero y todos son idénticos, y esto agiliza su creación.

(Para ahondar más en el tema refiérase a las páginas 81 a la 100 del libro de texto.)

Es necesaria la **comunicación entre procesos** de preferencia con un mecanismo bien estructurado que no utilice interrupciones.

Existen tres aspectos que cuidar. El primero, cómo puede un proceso pasar información a otro. El segundo tiene que ver con cómo asegurarse que dos o más procesos no se estorben al realizar actividades cruciales. El tercero con el ordenamiento correcto cuando existen dependencias: si el proceso A produce datos y el proceso B los imprime, B tendrá que esperar hasta que A haya producido unos datos antes de comenzar a imprimir.

El almacenamiento compartido podría estar en la memoria principal (tal vez es una estructura de datos del *kernel*), o bien ser un archivo compartido. La ubicación de la memoria compartida no altera la naturaleza de la comunicación ni los problemas que se presenten.

Situaciones como esta en la que dos o más procesos están leyendo o escribiendo datos compartidos y el resultado final depende de quien se ejecuta y precisamente cuándo, se denomina **condiciones de competencia**.

La clave para evitar problemas en las regiones críticas, y en muchas otras situaciones en las que se comparte memoria, archivos o cualquier otra cosa, es hallar alguna forma de impedir que dos o más procesos lean o escriban los datos compartidos al mismo tiempo. En otras palabras, lo que necesitamos es exclusión mutua, es decir, alguna forma de asegurarnos de que si un proceso está utilizando una variable compartida o un archivo compartido, los demás procesos no podrán hacer lo mismo.

La parte del programa en la que se tiene acceso a la memoria compartida se denomina **región crítica** o **sección crítica**. Si pudiéramos organizar las cosas de modo que dos procesos nunca estuvieran en sus regiones críticas al mismo tiempo, podríamos evitar las competencias.

La **exclusión mutua con espera activa** tiene varias soluciones:

a. La **inhabilitación de interrupciones**: Es la solución más sencilla. Es hacer que cada proceso inhabilite todas las interrupciones inmediatamente después de ingresar en su región crítica y las vuelva a habilitar justo antes de salir de ella.

b. Las **variables de bloqueo**: Como ejemplo, consideremos el uso de una sola variable compartida (de bloqueo) que inicialmente es cero. Cuando un proceso quiere entrar en su región crítica, primero prueba el bloqueo. Si este es cero, el proceso lo establece a uno (1) y entra en la región crítica.

c. La **alternativa estricta**: Requiere que los procesos alternen en forma estricta su entrada a las regiones críticas respectivas. Ninguno de los procesos debe entrar dos veces seguidas. Los turnos no son una buena idea cuando uno de los procesos es mucho más lento que el otro.

d. La **solución de Peterson**: (Ver detalle en página 105 del libro de texto.)

e. La instrucción **TSL** (***Test and Set Lock***): Es un método que requiere poca ayuda del *hardware*. La CPU que ejecuta la instrucción TSL cierra el bus de memoria para prohibir a las demás CPU's el acceso a la memoria hasta terminar. En esencia, esta solución, cuando un proceso desea entrar a su región crítica, verifica si está permitida la entrada. Si no, el proceso se queda esperando hasta obtener el permiso, por lo que tiene el inconveniente de la espera ocupada.

Ahora veamos algunas primitivas de comunicación entre procesos que se bloquean, en lugar de desperdiciar tiempo de CPU, cuando no se les permite entrar en su región crítica.

Una de las soluciones más sencillas para activar y desactivar es el par ***sleep*** y ***wakeup***. *Sleep* es una llamada al sistema que hace que el invocador se bloquee, es decir, quede suspendido hasta que otro proceso lo active. La llamada *wakeup* tiene un parámetro, el proceso por activar. De manera alternativa, tanto *sleep* como *wakeup* tienen un parámetro: una dirección de memoria que sirve para relacionar llamadas *sleep* con llamadas *wakeup*. (Ver ejemplo del productor-consumidor, página 108 del libro de texto.)

Los **semáforos**: Es el uso de una variable entera para contar el número de llamadas *wakeup* guardadas para uso futuro. Un semáforo puede tener el valor 0, que indica que no se llamaron llamadas *wakeup*, o algún valor positivo si hay llamadas pendientes.

Utiliza dos opciones ***down*** (generalizaciones de *sleep* y *wakeup*, respectivamente). La aplicación *down* aplicada a un semáforo determina si su valor es mayor que 0. En tal caso, decrementa el valor (es decir, gasta un despertar almacenado) y simplemente continúa. Si el valor es 0, el proceso se desactiva sin terminar la operación *down* por el momento.

Los semáforos binarios son semáforos que se inicializan con valor de 1 (uno) y que son usados por dos o más procesos para garantizar que solo uno de ellos pueda ingresar a su región crítica.

Cuando no se necesita la capacidad de contar del semáforo, suele utilizarse una versión simplificada del semáforo llamada ***mutex*** (abreviatura de "exclusión mutua", en inglés).

Un *mutex* es una variable que puede estar en uno de dos estados: desbloqueado o bloqueado. Por ello, solo se necesita un bit para representarlo, aunque en la práctica es común que se utilice un entero, de modo que 0 signifique desbloqueado y todos los demás valores signifiquen bloqueado.

El **monitor** se define como una primitiva de sincronización de alto nivel. Un monitor es una colección de procedimientos, variables y estructuras de datos que se agrupan en un tipo especial de módulo o paquete.

Una propiedad importante del monitor para conseguir la exclusión mutua es que solo uno de los procesos puede estar activo en un monitor en cada momento.

El método de **transferencia de mensajes** es un método de comunicación entre procesos. Utiliza dos primitivas: *send* y *receive*. Al igual que los semáforos y a diferencia de los monitores, son llamadas al sistema o construcciones del lenguaje.

Nuestro último mecanismo de sincronización, llamado **barreras**, está pensado para grupos de procesos más que para situaciones de tipo productor-consumidor en las que solo existen dos procesos. Algunas aplicaciones se dividen en fases y tienen la regla de que ningún proceso puede pasar a la siguiente fase antes de que todos los procesos estén listos para hacerlo. Este proceso puede lograrse colocando una barrera al final de cada fase. (Ver figura 2-30, en la página 124 del libro de texto.)

(Para profundizar estos comentarios, refiérase a las páginas de la 81 a la 124 del libro de texto.)

Entre los problemas clásicos de comunicación entre procesos, se encuentran:

La cena de los filósofos: Este programa utiliza un arreglo de semáforos, uno por filósofo, lo que permite a los filósofos hambrientos bloquearse si los tenedores que necesitan están ocupados. Se introduce el concepto de inanición, que consiste en que todos los programas se ejecutan de manera indefinida pero no logran avanzar.

Este problema es útil para modelar procesos que están compitiendo por obtener acceso exclusivo a un número limitado de recursos, como dispositivos de E/S.

Los lectores y escritores: En este, se modela el acceso a una base de datos. Es aceptable tener varios procesos leyendo la base de datos al mismo tiempo, pero si un proceso está actualizando (escribiendo en ella) la base de datos, ningún otro proceso podrá tener acceso a ella, ni siquiera los lectores.

La desventaja de esta solución es que permite menos concurrencia y, por lo tanto, merma el desempeño.

El barbero dormilón: Este problema es similar a diversas situaciones de colas. Esta solución utiliza tres semáforos: clientes, que cuenta los clientes que están esperando; barberos, el número de barberos que están inactivos (0 o 1) esperando clientes y *mutex*, que controla la exclusión mutua.

(Para ampliar sobre estos temas refiérase a las páginas 124 a la132 del libro de texto.)

Una computadora multiprogramada suele tener varios procesos compitiendo por la CPU al mismo tiempo. Si solo se cuenta con una CPU, es preciso decidir cuál proceso a ejecutar será el siguiente. La parte del SO que toma la decisión se denomina **calendarizador**, y el algoritmo que usa, se llama **algoritmo de calendarización**.

Dado que el tiempo de CPU es un recurso preciado un buen calendarizador, puede ser crucial para el desempeño percibido y la satisfacción de los usuarios.

El comportamiento de los procesos: Casi todos los procesos alternan ráfagas de cómputo con solicitudes E/S (de disco), como se muestra en la figura 2-37 (página 134 del libro de texto). Por lo general, la CPU opera durante un tiempo sin parar, y luego se emite una llamada al sistema para leer de un archivo o escribir en uno. Cuando la llamada al sistema termina, la CPU vuelve a computar hasta que necesita o tiene que escribir más datos, y así en forma sucesiva.

Cuándo calendarizar

Existen diversas situaciones en las que es necesario calendarizar:

La primera es cuando se crea un proceso, pues hay que decidir si se ejecutará el proceso padre o el hijo. El calendarizador está en su derecho de escoger al padre o al hijo para ejecutar a continuación.

La segunda situación se presenta cuando un proceso termina, ya que se debe tomar una decisión de calendarización. Se deberá escoger otro del conjunto de procesos listos. Si ningún proceso está listo, lo normal es que se ejecute un proceso inactivo suministrado por el sistema.

La tercera situación ocurre cuando un proceso se bloquea por E/S, un semáforo o algún otro motivo, y es preciso escoger otro proceso que se ejecute. En ocasiones, el motivo del bloqueo puede afectar la decisión.

Un algoritmo de calendarización **no expropiativo** (o no referente) escoge el proceso que se ejecutará y luego simplemente le permite ejecutarse hasta que se bloquee o hasta que sea cedido por la CPU de manera voluntaria.

Un algoritmo de calendarización **expropiativo** (o preferente) escoge un proceso y le permite ejecutarse durante un tiempo establecido. Si al término de ese tiempo el

proceso continúa en ejecución, se le suspende, y el calendarizador escoge otro proceso para que se ejecute (si existe alguno disponible).

Categorías de algoritmos de calendarización

Tres entornos que vale la pena distinguir son:

1. **Por lotes:** no existen usuarios esperando impacientes ante sus terminales, a que el sistema responda con rapidez. Por ello, suelen ser aceptables los algoritmos **no expropiativos** o los **expropiativos** con intervalos de tiempo largos para cada proceso.

2. **Interactivo:** La expropiación es indispensable para evitar que un proceso acapare la CPU y niegue servicio a otros.

3. **Tiempo real:** En los sistemas con restricciones en tiempo real a veces no es necesaria la expropiación, porque los procesos saben que tal vez no se ejecuten durante mucho tiempo y, por lo general, realizan su trabajo y se bloquean rápido.

Las metas de los algoritmos de calendarización

Algunos objetivos dependen del entorno (lotes, interactivo, tiempo real), pero también existen objetivos que son deseables en todos los casos. (Algunos de ellos se presentan en la figura 2-38, página 137 del libro de texto.)

El **rendimiento** es el número de trabajos que el sistema termina por hora.

El **tiempo de retorno** es el promedio estadístico del tiempo que transcurre entre que se presenta un trabajo por lotes y el momento en que termina.

El **tiempo de respuesta** es el tiempo que transcurre entre que se emite un comando y se observa el resultado.

Para la **calendarización en sistemas por lotes,** existen varios métodos:

- **Primero en llegar, primero en ser atendido:** Es el algoritmo de calendarización más sencillo, el primer proceso que llega, es el primer proceso en ser atendido. Es no expropiativo.

- **Trabajo más corto primero:** Si existen varios trabajos de la misma importancia en la cola de entrada, el calendarizador escoge el trabajo más corto primero.

- **Tiempo restante más corto a continuación:** El calendarizador siempre escoge el proceso con base en el tiempo que falta para que termine de

ejecutarse el proceso en ejecución. Cuando llega un trabajo nuevo, su tiempo total se compara con el tiempo que resta para que el proceso actual termine de ejecutarse. Si es menor, el proceso actual se suspende y se pone en marcha el trabajo recién llegado.

- **Calendarización de tres niveles:** El primer nivel es el calendarizador de admisión. Este algoritmo decide cuáles trabajos admitirá en el sistema. Los demás se mantienen en la cola de entrada hasta que se les seleccione. El segundo nivel de calendarización, **calendarizador de memoria**, implica decidir cuáles procesos se conservarán en la memoria y cuáles se enviarán a disco. El tercer nivel consiste en decidir cuál de los procesos listos que están en la memoria principal se ejecutará a continuación.

En la **calendarización en sistemas interactivos** existen varios métodos:

- **Calendarización por turno circular** (*Round-Robin*): Es uno de los más antiguos y sencillos, equitativos y ampliamente utilizados. A cada proceso se le asigna un intervalo de tiempo, llamado cuanto, durante el que se le permitirá ejecutarse. Si al término del cuanto el proceso se sigue ejecutando, se le expropia la CPU para asignársela a otro proceso.

- **Calendarización por prioridades**: A cada proceso se le asigna una prioridad, y el proceso listo que tenga la prioridad más alta es el que se ejecuta. (Ver figura 2-42, página 2-42 del libro de texto.)

- **Múltiples colas**: Se crean varias colas. Cada cola tiene una prioridad. Los procesos de la clase (prioridad) más alta se ejecutan durante un cuanto. Los procesos de la siguiente clase más alta se ejecutan durante dos cuantos y así sucesivamente. Cada vez que un proceso se gasta, todos los cuantos asignados se bajan a la clase inmediata inferior.

- **Proceso más corto a continuación**: Consiste en estimar valores con base en comportamientos anteriores y ejecutar el proceso que tenga el tiempo de ejecución estimado más corto.

- **Calendarización garantizada**: El sistema necesita llevar la cuenta de cuánto tiempo de CPU ha recibido cada proceso desde su creación. Luego se calcula el tiempo de CPU al que cada uno tiene derecho, que sería el tiempo desde la creación dividido entre n. Puesto que también se conoce el tiempo de CPU que ha tenido cada proceso, se puede calcular el cociente del tiempo de CPU consumido realmente entre el tiempo al que el proceso tiene derecho. El algoritmo consiste, entonces, en ejecutar el proceso cuyo cociente es más bajo, hasta que el cociente rebase al de su competidor más cercano.

- **Calendarización por lotería**: La idea fundamental consiste en entregar a los procesos "billetes de lotería" para los distintos recursos del sistema, como el tiempo de CPU. Cada vez que se deba tomar una decisión de calendarización, se escoge un "billete de lotería" al azar, y el proceso que lo tiene obtiene el recurso.

- **Calendarización por porción equitativa**: En este modelo a cada usuario se le asigna cierta fracción de tiempo de CPU y el calendarizador escoge los procesos a modo de respetar esa división.

Los sistemas en tiempo real se clasifican en general como en tiempo real estricto. Esto implica que hay plazos absolutos que deberán cumplirse, pase lo que pase, y en tiempo real no estricto, en los que pueden tolerarse incumplimientos ocasionales, aunque indeseables, de los pasos.

En ambos casos, el comportamiento en tiempo real se logra dividiendo el programa en varios procesos cuyo comportamiento es predecible y se conoce con antelación. Por lo general, tales procesos son cortos y pueden terminar su trabajo en mucho menos de un segundo.

Política en comparación con mecanismo: Para comprender este tema referirse a la sección 2.5.5 (páginas 149 y 150 del libro de texto).

En la **calendarización de subprocesos**, si cada uno de varios procesos tiene múltiples subprocesos, tenemos dos niveles de paralelismo: procesos y subprocesos. La calendarización en tales sistemas presenta diferencias considerables, dependiendo de si soportan subprocesos en el nivel de usuario o en el nivel de *kernel* (o en ambos).

Una diferencia importante entre los subprocesos en el nivel de usuario y en el de *kernel* es el desempeño. Una conmutación de subprocesos en el nivel de usuario requiere unas cuantas instrucciones de máquina. En el caso de los subprocesos en el nivel de *kernel*, se requiere una conmutación de contexto completa.

Otro factor importante es que los subprocesos en el nivel de usuario pueden utilizar un calendarizador de subproceso específico para la aplicación. En ese nivel se sabe lo que hace cada subproceso. Con subprocesos en el nivel de *kernel*, este nunca sabe lo que hace cada subproceso.

(Para ampliar sobre estos temas, refiérase a las páginas 132 a la 151 del libro de texto.)

Ejercicios sugeridos

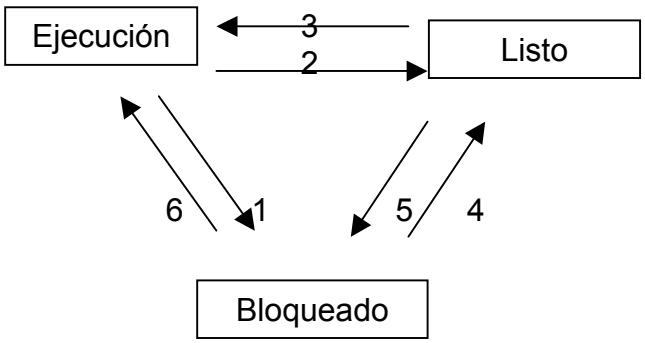
A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 2, realice

los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 153, 154, 155, 156, 157 y 158: 1, 3, 18, 19, 25, 31.

Resolución de ejercicios sugeridos

Ejercicio 1

En la figura 2-2 se muestran tres estados de procesos. En teoría, si hay tres estados, podría haber seis transiciones, dos por cada estado. No obstante, solo se muestran cuatro transiciones. ¿Hay circunstancias en las que podría darse alguna de las transiciones faltantes, o ambas?



Es posible que se diera la situación apuntada arriba. Sin embargo, el diseño del sistema operativo sería más complicado, y se trata de simplificar. Imagine que tengamos simultáneamente las transiciones 3 y 6 (obviando la transición 4). Sería sumamente complicado para el sistema operativo determinar a cuál proceso le da prioridad, al que viene de desbloquearse o al que viene del estado Listo.

Si hubiera una transición 5, un proceso podría quedarse "dormido" entre los procesos **Listo** y **Bloqueado**. Las transiciones 5 y 6 se han eliminado para evitar mayores problemas a los que de por sí, ya debe atender el sistema operativo.

Ejercicio 3

En todas las computadoras actuales, al menos una parte de los manejadores de interrupciones se escribe en lenguaje ensamblador. ¿Por qué?

Porque se requiere de un lenguaje de bajo nivel (más cerca del sistema operativo) que haya sido diseñado especialmente para este propósito, comunicarse directamente con el sistema operativo.

Ejercicio 18

¿Qué es una condición de competencia?

Una situación de competencia se presenta cuando dos o más procesos están leyendo o escribiendo datos compartidos y el resultado final depende de quién ejecute y precisamente cuándo.

Ejercicio 19

Cuando se está desarrollando una computadora, lo común es simularla con un programa que ejecuta una instrucción a la vez. Incluso los multiprocesadores se simulan en forma estrictamente secuencial. ¿Es posible que se presente una condición de competencia cuando no hay sucesos simultáneos, como en la simulación?

Si, y para eso se crean estos ambientes de simulación, para garantizar que la instrucción anterior, una vez finalizada, haya liberado el o los recursos que estaba utilizando. Si no lo hizo, la siguiente instrucción trataría de accesar un recurso no liberado y así, se daría la condición de competencia.

Ejercicio 25

Si un sistema sólo tiene dos procesos. ¿Tiene sentido usar una barrera para sincronizarlos? ¿Por qué sí o por qué no?

No tiene sentido, pues hay métodos más sencillos, diseñados especialmente para dos procesos, para administrar las competencias. Las barreras fueron diseñadas para grupos de procesos.

Ejercicio 31

En la solución del problema de la cena de los filósofos, en la figura 2.33 (en el libro está errónea la referencia)*, ¿por qué se asigna HAMBRE a la variable de estado en el procedimiento tomar_tenedor?*

Porque esta variable es la que le indica al procedimiento si el “filósofo” debe proceder a tomar el tenedor para comer, si no hay una “necesidad” (hambre), no hay razón de ejecutar el proceso “comer”.

CAPÍTULO 3

BLOQUEOS IRREVERSIBLES

Sumario

- Recursos
- Introducción a los bloqueos irreversibles
- El algoritmo del avestruz
- Detección de bloqueos irreversibles y recuperación posterior
- Cómo evitar los bloqueos irreversibles
- Prevención de bloqueos irreversibles
- Otros aspectos

Propósito de capítulo

Los sistemas de cómputo abundan en recursos que sólo pueden ser utilizados por un proceso a la vez. Si dos procesos escriben de manera simultánea en la impresora, el resultado es basura. Por ello, todos los sistemas operativos tienen la prerrogativa de otorgar a un proceso acceso exclusivo a ciertos recursos.

En muchas aplicaciones, un proceso necesita acceso exclusivo no a un recurso, sino a varios. Cuando un proceso retiene un recurso por tiempo indefinido y otro recurso lo solicita, y en vez de liberarlo solicita otro recurso y se mantienen bloqueados ambos procesos, a esta situación se le llama **bloqueo irreversible**. También pueden presentarse bloqueos entre máquinas.

El propósito de este capítulo es conocer los diferentes tipos de bloqueos irreversibles que se pueden dar (orígenes y características) y presentar una serie de algoritmos que se han diseñado para su prevención, detección y solución.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar el concepto de recursos expropiables y no expropiables.
- Identificar las situaciones que pueden dar origen a un proceso irreversible.
- Identificar y analizar algunos algoritmos que se han diseñado para evitar los bloqueos irreversibles.
- Identificar y conocer los métodos que se han diseñado para la detección de este tipo de bloques
- Identificar y analizar los algoritmos que se han diseñado para la recuperación posterior de los procesos bloqueados
- Identificar y analizar algunos de las maneras en que se pueden prevenir los bloqueos irreversibles.

Guía de lecturas

Para lograr los objetivos descritos anteriormente, es importante que realice las siguientes lecturas:

Subtema	Pp.	Subtema	Pp.
3.1 Recursos	160	3.5.3 El algoritmo del banquero para un solo recurso	178
3.1.1 Recursos expropiables y no expropiables	160	3.5.4 El algoritmo del banquero para múltiples recursos	179
3.1.2 Adquisición de recursos	161	3.6 Prevención de bloqueos Irreversibles	180
3.2 Introducción a losbloqueos irreversibles	163	3.61 Cómo atacar la condición de exclusión mutua	181
3.2.1 Condiciones para el bloqueo irreversible	164	3.6.2 Cómo atacar la condición de retener y esperar	181
3.2.2 Modelado de bloqueos irreversibles	164	3.6.3 Cómo atacar la condición de no expropiación	182
3.3 El algoritmo del avestruz	167	3.6.4 Cómo atacar la condición de espera circular	182
3.4 Detección de bloqueos Irreversibles y recuperación posterior	168	3.7 Otros aspectos	183
3.4.1 Detección de bloqueos irreversibles con un recurso de cada tipo	168	3.71 Bloqueos de dos fases	183
3.4.2 Detección de bloqueos irreversibles con múltiples recursos de cada tipo	171	3.7.2 Bloqueos irreversibles que no son por recursos	184
3.4.3 Cómo recuperarse de un bloqueo irreversible	173	3.7.3 Inanición	184
3.5 Cómo evitar los bloqueos irreversibles	175	3.8 Investigación sobre los bloqueos Irreversibles	185
3.5.1 Trayectorias de recursos	175	3.9 Resumen	185
3.5.2 Estados seguros e inseguros	176		

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

Los sistemas de cómputo abundan en recursos que solo pueden ser utilizados por un proceso a la vez. Por ejemplo, si dos procesos escriben de manera simultánea en la impresora, el resultado es basura. Por ello, todos los sistemas operativos tienen la prerrogativa de otorgar a un proceso (en forma temporal) acceso exclusivo a ciertos recursos.

Los bloqueos irreversibles son un problema que puede presentarse en cualquier SO. Ocurren cuando a cada proceso de un grupo se le ha otorgado acceso exclusivo a ciertos recursos, y cada uno quiere aún más recursos que están en manos de otro proceso del grupo. Todos los procesos están bloqueados y ninguno podrá ejecutarse más.

Los bloqueos irreversibles pueden ocurrir cuando se han otorgado a los procesos acceso exclusivo a dispositivos, archivos, etc.

Un **recurso** puede ser un dispositivo de *hardware* o información. Las computadores generalmente tiene muchos recursos. En el caso de algunos, podría haber varias instancias, como: tres unidades de cinta. Si se cuenta con varias copias de un recurso, cualquiera de ellas puede utilizarse para atender una solicitud de ese recurso.

Un **recurso** es cualquier cosa que solo un proceso puede usar en un instante dado.

Un **recurso expropiable** es uno que se le puede quitar al proceso que lo tiene sin causar daños. La memoria es un ejemplo de recurso expropiable.

Un **recurso no expropiable**, en cambio, no puede quitársele a su actual dueño sin hacer que el cómputo falle.

La secuencia de sucesos necesarios para usar un recurso son:

1. Solicitar el recurso
2. Usar el recurso
3. Liberar el recurso

El uso de algunos tipos de recursos, como los registros de un sistema de base de datos, es administrado por los procesos de usuario. Una forma de permitir a los usuarios administrar los recursos es asociar un semáforo con cada recurso. Todos estos semáforos al principio tienen el valor 1. También pueden utilizarse *mutexes*. Luego se implementan los tres pasos que anteriormente se mencionaron, ejecutando *down* con el semáforo para adquirir el recurso, usándolo y, por último, ejecutando up con el semáforo para liberar el recurso.

(Para ahondar más en el tema, refiérase a las páginas 159 a la 163 del libro de texto.)

Un conjunto de procesos cae en un **bloque irreversible** si cada proceso del conjunto está esperando un suceso que solo otro proceso del conjunto puede causar.

Cada miembro está esperando un recurso poseído por un proceso que también cayó en el bloqueo irreversible.

Para que exista un bloqueo irreversible deben cumplirse cuatro condiciones:

- Condición de exclusión mutua: Cada recurso está asignado a exactamente un proceso o está disponible.

- Condición de retención y espera: Los procesos que tienen recursos previamente otorgados pueden solicitar otros recursos.

- Condición de no expropiación: Los recursos ya otorgados no pueden arrebatarse al proceso que los tiene; este debe liberarlos en forma explícita.

- Condición de espera circular: Debe haber una cadena circular de dos o más procesos, cada uno de los cuales está esperando un recurso que está en manos del siguiente miembro de la cadena.

Si falta alguna de ellas, no podrá ocurrir.

(El **modelado de bloqueos irreversibles** puede visualizarse en las figuras 3-3 y 3-4, páginas 165 y 166, respectivamente, del libro de texto.)

En general, se usan cuatro estrategias para manejar bloqueos irreversibles.

1. Simplemente ignorar el problema: Quizá si nos olvidamos de él, él se olvidará de nosotros (analizar la sección 3.3, página 167 del libro de texto).

2. Detección y recuperación: Dejar que se presenten bloqueos irreversibles, detectarlos y tomar medidas (analizar la sección 3.4, página 168 del libro de texto).

3. Evitación dinámica: Mediante una asignación cuidadosa de recursos, analizar la sección 3.5, página 175 del libro de texto.

4. Prevención, anulando en forma estructural una de las cuatro condiciones necesaria para que haya un bloqueo irreversible (analizar la sección 3.6, página 180 del libro de texto).

La mayoría de los sistemas operativos pueden padecer bloqueos irreversibles que ni siquiera se detectan mucho menos se truncan en forma automática. Una estrategia razonable para el programa que emitió la llamada sería esperar un tiempo aleatorio o intentarlo otra vez (**algoritmo del avestruz**).

Existen varios métodos para la detección de bloqueos irreversibles, los cuales se describen a continuación:

- **Detección de bloqueos irreversibles con un recurso de cada tipo:** Solo hay un recurso de cada tipo. Un sistema con estas condiciones podría tener un escáner, una grabadora de CD, un graficador y una unidad de cinta: No más de un elemento de cada clase de recurso.

  (Analizar figura 3-5 y estudiar el algoritmo sugerido en la página 170 del libro de texto.)

- **Detección de bloqueos irreversibles con múltiples recursos de cada tipo:** Si hay varias copias de algunos de los recursos, se requiere un enfoque distinto para detectar bloqueos irreversibles.

  El algoritmo para detectar bloqueos irreversibles se basa en la comparación de vectores. En un principio, decimos que ninguno de los procesos está marcado. A medida que el algoritmo avanza, se irán marcando procesos para indicar que pueden terminar y, por lo tanto, que no han caído en un bloqueo irreversible. Cuando el algoritmo finaliza, todos los procesos que no estén marcados habrán caído en un bloqueo irreversible.

Para **recuperarse de un bloqueo irreversible**, existen varias maneras que a continuación se detallan:

- **Recuperación mediante expropiación**: En algunos casos sería posible arrebatarle de manera temporal un recurso a su actual poseedor y dárselo a otro proceso, en muchos casos podría requerirse una intervención manual, sobre todo en los sistemas operativos de procesamiento por lotes que se ejecutan en *mainframes*.

Esta forma de recuperación suele ser difícil o imposible.

- **Recuperación mediante reversión:** Si los diseñadores del sistema y los operadores de la máquina saben que existe una probabilidad elevada de bloqueos irreversibles, pueden disponer que los procesos pasen por puntos de verificación en forma periódica. En un punto de verificación, el estado del proceso se graba en un archivo para poder reiniciarlo más tarde. El punto de verificación no solo contiene la imagen de memoria, sino también el estado de los recursos. Es decir, cuáles están asignados en la actualidad al proceso.

- **Recuperación de eliminación de procesos:** La forma más burda, pero eficaz, de romper un bloqueo irreversible es eliminar uno o más procesos. Una posibilidad sería eliminar un proceso del ciclo. Con un poco de suerte, los demás procesos podrán continuar. Si esto no ayuda, podría repetirse hasta romper el ciclo.

  En este esquema, el proceso de eliminar se escoge con mucho cuidado, porque está reteniendo recursos que algún proceso del ciclo necesita.

(Para ahondar más en el tema, refiérase a las páginas 168 a la 175 del libro de texto.)

Para evitar los **procesos irreversibles**: Los principales algoritmos para evitar bloqueos irreversibles se basan en el concepto de **estados seguros**.

Decimos que un **estado es seguro** si no ha caído en un bloqueo irreversible y existe algún orden de calendarización en el que todos los procesos pueden ejecutarse hasta terminar, aunque todos ellos soliciten en forma repentina y de inmediato su número máximo de recursos.

La diferencia entre un **estado seguro** y uno **inseguro** es que desde un estado seguro, el sistema puede garantizar que todos los procesos terminarán; desde un estado inseguro no se puede ofrecer semejante garantía.

A continuación se mencionan los métodos siguientes:

a. Un algoritmo de calendarización que puede evitar bloqueos irreversibles se conoce como **algoritmo del banquero** y es una extensión del algoritmo para detectar bloqueos irreversibles que presentamos en la sección 3.4.1.

   Este algoritmo examina cada solicitud en el momento en que se hace, y determina si otorgar lo que se pide a un estado seguro o no. Si es así, concede la solicitud; de lo contrario, la pospone. Para ver si un estado es seguro, el banquero determina si tiene suficientes recursos para satisfacer a algún cliente.

b. El algoritmo del banquero para múltiples recursos se analiza con base en la sección 3.5.3., páginas 179 y 180 del libro de texto.

(Para ahondar más en el tema, refiérase a las páginas 175 a la 180 del libro de texto.)

Si se quiere atacar la condición de exclusión mutua se debe evitar asignar un recurso si no es absolutamente necesario, y tratar de asegurarse de que el número de procesos que podrían solicitar el recurso sea lo más pequeño posible.

Para atacar la condición de retener y esperar se debe evitar que los procesos que tienen recursos esperen cuando tratan de adquirir más recursos podremos evitar los bloqueos irreversibles. Una forma de lograr este objetivo es exigir que todos los procesos soliciten todos sus recursos antes de comenzar a ejecutarse. Si todo está disponible, se asignará al proceso lo que necesita y podrá ejecutarse hasta terminar.

Si uno o más recursos están ocupados, no se asignará nada y el proceso tendrá que esperar.

(Para atacar la condición de no expropiación, estudie la sección 3.6.3, página 182 del libro de texto.)

Para atacar la condición de espera circular existen varias formas: una sería simplemente tener una regla en el sentido de que cualquier proceso tiene derecho tan solo a un recurso en todo momento. Si necesita un segundo recurso, deberá liberar el primero.

Otra forma de evitar la espera circular es con una numeración global de todos los recursos. Los procesos pueden solicitar recursos cuando los necesiten, pero todas las solicitudes deben efectuarse en orden numérico. (Ver figura 3-14, página 183 del libro de texto.)

Los bloqueos de dos fases: En muchos sistemas de bases de datos, una operación que se presenta con frecuencia es solicitar bloqueos para varios registros y luego actualizar todos esos registros. Si se están ejecutando varios procesos al mismo tiempo, el peligro de un proceso irreversible es muy real.

En la primera fase, el proceso trata de bloquear todos los registros que necesita, uno por uno. Si lo logra, inicia la segunda fase, donde realiza sus actualizaciones y desbloquea. En la primera fase no se efectúa trabajo real.

El bloqueo irreversible que no son por recursos: También pueden presentarse bloqueos irreversibles en otras situaciones, incluso algunas en que no intervienen recursos. Por ejemplo, puede suceder que dos procesos se bloqueen mutuamente esperando que el otro haga algo. Esto sucede a menudo con los semáforos.

La **inanición** es un problema íntimamente relacionado con los bloqueos irreversibles. En un sistema dinámico se presentan en forma continua solicitudes de recursos. Se necesita alguna política para decidir quién recibe qué recurso y cuándo. Esta política, aunque al parecer es razonable, puede hacer que algunos procesos nunca sean atendidos, aunque no hayan caído en un bloqueo irreversible.

La inanición puede evitarse utilizando una política de asignación de recursos tipo primero que llega, primero que se atiende. Así, el proceso que ha esperado más tiempo será el que reciba servicio a continuación. Tarde o temprano llegará a ser el más antiguo y obtendrá el recurso requerido.

(Para ampliar mejor los conocimientos acerca de los temas aquí expuestos, refiérase a las secciones 3.5, 3.6 y 3.7, páginas 175 a la 185 del libro de texto.)

Ejercicios sugeridos

A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 3, realice los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 186, 187 y 188: 1, 2, 14, 18, 25, 28.

Resolución de ejercicios sugeridos

Ejercicio 1

Estudiantes que trabajan en PC individuales en un laboratorio de computación envían los archivos que desean imprimir a un servidor que hace spooling *en su disco duro. ¿En qué condiciones podría presentarse un bloqueo irreversible, si el espacio en disco para el* spooling *de impresión es limitado? ¿Cómo puede evitarse el bloque irreversible?*

Se presentaría si el servidor de archivos no es capaz de liberar el espacio de impresión cada vez que termine una tarea de impresión. En cierto momento, el *spooler* puede llenarse pero, si no se libere el espacio, el sistema operativo podría asumir que no hay espacio para más trabajos y quedarse esperando que se libere, cosa que no va a suceder. Lo anterior puede evitarse justamente mediante un algoritmo que indique al sistema operativo que se ha finalizado un trabajo de impresión y, por ende, ese espacio quedaría disponible para un nuevo trabajo.

Ejercicio 2

En la pregunta anterior, ¿cuáles recursos son expropiables y cuáles no?

La impresora no es expropiable, el *spooler* de impresión sí.

Ejercicio 14

Un sistema tiene dos procesos y tres recursos idénticos. Cada proceso necesita un máximo de dos recursos. ¿Es posible caer en un bloqueo irreversible? Explique su respuesta.

Sí es posible caer en un bloqueo irreversible. Veamos el siguiente ejemplo:

El proceso A tiene el recurso 1 y el proceso B tiene el recurso 2. Si el proceso A requiere del recurso 2 para completar su trabajo y el proceso B, simultáneamente, requiere del recurso 1 para ejecutar su trabajo, ambos se quedarían esperando que el otro proceso libere su recurso para poder tener acceso a él. Ambos quedarían en un bloqueo irreversible.

Ejercicio 18

Una computadora tiene seis unidades de cinta y n procesos compiten por ellas. Cada proceso podría necesitar dos unidades. ¿Con qué valores de n el sistema está a salvo de bloqueos irreversibles?

Con *n* igual a 1 o 2 o 3, siempre y cuando no sean las mismas unidades de cinta.

Ejercicio 25

Una forma de prevenir bloqueos irreversibles consiste en eliminar la condición de retener y esperar. En el texto se propuso que, antes de pedir un recurso nuevo, los procesos deben liberar los que tengan (suponiendo que sea posible) (puntuación). *Sin embargo, esto introduce el peligro de que un proceso obtenga el recurso nuevo pero pierda algunos de los que tenía antes porque procesos competidores se quedaron con ellos. Proponga una forma de mejorar el esquema.*

Como tutor, mi propuesta particular es que un proceso (ejemplo A) libere sus recursos para que otro proceso (B) que tiene el recurso que él necesita finalice su tarea y lo libere, el proceso A entraría en una lista de prioridades del tipo FIFO y, cuando le toque su turno, los demás procesos deberán liberar sus recursos para que el finalice su labor, los procesos que liberaron recursos ingresaran en la lista de prioridades para que en su turno otros procesos liberen los recursos, y así sucesivamente.

Ejercicio 28

Leer enunciado en el libro de texto, página 188.

Un algoritmo sería tener dos semáforos, uno a cada lado del río. El último babuino que quiera pasar el río enciende el semáforo, lo pone en rojo (el de su lado) para indicar que él es el último en pasar. Si llega otro no puede pasar hasta que alguien que venga del otro lado lo ponga en verde. En la otra orilla el semáforo está en rojo, de manera que nadie puede pasar hasta que el último babuino que viene lo ponga en verde. La historia se repite hacia el otro lado. Para que no se acapare la pasada solamente de un lado, se puede poner un indicador que no pueden pasar más de 10 babuinos por turno. En este caso, el número 10 será el encargado de poner en rojo el semáforo de su lado y el deponer en verde el semáforo del otro lado cuando llegue.

CAPÍTULO 4

ADMINISTRACIÓN DE MEMORIA

Sumario

- Administración de memoria básica
- Intercambio
- Memoria virtual
- Algoritmos para reemplazo de páginas
- Modelado de algoritmos de reemplazo de páginas
- Aspectos de diseño de los sistemas con paginación
- Aspectos de implementación
- Segmentación

Propósito del capítulo

La **memoria** es un recurso importante que debe administrarse con cuidado. Parafraseando la ley de Parkinson, "...los programas se expanden hasta llenar la memoria con que se cuenta para contenerlos". (Tanenbaum (2003). *Sistemas operativos modernos*, capítulo 4, página 189)

El propósito de este capítulo es conocer la manera en que los sistemas operativos administran la memoria. Estudiaremos varios esquemas de administración de memoria, que van desde los sencillos hasta los más avanzados.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar el concepto de administración de memoria básica.
- Explicar los procesos de intercambio.
- Explicar el concepto de memoria virtual.
- Comprender los algoritmos de reemplazo de páginas.
- Comprender el modelado de algoritmos de reemplazo de páginas.
- Comprender aspectos de diseño de los sistemas con paginación.
- Comprender aspectos de implementación.
- Explicar el concepto de segmentación.

Guía de lecturas

Para lograr los objetivos descritos anteriormente, es importante que realice las siguientes lecturas:

Subtema	Pp.	Subtema	Pp.
4. Administración de memoria	189	4.5 Modelación de algoritmos de reemplazo de páginas	
4.1. Administración de memoria básica	190	4.5.1. Anomalía de Belady	229
4.1.1. Monoprogramación sin intercambio ni paginación	190	4.5.2. Algoritmos depila	229
4.1.2. Multiprogramación con particiones fijas	191	4.5.3. La cadena de distancias	232
4.1.3. Modelado de la multiprogramación	192	4.5.4. Predicción de tasas de fallos de páginas	233
4.1.4. Análisis del desempeño de un sistema multiprogramado	194	4.6. Aspectos de diseño de los sistemas con paginación	234
4.1.5. Reubicación y protección	194	4.6.1. Políticas de asignación local y global	234
4.2. Intercambio	196	4.6.2. Control de carga	236
4.2.1. Administración de memoria con mapas de bits	199	4.6.3. Tamaño de página	237
4.2.2. Administración de memoria con listas enlazadas	200	4.6.4. Espacio de instrucciones y de datos separados	239
4.3. Memoria virtual	202	4.6.5. Páginas compartidas	239
4.3.1. Paginación	202	4.6.6. Política de aseo	241
4.3.2. Tablas de páginas	205	4.6.7. Interfaz de memoria virtual	241
4.3.3. Búferes de consulta para traducción	211	4.7. Aspectos de implementación	242
4.3.4. Tablas de páginas invertidas	213	4.7.1. Intervención del sistema operativo en la paginación	242
4.4 Algoritmos para reemplazo de páginas	214	4.7.2. Manejo de fallos de página	243
4.4.1. El algoritmo óptimo de reemplazo de páginas	215	4.7.3. retroceso de instrucciones	244
4.4.2. El algoritmo de reemplazo de páginas no usadas recientemente	216	4.7.4. Fijación de páginas de memoria	246
4.4.3. El algoritmo de reemplazo de páginas de primero en entrar, primero en salir	217	4.7.5. Almacén de respaldo	246
4.4.4. El algoritmo de reemplazo de páginas de segunda oportunidad	217	4.7.6. Separación de política y mecanismo	247
4.4.5. El algoritmo de reemplazo de páginas tipo reloj	218	4.8 Segmentación	249
4.4.6. El algoritmo de reemplazo de página menos recientemente usada	218	4.8.1. Implementación de la segmentación pura	253
4.4.7. Simulación de LRU en *software*	220	4.8.2. Segmentación con paginación: MULTICS	254
4.4.8. El algoritmo de reemplazo de páginas de conjunto de trabajo	222	4.8.3. Segmentación con paginación: Pentium de Intel	2257
4.4.9. El algoritmo de reemplazo de páginas *WSClock*	225	4.9 Investigaciones sobre administración de memoria	262
4.4.10.Resumen de algoritmos de reemplazo de páginas	227	4.10. Resumen	262

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

Casi todas las computadoras tienen una **jerarquía de memoria**, con una pequeña cantidad de memoria caché muy rápida, costosa y volátil, decenas de megabytes de memoria principal (RAM) de mediana velocidad, de mediano precio y volátil, y decenas o centenas de gigabytes de almacenamiento en disco lento, económico y no volátil. Corresponde al SO coordinar el uso de estas memorias.

La parte del SO que administra la jerarquía de memoria se llama **administrador de memoria**. Su obligación es mantener al tanto al SO sobre qué partes de la memoria están en uso y cuáles no, asignar memoria a los procesos cuando la necesitan y liberarla cuando terminan, y administrar los intercambios entre la memoria principal y el disco cuando la primera es demasiado pequeña para contener todos los procesos.

Los sistemas de administración de memoria pueden dividirse en dos clases: los que traen y llevan procesos entre la memoria principal y el disco durante la ejecución (intercambio y paginación), y los que no lo hacen.

Es importante tener presente que el intercambio y la paginación son, en gran medida, mecanismos artificiales, obligados por la falta de suficiente memoria principal para contener todos los programas a la vez.

La **monoprogramación sin intercambio ni paginación** es el esquema de administración más sencillo posible. Consiste en ejecutar únicamente un programa a la vez, repartiendo la memoria entre ese programa y el SO. En la figura 4-1, página 91 del libro de texto, se muestran tres variaciones de este caso.

Si el sistema está organizado de esta manera, únicamente puede ejecutarse un proceso a la vez. Tan pronto como el usuario teclea un comando, el SO copia el programa solicitado del disco a la memoria y lo ejecuta. Cuando el proceso termina, el SO exhibe un indicador de comandos y aguarda un nuevo comando.

La mayoría de los sistemas modernos permiten la ejecución de varios procesos al mismo tiempo. Con esto se eleva el aprovechamiento de la CPU. A esta figura se le llama **multiprogramación**.

La forma más fácil de lograr la multiprogramación es simplemente dividir la memoria en *n* particiones (tal vez desiguales). Esta división puede efectuarse en forma manual. (Ver figura 4-2, página 192 del libro de texto.)

Cuando se usa multiprogramación, es posible mejorar el aprovechamiento de la CPU. Si un proceso representativo realiza sólo el 20% del tiempo que está en memoria, y se tienen cinco procesos en la memoria a la vez, la CPU deberá estar ocupada todo el tiempo. Sin embargo, este modelo es optimista en exceso, pues supone que los cinco procesos nunca estarán esperando E/S al mismo tiempo. El aprovechamiento de la CPU está dado por la fórmula:

Aprovechamiento de la CPU = 1-pn (p a la n)

(La figura 4-3 de la página 193 del libro de texto muestra el aprovechamiento de la CPU en función de *n*, que se denomina **grado de multiprogramación**.)

(Para ampliar este concepto, refiérase a las secciones 4.1.1 a la 4.1.4. y a la figura 4-4, páginas 190 a la 194 del libro de texto.)

La multiprogramación introduce dos problemas fundamentales que es preciso resolver: **reubicación y protección**. Examinemos la figura 4-2 (página 192 del libro de texto). Es evidente que los distintos trabajos estarán en direcciones distintas. Cuando se enlaza un programa (es decir, cuando se combinan el programa principal, los procedimientos escritos por el usuario y los procedimientos de biblioteca en un solo espacio de direcciones), el enlazador necesita saber en qué dirección de la memoria comenzará el programa.

(Para ampliar más acerca de los conceptos de reubicación y protección, refiérase a la sección 4.1.5, páginas 194 a la 196 del libro de texto.)

Con los sistemas de tiempo compartido, o las computadoras personales orientadas a los gráficos, la situación es distinta. A veces no hay suficiente memoria principal para contener todos los procesos que están activos, así que los procesos excedentes deben mantenerse en disco y traerse a la memoria en forma dinámica para que se ejecuten.

Pueden usarse dos enfoques generales para la administración de memoria, dependiendo (en parte) del *hardware* disponible. La estrategia más sencilla, llamada **intercambio** (*swapping*), consiste en traer a la memoria un proceso entero, ejecutarlo durante un rato y volver a guardarlo en disco. La otra estrategia, llamada **memoria virtual**, permite que los programas se ejecuten aunque solo una parte de ellos estén en la memoria principal.

Cuando el intercambio crea múltiples huecos en la memoria, es posible combinar tales huecos en uno sólo más grande, desplazando todos los procesos hacia abajo hasta donde sea posible. Esta técnica se llama **compactación de memoria**.

Si la memoria se asigna en forma dinámica, el SO debe administrarla. En términos generales, hay dos formas de llevar el control del uso de la memoria: **mapa de bits** y **listas libres**.

Con un mapa de bits, la memoria se divide en unidades de asignación, que pueden ser desde unas cuantas palabras hasta varios kilobytes. A cada unidad de asignación corresponde un bit del mapa de bits. El bit es 0 si la unidad está desocupada y 1 si está ocupada (o viceversa).

Otra forma de llevar el control de la memoria es mantener una lista enlazada de segmentos de memoria asignados y libres, donde un segmento es un proceso, o bien, un hueco entre dos procesos.

Si los procesos y huecos se mantienen en una lista ordenada por dirección, pueden usarse varios algoritmos para asignar memoria a un proceso recién creado. Estos algoritmos son:

- Primer ajuste
- Siguiente ajuste
- Mejor ajuste
- Peor ajuste
- Ajuste rápido

(Para ampliar sobre el tema de administración de memoria con mapas de bits y con listas enlazadas, consulte las secciones 4.2.1 y 4.2.2, páginas 199 a la 202 del libro de texto.)

La idea básica del método conocido como **memoria virtual** es que el tamaño combinado del programa, sus datos y su pila podrían exceder la cantidad de memoria física que se le puede asignar. El sistema mantiene en la memoria principal las partes del programa que se están usando en ese momento, y el resto en el disco.

La mayoría de los sistemas con memoria virtual utilizan una técnica llamada **paginación**.

En cualquier computadora existe un conjunto de direcciones de memoria que los programas pueden producir. Las direcciones pueden generarse empleando indización, registros base, registros de segmentos y otros métodos.

Estas direcciones generadas por el programa se denominan **direcciones virtuales** y constituyen el **espacio de direcciones virtual**.

Cuando se usa memoria virtual, las direcciones virtuales no se envían de manera directa al bus de memoria, sino a una **unidad de administración de memoria (MMU**; *memory management unit***)** que establece una correspondencia entre las

direcciones virtuales y físicas de la memoria (como se ilustra en la figura 4-9, página 203 del libro de texto).

El espacio de direcciones virtuales se divide en unidades llamadas **páginas**. Las unidades correspondientes en la memoria física se denominan **marcos de página**.

El número de página se utiliza como índice para consultar la **tabla de páginas** y así, obtener el número del marco de página que corresponde a la página virtual.

El propósito de la tabla de páginas es establecer una correspondencia entre las páginas virtuales y los marcos de página. A pesar de lo sencillo de esta descripción, hay que resolver dos problemas importantes:

1. La tabla de páginas puede ser extremadamente grande.
2. La transformación (correspondencia) debe ser rápida.

(Para ampliar el tema correspondiente a la memoria virtual y sus subtemas refiérase a las páginas 202 a la 214 del libro de texto.)

Cuando se presenta un fallo de página, el sistema operativo tiene que escoger la página que desalojará de la memoria para hacer espacio para colocar la página que traerá del disco. Si la página a desalojar fue modificada mientras estaba en memoria, deberá rescribirse en el disco para actualizar la copia. En cambio, si la página no se ha modificado, no será necesario rescribirla.

Para el reemplazo de páginas existen varios algoritmos:

- En el **algoritmo óptimo de reemplazo de páginas**, cada página puede rotularse con el número de instrucciones que se ejecutarán antes de que se haga la primera referencia a esa página. El algoritmo de página óptima simplemente dice que debe desalojarse la página con el rótulo más grande.

- En el **algoritmo de reemplazo de páginas no usadas recientemente (NRU)**, el SO lleva una estadística acerca de cuáles páginas se están usando y cuáles no. Casi todas las computadoras con memoria virtual asocian a cada página dos bits de estado. R se enciende cada vez que se hace referencia a la página (leer o escribir). M se enciende cada vez que se escribe en la página (se modifica). El algoritmo desaloja al azar una página de la clase de número más bajo que no esté vacía.

- Otro algoritmo de paginación con bajo gasto adicional es el de **primero en entrar, primero en salir (FIFO; *first-in*, *first-out*)**. El SO mantiene una lista de todas las páginas que están en memoria, con la más antigua al principio de la lista y la más nueva al final. Al presentarse un fallo de página, se desaloja la página que está al principio de la lista, y la nueva se anexa al final.

- El funcionamiento del algoritmo llamado de **segunda oportunidad** se muestra en la figura 4.6 (página 218 del libro de texto).

- En el **algoritmo de reemplazo de páginas tipo reloj** la estrategia es mantener todas las páginas en una lista circular parecida a un reloj, una manecilla apunta a la página más antigua.

- El **algoritmo de reemplazo de página menos recientemente usada (LRU)** lo que hace es desalojar la página que tiene más tiempo sin usarse, al presentarse un fallo de página.

El **algoritmo de reemplazo de páginas de conjunto de trabajo** y el **algoritmo de reemplazo de páginas *WSClock*** deben estudiarse detalladamente. (Refiérase a las secciones 4.4.8 y 4.4.9., páginas de la 222 a la 227 del libro de texto.)

La memoria virtual que hemos tratado hasta ahora es unidimensional, porque las direcciones virtuales van desde cero hasta alguna dirección máxima, y son consecutivas. En el caso de varios problemas, podría ser mucho mejor tener dos o más espacios de direcciones distintos que sólo uno.

Un **segmento** es proporcionar a la máquina varios espacios de direcciones independientes por completo. Dado que cada segmento constituye un espacio de direcciones distinto, los segmentos pueden crecer o decrecer en forma independiente, sin afectarse unos a otros.

Un segmento podría contener un procedimiento, un arreglo, una pila o una colección de variables escalares, pero por lo regular no contiene una mezcla de tipos distintos.

(En la figura 4-37, página 252 del libro de texto, se comparan la paginación y la segmentación.)

La implementación de la segmentación difiere de la paginación en un aspecto fundamental: las páginas son de tamaño fijo y los segmentos no.

(Para ampliar sobre el tema de segmentación refiérase a la sección 4.8 Segmentación, páginas de la 249 a la 262 del libro de texto.)

Ejercicios sugeridos

A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 4, realice

los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 263, 264, 265, 266, 267 y 268: 1, 2, 5, 24, 33, 37.

Resolución de ejercicios sugeridos

Ejercicio 1

Un sistema de cómputo tiene espacio para cuatro programas en su memoria principal. Estos programas están activos la mitad del tiempo esperando E/S. ¿Qué fracción del tiempo de CPU se desperdició?

Aprovechamiento de CPU = 1 – pn (se lee *p* a la *n)*

CPU = 1 - (0.50) 4
CPU = 1 – 0.0625
CPU = 0.9375 El aprovechamiento es del 93.75% El desperdicio es de 6.25%

Ejercicio 2

En la figura 4-21 vimos un ejemplo de cómo puede ejecutarse múltiples trabajos en paralelo y terminar antes de que si se hubieran ejecutado de manera sucesiva. Suponga que hay dos trabajos, cada uno de los cuales necesita 10 minutos de tiempo de CPU, se inician en forma simultánea. ¿Cuánto tardará en terminar el último si se ejecutan de manera sucesiva? ¿Cuánto tardará si se ejecutan en paralelo? Suponga un 50% de espera por E/S.

En forma sucesiva se requerirá de 40 minutos para terminar ambos. En forma simultánea tardarán 30 minutos pues el aprovechamiento del CPU es del 75%. Aplicando la fórmula del problema anterior.

Ejercicio 5

Considere un sistema de intercambio en el que la memoria contiene los siguientes huecos en orden según su posición en la memoria: 10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB y 15 KB. Determine cuál hueco se usará si se reciben solicitudes sucesivas pidiendo:

a) 12 KB
b) 10 KB
c) 9 KB

¿Si se usa primer ajuste? Repita el problema usando mejor ajuste, peor ajuste y siguiente ajuste.

Primer ajuste: El hueco de 20 KB, el hueco de 18 KB y el hueco de 12 KB.
Mejor ajuste: El hueco de 20 KB, el hueco de 10 KB y el hueco de 9 KB.
Peor ajuste: El hueco de 20 KB, el hueco de 18 KB y el hueco de 15 KB.
Siguiente ajuste: El hueco de 20 KB, el hueco de 18 KB y el hueco de 12 KB.

Ejercicio 24

Considere la sucesión de páginas de la figura 4-16b- Suponga que los bits R para las páginas B a A son 11011011, respectivamente. ¿Qué página desalojará el algoritmo de segunda oportunidad?

B C D E F G H A
1 1 0 1 1 0 1 1

El algoritmo desalojará la página D, puesto que es la página que va por su segunda oportunidad, de desalojar otra página seguiría la página G. Debe recordarse que las páginas marcadas en el bit R con un 0 significan que ya han sido tomadas en cuenta y que son candidatas a ser desalojadas.

Ejercicio 33

Si dos procesos comparten una página, ¿es posible que la página sea de solo lectura para un proceso y de lectura-escritura para el otro? ¿Por qué sí y por qué no?

Es posible si el sistema operativo así lo permite, pero no es recomendable desde el punto de vista de confiabilidad de los datos, pues en un momento dado se puede brindar información que está siendo actualizada simultáneamente, lo cual puede provocar errores en la integridad de la información.

Ejercicio 37

Explique la diferencia entre fragmentación interna y fragmentación externa. ¿Cuál se presenta en los sistemas de paginación? ¿ Cuál se presenta en los sistemas que usan segmentación pura?

Fragmentación interna: Es un marco de página (tamaño fijo) que se requiere para asignar una cantidad fija de memoria (página). Si no se utiliza toda la página, se desperdicia el resto de memoria no utilizado.

Fragmentación externa: Se da en memoria dinámica. Cuando esta se ha ido fragmentando, los huecos entre procesos pueden compactarse de manera que no queden espacios entre segmentos sin utilizar.

En síntesis, las páginas son de tamaño fijo y los segmentos no.

CAPÍTULO 5

ENTRADA/SALIDA

Sumario

- Principios de *hardware* de E/S
- Principios de *software* de E/S
- Capas del *software* de E/S
- Discos
- Relojes
- Terminales orientadas a caracteres
- Interfaces gráficas del usuario
- Terminales de red
- Administración de energía

Propósito del capítulo

Una de las principales funciones de un sistema operativo es controlar todos los dispositivos de entrada y salida de la computadora. Debe enviar comandos a los dispositivos, atrapar interrupciones y manejar errores. También, debe proporcionar una interfaz sencilla y fácil de usar entre los dispositivos y el resto del sistema.

El propósito de este capítulo es ocuparnos de la programación de los dispositivos de E/S. Examinaremos algunos principios del *hardware* de E/S y estudiar el *software* de E/S general. Además, se estudiarán en detalle varios dispositivos de E/S, tales como discos, relojes, teclados y pantallas, y el *hardware* y *software* para cada caso.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar los principios del *hardware* de E/S.
- Explicar los principios de *software* de E/S.
- Explicar el concepto de capas del *software* de E/S.
- Explicar los conceptos de Disco, relojes, teclados y pantallas como dispositivos de E/S.

Guía de lecturas

Para lograr los objetivos descritos anteriormente**,** es importante que realice las siguientes lecturas**:**

Subtema	**Pp.**	**Subtema**	**Pp.**
5.1 Principios de *hardware* de E/S	269	5.4.5 Almacenamiento estable	324
5.1.1. Dispositivos de E/S	270	5.5 Relojes	327
5.1.2 Controladoras de dispositivos	271	5.5.1 *Hardware* del reloj	328
5.1.3 E/S con correspondencia en memoria	272	5.5.2 *Software* del reloj	329
5.1.4 Acceso directo a memoria	276	5.5.3 Temporizadores de *software*	332
5.1.5 Repaso de interrupciones	279	5.6 Terminales orientadas a caracteres	333
5.2 Principios de *software* de E/S	282	5.6.1 *Hardware* de terminal RS-232	334
5.2.1. Metas del *software* de E/S	283	5.6.2 *Software* de entrada	336
5.2.2 E/S programada	274	5.6.3 *Software* de salida	341
5.2.3 E/S controlada por interrupciones	286	5.7 Interfaces gráficas del usuario	342
5.2.4 E/S con DMA	287	5.7.1 *Hardware* de teclado, ratón y pantalla para computadora personal	343
5.3 Capas del *software* de E/S	287	5.7.2 *Software* de entrada	347
5.3.1 Manejadores de interrupciones	287	5.7.3 *Software* de salida para *Windows*	347
5.3.2 Controladoras de dispositivos	289	5.8 Terminales de red	355
5.3.3 *Software* de E/S independiente del dispositivo	292	5.8.1 El sistema X *Windows*	356
5.3.4 *Software* de E/S en espacio de usuario	298	5.8.2 La terminal de red SLIM	360
5.4 Discos	300	5.9 Administración de energía	363
5.4.1 *Hardware* de disco	300	5.9.1 Aspectos de *hardware*	364
5.4.2 Formateo de discos	315	5.9.2 Aspectos del sistema operativo	365
5.4.3 Algoritmos para calendarizar el brazo del disco	318	5.9.3 Merma en el funcionamiento	370
5.4.4 Manejo de errores	322		

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

Existen diversos puntos de vista en lo relacionado al *hardware* de E/S. Los ingenieros eléctricos lo ven en términos de *chips*, cables, fuentes de energía, motores y todos los demás componentes físicos que constituyen el *hardware*.

Los programadores ven la interfaz que se presenta al *software*: los comandos que acepta el *hardware*, las funciones que desempeña y los errores que puede informar.

Los dispositivos de E/S pueden dividirse en dos categorías: **dispositivos de bloque** y **dispositivos de caracteres**.

Un dispositivo de bloque almacena información en bloques de tamaño fijo, cada uno con su propia dirección. Su propiedad fundamental consiste en que es posible leer o escribir cada bloque con independencia de todos los demás. Los discos son los dispositivos de bloque más comunes.

Un dispositivo de caracteres suministra o acepta un flujo de caracteres sin estructurarlos en bloques; no es direccionable ni tiene una operación de desplazamiento. Las impresoras, interfaces de red, ratones y casi todos los demás dispositivos que no son tipo disco, pueden considerarse como dispositivos de caracteres.

Por lo regular, las unidades de E/S constan de un componente mecánico y uno electrónico. En muchos casos es posible separar las dos partes para tener un diseño más modular. El componente electrónico se denomina **controladora** o **adaptador de dispositivo**. El **componente mecánico** es el dispositivo mismo.

Cada controladora tiene unos cuantos registros que le sirven para comunicarse con la CPU. ¿Cómo se comunica la CPU con los registros de control y los búferes de datos de los dispositivos? Existen dos alternativas. Con el primer enfoque, a cada registro de control se le asigna un número de **puerto de E/S**. El segundo enfoque consiste en establecer una correspondencia entre todos los registros de control y el espacio de memoria. A cada registro de control se le asigna una dirección de memoria única a la cual no se asigna memoria. Este sistema se denomina **E/S con correspondencia en memoria**.

Tenga o no E/S con correspondencia en memoria, la CPU necesita direccionar las controladoras de dispositivos para intercambiar datos con ellas. La CPU puede solicitar datos a una controladoras de E/S, *byte* por *byte*, pero ello obliga a la CPU a perder tiempo. Por esta razón, es común utilizar un esquema distinto, llamado **acceso directo a memoria** (**DMA**, *direct memory access*).

Cuando un dispositivo de E/S termina el trabajo que se le encomendó, causa una interrupción. Esto se hace aplicando una señal a una línea de bus que se le haya asignado.

Una interrupción que deja a la máquina en un estado bien definido se denomina **interrupción precisa**. Una interrupción de este tipo tiene cuatro propiedades:

1. El contador de programa (CP) se guarda en un lugar conocido.
2. Todas las instrucciones previas a aquella, a la que apunta el CP, ya se ejecutaron por completo.
3. No se ha ejecutado ninguna instrucción posterior a aquella a la que apunta el CP.
4. Se conoce el estado de ejecución de la instrucción a la que apunta el CP.

Una interrupción que no cumple con estos requisitos es una **interrupción imprecisa**.

(Para ahondar más en estos temas refiérase a la sección 5.1 del libro de texto. Páginas de la 269 a la 282.)

Un concepto clave en el diseño del *software* para E/S se conoce como **independencia del dispositivo**. Esto significa que debe ser posible escribir programas capaces de acceder a cualquier dispositivo de E/S sin tener que especificar por adelantado de qué dispositivo se trata.

Algo estrechamente relacionado con la independencia del dispositivo es la meta de **nombres uniformes**. El nombre de un archivo o dispositivo deberá ser simplemente una cadena o un entero y no depender en absoluto del dispositivo.

Otro aspecto importante del *software* de E/S, es el **manejo de errores**. En general, los errores deben manejarse tan cerca del *hardware* como sea posible. Si la controladora descubre un error de lectura, deberá tratar de corregirlo ella misma si puede. Si no puede, el controlador de dispositivo deberá corregirlo, tal vez repitiendo el intento de leer el bloque.

Otro aspecto clave es la diferencia entre las transferencias **síncronas** (por bloqueo) y **asíncronas** (controladas por interrupciones). Casi toda la E/S física es síncrona: la CPU inicia la transferencia y se pone a hacer alguna otra cosa hasta que llega la interrupción.

Otra cosa que el *software* de E/S debe manejar es el **uso de búferes**. Es común que los datos provenientes de un dispositivo no puedan almacenarse en forma directa en su destino final.

Existen tres formas fundamentalmente distintas de efectuar la E/S. La forma más sencilla deja que la CPU haga todo el trabajo. Este método se denomina **E/S programada**.

La segunda forma es la **E/S controlada por interrupciones**. Cuando la impresora ha impreso un carácter y está preparada para aceptar el siguiente, genera una interrupción, lo que detiene el proceso actual y guarda su estado. Luego se ejecuta el servicio de interrupciones. Una desventaja obvia es que se presenta una interrupción con cada carácter.

La **E/S con DMA**. Aquí la idea consiste en dejar que la controladora DMA alimente los caracteres a la impresora uno por uno, sin molestar a la CPU. Básicamente, el DMA es E/S programada, solo que la controladora DMA es la que realiza todo el trabajo, no la CPU.

(Para ahondar más en estos temas refiérase a la sección 5.2 del libro de texto. Páginas de la 282 a la 287.)

Aunque la E/S programada a veces es útil, en la mayoría de las operaciones de E/S las interrupciones son inevitables, por molestas que sean. Lo mejor es ocultarlas en las profundidades del SO, de modo que la parte de este último que tiene conocimiento de ellas sea lo más reducido posible. La mejor manera de ocultar las interrupciones es hacer que el controlador que inicia una operación de EPS se bloquee hasta que la E/S haya terminado y se presente la interrupción.

Cada dispositivo de E/S conectado a una computadora necesita código específico que sirva para controlar ese dispositivo. Este código, llamado **controlador de dispositivo** (*device driver*), por lo general es escrito por el fabricante del dispositivo y se proporciona junto con el *hardware*.

Para acceder al *hardware* del dispositivo, es decir, a los registros de la controladora, por lo general es necesario que el controlador forme parte del *kernel* del SO, al menos en las arquitecturas actuales.

Los sistemas operativos por lo regular clasifican los controladores en unas cuantas categorías. Las más comunes son los **dispositivos de bloques** como discos, que contienen varios bloques de datos susceptibles de direccionarse en forma independiente, y los **dispositivos de caracteres**, como los teclados, impresoras, que generan o aceptan un flujo de caracteres.

La función básica del *software* independiente del dispositivo es realizar las operaciones de E/S que son comunes a todos los dispositivos y presentar una interfaz uniforme al *software* de usuario.

Si la interfaz con los discos, impresoras, teclados, etc. es muy distinta para cada caso en particular, entonces cada vez que llegue un nuevo dispositivo será preciso modificar el sistema operativo para ese dispositivo. Esto no es muy recomendable.

(Para una mayor comprensión del tema, vea la figura 5-13 del libro de texto, página 294.)

Aunque casi todo el *software* de E/S está dentro del sistema operativo, una porción pequeña consiste en bibliotecas enlazadas con programas de usuario, e incluso en programas enteros que se ejecutan fuera del *kernel*. Los procedimientos de biblioteca generalmente emiten llamadas al sistema, entre las que se encuentran las de E/S.

No todo el *software* de E/S en el nivel de usuario consiste en procedimientos de biblioteca. Otra categoría importante es el sistema de *spooling*. El ***spooling*** es una forma de manejar dispositivos dedicados en un sistema con multiprogramación.

(Para ahondar más en estos temas refiérase a la sección 5.3 el libro de texto. Páginas de la 287a la 300).

Ahora comenzaremos a estudiar algunos dispositivos reales de E/S. Los primeros son los discos.

Los discos son de diversos tipos. Los más comunes son los discos magnéticos (discos duros y disquetes). Estos se caracterizan por el hecho de que las lecturas y escrituras son igual de rápidas, lo que los hace ideales como memoria secundaria (paginación, sistemas de archivos, etc.).

Los discos magnéticos se organizan en cilindros, cada uno de los cuales tiene tantas pistas como haya cabezas apiladas en forma vertical. Las pistas se dividen en sectores.

Una característica del dispositivo que tiene implicaciones importantes para el controlador de disco es la posibilidad de que una controladora desplace las cabezas lectoras de dos o más unidades de disco, al mismo tiempo. Esto se denomina **desplazamiento de cabeza traslapado**. Mientras la controladora y el *software* están esperando que el desplazamiento de la cabeza termine en una unidad, la controladora puede iniciar un desplazamiento de cabeza en otra unidad. Muchas controladoras también, pueden leer o escribir en una unidad mientras desplazan la cabeza en otra u otras unidades, pero una controladora de disquete no puede leer o escribir en dos unidades al mismo tiempo.

Un disco duro consiste en una pila de platos de aluminio. Para que pueda usarse el disco, cada plato debe recibir un **formato de bajo nivel** efectuado por el *software*. El formato consiste en una serie de pistas concéntricas, cada una de las cuales contiene cierto número de sectores, con espacios cortos entre ellos.

El tiempo que se toma para leer o escribir en un disco está determinado por tres factores:

1. Tiempo de desplazamiento (*seek time*), o sea, el tiempo que tarda el movimiento del brazo hasta el cilindro correcto.
2. Retraso rotacional, ó sea, el tiempo que tarda el sector correcto en girar hasta colocarse bajo la cabeza.
3. Tiempo real de transferencia de datos.

Los defectos de fabricación introducen sectores defectuosos. Existen dos enfoques generales para manejar los bloques defectuosos: ocuparse de ellos en la controladora o hacerlo en el sistema operativo. Con el primer enfoque, antes de

que el disco salga de la fábrica se le prueba y se escribe en el disco una lista de sectores defectuosos.

En el segundo enfoque, si el sistema operativo se está encargando del ajuste de correspondencia, deberá asegurarse de que no haya sectores defectuosos en los archivos y tampoco en la lista o mapa de bits de sectores libres. Una forma de hacerlo es crear un archivo secreto integrado por todos los sectores defectuosos. Si este archivo no se incorpora al sistema de archivos, los usuarios no podrán leerlo accidentalmente o, lo que es peor, liberar su espacio.

En algunas aplicaciones es indispensable que los datos nunca se pierdan ni se corrompan, aunque se presenten errores de disco o de CPU. De manera ideal, un disco deberá trabajar todo el tiempo sin errores. Lo malo es que dicha meta es inasequible. Lo que sí es factible es tener un subsistema de disco con la siguiente propiedad: cuando se le ordena escribir algo, se escriben correctamente los datos o no se escribe nada, dejando los datos existentes intactos. Un sistema así se denomina **almacenamiento estable** y se implementa en *software*.

El almacenamiento estable utiliza un par de discos idénticos en el que los bloques correspondientes colaboran para formar un bloque sin errores. En ausencia de errores, los bloques correspondientes en ambas unidades son iguales. Es posible leer cualquiera de ellos y obtener el mismo resultado. Para lograr esta meta, se definen tres operaciones: **escrituras estables, lecturas estables y recuperación después de caídas**.

(Para ahondar más en estos temas refiérase a la sección 5.4 del libro de texto. Páginas de la 300 a la 326.)

Los **relojes** (también llamados **temporizadores**) son indispensables para el funcionamiento de cualquier sistema multiprogramado por diversas razones.

El *software* de reloj puede adoptar la forma de un controlador de dispositivo, aunque un reloj no es un dispositivo de bloques, ni un dispositivo de caracteres.

Por lo general, en las computadoras se usan dos tipos de reloj, ambos muy diferentes a los que usan las personas. Los más sencillos están conectados a la línea de alimentación eléctrica y causan una interrupción en cada ciclo de voltaje. Ahora éstos son poco comunes.

El otro tipo de reloj se construye con tres componentes: un cristal oscilador, un contador y un registro de retención. Este genera una señal periódica de gran precisión.

Lo único que hace el *hardware* de reloj es generar interrupciones a intervalos conocidos. Todo lo demás relacionado con el tiempo debe efectuarse en el *software*, con el controlador de reloj. Sus tareas son:

1. Mantener la hora del día.
2. Evitar que se ejecuten los procesos durante más tiempo del debido.
3. Contabilizar el consumo de CPU.
4. Procesar la llamada al sistema *alarm* emitida por procesos de usuario.
5. Proporcionar temporizadores de vigilancia a ciertas partes del sistema.
6. Realizar perfiles, supervisión y recolección de datos estadísticos.

Casi todas las computadoras tienen un segundo reloj programable que puede ajustarse de modo que cause interrupciones con la frecuencia que las necesite un programa. Este temporizador es adicional al temporizador principal del sistema.

Los **temporizadores de *software*** evitan interrupciones. Cada vez que el *kernel* se ejecuta por algún motivo, justo antes de volver al modo de usuario examina el reloj de tiempo real para ver si ha expirado un temporizador de *software*.

(Para ahondar más en estos temas refiérase a la sección 5.5 del libro de texto. Páginas de la 327 a la 333.)

Toda computadora de uso general tiene al menos un teclado y una pantalla que sirve para comunicarse con ella.

La tarea básica del controlador de teclado consiste en obtener entradas del teclado y pasarlas a los programas de usuario cuando estos lean de la terminal.

Las salidas son más sencillas que las entradas. En general, la computadora envía caracteres a la terminal, donde se exhiben.

(Para ahondar más en estos temas refiérase a la sección 5.6 del libro de texto. Páginas de la 333 a la 342.)

Ejercicios sugeridos

A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 5, realice los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 373, 374, 375, 376, 377 y 378: 1, 2, 3, 9, 14, 21, 23.

Resolución de ejercicios sugeridos

Ejercicio 1

Los adelantos en la tecnología de chips han hecho posible colocar una controladora entera, incluida toda la lógica de acceso al bus, en un chip de bajo costo. ¿Cómo afecta eso el modelo de la figura 1-5?

Lo afecta de manera positiva. Lo anterior debido a que el chip, al tener incorporada la lógica de acceso al bus de datos, evita que la controladora tenga que recurrir al *kernel* liberando al SO de operaciones tal vez rutinarias y permitiéndole que se dedique a tareas más importantes. Entre menos interrupciones haya, el SO será más eficiente. (Página 274 del libro de texto.)

Ejercicio 3

La figura 5-3b muestra una forma de tener E/S con correspondencia en memoria aún se usan buses separados para la memoria y los dispositivos de E/S, probando primero el bus de memoria y, si este falla, probando el de E/S. Un ingenioso estudiante de ciencias de la computación ha ideado una mejora: probar ambos buses en paralelo, a fin de acelerar el proceso de acceso a dispositivos de E/S. ¿Qué le parece la idea?

Esto sería muy bueno si realmente el bus de memoria tuviera fallas frecuentes. Si así fuera, no sería un SO confiable. Como habría un alto porcentaje de fallas la idea no es tan descabellada, más bien sería una manera de hacer más eficiente el tiempo de respuesta. Como las fallas son mínimas, no vale la pena tener dos buses pues de lo que se trata es de separar tareas, no de duplicar.

Ejercicio 9

¿Qué es independencia de dispositivo?

Se refiere al *software* que depende del sistema y no del propio dispositivo. Este *software* tiene como función realizar operaciones básicas de E/S que son comunes a todos los dispositivos, y presentar una interfaz uniforme al *software* del usuario. (Páginas 292 y 293 del libro de texto.)

Ejercicio 13

¿Por qué los archivos de salida para la impresora se colocan generalmente en spooling *en disco antes de imprimirse?*

Aunque desde el punto de visa técnico sería más fácil permitir que comúnmente un proceso de usuario abra un archivo especial por caracteres correspondiente a la impresora, vamos a suponer que un proceso lo abre y luego no hace nada durante horas. Ningún otro proceso podría imprimir nada. En vez de eso, lo que se hace es crear un proceso especial, llamado **demonio**, y un directorio especial, llamado **directorio de *spool***. Para imprimir un archivo, lo primero que hace un proceso es generar el archivo completo que se imprimirá y colocará en el directorio de *spool*. Corresponde al demonio administrar los archivos y es el único proceso autorizado para usar el archivo. Proteger el archivo especial contra el uso directo de los usuarios elimina el problema de que uno de ellos lo mantenga

abierto durante un tiempo innecesariamente largo. (Páginas 298 y 299 del Libro de texto.)

Ejercicio 19

Si una controladora escribe en la memoria los bytes *que recibe del disco a la misma velocidad a la que los recibe, sin colocarlos en un búfer interno, ¿podría tener alguna utilidad la intercalación? Explique.*

No sería necesaria la intercalación, pues la misma memoria haría las funciones de *búfer*. Lo único que habría que cuidar es que el número de *bytes* a copiar no sea mayor que la memoria disponible.

CAPÍTULO 6

SISTEMAS DE ARCHIVOS

Sumario

- Archivos
- Directorios
- Implementación de sistemas de archivos
- Ejemplos de sistemas de archivos

Propósito del capítulo

Todas las aplicaciones de computadora necesitan almacenar y recuperar información. Mientras está en ejecución, un proceso puede almacenar una cantidad limitada de información dentro de su propio espacio de direcciones. Sin embargo, la capacidad de almacenamiento está restringida al tamaño del espacio de direcciones virtual.

Conservar información dentro del espacio de direcciones de un proceso implica otro problema: que cuando el proceso termine se perderá la información. En muchas aplicaciones (por ejemplo: las de bases de datos) la información debe conservarse durante semanas, meses o, incluso, de manera indefinida. No es aceptable que la información desaparezca cuando termina el proceso que la usa.

Así pues, tenemos tres requisitos indispensables para el almacenamiento de información a largo plazo:

1. Debe tener la posibilidad de almacenar una cantidad muy grande de información.
2. La información debe sobrevivir a la terminación del proceso que la usa.
3. Debe existir la capacidad de que múltiples procesos accedan a la información de manera concurrente.

La solución usual a todos estos problemas, es almacenar la información en discos y otros medios externos en unidades llamadas **archivos**.

El sistema operativo administra los archivos. La forma en que se estructuran, se nombran, se tiene acceso a ellos, se usan, se protegen y se implantan son temas

importantes del diseño de sistemas operativos. En general, la parte del sistema operativo que se ocupa de los archivos se denomina **sistema de archivos** y es el tema de este capítulo.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar lo referente a los archivos de E/S.
- Explicar lo referente a directorios.
- Explicar lo concerniente a la implementación de sistemas de archivos.
- Conocer ejemplos de sistemas de archivos.

Guía de lecturas

Para lograr los objetivos descritos anteriormente**,** es importante que realice las siguientes lecturas**:**

Subtema	Pp.	Subtema	Pp.
6.1 Archivos	380	6.3 Implementación del sistema de archivos	399
6.1.1 Nombres de archivo	380	6.3.1 Organización de sistemas de archivos	399
6.1.2 Estructura de archivos	382	6.3.2 Implementación de archivos	400
6.1.3 Tipos de archivos	272	6.3.3 Implementación de directorios	405
6.1.4 Acceso a archivos	385	6.3.4 Archivos compartidos	408
6.1.5 Atributos de archivos	386	6.3.5 Administración de espacio en disco	410
6.1.6 Operaciones con archivos	387	6.3.6 Confiabilidad del sistema de archivos	416
6.1.7 Ejemplo de programa que usa llamadas al sistema de archivos	389	6.3.7 Desempeño del sistema de archivos	424
6.1.8 Archivos con correspondencia en memoria	391	6.3.8 Sistema de archivos con estructura de registro	428
6.2 Directorios	393	6.4 Ejemplos de sistemas de archivos	430
6.2.1 Sistemas de directorios a un solo nivel	393	6.4.1 Sistemas de archivos de CR-ROM	430
6.2.2 Sistemas de directorios de dos niveles	394	6.4.2 El sistema de archivos CP/M	435
6.2.3 Sistema de directorios jerárquicos	395	6.4.3 El sistema de archivos MS-DOS	438
6.2.4 Nombres de ruta	395	6.4.4 El sistema de archivos de Windows 98	442
6.2.5 Operaciones con directorios	398	6.4.5 El sistema de archivos de UNIX V7	356

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

Cuando un proceso crea un archivo, le asigna un nombre. Cuando el proceso termina, el archivo sigue existiendo y otros programas pueden tener acceso a él utilizando su nombre.

Las reglas exactas para nombrar archivos varían un tanto de un sistema a otro, pero todos los sistemas operativos actuales permiten usar cadenas de una a ocho letras como nombres de archivo válidos.

Muchos sistemas de archivos manejan nombres de archivo de dos partes, separadas con un punto. La parte que sigue al punto se denomina **extensión de archivo** y, por lo regular, indica algo acerca del archivo. Para mayor comprensión ver figura 6.1., en la página 381 del libro de texto.

Los archivos pueden estructurarse de varias maneras. En la figura 6-2 de la página 382, del libro de texto se ilustran tres posibilidades comunes.

Hacer que el sistema operativo vea los archivos únicamente como sucesiones de *bytes* ofrece el máximo de flexibilidad. Los programas de usuario pueden colocar lo que deseen en sus archivos y darles el nombre que les convenga.

Los **archivos normales** son los que contienen información de usuario. Todos los archivos de la figura 6.2. son archivos normales. Los **directorios** son archivos de sistema que sirven para mantener la estructura del sistema de archivos.

Los **archivos especiales de caracteres** tienen que ver con entrada/salida y sirven para modelar dispositivos de E/S en serie, tales como: terminales, impresoras, redes. Los **archivos especiales de bloques** sirven para modelar discos.

Los primeros sistemas de archivos solo permitían un tipo de acceso a los archivos: **acceso secuencial**. En esos sistemas, un proceso podía leer todos los *bytes* o registros de un archivo en orden, comenzando por el principio, pero no podía efectuar saltos y leerlos en otro orden.

Los archivos cuyos *bytes* o registros pueden leerse en cualquier orden se denominan **archivos de acceso aleatorio** y, muchas aplicaciones los necesitan.

Todo archivo tiene un nombre y datos. Además, todos los sistemas operativos asocian otra información a cada archivo, como la fecha y la hora en que se creó y su tamaño. Llamaremos a esta información adicional **atributos** de archivo.

(Para mayor comprensión vea figura 6-4 en la página 387 del libro de texto.)

Los archivos existen para guardar información y poder recuperarla después. Los distintos sistemas ofrecen diferentes operaciones de almacenamiento y recuperación. (Estudie las llamadas al sistemas contempladas en las páginas 387 y 388 del libro de texto.)

Desde el punto de vista conceptual, podemos imaginar la existencia de dos nuevas llamadas al sistema, *map* y *unmap*. La primera proporciona un nombre de archivo y una dirección virtual, y hace que el sistema operativo establezca una correspondencia del archivo con el espacio de direcciones a partir de la dirección virtual. El proceso utiliza la llamada al sistema *unmap* para eliminar los archivos del espacio de direcciones y luego terminar.

(Para ahondar más en los temas vistos, refiérase a la sección 6.1 del libro de texto, páginas de la 380 a la 393.)

Para llevar el control de los de los archivos, los sistemas de archivos suelen tener **directorios** o **carpetas**.

La forma más sencilla de sistema de directorios es que un directorio contenga todos los archivos. A veces se le llama **directorio raíz**, pero dado que es el único, el nombre no importa mucho.

El problema de tener un solo directorio en un sistema con múltiples usuarios es que diferentes usuarios podrían utilizar por accidente los mismos nombres para sus archivos.

Para evitar conflictos cuando dos usuarios escogen el mismo nombre para sus propios archivos, el siguiente escalón sería dar a cada usuario un directorio privado. Así, los nombres escogidos por un usuario no chocarán con los escogidos por otro, y no habrá problemas si el mismo nombre aparece en dos o más directorios.

La jerarquía de dos niveles elimina los conflictos de nombres entre usuarios pero no es satisfactoria para usuarios que tienen un gran número de archivos. Lo que se necesita es una jerarquía general (es decir, un árbol de directorios). Con este enfoque, cada usuario puede tener tantos directorios como necesite para agrupar sus archivos en categorías naturales.

La posibilidad de crear un número arbitrario de subdirectorios ofrece a los usuarios una herramienta potente para organizar su trabajo.

Cuando un sistema de archivos está organizado como árbol de directorios, se necesita un mecanismo para especificar los nombres de archivo. Por lo común se utilizan dos métodos. En el primero, cada archivo recibe un **nombre de ruta absoluta**, que consiste en el camino que debe seguir para llegar del directorio raíz hasta el archivo.

El otro tipo de nombre es el **nombre de ruta relativa**. Este se usa junto con el concepto de **directorio de trabajo** (o **directorio actual**). Un usuario puede designar un directorio como su directorio de trabajo actual, en cuyo caso todos los nombres de ruta que no comiencen en el directorio raíz se considerarán relativos al directorio de trabajo.

(Estudie las operaciones con directorios descritas en la sección 6.2.5 del libro de texto, páginas 398 a 399.)

(Para ahondar más en los temas vistos refiérase a la sección 6.2 del libro de texto, páginas de la 393 a la 398.)

Los sistemas de archivos se almacenan en discos. Casi todos los discos pueden dividirse en una o más particiones, con sistemas de archivos independientes en cada partición. El sector 0 del disco se llama **registro maestro de arranque** (**MBR**; *master boot record*), y sirve para arrancar la computadora. El final del MBR contiene la tabla de particiones. Esta tabla contiene las direcciones inicial y final de cada partición.

Tal vez el aspecto más importante de la implementación del almacenamiento de archivos sea llevar el control de cuáles bloques de discos corresponden a cuál archivo.

El esquema de asignación más simple es almacenar cada archivo en una serie contigua de bloques de disco.

La asignación de espacio contiguo en disco tiene dos ventajas importantes. La primera es que su implementación es sencilla porque, para llevar el control de dónde están los bloques de un archivo, basta con recordar dos números: la dirección en disco del primer bloque y el número de bloques del archivo.

La segunda es que el desempeño de lectura es excelente porque puede leerse todo el archivo del disco en una sola operación. Solo se necesita un desplazamiento del brazo.

El segundo método para almacenar archivos consiste en mantener cada uno como una lista enlazada de bloques de disco (como se muestra en la figura 6-13 del libro de texto, página 403). La primera palabra de cada bloque se usa como apuntador a la siguiente. El resto del bloque es para datos.

Una modificación a la lista enlazada de bloques se da sacando el apuntador de cada bloque de disco y colocándolo en una tabla en la memoria. Una tabla así en la memoria principal se denomina **tabla de asignación de archivos** (**FAT**; *file allocation table*).

Con esta organización puede llenarse de datos el bloque completo. Además, el acceso aleatorio es mucho más fácil. Aunque todavía es necesario seguir la cadena para hallar un desplazamiento dado dentro del archivo, la cadena está por completo en la memoria, así que puede seguirse sin tener que leer del disco.

Nuestro último método para llevar el control de cuáles bloques pertenecen a cuáles archivos consiste en asociar a cada archivo una estructura de datos llamada **nodo-i** (**nodo índice**), que contiene los atributos y direcciones en disco de los bloques del archivo.

La función principal del sistema de directorios es establecer una correspondencia entre el nombre del archivo ASCII y la información necesaria para localizar los datos.

Un aspecto estrechamente relacionado es dónde deben guardarse los atributos. Todo sistema de archivos mantiene atributos de los archivos, como su dueño y tiempo de creación, y deben almacenarse en algún lado. Una posibilidad obvia es guardarlos directamente en la entrada de directorio. Muchos sistemas hacen precisamente eso.

Cuando varios usuarios colaboran en un proyecto, es común que necesiten compartir archivos. Por ello, en muchos casos conviene que un archivo compartido aparezca al mismo tiempo en diferentes directorios que pertenecen a usuarios distintos.

Casi todos los sistemas de archivos dividen los archivos en bloques de tamaño fijo que no tienen que ser adyacentes.

Una vez que se ha decidido almacenar archivos en bloques de tamaño fijo, surge la pregunta de qué tamaño debe tener un bloque. Dada la forma en que están organizados los discos, el sector, la pista y el cilindro son candidatos obvios para ser la unidad de asignación. En un sistema con paginación, el tamaño de la página también es un contendiente importante.

Una vez escogido un tamaño de bloque, la siguiente cuestión es cómo llevar el control de los bloques libres. Se usan dos métodos: el primero consiste en usar una **lista enlazada de bloques de disco**, en cada uno de los cuales se guardan tantos números de bloques de disco como quepan. La otra técnica de administración del espacio libre es el **mapa de bits**. Un disco con n bloques requiere un mapa de n bits. Los bloques libres se representan con unos en el mapa, y los bloques asignados con ceros (o viceversa).

Para evitar que las personas acaparen demasiado espacio en disco, los sistemas operativos multiusuario a menudo tienen un mecanismo para imponer cuotas de disco. La idea consiste en que el administrador del sistema asigne a cada usuario una porción máxima de archivos y bloques, y que el sistema operativo cuide que los usuarios no excedan la cuota.

Aunque el sistema de archivos no puede ofrecer protección contra la destrucción física del equipo y los medios, sí puede ayudar a proteger la información. Analicemos algunos de los aspectos de protección del sistema de archivos.

Los respaldos en cinta generalmente tienen por objeto resolver uno de dos problemas potenciales:

1. Recuperarse de desastres (falla del disco, incendio u otra catástrofe natural).
2. Recuperarse de estupidez (borrado accidental de archivos).

Hacer un respaldo toma mucho tiempo y ocupa una gran cantidad de espacio, por lo que es importante hacerlo en forma eficiente y conveniente. Para esto, se siguen las siguientes técnicas:

Vaciados incrementales: La forma más sencilla es efectuar un vaciado (respaldo) completo en forma periódica, semanal o mensual, por ejemplo) y hacer un vaciado diario solo de los archivos que han cambiado desde el último respaldo completo.

Vaciado físico: Inicia en el bloque 0 del disco. Escribe en orden todos los bloques del disco en la cinta de salida y se detiene cuando ha copiado el último.

Vaciado lógico: Inicia en uno o más directorios específicos y respalda en forma recurrente todos los archivos y directorios que se encuentran ahí y que hallan sido modificados desde alguna fecha base dada.

Desempeño del sistema de archivos

El acceso a un disco es mucho más lento que el acceso a memoria. En vista de la diferencia en el tiempo de acceso, muchos sistemas de archivos se han diseñado con diversas optimizaciones para mejorar el desempeño. Veamos tres de ellas.

- La técnica más común empleada para reducir los accesos a disco es el **caché de bloques** o **caché de búfer**. (La palabra caché proviene del verbo francés cacher, que significa esconder.) En este contexto, un caché es una colección de bloques que lógicamente debían estar en el disco pero que se están manteniendo en la memoria por razones de desempeño.

- Una segunda técnica para mejorar el desempeño aparente del sistema de archivos es y tratar de colocar bloques en el caché antes de que se necesiten, a fin de mejorar la tasa de aciertos. En particular, muchos archivos se leen en forma secuencial.

- Otra técnica importante consiste en reducir los movimientos del brazo del disco colocando juntos, de preferencia en el mismo cilindro, los bloques a los que tal vez se tendrá acceso en forma secuencial.

Las CPU cada día son más rápidas. Los discos cada día son más grandes y de más bajo costo y el tamaño de la memoria está creciendo en forma exponencial. El único parámetro que no está mejorando a ritmo vertiginoso es el tiempo de desplazamiento del brazo del disco, esto provoca un cuello de botella.

Para aliviar este problema se ha diseñado un tipo de sistema de archivos totalmente nuevo, el **sistema de archivos con estructura de registro** (**LFS**; log-structured file system). La idea que impulsó el diseño del LFS es que conforme se vuelven más rápidas las CPU y las memorias RAM aumentan de tamaño, los cachés de disco también están creciendo con rapidez. Por ello, ahora es posible satisfacer una fracción considerable de todas las solicitudes de lectura directamente del caché del sistema de archivos, sin necesidad de acceso a disco.

(Para ahondar más en los temas vistos refiérase a la sección 6.2 del libro de texto, páginas de la 399 a la 430.)

Ejercicios sugeridos

A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 6, realice los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 449, 450, 451y 452: 4, 5,6, 7, 16.

Resolución de ejercicios sugeridos

Ejercicio 4

En la figura 6-4, uno de los atributos es la longitud de registro. ¿Por qué podría interesarle este dato al sistema operativo?

Recordemos que el sistema operativo también administra la memoria, o sea, es el encargado de asignar segmentos de memoria a los procesos. Si un atributo es la longitud del registro, el SO no tendrá primero que leer el registro para saber su longitud y luego buscar un segmento donde quepan varios registros sino que, con sólo saber este atributo puede realizar una operación, según el número de registros que desee subir a memoria, y buscar el espacio en memoria adecuado. Esto le ahorraría tiempo al SO.

Ejercicio 5

¿Es absolutamente indispensable la llamada al sistema OPEN *en* UNIX*? ¿Qué consecuencias tendría su ausencia?*

No es indispensable la llamada al sistema *OPEN*, pues el sistema operativo, al ver que se utilizará un archivo en ese momento, puede realizar las tareas que hace *OPEN*. Sin embargo, las consecuencias que se pueden dar son: si un programa en ejecución quiere abrir un archivo de datos y no hay espacio en la memoria, el programa puede quedarse por mucho rato esperando espacio. Otra es que el archivo esté siendo utilizado por otra aplicación y no pueda ejecutarse hasta tanto la aplicación que lo tiene ocupado lo libere. Con la llamada *OPEN,* el programa se garantiza que tendrá el archivo a su disposición. (Página 387 del libro de texto.)

Ejercicio 6

Los sistemas que manejan archivos secuenciales siempre tienen una operación para "rebobinar" archivos. ¿Los sistemas que manejan archivos de acceso aleatorio también la necesitan?

No. La instrucción "rebobinar" se aplica especialmente a archivos que se almacenan en cinta. En cambio los archivos aleatorios se almacenan en disco generalmente.

Ejercicio 7

Algunos sistemas operativos proporcionan una llamada al sistema rename *para asignar un nuevo nombre a un archivo. ¿Hay alguna diferencia entre usar esta llamada al sistema para cambiar el nombre de un archivo y simplemente copiar el archivo en un archivo nuevo con el nuevo nombre y después borrar el archivo viejo?*

Con ambos procedimientos se consigue el fin perseguido: cambiar el nombre del archivo. Sin embargo, habrá que tomar en cuenta que en un momento dado tendremos dos archivos similares, del mismo tamaño y con la misma información. Esto ocupará espacio en disco que puede necesitarse en un momento dado. (Página 388 del libro de texto.)

Ejercicio 16

¿Cómo implementa MS-DOS el acceso aleatorio a archivos?

*Utiliza lo que se denomina la FAT (*File Allocation Table*). El MS-DOS lleva el control de los bloques de archivos con una tabla de asignación de archivos (FAT) en la memoria principal. La entrada de directorio contiene el número del primer bloque del archivo. Este número se utiliza como índice para consultar la FAT. (Páginas 403 y 440 del libro de texto.)*

CAPÍTULO 9

SEGURIDAD

Sumario

- El entorno de la seguridad
- Aspectos básicos de criptografía
- Autenticación de usuarios
- Ataques desde adentro del sistema
- Ataques desde afuera del sistema
- Mecanismos de protección
- Sistemas de confianza

Propósito del capítulo

Muchas compañías poseen información valiosa que resguardan celosamente. Dicha información puede ser técnica, comercial, financiera, legal, entre muchas otras posibilidades.

A medida que aumenta la fracción de esta información que se almacena en sistemas de computación, crece en importancia la necesidad de protegerla. Por ello, la protección de esta información contra un uso no autorizado es una tarea fundamental para todos los sistemas operativos.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar lo referente al entorno de la seguridad.
- Explicar los aspectos básicos de criptografía.
- Explicar lo concerniente a la autenticación de usuarios.
- Explicar lo referente a los ataques desde dentro del sistema.
- Explicar lo referente a los ataques desde afuera del sistema.
- Explicar lo referente a mecanismos de protección.

Guía de lecturas

Para lograr los objetivos descritos anteriormente**,** es importante que realice las siguientes lecturas**:**

Subtema	Pp.	Subtema	Pp.
9.1 Seguridad	584	9.4.8 Principios de diseño que proporcionan seguridad	616
9.1.1 Amenazas	584	9.5 Ataques desde afuera del sistema	617
9.1.2 Intrusos	585	9.5.1 Posibles daños causados por virus	618
9.1.3 Pérdida accidental de datos	586	9.5.2 Cómo funcionan los virus	619
9.2 Aspectos básicos de criptografía	587	9.5.3 Cómo se diseminan los virus	626
9.2.1 Criptografía de clave secreta	588	9.5.4 técnicas antivirus y anti-antivirus	628
9.2.3 Funciones unidireccionales	589	9.5.5 El gusano de Internet	635
9.2.4 Firmas digitales	590	9.5.6 Código móvil	637
9.3 Autenticación de usuarios	591	9.5.7 Seguridad en Java	642
9.3.1 Autenticación de contraseña	592	9.6 Mecanismos de protección	645
9.3.2 Autenticación empleando un objeto físico	601	9.6.1 Dominios de protección	645
9.3.3 Autenticación por biométrica	603	9.6.2 Listas de control de acceso	647
9.3.4 Remedios	606	9.6.3 Capacidades	650
9.4 Ataques desde dentro del sistema	606	9.7 Sistemas de confianza	653
9.4.1 Caballos de Troya	607	9.7.1 Base de cómputo de confianza	654
9.4.2 Falsificación de inicio de sesión	608	9.7.2 Modelos formales de sistemas seguros	655
9.4.3 Bombas lógicas	609	9.7.3 Seguridad multinivel	657
9.4.4 Trampas	610	9.7.4 Seguridad de libro naranja	659
9.4.5 Desbordamiento de búfer	610	9.7.5 Canales encubiertos	661
9.4.6 Ataques genéricos contra la seguridad	613	9.8 Investigación sobre seguridad	665
9.4.7 Defectos de seguridad famosos	614	9.9 Resumen	666

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

Para evitar confusiones, se utilizará el término **seguridad** para referirnos al problema general, y el término **mecanismos de protección** para referirnos a los mecanismos específicos del sistema operativo que sirven para salvaguardar la información en la computadora.

Desde una perspectiva de seguridad, los sistemas de computación tienen tres metas generales, con sus respectivas amenazas (como se muestra en la figura 9-1 en la página 584 del libro de texto). La primera, la **confiabilidad de los datos**, tiene que ver con mantener en secreto los datos secretos. La segunda meta, la

integridad de los datos, implica que los usuarios no autorizados no podrán modificar ningún dato sin permiso del dueño. La tercera meta, **disponibilidad del sistema**, implica que nadie podrá alterar el sistema de modo que no pueda usarse.

Otro aspecto del problema de la seguridad es la **privacidad**: proteger a las personas contra el mal uso de su información personal.

A quienes se meten donde no les incumbe se les denomina **intrusos**. Los intrusos actúan de dos maneras. Los intrusos pasivos solo quieren leer archivos que no están autorizados para leer. Los intrusos activos tienen peores intenciones: quieren efectuar cambios no autorizados a los datos.

Entre las categorías más comunes de intrusos están:

1. Curioseo casual por parte de los usuarios no técnicos
2. Husmeo por parte de personal interno
3. Intentos decididos por hacer dinero
4. Espionaje comercial o militar

Entre las causas más comunes de la pérdida accidental de datos están:

1. Actos fortuitos
2. Errores de *hardware* o *software*
3. Errores humanos

El propósito de la **criptografía** es tomar un mensaje o un archivo, llamado **texto simple**, y convertirlo en **texto cifrado**, de tal manera que solo las personas autorizadas puedan convertirlo otra ven en texto simple. Esta táctica se conoce como **seguridad de ocultamiento**, y solo los aficionados en materia de seguridad la utilizan.

Muchos sistemas criptográficos tienen la propiedad de que, dada la clave de cifrado, es fácil deducirla, y viceversa. Tales sistemas se llaman **criptografía de clave secreta** o **criptografía de clave simétrica**.

La **criptografía de clave pública** tiene la propiedad de que se usan claves distintas para descifrar y cifrar. Si se escoge bien la clave de cifrado, es casi imposible descubrir a partir de ella la clave de descifrado correspondiente.

Cuando un usuario inicia una sesión en una computadora, lo normal es que el sistema operativo quiera determinar quién es el usuario. Este proceso se denomina **autenticación de usuarios.**

La forma de autenticación más utilizada consiste en pedir al usuario que teclee un nombre de inicio de sesión y una contraseña. La protección por contraseña es fácil de entender y de implementar.

Casi todos los *crackers* se introducen al sistema objetivo estableciendo una conexión con este y probando muchas combinaciones (inicio de sesión, contraseña) hasta hallar una que funcione.

El programa que invocan los usuarios para instalar o cambiar su contraseña también puede prevenir al usuario de que se está escogiendo una mala contraseña. Entre las cosas por las que podría emitir una advertencia están:

1. Las contraseñas deben tener como mínimo siete caracteres.
2. Deben contener letras tanto mayúsculas como minúsculas.
3. Deben contener al menos un dígito o carácter especial.
4. No deben ser palabras del diccionario, nombres de personas, etc.

Un segundo método para autenticar usuarios consiste en examinar algún objeto físico que poseen en lugar de algo que saben. El objeto físico empleado a menudo es una tarjeta de plástico que se inserta en un lector enlazado con la terminal o la computadora. Por lo regular, el usuario no solo debe insertar la tarjeta sino también teclear una contraseña, para evitar que alguien use una tarjeta extraviada o robada.

Un tercer método de autenticación mide características físicas del usuario que son difíciles de falsificar. Esto se conoce como **biométrica**. Por ejemplo, un lector de huellas dactilares o de patrón de voz en la terminal podría verificar la identidad del usuario.

(Para ahondar más en este tema, estudie las secciones 9-1, 9-2 y 9-3 del libro de texto, páginas de la 583 a la 606.)

Una vez que un *cracker* ha iniciado sesión en una computadora, puede comenzar a causar daños.

Un añejo ataque desde dentro es el **caballo de Troya**. Es un programa al parecer inocente, que contiene código para realizar una función inesperada e indeseable. Dicha función podría consistir en modificar, borrar o cifrar los archivos del usuario, por ejemplo.

Algo con cierta relación con los caballos de Troya es la **falsificación de inicio de sesión**, que funciona como sigue: Se utiliza una pantalla similar ala de inicio de sesión, el usuario teclea su contraseña e identificación. Al ser digitadas, se guardan en un archivo en donde se recolectan estos datos y podrán ser usados posteriormente por un intruso.

Otro ataque desde el interior es la **bomba lógica**. Este dispositivo es un fragmento de código escrito por uno de los programadores de una compañía e insertado de manera subrepticia en el sistema operativo de producción. Mientras este

programa reciba la contraseña del programador diariamente no pasará nada. Pero si el programador es despedido, el programa, al no recibir la contraseña, en cualquier momento puede "estallar".

Otra brecha de seguridad de origen interno es la trampa. Este problema se presenta cuando un programador del sistema inserta código que permite pasar por alto alguna verificación normal. Por ejemplo, podría incluir una sección de código que se salte la verificación de usuario para una contraseña dada que sólo el programador sabe.

El proceder normal para probar la seguridad de un sistema consiste en contratar a un grupo de expertos, conocido como **equipos tigre** o **equipos de penetración**, que tratarán de introducirse en el sistema.

(Para conocer más en este tema estudie la página 613 del libro de texto.)

(Para ahondar más en estos temas, estudie la sección 9-4 del libro de texto, páginas de la 606 a la 614.)

En el caso de máquinas conectadas a Internet o a otra red, existe una creciente amenaza externa. Una computadora en una red puede ser atacada desde una computadora distante a través de la red. En casi todos los casos, semejante ataque consiste en transmitir cierto código por la red a la máquina objetivo, donde se ejecutará y causará daños.

Un **virus** es un programa que puede reproducirse anexando su código a otro programa, de forma análoga a como se producen los virus biológicos. Veamos cómo funcionan los virus. Virgilio escribe su virus, tal vez en lenguaje ensamblador, y luego lo inserta con cuidado en su propia máquina utilizando una herramienta llamada **colocador**. Luego se distribuye ese programa infectado, quizá publicándolo en un tablero de boletines o en una colección de software gratuito por Internet.

El **gusano de Internet** se dio cuando un estudiante descubrió dos errores de programación en Berkeley UNIX, que permitían obtener acceso no autorizado a máquinas por toda Internet. Trabajando solo, escribió un programa que se reproducía a sí mismo, un gusano que aprovechaba esos errores y se reproducía en segundos dentro de cada máquina a la que podía obtener acceso.

En algunos sistemas, la protección se implementa con un programa llamado **monitor de referencias**. Cada vez que se intenta tener acceso a un recurso que podría estar protegido, el sistema pide al monitor de referencias verificar antes si está permitido. El monitor de referencias consulta entonces sus tablas de políticas y toma una decisión.

Un **dominio** es un conjunto de pares (objeto, derechos). Cada par especifica un objeto y algún subconjunto de operaciones que pueden ejecutarse con él. Un **derecho** en este contexto implica permiso para realizar una de las operaciones. Es común que un dominio corresponda a un solo usuario, e indique lo que el usuario puede y no puede hacer realizar con él.

(Para ahondar más en este tema, estudie las secciones 9-5 y 9-6 del libro de texto, páginas de la 617 a la 653.)

La razón por la que no se están construyendo sistemas seguros es más complicada, pero se reduce a dos cuestiones fundamentales. Primera, los sistemas actuales no son seguros pero los usuarios no están dispuestos a deshacerse de ellos.

El segundo problema es más sutil. La única forma de construir un sistema seguro es que sea sencillo. Las funciones son enemigas de la seguridad. Los diseñadores creen que los usuarios quieren más funciones. Más funciones implican más complejidad, más código, más errores de programación y más fallas de seguridad.

En el mundo de la seguridad se habla a menudo de **sistemas de confianza** más que de sistemas seguros. Estos son sistemas que han planteado formalmente requisitos de seguridad y cumplen con ellos. En el corazón de todo sistema de confianza está una **base de computo de confianza** Consiste en el hardware y el software necesarios para hacer cumplir todas las reglas en materia de seguridad. Si la base de cómputo de confianza opera según las especificaciones, la seguridad del sistema no podrá ser violada, sin importar que lo demás esté mal.

Casi todos los sistemas operativos permiten a usuarios individuales determinar quién puede leer y escribir sus archivos y otros objetos. Esta política se denomina **control de acceso a discreción**.

(Para ahondar más en este tema, estudie las secciones 9-7 del libro de texto, páginas de la 553 a la 666.)

Ejercicios sugeridos

A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 9, realice los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 667, 668, 669 y 670: 9, 14, 15, 9, 14, 15, 17, 27.

Resolución de ejercicios sugeridos

Ejercicio 9

Cite tres características que debe reunir un buen indicador biométrico para que pueda utilizarse en la autenticación de inicios de sesión.

a. Debe tener suficiente variabilidad como para que el sistema pueda distinguir sin error entre muchas personas.

b. Debe ser una característica que no varíe mucho con el pasar del tiempo.

c. Debe ser una característica física del usuario difícil de calificar.

(Páginas 603 y 604 del libro de texto.)

Ejercicio 14

Con la proliferación de los cafés Internet, las personas van a exigir que sea posible acudir a uno de ellos en cualquier lugar del mundo y trabajar desde él. Describa una forma de producir documentos firmados desde uno de esos cafés empleando una tarjeta inteligente (suponga que todas las computadoras están equipadas con lectores de tarjetas inteligentes). ¿Es seguro su esquema?

Una tarjeta inteligente contiene un chip que almacena información. En esta se puede almacenar la firma digital. El problema sería que las computadora a la cual está conectada la lectora de la tarjeta guarde en memoria la firma digital. Esto podría ser contraproducente desde el punto de vista de seguridad.

Ejercicio 15

¿El ataque por caballo de Troya puede funcionar en un sistema protegido por capacidades?

Sí. Recordemos que el caballo de Troya es un programa al parecer inocente que contiene código para realizar una función inesperada e indeseable y una capacidad otorga al dueño del proceso ciertos derechos sobre cierto objeto.
Si alguien me envía un archivo público, y yo lo tomo y lo sumo a mi lista de objetos, ese objeto ya es de mi propiedad, y en cualquier momento podría aparecer el código indeseado en operación. Muy distinto es sí alguien rata de tener acceso a un objeto de mi propiedad sin el permiso requerido, ahí no podría tener acceso a ese objeto. (Páginas 607 y 650 del libro de texto.)

Ejercicio 17

Cuando se elimina un archivo, sus bloques por lo general se devuelven a la lista libre, pero no se borran. ¿Cree que sería recomendable que el SO borre todos los bloques antes de liberarlos? Considere factores tanto de seguridad como de desempeño en su respuesta, y explique el efecto de cada una.

Desde el punto de vista de desempeño no sería óptimo, pues no intervienen en nada, y cuando un nuevo proceso quiera utilizarse simplemente la información nueva borra la anterior. Desde el punto de visa de seguridad, habría problemas si hubiera algún proceso que leyera bloques de memoria en desuso pero que no los borre. El proceso debería ser muy sofisticado, pues no siempre la información se guarda en un solo bloque.

Ejercicio 27

¿Cuál es la diferencia entre un virus y un gusano? ¿Cómo se reproduce cada uno?

Un virus es un programa que puede reproducirse anexando su código a otro programa. Un gusano consiste en dos programas: el autoarranque y el gusano propiamente dicho. El autoarranque se compila y se ejecuta en el sistema atacado. Una vez que se está ejecutando se conecta con la máquina de procedencia, carga el gusano principal y se ejecuta. La diferencia está en que el virus se añade a otro programa, el gusano se propaga por sí mismo. (Páginas 617 y 635 del libro de texto.)

CAPÍTULO 12

DISEÑO DE SISTEMAS OPERATIVOS

Sumario

- La naturaleza del problema de diseño
- Diseño de interfaces
- Implementación
- Desempeño
- Administración de proyectos
- Tendencias en el diseño de sistemas operativos

Propósito del capítulo

En este capítulo examinaremos brevemente algunos de los problemas y compromisos que deben tener en cuenta los diseñadores de sistemas operativos al diseñar e incorporar un sistema nuevo.

Objetivos

Al finalizar el estudio de este tema, el estudiante deberá estar en capacidad de:

- Explicar la naturaleza del problema de diseño.
- Explicar lo referente a diseño de interfaces.
- Explicar los pasos de la implementación.
- Explicar lo referente al desempeño.
- Conocer acerca de la administración de proyectos.

Guía de lecturas

Para lograr los objetivos descritos anteriormente, es importante que realice las siguientes lecturas:

Subtema	Pp.	Subtema	Pp.
12.1 La naturaleza del problema de diseño	856	12.4.3 Equilibrio espacio-tiempo	884
12.1.1 Metas	856	12.4.4 Uso de cachés	886
12.1.2 ¿Por qué es difícil diseñar sistemas operativos?	587	12.4.5 Sugerencias	888
12.2 Diseño de interfaces	859	12.4.6 Aprovechamiento de la localidad	888
12.2.1 Principios orientadores	859	12.4.7 Optimización del caso común	889
12.2.2 Paradigmas	861	12.5 administración de proyectos	889
12.2.3 La interfaz de llamadas al sistema	864	12.5.1 El mes-hombre mítico	890
12.3 Implementación	867	12.5.2 Estructura de equipo de trabajo	891
12.3.1 Estructura del sistema	867	12.5.3 El papel de la experiencia	893
12.3.2 Mecanismo en comparación con políticas	870	12.5.4 No hay bala de plata	894
12.3.3 Ortogonalidad	871	12.6 Tendencias en el diseño de sistemas operativos	894
12.3.4 Asignación de nombres	872	12.6.1 Sistemas operativos con espacio de direcciones grande	894
12.3.5 Tiempo de enlace	874	12.6.2 Redes	895
12.3.6 Estructuras estáticas o dinámicas	875	12.6.3 Sistemas paralelos y distribuidos	896
12.3.7 Implementación descendente o ascendente	876	12.6.4 Multimedia	896
12.3.8 Técnicas útiles	877	12.6.5 Computadoras alimentadas por baterías	896
12.4 Desempeño	882	12.6.6 Sistemas incrustados	897
12.4.1 ¿Por qué son lentos los sistemas operativos?	882	12.7 Resumen	897
12.4.2 ¿Qué debe optimizarse?	883		

Comentarios generales

Una vez que usted ha realizado las lecturas anteriores, analice cuidadosamente los comentarios siguientes, tomados del libro de texto, ya que con ellos se pretende enfatizar o ampliar algunos contenidos importantes del tema. Existen varios aspectos importantes que debemos señalar, antes de entrar al desarrollo de los subtemas.

Para poder diseñar con éxito un sistema operativo, los diseñadores deben tener una idea clara de lo que quieren. La falta de una meta dificulta sobremanera la toma de decisiones subsiguientes.

En el caso de los sistemas operativos de propósito general hay cuatro objetivos principales:

1. Definir abstracciones.
2. Proporcionar operaciones primitivas.
3. Garantizar el aislamiento.
4. Administrar el *hardware*.

La tarea más difícil es definir las abstracciones correctas. Algunas de ellas, como los procesos y archivos, se han usado desde hace tanto tiempo que podrían parecer obvias. Otras, como los subprocesos, son más recientes y menos maduras.

Cada una de las abstracciones puede ilustrarse en forma de estructuras de datos concretas. Los usuarios pueden crear procesos, archivos semáforos, etc. Las operaciones primitivas manipulan estas estructuras de datos.

Puesto que puede haber múltiples usuarios en sesión al mismo tiempo en una computadora, el sistema operativo debe proporcionar mecanismos para mantenerlos separados. Un usuario no debe interferir con otro.

Una meta clave del diseño de sistemas es asegurarse de que cada usuario sólo pueda ejecutar operaciones autorizadas con datos autorizados.

El sistema operativo tiene que administrar el *hardware*. En particular, tiene que ocuparse de todos los chips de bajo nivel, como las controladoras de interrupciones y las controladoras de bus. También tiene que proporcionar un marco en el que las controladoras de dispositivos puedan controlar los dispositivos de E/S más grandes, como discos, impresoras y la pantalla.

Mencionemos ocho de las cuestiones que hacen que el diseño de un sistema operativo sea mucho más difícil que el de un programa de aplicación:

1. Los sistemas operativos se han convertido en programas extremadamente grandes. Ninguna persona puede sentarse frente a una PC y producir un sistema operativo serio en unos cuantos meses.

2. Los sistemas operativos tienen que manejar concurrencia. Existen múltiples usuarios y múltiples dispositivos de E/S, todos los cuales están activos a la vez. Administrar concurrencias es, por naturaleza, mucho más difícil que administrar una sola actividad secuencial.

3. Los sistemas operativos tienen que enfrentar a usuarios hostiles en potencia. El sistema operativo necesita tomar medidas para impedir que esos usuarios se comporten de manera indebida.

4. A pesar de que no todos los usuarios confían en los demás, muchos sí quieren compartir parte de su información y recursos con otros usuarios selectos. El sistema operativo tiene que posibilitar esto, pero de tal manera que los usuarios malintencionados no puedan interferir.

5. Los sistemas operativos tienen una vida larga. Por ello, los diseñadores tienen que tomar en cuenta la manera en que el *hardware* y las aplicaciones van a cambiar en el futuro y cómo deben prepararse para tales cambios.

6. Los diseñadores de sistemas operativos en realidad no tienen una idea muy clara de cómo van a usar sus sistemas, por lo que necesitan incorporar un alto grado de generalidad en ellos.

7. Los sistemas operativos modernos suelen diseñarse de modo que sean portables, lo que significa que tienen que funcionar en múltiples plataformas de *hardware*. También tienen que reconocer cientos o incluso miles de dispositivos de E/S, los cuales se diseñan en forma independiente.

8. Y, por último, la necesidad frecuente de ser compatible con algún sistema operativo anterior. Ese sistema pudo haber tenido restricciones en cuanto a longitud de palabras, nombre de archivos u otros aspectos que los diseñadores ahora consideran obsoletos, pero de los que no pueden liberarse.

(Para ahondar más en este tema, estudie la sección 12.1 del libro de texto, páginas de la 855 a la 859.)

Un sistema operativo presta un conjunto de servicios, en su mayoría tipos de datos (por ejemplo, archivos) y operaciones que se realizan con ellos. Juntos constituyen la interfaz con los usuarios.

Un adecuado **diseño de interfaces** debe seguir ciertos principios. Las interfaces deber ser sencillas, completas y susceptibles de implementarse de manera eficiente.

Principio 1: Sencillez. Una interfaz sencilla es más fácil de entender e implementar sin errores

Principio 2: Integridad. La interfaz debe permitir hacer todo lo que los usuarios necesitan hacer, es decir, debe ser completa o íntegra.

Principio 3: Eficiencia. Si una característica o llamada al sistema no puede implementarse de forma eficiente, quizá no vale la pena tenerla. También debe ser intuitivamente obvio para el programador cuánto cuesta una llamada al sistema.

Una vez establecidas las metas, puede comenzar el diseño. Un buen punto de partida es pensar en cómo van a ver el sistema los clientes. Una de las cuestiones más importantes es cómo hacer que todas las características del sistema formen

un conjunto consistente, y presenten lo que se conoce como **coherencia arquitectónica**.

Es este sentido, es importante distinguir dos tipos de **clientes de los sistemas operativos**. Por un lado están los **usuarios**, que interactúan con programas de aplicación; por el otro lado están los **programadores**, que los escriben. Los usuarios tratan sobre todo con la GUI. Los programadores tratan principalmente con la interfaz de llamadas al sistema.

Tanto para la interfaz en el nivel GUI como para la interfaz de llamadas al sistema, el aspecto más importante es tener un buen **paradigma** que proporcione una forma de ver la interfaz. Por ejemplo, el paradigma WIMP utiliza acciones de apuntar y hacer clic, apuntar y hacer doble clic, arrastrar, entre otras, para conferir una coherencia arquitectónica a la totalidad.

Sea cual sea el paradigma que se escoja, es importante que lo usen todos los programas de aplicación. Por ello, los diseñadores de sistemas deben proporcionar a los creadores de aplicaciones bibliotecas y conjuntos de herramientas con los que puedan acceder a procedimientos que produzcan el aspecto y el manejo uniformes.

Hay dos paradigmas de ejecución que se usan en forma amplia: el algoritmo y el controlado por sucesos.

El **paradigma algorítmico** se basa en la idea de que un programa se inicia para realizar alguna función que conoce con antelación o que obtiene de sus parámetros. La lógica básica viene incorporada en el código, y el programa emite llamadas al sistema cada cierto tiempo para obtener entradas del usuario, solicitar servicios del sistema operativo, etc.

El otro paradigma de ejecución es el **paradigma controlado por sucesos**. Aquí el programa realiza algún tipo de preparación inicial, digamos exhibir cierta pantalla, y luego espera a que el sistema operativo le notifique del primer suceso. Muchas veces el suceso es la pulsación de una tecla o un movimiento del ratón. Este diseño es muy útil para los programas muy interactivos.

Otro paradigma igualmente importante es el **paradigma de datos**. Se refiere a datos en cinta, archivos, objetos o documentos. En los cuatro casos, la intención es unificar los datos, dispositivos y demás recursos para que sea más fácil manejarlos. Todo sistema operativo debe tener un paradigma de datos que lo unifique.

El sistema operativo debería ofrecer el menor número posible de llamadas al sistema, y cada una deberá ser lo más sencilla posible.

En algunos casos, podría parecer que las llamadas al sistema necesitan ciertas variantes, pero muchas veces es más recomendable tener una sola llamada al

sistema que maneje el caso general, con diferentes procedimientos de biblioteca para ocultar ese hecho a los programadores.

Si el *hardware* cuenta con un mecanismo muy eficiente para hacer algo, debe exponerse a la vista de los programadores de forma sencilla, y no enterrarlo en alguna otra abstracción. El propósito de las abstracciones es ocultar propiedades indeseables, no ocultar las deseables.

(Para ahondar más en este tema, estudie la sección 12.2 del libro de texto, páginas de la 861 a la 866.)

Dejando a un lado las interfaces y las llamadas al sistema, examinemos ahora cómo puede **implementarse un sistema operativo**.

Un método razonable que se ha establecido bien con el paso de los años es **sistema de capas.** En el caso de un sistema nuevo, los diseñadores que decidan seguir esta ruta primero deberán escogen con mucho cuidado las capas y definir la funcionalidad de cada una de ellas. (Estudie lo referente a este tema en las páginas 867 y 868 del libro de texto.)

Aunque la estructura de capas tiene partidarios entre los diseñadores del sistema, también hay una fracción que adopta una perspectiva diametralmente opuesta. Su punto de vista se basa en **argumento de extremo a extremo**. Este concepto dice que si el programa de usuario tiene que hacer algo él mismo, es un desperdicio hacerlo también en un capa más baja.

Un término medio entre hacer que el sistema operativo se encargue de todo o que no haga nada, es que el SO haga un poquito. Este diseño, llamado **cliente.-servidor**, lleva a un *microkernel,* con una buena parte del sistema operativo ejecutándose como procesos servidores en el nivel de usuario. Este es el modelo más modular y flexible de todos. Lo fundamental en flexibilidad es que cada controlador de dispositivo se ejecute también como proceso de usuario, protegido en forma plena contra el *kernel* y otros controladores. El problema principal con este método, y con los *microkernels* en general, es la merma en desempeño causada por todas las conmutaciones de contexto adicionales.

En los sistemas cliente-servidor se buscaba aprovechar el *kernel* al máximo. El enfoque opuesto consiste en colocar más módulos en el *kernel*, pero de forma protegida.

Otra cuestión pertinente aquí es la de los subprocesos de sistema, sea cual sea el modelo de estructuración escogido. A veces conviene permitir la existencia de subprocesos de *kernel* independientes de cualquier proceso de usuario. Estos procesos pueden ejecutarse en segundo plano.

(Para ahondar más en este tema, estudie la sección 12.3 del libro de texto, páginas de la 867 a la 882.)

En condiciones iguales, un sistema operativo rápido es mejor que uno lento. Sin embargo, un sistema operativo rápido y poco confiable no es tan bueno como uno lento pero confiable.

Tal vez lo más importante que los diseñadores de sistemas pudieran hacer para mejorar el desempeño sería mostrarse mucho más selectivos en cuanto a añadir nuevas funciones y características.

Un método general para mejorar el desempeño es sacrificar tiempo a cambio de espacio. Al efectuar una optimización importante, conviene buscar algoritmos que ganen velocidad ocupando más memoria o, por otro lado, que ahorren memoria escasa realizando más cálculos.

Una técnica muy conocida para mejorar el desempeño es el uso de **cachés**. La estrategia general consiste en efectuar todo el trabajo la primera vez y luego guardar el resultado en un caché. En los siguientes intentos, primero se verifica el caché. Si el resultado está ahí, se usa; si no, se vuelve a realizar todo el trabajo.

Las **entradas de caché** siempre son correctas. Una búsqueda en caché podría fallar, pero si encuentra una entrada, se garantiza que es correcta y puede usarse sin más. En algunos sistemas conviene tener una **tabla de sugerencias**. Estas son sugerencias en cuanto a la solución, pero no se garantiza que sean correctas.

Los procesos y programas no actúan al azar: exhiben un alto grado de localidad en el tiempo y en el espacio, y hay varias formas de aprovechar esta información para mejorar el desempeño. El **principio de localidad** también es válido para los archivos. Una vez que un proceso ha seleccionado su directorio de trabajo, es probable que muchas de sus referencias futuras sean a archivos que están en ese directorio.

Otra área en la que la localidad desempeña un papel es la **calendarización de subprocesos en multiprocesadores**. Una forma de calendarizar subprocesos en un multiprocesador es tratar de ejecutar cada subproceso en la última CPU que utilizó, con la esperanza de que algunos de sus bloques de memoria todavía estén en el caché de memoria.

En muchos casos es recomendable distinguir entre el caso más común y el peor caso posible y tratarlos de distinta manera. Con frecuencia el código para los dos casos es muy diferente. Es importante lograr que el caso común sea rápido. El peor caso, si no se presenta a menudo, solo tiene que manejarse en forma correcta.

(Para ahondar más en este tema, estudie la sección 12.4 del libro de texto, páginas de la 882 a la 889.)

Ejercicios sugeridos

A continuación, se presentan ejercicios que tienen como objetivo evaluar cada uno de los puntos contenidos en el desarrollo del capítulo. Para el capítulo 12, realice los siguientes ejercicios que se encuentran localizados en el libro de texto en las páginas 898, 899 y 900: 2, 6, 10, 11, 14, 27.

Resolución de ejercicios sugeridos

Ejercicio 2

En la figura 12-1 se muestran dos paradigmas, el algorítmico y el controlado por eventos. Indique qué paradigma puede ser más fácil de usar cada uno de los tipos de programas siguientes:

a. Un compilador
b. Un programa para editar fotografías
c. *Un programa de nómina*

a. Algorítmico
b. Controlado por suceso
c. Algorítmico

(Páginas 862 y 863 del libro de texto.)

Ejercicio 6

Supongamos que se intercambian las capas 3 y 4 de la figura 12-2. ¿Qué implicaciones tendría eso para el diseño del usuario?

Lo que se busca con la capa 3 es tener todo el ambiente preparado, para que, al ejecutarse a capa 4, se tengan subprocesos correctos que se calendaricen en forma normal y se sincronicen empleando un mecanismo estándar. Si se intercambian estos procesos, no se tendrá el ambiente debidamente preparado, y el incremento de fallas en los procesos puede ser mayor. (Páginas 867 y 868 del libro de texto.)

Ejercicio 10

Muchas veces, los sistemas operativos usan nombres en dos niveles: externos e internos. Indique qué diferencias hay entre esos nombres en cuanto a:

a. *Longitud*

El nombre externo le permite al usuario identificar más claramente un archivo, pues el nombre es más significativo. Internamente, el nombre debe ser más corto, para no utilizar mucho espacio en memoria.

b. *Carácter único*

Externamente, el archivo es una cadena de caracteres. Internamente, es un código en una tabla de referencia o índice.

c. Jerarquías

Externamente, es importante que el usuario tenga una idea de dónde está ubicado el archivo que busca, o sea, en cuál subdirectorio. Lo anterior le puede ayudar a diferenciar entre dos archivos con el nombre igual o similar. Desde el punto de vista interno, se crea una correspondencia directa entre el nombre externo y una dirección de memoria, donde se encuentra el archivo. Internamente, no se manejan jerarquías de directorios y subdirectorios, sino un índice a una tabla.

(Páginas 872 a la 874 del libro de texto.)

Ejercicio 11

Una forma de manejar tablas cuyo tamaño no se conoce con antelación es hacerlas fijas. Pero cuando una se llena, se sustituye por una más grande, se copian las entradas antiguas en la nueva tabla y se destruye la tabla vieja. ¿Qué ventajas y desventajas tiene hacer que la nueva tabla sea dos veces mayor que la original, en comparación con hacerla sólo 1.5 veces mayor?

Las ventajas son que, en una sola llamada al sistema, se puede duplicar el espacio de la tabla, esto siempre y cuando se cuente con suficiente espacio en memoria. Además, con un método así, la tabla siempre será continua, no enlazada.

La desventaja aquí es que se requiere de cierta administración del almacenamiento, y la dirección de la tabla ahora es una variable, no una constante.

(Página 875 del libro de texto.)

Ejercicio 14

La indirección es una forma de hacer más flexible un algoritmo. ¿Tiene desventajas? Y en caso de tenerlas, ¿cuáles son?

La desventaja que podría tener es que se deben crear tablas indizadas para direccionar cada uno de los dispositivos. Por ejemplo, en el caso del teclado, cada letra minúscula y mayúscula, así como los caracteres especiales y teclas de función, están en una tabla que las relaciona con su respectivo código ASCII. En otras palabras, al causar una interrupción por teclado (apretar una tecla), primero hay que consultar una tabla que es fija, o sea, no va directamente al código ASCII. La desventaja sería espacio para la tabla y tiempo en la consulta de la tabla.

(Páginas 879 y 880 del libro de texto.)

Ejercicio 27

Cite algunas características de un sistema convencional que no se necesiten en un sistema incrustado para un aparato doméstico.

- Atención de diferentes usuarios: El aparato doméstico es de programación física, para lo cual ya está todo programado.
- Procesos de E/S diversos: El aparato doméstico tiene entradas específicas, no se da la posibilidad de hacer combinaciones de entradas. Un aparato doméstico tiene perillas o ingreso por teclado, o funciones específicas con nombres.
- Los SO convencionales permiten estarse renovando, y rara vez se requiere de un cambio en el *hardware*. Si se quiere renovar un aparato doméstico por mejores funciones, debe adquirir uno nuevo y desechar el anterior.

(Página 897 del libro de texto.)