Daniel
Sol Llaven

CD interactivo en esta edición

SISTEMAS OPERATIVOS

Panorama para ingeniería en computación e informática

Sistemas operativos

Panorama para la Ingeniería en Computación e Informática

Ing. Daniel Sol Llaven
Universidad Nacional Autónoma de México

México, 2015

GRUPO EDITORIAL PATRIA

Para establecer comunicación con nosotros puede hacerlo por:

correo:
Renacimiento 180, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México, D.F.

fax pedidos:
(01 55) 5354 9109 • 5354 9102

e-mail:
info@editorialpatria.com.mx

home page:
www.editorialpatria.com.mx

Dirección editorial: Javier Enrique Callejas
Coordinadora editorial: Estela Delfín Ramírez
Supervisor de preprensa: Gerardo Briones González
Diseño de portada: Juan Bernardo Rosado Solís/Signx
Ilustraciones: Adrián Zamorategui Berber
Fotografías: © Thinkstockphoto

Revisión técnica: Jorge Cortés Galicia
Instituto Politécnico Nacional
Laura Sandoval Montaño
Universidad Nacional Autónoma de México

Sistemas operativos. Panorama para la Ingeniería en Computación e Informática

Renacimiento 180, Colonia San Juan Tlihuaca
Azcapotzalco, México D. F.

Miembro de la Cámara Nacional de la Industrial Editorial Mexicana
Registro Núm. 43

ISBN ebook: 978-607-744-267-7

Impreso en México
Printed in Mexico

Primera edición ebook: 2015

Dedicado a mi familia, por hacerme quien soy; y a mis colegas, por todo lo que he aprendido de ustedes.

Y, por supuesto, mil gracias a todo el equipo editorial y revisores por el apoyo para realizar este esfuerzo.

Daniel Sol

Contenido

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 6

Capítulo 7

Capítulo 8

Estructura del libro

En el presente libro se revisan los temas presentes en la mayoría de los temarios de Ingeniería en Computación, Ingeniería en Sistemas e Ingeniería Informática. El sistema operativo es un elemento básico para la operación de los sistemas y para el desarrollo de aplicaciones, por lo que constituye un tema relevante que la mayoría de las instituciones de educación superior en el ramo de la ingeniería decidieron reconocer como materia. El enfoque clásico del sistema operativo como administrador de recursos, expuesto por autores como Andrew Tanenbaum, Glenn Brookshear y William Stallings en los libros de su autoría, permea en los temarios y en los primeros capítulos de este libro.

El capítulo 1 de la presente obra delimita el campo de estudio y establece las definiciones necesarias para abordar el tema que se detalla en los capítulos siguientes.

El capítulo 2 explora la definición del sistema operativo, los objetivos que persigue y la estructura general de las diversas versiones existentes.

En el capítulo 3 se analiza de manera detallada la administración de la capacidad de procesamiento en la atención de múltiples tareas de forma concurrente, mejor conocida como modelo de procesos. Se exploran también algunos problemas a los que se enfrentan los sistemas que aprovechan las capacidades de multiprocesamiento, así como las herramientas de que dispone el sistema operativo para enfrentarlos.

En el capítulo 4 se aborda cómo se administra la capacidad de almacenamiento de la información de la computadora, o memoria, que ha de ser transformada por los procesos. Asimismo, se exponen las formas en que se divide la memoria para aislar las actividades de los procesos y se revisan los modelos que combinan las ventajas de la paginación y la segmentación en los sistemas operativos más usados en la actualidad.

El capítulo 5 se centra en el manejo de los dispositivos de entrada y salida, y se explora la diversidad de mecanismos con que cuenta una computadora para establecer la comunicación con sus dispositivos. Además, se revisan las capas en que se organiza la funcionalidad que el sistema operativo implementa para soportar el aprovechamiento de los dispositivos por medio de las aplicaciones.

El capítulo 6 señala las características de los sistemas de archivos y su relación con el sistema de protección, así como los mecanismos que implementa para funcionar como medio de representación uniforme para otros elementos del sistema operativo.

Los capítulos 7 y 8 se ciñen a los contenidos más recientes de los temarios de sistemas operativos que abordan las tendencias actuales en el desarrollo de sistemas y su influencia en estos sistemas.

El capítulo 7 aborda los principales estilos arquitectónicos para sistemas distribuidos y los servicios que el *middleware* y el sistema operativo implementan para soportar el desarrollo de este tipo de sistemas.

Por último, el capítulo 8 muestra un panorama de la seguridad informática. A lo largo de los primeros capítulos de esta obra se exponen las medidas de protección que deben incluirse en los sistemas operativos, y en este se completa el panorama mediante el establecimiento de un marco conceptual en el que pueda orientarse el esfuerzo y racionalizar la inversión para implementar un nivel de protección adecuado en los sistemas con referencia a las medidas mencionadas en capítulos anteriores.

Aunque en los temarios se suele manejar como un tema específico, la realidad de la elaboración, el diseño y el uso de los sistemas hace que las medidas de seguridad y protección se consideren de manera constante, desde el principio del diseño y a lo largo del ciclo de vida de los sistemas. Por lo anterior, sería inapropiado tratar estos temas como si fueran un aspecto separado del diseño y de la operación de la funcionalidad de los sistemas operativos y no como una parte integral de sus características.

Explicación de las notas de apoyo

Para el ingeniero especializado en sistemas es importante alternar diversas funciones y tener la capacidad de aportar la visión, los conocimientos y el enfoque adecuados. El estudio de los sistemas operativos no es la excepción, y por este motivo el autor se ha enfocado en las observaciones sobre los temas tratados de acuerdo con algunas de dichas funciones.

A continuación se describen las cuatro perspectivas generales que el lector de esta obra puede adoptar. Cada uno de estos enfoques está representado por un símbolo que lo identificará a lo largo del texto, y en cada capítulo se incluirán notas de apoyo específicas referentes a cada uno de estos símbolos. Sin embargo, esto no significa que el lector deba limitarse a un solo enfoque, aunque puede encontrar cierta afinidad por alguno y, de esta manera, aprovechar mejor toda la obra.

♠ **Arquitecto (ἀρχιτέκτων Primero-obra).** Orientado al desarrollo de los sistemas desde las habilidades básicas de programación hasta el "estado del arte" de los sistemas.

♦ **Líder de proyecto.** Orientado a la coordinación y control de los proyectos de software y equipos de trabajo de sistemas.

♥ **Docente.** Orientado a la formación de personal con cualquiera de los cuatro perfiles.

- **Usuario o desarrollador.** Orientado al aprovechamiento, revisión y mejora de los sistemas, en particular en cuestiones de seguridad, tolerancia a los fallos y escalabilidad. Siempre en una perspectiva de retorno de la inversión.

COMPETENCIAS A DESARROLLAR

- El alumno podrá distinguir al sistema operativo de los demás elementos que conforman un sistema de información y conocerá las principales perspectivas que se emplean para su definición.
- Identificará las influencias principales en el desarrollo de los sistemas operativos a lo largo de la evolución del cómputo, así como algunas de las tendencias presentes en la etapa de desarrollo actual.
- Identificará los principales tipos de sistemas operativos según la organización de su código y su esquema de comercialización, así como la relación entre estos.
- Conocerá la organización del presente material.

Introducción al estudio de los sistemas operativos

1

¿QUÉ SABES...?

1. ¿Qué son los sistemas operativos? ¿Para qué sirven?
2. Realiza consultas en Internet acerca de las características que promocionan los principales sistemas operativos. Presta especial atención a los productos o servicios concretos que brindan.
3. Con base en la información que obtuviste, identifica los aspectos más relevantes con los que los sistemas operativos aportan valor a los individuos y a las organizaciones que los usan.
4. ¿Por qué son rentables? Aunque no se puede responder de manera formal a esta pregunta sin información financiera, para el plano personal es factible hacer un análisis aproximado de la ganancia en tiempo y dinero ahorrado o ganado debido al uso correcto de las computadoras en la vida diaria. ¿En qué grado depende este uso de los aspectos considerados en las preguntas anteriores? ¿Podría evaluarse el costo de carecer de esta contribución? Y, finalmente, si comparamos el costo con la ganancia de usarlos, ¿se justifica la inversión realizada en los sistemas operativos?
5. Como profesional del cómputo, ¿qué importancia consideras que tiene conocer las características de los sistemas operativos y a qué nivel deberías hacerlo?
6. ¿Cuáles son las necesidades que atienden los sistemas operativos y cómo evolucionan estas al paso del tiempo?
7. ¿Qué ventaja aportó o promocionó la última versión del sistema operativo de tu preferencia y qué tan relevante te parece?

1.1 Definiciones

Comencemos por delimitar nuestro campo de estudio con una definición:

El **sistema operativo** es una colección de programas que comparten los mismos mecanismos de distribución. Se genera con el propósito de administrar y extender los recursos o capacidades de los sistemas de información.

Las capacidades a las que se refiere nuestra definición son: el procesamiento, realizado por la CPU; el almacenamiento de información, que llevan a cabo la memoria y los dispositivos de almacenamiento masivo, en los que se incluyen los sistemas de archivos; el manejo y aprovechamiento de los demás dispositivos conectados a la computadora, por lo común conocidos como dispositivos de entrada y salida; y también los demás programas o aplicaciones presentes en el sistema que serán los que implementen la fun-

cionalidad concreta que los usuarios requieren. Aunque en realidad no son recursos del sistema, los usuarios también son administrados y potenciados por el sistema operativo.

El sistema operativo desempeña tanto el cometido de administrador, con el fin de vigilar que los recursos sean empleados de manera eficiente y respetuosa de las necesidades de las aplicaciones, como el de potenciador de la funcionalidad. Así, los procesadores podrán atender a un mayor número de procesos, la memoria disponible a los procesos podrá sobrepasar la RAM disponible en el sistema, las aplicaciones podrán colaborar entre sí de formas desconocidas cuando se desarrollan y, con ello, podrán aprovechar la funcionalidad que se logre posteriormente o en formas novedosas gracias a las facilidades de comunicación que el sistema operativo les proporciona, entre otras cualidades.

La relación entre el sistema operativo y los usuarios se deja a menudo en segundo plano debido a que las necesidades y características completas de los usuarios rebasan por mucho el alcance de los sistemas de información; sin embargo, en la relación del usuario con las aplicaciones hay algunos aspectos de gran importancia que el sistema operativo ayuda a implementar y uniformar para facilitar el desarrollo de las aplicaciones y el aprovechamiento de los sistemas por parte de los usuarios. La autenticación de los usuarios, herramienta fundamental para proteger la información que ellos depositan en los sistemas, es uno de los temas principales, así como la implementación de normas y mecanismos para las interfaces de interacción con los usuarios.

Asimismo, es conveniente revisar las definiciones de algunos otros autores para complementar la aquí propuesta:

> El sistema operativo es el software que controla la operación general de una computadora, proporciona los medios por los que un usuario puede almacenar y recuperar archivos, provee la interfaz por la que un usuario puede solicitar la ejecución de programas y provee el ambiente necesario para que los programas solicitados se ejecuten.
>
> J. Glenn Brookshear. *Computer Science, an Overview*, Pearson/Addison- Wesley, 9a ed. 2007.

> Un sistema operativo es un programa que maneja el hardware de la computadora. También provee la base para los programas de aplicación y actúa como intermediario entre el usuario de la computadora y el hardware de esta.
>
> Abraham Silberschatz *et al. Operating System Concepts*, John Wiley & Sons, Inc., 7a ed.

En estas definiciones se aprecia el énfasis en la parte del sistema operativo que se desarrolló primero, que es la capacidad de administrar otros procesos y soportar su ejecución. Esta funcionalidad central y exclusiva del sistema operativo se encuentra en

lo que se conoce como núcleo o *kernel* del sistema operativo, y será de gran importancia en el capítulo 3 de *Administración de procesos*.

Kernel proviene de la raíz germánica *Kern*, que significa núcleo o hueso, por lo que nos referiremos a esta parte del sistema operativo como núcleo o *kernel* de manera indistinta.

Otros autores, como Andrew Tannenbaum y William Stallings, reconocen la dificultad de emitir una definición debido a la gran variedad de productos y enfoques existentes, y, por tanto, analizan los aspectos por separado, ya sea como administrador de recursos o extendiendo las capacidades del equipo. Ambos ponen especial atención en otro aspecto fundamental de los sistemas operativos: el soporte que este brinda para el desarrollo de aplicaciones. El sistema operativo expone interfaces de programación, programas de utilería y servicios que las aplicaciones utilizan para reducir el esfuerzo de su desarrollo y mejorar su independencia física y lógica de las características específicas del hardware, e incluso de otras aplicaciones que sean parte del mismo sistema de información.

Dos conceptos que conviene revisar también son el de *sistema de información* y el de *middleware*, ya que ambos están muy cercanos a los del sistema operativo, pero no deben confundirse con él.

Sistema de información es la colección de programas, equipo de cómputo y telecomunicaciones, junto con los usuarios, con un propósito particular.

Por sistema se entiende desde una sola computadora portátil con un único usuario hasta colecciones de millones de computadoras cooperando mediante Internet para hacer simulaciones moleculares. La parte que define y distingue a los sistemas es el propósito que guía los esfuerzos realizados en ellos.

El **middleware** es el software empleado por dos o más programas para comunicarse entre sí o con los diversos componentes de un sistema de información. El primer *middleware* fue la funcionalidad de comunicación entre procesos presente en sistemas operativos como Unix; sin embargo, conforme se lograron arquitecturas más complejas, el software desarrollado para soportar los requerimientos de estas se trabajó por separado, y aunque suele acoplarse al sistema operativo, busca proporcionar plataformas uniformes para las aplicaciones que las empleen en favor de la independencia física y lógica.

1.2 Evolución histórica

La historia reciente del cómputo es un tema de gran riqueza e interés que debe revisarse por su propio mérito. Hay algunos aspectos que conviene repasar en particular para mejorar la comprensión de los motivos que guían el desarrollo de los sistemas operati-

vos y que nos han conducido a muchas de las características que tienen actualmente, así como a la creación y desaparición de otras.

Debido al rápido avance de la tecnología y al costo que implica modificar una aplicación que funciona, es común, como en muchos otros fenómenos históricos, que técnicas o productos de diversas etapas de desarrollo convivan, incluso durante largos periodos; por esta razón, en vez de seguir una cronología comentaremos las características y presiones que afectan una serie de etapas de desarrollo de los sistemas operativos, e incluso las más primitivas de estas pueden tener variantes relevantes para diversas aplicaciones aun en la época futura en que se lea este libro, y han de convivir con soluciones que aún estén en proceso de ser desarrolladas.

Etapa 0. Computadoras de propósito particular

Las primeras computadoras electromecánicas se realizaron con ciertos propósitos específicos. En el caso de Mark II, este propósito fue el cálculo de tablas de tiro para la Marina al final de la Segunda Guerra Mundial. Como la aplicación era específica, las capacidades del equipo, la forma en que operaban, e incluso los lenguajes, se diseñaron con las capacidades necesarias sin emplear recursos o esfuerzos significativos adicionales. Aunado a esto, los investigadores que desarrollaron el equipo dieron continuidad al diseño de las aplicaciones y soporte a la operación.

En esta situación, y para todos los dispositivos de cómputo anteriores, no existían recursos disponibles ni una necesidad significativa para generar los elementos de un sistema operativo. Las aplicaciones aprovechan los recursos y características que causaron su inclusión en el equipo sin que quedara un margen significativo, y los conceptos de reutilización de código estaban comenzando con el diseño de las rutinas y bibliotecas en los lenguajes de programación.

Hoy día, las computadoras de propósito particular, basadas en procesadores o microcontroladores que en ocasiones no requieren modos protegidos de operación, pueden seguir modelos similares de desarrollo de sistemas de información para ser aplicados con recursos específicos acordes a sus necesidades, con mínimos requerimientos de compatibilidad y muy poca o nula conectividad. Con esas características es común que carezcan de sistemas operativos, o que lo reduzcan a una biblioteca de rutinas y, a lo sumo, rutinas de atención de interrupciones y programas para atender la comunicación e inicio de los programas de aplicación.

Etapa 1. Computadoras comerciales monoproceso

Con el éxito de las primeras computadoras electromecánicas, el uso de dispositivos electrónicos, como los tubos de vacío y después los transistores, y con los avances

en materia de lenguajes de programación como el ensamblador, los compiladores y las subrutinas, resultó rentable el desarrollo de computadoras comerciales como la UNIVAC o los sistemas 360 de IBM. Estas computadoras se comercializaban junto con los servicios necesarios para el desarrollo de las aplicaciones y los programas, bibliotecas para control de dispositivos de entrada y salida y también programas orientados a cargar los programas y los datos, usualmente de medios como las cintas y tarjetas perforadas, y después cintas magnéticas, para presentar los resultados también en cintas y tarjetas perforadas, hasta llegar a teletipos y discos magnéticos.

Las grandes inversiones en infraestructura para adquirir estos equipos, así como la necesidad de las empresas de generar un flujo de utilidades, trasladó parte de la atención al desarrollo de equipos de entrada y salida progresivamente mejores con énfasis en las capacidades para almacenar la información a procesar mediante diversos tipos de memoria, como la de tambor, núcleos de ferrita, discos duros y, finalmente, la memoria basada en semiconductores. También se han desarrollado dispositivos dedicados a almacenar la información a largo plazo con capacidad para procesarla y extraer los resultados, como las cintas magnéticas o los teletipos para la salida de información y, finalmente, las impresoras. Estos dispositivos pueden reemplazarse con mayor frecuencia y aprovechar las características favorables de nuevos modelos con mucho menor impacto económico y que además implican menos modificaciones a los sistemas de información, lo que los hace fuentes sostenibles de ingresos en los periodos de vida útil de las computadoras.

Es de gran importancia comercial dar soporte a los nuevos dispositivos, por lo que el software dirigido a las aplicaciones orientadas a modelos anteriores es prioritario; por ello, se distribuye con el equipo y se hace llegar a todos los propietarios de servidores del mismo proveedor a fin de facilitar la comercialización de los dispositivos. Sin embargo, esto muy pronto acumula gran cantidad de rutinas, lo que aunado a las bibliotecas orientadas a apoyar el desarrollo de aplicaciones hace que diversos errores comiencen a afectar los sistemas.

A diferencia de las primeras computadoras, el equipo que las diseña y construye no está relacionado de manera directa con el desarrollo de aplicaciones; además, el tipo de aplicaciones que se desarrollan son más variadas y cercanas a las necesidades productivas de las empresas que las adquieren. Las aplicaciones de corte administrativo y las diseñadas para realizar cálculos matemáticos sobresalen, y ninguno de estos problemas es fácilmente representado por el ensamblador o por lenguajes de código ejecutable particulares de las unidades de procesamiento; por ello se generan nuevos lenguajes de programación orientados a los dominios de problemas que se van a atacar, como COBOL o Fortran. Estos lenguajes son elaborados y ofertados por los fabricantes de las

computadoras como una ventaja competitiva al facilitar el desarrollo de las aplicaciones pues representan los primeros intentos de estandarización o certificación de las capacidades de las implementaciones de diversos fabricantes con miras a evitar que las aplicaciones desarrolladas en una plataforma requieran demasiado esfuerzo para ser migradas a otras.

También surgen los programas orientados a procesar los conjuntos de programas que alimentan a la computadora mediante dispositivos de almacenamiento; básicamente preparan el ambiente de ejecución, recuperan el programa siguiente, lo ejecutan, recaban los resultados y los envían a impresión, y además preparan el ambiente para el programa siguiente, con lo que se repite el ciclo. A esto se le denomina procesamiento por lotes, donde los conjuntos de trabajos son recabados por los operadores con anterioridad y, una vez que se ha terminado el procesamiento, se procede a recopilar los resultados y entregarlos a los usuarios.

Al paso de los años, y conforme se mejoran los diseños de las computadoras, los dispositivos para implementar la memoria y la propia construcción de las unidades de procesamiento, se han llegado a obtener computadoras con capacidades muy superiores a las anteriores y, además, tienen menos requerimientos en cuanto a la infraestructura necesaria para su operación.

En esta etapa, el sistema operativo consta sobre todo de los programas para el procesamiento por lotes, las bibliotecas y los lenguajes para soportar el desarrollo de las aplicaciones y el manejo de los dispositivos de entrada y salida. Poco a poco se reúnen colecciones de programas de utilería que se desarrollan tanto por los fabricantes de las computadoras como por los usuarios que las emplean, y se incluyen en las distribuciones que acompañan a los *mainframes* cuando se populariza su uso.

Etapa 2. Multiproceso

Conforme crecen las capacidades de las computadoras, las aplicaciones que en modelos anteriores hubieran ocupado la totalidad de recursos como la memoria, pasan a ocupar solo una fracción de ellos. Además, nuevas técnicas de construcción de computadoras a base de circuitos grabados sobre placas de baquelita y el uso de semiconductores con mayores escalas de integración llevan a capacidades de procesamiento cada vez mayores, cumpliendo incluso con la ley de Moore, la cual describe que dicha capacidad prácticamente se podía duplicar por la mitad del costo cada 18 meses gracias a las técnicas empleadas en la fabricación de nuevos procesadores. Esta tendencia se mantuvo entre 1965 y 2008. Los dispositivos de entrada y salida se diversifican, y aunque siguen mejorando sus capacidades, la velocidad a la que pueden atender las peticiones

es menor que la de los procesadores, por lo que resulta excesivo el impacto en el desempeño del sistema. Para evitar que la CPU se ocupe en la espera de las operaciones de entrada y salida, se implementan esquemas de interrupciones que permiten al procesador continuar la atención de los procesos durante la operación de entrada y salida y que pueda ser notificado cuando estas operaciones terminen para suspender durante un mínimo de tiempo la atención de los procesos. Así, una vez atendida la operación de entrada o de salida se retoman las tareas que requirieron las operaciones en dispositivos de entrada y salida en primer lugar.

Sin embargo, para continuar con el procesamiento se requieren más tareas, además de aquellas que solicitaron la operación, y a menudo deben suspender su labor en espera de que la operación solicitada termine.

Gracias a la reducción del costo y a la disminución de potencia, requerimientos térmicos y espacio que cada procesador tiene, también se comienzan a desarrollar equipos que emplean múltiples procesadores en lo que se conoce como multiprocesamiento. Estas computadoras pueden distribuir la capacidad de procesamiento en una serie de particiones, donde cada una tiene una porción de memoria dedicada a un proceso y que es atendida de manera exclusiva por un procesador, pero resulta deseable buscar esquemas más flexibles que permitan una forma de multitarea sin importar el número de procesadores.

Con la relativa abundancia de recursos disponibles en las computadoras, es posible albergar múltiples programas en la memoria y alternar la atención del procesador para atenderlas con una frecuencia que no solo aproveche los tiempos de bloqueo sino que incluso genere la ilusión de que todos se atienden de forma simultánea. Se inicia el manejo del concepto de procesos para proteger los recursos de la ejecución de cada programa de las acciones de los otros.

La implementación de un ambiente de ejecución de programas que soporte el multiproceso es considerablemente más complejo que los procesos por lotes preexistentes, por lo que varias implementaciones, como Multics, prácticamente fracasan. Las primeras iniciativas que lograron implementar esquemas de multiprocesamiento efectivos fueron cooperativas basadas en desarrollos modestos y graduales, como el de sistema operativo UNIX y el lenguaje de programación C empleados mutuamente en el desarrollo del otro.

Con la acumulación de programas de utilería, controladores, lenguajes y bibliotecas, junto con el soporte a aplicaciones desarrolladas con anterioridad, se tiene una cantidad de software a la que no se puede dar mantenimiento y aplicaciones muy complicados, a fin de que las pruebas a las que son sometidos no puedan ser desarrolladas de manera intuitiva. Llega incluso a suceder con frecuencia que las versiones de

distribución de los paquetes de un sistema operativo generan más errores que los que corrigen. El número de errores en las aplicaciones también tiende a incrementarse en lo que se conoce como la crisis del software. El principal enfoque para solucionar esta tendencia es la generación de nuevas técnicas de análisis, diseño y pruebas de sistemas, con propuestas hechas por investigadores, como Edsger Dijkstra, de modificar el paradigma de programación para que los programas puedan asegurar, mediante técnicas formales, su estabilidad y cumplimiento de requerimientos.

En el frente de los sistemas operativos se observa el desarrollo de nuevos lenguajes de programación orientados a nichos muy variados, desde lenguajes de nivel intermedio dedicados al desarrollo de las prestaciones básicas del propio sistema operativo como C, hasta lenguajes que exploran nuevos paradigmas de programación, como Lisp o Prolog, entre otros.

Se separa el desarrollo de las aplicaciones (incluso aquellas que implementan los mecanismos para emplear las funciones básicas del sistema operativo del desarrollo del núcleo, que debe proporcionar el ambiente de ejecución a las aplicaciones) y se comienza el desarrollo de núcleos con funcionalidades mínimas (conocidos como micro *kernel*) para facilitar la verificación y optimización de este elemento fundamental al margen del cada vez mayor conjunto de software que debe acompañarlo en el paquete del sistema operativo. Además, el desarrollo de los controladores de dispositivos de entrada y salida tratan de ser desarrollados de forma separada, por lo que se implementan estándares para que estos sean capaces de integrarse a la funcionalidad del sistema operativo.

Para soportar la portabilidad de las aplicaciones entre diversos sistemas operativos, se proponen estándares de funciones que el sistema operativo exponga mediante interfaces de programación para el desarrollo de aplicaciones, entre los que se cuenta POSIX.

Etapa 3. Cómputo personal

Una segunda consecuencia de la disponibilidad de procesadores fabricados en masa y de bajo costo es la posibilidad de construir computadoras basadas en estos procesadores a un bajo costo y con capacidades rudimentarias, pero con una potencia considerable. Así surgió una multitud de proyectos aficionados y pequeños fabricantes de arquitecturas peculiares de computadoras de bajo costo orientadas al usuario individual. Originalmente, estos equipos tenían sistemas operativos destinados a proporcionar un lenguaje de programación, además de dispositivos y aplicaciones orientados a fines específicos. De esta manera, Basic surgió en esta etapa como un lenguaje que fuera sencillo de aprender.

El incremento en el número de computadoras en uso hace énfasis en la necesidad de intercambiar información entre ellas, lo que ayuda a popularizar redes de bajo costo y medios de transferencia compartidos como Ethernet mediante cable coaxial y después con cableado sin blindar (UTP5).De manera eventual, las computadoras que integraron mejor el uso de redes de área local a sus sistemas operativos y a sus arquitecturas de hardware tuvieron una ventaja competitiva importante.

Las terminales en sistemas *mainframe*, al no ser producidas en masa, mantuvieron costos relativamente elevados ante la creciente presión de las computadoras personales. Una vez que las redes de área local se esparcieron en casi todas las instituciones, la simulación de la terminal mediante programas que se ejecutan en una computadora personal, conectada mediante la red al *mainframe*, resulta más económica y además da al usuario final una computadora personal para obtener beneficios adicionales. De manera eventual, los sistemas de *mainframe* reemplazan el modelo de terminal por completo en favor del cliente servidor o de algún otro modelo de comunicación distribuida, las cuales revisaremos en el capítulo 7 de *Sistemas distribuidos*. Estas tendencias hacen énfasis en la atención de las operaciones de red y no en el control a bajo nivel de los dispositivos de entrada y salida para los servidores, por lo que la implementación de los protocolos de red también es una preocupación importante de los sistemas operativos para servidores.

Debido a lo anterior, surgió una gran variedad de equipos y fabricantes de computadoras personales. Algunos de ellos lograron ser los proveedores más importantes en sus regiones, pero la falta de compatibilidad y la limitada capacidad de producción de una pequeña compañía restringieron su interés y capacidad de atacar mercados globales. IBM, uno de los principales fabricantes de computadoras y el principal fabricante de software del mundo, incursionó en el mercado diseñando y construyendo un tiraje limitado de computadoras orientadas a ser extensibles, usar productos como los procesadores Intel que se pudieran fabricar en masa y estándares para los puertos y algunos elementos de la arquitectura, todo esto con el fin de lograr una plataforma de cómputo personal que integra con sus propuestas de sistemas de información todo incluido.

Aunque los equipos que diseña y fabrica operan de manera correcta, su costo es mucho más elevado que el de otras propuestas de cómputo personal; además, en sus primeras versiones no proporciona prestaciones superiores a las de la competencia. IBM estimó en su momento que el mercado de estas plataformas personales sería limitado y poco rentable. Esta compañía publicó las especificaciones como lo había hecho antes con las de los puertos de comunicación con lo que favoreció el desarrollo independiente de fabricantes que pudieran implementarlas con menores márgenes de ganancia a fin de abatir su costo para los consumidores. En ese contexto, tampoco compra

a Intel, el fabricante de los procesadores en que basó la arquitectura, ni a Microsoft, la naciente compañía que contrató para desarrollar el sistema operativo rudimentario.

El posterior desarrollo de la plataforma, gracias a la ley de Moore, y el creciente interés en computadoras personales, desde la perspectiva empresarial, hizo que las IBM PC compatibles dominaran después el mercado de las computadoras personales, orientándose en primer lugar a proporcionar un ambiente en el que las operaciones de ofimática tuvieran progresivamente mejores capacidades de multiproceso, además de ofrecer control de dispositivos de entrada y salida, soporte a la red y, finalmente, seguridad.

A la par, la diversidad de fabricantes de plataformas IBM PC compatibles y de dispositivos de entrada y salida crece muy rápido gracias a la respuesta favorable del mercado. Esto ayuda a que los fabricantes de componentes, incluyendo los procesadores, puedan costear la inversión a fin de lograr incrementar la producción para satisfacer ese mercado y abatir los costos, con lo que hoy día se cuenta con dispositivos a costos muy atractivos. La variedad de fabricantes y dispositivos cuyas características buscan representar una alternativa para ganar la preferencia de los usuarios genera problemas para la administración de dispositivos de entrada y salida en el cómputo personal. Esto lo revisaremos en el capítulo 5 de *Administración de dispositivos de entrada y salida*.

Ante el aumento de los controles para limitar el uso de los servidores en las instituciones comerciales o académicas y la popularización de computadoras personales, se comenzó a tratar de implementar los mismos programas y sistemas operativos en todas las plataformas, incluyendo las personales. Aunque el proyecto GNU contaba con una colección de programas de utilería para sistemas operativos UNIX distribuidas como software libre, el desarrollo del núcleo aún estaba orientado a servidores, estaba incompleto y no era compatible con los procesadores populares de cómputo personal. Por ello, Linus Torvalds inició un proyecto de *kernel* simplificado, monolítico y dependiente de la arquitectura 80×86 de Intel de su propia computadora y logró un producto funcional en relativamente poco tiempo, con lo que se inició el desarrollo de Linux, el cual, junto con las utilerías de GNU, generó una serie de variantes de Linux para computadoras personales que sigue desarrollándose y ha llegado a presentar una alternativa viable para aplicaciones comerciales y personales de la que los sistemas operativos comerciales han tomado muchos elementos.

Por su parte, sistemas operativos como Windows y Mac OS se han enfocado a incorporar elementos que hagan más sencillo el uso del sistema para los usuarios con poca capacitación en materia informática, incorporando de manera gradual elementos funcionales una vez que estos se refinan de sistemas operativos para otras plataformas. Esta facilidad de uso y una fuerte inversión en convenios comerciales y publicitarios les han dado a ganar la mayor parte del mercado en computadoras personales.

Etapa 4. Masificación de la Internet

La popularidad de las redes de área local para intercomunicar las computadoras personales y los servidores ayudó a las instituciones y a los individuos a aprovechar de mejor manera la capacidad de sus equipos, y comenzó así el desarrollo de los sistemas distribuidos.

El paso siguiente era conseguir información de sistemas localizados en ubicaciones remotas. Esto se lograba originalmente reutilizando tecnología telefónica mediante acopladores acústicos que pudieran transferir la información en una llamada, es decir, los módems telefónicos, con enlaces satelitales y con sistemas de radiotransmisión o de microondas capaces de establecer enlaces punto a punto. Iniciativas como la de ARPANET, del ejército de Estados Unidos de América, ponen a punto los equipos y programas necesarios para dirigir los paquetes de las redes Ethernet a lo largo de múltiples nodos para llevarlos al destino adecuado en otra red de área local. Servicios como la resolución de nombres (DNS), el ruteo de paquetes de red, el correo electrónico y la transferencia de documentos de hipertexto por el protocolo HTTP que da origen a la World Wide Web se basan en este tipo de redes para satisfacer las necesidades de comunicación de las instituciones que ayudan a desarrollar la tecnología, pero posteriormente se popularizan para el uso empresarial y personal.

La respuesta de las compañías de telecomunicaciones para proporcionar servicios de acceso a Internet a los usuarios domésticos y de pequeñas empresas, primero mediante líneas telefónicas y hoy día mediante fibra óptica y redes de telefonía celular, ha sido muy positiva, pues se ha logrado distribuir los sistemas entre un número considerable de usuarios con capacidad adquisitiva suficiente para soportar una economía donde productos y servicios se pueden comercializar de forma global. Los medios de comunicación han disminuido su costo progresivamente y han llegado incluso a reemplazar la computadora personal (debido al costo que implican las instalaciones que estas requieren y el costo mismo del equipo) por las plataformas móviles o smartphone como el principal dispositivo empleado para acceder a Internet, aprovechando la facilidad de estos dispositivos de acompañar a los usuarios y operar prácticamente en cualquier lugar, incluso sin conexión a Internet, lo que se conoce como cómputo ubicuo y que analizaremos a continuación.

Etapa 5. Cómputo ubicuo

Este tipo de sistemas y aplicaciones requieren que los sistemas operativos se adapten a un conjunto de requerimientos con diferencias importantes. Ubicuo es un término de-

rivado del latín que significa *en todas partes*, y se aplica a los sistemas que cuentan con nodos en operación fuera del acceso sostenido a las instalaciones o servicios, como el acceso a la red, a la potencia eléctrica y a instalaciones convencionales. Fue empleado originalmente en aplicaciones como el monitoreo oceánico para prevención de tsunamis del Deep-ocean Assessment and Reporting of Tsunamis, en el uso de algoritmos de redes autoconfigurables para establecer redes de telefonía celular de emergencia sin usar los proveedores convencionales, como el proyecto Serval, y otras propuestas de telefonía para respuesta a desastres.

Estas nacientes propuestas deben lidiar con las necesidades de un gran número de dispositivos y modelos de mercados poco claros, sin importar el rumbo que tome su desarrollo y su popularidad, pues están ayudando a desarrollar tecnologías necesarias para proseguir la tendencia a incrementar el número de dispositivos de cómputo empleados por usuarios que requieren mantenerse en comunicación de manera económica, oportuna y segura.

Etapa 6. Internet de cosas

Es otro tipo de sistemas con necesidades diferentes que deben ser atacadas por el sistema operativo. En estos, las ventajas que se pueden lograr con un gran número de dispositivos con capacidades de cómputo moderadas que interactúan en redes de área personal, es decir, en torno a una persona, a una casa, a un auto, etc., constituyen uno de los frentes de expansión de las tecnologías de computación actuales. Comenzamos a ver beneficios en este tipo de dispositivos en el control de edificios, orientados a disminuir los costos de operación y el impacto al medio ambiente; en los autos, para que cuenten con servicios de navegación y entretenimientos superiores, e incluso, en los individuos, quienes pueden monitorear ciertos factores que contribuyan a su salud, entre otros servicios.

Este tipo de dispositivos parten de las tecnologías de cómputo ubicuo pero atienden a las necesidades de costo beneficio y seguridad que debe proporcionarse a las aplicaciones y plataformas a fin de poder garantizar que esta multitud de dispositivos no sean un campo fértil para diversos ataques y el uso inapropiado y contrario a los intereses de las personas que los pongan en operación.

Tendencias

En este libro procuramos prestar atención a tres aspectos que, por ser relativamente recientes, no suelen encontrarse en la literatura pero que son de gran importancia para

dirigir el estudio que un profesional del cómputo mexicano en pleno siglo XXI debe considerar.

La profesionalización del cómputo como disciplina es el primer aspecto a considerar en la formación de nuevos ingenieros o profesionales de carreras asociadas. La importancia del software ha crecido de manera constante en las últimas décadas, y la tendencia indica que seguirá accediendo a tareas cada vez más importantes a nivel individual y para las sociedades con acceso a tecnología avanzada. Por lo anterior es necesario responder de manera responsable para garantizar el cumplimiento de las necesidades que la sociedad satisface con esta tecnología. Para este fin se requiere trabajar con solvencia, consistencia y ética dentro de restricciones de costo y tiempo muy estrictas. La única forma confiable de lograrlo es mediante la construcción de una disciplina como punto de partida para el profesional del área, la cual le brinde una plataforma tanto para evaluar sus propias competencias como para superar las expectativas y generar ventajas. Asimismo, apoya también a la industria, la cual podrá construir una imagen para que los consumidores de servicios de ingeniería de cómputo puedan aprender expectativas realistas sobre lo que se debe esperar, así como las virtudes y las limitaciones del gremio.

Originalmente, no parecía ser factible definir un cuerpo estandarizado de conocimientos, habilidades y prácticas, pero al madurar la industria se han detectado diversas prácticas que resulta conveniente replicar y estandarizar. Como cualquier área del conocimiento, la disciplina de la ingeniería en computación deberá estar sujeta a una constante revisión y debe aplicar los principios de racionalidad, escepticismo y rentabilidad para mantener su relevancia. Es precisamente por ello que se impone la necesidad de facilitar el progreso a las nuevas generaciones con currículos claros, que sirvan como una base común para el desarrollo de cada ingeniero sin importar su título.

Desde el punto de vista personal, es vital afrontar la competencia en el mercado laboral globalizado con pleno dominio de las ventajas que cada persona pueda traer al mercado y estar orientadas de manera correcta en relación con las capacidades de la competencia.

El predominio de los dispositivos móviles como medio de acceso a Internet está cambiando la forma en la que se entiende el mercado de las aplicaciones, exige nuevos y mejores mecanismos de distribución y está ampliando el alcance de la tecnología, que aunque aún se encuentra lejos de poder ser accesible para todos, al menos está dejando de ser exclusiva para las grandes empresas y las universidades.

La necesidad de considerar a la seguridad informática como una parte integral del proceso de elaboración del software es el tercer aspecto, el cual está impulsado por la masificación de la Internet, la alta interconectividad de los sistemas y la creciente im-

portancia de los sistemas de información en las tareas fundamentales de la sociedad y en los servicios básicos. Se ha puesto de manifiesto y se ha requerido de un nivel de objetividad y consistencia en la evaluación del problema de la seguridad que está redefiniendo no solo la forma en que se realiza el software, sino la propia percepción que tiene la sociedad acerca de cuestiones como la privacidad, la seguridad y la toma de riesgos de manera informada.

1.3 Clasificación de los sistemas operativos

Hay diversas clasificaciones que se pueden aplicar a los sistemas operativos actuales. Las principales se refieren a la organización de su funcionalidad, al modelo de distribución que siguen y a la forma en que administran los procesos.

Sistemas monolíticos y micro *kernel*

La primera clasificación se realiza de acuerdo a la estrategia que el sistema operativo siga para organizar su funcionalidad. Debido a la gran cantidad de responsabilidades que suele traer, es común que los proyectos de código que conforman un sistema operativo tengan millones de líneas, por lo que es muy conveniente mantener solo la funcionalidad mínima en un núcleo de alta coherencia y que se desarrolle el resto de la funcionalidad con un énfasis en el desacoplamiento de esta aun a costa de complicar la estructura del sistema y los pasos necesarios para diagnosticar y corregir errores.

Los sistemas operativos que por su simplicidad tienden a integrar una buena parte de la funcionalidad del sistema en el *kernel* o núcleo se conocen como monolíticos y resultan convenientes para sistemas especializados y de limitada magnitud por ser más sencillos en su estructura.

Los sistemas micro *kernel* son aquellos que procuran incluir solo el mínimo de funcionalidad en el núcleo y desacoplar el resto de esta incluso a costa de usar mecánicas más complejas.

Propietarios y software libre

La segunda gran distinción se refiere no tanto a la funcionalidad o estructura (de hecho, a menudo encontramos grandes similitudes entre sistemas que siguen modelos de distribución contrarios) sino al modelo de propiedad intelectual, comercialización y distribución que los sistemas siguen. Aunque hay algunos que usan esquemas particulares, existen dos grandes alternativas:

1. El software propietario o comercial busca mantener en secreto los detalles de la implementación del sistema con miras a ganar una ventaja competitiva respecto a otros fabricantes de sistemas operativos y así poder seguir un modelo de distribución comercial basado en licencias de uso con las que los usuarios finales pagarán por usar el sistema durante un periodo determinado sin contar con los derechos para modificar, revisar o conservar el sistema operativo. Este es el modelo que siguen la mayoría de las computadoras con sistemas, como los de Apple o Microsoft.
2. Por otra parte, el software libre retoma las prácticas colaborativas de los sistemas tempranos de cómputo y busca desarrollar de manera cooperativa el sistema operativo. Suele recibir participaciones de personas que son financiadas por empresas privadas pero que pueden aprovechar los avances que se logren en este esquema, ya que el código se publica y se manejan diversos tipos de licencias orientadas a que los usuarios logren revisar, aprovechar y ampliar el software.

Hoy en día, los dos modelos de negocio están tendiendo a converger ya que empresas como Red Hat, MySQL, Sun y otras que de manera tradicional aprovechaban el software libre, están siendo adquiridas por empresas como Oracle, que mantiene los proyectos de software libre como una especie de semillero de ideas para posteriores productos de código propietario.

Según la administración de procesos

Por último, se clasifica a los sistemas operativos según los objetivos que persigue en la administración de procesos.

- **Por lotes (Batch).** Es aprovechado para una administración de procesos rudimentaria. Se emplea hoy día en aquellos sistemas de propósito particular que solo utilizan una aplicación, o un número reducido de ellas, e incluso pueden prescindir del modelo de procesos.
- **Tiempo compartido o interactivo.** Es destinado a los sistemas que habitualmente empleamos los usuarios finales, orientados a mantener una interacción fluida con las interfaces de usuario o para la atención de peticiones a los procesos que ejecutan.
- **Sistemas de tiempo real.** Estos sistemas son responsables de establecer y cumplir compromisos en el tiempo de atención de la CPU para los procesos que ejecutan. Gracias a ello pueden brindar una atención predecible y constante a procesos de control o de multimedios a fin de evitar que las variaciones en el nivel de atención sean perjudiciales en sus aplicaciones.

- **Distribuido.** Estos sistemas dan mayor soporte al desarrollo de arquitecturas de sistemas distribuidos haciendo énfasis en la atención a las operaciones de red, a la automatización de la instalación y a la administración de los sistemas, así como en el soporte a la funcionalidad empleada por los sistemas distribuidos, tales como la invocación remota de métodos o el paso de mensajes.
- **Embebido.** En sistemas que tienen propósitos específicos con dispositivos fabricados a la medida, originalmente se usaban versiones reducidas de otros tipos de sistemas operativos que eran adaptadas a las necesidades particulares del dispositivo y su aplicación. Gracias al progreso del mercado y de los dispositivos, estos sistemas comenzaron a incluir capacidades de cómputo de propósito general; tal es el caso de los sistemas operativos de smartphones como Android e iOS, que sin perder de vista las funciones básicas, como el uso de teléfono, cámara y demás características específicas del dispositivo, permiten también la instalación de aplicaciones de propósito general.

Estas arquitecturas y sus objetivos se revisan en el tema conceptos básicos de planificación de procesos en el capítulo 3 *Administración de procesos*.

EVALUACIÓN

1.1 ¿Qué constituye un sistema operativo y de qué está compuesto?

1.2 ¿Qué objetivo tiene el sistema operativo de cara a las aplicaciones y desarrollo de sistemas?

1.3 ¿Por qué hay diversas definiciones de sistema operativo?

1.4 ¿Productos de ofimática como Microsoft Word son parte del sistema operativo?

1.5 ¿Pueden encontrarse actualmente sistemas de información con capacidad de procesamiento que no cuenten con sistemas operativos? ¿Qué tipo de sistemas son?

1.6 ¿Qué distingue a los sistemas operativos de tiempo real de los interactivos?

Referencias bibliográficas

Brookshear, J. Glenn, *Computer Science, an Overview*, Pearson/Addison-Wesley, 9a ed., 2007.

Silberschatz, Abraham, *et al.*, *Operating Systems Concepts*, John Wiley & Sons, Inc., 7a ed., 2005.

Tanenbaum, Andrew S., *Modern Operating Systems*, Pearson Education International, 3a ed., 2009.

COMPETENCIAS A DESARROLLAR

- El alumno comprenderá la forma como se organizan los componentes de un sistema operativo y las funciones de estos.
- Conocerá la definición y el propósito del núcleo o *kernel* en contraste con el resto del sistema operativo.
- Identificará la importancia de los modos de protección del procesador en la implementación y segregación de las funciones del sistema operativo.
- Conocerá el concepto de las interrupciones y su utilidad como mecanismo de interacción con el hardware.
- Conocerá el concepto de la planificación de procesos realizada por el scheduler.
- Conocerá los aspectos básicos de la organización de sistemas operativos importantes al momento de escribir este libro.

Generalidades de sistemas operativos

2

¿QUÉ SABES...?

1. ¿Cuáles de las funciones que acostumbramos usar en nuestras computadoras son implementadas en el sistema operativo y cuáles en las aplicaciones que instalamos en él?
2. El sistema operativo es una de las inversiones iniciales más importantes en software al establecer cualquier sistema de información, debe proporcionar una cantidad importante de funcionalidad a las aplicaciones que dependen de él. ¿Cómo organiza el sistema operativo esta funcionalidad?
3. ¿Cómo está organizado el sistema operativo que empleas de forma regular en tu computadora personal o en tu smartphone? ¿Consideras que puedan organizarse de la misma manera a pesar de sus diferencias?

2.1 Estructura de un sistema operativo

Los sistemas operativos actuales deben lidiar con un gran número de requerimientos a fin de brindar las facilidades necesarias para dar soporte al desarrollo de aplicaciones y a la operación de los sistemas. Esto hace que los productos comercializados o distribuidos de forma gratuita no solo incluyan los elementos esenciales del sistema operativo sino que lo enriquezcan con diversos elementos de apoyo, como los siguientes:

- **Núcleo del sistema operativo.** Funcionalidad básica del sistema operativo que proporciona el ambiente en el que operarán las aplicaciones. Suele incluir la administración de procesos y memoria, el control de algunos dispositivos de entrada y salida, así como parte de la administración de la red y del sistema de archivos. Se ejecuta en el modo protegido en las plataformas de la CPU que lo soportan, con lo que tiene acceso pleno a los recursos y lleva a cabo las operaciones que las aplicaciones de usuario no pueden realizar por ejecutarse en un modo con privilegios reducidos conocido como modo usuario. El núcleo de sistema operativo también es conocido como *kernel.*
- **Intérprete de comandos.** Mecanismo incluido en las primeras computadoras comerciales. Permiten a los usuarios interactuar con el sistema operativo para realizar consultas y ejecutar programas sin necesidad de tener un operador que manipule medios donde se envíen los comandos o programas. Los lotes de tarjetas perfora-

das y las cintas empleadas como el principal medio de interacción del usuario con la computadora hoy día están siendo reemplazados por las interfaces gráficas.

- **Interfaz gráfica.** Ambiente gráfico que el usuario puede apreciar en la pantalla donde se tienen elementos estandarizados a fin de representar aquellos que se emplean para interactuar con el sistema. Según cada sistema operativo o implementación de interfaz gráfica, se establece una serie de estándares con el objetivo de presentar elementos que resulten familiares para los usuarios en todas las aplicaciones, de modo que se facilite el aprendizaje del uso de cada una de las aplicaciones en virtud de que el conocimiento adquirido sobre cómo manipular y qué elementos usar sean comunes a todas las aplicaciones.
- **Mecanismo de distribución de aplicaciones.** Dispositivo que ayuda a disminuir las fallas de seguridad ocasionadas por *malware*, es decir, por programas desarrollados para atacar los sistemas de información, los cuales se discuten en el capítulo 8 de *Seguridad informática*. Se ha optado por incluir un mecanismo de distribución de aplicaciones desarrolladas por otras entidades para someterlas a verificaciones mínimas y generar así un canal de distribución confiable y accesible para los usuarios.
- **Utilerías de sistema.** Serie de programas que, aunque no forman parte del núcleo del sistema operativo, resultan necesarios para que los usuarios y las aplicaciones puedan utilizar sus servicios y realizar operaciones básicas de mantenimiento, administración y monitoreo.
- **Programas de aplicación básicos.** Al reconocer las tareas comunes que habrán de realizarse con una computadora, es común incluir en el paquete o en el mecanismo de distribución algunas aplicaciones con funcionalidad básica de ofimática, como la edición de documentos o la manipulación de imágenes, o atender tareas propias del dispositivo como el uso de la cámara fotográfica, así como realizar llamadas desde un smartphone. Estas son meramente aplicaciones, y pueden ser reemplazadas o retiradas sin perjuicio del sistema operativo como tal; sin embargo, suelen distribuirse junto con este para brindar a los usuarios un ambiente funcional de partida.
- **Servicios básicos.** Servicios que proveen funcionalidad a los sistemas y usuarios, y que suelen incluir los sistemas de archivos, el control de los dispositivos de entrada y salida, la implementación de las capas de funcionalidad requeridas para controlar la red, el manejo de errores, la asignación y protección de recursos, el seguimiento del consumo de recursos por las aplicaciones y las facilidades para protección, autenticación y control de acceso a dichos recursos.

Núcleo o *kernel* del sistema operativo

Como mencionamos en la estructura, el núcleo es el elemento básico del sistema operativo que proporciona el ambiente necesario para que las demás aplicaciones y servicios puedan operar.

El núcleo se encarga de la administración de procesos y de las asignaciones de recursos como la memoria y la atención de la CPU a estos. En los sistemas que los soportan, también es responsable de la administración de los hilos de ejecución.

Asimismo, proporciona los mecanismos necesarios para generar la comunicación entre los procesos, las alarmas y el manejo de interrupciones que requieren elementos compartidos entre procesos y, por tanto, no deben dejarse bajo el control exclusivo de uno de ellos. Estos mecanismos serán estudiados en el capítulo 3, *Administración de procesos.*

Mediante interfaces de programación, el *kernel* permite que las aplicaciones soliciten operaciones sin necesidad de conocer los detalles de los recursos que atenderán dichas peticiones. Esto ayuda a lograr la independencia física entre las aplicaciones y las características del equipo y establece un punto de control para implementar las protecciones y permisos correspondientes. Dichos mecanismos serán revisados en el capítulo 5 de *Administración de dispositivos de entrada y salida*.

Interrupciones

Parte de la funcionalidad que el sistema requiere para reaccionar ante eventos detectados por el hardware. Se maneja mediante una serie de rutinas que deben ser ejecutadas cuando los elementos conectados al equipo transmiten señales por las líneas de interrupción (IRQ). Como estas funciones deben ejecutarse con privilegios de *kernel* y no deben ser alteradas libremente por las aplicaciones, entonces son definidas al arranque del sistema por el BIOS y después quedan al cuidado del sistema operativo, el cual será el único capaz de redefinir su funcionalidad y regular de manera preferente el acceso a utilizarlas. En el capítulo 5 de *Control de dispositivos de entrada y salida* se revisará este tema con detalle.

Planificador de la CPU (scheduler)

Sistema encargado de administrar la atención de la CPU. Hace que el tiempo del procesador sea un recurso a repartir entre los procesos, por lo que es importante que los sistemas operativos que soporten multiprocesamiento incluyan mecanismos para realizar una distribución acorde con los objetivos del sistema. A esta distribución se le denomina **planificación del procesador**, y la revisaremos con detalle en el capítulo 3, *Administración de procesos*.

2.2 Arquitecturas de sistemas operativos de referencia

Linux

Linux es una implementación popular del núcleo del sistema operativo UNIX, generado debido a la adopción del modelo de distribución de software libre bajo una licencia GNU y que se ha orientado a soportar la plataforma de computadoras personales IBM compatibles, a las que se han agregado otras arquitecturas gracias a su popularidad. Fue creado en 1991 por Linux Torvalds, que se mantiene involucrado en su desarrollo y es propietario de la marca registrada "LINUX". Para usar el sistema operativo, diversos grupos toman una versión del *kernel* Linux, la optimizan para las plataformas y aplicaciones objetivo, la incluyen en un paquete con herramientas y aplicaciones de soporte y se genera una distribución de Linux que los usuarios finales pueden instalar y utilizar en sus equipos, por un precio muy bajo.

Las primeras versiones del *kernel* Linux consistían de un solo módulo, se compilaban y ligaban en un único programa en cuya ejecución compartía todos los recursos del modo protegido del procesador. Para continuar su ejecución es necesario que el código del *kernel* se desarrolle de forma modular, con una estructura que facilite la independencia de la plataforma específica que lo ejecutará y resulte económico en el consumo de memoria y en la pérdida de desempeño por consumo de la CPU en la propia implementación del núcleo. Para satisfacer estas necesidades Linux se organiza en módulos.

Los módulos son código que puede ligarse de forma dinámica al núcleo durante la ejecución del sistema. Así, el desarrollo de cada módulo se realiza de forma independiente y una implementación del sistema operativo cuenta solo con los módulos que requiere. Durante la ejecución, los módulos pueden cargarse y descargarse a memoria según las actividades que se estén realizando, con lo cual se evita consumir memoria en la función no requerida. Cabe aclarar que aunque hay una penalización por el desempeño durante la carga y descarga de los módulos, al estar en uso tienen la ventaja de permanecer ligados al *kernel* del sistema operativo, con lo que se evita el costo de interfaces de programación más complejas.

Algunos de los módulos habituales son administración de procesos, administración de memoria y administración de dispositivos de E/S, que dependen mutuamente entre sí. Otros módulos corresponden a la implementación de los protocolos de red, los controladores de dispositivos de video y de red que requieren una atención privilegiada, el sistema virtual de archivos, los controladores de los dispositivos de bloque que lo soportan y los controladores genéricos de dispositivos de terminales. A estos se suman otros, lo cual dependerá de la configuración que cada distribución realice.

Como los demás UNIX, Linux proporciona un ambiente de ejecución a las aplicaciones en tres niveles: el *kernel,* la sesión y el proceso de usuario.

El ***kernel*** en que se tiene acceso pleno a los recursos del sistema usa el modo protegido de los procesadores y es utilizado solo por el código del núcleo del sistema operativo. Como se revisará con más detalle en el capítulo 3, *Administración de procesos*, el *kernel* es el encargado principal de controlar los procesos junto con la base de la administración de los demás recursos de la computadora. Las solicitudes que se realizan al *kernel* controlan el cambio en el nivel de privilegios al pasar por una serie de llamadas al sistema, que son rutinas desarrolladas de manera específica para ser usadas por las aplicaciones y las sesiones.

La **sesión**, ya sea gráfica o mediante un intérprete de comandos, o *shell* (línea de comandos), es el proceso generado para interactuar con el usuario considerando los privilegios de este, y proporciona elementos importantes para iniciar, administrar y apoyar la ejecución de los procesos de usuario.

El **proceso de usuario** es la instancia de ejecución de las aplicaciones y sistemas de información que los usuarios utilizan para satisfacer sus necesidades puntuales. En general se emplean las interfaces de programación de aplicaciones (API) proporcionadas por el propio sistema operativo, incluyéndolas en las aplicaciones, con lo que se logra reutilizar la implementación de funcionalidad común (véase figura 2.1).

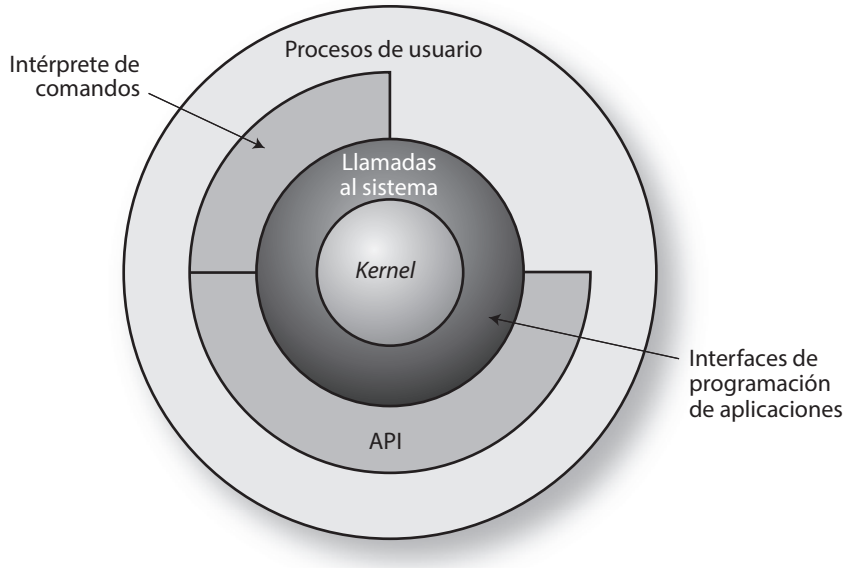


Figura 2.1 Niveles de ejecución de UNIX.

ACTIVIDAD PROPUESTA

En acción

Si quieres escuchar la exposición de los niveles de ejecución y la forma en la que estos se relacionan con la finalidad de desarrollar aplicaciones útiles para los usuarios, consulta la película explicativa que se encuentra en los archivos de AT&T publicados en YouTube en el siguiente URL:

`https://www.youtube.com/watch?v=tc4ROCJYbm0`

Windows

El desarrollo de Windows ha tenido diferentes vertientes para diversos mercados. Se ha colocado como uno de los exponentes más firmes del modelo de distribución del software comercial, pues presenta una funcionalidad similar a la de otros sistemas operativos para ajustarse a las necesidades del mercado. Las versiones actuales han evolucionado a partir de Windows NT, y por ello comparten la arquitectura de este.

Algunos aspectos importantes que deben tenerse en cuenta es la estrecha relación de Windows con los procesadores Intel y su modo protegido, lo que no lo excluye de la necesidad de ser útil para muchas de las arquitecturas que emplean estos procesadores y para una variedad de modelos que los propios procesadores Intel presentan, razón por la cual la portabilidad siempre ha sido una preocupación para Microsoft.

Por otra parte, Windows implementa el multiprocesamiento simétrico, de modo que los procesos tanto de *kernel* como de usuario pueden ser atendidos por cualquier procesador. Se han implementado optimizaciones para evitar cambios de contexto innecesarios. Los cambios de contexto se realizan cuando se suspende la atención de un proceso y se debe dedicar tiempo de la CPU a preparar la ejecución del proceso siguiente que habrá de atenderse. Se busca agrupar los turnos de ejecución de los procesos en un solo procesador cuando resulte adecuado, pero si se requiere cargar el estado de un proceso se elige entre todos los procesadores disponibles.

En cuanto a su arquitectura, el sistema operativo se divide, en primer lugar, en la funcionalidad que operará en modo usuario y la funcionalidad del núcleo que lo hará en el modo protegido del procesador. El núcleo de Windows es monolítico e integra toda la funcionalidad del modo protegido en una sola entidad que comparte la memoria que emplea. Para mitigar los riesgos de que alguno de los componentes del *kernel* perturbe la operación de los demás, implementa mecanismos con los que logra controlar la modificación de los elementos del *kernel,* como Patchguard y la firma del código de los elementos del *kernel*.

Los elementos que se ejecutan en el modo usuario no tienen acceso directo a la memoria o a los recursos del modo protegido y aprovechan los elementos de administración de memoria que previenen el acceso a los rangos asignados a otros procesos, por lo que se cuenta con más herramientas para controlar las actividades que realizan y garantizar la estabilidad del sistema.

Tanto en el núcleo como en el modo usuario se tienen diversos elementos que conviene revisar, como se muestra en la figura 2.2.

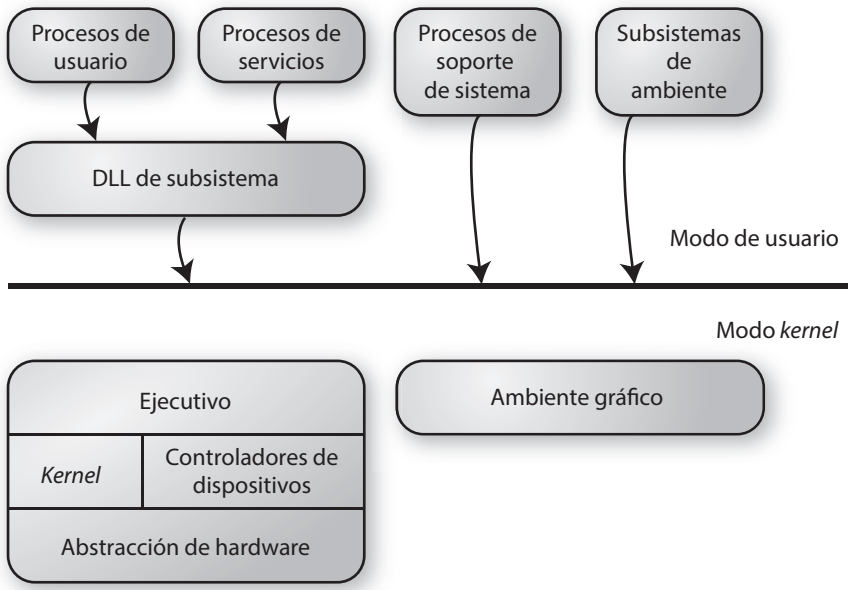


Figura 2.2 Diagrama de componentes en el modo protegido y de usuario.

En el modo usuario se tienen cuatro tipos básicos de procesos:

- **Procesos de usuario.** Procesos de las aplicaciones desarrolladas para satisfacer las necesidades específicas de los usuarios. Pueden ser de diversos tipos: las aplicaciones desarrolladas exclusivamente para Windows (ya sea en sus versiones de 32 o 64 bits), las aplicaciones destinadas a versiones anteriores de Windows 3.1 de 16 bits y las desarrolladas para MS-DOS o que empleen la implementación de POSIX de Microsoft de 32 o 64 bits. De acuerdo con el modelo del procesador y la versión de Windows (16 o 32 bits), se limita el tipo de aplicaciones que se pueden ejecutar en una computadora en particular.
- **Procesos de servicios.** Son aquellos supervisados por el administrador de servicios. Estos servicios pueden ejecutarse de forma independiente a las sesiones de

usuario; muestra de ello son algunas aplicaciones que utilizan servicios asociados a los privilegios de algún usuario para poder establecer procesos que actúen como servidores (por ejemplo, de base de datos, web, etc.).

- **Procesos de soporte de sistema.** Son iniciados durante la carga del sistema y no por el administrador de servicios. Establecen los procesos originales para administrar sesiones, así como la autenticidad de sesión de usuarios necesarios para operar el sistema.
- **Subsistemas de ambiente.** Implementan el ambiente en el que las aplicaciones se ejecutan, proporcionando los servicios que requieren del sistema operativo. Desde NT hasta Windows 2000 se incluían ambientes de Windows, POSIX y OS/2. Hoy día se cuenta con una implementación nueva de POSIX llamado SUA y el ambiente de Windows únicamente.

Las **DLLs de subsistema** son las bibliotecas de carga dinámica que serán ligadas y utilizadas por las aplicaciones para abstraer las peticiones que dirijan a los servicios nativos del sistema operativo Windows. Estas refuerzan la independencia lógica al presentar las API de programación uniforme que adecuan las peticiones a las características específicas presentes en cada computadora.

El núcleo tiene los siguientes componentes:

- **Ejecutivo.** Contiene los servicios básicos del sistema operativo, como la administración de la memoria y la de procesos, los mecanismos de control de acceso, el control de dispositivos de entrada y salida, el manejo de la red y la comunicación entre procesos.
- ***Kernel.*** Esta es la parte del núcleo que se encarga de las funciones de bajo nivel, como la planificación de hilos, el manejo de interrupciones y el origen de excepciones. Cuenta con las rutinas apropiadas para cada arquitectura que las capas superiores integran en las API uniformes.
- **Controladores de dispositivos.** Incluyen tanto a los controladores específicos para dispositivos de hardware como aquellos controladores que implementan la funcionalidad de dispositivos, como el sistema de archivos y los controladores de red que no están sujetos a un dispositivo en particular.
- **Abstracción de hardware.** Conocida como HAL (Hardware Abstraction Layer), contiene el código que aísla al núcleo y los controladores de dispositivos de las peculiaridades de cada arquitectura de hardware en la que opera el sistema operativo.

Para el **ambiente gráfico**, el *kernel* implementa las funciones de la interfaz gráfica de usuario que se encargan de la presentación de las ventanas, los controles de la interfaz estándar de usuarios y la presentación de gráficos.

Es importante recalcar que Windows se basa en una arquitectura que separa la funcionalidad en múltiples capas, donde las capas inferiores se orientan a aprovechar las características del hardware en el que opera y las capas superiores dedican su funcionalidad a la estandarización de la atención de las solicitudes de usuario.

Android

El sistema operativo Android es un ejemplo de las aplicaciones del *kernel* Linux a aplicaciones específicas, en este caso para dispositivos móviles. Parte de un *kernel* de Linux optimizado y con los módulos necesarios para interactuar con el hardware específico de una tablet o smartphone, el cual tiende a mantenerse constante, en contraste con el de una computadora personal. En el caso de Android, se hizo hincapié en las consideraciones de seguridad y protección necesaria de la información de las aplicaciones, ya que se conocía de antemano que los dispositivos que usen este sistema operativo manejarán información personal sensible y recibirían aplicaciones muy variadas, incluyendo algunas potencialmente maliciosas. Además, es necesario prestar atención a problemas específicos como la administración de la potencia y la optimización del uso de procesador y memoria por ser más limitados que los de otras plataformas.

Las capas y componentes típicos en un sistema Android se muestran en la figura 2.3.

En la parte superior de la figura 2.3 se sitúa la capa en que se ejecutan todas las aplicaciones (implementaciones preestablecidas por Android para atender operaciones como realizar una llamada telefónica, mandar un SMS, abrir un URL en el navegador, etc.). Estos servicios aprovechan las facilidades que el sistema operativo brinda a sus aplicaciones a fin de facilitar su desarrollo y optimizar el uso de recursos.

El ambiente de trabajo para las aplicaciones permite a estas dedicar todo su esfuerzo a la implementación de funciones específicas de cada aplicación reusando la proporcionada por el sistema operativo para manejar los elementos gráficos o controlar el ciclo de vida de la aplicación, entre otros muchos servicios. Lo anterior favorece el apego de las aplicaciones a los estándares de desarrollo e implementación de la plataforma y reduce la inversión necesaria en la mayoría de las aplicaciones.

En la capa siguiente se tiene el Android runtime. Este ilustra la implementación de la máquina virtual Dalvik que proporciona la API básica de programación Java, con lo que se permite que las aplicaciones Android se desarrollen en este lenguaje a pesar de que no se trata de una JVM completa. Cada aplicación genera una instancia de proceso desde el punto de vista del *kernel,* y dicho proceso usa como ambiente de ejecución de las capacidades de la máquina virtual Dalvik y la invocación a una serie de rutinas que proporcionan acceso a las funciones del *kernel*. Además, en esta capa se cuenta con

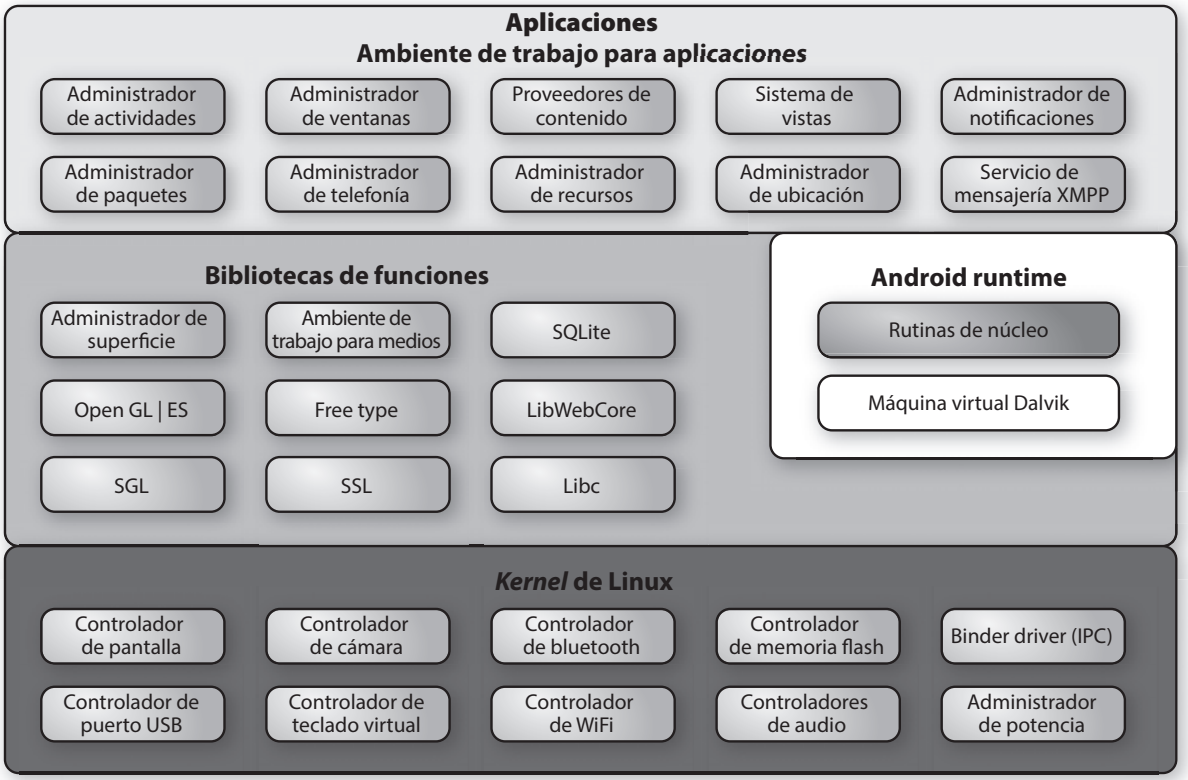


Figura 2.3 Estructura del sistema operativo Android.

una serie de bibliotecas de funciones orientadas a la implementación de tareas específicas, como el manejo de información en una base de datos relacional en memoria SQLite, las rutinas para reproducir archivos de audio y video o interpretar diversos formatos de imágenes, entre otras tareas.

En la capa más cercana al hardware está el núcleo optimizado de Linux para la plataforma particular, en la que se incluyen módulos que permiten manejar los dispositivos incluidos con el dispositivo en particular. Esto facilita trabajar la potencia, reducir la velocidad del procesador y administrar la batería. En esta capa también se encuentran los servicios habituales en Linux de administración de memoria, procesos e integración del control de dispositivos de entrada y salida, aunque las capas superiores ocultan esta funcionalidad de las aplicaciones convencionales.

Mac OS X

El núcleo de Mac OS X se conoce como Darwin, y, junto con los servicios que dan acceso a la funcionalidad del *kernel*, están basados en BSD UNIX. También cuenta con

una arquitectura modular mediante la cual integra funcionalidad para soportar el manejo de red y el soporte a múltiples sistemas de archivos.

El control de las arquitecturas en las cuales se utilizan los productos de Apple, y en particular el sistema operativo, permite implementar modos de control de la memoria particulares para la arquitectura, de modo que cada proceso tiene su propio espacio de direcciones protegido por el hardware y controlado por el sistema operativo.

Darwin también proporciona facilidades para atender problemas que revisaremos en el capítulo 3, *Administración de procesos*, como:

- Multitarea cooperativa y con multiproceso.
- Multiprocesamiento simétrico con soporte a multihilos.
- Soporte a aplicaciones de tiempo real.

Los componentes principales se distribuyen en cuatro capas (véase figura 2.4):

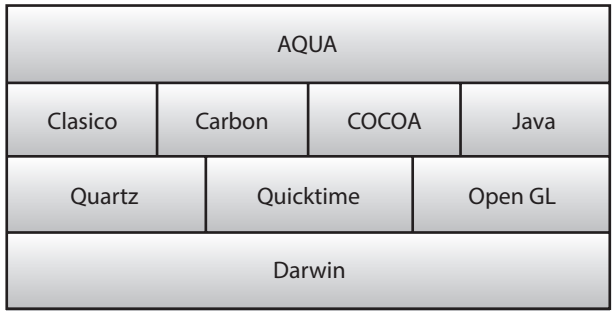


Figura 2.4 Estructura del sistema operativo iOS, que se basa en el *Kernel* Darwin al igual que Mac OS X.

A continuación se presenta una breve explicación de lo que representa cada una de las etapas mencionadas en la figura 2.4.

- **AQUA.** Interfaz de usuario que se utiliza en las aplicaciones junto con los elementos gráficos para las pantallas.

Mac OS X. Soporta múltiples ambientes operativos para sus aplicaciones:

- **Clasico.** Soporte para las aplicaciones desarrolladas para Mac OS 9.1 en un espacio de memoria protegida para apoyar a la mayoría de las aplicaciones desarrolladas en sistemas anteriores junto con las aplicaciones modernas.
- **Carbon.** Ambiente nativo de Mac OS X basado en las API de Mac OS con capacidades más avanzadas. Hace énfasis en el desarrollo mediante el uso de técnicas de programación orientada a objetos y API de programación para Objective C y Java.
- **COCOA.** Usa herramientas de desarrollo acelerado (RAD) y técnicas de programación orientada a objetos, tanto en Java como en Objective C.

- **Java.** Mac OS X incluye una implementación completa de Java 2 Standard Edition (J2SE) con su máquina virtual Hotspot, la cual permite aprovechar mejor los recursos del sistema operativo y optimizar el desempeño junto con adiciones para acceder a los elementos de la interfaz de usuario de Aqua mediante la API de Swing como elementos nativos del ambiente gráfico, lo que mejora el rendimiento de la interfaz gráfica, junto con el tratamiento de archivos JAR como bibliotecas de carga dinámica compartidas de parte del sistema operativo. Esto facilita el ahorro de memoria al usar memoria compartida en bibliotecas de uso común para múltiples aplicaciones Java.

En lo referente al manejo del sistema gráfico, se combinan tres tecnologías:

- **Quartz.** Servidor de ventanas ligero, de alto rendimiento para gráficas de dos dimensiones. Está orientado a brindar las capacidades avanzadas de las tarjetas aceleradoras de video mediante las API que den a las aplicaciones una versión con independencia física y acceso a funciones avanzadas de manejo de ventanas y gráficos 2D.
- **Open GL.** Implementación optimizada del estándar de la industria de tarjetas aceleradoras de video para la elaboración de gráficos en 3D. Este es el estándar con mayor aceptación entre los fabricantes de hardware de tarjetas de video, por lo que constituye una interfaz de programación independiente del hardware específico de los equipos que proporciona acceso a las funciones avanzadas del hardware que están destinadas a la elaboración de aplicaciones que exploten las capacidades del equipo para elaborar gráficos en 3D.
- **Quicktime.** Tecnología para soporte de multimedia con la funcionalidad necesaria para manipular archivos de imagen, audio, video, texto e incluso realidad virtual inmersiva con la capacidad de transmisión en tiempo real con soporte a múltiples formatos de archivos de medios.

2.3 Modos de protección

La capacidad de tener múltiples programas en ejecución para garantizar que los recursos de memoria y procesamiento asignados a cada uno de ellos serán respetados por los demás requiere de mecanismos que limiten las acciones que los programas puedan realizar e impidan que logren invadir rangos de memoria y recursos que no les corresponden.

Para ello muchos procesadores, como los de la familia 80×86 y posteriores de Intel, implementan diversos modos de protección en los que diversas capacidades y opera-

ciones del procesador son limitadas. En el caso de los procesadores Intel, hay cuatro modos de protección y otros orientados al soporte de aplicaciones desarrolladas para modelos anteriores. Sin embargo, aprovechar los modos de protección requiere de una cercana colaboración con el sistema operativo para realizar los cambios de modo del procesador de forma eficiente y oportuna, y estos deben evitar la dependencia de modelos específicos de procesador para mantener los costos de portabilidad a otras plataformas bajo control.

Por esta razón, la mayoría de los sistemas operativos solo usa dos modos de protección para el procesador:

- El modo de protección 0, o modo supervisor, es reservado para el sistema operativo y permite realizar operaciones de administración de memoria para activar los rangos correspondientes a las diversas aplicaciones, o para emplear rangos de memoria asignados a dispositivos o de uso exclusivo del sistema operativo. Este modo tiene acceso pleno a las capacidades del equipo, por lo que debe ser utilizado solo por la función que se revise de manera exhaustiva y que forme parte del sistema operativo. En el capítulo 4, *Administración de memoria,* hablaremos más de los cambios de contexto, y en el capítulo 5, *Administración de dispositivos de entrada y salida,* detallaremos el uso de interfaces de programación empleadas para que los programas de las aplicaciones que se ejecutan sin estos privilegios puedan acceder a los recursos administrados por el sistema operativo mediante el escalamiento de privilegios.
- El modo de protección 3 es utilizado por las aplicaciones. En este modo, el procesador limita el acceso a la memoria a rangos de direcciones, o segmentos, que fueron preparados en el modo supervisor. Las operaciones que una aplicación intente rebasar los límites de estos segmentos de memoria son atrapadas por el procesador y generan errores de acceso a memoria. Con ello se protege de manera efectiva el acceso de un programa en ejecución a la memoria que no le corresponde.

Los modos de protección 1 y 2 pueden emplearse para que los controladores de dispositivos tengan acceso a determinadas áreas de memoria y recursos sin necesidad de tener los privilegios del modo supervisor; sin embargo, la sobrecarga asociada a los cambios de modo del procesador y la dependencia física a los modelos de procesador con cuatro modos ha llevado a los fabricantes de sistemas operativos a no utilizarlos (véase figura 2.5).

Gracias al uso de máquinas virtuales se han popularizado los programas hipervisores que coordinan la ejecución de múltiples sistemas operativos en un solo equipo, alternando la atención de los procesadores y protegiendo rangos de memoria y recursos para que sean utilizados solo por una máquina virtual.

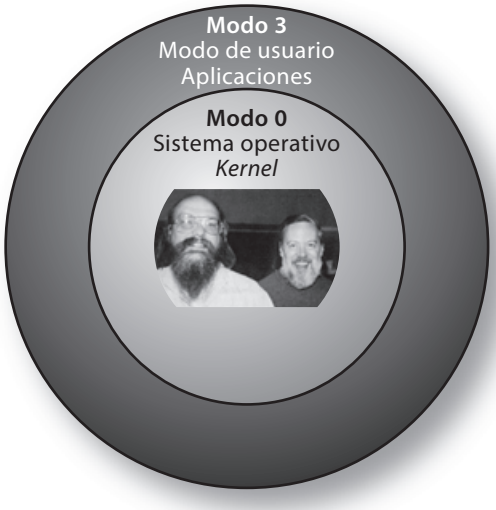


Figura 2.5 Modos de protección en UNIX.

A fin de proteger las operaciones que realiza el hipervisor para administrar los recursos de las máquinas virtuales, se ha generado un modo de operación adicional en el procesador que a menudo se conoce como modo de protección -1.

2.4 Funciones y características

Los sistemas evolucionan como respuesta a las necesidades de los usuarios, a las características del hardware y en particular a los requerimientos del desarrollo de aplicaciones. Esto ha ocasionado que en vez de partir de una serie de objetivos, sus características hayan crecido de manera paulatina. Sin embargo, hoy día se ha establecido una serie de funciones que se espera sean atendidas por el sistema operativo y que determinan muchas de las características que deben tener.

En general, los sistemas operativos actuales obedecen a tres objetivos generales:

- **Soporte a las aplicaciones.** Consiste en facilitar el desarrollo, el uso y la administración de las aplicaciones que los usuarios de los sistemas de información requieren.
- **Eficiencia.** Busca apoyar a las aplicaciones y a los usuarios para que empleen los recursos disponibles de manera eficiente.
- **Evolución.** Se basa en permitir el desarrollo, las pruebas y la instalación de nuevas funciones en los sistemas de información con poco impacto en el resto de los componentes presentes.

2.5 Mecanismos de comunicación entre procesos (IPC)

Una de las características que tienen los sistemas operativos orientados a multiproceso, como UNIX, es la funcionalidad, la cual consiste en facilitar que los procesos intercambien información y coordinen sus actividades entre ellos, con recursos protegidos para que no puedan emplearse por otros procesos no relacionados. No es posible usar simples variables o rangos de memoria con ese fin, ya que el uso prácticamente al mismo tiempo llevaría a comportamientos inconsistentes y a menudo perjudiciales para la operación del sistema de información.

En el capítulo 3, *Administración de procesos*, en el tema Principios generales de concurrencia, tenemos algunos de los mecanismos básicos tales como:

- **Sleep y Wakeup.** Estos permiten a un proceso bloquearse para dejar de recibir atención de la CPU y despertar a otro proceso para que de nueva cuenta reciba atención luego de haberse bloqueado.
- **Semáforos, mutex y monitores.** Estos dispositivos permiten establecer momentos en la ejecución de los procesos en los que se coordine el funcionamiento, de modo que todos los participantes puedan esperar un evento en particular o evitar el uso de un recurso por más de un proceso a la vez.
- **Paso de señales.** Permiten interrumpir la ejecución de un proceso desde otro proceso que cumple determinadas condiciones, ya sea para terminar su ejecución o para que esa señal sea atendida por una función específica del proceso que recibe la señal y pueda entonces continuar su ejecución normal.
- **Alarmas.** Mecanismos similares al paso de señales, pero que son enviadas por un proceso a sí mismo en un momento futuro.

EVALUACIÓN

1. ¿Cómo se relaciona el *kernel* de un sistema operativo con el modo de protección de la CPU?
2. ¿Qué componente del sistema operativo busca estandarizar la interfaz de usuario, y en cuáles sistemas operativos ha recibido más atención este componente?
3. ¿Los programas de aplicación son parte del sistema operativo? ¿Se pueden distribuir con estos?
4. ¿Qué objetivo distingue a los sistemas operativos de plataformas móviles como Android de los de cómputo personal como Linux?

Referencias bibliográficas

Bovet, Daniel P., Cesati, Marco, *Understanding The Linux Kernel,* 3a edición, 2005.

Mac OS X, An Overview for Developers, Apple Computer Inc., 2002.

Russinovich, Mark, *et al.*, *Windows Internals.* Microsoft Press, 6a edición, 2012.

Silberschatz, Abraham, *et al.*, *Operating Systems Concepts,* John Wiley & Sons Inc., 7a edición, 2005.

Tanenbaum, Andrew S., *Modern Operating Systems,* Pearson Education International, 3a edición, 2009.

Referencias electrónicas

386 protected mode, Jack Gansle,http://www.ganssle.com/articles/aprot1.htm

Modo protegido, Wikipedia:http://es.wikipedia.org/wiki/Modo_protegido

Protected Mode, Wikipedia: http://en.wikipedia.org/wiki/Protected_mode

Protection Ring, Wikipedia: http://en.wikipedia.org/wiki/Protection_ring

COMPETENCIAS A DESARROLLAR

- El alumno comprenderá la relación entre la operación del procesador de la computadora y el modelo de procesos, como una abstracción de esta orientada al manejo de la multitarea y la protección de recursos.
- Identificará las características de un proceso, un hilo y un programa, y la diferencia entre estos.
- Conocerá las características de los procesadores modernos orientadas a sostener el multiproceso, como el modo protegido, y las operaciones para salvar y recuperar su estado mediante el uso de la memoria RAM.
- Identificará y aplicará los elementos básicos de las interfaces de programación del estándar POSIX para el desarrollo de programas que implementen funcionalidad de multiprocesamiento.
- Conocerá los principales problemas que surgen al desarrollar aplicaciones que aprovechen el multiprocesamiento y las soluciones disponibles para estos.
- Se familiarizará con las principales herramientas proporcionadas por el sistema operativo para soportar la sincronización y la comunicación entre procesos, como semáforos, mutex, monitores, memoria compartida, alarmas, señales y pipes.
- Aplicará las principales variantes de la planificación de procesos según el propósito de los sistemas de información que soportan.

Administración de procesos

3

¿QUÉ SABES...?

1. ¿Qué actividades realizas cuando trabajas con computadoras?
2. En equipos de dos o tres personas, elaboren una lista de las actividades que realizan cuando están frente a una computadora o utilizan algún dispositivo electrónico que tenga capacidades de procesamiento de datos (smartphone, tablet, wearable, computadora de un automóvil, control automatizado de algún electrodoméstico, etc.). Anoten el mayor número de acciones que les sea posible recordar en cinco minutos, procurando que sean lo más variadas entre sí. Si en algún momento se les dificulta recordar, piensen en algunas personas que trabajan frente a una computadora o máquina compleja y moderna, anotando todo aquello que recuerden.
3. Luego de que hayan transcurrido los cinco minutos y tengan su lista terminada, respondan las siguientes preguntas respecto a las acciones descritas en ella:
   - Al realizar las actividades de la lista, ¿las computadoras o los dispositivos electrónicos desempeñaban un papel activo o pasivo? Consideren que la toma de decisiones y la recuperación de información son actividades, incluso si no se aprecia el movimiento de componentes mecánicos.
   - Con respecto a equipos de cómputo compartidos, por ejemplo en una escuela o un Internet comercial, ¿cuántas personas usan los sistemas que se utilizan para realizar las actividades que identificaron?
   - ¿Cuántas operaciones creen que se realizan por día o por hora en los ejemplos que identificaron?
   - ¿Dónde se encuentra físicamente el equipo de cómputo? (A menudo los equipos se encuentran ocultos. Estime la distancia probable hasta el centro de cómputo, si en el caso que detectó se emplea uno.)
   - Para los usuarios que usan los sistemas involucrados, ¿qué volumen de información aproximado (expresado en MB o GB) se requiere almacenar?, ¿qué volumen de transferencia de datos (expresado en Mbps) implica la cantidad de operaciones realizadas por hora?
   - Con respecto a las actividades que llevan a cabo en su lista en equipos de cómputo propios, ¿cuántas tareas se pueden realizar cuando sus enlaces a Internet y a otras computadoras no funcionan?
   - ¿Qué aplicaciones de las que identificaron tienen características radicalmente diferentes de las demás?
   - ¿Pueden elaborar una clasificación de las aplicaciones por sus características generales?
4. Ahora tomen en cuenta la aplicación de cómputo que usan con más frecuencia.
   - ¿Cuáles son las características de interacción entre el usuario y la aplicación?
   - Cuando deben esperar algún resultado, ¿qué muestra la aplicación?, ¿qué suelen hacer durante el tiempo de espera?
   - Desde que comenzaron a utilizar la aplicación en cuestión, ¿han notado que el desempeño mejora o empeora? Es decir, ¿ha cambiado el tiempo de espera y la velocidad de respuesta?

- Con los conocimientos de programación que tienen en la actualidad, ¿podrían desarrollar esta aplicación o una similar?
- ¿Qué partes de la funcionalidad de la aplicación comprenden mejor y cuáles les resultan más complejas con base en su experiencia como desarrolladores?
- ¿Qué estructura de programación creen que tienen los programas que conforman la aplicación?, ¿qué elementos son comunes a otras aplicaciones y qué elementos pertenecen al sistema operativo?

3.1 Introducción

Para iniciar el estudio de las tareas que realiza un sistema operativo, lo primero es comprender la administración de los recursos de procesamiento de un equipo de cómputo, ya sea una computadora personal (de escritorio o laptop) o un dispositivo móvil (tableta, teléfono inteligente, etc.), los cuales cambian día con día debido a los adelantos tan acelerados que experimenta la industria de la informática.

Los sistemas operativos modernos tienen a su disposición múltiples núcleos de procesamiento de tipos diversos, como los procesadores gráficos (GPU-Graphic Processor Unit), dedicados a operaciones matemáticas (ALU-Arithmetic and Logic Unit), de procesamiento general y microcontroladores integrados con los dispositivos de entrada y salida, placa base e incluso sistemas de alimentación de potencia, los que permiten tener mayor capacidad de procesamiento que la que se tenía hace algunos años; no obstante, para poder aprovechar al máximo dicha capacidad, debemos vigilar que la carga de trabajo a ejecutar se distribuya conforme a los objetivos del sistema y las características específicas de cada tipo de procesador. Asimismo, estas aplicaciones requieren una apariencia de simplicidad, al tiempo que son atendidas por los recursos de procesamiento.

Por otra parte, para lograr la independencia física y de uso de los diferentes equipos y dispositivos electrónicos, no debe requerirse conocer las características del procesador que ha de ejecutar las aplicaciones, la frecuencia, e incluso si estas serán atendidas en un momento determinado para su correcta operación. De este modo, cuando se desarrollan aplicaciones solo se debe percibir un entorno de procesamiento idealizado en el que se puedan generar todas las instancias de ejecución que se requieran y donde se cuente con diversas formas de comunicar, coordinar y controlar la atención que se recibe de parte del procesador.

3.2 Modelo de procesos

Para poder manejar la complejidad inherente de los esquemas de administración del procesador se ha generalizado el uso de una abstracción conocida como **modelo de procesos**, que se basa en el siguiente concepto de proceso, propio del ámbito de los sistemas operativos.

Proceso

Representación de todos los elementos que constituyen una instancia de ejecución de un programa.

Los elementos de un proceso son los siguientes:

- Una identificación única para el proceso, a menudo numérica.
- El comportamiento definido por el programa.
- El estado, que consiste en la memoria asignada y en los valores de los registros del procesador.

Asimismo, hay dos conceptos relacionados que debemos diferenciar:

- **Hilo de ejecución (Thread).** Estado de una secuencia de ejecución de instrucciones dentro de un proceso. Todos los procesos tienen al menos un estado con el apuntador a la instrucción y el estado de los registros del procesador, pero en muchos sistemas operativos se permite que un proceso tenga múltiples hilos de ejecución con estados independientes que comparten los recursos del proceso, como memoria, código del programa, archivos, dispositivos de entrada y salida asignados, etc. (En algunos libros, *thread* se traduce como *hebra*.)
- **Programa.** Constituye una secuencia de instrucciones ordenadas en un lenguaje regular, que implementa un algoritmo que cumple un propósito particular.

  Los programas, a diferencia de los procesos y los hilos, representan información estática o pasiva que es utilizada por los procesos para realizar su ejecución, y por ello no debe confundirse con hilos o proceso, a pesar de su relación con estos.

  Cuando se ejecuta un programa, además de cargar sucesivamente las instrucciones que lo componen en el registro de instrucciones del procesador y permitir que a lo largo de los ciclos de reloj este haga las transiciones de estado correspondientes, se requiere asignar segmentos de la memoria del equipo para almacenar los datos que el programa ha de procesar y llevar el control de los dispositivos de entrada y salida que se reserven para el programa y la interacción con estos. A menudo, también se dedica especial atención al uso que el proceso da a los archivos.

En los ambientes de multiprogramación actuales se requiere proteger estos recursos de la intervención de otros procesos que puedan estar en ejecución y que pudieran tratar de afectar, incluso de manera accidental, el uso de dichos recursos.

Es importante resaltar que aunque la gran mayoría del procesamiento que realiza una computadora o dispositivo está a cargo de diversos procesos, no todas las operaciones que realizan los procesadores están cubiertas por este modelo. Principalmente, los dispositivos de entrada y de salida pueden incluir considerables capacidades de procesamiento que operen al margen de este modelo, en virtud de servir a propósitos particulares y por ello no restan validez al modelo.

Modo de usuario y modo de sistema (o protegido)

El **modo de sistema**, también conocido como ***Kernel* Mode** o **modo protegido**, se utiliza para las operaciones que deben realizarse mediante un software que se considera confiable como parte del sistema operativo, y que puede manipular recursos del sistema a un nivel que las aplicaciones normales de usuario no deben realizar.

En tanto, las aplicaciones más comunes operan en **modo de usuario**, debido a que estas tienen segmentos de memoria asignada protegidos, con el fin de que otras aplicaciones no puedan modificar los valores. Asimismo, cuando es necesario llevar a cabo operaciones privilegiadas, como acceder a dispositivos de entrada, de salida, o ambos, o a segmentos de memoria distintos, se opera en modo de sistema, ya que en estos casos se requiere solicitar peticiones a rutinas de sistema que verifiquen los privilegios del proceso, y en caso de contar con autorización puedan realizarse las operaciones solicitadas. En caso contrario, el software que opera en modo de sistema funciona en un único espacio de memoria y no requiere verificar sus privilegios para acceder a cualquier elemento presente, por lo que deben tomarse precauciones para evitar intervenir en los datos de otros programas que también operen en modo de sistema.

En el caso particular de Windows, en todas las versiones de este sistema operativo los controladores de dispositivos de E/S, los contro-

Importante

♥ **Docente.** De manera tradicional, los cursos de programación dejan de lado los aspectos gráficos y de concurrencia en los proyectos de clase, por lo que usarlos resulta muy novedoso, a pesar de que son una parte fundamental de la mayoría de los sistemas computacionales y debe dedicarse tiempo para conocerlos y familiarizarse con estos.

♠ **Arquitecto.** Al diseñar y crear programas que usen multiprocesamiento, el arquitecto debe integrar a sus pruebas alguna forma de manipular los tiempos de retardo, así como algunas otras herramientas y estrategias de experimentación, a fin de poder observar la concurrencia en acción.

♦ **Líder.** El líder debe evitar confiar en pruebas que no detecten errores como evidencia de que la implementación es correcta, ya que muchos problemas pueden presentarse de forma aleatoria. En proyectos complejos es importante vigilar que las herramientas y librerías estén preparadas para la programación concurrente.

♥ **Usuario.** Una revisión de las recomendaciones de las interfaces de usuario y los mecanismos para implementarlas no solo puede aportar un valioso contexto al tema del multipro-

cesamiento, sino que también permite tomar conciencia acerca de lo difundido e indispensable que resulta en las aplicaciones.

ladores de sistemas de archivos, las rutinas de apoyo para atención de interrupciones y el núcleo del sistema operativo operan en el modo de sistema. En cambio, en Linux existe un núcleo del sistema, o rutinas de atención de interrupciones, y mediante el uso de llamadas a sistema los procesos de usuario pueden acceder a los privilegios administrativos, que suelen conocerse como `root`.

Por su parte, los sistemas operativos embebidos, como los que se usan en equipos móviles (como Android), tienen núcleos que operan tanto en modo de sistema como en modo de usuario, siguiendo el mismo esquema que Linux. Por esta razón, a menudo se menciona que Android tiene un *kernel* basado en Linux, con lo que se alude a que toma la misma funcionalidad para las operaciones en modo de sistema y los procesos para el control de la interfaz gráfica, y para la operación de las aplicaciones se mantiene en un modo de usuario, sujeto a los controles definidos por la planificación de procesos.

Los sistemas operativos de tiempo real también respetan esta jerarquía de modo de usuario y de modo de sistema, aunque en versiones recientes del *kernel* se han hecho refinamientos para que las operaciones que se realizan en modo de sistema puedan ser interrumpidas, por ejemplo, por nuevos eventos en dispositivos de entrada y de salida que levanten interrupciones, así como para que puedan encolar las tareas pendientes por cada procesador, de manera que recuperen el estado de procesamiento en el que estaban sin efectos adversos (*preemptive*) y sean capaces de atender una operación de modo de sistema por procesador de modo simultáneo (capacidad conocida como *Reentrant*).

ACTIVIDAD PROPUESTA

En acción

1. Para que conozcas más acerca de las capacidades de procesamiento de los dispositivos de entrada y de salida, realiza una investigación acerca de una tecnología en particular (por ejemplo, las tarjetas aceleradoras de video orientadas a juegos o al diseño gráfico incorporan grandes capacidades de multiprocesamiento que pueden servir al procesamiento general del equipo con protocolos como OpenGL, o el uso de procesamiento en los dispositivos de almacenamiento masivo que permiten asegurar la información en la norma SMART).
2. Con la información que obtuviste responde las siguientes preguntas:
   - ¿Cómo se mide la capacidad de procesamiento de uno de estos dispositivos?
   - ¿Cómo se compara con la capacidad de los procesadores principales del equipo?

- ¿Cómo se realiza un programa que aproveche esa capacidad?

3. En equipo, recopilen información acerca de las posibles aplicaciones de uso cotidiano que podrían mejorarse, con el fin de que puedan ser aprovechadas al máximo. Comenten en clase, con sus compañeros, si sería rentable en términos de costo-beneficio.

Ciclo de vida de un proceso

El ciclo de vida de un proceso consta de tres etapas principales: la creación, la ejecución y la terminación de los procesos.

Creación de un proceso

Para tener un proceso activo es necesario:

1. Reservar los recursos que le serán asignados.
2. Asignar los elementos en la planificación de procesos para atenderlo.
3. Preparar un estado inicial en el procesador y la memoria que incluya cargar el código del programa a la memoria antes de llevar a cabo las instrucciones del programa.

Por lo general, para iniciar un proceso existen cuatro mecanismos:

1. Como parte del arranque del sistema operativo.
2. A partir de un proceso existente, mediante una llamada al sistema.
3. Con una petición del usuario mediante un intérprete de comandos o una interfaz gráfica.
4. Como parte del procesamiento por lotes en un sistema que lo realice de manera automática.

ACTIVIDAD PROPUESTA

En acción

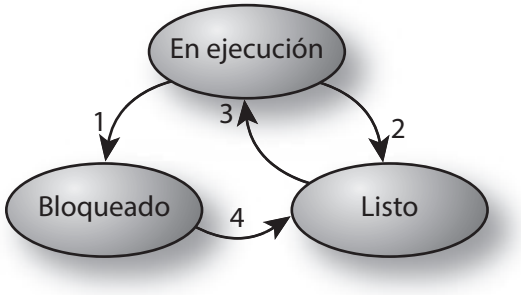
1. Identifica los diversos procesos de diferentes sistemas operativos que sean iniciados con los mecanismos antes mencionados (por ejemplo, Time-Share, UNIX y embebido como Android).
2. Busca información acerca de las facilidades correspondientes en tiempo real, como RTLinux y Wind River Linux.

3. En sistemas de tiempo compartido o interactivos, como UNIX, ¿cuáles son los procesos que se inician directamente al arrancar el SO?, ¿cómo lo contrasta con los procesos que inician mediante `init.d`?
4. Ejecuta un programa desde su intérprete de comandos y desde la interfaz gráfica de su sistema operativo, y responde las siguientes preguntas:
   - ¿Cómo te informa el sistema operativo que el proceso ha iniciado?
   - ¿Cuál es el identificador del proceso?
   - ¿Cómo puedes solicitar la terminación del proceso directamente al sistema operativo?
5. Para iniciar un proceso desde otro podemos usar en nuestro programa la llamada al sistema `fork()` del sistema operativo UNIX. Realiza un programa que genere una copia de sí mismo con esta función y observa el comportamiento de ambos.
6. Ejecutar un archivo de ejecución por lote en un sistema Time share solo es una forma más de iniciar un proceso desde otro proceso; no obstante, algunas aplicaciones han sido desarrolladas para llenar el nicho de automatizar la ejecución de procesos. Revisa la funcionalidad que ofrece `crond` y analiza cómo se distingue esta funcionalidad del procesamiento en un sistema Batch.

Estados durante la ejecución

El objetivo general de la administración de procesos es la realización de las tareas para las que los programas fueron desarrollados. Ahora bien, cuando los recursos son escasos, se puede atender un proceso a la vez, dedicando todos los recursos del equipo y minimizando el consumo de estos en tareas administrativas; sin embargo, debido a que las computadoras más recientes tienen relativa abundancia de recursos, es posible satisfacer las preferencias de interfaces de usuario más amigables, que dependen de tener múltiples procesos operando y que suspenden y renudan su ejecución varias veces por segundo generando la ilusión para el usuario de que operan de forma simultánea; este modo de operación por turnos se conoce como concurrente. Esto genera una carga de trabajo que no ayuda de manera directa a la realización de las tareas (proceso conocido como *overhead*); debemos tratar de minimizarlo al implementar la planificación de procesos.

Para atender múltiples procesos con un procesador, o varios, debemos establecer turnos de ejecución que duren un tiempo determinado, los cuales permitan que la atención del procesador se alterne entre los diversos programas que deben ser atendidos. Esta habilidad de atender múltiples programas en ejecución se conoce en general como **multiprogramación**, y en particular como **Time-Share**, ya que comparte el tiempo en el que la CPU atiende los diversos procesos.



1. El proceso se bloquea por una operación de E/S.
2. La planificación da el turno a otro proceso.
3. La planificación da el turno a este proceso.
4. La operación de E/S termina.

Figura 3.1 Esquema de las transiciones válidas.

Cuando un proceso está preparado para ser atendido por el procesador se dice que está "**listo**" para ejecución, cuando está siendo atendido se dice que está "**en ejecución**", y cuando no espera recibir atención del procesador se dice que está "**bloqueado**". Más adelante se tratan con mayor detalle estos tres estados.

Es importante destacar que un proceso no recibe atención del procesador cuando no puede continuar con la realización de sus instrucciones, debido a que se encuentra en espera de una situación, evento, instrucción u orden que no está bajo su control directo; por ejemplo, puede estar en espera de un evento en uno de los dispositivos de entrada, de salida, o ambos, como el hecho de que una unidad de disco termine de cargar información desde un archivo con los datos que habrán de procesarse. Asimismo, un proceso en espera también puede bloquearse hasta que otros procesos terminen de realizar algunas acciones que son necesarias para que el proceso bloqueado progrese. De lo contrario, si dicho programa recibiera atención del procesador antes de que los otros procesos necesarios logren el punto requerido, se impediría que el procesador atendiera a los procesos indispensables para su ejecución y la espera solo sería más larga o, en el peor de los casos, se produciría un error que provocaría un bloqueo definitivo del procesador. Cuando los procesos deben comunicarse entre sí para organizar su operación en el tiempo, como en el caso de los bloqueos, se dice que estos requieren una "**sincronización de procesos**", por lo que el sistema operativo proporciona una serie de mecanismos para comunicar los procesos e implementar las acciones necesarias para soportar diversos casos; es decir, este es el responsable de dicha sincronización de procesos.

Terminación de la ejecución

Algo muy importante que debe tomarse en cuenta en este punto es que todos los algoritmos que se diseñen deben considerar las condiciones necesarias para que se con-

cluya la tarea en ejecución. Esto significa que los procesos deben terminar en algún momento, por lo que debe vigilarse que los recursos del equipo sean recuperados para atender las necesidades de otros procesos, a fin de que el sistema operativo pueda seguir operando durante periodos prolongados.

Lo anterior denota que los programas deben considerar los pasos y las instrucciones necesarias y pertinentes para liberar todos los recursos que reservan, en especial las estructuras de datos y los buffers que soportan el manejo de archivos y los segmentos de memoria que se reservan durante la ejecución del proceso. En este mismo sentido, se puede decir que algunos recursos pueden ser manejados de manera directa por el sistema operativo o por la máquina virtual (como la máquina virtual Java, que proporciona ambientes de ejecución para aplicaciones desarrolladas para estas plataformas), e incluso por las interfaces de programación, para liberar o simplificar el trabajo de programación, por lo que se recomienda vigilar con sumo cuidado el uso de los recursos luego de la terminación de los procesos durante la etapa de pruebas de los sistemas, con el fin de evitar omisiones que puedan comprometer la estabilidad del sistema. Las condiciones usuales por las que un proceso debe terminar son:

- **Salida normal.** Ocurre cuando el propio algoritmo del programa considera que debe concluirse la ejecución.
- **Por un error crítico.** Sucede cuando el programa detecta una situación que le impide seguir con su operación normal, y debe proceder a ejecutar la funcionalidad que le permita recuperar todos los recursos posibles y terminar su ejecución con el menor impacto negativo posible al sistema.
- **Por una condición de excepción.** Ocurre cuando, ante diversas situaciones inesperadas por el algoritmo, el proceso procede a terminar, incluso si existe un procedimiento de recuperación.
- **Recibir una señal de otro proceso.** Ocurre cuando algunos procesos tienen autoridad para enviar señales a otros procesos; el comportamiento básico al recibir una de estas señales es ejecutar una función asignada a ese tipo de señal y luego terminar el proceso.

ACTIVIDAD PROPUESTA ▶

En acción

Todos los mecanismos de terminación contemplan que los programas implementen todos los pasos necesarios para liberar los recursos que se reservaron antes de ser terminados. Pero si un progra-

ma no libera todos los recursos que reservó se habla de "fuga de recursos"; una de las formas más comunes es la **fuga de memoria**.

1. Investiga la definición y diferentes ejemplos de fugas de memoria o fugas de conexión a base de datos.
2. Con base en la información que recabaste mediante tu investigación, realiza lo que se te pide a continuación.

   ¿Cómo afectan estos problemas al usuario final?

   - Cita un caso donde ese tipo de efectos negativos (fugas de memoria o fugas de conexión) se presenten en la operación cotidiana del software que usas como apoyo en tus clases diarias.
3. Recopilen los ejemplos de todo el grupo y comenten acerca del nivel de calidad de los diversos productos usados y su impacto en la operación.

Implementación de procesos

La administración de procesos requiere de diversos mecanismos para mantener bajo control la información de todos los procesos presentes en el sistema durante su operación, así como para almacenar aquella información que representa el estado de los procesos que se mantienen inactivos, con el fin de poder cargarla y ejecutarla cuando se le asigne su turno de ejecución.

Por lo regular, esta información se almacena en una estructura de datos, donde cada proceso recibe un registro y en la que se mantiene un conjunto de información de los aspectos que el sistema operativo debe controlar para cada proceso. Por ejemplo, con base en los campos que maneja la API `procps`, en su estructura `proc_tse` tiene lo que refiere la tabla 3.1.

Tabla 3.1 Tabla de procesos

Campo	Contenido
`tid`	Task ID, identificación del hilo de ejecución según el estándar POSIX.
`ppid`	Identificador del proceso padre (proceso que a su vez creó al proceso que se está describiendo).
`state`	Estado en el que se encuentra el proceso; por ejemplo, "S" (Sleep) cuando está bloqueado.
`utime`	Tiempo de atención de la CPU en modo de usuario dedicado al proceso.
`stime`	Tiempo de atención de la CPU en modo de sistema (modo protegido) al proceso.
`cutime`	Tiempo acumulado de atención en modo de usuario del proceso y todos los hijos terminados.

(*Continúa*)

Tabla 3.1 Tabla de procesos

Campo	Contenido
cstime	Tiempo acumulado de atención en modo de sistema del proceso y todos los hijos terminados.
start_time	Tiempo de inicio del proceso.
señales	Se tienen diversos arreglos para registrar las señales que reciben el hilo y el proceso.
codigo	Rango de direcciones de memoria en las que se encuentra el código del programa.
stack	Rango de direcciones de memoria en las que se almacena el stack y el cursor.
Apuntador de instrucción	Apuntador a la siguiente instrucción que será procesada por la CPU.
wchan	Dirección del canal de espera del núcleo por el cual el proceso está bloqueado.
Prioridad	Indicador de la preferencia del proceso para la planeación.
Nivel de Nice	Indicador numérico de modificador de prioridad (Nice) actual del proceso.
rtprio	Prioridad de tiempo real.
sched	Tipo de planificación (tiempo real o normal).
rss	Tamaño del conjunto de páginas que habrá de mantenerse en memoria para el proceso.
alarma	Información de la siguiente alarma pendiente del proceso.
size	Total de páginas de memoria asignadas al proceso.
resident	Número de páginas residente (no enviadas a memoria virtual).
share	Número de páginas compartidas usadas por el proceso.
environ	Vector de variables de ambiente.
cmdline	Vector de línea de comandos.
pgrp	Identificador del grupo de procesos.
session	Identificador de la sesión en la que se creó el proceso.
nlwp	Número de hilos del proceso.
euid, egid	Identificadores efectivos de usuario y de grupo, respectivamente.
ruid, rgid	Identificadores reales de usuario y de grupo, respectivamente.
suid, sgid	Identificadores originales de usuario y de grupo, respectivamente.
fuid, fgid	Identificadores para usar el sistema de archivos de usuario y de grupo, respectivamente.
exit_signal	Código de terminación del proceso.
processor	Identificador que atendió por última vez al proceso o que lo está atendiendo.

Ahora bien, es muy importante destacar aquí que el control de las fechas y horas para el manejo de procesos en el sistema operativo se realiza mediante un contador que se encuentra ligado con un cristal de cuarzo que vibra en una frecuencia que es un

múltiplo casi exacto de un hertz (una vez por segundo), tradicionalmente 32 768 Hz. De este modo, resulta muy sencillo calcular el número de segundos que han transcurrido al dividir el valor de dicho contador entre esa frecuencia. Esto se realiza al cargar un valor que represente la fecha actual y medir los segundos transcurridos desde una fecha predefinida hasta la fecha actual, siempre y cuando el cristal y el contador se mantengan en operación. Incluso cuando apagamos el equipo, es posible mantener un registro del paso de tiempo como un reloj con muy buena precisión mediante el uso de una pila. El registro que se usa para la medición del tiempo se conoce como `Time`; ahora bien, cuando en la tabla de procesos se hace referencia al **Tiempo**, en realidad nos referimos al valor que tenía el registro en un momento en particular o a la variación en el valor del registro `Time`.

3.3 Principios generales de concurrencia

En primer lugar, podemos decir que un procesador constituye una máquina de estados que puede realizar una sola operación (transición de estado) a cada ciclo de reloj. En este sentido, los programas en lenguajes imperativos, como el Lenguaje C, dependen de esta secuencia de operación para estructurar su lógica; asimismo, incluso la mayoría de los algoritmos reconocen la necesidad de una secuencia de operaciones en su representación. No obstante, no es extraño que al iniciar el aprendizaje de las técnicas de programación, la gran mayoría de las personas que se inicia en la informática considere solo una secuencia de operaciones al diseñar sus programas.

Sin embargo, desde hace algún tiempo las computadoras han tenido la capacidad de atender múltiples transiciones de estados de diferentes procesos, ya sea en virtud de tener auténticamente múltiples procesadores o de simular la existencia de estos al alternar turnos de ejecución muchas veces por segundo. A esto debemos añadir los requerimientos de las aplicaciones, centrados en interfaces de usuario gráficas e interactivas, el uso de una infraestructura de comunicaciones de alta velocidad, así como la existencia de múltiples y variados dispositivos de entrada y de salida y en general de computadoras de propósito general. Esto hace imperativo que la gran mayoría de las aplicaciones soporten al menos algunas de las funciones de multipro-

Importante

♠ **Arquitecto.** La información de los procesos que mantiene el sistema operativo y, en su caso, la máquina virtual en la que desarrollamos nuestras aplicaciones sobre los procesos son de gran importancia para comprender el cuidado que debe tenerse en el manejo de los recursos que se asignen a estos; por ejemplo, si se almacena información acerca de los archivos abiertos, podemos esperar algún apoyo del SO para salvaguardar la integridad de la información, pues de lo contrario, como en Windows, será necesario revisar con sumo cuidado los mecanismos a cargo de las aplicaciones necesarios para evitar comprometer la información.

♥ **Docente.** Como se puede observar, la tabla de procesos ilustra las tareas que debe atender el administrador de procesos; por tanto, es importante lograr establecer estas relaciones entre los datos y las funciones con el fin de brindarles un contexto y una realidad concreta, respectivamente.

♣ **Usuario.** Durante la operación de los sistemas hay diversas herramientas que nos permiten obtener información de la tabla de procesos, con el objeto de vigilar el comportamiento de nuestras aplicaciones, identificar los datos y las herra-

mientas disponibles en nuestro ambiente de trabajo particular, lo que resulta de gran utilidad para mantener en operación nuestros sistemas.

cesamiento, incluso si el programador aún no ha estudiado las implicaciones y los principios que lo rigen.

Así, algunas definiciones que serán de utilidad para facilitar una breve revisión de los principios de la concurrencia son las que se exponen a continuación.

- **Concurrencia.** Capacidad de un sistema para realizar transiciones de estado aparentemente simultáneas mediante la rápida alternancia de atención de los procesos ejecutándose en un procesador. También contempla la capacidad de dichos procesos de comunicarse e interactuar entre sí.
- **Multiprocesamiento.** Capacidad que posee un sistema para asignar tareas o aprovechar más de un procesador en la atención de sus procesos.
- **Serialización.** En el contexto de la concurrencia, se dice de las operaciones que se ordenan de forma seriada deben realizarse en una secuencia estricta, por lo que no pueden ser concurrentes.
- **Bloqueo.** Estado de un proceso en el que este deja de recibir atención de la CPU, como se describió antes.
- **Señal.** Mecanismo de notificación entre procesos mediante el cual se avisa a la planificación de procesos el hecho de que un proceso en particular debe recibir un código de operación; por lo general, este se representa por un número entero. Esto significa que si el proceso está bloqueado, deja de estarlo y retorna a esperar turno de atención por parte de la CPU, para luego proceder a ejecutar el comportamiento por defecto asociado a la señal recibida o por una rutina de su propio programa que haya registrado como de atención a la señal (*handler*). Por defecto, se sabe que diversas señales tienen comportamientos distintos; por ejemplo, son ignoradas o terminan la ejecución del proceso.
- **Alarma.** Tipo de señal con la que un proceso puede enviarse a sí mismo una notificación después de un periodo de espera.
- **Condición de carrera.** Comportamiento de un sistema por el cual el estado final de una operación depende del momento en que se reciben diversas entradas, bajo circunstancias en las que la sincronía de estas entradas no puede predecirse de forma práctica. Esto genera variaciones impredecibles en la salida que a menudo constituyen errores en la operación y pueden corromper la información que se maneja. Esta también se conoce como *race condition*.
- **Colisión.** Evento, también conocido como estado, en el que se pierde o corrompe la información de un sistema debido a una condición de carrera.

El uso de concurrencia en nuestros sistemas nos permite resolver algunos problemas de formas accesibles, los cuales sería muy difícil solucionar mediante operaciones secuenciales.

Para ilustrar los principios generales de concurrencia, lo más conveniente es utilizar ejemplos de problemas habituales y la forma en que se pueden resolver al usar las facilidades del multiprocesamiento.

Bloqueos

Uno de los primeros problemas que se beneficiaron del multiprocesamiento fueron las interacciones con los usuarios a través de mecanismos interactivos. Por ejemplo, cuando se requiere esperar a que ocurra un evento de entrada, de salida, o ambos, el simple hecho de que el usuario oprima una tecla implica que el sistema tiene que elegir entre iniciar un ciclo de espera en el que a cada vuelta se verifica si el usuario ha oprimido una tecla o bloquear el proceso en espera del evento de E/S, con lo que se pierde la capacidad de hacer otras acciones hasta que se produzca el evento. En este caso, ambas opciones tienen importantes efectos negativos, ya que consumen atención de la CPU de forma poco provechosa o dejan de atender las tareas del proceso para, por ejemplo, desplegar información necesaria.

Al usar la concurrencia, podemos usar dos procesos o dos hilos de ejecución; en este caso, uno de los hilos se bloquea en espera de que ocurra el evento que se está ejecutando, mientras que el otro continúa con las tareas que se deben realizar durante la espera. Una vez concluido u ocurrido el evento en espera, incluso podemos finalizar uno de los hilos o procesos para economizar recursos en el sistema.

ACTIVIDAD PROPUESTA

En acción

Podemos observar el comportamiento de un programa desarrollado para aprovechar el multiprocesamiento empleando las interfaces de programación del estándar POSIX del sistema operativo. Estas tienen la ventaja de ser comunes a muchos sistemas operativos e influencian las implementaciones posteriores de los lenguajes de alto nivel que brindan acceso a esta funcionalidad.

Usaremos la programación en C para ambientes Linux para los ejemplos, ya que estas son cercanas a las API del sistema operativo y pueden prescindir de otro tipo de consideraciones que pueden añadir líneas de código y complicaciones a nuestros programas que no están relacionadas con el tema a tratar.

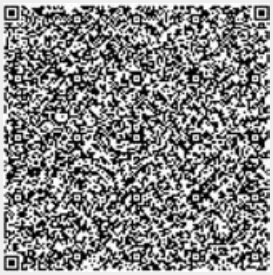
Comencemos con un programa sencillo que realice una tarea (por ejemplo, incrementar un contador), pero que en vez de detenerse al llegar a un valor determinado termine cuando el usuario oprima una tecla. El hilo de ejecución que atiende el contador es más eficiente y sencillo si se dedica exclusivamente a atender el ciclo que incremente dicho contador, y podemos dejar la función de capturar la entrada por parte del usuario a un segundo hilo, que también resulte más sencillo por atender exclusivamente esta tarea. De hecho, este segundo hilo puede ser el hilo original de nuestro proceso, conocido como "hilo principal".

El código del programa es el siguiente:

```
/*El siguiente programa inicia un hilo secundario que incrementa
  continuamente el valor de un contador, entre tanto, el hilo
  principal se bloquea esperando una entrada por parte del usuario en
  la consola y una vez que la recibe imprime el valor que alcanzó el
  contador y termina el proceso.
  Para compilar y ligar, no olvide usar el parámetro -pthread de gcc
  Daniel Sol Llaven 2015-03-31
*/

#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>

long contador;

void incrementaContador() {
     while(1) contador++; //Loop infinito para incrementar el contador
}

int main() {
      //Estructuras de datos usados por el hilo secundario
      pthread_t id_h; //Identificador del hilo
      int val_ret; //Valor de retorno de la creación
      void *arg=NULL; //Argumentos, no los hay para incrementaContador
      contador=0;
      //Creación del hilo con atributos por defecto
      val_ret = pthread_create(&id_h, NULL, &incrementaContador, arg);
      if(val_ret) {
            // Recibe código de error en la creación del hilo
            printf("Error en la creación del hilo: %d\n", val_ret);
            exit(1); //Terminamos con código de error.
      }
      printf("Se creó hilo secundario\n");
      //El hilo principal prosigue, espera un carácter en la entrada
      //estándar y al recibirlo procede a terminar el proceso
      getchar();
      printf("El contador llegó al valor: %ld\n", contador);
```

```
        exit(0); //Terminar el proceso termina todos los hilos secundarios
}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.



Nótese cómo el código contenido entre las etiquetas (o comentarios) "**Estructuras de datos usados por el hilo secundario**" y "**El hilo principal prosigue, espera un carácter en la entrada**" se dedican a crear el hilo secundario.

El hilo secundario, que atiende la tarea de incrementar el contador, define su comportamiento con la función "**incrementaContador**"; posteriormente, al crear el hilo secundario hacemos referencia a ella obteniendo un apuntador con el operador **&** en el tercer parámetro de la función `pthread_create`.

Siempre es importante verificar que la ejecución de nuestras funciones ha terminado con el resultado deseado, por lo que debemos comprobar el valor que retorna y que colocamos en la variable `val_ret`. Idealmente, buscaríamos atender posibles excepciones, pero por brevedad hemos omitido esa parte del código, con lo que cualquier excepción terminaría el proceso por sus mecanismos por defecto.

Condiciones de carrera

Otro de los problemas que se presentan con frecuencia al usar concurrencia de procesos es el uso de recursos que no están preparados para atender peticiones distintas de forma concurrente. Estos recursos se conocen como `non-preemptive` o `non-threadsafe`, debido a que ambos casos se refieren a la imposibilidad que tienen estos de intercalar los pasos de la atención de peticiones separadas. Cuando estos recursos se usan de forma concurrente se suscita un problema que se conoce como condiciones de carrera. Entonces, para resolver este problema se requieren mecanismos que garanticen que solo uno de los procesos haga uso del recurso `non-preemptive` en un momento determinado, como las diversas técnicas de modelado que permiten diseñar una solución que garantice que no se generarán colisiones; una de las técnicas de modelado más sencillas es la que se conoce como **región crítica**, la cual se describe en la siguiente sección. Es importante destacar que por lo general se prefieren técnicas de diseño estáticas sobre pruebas de ejecución (dinámicas) debido a que la

colisión no se presenta siempre, aunque la condición de carrera esté presente, por lo que una prueba con un resultado exitoso no permite asegurar que se ha resuelto el problema.

ACTIVIDAD PROPUESTA

En acción

Ilustremos una condición de carrera con un programa que usa dos hilos. El objetivo del programa es incrementar de forma concurrente dos contadores hasta que lleguen a un valor determinado e imprimir en la consola el progreso de ambos. Al realizarlos, observamos que uno de ellos progresa invariablemente más rápido, por lo que también se presenta una creciente diferencia entre ellos.

En su forma original, que se presenta a continuación, el programa no funciona correctamente; cuando se ejecuta, en algunas ocasiones parece generar solo el segundo contador, aunque no se detecta fallo, y de hecho sí genera el hilo secundario. Si colocamos una demora antes de iniciar el segundo contador en el hilo principal, el problema no se presenta.

Este comportamiento errático se genera porque estamos creando una condición de carrera en el programa, de modo que este es sensible al tiempo que toma la atención de los diversos elementos del programa como tal. En ocasiones uno de los hilos puede tomar primero el recurso `non-preemptive` y generar un comportamiento, y en ocasiones puede ser el otro hilo y generar un comportamiento diferente.

¿Puedes determinar, mediante experimentación con el código y observación de lo que hace, sobre qué recurso se genera la condición de carrera?

```
/* Este programa utiliza la consola de forma concurrente para que dos
   hilos presenten información, la consola no está preparada para
  ello siendo un recurso non preemptive, lo que generará
  comportamientos indeseados.
Daniel Sol Llaven 2015-04-1
*/

#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

long contador[2]; //Generamos un arreglo, un contador por hilo.

void* salidaContador(void *id_pt) { //Firma correcta para
                                    pthread_create
      int id = *(int *)id_pt; //Tomamos el valor referido por el
                              parámetro
```

```
      id--; //Lo decrementamos para facilitar cálculos posteriores
      for(contador[id]=0l;contador[id]<1000000;contador[id]++) {
            printf("%sContador %d: %07ld,%07ld\r" //Todo en la misma
                                                  línea
                  ,(id==1?"\t\t\t\t":"") //Indentamos salida de un hilo
                  ,id+1,contador[id] //Contador y valor que lleva
                  ,contador[id]-contador[1-id]); //Diferencia con el otro
      }
      return NULL; //Para satisfacer la firma retorno válido
}

int main() {
      pthread_t id_hs; //Para la información del hilo
      int val_ret; //Para valores de retorno
      int id=1; //Para pasar el número de contador como parámetro
      //Creamos un hilo secundario para contador 1
      val_ret = pthread_create(&id_hs,NULL,&salidaContador,(void
      *)&id);
      if(val_ret) {
            printf("Error en la creación del hilo: %d\n", val_ret);
            exit(1);
      }
      printf("\nCreados los hilos\n");
      //sleep(0); //Descomentar para que funcione
      id=2; //Cambiamos el valor del parámetro a pasar
      salidaContador((void *)&id); //Invocamos en el hilo principal
      val_ret = pthread_join(id_hs, NULL); //Esperamos terminación
      if (val_ret) printf("Error al término del hilo: %d\n", val_ret);
      printf("\nTerminaron los hilos\n");
      exit(0);
}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.

La condición de carrera se genera sobre la variable en la que indicamos el número de contador que habrá de usarse. Cuando generamos el hilo secundario no pasamos el parámetro por valor, sino que pa-

samos la referencia a la variable; esta dirección se emplea para recuperar el valor y colocarlo en la variable `localid`, en la primera línea de `salidaContador`, pero en principio, la variable solo puede tener un valor en un momento determinado, y cuando el hilo secundario la está empleando, el hilo principal también puede modificar el valor (y en ocasiones efectivamente lo consigue) para hacer la segunda ejecución de la función.

Cuando el hilo secundario alcanza a usar el valor de `id` antes que el hilo principal lo modifique, el programa opera correctamente; de lo contrario, los dos hilos emplean el mismo valor de **`id`** para el contador, lo que genera resultados incorrectos.

La demora antes de hacer la invocación a `salidaContador` alivia el problema y proporciona tiempo suficiente al hilo secundario de usar el valor de la variable antes de modificarlo; sin embargo, no es una buena solución ya que desperdicia un segundo de ejecución en el hilo principal.

Podemos observar que las condiciones de carrera no son fáciles de detectar con pruebas ni mediante la revisión del código, y las formas de resolverlas tampoco son evidentes; por tanto, es necesario emplear técnicas formales para identificar y solventar dichas condiciones en nuestros sistemas para evitar comportamientos erráticos y poder garantizar su correcta operación. Una de estas técnicas es la de región crítica.

Importante

♥ **Docente.** La programación a prueba y error es una práctica muy difundida por su bajo costo; sin embargo resulta inapropiada para resolver problemas como el de las condiciones de carrera. Por tanto, debe procurarse tener la sensibilidad adecuada y los conocimientos necesarios para identificar cada tipo de problema y las diversas técnicas que funcionan mejor para cada uno. En el caso del problema de condición de competencia, la mejor técnica es el análisis formal.

♣ **Usuario.** Al realizar la validación de proyectos que impliquen multiprocesamiento, siempre debe considerarse que

Región crítica

En un programa, la región crítica se define como la parte del código en la que se hace uso del recurso `non-preemptive` y se comparte por dos o más procesos sobre el que se desea evitar la condición de carrera. Esto quiere decir que si se tiene más de un recurso `non-preemptive`, la región crítica para cada recurso debe analizarse por separado.

Una vez que se ha identificado la región crítica de un programa, se deben garantizar las condiciones sobre la región crítica que se describen a continuación, con el fin de asegurar que no se tiene una región crítica y, por tanto, que no puede ocurrir una colisión.

1. Dos procesos no deben estar simultáneamente en sus regiones críticas.
2. No deben hacerse suposiciones acerca de la velocidad o el número de procesadores.
3. Un proceso que no se encuentre en su región crítica no debe impedir que otro ingrese a su región crítica correspondiente.
4. Ningún proceso debe esperar indefinidamente para ingresar a su región crítica.

Los mecanismos específicos para lograr estas cuatro condiciones se revisarán en la sección de sincronización de procesos y parten del concepto de bloquear el hilo o proceso cuando desea ingresar a su región crítica pero otro se encuentra en ella en ese momento y desbloquearlo cuando salga de dicha región crítica.

Barreras de sincronización

Otro problema común, en especial cuando se trata de procesamiento de información, se refiere al hecho de que a menudo se debe esperar a que varios hilos de ejecución que trabajan en partes separadas del problema terminen la parte que les corresponde, a fin de reunir los resultados y proseguir con el procesamiento general. Para entender mejor este proceso se puede recurrir a la analogía de los equipos de alto rendimiento de Fórmula Uno, donde el equipo encargado de los pits se divide las tareas de mantenimiento y revisión que se deben realizar al auto a fin de hacerlas de manera simultánea y terminarlas en periodos muy cortos; sin embargo, antes de que el auto regrese a la pista, el responsable del equipo debe estar seguro de que todas las tareas se han concluido de manera satisfactoria y que cada participante del equipo se ha puesto a salvo, pues de lo contrario podría suceder un accidente potencialmente desastroso.

Cuando nuestro procesamiento de datos requiere verificar que un conjunto de hilos o procesos ha concluido, existe un patrón, conocido como barrera (`barrier`), que impide continuar hasta que todos los hilos hayan alcanzado el mismo punto de la ejecución, como sucede con las barreras de los hipódromos y otras carreras que impiden que los competidores avancen hasta un momento determinado, cuando es seguro continuar. Para implementar las barreras se usan los servicios del sistema operativo, que se estudian más adelante en la sección de sincronización y comunicación entre procesos.

Deadlock o interbloqueo

Una vez que se cuenta con bloqueos, surge un segundo problema: cuando dos o más recursos `non-preemptive` están generando bloqueos pueden presentarse situaciones que bloqueen a todos los

las pruebas funcionales no son suficientes y, por tanto, deben incluirse validaciones formales.

♠ **Arquitecto.** En el código 2, la región crítica consiste en las instrucciones que emplean el valor de la variable **`id`**, es decir, la primera línea de la función `salidaContador` y la expresión donde cambiamos el valor `id=2`.

Importante

♦ **Líder.** Las barreras de sincronización responden a un patrón de diseño que se representa de forma explícita en los diagramas de secuencia en UML, un lenguaje estándar de diseño; por tanto, es muy importante identificar la forma correcta de representar este y todos los demás patrones de concurrencia en nuestras metodologías de diseño.

♥ **Docente.** Los problemas de concurrencia han sido estudiados durante varias décadas, por lo que conocer y comprender las soluciones que se han propuesto y destacado a lo largo del tiempo constituye una parte importante de nuestras herramientas al buscar soluciones a problemas nuevos, además de que representan una valiosa inversión de tiempo.

procesos participantes para siempre. Dichas situaciones se conocen como **deadlocks**, aunque algunos autores también suelen llamarlas **bloqueos mutuos**; sin embargo, evitaremos esta última notación debido a que tiende a causar confusión respecto a la primera condición de la correcta solución de las condiciones de carrera, y en su lugar les llamaremos interbloqueos.

Un interbloqueo se define como la condición de un conjunto de procesos en la que todos estos están bloqueados, en espera de un evento o una condición que solo otro proceso del conjunto puede generar.

Una vez que se presenta un interbloqueo, y sin importar el tiempo de espera, los procesos seguirán bloqueados, ya que ninguno de estos, ni en conjunto ni de manera individual, pueden realizar las operaciones que resuelvan la situación debido a que también se encuentran bloqueados.

Para estar seguros de que un conjunto de bloqueos en espera de recursos en efecto constituye un interbloqueo, se busca que ocurran las cuatro condiciones siguientes: exclusión mutua, asignación y espera, no apropiación y espera circular.

1. **Exclusión mutua.** Las tareas requieren control exclusivo del recurso; es decir, respetan la primera condición de la solución de condiciones de carrera por región crítica.
2. **Asignación y espera.** Las tareas mantienen el control de los recursos que se les han otorgado, mientras esperan a que se les asignen el resto de los recursos que necesitan.
3. **No apropiación.** La asignación de los recursos a las tareas no puede revocarse sin tener efectos adversos, por lo que debe esperarse a que las propias tareas terminen las operaciones que deben realizar y liberen los recursos conforme a sus programas.
4. **Espera circular.** Existe una cadena circular de tareas, en la que cada tarea tiene asignado uno o más recursos que están siendo solicitados por la siguiente tarea de la cadena hasta completar un ciclo.

Prevención y formas de evitar interbloqueos

La prevención de interbloqueos suele partir de atacar alguna de las cuatro condiciones que lo definen para evitar que pueda concretarse.

Exclusión mutua

Con frecuencia podemos reemplazar los recursos `non-preemptive` por otros que ofrezcan una funcionalidad equiparable, pero que además sean `preemptive`; es decir,

que puedan atender diversas peticiones de forma concurrente. Por ejemplo, es posible reemplazar una implementación de una cola tipo FIFO (First In First Out) por otra que haya sido desarrollada con la concurrencia en mente. A menudo podemos encontrar estas implementaciones etiquetadas como "threadsafe", las cuales se comercializan a precios elevados, además de que con frecuencia implican cierto overhead por tener una complejidad mayor. En el caso de dispositivos de entrada, de salida, o ambos, por ejemplo, podemos reemplazar una impresora por un centro de impresión con plena capacidad para atender múltiples trabajos de impresión, almacenarlos en un disco duro local y procesar las impresiones en secuencia posterior. Estos dispositivos son a todas luces más complejos y con frecuencia tienen costos considerables; no obstante, pueden resultar viables de acuerdo con las necesidades del sistema.

ACTIVIDAD PROPUESTA

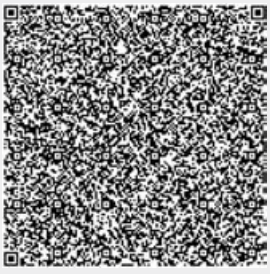
En acción

Podemos ilustrar el reemplazo de un recurso `non-preemptive` por uno `preemptive` mediante el programa que usaba dos contadores, en el cual podemos tener dos hilos secundarios, cada uno de los cuales atiende un contador, y eliminar la condición de carrera sobre la variable `id` generando una variable para cada uno. Como el valor de cada una de las variables no es usado de forma concurrente, desaparece la condición de carrera y el programa opera correctamente en todas sus invocaciones.

```
/* Este programa utiliza la consola de forma concurrente para que dos
   hilos presenten información, la consola no está preparada para ello
   siendo un recurso non-preemptive, lo que generará comportamientos
   indeseados.
Daniel Sol Llaven 2015-04-1
*/

#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

long contador[2];

void* salidaContador(void *id_pt) {
      int id = *(int *)id_pt;
      id--;
      for(contador[id]=0l;contador[id]<1000000;contador[id]++) {
            printf("%sContador %d: %07ld,%07ld\r",(id==1?"\t\t\t\t":"")
```

```
                  ,id+1,contador[id]
                  ,contador[id]-contador[1-id]);
      }
      return NULL;
}

int main() {
      pthread_t id_h1, id_h2;
      int val_ret;
      void *arg1,*arg2;
      int id1=1, id2=2; //Dos variables para paso de parámetros

      arg1=(void *)&id1; //Obtenemos sus direcciones por separado
      arg2=(void *)&id2;
      val_ret = pthread_create(&id_h1, NULL, &salidaContador, arg1);
      if(val_ret) {
            printf("Error en la creación del hilo1: %d\n", val_ret);
            exit(1);
      }
      val_ret = pthread_create(&id_h2, NULL, &salidaContador, arg2);
      if(val_ret) {
            printf("Error en la creación del hilo2: %d\n", val_ret);
            exit(1);
      }
      printf("Se crearon hilos secundarios\n");
      //Esperamos a que terminen las funciones de los hilos
      val_ret = pthread_join(id_h2, NULL);
      if (val_ret) printf("Error al término del hilo2: %d\n", val_ret);
      val_ret = pthread_join(id_h1, NULL);
      if (val_ret) printf("Error al término del hilo1: %d\n", val_ret);
      printf("\nTerminaron los hilos\n");
      exit(0);
}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.



Asignación y espera

Para evitar que una de las tareas tenga asignado un proceso y luego se bloquee en espera de otro, podemos controlar el orden en el que se les asignan los recursos. Una forma de hacer esto cuando diversas tareas usan prácticamente los mismos recursos, consiste en agrupar todos los recursos que requerirán dichas tareas con el fin de que se les asignen todos o ninguno de estos recursos, con lo que se evita que estas tengan asignaciones parciales.

No apropiación

Para evitar la condición de no apropiación cuando un proceso se encuentra en espera de un recurso que le falta, el cual tiene una prioridad mayor, o lo dicta así su algoritmo, se revocan todas las asignaciones previas de los recursos. No obstante, para ello debemos modificar los recursos mediante el uso de mecanismos como las transacciones, que permiten asegurar que si la operación no se concluye el recurso puede regresar a su estado original, a fin de que estas interrupciones en la atención no generen pérdidas o corrupción de la información por tratarse de recursos `non-preemptive`.

Espera circular

Constituye un concepto similar al de asignación y espera, solo que en este caso se asigna un número ordinal a cada recurso y se obliga a todas las tareas a solicitar dichos recursos en el mismo orden. Con esto se impide que una tarea que haya obtenido uno de los últimos recursos se encuentre en espera de uno de los primeros, según su número ordinal, y por tanto, no pueda cerrarse el ciclo de espera entre las tareas.

Detección

Otra forma de atacar el problema es proporcionar mecanismos al sistema operativo con objeto de detectar las ocasiones en que ha ocurrido un interbloqueo y luego proceder a tomar las medidas correspondientes. Esto implica un monitoreo constante de las asignaciones de recursos, lo que puede resultar muy costoso, además de que por este motivo suelen evitarse algoritmos de análisis de las asignaciones de recursos. Algunas implementaciones populares de detección son **ignorar el interbloqueo** y la **detección por tiempo de espera**.

Ignorar el interbloqueo

La mayoría de los sistemas interactivos no realiza intentos de detección y deja que el usuario identifique por sí mismo los periodos sin progreso y proceda a terminar los procesos y a liberar los recursos mediante el reinicio del equipo. Ignorar el interbloqueo se considera una solución viable de bajo costo para aplicaciones no críticas. Dadas sus características, esta estrategia también se conoce como el algoritmo del avestruz, aludiendo a que estas aves esconden la cabeza en la arena en caso de peligro, como sucede al ignorar el interbloqueo.

Detección por tiempo de espera

Se usa en conjunto con operaciones que soporten transacciones y muchos recursos, como las bases de datos relacionales. Por lo general, implementan periodos máximos de espera por las operaciones. De este modo, en caso de que se rebase el tiempo de espera establecido, independientemente de que sea por un interbloqueo o por algún otro factor que impida el progreso, se cancela la operación. En caso de contar con transacciones, entonces se retorna al estado original y se deja a criterio de la aplicación si desea reintentarla, en cuyo caso se espera que implemente un periodo de espera de longitud aleatoria para evitar que vuelva a coincidir con otras tareas, en caso de haber incurrido en un interbloqueo.

Importante

- **Usuario.** Es importante destacar que la prevención de interbloqueos tiene costos asociados, por lo que es importante considerar la criticidad de las aplicaciones y los costos asociados a fallos cuando se decide el tipo de solución que habrá de buscarse. En todo caso, es preferible hacer las inversiones correspondientes antes de que se presenten los problemas, lo cual se logra mediante un análisis racional previo.

Predicción

El sistema operativo puede utilizar diversos algoritmos para analizar la asignación de los recursos e impedir las secuencias de asignaciones que llevarían a interbloqueos; sin embargo, esto implicaría una sobrecarga en todas las operaciones de asignación de recursos e impondría requerimientos adicionales a todos los desarrollos de software que hicieran uso de recursos `non-preemptive`, incrementando el costo no solo de operación sino de desarrollo.

Sincronización de procesos

Una vez que tenemos concurrencia de múltiples procesos o múltiples hilos en un proceso, con frecuencia es necesario controlar el flujo de la ejecución para asegurar que algunos estados han sido alcanzados

antes de proseguir. Para ello se requieren mecanismos de sincronización que permitan detener las tareas en espera de algún elemento o condición externa.

Uno de los casos básicos es el de las esperas, que consiste en una tarea que puede necesitar detenerse y deba esperar a que se alcance una condición. Así, en vez de desperdiciar ciclos de la CPU entrando en un ciclo que verifique continuamente si la condición se ha alcanzado, podemos solicitar que la tarea deje de recibir atención de la CPU; es decir, que se bloquee y que otra tarea (mientras haya otras tareas que realizar en lo que se alcanza el estado deseado) reactive la que está bloqueada cuando detecte que es el momento indicado. Para realizar este tipo de implementaciones se usan dos tipos de llamadas a sistema; las primeras, conocidas como `sleep`, se usan en un inicio para que los procesos o hilos puedan solicitar que se les bloquee, mientras que las segundas (conocidas como `wakeup` y que constituyen la contraparte de las primeras) se utilizan posteriormente para reactivar un proceso bloqueado.

No obstante, estas llamadas tienen la desventaja de que si la llamada de `wakeup` llega antes de que la tarea tenga ocasión de bloquearse con `sleep`, la señal de despertar se perderá y la tarea permanecerá bloqueada.

Semáforos

Para evitar problemas como los descritos antes, E. W. Dijkstra propuso asociar contadores a las señales enviadas, para evitar así que se perdieran por problemas de sincronización.

Formalmente, un semáforo es una variable o tipo de dato abstracto usado para controlar el acceso, para múltiples procesos o hilos, a un recurso compartido en ambientes de multiproceso o multiusuario.

El semáforo cuenta con un contador entero y dos operaciones análogas a `sleep` y `wakeup` que en POSIX se conocen como `sem_wait` y `sem_post`, respectivamente. La implementación que de estas funciones haga el sistema operativo debe garantizar que no puedan ser interrumpidas, por lo que se conocen como atómicas (es decir, sin partes).

- `sem_wait`. Prueba el valor del semáforo con el fin de comprobar si este tiene un valor mayor a 0. De ser así, lo decrementa y termina la operación, permitiendo progresar a la tarea que hizo la llamada. Por el contrario, si el semáforo tiene un valor de 0, bloquea la tarea que lo invocó como un `sleep`.
- `sem_post`. Incrementa el valor del semáforo. Esto es, si el valor del semáforo termina con un valor mayor a 0, alguna de las tareas bloqueadas por invocar

`sem_wait` puede despertar y proceder a decrementar el valor del semáforo y proseguir.

Mutex

Un caso especial de los semáforos es aquel donde este solo se emplea para controlar el acceso a la región crítica, como se describió antes en este capítulo. Estos semáforos se conocen como `mutex` y solo toman valores de 0 y de 1, ya que solo deben permitir que al menos una tarea ingrese a la región crítica. Las funciones sobre el `mutex` también reciben nombres especiales (de nuevo para la API de POSIX utilizado en el sistema operativo UNIX).

- `pthread_mutex_lock`. El nombre hace referencia a su uso para controlar threads individuales de ejecución dentro de los procesos. La operación `lock` verifica si el `mutex` está libre (valor de 1), en cuyo caso lo cierra para que otro hilo no pueda ingresar a la región crítica, para lo cual decrementa el valor del `mutex` de manera análoga a un `sem_wait` en un semáforo.
- `pthread_mutex_unlock`. Función análoga a la operación `sem_post`, que incrementa el valor del `mutex` si este es de 0 y desbloquea alguno de los `threads` que estén en espera de este `mutex`. Como solo una tarea debería estar en región crítica, esta función puede generar un error de operación en caso de que se trate de liberar un `mutex` que no está cerrado.

ACTIVIDAD PROPUESTA

En acción

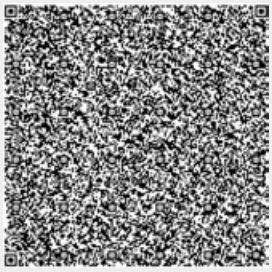
Retomando el ejemplo de la condición de carrera sobre el valor de una variable, podemos definir una región crítica que inicia cuando la variable recibe el valor que deseamos utilizar como parámetro y termina cuando el recién creado hilo utiliza dicho valor. Si protegemos esta región con un `mutex` eliminaremos el comportamiento anómalo. A continuación se utiliza el código del programa donde se usa una función `proxy` (o intermediaria) para definir con más claridad la región crítica de ambos hilos y que emplea un `mutex` para resolver la condición de carrera. Se recomienda que al revisar la ejecución del programa se comente el uso del `mutex` para apreciar la aparición de la colisión sobre el valor de la variable.

```
/* Este programa utiliza la consola de forma concurrente para que dos
   hilos presenten información, se utiliza un mutex para evitar
   condición de carrera sobre el valor de la variable id.
```

```
Daniel Sol Llaven 2015-04-1
*/


#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>


long contador[2];
int id; //Recurso non_preemptive de ejemplo
pthread_mutex_t id_mutex;
void* salidaContador(void *id_pt);

pthread_t proxy(int numero) { //Función encargada de iniciar hilos
                              contadores
      void *arg;
      int val_ret;
      arg=(void *)&id; //Obtenemos dirección de la variable
      pthread_t id_h;
      pthread_mutex_lock(&id_mutex);
      //Inicia región crítica, hemos tomado el mutex
      id=numero;
      val_ret = pthread_create(&id_h, NULL, &salidaContador, arg);
      //Región crítica terminó en salidaContador
      if(val_ret) {
            printf("Error en la creación del hilo%d:%d\n",numero,
                  val_ret);
            exit(1);
      }
      return id_h;
}


void* salidaContador(void *id_pt) {
      //Continúa región crítica iniciada en proxy
      int id = *(int *)id_pt;
      //Termina región crítica, liberamos mutex
      pthread_mutex_unlock(&id_mutex);
      id--;
      for(contador[id]=0l;contador[id]<1000000;contador[id]++) {
            printf("%sContador %d: %07ld,%07ld\r",(id==1?"\t\t\t\t":"")
                  ,id+1,contador[id]
                  ,contador[id]-contador[1-id]);
      }
      return NULL;
}
```

```
int main() {
      pthread_t id_h1, id_h2;
      int val_ret;
      //Iniciar atributos del mutex a valores por defecto
      val_ret = pthread_mutex_init(&id_mutex, NULL);
      if (val_ret) printf("Error al crear mutex: %d\n", val_ret);
      id_h1=proxy(1); //Invocamos proxy para generar hilos
      id_h2=proxy(2);
      printf("Se crearon hilos secundarios\n");
      //Esperamos a que terminen las funciones de los hilos
      val_ret = pthread_join(id_h2, NULL);
      if (val_ret) printf("Error al término del hilo2: %d\n", val_ret);
      val_ret = pthread_join(id_h1, NULL);
      if (val_ret) printf("Error al término del hilo1: %d\n", val_ret);
      val_ret = pthread_mutex_destroy(&id_mutex);
      if (val_ret) printf("Error al término de mutex: %d\n", val_ret);
      printf("\nTerminaron los hilos\n");
      exit(0);
}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.



Monitores

Hoy día se considera a los semáforos como primitivos para el desarrollo de soluciones de sincronización, por lo que se espera que los lenguajes de programación proporcionen interfaces más convenientes a sus desarrolladores para soportar sus sistemas. El nombre común que se suele dar a las diversas implementaciones de alto nivel es el de monitores.

Los monitores fueron propuestos por Brinch Hansen como una estructura base o primitiva en los lenguajes de programación, que consta de una colección de procedimientos, variables y estructuras de datos que están agrupadas en un módulo o paquete especial. Los datos del monitor solo pueden emplearse por los procesos mediante in-

vocaciones a los procedimientos del propio monitor, y la implementación del monitor debe garantizar la exclusión mutua en estas invocaciones.

En este punto resulta muy conveniente revisar algunas de estas implementaciones con sus nomenclaturas y consideraciones especiales.

- **POSIX C++ - Variables de condición (Condition Variable Attributes)**

  Como se menciona en el libro *Programming in* C, de A.D. Marshall, estos son objetos que se pueden usar para bloquear `threads` hasta el momento en que una condición particular sea verdadera. Siempre se usan en conjunto con `mutex` que protegen el uso de la variable de condiciones de competencia.

  De este modo, con una variable de condición, un `thread` se puede bloquear hasta que se satisfaga una condición.

  La condición se evalúa en una región crítica propia, bajo la protección de un `mutex`.

  Cuando la condición es falsa, usualmente se bloquea la tarea en espera de la variable de condición y de forma atómica se libera el `mutex`, para quedar en espera de que se libere un cambio en la condición.

  Cuando otra tarea modifica la variable de condición, esta puede enviar señales para revocar el bloqueo de una o más tareas bloqueadas en espera de esta, para que vuelvan a adquirir el `mutex` y evalúen de nueva cuenta sus condiciones.

ACTIVIDAD PROPUESTA

En acción

Retomando el ejemplo anterior, podemos controlar el acceso a la región crítica definiendo un monitor con las *control variable*s, o variables de control, de POSIX en C para el uso de la variable global `id` y el `mutex id_mutex` para controlar la exclusión mutua en el uso de esta. La variable de control se inserta para que en la región crítica de los contadores, una vez que ha empleado el valor de `id` excluyendo a los demás, queden en espera de una señal de parte del hilo principal para iniciar su cuenta, lo que hace que el inicio de ambos hilos sea mucho más cercano en tiempo y la diferencia inicial en las cuentas se minimice. Para resaltar esto se disminuyó a 100 la cuenta total para que se pueda apreciar la diferencia inicial debido a que un hilo recibe atención del procesador antes que el otro.

```
/* Este programa utiliza la consola de forma concurrente para que dos
   hilos presenten información. Se utiliza un monitor de variable de
   condición para evitar la condición de carrera sobre el valor de la
   variable id.
```

```
Daniel Sol Llaven 2015-04-1
*/

#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

long contador[2];
int id; //Recurso non_preemptive de ejemplo
pthread_mutex_tid_mutex;
pthread_cond_t cv;
void* salidaContador(void *id_pt);

pthread_tproxy(int numero) { //Función encargada de iniciar hilos
                             contadores
      void *arg;
      intval_ret;
      arg=(void *)&id; //Obtenemos dirección de la variable
      pthread_tid_h;
      pthread_mutex_lock(&id_mutex);
      //Inicia región crítica, hemos tomado el mutex
      id=numero;
      val_ret = pthread_create(&id_h, NULL, &salidaContador, arg);
      //Región crítica terminó en salidaContador
      if(val_ret) {
            printf("Error en la creación del hilo%d:%d\n",numero,
                  val_ret);
            exit(1);
      }
      returnid_h;
}

void* salidaContador(void *id_pt) {
      //Continúa región crítica iniciada en proxy
      int id = *(int *)id_pt;
      //Termina región crítica, espera de señal de CV
      pthread_cond_wait(&cv,&id_mutex);
      pthread_mutex_unlock(&id_mutex); //Finalmente libera el mutex
      id--;
      for(contador[id]=0l;contador[id]<100;contador[id]++) {
            printf("%sContador %d: %07ld,%07ld\r",(id==1?"\t\t\t\t":"")
                  ,id+1,contador[id]
                  ,contador[id]-contador[1-id]);
      }
      return NULL;
}
```

```
int main() {
      pthread_t id_h1, id_h2;
      intval_ret;
      //Iniciar atributos del mutex a valores por defecto
      val_ret = pthread_mutex_init(&id_mutex, NULL);
      if (val_ret) printf("Error al crear mutex: %d\n", val_ret);
      //Iniciar variable de condición con atributos por defecto
      val_ret = pthread_cond_init(&cv, NULL);
      if (val_ret) printf("Error al crear CV: %d\n", val_ret);
      id_h1=proxy(1); //Invocamos proxy para generar hilos
      id_h2=proxy(2);
      printf("Se crearon hilos secundarios\n");
      //Usamos CV para indicar a ambos que deben proseguir, en RC
      pthread_mutex_lock(&id_mutex);
      val_ret = pthread_cond_signal(&cv);
      if (val_ret) printf("Error en señal de CV: %d\n", val_ret);
      val_ret = pthread_cond_signal(&cv);
      if (val_ret) printf("Error en señal de CV: %d\n", val_ret);
      pthread_mutex_unlock(&id_mutex);
      //Esperamos a que terminen las funciones de los hilos
      val_ret = pthread_join(id_h2, NULL);
      if (val_ret) printf("Error al término del hilo2: %d\n", val_ret);
      val_ret = pthread_join(id_h1, NULL);
      if (val_ret) printf("Error al término del hilo1: %d\n", val_ret);
      val_ret = pthread_mutex_destroy(&id_mutex);
      if (val_ret) printf("Error al término de mutex: %d\n", val_ret);
      val_ret = pthread_cond_destroy(&cv);
      if (val_ret) printf("Error al término de CV: %d\n", val_ret);
      printf("\nTerminaron los hilos\n");
      exit(0);
}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.



Calificador `synchronize` en Java

En el lenguaje de programación orientado a objetos Java existe un símbolo especial que podemos aplicar como un calificador de un método o de una expresión. Su forma más sencilla es la que se aplica a métodos y, por tanto, es la que revisaremos a continuación.

Cuando a uno de los métodos en una clase se le añade el calificador `synchronize`, este garantiza, por la máquina virtual, las condiciones que se describen a continuación.

- No es posible que dos invocaciones del método calificado operen de forma concurrente. Esto significa que cuando un `thread` esté ejecutando el método, todas las operaciones que intenten ejecutarlo en la misma máquina virtual se bloquearán hasta que termine la ejecución activa.
- Cuando el método termine su ejecución, se garantiza que todos los demás `threads` tendrán acceso al mismo estado actualizado del objeto que contiene el método cuando procedan a ingresar a este.

Como se puede deducir, lo que logra este calificador es una forma sencilla de garantizar que todas las invocaciones concurrentes del método serán serializadas en el ámbito de una máquina virtual Java.

ACTIVIDAD PROPUESTA

En acción

Recreando la condición de carrera sobre una variable con información importante para los nuevos hilos, que estos tienden a recibir erróneamente al modificar el valor antes de que puedan utilizarla pero en lenguaje Java, usamos los métodos `synchronized` y reemplazamos los `proxy` de proyectos pasados por el método `ContadoresConcurrentes.getId`, que contiene todas las operaciones de nuestra región crítica.

El proyecto Java tiene tres clases: una encargada de manejar el arreglo de contadores, llamada Contador Dao; la que implementa la funcionalidad a ejecutar en los hilos secundarios, llamada Salida Contador; y la del hilo principal, llamada Contadores Concurrentes.

```
packagecom.patria.soping.dao.contador;

public class ContadorDao {
      private long[] valores;
```

```
	/**
	 * constructor, recibe número de contadores a mantener
	 */
	public ContadorDao(int capacidad) {
		if(capacidad<1 || capacidad>100) return; //Fuera de rango
		valores = new long[capacidad];
		for(int c=0; c<capacidad; c++) {
			valores[c]=0L;
		}
	}

	public long getValor(int indice) {
		return valores[indice];
	}

	public long incrementaValor(int indice) {
		return ++valores[indice];
	}
}
```

Puede usar este código QR para introducir rápidamente el de la clase a un dispositivo para modificarlo, experimentar con ella y observar su comportamiento en ejecución.

```
/**
 * Clase que implementa los hilos que usarán de manera concurrente los
 * contadores y presentan el progreso a consola.
*/
packagecom.patria.soping.salidaSinColision;

/**
 * @author Daniel Sol Llaven
 * @since 2015-04-04
*/
public class SalidaContador   implements Runnable {
	private int id;
	public int getId() {
```

```
            return id;
      }
      public void setId(int id) {
            this.id = id;
      }
      @Override
      public void run() {
            // Utiliza el contador para progresar a un valor
            predefinido
            // imprimiendo a consola
            int c;
            char tab;
            id=ContadoresConcurrentes.getId();
            //Toma el valor de id del estado actual en
            ContadoresConcurrentes
            if(id==2) {
                  tab='\t';
            }
            else {
                  tab=' ';
            }
            for(c=0;c<100000;c++) {
                  System.out.format("%c%c%cContador %d:%06d\r", tab,
                     tab,tab,id
                        ,ContadoresConcurrentes.cont.incrementaValor
                        (id-1));
            }
      }
}
```

Puede usar este código QR para introducir rápidamente el de la clase a un dispositivo para modificarlo, experimentar con ella y observar su comportamiento en ejecución.

```
/**
 * Ejemplo de programa multihilos sin colisión en el uso de la variable
 * usada para manejar los identificadores de contador.
```

```
*/
packagecom.patria.soping.salidaSinColision;

importcom.patria.soping.dao.contador.ContadorDao;


/**
 *
 * @author Daniel Sol Llaven
 * @since 2015-04-04
 */
public class ContadoresConcurrentes {

      /**
      * Atributos estáticos a usar como variables globales para el ejemplo
      */
      public static ContadorDao cont;
      //Volatile asegura que los hilos reciban el estado correcto.
      public static volatile int id;

      /**
       * Al ser invocado de forma concurrente, garantiza mediante synchronized
       * que solo un hilo es atendido en un momento determinado, reune
       * el retornar el valor y modificarlo para el hilo siguiente
      */
      public synchronized static int getId() {
            return id++;
      }

      public synchronized static void setId(int id) {
            ContadoresConcurrentes.id = id;
      }


      /**
       * @param No recibe parámetros de importancia.
      */
      public static void main(String[] args) {
            Thread t1,t2;
            SalidaContador sc1, sc2;
            // Rutina principal, iniciar los contadores y los hilos
            cont=new ContadorDao(2);
            System.out.println("Iniciando Hilos");
            // Al crear los objetos sería el momento ideal para
               establecer id
            // lo omito para recrear la condición de carrera con
               fines ilustrativos
            sc1=new SalidaContador();
```

```
            sc2=new SalidaContador();
            t1=new Thread(sc1);
            t2=new Thread(sc2);
            try {
                  setId(1);
                  t1.start(); //Inicia primer thread
                  t2.start(); //Inicia segundo thread
                  t1.join();
                  t2.join();
            } catch (InterruptedException e) {
                  // En caso de que se interrumpa alguno de los hilos
                  System.out.println("Término prematuro");
                  e.printStackTrace();
            }
            System.out.println("");
      }

}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.

3.4 Comunicación entre procesos

Señales

Las señales son mecanismos que se usan para comunicar entre sí a los procesos, interrumpiendo el flujo normal de la ejecución del proceso que recibe la señal y entregando un código de señal que dirige la forma en que el proceso debe atender dicha interrupción.

Como las señales interrumpen la ejecución, se requieren mecanismos de protección con el fin de evitar que procesos que no deberían interferir obstruyan la operación de los sistemas, ya sea por error o por ser desarrollos mal intencionados. El principal

mecanismo de protección es permitir el envío de señales solo a procesos que pertenecen al mismo usuario. Además, se permite que un proceso envíe señales a los procesos que generó o que sean sus descendientes.

Alarmas

Las alarmas son mecanismos análogos a las señales; estas demoran la entrega de la señal por un periodo, el cual se especifica al enviarla, y envían un solo tipo de señal (SIGALRM en POSIX). Las alarmas son en especial útiles para implementar tiempos de espera de todo tipo de eventos, lo que permite que un hilo de ejecución se bloquee sin requerir de otro que lo reactive, ya que él mismo puede programar la alarma con este fin.

EJEMPLO

Programa que implemente el envío y recepción de una alarma.

```
/**
* Programa que se programa una alarma en un minuto y luego comienza a
* esperar una entrada de parte del usuario, si la alarma caduca antes
* de recibir la entrada se recibe la señal y se termina el proceso.
* Daniel Sol Llaven 2015-04-07
**/

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void handlerSIGALRM(int sig);

int main() {
      int secRestantes;
      char *entrada;
      entrada=malloc(10*sizeof(char));
      signal(SIGALRM, handlerSIGALRM); //Redefine handler de SIGALRM
      printf("Espero una cadena de entrada con enter hasta 10
            segundos\n");
      secRestantes=alarm(10); // Espera hasta 10 segundos la entrada
      if(secRestantes>0) {
            printf("Existía una alarma previa con %d segundos restantes",
            secRestantes);
      }
```

```
      scanf("%10s", entrada); //El proceso se detendrá aquí
      printf("Se recibe la entrada de usuario: %s",entrada);
      return 0;
}

//Rutina de atención de la señal, notifica señal que se recibió
voidhandlerSIGALRM(intsig) {
      printf("\nSe recibe la señal de la alarma #%d\n",sig);
      exit(0);
}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.

Pipes

Los `pipes` o tuberías son una interfaz de comunicación entre procesos, creada por UNIX y adoptada posteriormente por el estándar POSIX, que actúan como un archivo que genera dos descriptores, uno por el cual se puede escribir información y otro por el que esa información puede ser leída.

Los `pipes` no son archivos, por lo que a pesar de representarse como tales al darles nombres y de usar las mismas interfaces de programación no escriben información a los dispositivos de almacenamiento que soportan los sistemas de archivos; por el contrario, almacenan la información exclusivamente en un buffer de memoria. Esto significa que solo pueden almacenar una cantidad limitada de información, la cual depende de la implementación, aunque suele estar en el orden de los KB. Debido a esto, se espera que al usar los `pipes` la información sea leída continuamente, pues cuando el buffer se sature, las operaciones de lectura se bloquearán cuando intenten leer y no se encuentre información disponible.

Los `pipes` no reconocen paquetes de información o mensajes con algún terminador, por lo que es responsabilidad de las aplicaciones saber distinguir el formato y la organización de los datos recibidos en el flujo de información.

EJEMPLO

```
/**
* Programa que genera un par de pipe, genera un proceso hijo con fork y
* comunica la entrada recibida por el padre al hijo mediante el pipe.
* Daniel Sol Llaven 2015-04-07
**/

#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

void imprimeDePipe(int leePipe); //Funcionalidad para el hijo
void enviaAPipe(int escribePipe); //Funcionalidad para el padre

int main() {
      pid_t procHijo;
      int pipeFileDescriptors[2]; //Descriptores de ambos extremos
      if(pipe(pipeFileDescriptors) == -1) { //Genera pipe
            printf("Error al crear pipe\n");
            exit(1);
      }

      procHijo=fork(); //Genera proceso hijo
      if(procHijo<0) {
            interrnum=errno; //Preservamos código de error
            printf("Error %d al generar proceso hijo con
            fork\n",errnum);
            exit(1);
      }

      if(procHijo==0) {
            //Es el hijo, cierra pipe de envío y procede
            close(pipeFileDescriptors[1]);
            imprimeDePipe(pipeFileDescriptors[0]);
      }

      if(procHijo>0) {
            //Es el padre, cierra pide de recepción y procede
            close(pipeFileDescriptors[0]);
            enviaAPipe(pipeFileDescriptors[1]);
      }

      return 0;
}
```

```
/**
* Funcionalidad para el hijo, lee del pipe la cadena enviada, imprime
* el contenido de esta a STDOUT, cierra su extremo del pipe (lectura)
* y termina su proceso.
**/
void imprimeDePipe(int leePipe) {
      char buf; //Carácter de buffer
      printf("Proceso hijo, esperando cadena...\n");
      while (read(leePipe, &buf, 1) > 0)
            write(STDOUT_FILENO, &buf, 1);
      write(STDOUT_FILENO, "\n", 1); //Fin de línea
      close(leePipe);
      printf("Proceso hijo, finalizando\n");
      exit(0);
}

/**
* Funcionalidad para el padre, lee una cadena de hasta 10 caracteres,
* la escribe a su extremo del pipe, cierra su extremo y espera a que
* un proceso hijo termine antes de terminar su propio proceso.
**/
void enviaAPipe(int escribePipe) {
      char buf[10]; //Buffer de hasta 256 caracteres.
      printf("Proceso padre, ingresa una cadena de 10 caracteres y
            enter:\n");
      scanf("%10c", &buf);
      printf("\n\n"); //Separa la entrada de las futuras salidas.
      write(escribePipe, &buf, strlen(buf));
      close(escribePipe); //Presenta EOF al proceso hijo
      wait(NULL); //Espera terminación de hijo
      printf("Hijo terminado, terminando proceso padre\n");
      exit(0);
}
```

Puede usar este código QR para introducir rápidamente el del programa a un dispositivo para modificarlo, experimentar con él y observar su comportamiento en ejecución.

Mecanismos de System V

El System V, o sistema cinco, fue una de las primeras implementaciones comerciales de UNIX, lo que estableció la propiedad de AT&T sobre la base de la mayoría de las versiones posteriores. Por tanto, Linux y BSD han necesitado esfuerzos dirigidos de manera explícita a verificar que su implementación no sea un derivado de este. Independientemente de ello, las implementaciones de diversos conceptos de comunicación entre procesos y de la propia administración de estos constituyeron la base sobre la que posteriormente se desarrollaron los estándares que hoy día buscan soportar sistemas operativos como POSIX.

ACTIVIDAD PROPUESTA

En acción

Cuando aparecieron las primeras versiones de System V, las aplicaciones aún no adoptaban muchos de los requerimientos que hoy día son comunes. Parte del esfuerzo de AT&T por lograr su adopción se dedicó a promocionar las bondades del entonces nuevo sistema operativo UNIX.

Revisa en Internet algunos materiales promocionales de nivel técnico que la propia compañía AT&T haya conservado y publicado (puedes consultar un video muy ilustrativo en la siguiente dirección de Internet: https://www.youtube.com/watch?v=XvDZLjaCJuw) e identifica en estos los siguientes aspectos:

1. A los creadores del SO y de C (Ritchie, Thompson, Brian Kernighan).
2. Las facilidades de comunicación entre procesos de UNIX resaltadas.
3. Los tipos de problemas que se trabajaban durante la época en que se creó System V, así como la participación del multiprocesamiento y la comunicación entre procesos en la elaboración de soluciones.
4. Los elementos en las interfaces de usuario presentes y los que aún no se tenían, en relación con las interfaces modernas.
5. La postura acerca de los derechos de autor de los creadores del System V en la época de su creación.

IPC en System V

En la actualidad la mayoría de las implementaciones de UNIX cuentan tanto con la API de POSIX como con el de System V para la comunicación entre procesos (IPC). No obstante, la implementación de POSIX suele ser la más recomendable para desarrollos nuevos por ser la más reciente, estándar y consumir menos recursos.

Sin embargo, el System V aún se utiliza con mucha frecuencia debido a que la implementación de POSIX aún resulta incompleta en muchas plataformas que sí tienen implementaciones completas de la API de System V, además de que también es idónea para mantener el soporte de sistemas antiguos y porque en ocasiones la portabilidad no es importante o porque es deseable alguna característica de System V.

Identificadores de IPC de System V

La funcionalidad de IPC de System V identifica a los distintos elementos en su totalidad, ya sean semáforos, `pipes` o memoria compartida con un número único para todos los procesos que operen bajo el mismo sistema operativo. En diversas ocasiones, estos son llamados objetos, a pesar de no seguir el paradigma de programación orientada a objetos.

Comandos de soporte

A la fecha, IPC de System V cuenta con dos programas de soporte para los elementos: `ipcs` e `ipcrm`.

- `ipcs` permite obtener información de los objetos de IPC de todo el sistema.
- `ipcrm` permite destruir el registro del sistema operativo de un objeto de IPC.

Semáforos en System V

Algunas de las características más notables de la implementación de semáforos de System V son:

- Cada objeto semáforo utiliza un arreglo de contadores en vez de uno solo.
- Permite controlar la cantidad en la que se debe incrementar o decrementar el contador en una operación.
- Cada objeto semáforo está orientado a dar servicio a todo el sistema, por lo que resulta necesario identificar cada uno de los semáforos y mantener el control de uso de los identificadores en cada una de las operaciones y para todos los procesos que puedan coexistir en el sistema informático.

Memoria compartida en System V

La memoria compartida es un conjunto de posiciones de memoria del tamaño que se requiera. Esta se compone de posiciones de memoria que están juntas, también cono-

cidas como segmentos de memoria. Al inicio la memoria es reservada por un proceso, pero luego puede ser utilizada por varios procesos que la están compartiendo.

Como esta no tiene operaciones intermediarias, constituye el mecanismo más rápido de comunicación entre procesos; sin embargo, deben tomarse las precauciones necesarias para evitar condiciones de carrera sobre su contenido, además de que tampoco tiene ninguna garantía de sincronización.

De este modo, el uso de memoria compartida en System V debe seguir los pasos generales que se describen a continuación:

- Obtener un identificador de IPC para el segmento de memoria compartida con la llamada de sistema `shmget`, que indica si se debe crear o recuperar uno ya existente y el tamaño de este.
- Obtener la dirección base del segmento para proceder a utilizarlo.
- Cuando terminamos de usar el proceso, debemos disociarlo del segmento de memoria compartida con una llamada al sistema `shmdt`.
- Al usar la llamada al sistema `shmctl`, se puede indicar al sistema operativo que se debe liberar el segmento de memoria compartida cuando se llegue al punto en que ningún proceso lo usará, así como controlar sus privilegios y recuperar la información de su estructura de control.

Paso de mensajes

En System V, la funcionalidad del paso de mensajes se implementa mediante colas identificadas con un número único, las cuales se almacenan en memoria reservada del sistema operativo a fin de proporcionar llamadas al sistema para agregar mensajes o recuperarlos de la cola.

Los mensajes se representan con estructuras de datos derivadas del buffer de mensajes definido por la API, conocido como `msgbuf`, el cual tiene un campo entero de tipo entero largo y se redefine para incluir otros campos para el contenido. Es importante resaltar que la API no hace suposiciones acerca de la estructura o la longitud de estos.

El sistema operativo maneja la colección de mensajes dentro de una cola, como una lista ligada de estructuras derivadas de `msgbuf`, de las cuales, al inicio, registra apuntadores de los datos y la longitud de estos. Esta información se almacena en registros de tipo `msg`. Para representar la información sobre la cola completa se usa la estructura de datos `msqid_ds`, que incluye apuntadores al primer y al último mensajes, marcas de tiempo del último envío, recepción y cambio, identificadores de proceso del último remitente y destinatario, y el máximo número de bytes que puede almacenar toda la cola. Este proceso consta de los siguientes pasos:

1. Para crear o empezar a utilizar una cola, debemos usar la llamada al sistema `msget`, indicando el identificador de IPC y una bandera acerca de cómo debemos proceder en caso de que la cola no exista. En este paso no es necesario indicarle si el proceso enviará, recuperará o incluso si tiene autorización de usar la cola que se solicitó.
2. Para enviar mensajes, la información debe representarse en estructuras derivadas de `msgbuf` y entregarse mediante una llamada a `msgsnd`, indicando el identificador IPC, el `msgbuf` con la información y la longitud del contenido del mensaje. Al invocarla se verifica que la cola siga activa y que el proceso que se invoca tenga privilegios suficientes para usar la cola; asimismo, se maneja un esquema de permisos igual al de los sistemas de archivos UNIX. Para indicar si se desea que la petición bloquee o no el proceso, en caso de que la cola haya saturado su espacio en memoria, se puede hacer uso de banderas.
3. Para recuperar los mensajes se debe hacer una llamada al sistema a `msgrcv`, indicando el identificador IPC, un apuntador a `msgbuf` donde copiar la información y el tipo y la longitud del mensaje. En caso de haber al menos un mensaje en la cola del tipo solicitado, la llamada copiará el contenido del último de estos en llegar al espacio señalado. Si se indica un tipo 0, entonces se recupera el último mensaje, independientemente de su tipo. También se le puede indicar al sistema si se desea que la llamada bloquee el proceso en caso de que la cola esté vacía o no.
4. Con la llamada al sistema `msgctl` se pueden solicitar copias de las estructuras de control de la cola para controlar sus características (por ejemplo, los permisos para solicitar que la cola sea eliminada).

Jerarquías de procesos y protección

Los procesos guardan información con respecto a su creación no solo con propósitos de registro sino también porque dicha información es importante en términos de limitar la influencia que un proceso puede tener en otros, a fin de asegurar que el trabajo de unos usuarios o aplicaciones no afecte innecesariamente al de otros.

Uno de los datos más importantes que se registran es el proceso que crea a cada proceso. Este dato puede tomar un valor nulo cuando el proceso es iniciado por el sistema operativo durante el arranque del sistema (`boot`); sin embargo, casi nunca es el caso para las aplicaciones de usuario.

Partiendo de los procesos iniciados en el arranque que no registran ningún proceso padre, se tiene una jerarquía de procesos construida con los procesos generados a partir de estos, que se conocen como sus hijos y los procesos hijos de estos, y así sucesivamente. Esta jerarquía permite limitar la comunicación y los privilegios de los pro-

cesos que generan a otros para enviar señales y mediante ellas, hasta terminar a sus procesos hijos, sin que por ello cualquier proceso tenga la capacidad de terminar a cualquier otro, ya que no todos los procesos serán descendientes suyos.

Una de las aplicaciones que tiene la jerarquía de procesos es ayudar al procesador a impedir que los procesos que generen gran cantidad de procesos hijos terminen por acaparar la atención del procesador con sus aplicaciones. De este modo, la planificación de procesos puede procurar que las diversas jerarquías de procesos asociados, por ejemplo, a usuarios distintos consuman cantidades similares de recursos, a pesar de tener números muy diferentes de procesos.

Las señales de comunicación entre procesos también se benefician de las jerarquías. En un principio solo se permitía que los procesos enviaran señales a sus hijos, sin requerir privilegios de administración; sin embargo, en la actualidad POSIX requiere que los procesos que intercambian señales pertenezcan al mismo usuario o tengan privilegios de administración.

API de gestión de procesos

Las interfaces de programación con las que se cuenta para el desarrollo de aplicaciones dependen de las plataformas de desarrollo que vayamos a utilizar y pueden ser muy variadas. Sin embargo, el uso de multiprocesamiento, comunicación entre procesos y paso de mensajes se inició con los sistemas UNIX, y posteriormente fue reforzado por las normas POSIX para uniformar las prestaciones de las API en diversas implementaciones o "sabores" de UNIX. Aunque a la fecha esta norma no se ha implementado por completo en todos los UNIX, junto con las API disponibles en las primeras versiones de UNIX con este tipo de funciones (conocido como System V), han sido el estándar de facto para los lenguajes de programación funcionales y declarativos como C. Otros tipos de lenguajes de programación, como la programación funcional de Haskell, no requieren exponer una interfaz de control de la concurrencia porque cuentan con mecanismos propios y particulares para manejarla. Así, basta con comprender las interfaces de programación de POSIX y System V para entender la funcionalidad similar disponible en las plataformas que se apegan a estas normas, o en aquellas que se basan en estas.

A continuación revisaremos los ciclos de vida de los elementos de multiprocesamiento que hemos revisado y haremos mención de las funciones que se requiere usar en cada uno de ellos. La documentación detallada y específica de cada una de estas

Importante

¿Por qué al arranque se le conoce como `boot`? Esto se debe a una analogíaalgo cómica que ha perdurado a lo largo del tiempo, la cual hace alusión a que un sistema operativo debe tirar de sí mismo para levantarse, de la misma forma que las botas deben calzarse tirando de estas por sus correas o cincha (en inglés llamada `bootstrap`).

♥ **Docente.** La historia y terminología de la computación está llena de pequeños intentos de humor, los cuales conviene conocer para desmitificar el progreso que se ha tenido en las últimas décadas y por la relevancia que suelen tener en la naturaleza de los conceptos a que se refieren, además de que ayudan a que los temas no se vuelvan áridos y memorísticos.

funciones está disponible para el programador mediante la herramienta `man` de UNIX o en documentación especializada de la versión del sistema operativo que se desee emplear para el desarrollo, y aunque es prácticamente igual para todas, es preferible consultarla en su oportunidad.

Manejo de procesos

El ciclo de vida de un proceso que habrá de controlarse involucra principalmente las siguientes etapas:

Inicio de un proceso

Para indicar en un programa que se debe iniciar un nuevo proceso, se tienen tres mecanismos principales: `fork`, `system` y `exec`, todos los cuales se definen en `unistd.h`.

- Crear un proceso hijo con `fork` tiene la siguiente firma:

  ```
  pid_t fork(void);
  ```

  Esta función toma los elementos del proceso en ejecución para crear un nuevo proceso (hijo del original) como un duplicado exacto del original, excepto por tener su propio identificador de proceso, reconocer al proceso original como su padre, no heredar los recursos asignados al padre, estado de los contadores de tiempo de atención de la CPU, señales, hilos secundarios, mensajes y recibir como resultado de la función un 0 en vez del número identificador del proceso hijo.

- Crear un proceso ejecutando un comando del sistema con `system` tiene la siguiente firma:

  ```
  int system(const char *command);
  ```

  Este genera un nuevo proceso mediante la ejecución de un intérprete de comandos (por ejemplo, `/bin/sh`) y solicita a ese intérprete de comandos que ejecute un programa. Este programa usará el mismo proceso del intérprete de comandos para su ejecución cuando esté definido por el Shell o podrá lanzar comandos nuevos en caso de tratarse de ejecuciones de programas. Una vez que la ejecución del comando ha terminado, el código de terminación se retorna al Shell y mediante este al programa que invocó `system` en primer lugar.

- Crear un proceso reemplazando la funcionalidad del proceso que invoca por un nuevo programa con `exec` tiene diversas firmas orientadas a soportar distintas formas de representación de los parámetros a pasar al programa que se desea ejecutar, como en:

```
intexecvpe(constchar *file, char *constargv[], char
*constenvp[]);
```

Se proporciona una serie de cadenas terminadas por caracteres nulos a la función, de las que la primera es el nombre del programa que se desea ejecutar, y a continuación se proporciona la lista de atributos, que debe terminarse con un apuntador nulo, y por último se suministra una lista de valores para variables de ambiente, que también debe terminarse con un apuntador nulo.

Verificación del estado del proceso

El proceso padre puede bloquearse en espera de que alguno de los procesos hijos cambie de estado a terminado con `wait`; por ejemplo, la firma puede ser:

```
pid_t waitpid(pid_t pid, int *status, int options);
```

En esta forma, se indica como primer argumento el número de identificación para el o los procesos hijo a verificar. El segundo parámetro `status` es un apuntador al estado del proceso hijo ante el que el proceso padre debe salir del bloqueo. Por último, `options` permite especificar si debemos esperar a un proceso en particular, un proceso del mismo grupo que el especificado por `pid`, cualquier proceso hijo de este o cualquier proceso que pertenezca al mismo grupo que el proceso que espera.

Terminación del proceso

El proceso padre, un proceso con privilegios de administración o el propio proceso, puede terminar la operación de otro enviándole una señal 9 o SIGKILL.

Manejo de hilos

Los hilos tienen un ciclo de vida que consta de preparar los elementos para su creación, su creación y puesta en marcha y su terminación, y se definen en `pthread.h`.

Los elementos que se utilizan para la creación de un nuevo hilo son los atributos. Estos se representan en la estructura `pthread_attr_t`, que debe construirse mediante una función llamada `pthread_attr_init` y en la que pueden definirse las opciones de prioridad y comportamiento del hilo por crear.

La creación del hilo se realiza con `pthread_create`, que tiene la siguiente firma:

```
intpthread_create(pthread_t *thread, constpthread_attr_t
                  *attr,
void *(*start_routine) (void *), void *arg);
```

El primer argumento es un apuntador a la variable donde se almacena el identificador del nuevo hilo, el segundo parámetro es la referencia a la estructura de datos con los atributos o una referencia nula si se desea usar el comportamiento por defecto, el tercer parámetro es la referencia de la función que será ejecutada por el hilo a crear, y por último se tiene un apuntador a un arreglo de apuntadores a los parámetros de la función que habrá de ser ejecutada.

La terminación del hilo puede darse de tres maneras:

1. Si se llega al final de la función que se ejecutará por el hilo y se retorna de ella.
2. Si el hilo hace una invocación a `pthread_exit`, que tiene la firma:

   ```
   void pthread_exit(void *retval);
   ```

   Este permite indicar como parámetro el valor de retorno que debe entregar a la función `pthread_join` que pudiera esperar por la terminación de este hilo.
3. Otro hilo da por terminada la ejecución de este invocando `pthread_cancel` con la firma:

   ```
   intpthread_cancel(pthread_t thread);
   ```

   En este se indica el identificador del hilo al que se envía la solicitud de terminación con el primer parámetro, y el código que retorna la función solo indica si la solicitud pudo enviarse correctamente y no si ha sido atendida.

   Para que un hilo se bloquee en espera de la terminación de otro puede usarse `pthread_join`, que tiene la firma:

   ```
   intpthread_join(pthread_t thread, void **retval);
   ```

El primer parámetro es la identificación del hilo que deseamos esperar y el segundo es una referencia al espacio de memoria donde se pueda copiar el código de salida del hilo que termina. La función retorna 0 si se realiza con éxito o un código de error.

Atención de señales

Las señales se atienden definiendo a las funciones que han de invocarse al recibir una señal en particular mediante la función `signal`, que está definida en `signal.h` y tiene la siguiente firma:

```
sighandler_t signal(intsignum, sighandler_t handler);
```

El primer parámetro indica la señal que deseamos atender mediante la ejecución de la función referida por el segundo parámetro, que también puede usar los símbolos

`SIG_IGN` para ignorar la señal o `SIG_DFL` para volver a atenderla con el comportamiento por defecto.

Programación de alarmas

La programación de alarmas se puede realizar de tres maneras principales, dependiendo del tipo de medición del tiempo por el que deseemos esperar. Para esperar una cantidad de segundos usamos la función `alarm`, que tiene la firma:

```
unsigned int alarm(unsigned int seconds);
```

El parámetro indica el número de segundos a esperar y retorna el número de segundos restantes en caso de que previamente se hubiese programado otra alarma.

Conceptos básicos de planificación de procesos

A continuación se presentan los conceptos básicos de planificación de procesos.

Objetivos y criterios

La planificación de procesos se encarga de decidir cuál proceso debe ser atendido a continuación. Esto debe realizarse siempre que un proceso deje de ser atendido, ya sea porque haya terminado, porque se haya bloqueado (en el caso de los sistemas de tiempo compartido) o porque haya terminado el tiempo de su ejecución.

Los distintos tipos de sistemas operativos, así como los sistemas computacionales que soportan, requieren dar prioridad a algunos objetivos sobre otros; sin embargo, hay una serie de objetivos que son comunes a todos, los cuales se definen a continuación:

- **Desempeño.** Se deben aprovechar los recursos del equipo para atender a los procesos.
- **Estabilidad.** El desempeño y la atención de los procesos debe ser predecible y confiable.
- **Justicia.** Los procesos de la misma prioridad deben recibir aproximadamente el mismo nivel de atención.
- **Cumplimiento de políticas.** Las políticas de distribución de la atención deben ser cumplidas por el mecanismo.
- **Balance.** Todos los recursos del sistema que tengan trabajo pendiente deben mantenerse ocupados.

Importante

♥ **Docente.** Los objetivos generales se contraponen o se apoyan entre sí, según sea la situación. Esto es de gran relevancia para comprender las decisiones tomadas por diversos mecanismos de planificación, así como para identificar todas las situaciones posibles. Por tanto, se le pide plantear una representación gráfica de las relaciones de todos los objetivos pertinentes para al menos un tipo de sistema operativo en particular (por ejemplo, para los sistemas operativos de las computadoras personales).

De acuerdo con la lista anterior, podemos observar un énfasis en los siguientes aspectos, según el tipo de sistema.

Batch. Constituye uno los primeros tipos de planificación en surgir, previos a la adopción del modelo de procesos y orientados a atender en orden lotes de programas (Batches) uno a uno. Por ese motivo se conocen como de procesamiento por lotes o de Batch.

- **Minimizar el tiempo de procesamiento.** Se refiere a minimizar el tiempo que toma procesar un trabajo individual.
- **Maximizar el uso del procesador.** Se refiere a evitar tiempos de inactividad del procesador, como una aproximación al balance. En principio se espera que esto ayude a los objetivos anteriores.
- **Multiprogramación.** Conforme se incrementó la capacidad de procesamiento de los procesadores, las esperas por eventos de entrada y de salida significaban más ciclos de espera, lo que trajo como consecuencia más memoria y recursos de los que necesitaba un solo programa. Por ello, para aprovechar las capacidades del equipo era deseable aprovechar los tiempos de espera de la CPU para atender a los siguientes procesos.
- **Maximizar la productividad.** Se refiere a mejorar el rendimiento del sistema, medido en trabajos terminados en un periodo.

  En ocasiones, mejorar el tiempo en el que termina el conjunto de trabajos genera cargas de trabajo que no tendríamos en caso de atender solo un propósito, por lo que en ocasiones el tiempo de procesamiento de un trabajo se supedita a la productividad de todo el lote.

Tiempo compartido. Conforme la capacidad progresaba y los dispositivos de entrada y de salida permitían que los usuarios interactuaran de manera directa con la computadora en lugar de depender de operadores que ordenaran las tareas en lotes, fue más conveniente repartir la atención del procesador en una serie de procesos que se generan y terminan continuamente. Dadas sus características de repartición de la atención de la CPU, a estos sistemas se les llama de **tiempo compartido**, aunque también se les conoce como sistemas interactivos por su capacidad de interactuar directamente con los usuarios.

- **Tiempo de respuesta.** Se refiere al tiempo que el proceso toma en generar una respuesta ante una solicitud por parte del usuario o de otro proceso. Los procesos no solo realizan una tarea y la terminan, sino que a menudo se encuentran en espera de peticiones, las cuales atienden a lo largo de un periodo. De este modo, el tiempo que toma entre que se recibe la petición y se entrega la respuesta es más significativo en estas que el tiempo de procesamiento total de la tarea.

- **Proporcionalidad.** Consiste en degradar de forma lineal el desempeño del sistema conforme se agrega carga de trabajo. Los usuarios generan nuevos procesos, y conforme la atención de la CPU debe repartirse entre un conjunto mayor de solicitudes, es inevitable que los tiempos de respuesta empeoren; sin embargo, debe evitarse que al tiempo que se incremente la carga, el deterioro crezca de manera exponencial.
- **Atención preferente a los eventos de interfaz de usuario.** En los sistemas diseñados para atender de manera directa a los usuarios finales (como los de computadoras personales), debe darse prioridad a los procesos que interactúan de manera directa con los usuarios a fin de mejorar la experiencia de su uso.

Tiempo real. En los sistemas de control y de manejo de flujos de información (por ejemplo, multimedia), a menudo resulta crítico mantener una frecuencia de atención predefinida para que el sistema operativo pueda cumplir con sus responsabilidades, lo que impone necesidades especiales de planificación, mediante la cual se busca lo siguiente:

- **Cumplir los compromisos de atención.** La planificación debe hacer todo lo posible por mantener la atención de los procesos, para cumplir de manera sostenida los compromisos establecidos en el diseño del sistema.
- **Predictibilidad.** La atención de los procesos debe realizarse a intervalos regulares dentro de variaciones relativamente pequeñas. No es suficiente que el nivel de atención promedio sea el solicitado; la periodicidad de la atención debe mantenerse sin grandes variaciones para mantener la operación estable.
- **Distribuido.** A partir de los sistemas de tiempo compartido, planificar otorga la capacidad de aprovechar los recursos de un conjunto de computadoras conectadas en red con apoyo del sistema operativo, con el objetivo de atender todas las aplicaciones. La forma y el alcance de la participación del sistema operativo varía en gran medida, al igual que la función que desempeña en la administración de los recursos distribuidos en múltiples computadoras independientes con unidad de propósito.

Soporte a los protocolos de red. La comunicación a través de red no sigue patrones ni periodos fácilmente modelables por lo que en algunos periodos deben manejar la transferencia de grandes cantidades de datos y en otros queda sin actividad; sin embargo, la capacidad del almacenamiento intermedio y la velocidad de transferencia es limitada, por lo que se debe garantizar un nivel de atención preferente a los procesos que se encuentren atendiendo las operaciones de red para evitar que durante el tiempo que los procesos no reciban atención de la CPU se pueda acumular tanta información en el buffer del dispositivo que llegue a rebasar su capacidad y resulte necesario des-

echar parte de esta, con lo que diversas operaciones de red fallarían o requerirían intentarse de nuevo.

- **Balanceo de trabajos.** Consiste en dirigir las tareas a los nodos con menos carga de trabajo. Al tener múltiples computadoras con capacidad de procesamiento, se requieren mecanismos para evaluar el nivel de utilización de sus recursos de procesamiento y para repartir la carga que se genere entre los nodos disponibles.

Embebido. Se trata de sistemas operativos con funcionalidad reducida, que se emplean para sistemas de propósito particular. Dependiendo del tipo de dispositivo de que se trate, estos suelen tener limitaciones en cuanto a su capacidad de procesamiento, memoria y dispositivos, por lo que siempre se deben considerar las limitaciones en la potencia que consumen y las comunicaciones en casos especiales y medios hostiles. Algunos ejemplos son los sistemas incluidos en los automóviles, los teléfonos celulares (sean smartphone o no), satélites artificiales y sondas de exploración espacial, así como los sistemas de monitoreo marítimo, como los usados para las alarmas de tsunamis (por ejemplo el Deep-ocean Assessment and Reporting of Tsunamis [DART]), el creciente número de aparatos eléctricos con capacidad de red, conocidos como Internet of things, entre muchos otros. De hecho, se espera que sean los sistemas de cómputo más frecuentes en la actualidad, aunque muchos de ellos aún no son reconocidos por los usuarios como sistemas de cómputo.

Reducción del overhead. Dada la limitación de recursos, resulta indispensable minimizar la carga generada por tareas administrativas, ya sea que se economice tiempo en la CPU al evitar que haya turnos de atención y ciclos de la CPU al considerar que los procesos en vez de terminar y reiniciar, se pueden mantener en memoria entre ejecuciones. Otras medidas que se pueden llevar a cabo son tan variadas como diversas son las aplicaciones de estos dispositivos.

Importante

♥ **Usuario.** Para comercializar las habilidades de un profesional de cómputo es conveniente elegir un área de especialización.

De los tipos de sistema descritos con anterioridad, ¿cuáles se usan más y con mayor predominio en tu área de interés?, ¿cuál te resulta más atractivo?

♦ **Líder.** ¿Comprendes las necesidades de las aplicaciones en los diversos tipos de sistemas? Describe cómo afecta eso a las técnicas de análisis y diseño o a los ciclos de vida de estas.

♠ **Arquitecto.** Muchos sistemas operativos se basan en UNIX, específicamente Linux. ¿A qué crees que se deba? ¿Qué impacto tiene eso en las herramientas de desarrollo que un profesional debe conocer para acceder al desarrollo en esas plataformas?

3.5 Algoritmos de planificación de procesos

Planificación por lotes (Batch)

Algunos ejemplos de estos son los FMS de IBM, IBSYS y el UMES, de la Universidad de Michigan. En un principio estos procesaban de forma secuencial, pero conforme las

capacidades de los equipos de cómputo progresaron, los programas que se desarrollaron originalmente para aprovecharlos ya no utilizaban de manera plena lo que ofrecían las nuevas computadoras, no obstante las considerables inversiones que representaban estos equipos, por lo que se buscaron mecanismos para aprovechar los ciclos de procesador que quedaban ociosos en lo que los procesos se bloqueaban, esperando operaciones de entrada y de salida, y la memoria disponible que excedía de lo requerido por el proceso heredado de generaciones anteriores de computadoras, lo que condujo a los sistemas de multiprogramación por lotes.

En la actualidad aún se conoce como procesamiento por lotes a la realización de una serie de operaciones que se especifican en algún lenguaje interpretado, como los propios de los Shell de UNIX o los Batch Files de Windows. Esto puede tender a confusión, pero no debe pensarse que por compartir el nombre los sistemas operativos con intérpretes de comandos operen de manera similar a los sistemas Batch de antaño.

Multiprogramación por lotes (Multiprogrammed Batching)

Agregan capacidades de multitarea a los primeros sistemas operativos de procesamiento por lotes; un ejemplo es el OS/360 de IBM.

Una de las primeras formas de multiprogramación por lotes consistía en cargar más de un programa a la memoria, que ahora podría contener varios procesos a la vez, y utilizar los tiempos de bloqueo del proceso que se estaba ejecutando, con el fin de avanzar al proceso siguiente del lote que tenía que procesarse. Estos turnos permitían reducir de forma considerable los tiempos ociosos de la CPU y aprovechaban óptimamente la memoria disponible, aunque aún no llegaban a los ideales de multiprocesamiento de la época planteados por el PARC (Palo Alto Research Center), que aún distaban de ser implementados cabalmente.

Primero en llegar primero en ser atendido (First In First Out, FIFO)

En su forma más sencilla, lo primero que hace este algoritmo de planificación de procesos es construir una cola de trabajos con los programas, junto con las instrucciones que indican cómo preparar el equipo para ejecutarlos. Luego, toma estos trabajos en orden y los ejecuta en su totalidad. Además de su simplicidad, tiene la ventaja de que al dedicar todos los recursos del equipo al trabajo que se está atendiendo en ese momento, se garantiza que este terminará en el menor tiempo posible.

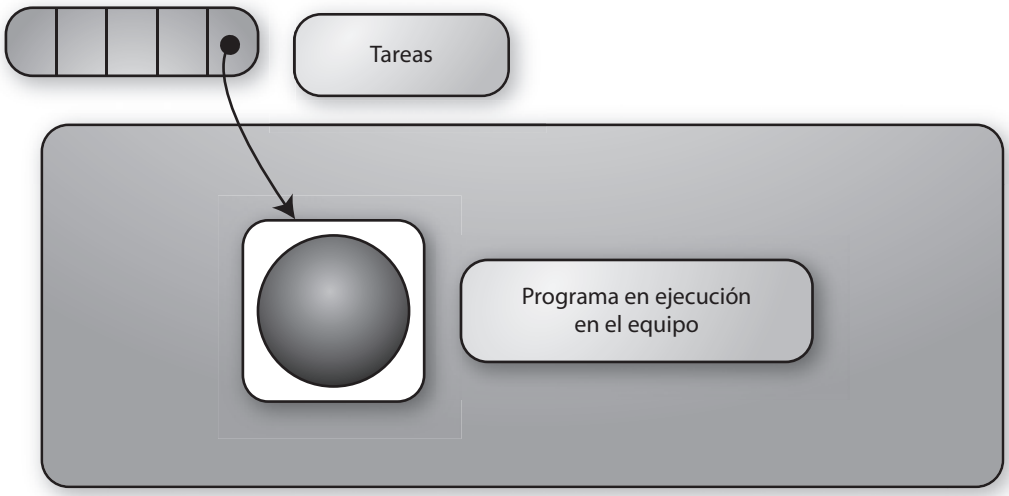


Figura 3.2 Fila de atención First In First Out.

A pesar de todas sus bondades, FIFO también tiene algunas desventajas, de las cuales la principal es que en caso de que el trabajo que está siendo atendido no tenga un buen balance, entonces se incrementan los tiempos de ejecución del conjunto de trabajos, debido a los tiempos muertos de los recursos. Es decir, si una tarea no mantiene ocupados los recursos del equipo (por ejemplo, por requerir de múltiples operaciones de entrada, de salida, o ambas, durante las cuales no puede aprovechar la atención de la CPU), el tiempo que tomará completar la siguiente tarea tendrá que incluir todos los tiempos de espera de todos los procesos anteriores.

Trabajo más corto a continuación (Shortest Job Next)

Si se toma un histórico de los tiempos de ejecución de los diversos trabajos, podemos reordenar la cola para procesar primero los trabajos más cortos que sean de la misma prioridad. Esto no modificará el tiempo que tomará ejecutar el trabajo individual, pero sí reducirá el tiempo promedio que tendremos que esperar para que un trabajo pueda entregar sus resultados, lo que facilitará que los dispositivos de entrada y de salida distribuyan la carga y aprovechemos un poco mejor el equipo.

No obstante, entre sus inconvenientes se tiene la inherente dificultad de predecir el tiempo de procesamiento de las tareas, ya que este no suele ser constante, además de que depende de que todas las tareas se ingresen de forma simultánea. Si durante la ejecución de la cola se agregan nuevas tareas, se corre el riesgo de que las tareas que requieren más tiempo de procesamiento sean enviadas de manera continua al final de la lista, e incluso que no sean atendidas.

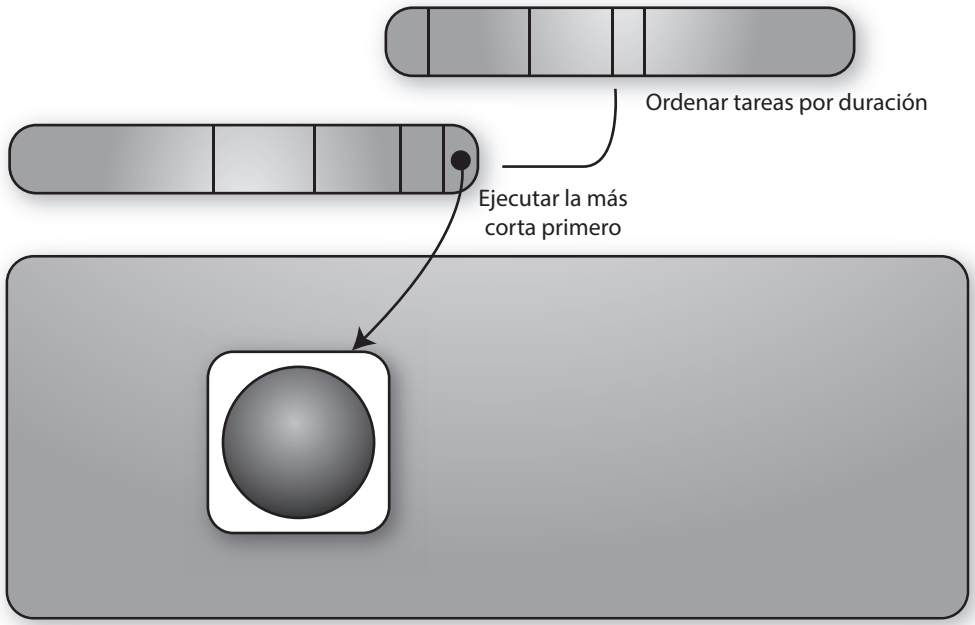


Figura 3.3 Atención de tareas por Shortest Job Next.

Para poder aprovechar los tiempos de bloqueo de los procesos durante sus operaciones de entrada, de salida, o ambas, necesitamos mejorar nuestra administración de tareas para incluir el modelo de procesos y poder alternar la atención del procesador entre diversos trabajos en ejecución. A estos sistemas también se les conoce como **preemptive** por su capacidad de alternar la atención de la CPU sin efectos adversos.

Importante

♥ **Docente.** ¿Cuál es la relevancia de estos modelos, ahora anticuados, de planificación de tareas?

Tiempo compartido (Time share) o interactivos

Los interactivos añaden multitarea y capacidades de interacción con el usuario y entre procesos en el sistema operativo; por lo general, estos se incluyen en sistemas operativos modernos como UNIX, Linux y Windows.

Primero el de menor tiempo restante (Shortest Remaining Time First)

Es una versión preemptive de Shortest Job Next, también conocido como Shortest Remaining Time Next. En este el proceso con el menor tiempo estimado para completar su

ejecución es el elegido para recibir la atención de la CPU con intención de terminarlo lo antes posible. Sin embargo, al igual que Shortest Job Next, corre el riesgo de que los procesos con tiempos de ejecución más largos queden sin atención de la CPU.

Mejor tasa de respuesta a continuación (Highest Response Ratio Next)

Para corregir los problemas de omisión en la atención de los procesos largos, Brinch Hansen realiza un refinamiento de Shortest Remaining Time Next, en el que se da preferencia a los procesos tanto por el tiempo estimado restante de ejecución como por el tiempo que ha pasado en espera de atención. Esto es, conforme los procesos esperan más tiempo adquieren mayor prioridad, lo que impide que puedan postergarse por tiempo indefinido.

Turno circular (Round Robin)

Constituye la política más sencilla de asignación de turnos de atención de procesos al tener multitarea. Una vez que se conoce la lista de procesos que habrá de atenderse, mediante **Round Robin** se alterna la atención hacia todos estos procesos, en secuencia y por igual. Al hacerlo de esta manera, los procesos que se encuentran bloqueados no interfieren con la habilidad de los demás de recibir atención del procesador y los procesos cortos tienen la oportunidad de progresar hasta terminar y liberar los recursos que ocupan. Una de sus principales características es que no muestra inconvenientes por agregar nuevos procesos después de iniciar la planificación con otro conjunto de procesos.

Pero su principal limitación consiste en que si tenemos una única cola de procesos por atender, no tiene elementos para distinguir distintas prioridades entre estos y dar atención preferente a un subconjunto de procesos críticos.

Round Robin de múltiples colas (o multinivel)

Con el fin de corregir la limitación de Round Robin para distinguir niveles de prioridad, se puede realizar una implementación donde en vez de una cola de procesos por atender se tenga una serie de colas de procesos, donde cada una de estas corresponda a una prioridad diferente. En este caso, para asignar la atención de la CPU se recorre la lista de prioridad más alta; así, una vez que se han atendido todos sus integrantes, se procesa a un integrante de la lista de la prioridad siguiente; posteriormente, se repite la

atención de todos los integrantes de la lista de nivel superior antes de atender el siguiente proceso, y así sucesivamente.

Esta forma de planificación tiende a dejar sin atención a los procesos de prioridades bajas conforme se incrementa el número de procesos en las prioridades superiores, lo que se puede evitar si se definen pocos niveles de prioridad distintos o se manejan solo procesos que no requieran atención frecuente del procesador en las prioridades inferiores.

De hecho, la mayoría de los sistemas operativos actuales basan su planificación en Round Robin, agregando refinamientos para implementar sistemas de prioridades, dar atención a aplicaciones que la requieren en tiempo real y evitar que los procesos pasen largos periodos sin atención.

Asignación garantizada

Otro enfoque que es posible tomar consiste en tratar de garantizar la asignación de una proporción del tiempo de la CPU a los procesos con base en sus prioridades. Un algoritmo de este tipo, con la planificación de **lotería**, se implementó en el sistema operativo Minix. En la **asignación garantizada**, un periodo se divide entre el número de turnos de atención que se espera que el procesador pueda realizar durante este. A cada uno de estos turnos de asignación se les otorga el equivalente de un boleto de rifa, los cuales se distribuyen entre los procesos en función de sus prioridades relativas.

De este modo, los procesos de una prioridad menor tienen un número menor de turnos, mientras que los procesos de prioridades mayores tienen un mayor número de turnos, con lo que se asegura que estos reciban más atención del procesador y en una proporción fácilmente predecible.

Así, cuando se hace una asignación de turno de atención, la planificación de procesos debe elegir uno de los boletos al azar, el cual corresponderá al proceso al que se asignó el boleto seleccionado, después de lo cual se retira el boleto, y al término del turno de asignación se elige otro para determinar el siguiente proceso que habrá de atenderse.

Eventualmente, todos los turnos serán seleccionados y se cumplirán todos los procesos que requieren atención como se habían asignado; acto seguido, se genera un nuevo sorteo con los procesos que aún estén activos y se continúa la operación.

Esta implementación no es complicada, aunque la realización del sorteo suele consumir una cantidad significativa de atención de la CPU, con lo que se genera una sobrecarga inconveniente.

Importante

- **Arquitecto.** ¿En cuál de las anteriores se basa la planificación de su sistema operativo de uso cotidiano? ¿Qué refinamientos hace sobre esta planificación?
- **Líder.** ¿Qué efectos apreciables tiene el modelo de planificación que empleas sobre la operación de los procesos en el sistema operativo que usas de manera cotidiana?

Tiempo real (Real Time)

Añaden elementos de predictibilidad en la frecuencia de atención de los procesos en el sistema operativo al planificar la atención de las tareas de acuerdo con los compromisos establecidos con estas. Algunos ejemplos de estos son VxWorks, de WindRiver, y DART OS (Data Access in Real Time), de EMC[2].

Por lo regular, los sistemas de tiempo real deben reaccionar a eventos de tiempo real de forma predecible en tiempos cortos, en vez de orientarse a la cantidad de trabajos que puede procesar en un periodo largo o a atender a un máximo de procesos manteniendo la ilusión de un multiprocesamiento, pues estos sistemas se enfocan en cargas de trabajo moderadas para las que resulta muy importante que se atiendan en periodos conocidos.

Los tipos de procesos para atender en sistemas de tiempo real se clasifican como sigue:

Eventuales (o **Terminating Process**). Procesos que definen sus requerimientos de tiempo de atención en función de lo que toma su carga; es decir, todo el procesamiento hasta la terminación.

Persistentes (o **Nonterminating Process**). Estos definen sus requerimientos de tiempo de atención con base en eventos a los que deben responder con determinada frecuencia; los sistemas de control digital y los de multimedia se incluyen en esta categoría. A su vez, estos procesos pueden ser clasificados, según su periodicidad, en periódicos y aperiódicos.

- **Periódicos.** Se refiere a los procesos que tienen una frecuencia determinada, que deben seguir para atender sus eventos. Por ejemplo, un programa decodificador de MP3 para audio puede necesitar llenar el búfer de la tarjeta de sonido cada medio segundo o menos, ya quede lo contrario se generarán distorsiones o lagunas en la reproducción. Si un sistema de tiempo real puede verificar que todos sus procesos son periódicos y que el tiempo que le toma atender las necesidades de un conjunto de estos no excede a su capacidad de procesamiento en un periodo determinado, se dice que es **planificable** (**shedulable**) y se prefiere para atender aplicaciones en las que cumplir los compromisos de atención sea muy importante.
- **Aperiódicos.** Se refiere a los procesos que reaccionan a eventos que no tienen una frecuencia determinada. Es importante destacar que estos no garantizan que el procesamiento requerido en una ocasión estará dentro de un rango de duración limitado; de este modo, para fines de análisis, se toma el periodo mínimo esperado de los eventos y los tiempos máximos esperados de procesamiento.

En la actualidad, las implementaciones de sistemas operativos de tiempo real deben modificar los mecanismos de planificación para responder mejor a estas necesidades de las aplicaciones.

De acuerdo con la capacidad final que exhiben estos para responder de forma predecible a los eventos se les clasifica como:

- **Estrictos (Hard Real Time).** Son aquellos que pueden garantizar el cumplimiento sostenido de compromisos de atención a aplicaciones periódicas.
- **Laxos (Soft Real Time).** Son aquellos que pueden cumplir la mayoría de sus compromisos y deben usarse para aplicaciones en las que el esporádico fallo en cumplir algún compromiso no es crítico, como es el caso de muchos sistemas multimedia.
- **Primer compromiso (Earliest dead line).** Se trata de una implementación sencilla en la que cada proceso indica al planificador el tiempo máximo en que requiere completar su ejecución si es eventual, o su siguiente evento si es persistente. Entonces se asigna la atención del procesador al proceso que tenga el menor tiempo disponible. Solo se interrumpe un proceso cuando se recibe un nuevo proceso, y este tiene un tiempo límite aún menor del que se está atendiendo en ese momento.

  En este caso, para poder asegurar que todos los procesos sean atendidos debe cumplirse que la suma del tiempo de procesamiento de todos los procesos sea menor que la capacidad del procesador para el periodo disponible en los compromisos.
- **Análisis de tasa monótona (Rate Monotonicanalysis).** A menudo se parte de planificadores de sistemas de tiempo compartido para implementar los de tiempo real. Por tanto, en estos casos se puede asignar una prioridad a los procesos periódicos e instruir al planificador para dar siempre el turno de atención al proceso de más alta prioridad que no se encuentre bloqueado. Los procesos que tengan la misma prioridad se deben atender en Round Robin.

  Para asignar las prioridades se ordenan los procesos según la frecuencia de atención que requieren. De este modo, a aquellos que requieran la máxima frecuencia se les asigna la prioridad más alta, a los que tengan la frecuencia más alta después de esa la prioridad siguiente, y así sucesivamente.

Los sistemas operativos de tiempo real basados en Linux se sustentan en dedicar las frecuencias más altas de la planificación a los procesos de tiempo real y utilizar esta técnica de planificación.

Importante

♦ **Líder.** Hoy día, prácticamente todos los sistemas operativos de cómputo personal están preparados para manejar multimedia con muy alta calidad. ¿Qué previsiones tienen esos sistemas operativos para procesos "de tiempo real" a pesar de no tener planificadores de tiempo real?

Distribuido

En un capítulo posterior de este libro se discute el impacto y las consideraciones específicas para varias tareas del sistema operativo, por lo que aquí solo nos limitaremos a mencionar algunos de los refinamientos acerca de la planificación de tareas, por lo que es pertinente hacer una clasificación general de los sistemas distribuidos según su alcance.

Sistemas operativos de red

Se refiere a los sistemas en los que cada equipo opera de manera autónoma, pero expone un conjunto de recursos a sus aplicaciones, los cuales se apoyan en el grupo de computadoras que participan en el sistema distribuido. Por lo general cuentan con sistemas de archivos, impresión, monitoreo de operación y de manejo de credenciales que complementan los recursos locales; sin embargo, se hacen pocos esfuerzos para ocultar el hecho de que muchos de estos recursos residen en otras computadoras y cada una puede operar a cierto nivel de forma autónoma.

La planificación de tareas en este tipo de recursos necesita reconocer las demandas de los dispositivos implementados por software que hace uso intensivo de la red de comunicaciones para garantizar la atención oportuna de estos eventos de E/S y la coordinación de las tareas relacionadas; Windows y Linux entran en esta categoría.

Sistemas operativos distribuidos

Estos exponen todos los recursos de las computadoras que participan del sistema como una sola computadora con las características reunidas de sus integrantes. De este modo, la planificación de tareas requiere ampliar la funcionalidad de sincronización y comunicación de procesos a fin de considerar tareas que se realizan en otras computadoras, para lo que debe prestarse especial atención a la sincronización de información entre varios equipos para la memoria compartida y para el paso de mensajes entre los sistemas. Asimismo, también se requieren mecanismos para distribuir las tareas de procesamiento entre los nodos, reunir sus resultados e incluso migrar tareas en ejecución entre computadoras, mismos que se describen a continuación.

- **Clúster.** Se trata de grupos de computadoras trabajando al unísono en los que sus sistemas operativos cuentan con una administración centralizada de las tareas. Reciben este nombre y se distinguen del resto por el alto acoplamiento que presentan las tareas en los diversos equipos. Suelen tener alta disponibilidad y poner énfasis en el balanceo de carga; algunos ejemplos de estos son Plan 9, Clusters Beowulf y Linux Virtual Server.

- **Grid.** Se trata de coaliciones de sistemas heterogéneos y dispersos geográficamente que pueden reunir sus recursos para atender aplicaciones. Difieren de los clusters en que no requieren el mismo equipo o sistema operativo en todos los nodos y en que no oculta por completo la naturaleza distribuida del sistema, lo que tiende a orientar la carga hacia trabajos no interactivos. Para reunir los recursos y exponerlos a las aplicaciones suele valerse de Middleware especializado. La arquitectura suele incluir protocolos estándar de comunicación que soportan las tareas de distribución del trabajo, detección, coordinación y balanceo de los nodos participantes, resistencia a fallos e interoperabilidad con las plataformas que alberguen los nodos trabajadores.

  Algunas plataformas de cómputo notables que emplean esta estrategia son Red BITCOIN, que alcanzó 1166652 petaflops en junio del 2014; BOINC, que procesó en promedio 9.2 petaflops en marzo del 2013, y Middleware Globus Toolkit, que se usa en universidades e instituciones gubernamentales como la European Grid Infrastructure, la Nordic Data Grid Facility y la Grid del Instituto Nacional del Cáncer de la Organización de la Naciones Unidas, entre otros.
- **Cloud Computing.** Consiste en proporcionar los recursos a las aplicaciones como servicios y no como productos. Es decir, los recursos que son proporcionados a las aplicaciones se obtienen mediante servicios basados en protocolos estándar y pasan de un modelo de consumo tradicional de adquisición de la infraestructura o de participación voluntaria (como en Grid) a un modelo de pago por consumo análogo al de los servicios urbanos, como la energía eléctrica o la red de distribución de agua potable. Así se pueden contratar cantidades variables según la demanda de capacidad de procesamiento, almacenamiento, capacidad de transferencia de información, etc. Con ello se logra un nivel de tolerancia a fallos y escalabilidad sin precedentes a disposición de las aplicaciones, aunque eso requiere el uso de técnicas de diseño por parte de las propias aplicaciones que sepan explotar estas capacidades. Este tipo de arquitecturas ha encontrado mercado en el soporte de las aplicaciones de plataformas de cómputo embebido por la dificultad de predecir los patrones de carga y los enormes números de clientes con los que cuenta. Aplicaciones de video con sobredemanda, como YouTube, soportado por la plataforma de sistemas distribuidos de Google con GFS, así como aplicaciones como MapReduce y BigTable, y de mensajería instantánea como WhatsApp, que es soportada por los servicios de SOFTLAYER (filial de IBM), son ejemplos de estas.

Importante

♠ **Arquitecto.** Los sistemas distribuidos necesitan arquitecturas complejas para dividir los requerimientos en partes manejables para el desarrollo. ¿Qué técnicas, herramientas y prácticas se tienen para apoyar al profesional en esta tarea?

♥ **Usuario.** Para el profesional interesado en una carrera de desarrollo de software, elegir una plataforma en la cual especializarse es una forma de incremen-

tar su atractivo para futuros contratantes. Hoy día, existe un gran mercado en materia de sistemas distribuidos, pero también una gran variedad de plataformas. ¿Cuál plataforma te resulta atractiva o qué mercado piensas seguir?

En general, la tendencia de los sistemas distribuidos se ha encaminado a disminuir el acoplamiento entre los componentes de software y buscar una mejor interoperabilidad y tolerancia a fallos, lo que ha hecho que la mayoría de los desarrollos comerciales tiendan hacia la nube en vez de hacia clusters de computadoras. Sin embargo, diversas aplicaciones especializadas siguen construyendo clusters para atender a sus necesidades, como el **rendering** de animaciones para cine y televisión.

Embebido (Embedded)

Estos sistemas, de acuerdo con sus aplicaciones, suelen usar sistemas operativos de tiempo real o de tiempo compartido, realizando modificaciones para ajustarse a las necesidades particulares de sus aplicaciones. Por ejemplo, la mayoría de los sistemas de planificación para sistemas móviles incorporan consideraciones del nivel de la batería para disminuir la velocidad del procesador y ahorrar energía.

Por ejemplo, el sistema operativo Android se basa en un *kernel* de Linux; sin embargo, no expone la funcionalidad de creación de procesos y manejo de prioridades como un sistema Linux. Por el contrario, este solo usa la planificación de procesos para ejecutar una máquina virtual Java modificada que recibe el nombre de Dalvik VM. Todos los servicios y las aplicaciones de usuario se generan a partir de esta mediante operaciones `fork`, a las cuales se les asignan identificadores de usuario distintos para proteger sus estructuras de datos.

El ciclo de vida de los procesos también se modifica. Se considera que todos los procesos son persistentes, por lo que se definen estados adicionales para la ejecución. Son motivo de particular interés los siguientes:

- **Pausada.** Aplicaciones que no se presentan en la zona principal de la pantalla. Este estado es análogo al de **Listo**, ya que no debe recibir atención de la CPU; sin embargo, puede generar hilos especiales para mantener procesamiento sin acceso a la interfaz de usuario y eventos que la regresen a ejecución.
- **Detenida.** Aplicaciones que están completamente ocultas y detenidas, también análogo a **Listo**, aunque en el caso de estas no se espera que reciban atención de la CPU hasta que un evento las retire de este estado. A diferencia de aplicaciones en sistemas interactivos, las aplicaciones que han finalizado su trabajo se dejan en este estado en vez de finalizarlas, lo que se hace con el objetivo de ahorrar ciclos de la CPU y de batería la próxima vez que se requieran, ya que se evita reprocesar los estados de inicio.

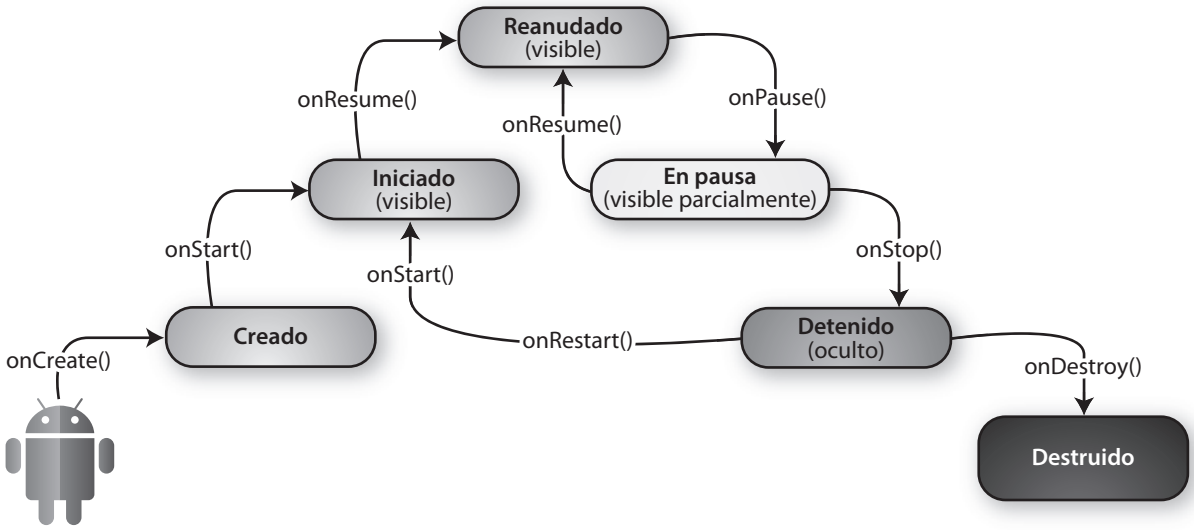


Figura 3.4 Estados de un proceso en Android.

- **Destruida.** En este caso, en lugar de alcanzar el estado de finalización auténtico de una aplicación cuando los hilos de ejecución llegan al final de su procesamiento, se alcanza cuando el sistema operativo detecta que se requiere liberar memoria. De este modo, cuando la aplicación está detenida se envía una señal `onDestroy` para que la aplicación almacene su información de estado y posteriormente libera los recursos de esta, destruyéndola efectivamente.

En el caso de iOS de Apple hay diversas versiones de *kernel*, derivadas del proyecto de código abierto de núcleo llamado Darwin, y que a su vez son descendientes del derivado de UNIX Free BSD, tomando algunos conceptos de NEXTSTEP. Su modelo de planificación de procesos, aunque se mantiene privado en sus detalles, es muy similar al de los demás UNIX de tiempo compartido usando procesos e hilos `preemptive`. Sin embargo, no está exento de optimizaciones para soportar las aplicaciones de dispositivos móviles.

En muchos casos se recomienda a los desarrolladores no usar la API POSIX de hilos y en su lugar utilizar el encapsulamiento de los objetos para enviar las actividades, dentro de los procesos, que requieran concurrencia a colas de atención que pueden ser concurrentes y que son atendidas por conjunto de hilos manejados por el propio sistema operativo, con lo que se ahorra el trabajo de la administración

Importante

♥ **Docente.** Realicen en grupo un trabajo de investigación acerca de algún tipo de sistema computacional de relevancia actual en el que se destaquen las características de diversos sistemas operativos. Los siste-

mas de código abierto constituyen una buena opción, ya que son mucho más abiertos y explícitos respecto a las características que implementan.

y la sobrecarga de la creación, así como la destrucción de los hilos de ejecución a las aplicaciones.

Es notable que la parte de administración de procesos en iOS ha sido uno de los aspectos con cambios más rápidos en el desarrollo de los sistemas operativos para smartphones, y es casi seguro que haya elementos novedosos en muy poco tiempo.

EVALUACIÓN

3.1 Explica las diferencias entre proceso, programa e hilo (`thread`).

3.2 Enumera los tipos de planificación que existen y explica sus diferencias principales. Identifica a cuál pertenece su plataforma de desarrollo habitual.

3.3 Escribe un programa que requiera hacer uso concurrente de dos recursos `non-preemptive` por al menos dos hilos de ejecución. Analiza y resuelve las condiciones de carrera y los posibles interbloqueos. Realiza pruebas y justifica tus resultados.

3.4 Explica la relevancia de UNIX en el desarrollo del modelo de procesos.

3.5 Realiza un trabajo de investigación acerca de las mejoras de la última versión de un sistema operativo de uso común. ¿Cuáles de esos cambios afectan a la planificación o la implementación de procesos e IPC?

Referencias bibliográficas

Brookshear, J. Glenn, *Computer Science an Overview*, Pearson/Addison-Wesley, 9a edición, 2007.

Guide to the Software Engeneering Body of Knowledge, versión 3.0, 2014.

Stallings, William, *Operating Systems Internals and Design Principles*, Pearson, 5a edición, 2008.

Tanenbaum, Andrew S., *Modern Operating Systems*, Person Education International, 3a edición, 2009.

Referencias electrónicas

The Linux Information Project, Kernel Mode Definition
http://www.linfo.org/kernel_mode.html

Windows Dev Center, User Mode and Kernel Mode
http://msdn.microsoft.com/en-us/library/windows/hardware/ff554836(v=vs.85).aspx

Documentación de la librería Linux **procps**:
http://fossies.org/dox/procps-3.2.8/readproc_8h_source.html

Paul Krzyzanowski, Real Time Process Scheduling, 2012,
https://www.cs.rutgers.edu/~pxk/416/notes/08-rt_scheduling.html

Wikipedia, Barrier:
http://en.wikipedia.org/wiki/Barrier_(computer_science)
Computer Cluster:
http://en. wikipedia.org/wiki/Computer_cluster,
http://en.wikipedia.org/wiki/Grid_computing, http://en.wikipedia.org/wiki/Cloud_computing

Google Research:
http://research.google.com/

VikramShukla Semaphores in Linux, 2007 en O'Reilly linuxdevcenter:
http://www.linuxdevcenter.com/pub/ a/linux/2007/05/24/semaphores-in-linux.html?page=1

Sven Goldt, *et al*., The Linux Programmer's Guide, Version 0.4, marzo de 1995:
http://www.tldp.org/LDP/lpg/lpg.html

COMPETENCIAS A DESARROLLAR

- Identificará los componentes y características principales de los diversos tipos de dispositivos que participan en la implementación de la memoria en una computadora moderna.
- Comprenderá la importancia e influencia de las características de los sistemas de información, como el multiprocesamiento y las limitaciones de potencia y del costo en el desarrollo de la administración de memoria, así como las prestaciones que esta proporciona a las aplicaciones.
- Comprenderá y será capaz de identificar las características propias de una administración de memoria por paginación y por segmentación, y de los esquemas híbridos que combinan ambas.
- Comprenderá la importancia del concepto de vecindad en el desarrollo de la memoria virtual y de la memoria caché.
- Identificará los principales mecanismos de protección orientados a soportar el multiproceso y sus implicaciones para la seguridad informática.

Administración de memoria

4

¿QUÉ SABES...?

1. ¿Qué es la memoria?
2. ¿Qué es la memoria de una computadora?
3. Cuando se adquieren los componentes para ensamblar una computadora personal, ¿qué concepto se tiene de la memoria?
4. ¿Es lo mismo memoria que disco duro?
5. Cuando se opera el equipo, ¿qué significa que se acabe la memoria?
6. ¿Qué dispositivos en la computadora tienen memoria integrada, a menudo memoria caché?
7. ¿Para qué sirve la memoria caché?
8. ¿Qué relación existe entre los procesos y la memoria?
9. ¿Las aplicaciones se comportan igual en una PC que en un smartphone?
10. ¿Cómo afectarán esas diferencias al manejo de la memoria?

4.1 Introducción

En primera instancia, y debido a los importantes avances tecnológicos de los que somos testigos día con día, como usuarios de sistemas de cómputo y como personas necesitamos tener un conocimiento funcional acerca de lo que es la memoria en el ámbito computacional y cómo funciona, así como de su capacidad de almacenamiento de información. Para ello es necesario comenzar por establecer y definir algunos conceptos importantes respecto a lo que constituye la memoria en las computadoras, así como de algunos aspectos que apoyan su funcionamiento.

Datos

Los **datos** son abstracciones que reducen la incertidumbre. Su unidad básica es el bit, que puede tomar únicamente dos valores: 0 y 1.

Podemos comparar esto con el hecho de describir si una puerta está abierta o no; en este caso, cuando no tenemos ningún dato, no podemos saber en qué estado se encuentra la puerta: abierta o cerrada. La situación de abierta o cerrada se puede representar con un solo bit de información, con lo que resolveríamos la incertidumbre original.

Información

La **información** la constituyen el conjunto de datos contenidos en un mensaje e interpretados en un contexto.

Una vez que la incertidumbre se resuelve en el marco de un contexto definido, los datos cobran un significado, ya que hasta ese momento se sabe qué es lo que representan estos. En el ejemplo anterior, el contexto consistía en conocer si una puerta en particular se encontraba abierta o cerrada. En ese orden de ideas, el mismo dato bajo contextos distintos podría reducir la incertidumbre respecto a cuestiones completamente diferentes.

Conocimiento

El **conocimiento** es información que conduce a un agente a la acción.

Una vez que los datos se ubican dentro de un contexto, un agente, ya sea humano o automático, puede actuar en función de dicha información para modificar su entorno. Tradicionalmente, el conocimiento se considera un atributo humano; no obstante, hay razones para empezar a considerar a algunos sistemas informáticos como capaces de interpretar el contexto, como auténticos agentes o al menos implementaciones de conocimiento. Es importante mencionar aquí que existen muchas otras definiciones de conocimiento, junto con toda una rama de la filosofía, que es la epistemología. La definición antes mencionada es la que resulta más útil para nuestro tema por las múltiples actividades y creciente operatividad del contexto en los sistemas informáticos, ya que nos permite distinguir la información que dirige el comportamiento (como los programas) de aquella que es transformada por ellos de una forma pasiva, que sería mera información.

Máquina universal de Turing

Esta definición está más allá de los alcances de este libro. Sin embargo, hay dos elementos que es importante rescatar y considerar. La máquina universal de Turing es un modelo matemático que permite simular el funcionamiento de cualquier computadora digital y en cuya definición se emplean descripciones tanto de las transiciones de estado (es decir, de las operaciones que aplicará) como de la serie de símbolos que serán operados. Ambas definiciones se representan como secuencias de símbolos a las que se denomina de modo coloquial como cintas.

Sus implicaciones en las computadoras digitales, y en particular en la memoria de estos sistemas, radican en que esta máquina constituye el antecedente directo de la

capacidad que tenemos los humanos de generar mecanismos que pueden recibir datos que actúan como instrucciones, con base en las cuales deberemos transformar más datos proporcionados al equipo, lo cual es, en esencia, lo que hacen las computadoras actuales, tal como estudió Hao Wang. Es importante destacar que a la fecha aún no se ha podido desarrollar algún dispositivo que controle su comportamiento con la información proporcionada, aunque quizá la computación cuántica representa el primer candidato prometedor para incursionar fuera de este modelo de computadora. En dicho modelo, las cintas con series de símbolos que las operaciones pueden recorrer y procesar constituyen la memoria y ponen de manifiesto la necesidad de almacenar a lo largo del tiempo el estado de los datos, para concluir la secuencia de operaciones que se han definido.

Memoria de computadora

Reciben este nombre los dispositivos físicos que se usan para almacenar los datos, que actualmente suelen ser circuitos de semiconductores y dispositivos de almacenamiento magnético. Los datos que almacenan pueden ser programas o información que habrá de procesarse. Asimismo, estos constituyen la implementación de la memoria de una máquina universal de Turing, aunque no tienen una capacidad ilimitada.

4.2 Organización de la memoria

El modelo de máquina universal de Turing considera que su memoria es infinita y que no tiene ninguna limitación en cuanto al tiempo físico que toman las operaciones o el periodo que se conserven los valores en esta. Solo se preocupa de que el número de operaciones sea finito, con lo que se garantiza que una implementación sí habrá de terminar en el tiempo y no requerirá de una cantidad infinita de energía, por lo que sabemos que sí será realizable.

Gracias al principio de Launder podemos asegurar que es imposible construir una memoria infinita funcional, ya que dicho principio indica que existe una cantidad mínima de energía necesaria para modificar un bit de información en un sistema físico. Así pues, al no tener ninguna fuente de energía infinita ni algún sistema que pueda manejarla, podemos aseverar que la memoria que podemos modificar y, por tanto, la capacidad de almacenamiento de la información, siempre serán limitadas.

Asimismo, es importante aclarar que también tenemos limitaciones respecto a la velocidad con la que podemos transferir la información. De este modo, conforme incre-

mentamos la velocidad, la pérdida de potencia en calor se incrementa y los circuitos son progresivamente más sensibles a señales electromagnéticas con frecuencias que se acercan cada vez más a las de operación de nuestros circuitos. Hoy día, las memorias de computadora presentan un límite, debido a que la longitud de los buses de memoria coincide con la longitud de una antena para las frecuencias de operación, lo que genera una máxima sensibilidad al ruido electromagnético del ambiente. Esta limitación no podrá superarse hasta que se cuente con arquitecturas con diferencias considerables respecto a las actuales.

Por lo que se refiere a los requerimientos de nuestras aplicaciones, en la actualidad necesitamos que el almacenamiento de la información atienda a las necesidades de múltiples procesos, a fin de proporcionar diversos grados de protección a la información de un proceso para que no sea dañada por la injerencia, ya sea accidental o maliciosa, de otros procesos. También debemos tener en cuenta que una parte considerable de la información debe sobrevivir a la ejecución de los procesos para que los sistemas de información puedan apoyar actividades humanas que excedan en duración a una sesión de trabajo.

4.3 Objetivos: Memoria ideal

Los objetivos de la administración de la memoria y la implementación final que reciba esta dependerán del tipo de sistema en el que opere; sin embargo, existen algunos objetivos generales que perseguimos en todos los sistemas y que pueden exponerse como características deseables en una memoria ideal:

- **Capacidad de almacenamiento ilimitada.** Sea cual fuere la magnitud de la información a almacenar, siempre deseamos que sea suficiente.
- **Velocidad de transferencia de información ilimitada.** Siempre se requiere que el tiempo para leer o escribir esa información sea el mínimo posible.
- **No volátil.** Es importante que la información no desaparezca cuando se apaga el equipo.
- **Costo mínimo.** Siempre deseamos minimizar el costo en la construcción del equipo.

Resulta evidente que no contamos con un solo tipo de memoria que siquiera se aproxime a tener **todas** estas características a la vez, en un solo dispositivo, por lo que se requiere conjugar las características de múltiples tipos de dispositivos, que proporcionan diversas prestaciones por distintos precios. La combinación que hagamos debe-

rá buscar acercarse lo suficiente a las características de la memoria ideal para atender las necesidades de nuestras aplicaciones, por un costo mínimo, con el fin de que todo el sistema tenga un mejor desempeño.

4.4 Distribución de tipos de memoria

La estrategia convencional para conjuntar las características de distintos dispositivos de almacenamiento reconoce cuatro capas principales, cada una de las cuales tiene más capacidad de manera progresiva a costa de la velocidad y del precio por capacidad de almacenamiento.

De este modo se suele tener mínima capacidad de almacenamiento incorporada en la CPU, debido, en primera instancia, a que los procesadores actuales tienen importantes limitaciones en la capacidad de circuitos utilizados en su construcción y la potencia que pueden consumir, ya que la construcción actual de los procesadores debe distribuirse entre los circuitos propios para el procesamiento junto con los de la memoria que se desee incluir.

En segundo lugar, se tiene la memoria caché, la cual, dados los requerimientos de velocidad actuales, suele incluirse en el mismo paquete que la CPU, aunque estos circuitos funcionan a frecuencias menores para reducir el consumo de potencia y disipación de calor asociada.

En tercer lugar, se tiene la memoria RAM, hoy día implementada con circuitos de memoria que se conectan a la CPU mediante buses de direcciones y de transferencia de valores; aunque la distancia física con respecto a la CPU limita la velocidad de transferencia que se puede tener hacia estos dispositivos, ya no sufren de las mismas limitaciones de consumo de potencia y tamaño de la memoria que se encuentra incorporada a la CPU.

Por último se tiene el almacenamiento secundario, mediante el cual se busca contar con la mayor cantidad de capacidad de almacenamiento, por lo que reducir el costo por megabyte en los dispositivos es fundamental; sin embargo, a menudo esto se logra a costa de la velocidad de operación y de transferencia.

Es importante resaltar que a la fecha se han desarrollado e incluido muchos servicios de red como medios de acceso a servicios de almacenamiento con enormes capacidades a costos muy bajos; no obstante, la velocidad de transferencia de estos medios no suele ser competitivo con respecto a dispositivos locales, por lo que el impacto en el esquema de administración de la memoria aún no ha sido significativo.

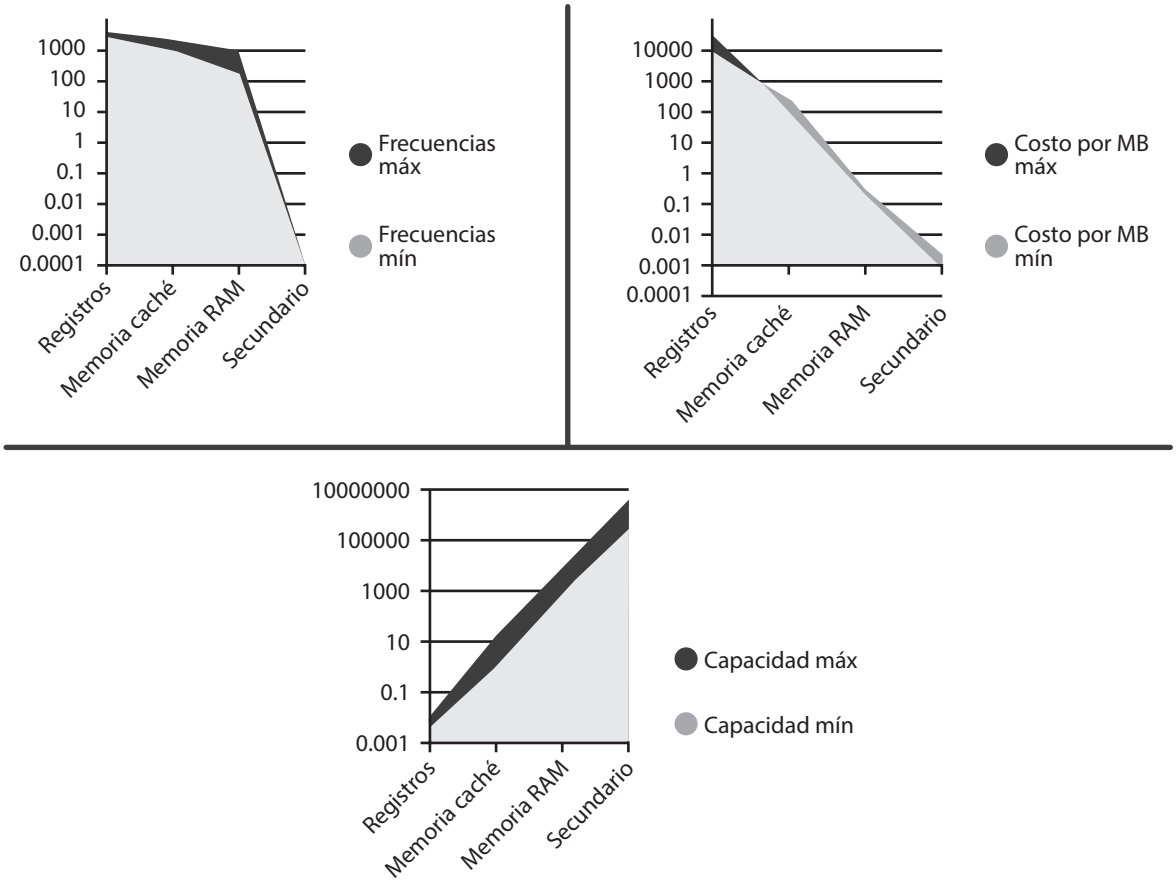


Figura 4.1 Comparativo de las características en las distintas capas de la memoria.

Objetivos de la administración de la memoria

Para poder aprovechar los distintos tipos de memoria es necesario un conjunto de mecanismos, así como su respectiva implementación. Estos mecanismos deben considerar las necesidades del sistema de información para que la atención a los procesos se mantenga con un buen nivel de desempeño. Así, podemos enumerar una serie de consideraciones constantes en el diseño de un administrador de la memoria:

- **Aprovechar los recursos del equipo.** Lo que incluye reconocer y utilizar los diversos tipos de dispositivos que puedan intervenir y las características de estos.

Importante

♥ **Docente.** Algunas aplicaciones actuales requieren almacenar cantidades masivas de información que rebasan la capacidad de los dispositivos que podemos conectar a una sola computadora, por lo que las soluciones para este tipo de requerimientos se encuentran dentro del terreno de los

siste mas distribuidos y se ven como una extensión del almacenamiento "fuera de línea" que se ha tenido en cintas y otros medios removibles que proporcionan bajo costo por MB. Si entendiéramos estas soluciones como una quinta capa en nuestra clasificación, ¿qué impacto tendría si se comenzará a rivalizar en velocidad con el almacenamiento secundario? Este fenómeno comienza a ocurrir con sistemas de almacenamiento en red y generará cambios interesantes en el terreno de los sistemas distribuidos.

♠ **Arquitecto.** Para aplicaciones móviles es común que cuando una aplicación requiere mantener una gran cantidad de información, lo haga mediante servicios de red, lo que permite que en lugar de almacenar la información de forma local la obtenga de servidores remotos, explotando el ancho de banda en vez de la capacidad de almacenamiento del dispositivo. Aunque este enfoque a menudo se use para servicios como el video en demanda, que lo requieren por la masiva cantidad de información en el acervo, también tiene usos para salvaguardar la integridad de la información cuando esta es modificada desde diversos dispositivos o se coordina con distintos usuarios en aplicaciones distribuidas. Asimismo, también es

En sistemas de multiprogramación y tiempo compartido:

- **Proporcionar a los procesos acceso a la memoria.** Resulta indispensable implementar mecanismos para asignar conjuntos de posiciones de memoria que se adecuen a las necesidades de los procesos, preservando la independencia lógica de los programas, comúnmente llamados segmentos. Asimismo, en los casos en que los procesos requieran más memoria de la que se les puede proporcionar, también se deben implementar mecanismos para rechazar tales peticiones sin efectos adversos a los procesos existentes.
- **Proteger el acceso a la memoria.** Una vez asignada la memoria a un proceso, debe evitarse que procesos no autorizados utilicen la información contenida en dicha memoria.

Distribuido

- **Manejo de memoria compartida entre los nodos de los sistemas distribuidos.** En sistemas de información contenidos en una sola computadora, todos los procesos tienen acceso a la memoria compartida, no así cuando los procesos se ejecutan en computadoras diferentes, por lo que se deben proporcionar facilidades para que los cambios a la información realizados en un equipo se propaguen a los demás.

Tiempo real

- **Reducción y predictibilidad de los tiempos de acceso a la memoria.** Funciones como la memoria virtual pueden causar que algunas operaciones de acceso a la memoria tomen mucho más tiempo que otras, y en ocasiones es muy difícil predecir en qué momento se presentarán estas demoras, lo que va en contra del objetivo de los sistemas de tiempo real de brindar un nivel de atención predeterminado y estable a las aplicaciones, por lo que el manejo de memoria debe evitar ese tipo de medidas.

Embebido

- **Reducción de número de operaciones para disminuir consumo de potencia.** Operaciones como la carga de un proceso a me-

moria a menudo requieren algunos segundos de tiempo de procesador y deben atenderse rápidamente para dar un buen servicio a los usuarios finales. En sistemas donde el consumo de potencia proporcionada por una batería resulta crítico a menudo es preferible mantener la memoria ocupada, con lo que se evita liberar la memoria de un proceso que se deja de usar por no repetir la carga del programa la próxima vez que se utilice.

de gran utilidad como respaldo de la información en caso de que el dispositivo móvil tenga algún percance.

- **Reducción del *overhead* en sistemas con pocos recursos.** No solo la potencia sino también la capacidad de procesamiento, el almacenamiento en sistema de archivos, la memoria RAM disponible y prácticamente todos los recursos de las computadoras incluidas en otros dispositivos estarán limitadas para minimizar el costo y acceder a aplicaciones donde las computadoras de propósito general no tendrían cabida, lo que hace que la implementación del sistema operativo deba minimizar su propio consumo de recursos a menudo a costa de funciones habituales en otros tipos de sistemas de información.

Funciones y operaciones del administrador de memoria

En los primeros sistemas de tipo monoproceso era suficiente con permitir que los programas usaran la totalidad de la memoria disponible del modo que les resultara más conveniente. Sin embargo, conforme se pasó al modelo de multitarea, esta estrategia no resultó viable debido a las "colisiones de memoria" que producían los distintos programas al intentar cargar elementos distintos en las mismas posiciones de memoria, propiciando estados inconsistentes.

Debido a esa situación, se impuso construir mecanismos por los cuales los programas pudieran usar solo una parte de la memoria, sin intervenir en la ejecución de los demás. De este modo, **la administración de memoria** comenzó como el esquema por el cual los procesos pueden solicitar rangos de memoria y recibir las asignaciones con diversos mecanismos de protección.

Tiempo después se idearon mecanismos mediante los cuales los procesos en su conjunto pudieran utilizar una cantidad de memoria superior a la cantidad de memoria RAM físicamente disponible en la máquina, como el almacenamiento secundario y los discos duros, y después la memoria no volátil, como la memoria Flash. Cuando aparecieron estos primeros mecanismos, se dejó a cargo de cada programa decidir qué partes de su memoria habría de enviarse a almacenamiento secundario y durante qué periodos; luego, este proceso de intercambio se automatizó, lo que dio origen al concepto de vecindad (Locality). Este intercambio automático de partes de la memoria asignada a un

proceso entre la memoria RAM y el almacenamiento secundario para simular una memoria grande empleando tan solo una cantidad pequeña de esta se conoce como **memoria virtual**.

A la fecha, la memoria virtual ha permitido que los sistemas operen bajo esquemas de tiempo compartido, usando cantidades de memoria extendidas de forma muy exitosa; sin embargo, eso implica que algunas operaciones de memoria, que requieren recuperar información del almacenamiento secundario, tomarán considerablemente más tiempo de lo habitual, aunque es muy complicado predecir cuáles o cuándo ocurrirán. Esto hace que el uso de memoria virtual no sea adecuado para los sistemas de tiempo real.

Asignación de memoria contigua

Cuando se ejecuta un único programa en todo el sistema, las tareas del administrador de memoria son mínimas, ya que dicho programa puede ocupar toda la memoria disponible de la manera en que le parezca más conveniente.

Por tanto, la función principal del administrador de memoria en este tipo de sistemas consiste en identificar los rangos de memoria disponible y llevar un registro de los rangos de dirección ocupados y libres para ayudar a los procesos a encontrar los rangos de memoria que requieren.

Los rangos de memoria deben constar de celdas de memoria sucesivas. Así, los desplazamientos en la memoria (por ejemplo, para obtener la instrucción siguiente del programa) obtendrán la información correcta y serán sencillos de implementar.

También es importante registrar el momento en que los distintos rangos de memoria dejen de ser utilizados por el proceso para liberarlos y poder asignarlos al proceso siguiente.

Importante

♠ **Arquitecto.** Algunos sistemas embebidos optan por una administración de memoria sencilla, como si se tratara de sistemas de tipo monoproceso; esto permite ahorrar recursos a costa de las prestaciones de otros modelos de administración de memoria.

Administración de memoria para multiprogramación

La implementación de los administradores de memoria para los sistemas de multiprogramación requirió un esfuerzo sostenido de casi una década, lo que al final condujo a los mecanismos de memoria virtual y aportó el principio de vecindad, que hoy día es fundamental en múltiples aspectos de arquitectura de computadoras y sistemas.

Para una mayor comprensión al respecto, comencemos con la revisión de algunas de las formas de administración por las que pasó de camino a la construcción y consolidación de los esquemas actuales.

Partición fija

La memoria se divide en una serie de particiones de tamaño fijo. En este tipo de memoria, cuando se cargan los procesos, estos se asignan a una de esas particiones, por lo que el administrador de memoria solo informa al proceso de la existencia de memoria dentro de la partición que se le asignó, de modo que solo pueda utilizar esta y evitar que tenga colisiones con otros procesos.

Una de las implementaciones de esta forma de administración de memoria fue usada por el sistema IBM OS/360, donde al cargar el sistema operativo se generaba una serie de particiones de memoria que, por el hecho de limitar el número de procesos que podían compartir recursos, se conoció como MFT (Multiprogramming with a Fixednumber of Tasks). En este sistema los trabajos formados para procesar en Batch eran atendidos en cuanto una partición de tamaño suficiente se liberaba y se asignaba a estos.

En este caso, el tamaño de las particiones debe hacerse pensando en que sea suficiente para una variedad de trabajos, por lo que deberá ser algo mayor al requerimiento de los programas que se cargarán en estas. De lo contrario, se generaría una sobredemanda de las particiones de gran tamaño debido a que las particiones de tamaño reducido no pueden atender programas que exceden su capacidad.

Sin embargo, esto implica que una parte de la memoria no será utilizada. A este sobrante de memoria dentro de la partición fija se le conoce como **fragmentación interna**, un aspecto que en todos los casos debe intentar reducirse.

Partición variable

En un principio, para evitar la fragmentación interna de la partición fija se plantearon sistemas en los que el tamaño de las particiones pudiera variar cuando la partición se liberaba por un proceso, para que se asignara de inmediato al proceso siguiente; con ello la partición podría ajustarse al tamaño requerido y se evitaría la fragmentación.

Sin embargo, al requerir las particiones que la memoria usa para que estas fueran contiguas, se presentó un nuevo problema que resultó tener aún más impacto en el desempeño. El espacio dejado por particiones pequeñas, creadas a la medida de procesos sencillos, quedaba inmersa entre particiones de gran tamaño y con tiempos de vida más largos. Estas pequeñas particiones contenían memoria que no podía reunirse para formar particiones más grandes, hasta que las particiones intermedias también fueran liberadas. Esta memoria que no se puede aprovechar por estar entre particiones variables se conoce como **fragmentación externa**, y en la práctica resultó dejar una cantidad de memoria sin uso a lo largo del tiempo mayor que la fragmentación interna, por lo que los sistemas de partición variable no tuvieron un uso muy extendido.

Espacio de direcciones con registros base y desplazamiento

Para implementar un rango de direcciones en el que el proceso pueda operar, y que no dependa de conocer el tamaño total de la memoria, se puede modificar el modo de direccionamiento de la memoria del procesador mediante el uso de una pareja de registros en vez de uno solo.

El primero de estos registros constituye la base y el punto de partida de las direcciones asignadas al proceso; por ejemplo, en el caso de la arquitectura Intel se conoce como el registro de segmento, mismo que permite señalar diversas posiciones en la memoria, con los 16 bits más significativos de una dirección lineal, que debiera ser de 20 bits. Esto le da una resolución de 16 palabras para el inicio del rango de memoria, o segmento, que habrá de manejarse.

Una vez establecido el valor del registro base, se usa un segundo registro, conocido como de desplazamiento, para indicar la posición de memoria, relativa a la base, en la que se desea realizar la operación. Este segundo registro de desplazamiento u offset también es de 20 bits y permite identificar una palabra individual.

Por sí solo, este mecanismo permite acceder a las posiciones de memoria desde una gran cantidad de combinaciones de valores del registro base y desplazamiento; sin embargo, no proporciona mecanismos de protección para que los procesos no excedan el tamaño de los segmentos de memoria que les son asignados, aunque sí suministra un punto de partida sencillo y funcional para implementar la paginación o el manejo de segmentos de memoria.

Segmentación

Las direcciones base y los desplazamientos se pueden aprovechar para dividir la memoria en rangos contiguos que se asignarán a los procesos. Pero para evitar los problemas de la partición variable es necesario usar determinados mecanismos a fin de evitar que los programas empleen direcciones fijas de memoria y sea posible relocalizar los segmentos en caso de requerirlo, con un mínimo impacto al desempeño del sistema. A este proceso se le conoce como **segmentación**.

En principio, la segmentación asigna tres rangos de memoria contigua del tamaño que requieran los procesos, cada una con diferentes funciones, como se describe a continuación:

1. **Código.** Para almacenar el código del programa.
2. **Datos.** Contiene los datos estáticos, constantes, cadenas y tablas de símbolos.

3. **Stack.** Con la secuencia de direcciones del programa por las que pasa el Program Counter en las llamadas a subrutinas anidadas.

Conforme los procesos soliciten más memoria para sus estructuras de datos dinámicas, se les proporcionarán nuevos segmentos del tamaño solicitado (siempre que la memoria disponible lo permita); sin embargo, la liberación de estos segmentos quedará a cargo del programa, ya que la tabla de procesos no registrará estos segmentos adicionales.

Las direcciones en sistemas con segmentación se manejan con direcciones lógicas que constan de la dirección base del segmento (por lo regular, un apuntador al origen del segmento) y un desplazamiento. Para permitir la relocalización se evita que estas direcciones se encuentren en saltos absolutos en el código del programa, para lo que se colocan por separado en el segmento de datos en "tablas de símbolos", en las que se indica el símbolo a resolver (que usualmente tiene un identificador entero) y en el que se registrarán los valores de base del segmento y desplazamiento correspondientes. Incluso se pueden compilar los programas en modo de depuración, con objeto de incluir también la cadena con el identificador en lenguaje de alto nivel del símbolo representado en las tablas de símbolos; por ejemplo, el nombre de la variable o de la función en el código fuente.

Cuando un proceso en un esquema de segmentación requiere realizar un salto absoluto (por ejemplo, para llamar a una subrutina), primero consulta en su tabla de símbolos la dirección base y el desplazamiento donde puede encontrar dicha subrutina; en caso de encontrarlas, usa esos valores para realizar el salto.

Las direcciones que finalmente se cargan en los registros del procesador luego de consultar las tablas correspondientes se conocen como direcciones físicas.

Es importante resaltar que siempre se mantiene una tabla de segmentos donde se indica la protección de los segmentos; por ejemplo, si la memoria estará compartida con otros segmentos y si permite operaciones de escritura o solo de lectura de la información que contiene. En el caso de memoria virtual, permite controlar qué segmentos se tienen en almacenamiento principal o secundario.

De este modo, cuando se requiere relocalizar un segmento, se pueden revisar las tablas de símbolos (y no todo el programa y las variables) para identificar en sus registros las entradas que usen el

Importante

♥ **Arquitecto.** Además de lo descrito en el texto, las tablas de símbolos también simplifican la construcción de compiladores de lenguajes orientados a objetos como C++, ya que la representación de las rutinas y variables en instancias concretas de una tabla genérica resulta análoga a los conceptos de objeto y de clase, respectivamente. Incluso conceptos como el polimorfismo pueden representarse con relativa facilidad gracias a dichas tablas.

♠ **Líder.** No obstante todas sus cualidades, un aspecto negativo de las tablas de símbolos radica en que estas mezclan información de control de la aplicación (ubicación de funciones y variables) con datos de la aplicación (cadenas y arreglos con datos arbitrarios), lo que permite la existencia de vulnerabilidades de seguridad que, mediante datos de entrada, pueden perturbar la operación

de los sistemas de información, por lo que es necesaria una disciplina de desarrollo exhaustiva y uniforme de saneo de datos de entrada.

valor antiguo de la base del segmento y reemplazarlos por el nuevo valor. Sin embargo, conviene evitar hacer relocalizaciones de segmentos debido a que aún se considera una operación costosa en términos de la CPU, por lo que se prefiere cierto nivel de fragmentación externa, ya que el resto de los procesos usan múltiples segmentos en vez de un único rango de memoria para todas sus necesidades.

Administración de memoria libre

Al tener rangos de memoria asignados a los procesos que van creándose y terminando, es necesario llevar un registro de la memoria que se tiene disponible para la asignación, así como recuperar la memoria que los procesos ya no requieren para que el sistema pueda seguir operando y no deje de aprovechar la memoria disponible.

El principal problema a atacar radica en que toda la memoria disponible puede estar libre, por lo que el mecanismo de administración debe ser lo bastante económico para no incurrir en una sobrecarga que impida la apropiada atención de los procesos.

De este modo, hay dos tipos de estrategias para administrar la memoria libre: los mapas de bits y las listas enlazadas.

Mapa de bits

Si deseamos representar a la memoria por unidades de asignación (es decir, como conjuntos de direcciones de memoria que pueden abarcar desde una palabra individual hasta grupos de algunos KB de tamaño), podemos hacerlo con un solo bit por unidad de asignación en el estado de asignado o libre para cada unidad, con lo que se reduce el tamaño de nuestro mapa de bits de memoria libre por varios órdenes de magnitud; por ejemplo, 4 GB de RAM en unidades de asignación de 4 KB requeriría únicamente 500 KB para su mapa de bits.

Este mecanismo es sencillo y eficiente, pero tiene dos inconvenientes principales. El primero es que si minimizamos el tamaño del mapa de bits haciendo unidades de asignación más grandes, tendremos un desperdicio de memoria por fragmentación interna en la última unidad de asignación dada a cada proceso más o menos de la mitad de una unidad por cada rango de memoria solicitada; esta memoria no utilizable es directamente proporcional al tamaño de la unidad de asignación, por lo que genera una presión para no incrementar mucho el tamaño de la unidad de asignación.

El segundo es que cuando los procesos solicitan conjuntos de direcciones de memoria contigua para albergar sus datos es necesario recorrer de manera secuencial el

mapa de bits en busca de una serie de unidades de asignación lo bastante larga para contener en su totalidad el conjunto solicitado. Aunque puede no ser lo mejor asignar memoria del primer rango de direcciones libres lo bastante grande que encontremos al revisar secuencialmente el mapa de bits, deberíamos continuar la búsqueda en todo el mapa de bits con objeto de encontrar el rango de tamaño suficiente cuyo tamaño se aproxime más al solicitado, a fin de disminuir con ello la fragmentación externa. Este proceso es tardado porque requiere la revisión de todo el mapa de bits, o al menos una parte considerable de este, lo que consume tiempo de la CPU en la serie de accesos a la información.

Listas enlazadas

Al generar un registro por cada rango de direcciones de memoria libre que se tenga, el sistema iniciaría de manera potencial con un rango que abarcaría la mayor parte de la memoria del sistema. Conforme se asigna la memoria a procesos, podemos dividir este registro para representar los segmentos de memoria que quedan libres y los que quedan ocupados por algún proceso, con el fin de facilitar la creación y el retiro de elementos de la lista de segmentos, además de que podemos usar una lista ligada.

En este caso, cada registro de la lista debería indicar:

- Si se trata de memoria libre o asignada a un proceso.
- La dirección de memoria en la que inicia el segmento.
- La longitud del segmento.
- Un apuntador al siguiente registro (y en ocasiones otro para el registro anterior).

Se puede organizar una sola lista ordenada por la posición física de los segmentos, en la que vayamos intercalando los segmentos asignados a procesos con los segmentos de memoria libre; en este caso, cuando liberamos un segmento de memoria, es sencillo verificar si los segmentos adyacentes están libres para unirlos en un solo segmento de memoria libre que incluya el rango completo de direcciones de los dos y que evite la fragmentación externa.

Sin embargo, si deseamos encontrar un segmento de memoria libre y de tamaño suficiente, pero que se acerque lo más posible al tamaño solicitado para un nuevo segmento, tendríamos que recorrer toda la lista para verificar el tamaño de todos los segmentos de memoria libres antes de poder elegir uno.

También podemos hacer listas separadas para los segmentos de memoria asignada a procesos y para los de memoria libre, e incluso ordenar estos por tamaño. En esta organización, encontrar el segmento del tamaño más adecuado para una petición es

sencillo, porque una vez que encontramos el primer segmento de tamaño suficiente está garantizado que también es el que se aproxima más al tamaño deseado. Sin embargo, cuando deseamos liberar un segmento, es mucho más complicado identificar los segmentos adyacentes para verificar si es posible unirlos en un solo segmento.

Distribución de espacio de intercambio (Swap)

Una vez que se tienen segmentos de memoria en los que los procesos pueden manejar su código y sus datos, es posible identificar periodos en los que esa memoria no estará en uso, por lo que esta podría aprovecharse mejor dejándose libre para ser usada por otros procesos.

Este caso puede ilustrarse con determinados servicios, como el de atención a terminales remotas, ya que este proceso se inicia con el sistema operativo, pero la mayor parte del tiempo está inactivo en espera de que se solicite una conexión; así, mientras está inactivo y bloqueado, el espacio en memoria que ocupa su código y datos no estará siendo activamente utilizado.

En estos casos, puede apartarse un área especial en el disco para almacenar la información de procesos inactivos que no se utiliza, a fin de liberar esa memoria, de modo que los procesos activos tengan más memoria disponible para trabajar. De esta forma, esta memoria estará siendo intercambiada de la RAM al almacenamiento secundario (por ejemplo, el disco duro) y se recuperará cuando el proceso deba salir del bloqueo para atender una petición o continuar sus actividades. A este manejo de los segmentos de memoria se le conoce como intercambio (Swap) y al área de disco que se reserva para recibir la información de la memoria se le conoce como espacio de intercambio.

Para administrar el espacio en disco con objeto de almacenar los segmentos, originalmente se utilizaron los mismos algoritmos que se emplean hoy día para la administración de memoria libre, aunque con el tiempo fueron reemplazados por los modelos de memoria virtual que se exponen a continuación.

Memoria virtual

A la habilidad que tiene un sistema operativo de permitir que sus procesos dispongan de una cantidad de memoria para sus datos y código que exceda a la capacidad de la memoria física disponible en

Importante

♥ **Docente.** Algunas de las primeras máquinas virtuales se usaron en el marco de los estudios de diversos algoritmos para memoria virtual; desde su aparición, el término "virtual" ha hecho referencia a todas las implementaciones de dispositivos y sistemas que exhiben una funcionalidad que no corresponde con los dispositivos físicos que la soportan, sino que mediante software simulan características diferentes.

♠ **Arquitecto.** La memoria virtual está tan difundida en los sistemas de tiempo compartido y funciona tan bien que los usuarios finales a menudo no perciben su operación y dejan de preocuparse de la cantidad de memoria disponible en el equipo. Aunque esto es todo un logro de la administración de la memoria, como desarro-

el sistema se le conoce como **memoria virtual**. Dicha habilidad se logra mediante el principio de vecindad (Locality Principle), con el fin de determinar qué parte de la memoria asignada a los procesos no se está utilizando y desplazando de forma temporal la información de esa parte al almacenamiento secundario.

El término memoria virtual surgió a raíz de la creaciónde las máquinas virtuales, simulaciones de sistemas operativos que se ejecutaban en servidores de capacidades mayores a las máquinas simuladas, que se emplearon para probar los algoritmos usados para automatizar el intercambio de páginas de memoria entre la RAM y el disco duro.

llador es poco recomendable ignorar el mecanismo que soporta nuestras aplicaciones, ya que eso compromete nuestra capacidad para diagnosticar de manera adecuada gran cantidad de posibles problemas y apreciar las mejores prácticas para aprovechar a cabalidad sus características.

Administración por paginación

Como ya se mencionó, luego de un tiempo de haberse creado, los procesadores comenzaron a implementar mecanismos que permitieran un mejor manejo de segmentos de memoria para el multiprocesamiento, e incluso se comenzó a experimentar con el intercambio de segmentos de memoria entre la RAM y el almacenamiento secundario; sin embargo, originalmente los propios procesos debían usar llamadas al sistema para solicitar que los segmentos que se sabía no tendrían actividad fueran desplazados a almacenamiento secundario, o bien limitaban el intercambio solo a procesos que pasaban largo tiempo inactivos.

Algunas de las dificultades que se generan con los segmentos son:

- Cuando un segmento de código se vuelve a cargar en la memoria, pero en direcciones diferentes a las que ocupaba originalmente, las referencias a memoria de saltos absolutos pueden terminar señalando a posiciones erróneas; esto es difícil de detectar y de corregir, aunque el uso de registros base puede ayudar a resolver estos problemas.
- La fragmentación externa entre los segmentos ocupados tiende a dejar una parte considerable de la memoria en condiciones en que no se puede aprovechar.
- Si el intercambio es invocado directamente por los programas, se requiere un gasto considerable de recursos para estudiar el comportamiento del sistema con el fin de asegurar que algunas partes de este pueden ser intercambiadas al almacenamiento secundario durante algunos periodos.

Una propuesta para resolver estos problemas fue la paginación de la memoria, la cual plantea que en vez de generar segmentos del tamaño que requieren los procesos, la memoria se dividiera en un conjunto de unidades de asignación, conocidas con el

nombre de páginas. En este caso, todas las páginas son del mismo tamaño y se asignan tantas como sea necesario para completar los requerimientos del proceso. Las páginas que se asignen no requieren estar juntas o en un orden particular en la memoria, por lo que evitan la fragmentación externa. Como la memoria requerida por el proceso puede no ser un múltiplo exacto del tamaño de la página, persiste la fragmentación interna, aunque esta es en promedio de la mitad del tamaño de una página por proceso, lo que se considera una cantidad muy razonable.

En particular, cuando las páginas se encuentran cargadas en memoria RAM se les conoce como **marcos de página** o **Page Frames**, para distinguirlas de las páginas que almacenan su información en otros dispositivos además de la memoria RAM.

No obstante, para cada página asignada a un proceso requerimos una tabla de páginas en la que se almacena toda la información necesaria. En dicha tabla se indica el número de página, el proceso al que se asignó y los atributos propios de la administración de páginas; además, indicar si es memoria compartida, también se incluyen los datos para los algoritmos de intercambio a almacenamiento secundario y el valor para el registro base de la CPU para usar las direcciones de memoria contenidas en la página.

De este modo, cada vez que requiramos hacer una operación de memoria de una página diferente, debemos modificar el valor en el registro base consultando la tabla de páginas. Como esto puede ocurrir con una frecuencia relativamente alta, los procesadores CISC comenzaron a incluir una tabla de memoria especializada para almacenar al menos una parte de la tabla de páginas y con ello acelerar estas operaciones; esta tabla se conoce como Translation Lookaside Búfer y se benefició también del concepto de vecindad, ya que se conoce que los procesos solo utilizan un subconjunto de las páginas que tienen asignadas en un periodo determinado.

Una vez que se conoce que las páginas tendrán un tamaño constante, en el procesador también es posible incorporar un mecanismo para verificar que los desplazamientos no excedan del tamaño máximo de la página, con el fin de evitar accesos ilegales a memoria, usualmente mediante un registro adicional de la CPU que contenga el desplazamiento máximo para la página en cuestión.

Los inconvenientes que surgen con la paginación se presentan en dos frentes principales:

- En la compilación de las aplicaciones, los saltos absolutos para condiciones y ciclos se ven limitados a direcciones que se encuentran en la misma página, así que cambiarlos por saltos absolutos a páginas diferentes resulta costoso en consumo de la CPU; por tanto, se requiere que los compiladores incrementen su complejidad para detectar estos ciclos y bifurcaciones y procuren acomodarlos en una sola página.

- En las estructuras de datos, los arreglos y las estructuras de datos dependen de desplazamientos sobre una dirección base; con la paginación resulta que estos desplazamientos solo son válidos dentro del rango de direcciones de una página, lo que limita el tamaño de las estructuras de datos que se pueden implementar de manera sencilla. Para hacer colecciones o estructuras de datos mayores a una página se requiere que las aplicaciones implementen soluciones jerárquicas, lo que genera complejidad para las aplicaciones.

Aspectos de diseño para sistemas

Para tener el control de las páginas de memoria se requiere un mecanismo de representación que consuma poco espacio por entrada, ya que por lo regular se tienen millones de páginas, además de que sea lo bastante sencillo para no generar una sobrecarga importante al utilizarlo.

Cuando se desee usar la información contenida en una página que la memoria virtual ha colocado en un dispositivo de almacenamiento secundario y no en la RAM, se requiere cargar la información a la RAM antes de utilizarla, para lo cual debe bloquearse el proceso que requiere la información hasta que esta termine de cargarse. A este proceso se le conoce como **fallo de página**, y siempre se busca minimizar la frecuencia con que ocurre debido al impacto negativo que tiene en el desempeño de los sistemas operativos.

La información de la tabla de páginas se utiliza principalmente para traducir los identificadores de página, usados por los programas en sus direcciones lógicas y por las direcciones de marco de página al usarse en las direcciones físicas del procesador al acceder a la memoria.

Adicionalmente, en la tabla de páginas también tenemos acceso a información usada por la administración de las páginas y por la memoria virtual, la cual se describe a continuación.

- Una bandera para indicar si la página ha sido referenciada recientemente, conocida como bit R.
- Una bandera para indicar si la página ha modificado su contenido desde la última vez que se escribió a almacenamiento secundario, conocida como bit W. La función principal del bit W es dar certeza de que la información está representada de manera íntegra en almacenamiento secundario cuando se reemplace la página, lo que permite omitir la escritura de la información y, por tanto, ahorrar tiempo en atender el fallo de página.

- Información de protección. Además se tiene una serie de bits para controlar aspectos como si la página es de solo lectura, si se trata de una página con código de programa, si la página permite modificaciones, si la página será de memoria compartida, e incluso detalles como si la página permite que se emplee como código de programas o por qué procesos puede ser utilizada.

Toda esta información se almacena en la memoria y está disponible para acceder a esta cuando los procesos requieran hacer consultas a memoria con direcciones que no pueden obtenerse por meros saltos locales.

Si la tabla de páginas siempre se maneja en memoria RAM, todas las operaciones que requieran utilizar una página diferente a la que estaban utilizando (es decir, que necesiten realizar un cambio de contexto) tendrán que hacer al menos un acceso a memoria previo para consultar la tabla de páginas y otro posterior para utilizar la memoria requerida originalmente. Esto implica un impacto importante al desempeño del sistema y debe apoyarse por memoria especializada para acelerar el proceso.

Translation Lookaside Búfer (TLB)

Para reducir el impacto, los procesadores implementan un área de memoria especializada para la tabla de páginas que no solo opera a una velocidad superior a la de la RAM sino que además se implementa como memoria asociativa, de modo que la entrada de la página puede encontrarse por su identificador y no necesariamente por su posición en la tabla.

Al igual que en la segmentación, los programas utilizan direcciones lógicas que permiten identificar las páginas a emplear sin conocer el lugar en la memoria en que se encuentran y que se resuelven a direcciones físicas cuando requieren utilizarse.

Dado el alto costo de implementar bancos de memoria especializada en el procesador, por lo general estos solo tienen un reducido número de entradas. Sin embargo, debido al principio de vecindad, estas entradas pueden atender una proporción muy elevada de las operaciones de cambio de contexto, con lo que efectivamente reducen de manera importante el impacto al rendimiento por resolución de direcciones lógicas en físicas. Así, cuando la dirección lógica buscada no se encuentra en la TLB, se recurre a la tabla de memoria en RAM, en lo que se conoce como una TLB Miss.

Adicionalmente, los procesadores Intel i7 tienen TLB de múltiples niveles; en este caso, el primer nivel tiene 64 registros y una velocidad de respuesta excelente, mientras que el segundo nivel tiene 512 entradas y una velocidad de respuesta un poco más lenta, aunque aún se considera mucho más rápida que la RAM. Cada núcleo de proce-

samiento tiene sus propias TLB de los primeros dos niveles. Así, su función consiste en manejar automáticamente las actualizaciones de la información de las TLB; por ejemplo, si se libera una página, todas las TLB que la incluyan deberán omitirla y se mantienen sincronizadas. Adicionalmente, también implementa una TLB especializada en páginas grandes, que en el caso de Linux no se emplea para procesos de usuario sino para áreas de memoria que atiende módulos del sistema operativo, al cual solo tiene 16 entradas, y también se replica por núcleo de procesamiento.

Tablas multinivel o jerárquicas

Otra consideración importante para la implementación de la tabla de páginas tiene que ver con las propias características de la implementación de las arquitecturas de las tarjetas principales de las computadoras de propósito general más actuales. Gracias a la popularización de las diversas plataformas, a la fecha se ha generado una gran variedad de modelos que se adaptan a los costos y las necesidades de diversas aplicaciones. De este modo, en equipos de bajo costo, por ejemplo, es común tener configuraciones entre los 2 y los 32 GB de memoria instalada. No hay que olvidar que por las características de la memoria RAM, los módulos de memoria suelen utilizarse en pares, con el fin de alternar su uso y mejorar la frecuencia de las operaciones de memoria (técnica conocida como Dual Channel).

Sin embargo, esto obliga a la arquitectura a contemplar los buses de memoria para el tamaño máximo de módulo de memoria que se soportará y a considerar el número de ranuras para colocar dichos módulos; así, el costo se impacta de forma importante si se implementan buses más grandes o ranuras adicionales, por lo que debe ubicarse dentro del rango de costos que se desea para el mercado objetivo, limitando la memoria máxima que se soportará.

Para las tarjetas principales fabricadas en masa la solución es colocar entre dos y cuatro ranuras para módulos de memoria y limitar la memoria soportada a unos 32 o 64 GB, ya sea que se quiera impactar al mercado doméstico o al de servidores de bajo nivel.

Incluso en este escenario tan modesto, con respecto al de los servidores de alto nivel que llegan a soportar Terabytes de RAM, es común que las computadoras solo tengan instalada una fracción de la memoria que soporta la tarjeta principal. Y como también se debe considerar el mapa de memoria para el tamaño máximo de los módulos, la memoria además se encuentra separada en rangos que corresponden a los módulos instalados.

En principio, es responsabilidad del BIOS detectar los módulos presentes y entregar esta información al sistema operativo con objeto de que conozca qué rangos de

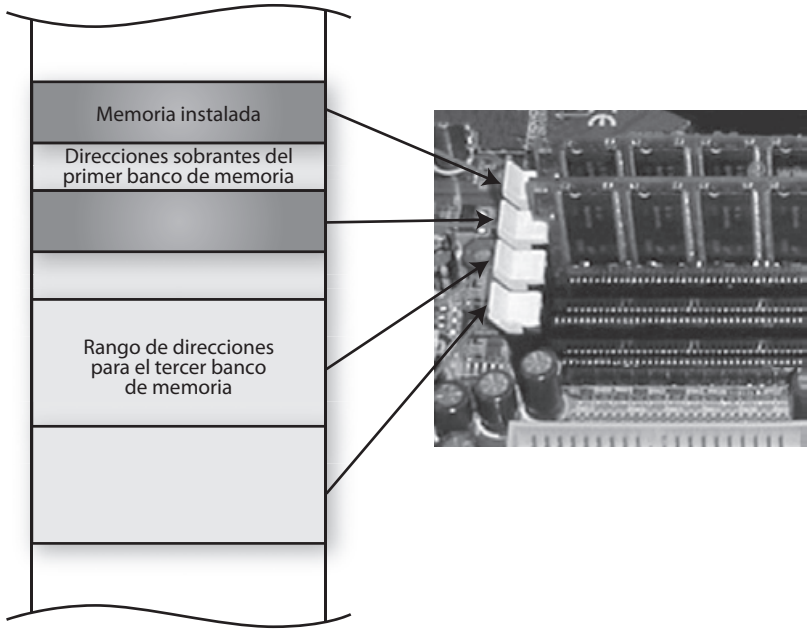


Figura 4.2 Memoria instalada y su ubicación en una parte del mapa de memoria.

memoria RAM resultan válidos en la presente sesión; no obstante, esto genera un problema adicional en la tabla de páginas, pues para representar todo el mapa de memoria de la arquitectura sería necesaria una cantidad de entradas mucho mayor que el número de páginas que en realidad es posible utilizar por encontrarse presente. Esto representaría una sobrecarga en el uso de memoria en la tabla de páginas y en las operaciones de búsqueda de la información en esta.

Con el fin de evitar esta sobrecarga, la tabla de páginas en la RAM puede implementarse en múltiples niveles, lo que permite la ubicación rápida de una página en arquitecturas con cantidades de memoria muy grandes, así como omitir los rangos de direcciones que no tienen memoria instalada.

Es importante resaltar que no tiene caso aplicar esto en la TLB debido a que, por definición, esta no tiene páginas que el sistema no utilice por no tener circuitos de memoria instalados.

Linux y Windows son ejemplos claros de sistemas operativos que usan múltiples niveles en la tabla de páginas en su forma más simple. Por lo general, una tabla de páginas multinivel divide la base de la dirección lógica en tantas partes como niveles se usarán; por ejemplo, Windows la divide en dos partes.

Con la parte más significativa de la dirección lógica, se indexa una tabla que señalará a las tablas del segundo nivel, en lugar de a las páginas. Dichas tablas están in-

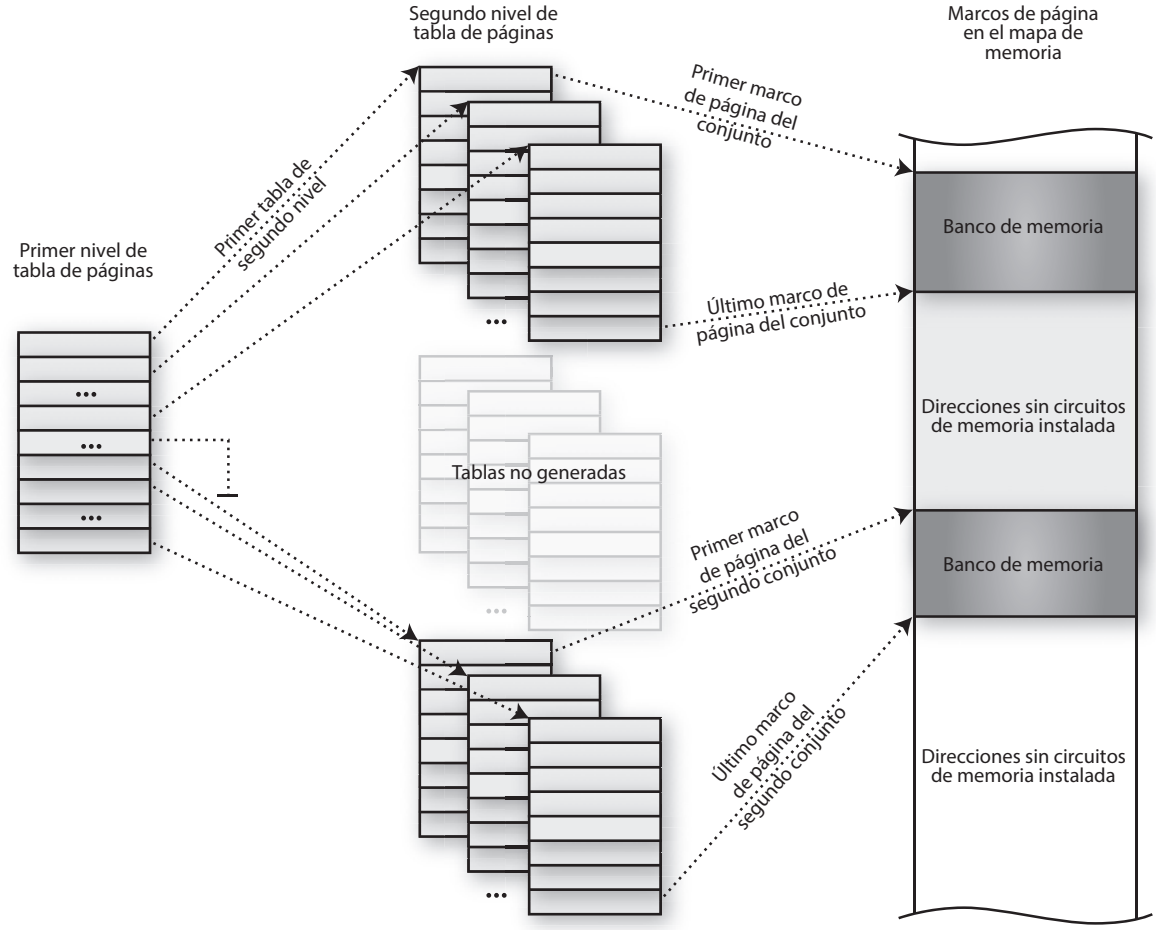


Figura 4.3 Tablas de páginas multinivel.

dexadas por la segunda parte de la dirección lógica y contienen los offset de las páginas en la memoria física.

Los rangos de memoria que no se encuentren instalados se corresponderán con tablas del segundo nivel que no necesitamos generar, y simplemente podemos marcar las entradas correspondientes en la tabla de primer nivel para indicar que corresponden con rangos inválidos de direcciones.

Arreglos asociativos de páginas o tablas invertidas

Las tablas multinivel proporcionan la flexibilidad y velocidad de referencia necesarias para mapas de memoria dispersos; sin embargo, en las arquitecturas de 64 bits la can-

tidad de memoria que se puede direccionar es de magnitud de muchos órdenes mayor, por lo que se requieren medidas adicionales para reducir el tamaño de las tablas de páginas y acelerar el acceso, como los arreglos asociativos (Hash) de páginas, que consiguen esto al generar entradas no por la dirección lógica a resolver, sino por la dirección física a recuperar. El objetivo es tener una tabla de páginas justamente del tamaño necesario para la memoria presente y no según la enorme capacidad de direccionamiento del procesador, a menudo subutilizada. Como en estas se tienen las direcciones lógicas supeditadas al conjunto de direcciones físicas, al contrario de las tablas de páginas tradicionales, también se les conoce como tablas de páginas invertidas.

Se espera que una parte importante de las referencias a memoria sean atendidas por la TLB. Para resolver una referencia de memoria que incurra en un fallo de la TLB necesitamos ubicar la entrada con el offset en nuestra tabla, lo que no resulta sencillo ya que no podemos asegurar qué posición ocupa en la tabla solo por su número de página. Recuérdese que la memoria instalada no está continua en el mapa de memoria, y para ubicarla se usa el arreglo asociativo o Hash.

El Hash de las páginas de memoria utiliza una función matemática que genera valores casi únicos para cada página en memoria física, por lo que en esta obra no ahondaremos en la forma como trabaja una función de Hash y solo nos limitaremos a aceptar que algunas páginas (la minoría) tienen el mismo valor Hash porque el espacio de valores generado por la función es considerablemente menor que el espacio de las direcciones accesibles en nuestro sistema operativo. A pesar de que los valores resultado de dos direcciones parecidas serán muy diferentes entre sí, con esto logramos reducir el tamaño de la tabla asociativa, ya que el valor que habrá de almacenarse por cada página solo requiere ser el resultado de la función Hash para la dirección lógica de la página.

La tabla asociativa se indexa por los valores resultantes de la función Hash sobre las direcciones lógicas, a menudo unidas al número de proceso al que se asociaron. En este caso, cada entrada contendrá una lista ligada de registros de página con el valor completo de la dirección lógica y el proceso, el offset para el acceso físico a la memoria y un apuntador al elemento siguiente de la lista. Así, cuando debemos resolver una dirección lógica, se calcula el valor Hash de la dirección lógica a resolver considerando el proceso, con lo que se recupera la ubicación de la lista ligada y podemos revisar solo unas pocas entradas (idealmente solo una) para identificar el offset de la página en la memoria física. Una vez obtenido el valor del offset, este se coloca en la TLB con el fin de mejorar las posibilidades de que un acceso posterior lo encuentre ahí y se evite una nueva consulta a la tabla de páginas en memoria RAM.

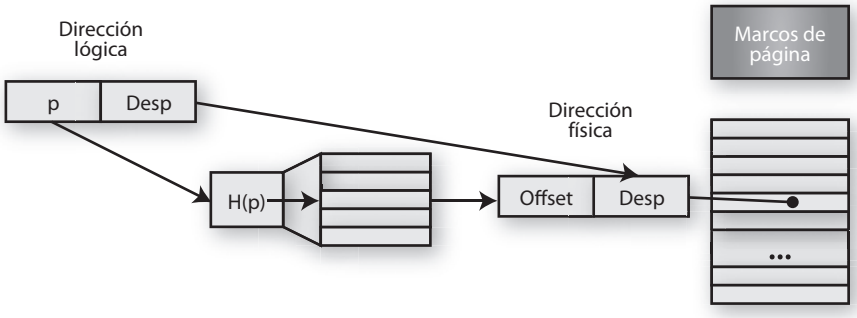


Figura 4.4 Solución de dirección lógica página (p), desplazamiento (Desp) mediante una función Hash (H).

Liberación de páginas

La liberación de la memoria que ya no requiere ninguno de los procesos es uno de los aspectos básicos que deben considerarse en el diseño y la realización de software. Por lo común, los lenguajes de alto nivel exponen a sus usuarios funcionalidad, que va desde la liberación explícita de los segmentos que reservaron hasta la recolección de basura, constituida por un conjunto de algoritmos que pretenden ahorrar al desarrollador el trabajo de liberar de forma explícita cada una de las estructuras que cree, con su correspondiente posibilidad de error y de omisión asociada. Para ello verifican si los objetos que fueron creados por un programa aún pueden ser utilizados o, en caso contrario, liberar la memoria que ocupan.

En ambos casos, es responsabilidad del compilador y la funcionalidad de las aplicaciones enviar al sistema operativo las instrucciones para liberar las páginas o segmentos de memoria que ya no requiere; por tanto, la principal preocupación del sistema operativo debe ser la fuga de memoria que puede ocurrir cuando, por alguna razón, una aplicación no libere toda la memoria que ha reservado.

En principio, los modelos de paginación puros pueden controlar esto al terminar los procesos, ya que las tablas de páginas indican los procesos para los que operan, y una vez que termina el proceso puede procederse a liberar la memoria correspondiente. La excepción está en la memoria compartida, ya que en caso de que los procesos no liberen la memoria explícitamente, el sistema operativo no puede asegurarse de que esa memoria en efecto ya no se utilizará y, por tanto, debe respetarla.

En esquemas de administración de memoria de segmentación, el sistema operativo puede liberar, a la terminación de los procesos, sus

Importante

♥ **Usuario.** Muchos compiladores refuerzan las advertencias respecto al manejo de memoria y las herramientas de análisis de código para la prevención y detección de errores, como splint. Suelen prestar especial atención a detectar las prácticas que lleven a fugas de memoria. En caso de programar

bajo un paradigma orientado a objetos o en un lenguaje con recolectores de basura, es recomendable revisar los mecanismos para detectar los objetos a liberar para hacer uso efectivo de estas facilidades.

♠ **Líder.** Desde el punto de vista de administración de sistemas, es indispensable estar al tanto de políticas como la del porcentaje de páginas libres que requiere Linux al momento de planificar el uso de nuestra memoria en aplicaciones que se benefician de mantener muchos datos en RAM, como pueden ser las máquinas virtuales o los servidores de bases de datos. Si solo observamos la cantidad de memoria libre al inicio del sistema, podríamos incurrir en una gran cantidad de fallos de página mientras aún tenemos memoria libre.

segmentos de código, datos y stack; sin embargo, no tiene conocimiento explícito de los segmentos usados para estructuras de datos dinámicos, por lo que de manera regular se deja a cargo del programa exclusivamente la responsabilidad de liberar todos los rangos de memoria que ocupe.

Sin embargo, no todas las páginas que se liberen pertenecerán a procesos que han terminado; por ejemplo, Linux procura tener al menos 10% de sus páginas de memoria libres para poder atender con rapidez las peticiones de memoria adicional de parte de los procesos, lo cual implica que debe liberar páginas incluso cuando los procesos que las reservaron aún están activos.

Aunque Linux es un sistema híbrido y no de paginación pura, constituye un ejemplo en extremo ilustrativo. Este sistema operativo debe buscar páginas que no hayan sido utilizadas recientemente y prepararlas para la carga de otras páginas, para lo que debe programar la escritura a almacenamiento secundario, en caso de que hayan sido modificadas, y luego sobreescribir los valores presentes en sus celdas de memoria con ceros, en caso de que así lo indiquen las políticas de protección. Con esto puede asegurar que cuando se requiera una página adicional para almacenar datos de un proceso, se tenga una página libre en memoria preparada para atender la petición, sin necesidad de bloquear al proceso que la solicitó.

Otro aspecto que debe considerarse en la liberación de páginas son aquellas que nunca deben ser desplazadas a memoria secundaria (por ejemplo, algunas secciones del sistema operativo). Dichas páginas reciben el nombre de memoria asegurada y son marcadas específicamente en la tabla de páginas.

Tamaño de páginas

En el diseño de un administrador de memoria de paginación es indispensable considerar el tamaño de las páginas. Aunque muchas arquitecturas actuales han optado por soportar páginas de 4 KB como norma, los procesadores suelen soportar al menos un rango de tamaños de páginas.

En caso de que un sistema operativo decida emplear páginas de menor tamaño, ese hecho lo obliga a tener tablas de páginas con un número mucho mayor de entradas, ya que para la misma cantidad de memoria RAM tendrá un mayor número de páginas

que controlar. Dado que la memoria disponible en las TLB es limitada, y a fin de evitar la sobrecarga en accesos a memoria para resolver direcciones lógicas, no se recomienda reducir los tamaños de página más allá de los 4 KB.

Por el contrario, si el tamaño de página se incrementa, tendremos ventajas en el número de entradas que requerimos para la tabla de páginas; sin embargo, al incrementar el tamaño de las páginas también incrementamos el desperdicio de memoria en fragmentación interna, en especial en los elementos dinámicos de menor tamaño, que a menudo no ocuparán sino una fracción de una página grande, los cuales pueden ser muchos al usarse para pequeñas estructuras de datos dinámicas generadas por los procesos.

Algoritmos de reemplazo de páginas

Al operar en sistemas con memoria virtual, de antemano sabemos que en ocasiones será necesario cargar a memoria principal una página que solo se encuentra en almacenamiento secundario. Todos los marcos de página con frecuencia estarán ocupados, por lo que deberemos desbloquear alguno de estos para liberar memoria y poder cargar la página que falló.

Al proceso de elegir una página para liberar el marco de página que ocupa y cargar una página que falló se le conoce como **reemplazo de páginas**. Como esto impide el progreso de los procesos en lo que se realiza y consume recursos de la CPU y de dispositivos de entrada y salida, debe elegirse la página a reemplazar de forma que se minimice el número de reemplazos de página futuros.

Paginación por demanda

Al iniciar un proceso, por lo general se reserva en memoria la totalidad de las páginas que necesitará; así, conforme se desarrolla su ejecución, el proceso irá reservando memoria adicional para sus estructuras de datos dinámicas. Si la capacidad de la memoria lo permite, el ciclo de vida completo del proceso puede usar exclusivamente páginas en memoria principal para todas las páginas del proceso.

Una parte considerable de la funcionalidad puede estar diseñada para situaciones límite, rutinas que atiendan situaciones infrecuentes, búfers que por lo común solo utilizan una fracción de su capacidad máxima, etcétera. Gracias al principio de vecindad (Locality) se sabe que en diversos periodos de la ejecución del proceso solo se usa un subconjunto de la memoria total asignada al proceso.

La paginación por demanda parte de la premisa de cargar en memoria solo las páginas que el proceso requiera utilizar activamente en memoria, con lo cual se evita cargar páginas que no están en uso e incluso acelerar el proceso de carga inicial del proceso.

Por otra parte, iniciar la ejecución del proceso con una gran cantidad de fallos de página no es eficiente pues cada atención de fallo de página requiere preservar el estado del proceso, bloquear el proceso terminando su turno de ejecución y realizar otras actividades que constituyen una sobrecarga que es mejor evitar. Por ello, la mayoría de los sistemas operativos modernos no usan procesos completamente por demanda (también conocidos como "flojos"o Lazy Loaders), ya que resulta más conveniente iniciar con la carga de los elementos principales del proceso.

Si podemos cargar sobre demanda, las librerías de carga dinámica y la memoria compartida pueden posponer la reserva de memoria y acelerar con ello la primera etapa de carga en memoria de los procesos.

Independientemente del inicio del proceso, cuando el nivel de multiprocesamiento o el consumo de memoria de los procesos activos rebasan la capacidad de la memoria RAM, se impone trasladar algunas páginas al almacenamiento secundario con el fin de liberar espacio para las páginas que se requiere utilizar. A lo largo de la ejecución de los procesos tendremos necesidad de usar páginas que fueron trasladadas a almacenamiento secundario, por lo que de manera inevitable siempre tendremos fallos de página.

Para que el desempeño de nuestro sistema sea aceptable debemos minimizar el número de fallos de página y elegir con sumo cuidado las páginas a reemplazar, por lo que se ha dedicado mucha atención al diseño de algoritmos con ese fin. A continuación se citan algunos de los algoritmos que ilustran mejor los mecanismos que se usan en la actualidad.

Algoritmo ideal (óptimo) de reemplazo de páginas

La mayoría de las veces podemos expresar nuestro objetivo con un algoritmo de términos muy sencillos:

> En caso de un fallo de página cuando no hay memoria disponible, debe reemplazarse la página que no se usará durante el más largo periodo.

Este algoritmo simplemente expresa que la página que debemos sacar de la memoria RAM debe ser aquella que puede generar un fallo de página durante el mayor tiempo posible, lo que disminuye por definición la frecuencia de los fallos de página y permite un rendimiento máximo.

Aunque este algoritmo no puede implementarse por la imposibilidad de predecir el comportamiento futuro de todas las posibles aplicaciones del sistema, este resulta relevante como expresión del objetivo.

FIFO – First In, First Out

Una forma de atender los fallos de página, que no pretende aproximarse al algoritmo ideal, consiste en reemplazar las páginas que llegaron primero a la memoria, con el fin de que las primeras páginas en entrar sean las primeras en salir. De ahí su nombre.

Aunque este algoritmo es sencillo, no proporciona un buen rendimiento justamente por el principio de vecindad, ya que en un periodo las páginas más susceptibles de ser usadas al inicio del periodo también son las más probables de ser usadas al final, y no obstante serán las primeras en ser desplazadas al almacenamiento secundario, lo que no tenderá a minimizar los fallos de páginas.

NRU – Not Recently Used

Como una aproximación al algoritmo ideal, se puede registrar el uso de las páginas a lo largo del tiempo para mantenerlas cargadas en memoria principal, ya que es posible suponer, de acuerdo con el principio de vecindad, que estas pertenecen al conjunto de páginas que el proceso está utilizando en ese periodo y, por tanto, deben permanecer en la memoria principal.

Para ello se requiere agregar a la información de la tabla de páginas dos bits por cada página, los cuales se describen a continuación.

- **R – Read.** Este bit debe encenderse cada que se realice una operación sobre la información de la página.
- **W – Write.** Este bit debe encenderse cuando se modifique la información de la página, para indicar que la copia presente en almacenamiento secundario ya no está actualizada.

Por lo general, los bits R tenderán a encenderse, razón por la cual se requiere un mecanismo para apagarlos, a fin de que sigan siendo útiles para diferenciar las páginas que no han tenido uso reciente de aquellas que sí. Mediante dicho mecanismo, todos los bits R se apagarán periódicamente; después de apagar todos los bits R hay un periodo en que no se cuenta con información para discriminar las páginas.

Cuando se requiere un reemplazo de página, las páginas en memoria principal se agrupan en cuatro grupos, de acuerdo con los valores que tengan en los bits R y W, como se observa en la tabla 4.1.

Tabla 4.1

R	W	Categoría
0	0	No se ha usado ni modificado.
0	1	No se ha usado, pero tiene modificaciones.
1	0	Se ha usado y no tiene modificaciones.
1	1	Usada con modificaciones.

Las páginas que no tienen modificaciones son más económicas de reemplazar porque no se requiere escribirlas a almacenamiento secundario, ya que la copia aún corresponde con la información que está en memoria y podrá recuperarse en el futuro.

NRU de segunda oportunidad con lista circular (reloj)

Para evitar el reinicio periódico de los bits R se tiene un refinamiento de NRU en el que se revisan en turno las páginas, y en caso de que el bit R esté encendido se les da una segunda oportunidad, pasándolas por alto y apagando su bit R. Para organizar el recorrido de las páginas, evitando iniciar siempre por las mismas en un FIFO, estas se organizan en una lista ligada circular y se maneja un cursor para recorrer la lista en orden.

Por ejemplo, supongamos una memoria virtual que tiene cuatro páginas cargadas, a las que llamaremos 1, 2, 4 y 5, en las que colocamos diversos valores en sus bits R y W

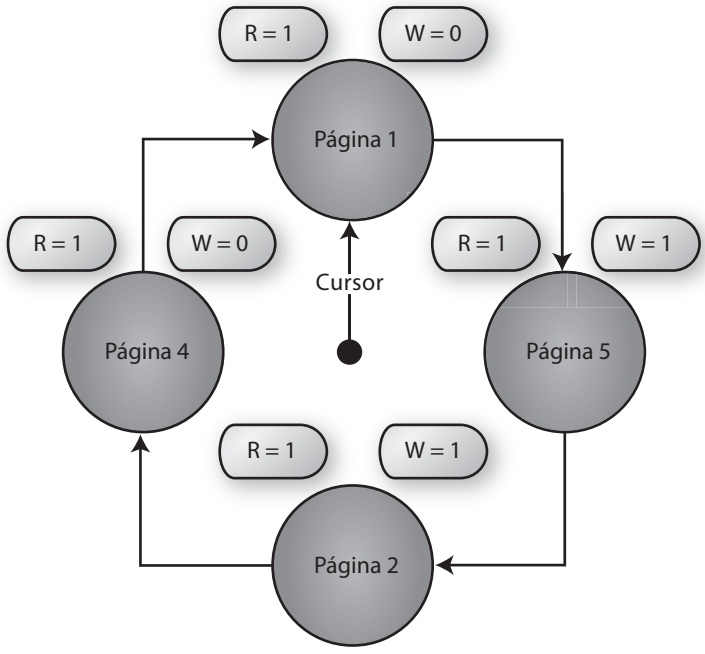


Figura 4.5 Lista circular con las páginas 1, 2, 4 y 5.

dependiendo de las operaciones que realizaron anteriormente. Otras páginas se encuentran en almacenamiento secundario y no se incluyen en la lista circular. Podemos representar la lista circular con la figura 4.5.

Si a continuación un proceso requiere usar una página que se encuentra exclusivamente en almacenamiento secundario (digamos, la página 3), se presenta un fallo de página que atenderemos con el algoritmo de segunda oportunidad, iniciando por la página señalada por el cursor (la página 1), verificamos el estado del bit R, al estar encendido este se apaga, pasamos el cursor a la página siguiente y repetimos la verificación (veáse figura 4.6).

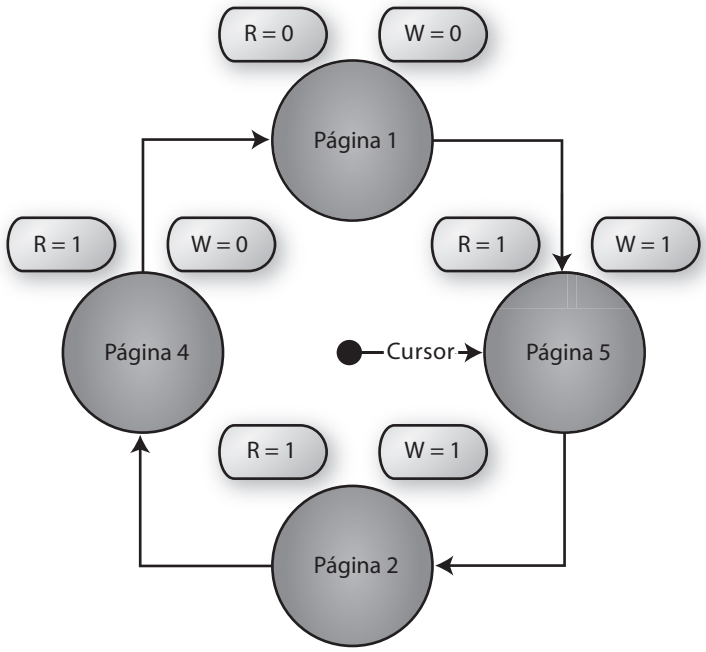


Figura 4.6 Estado de la lista luego de dar una segunda oportunidad a la página 1.

Como todos los bits R estaban encendidos al inicio, deberemos apagarlos para las páginas 5, 2 y 4, también regresando a la página 1 y buscando una página para reemplazar (veáse figura 4.7).

En la segunda ocasión que verificamos la página 1 encontramos el bit R apagado, por lo que procedemos a reemplazar esta página en la memoria RAM. El valor de bit W nos indica si tenemos una copia en la memoria secundaria, o en caso de estar encendido, que es necesario escribir la información antes de cargar la página que falló. En este caso, como W = 0, se procede a cargar la información sin escribir el contenido de la página 1 en almacenamiento secundario. La información cargada se usará, por lo que el bit R debe encenderse para la página 3, y el valor del bit W dependerá del tipo de

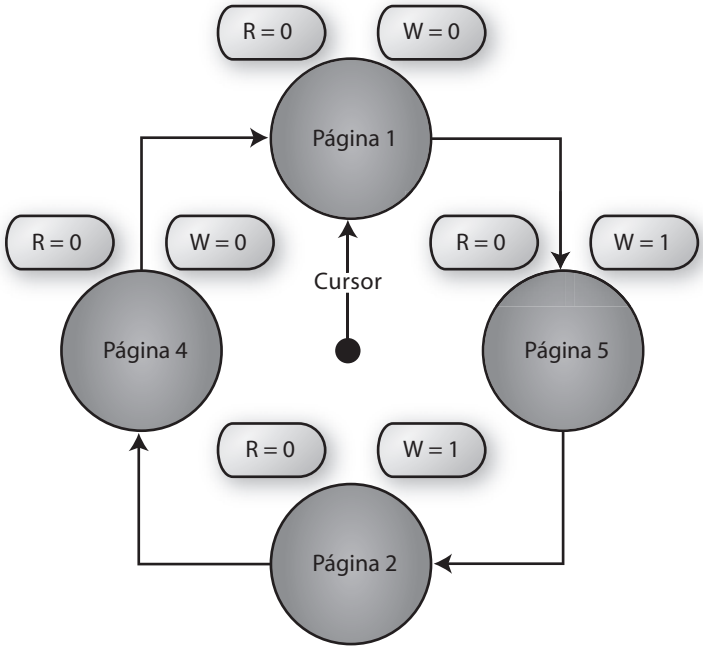


Figura 4.7 Estado de la lista luego de apagar todos los bits R.

operación que se realice (supongamos que se modifica la información, por lo que se tendrá un valor 1).

Al concluir el reemplazo de página se avanza el cursor a la página siguiente.

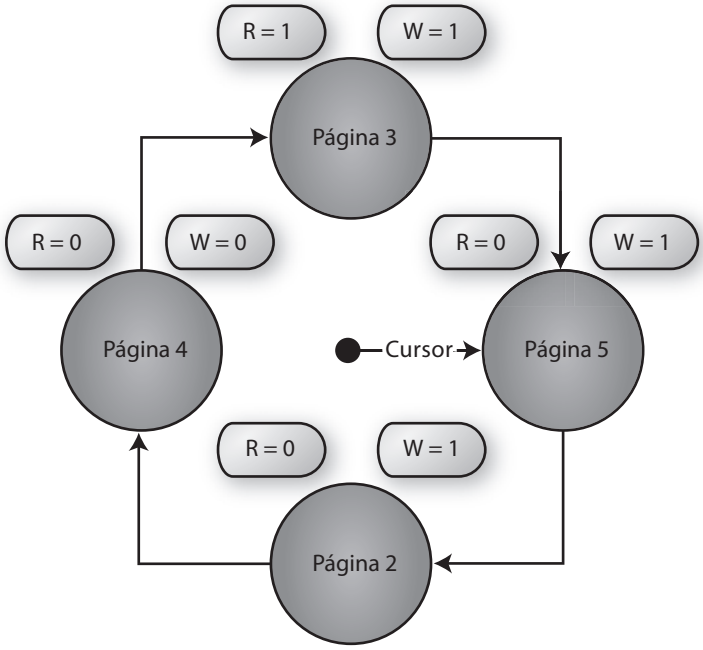


Figura 4.8 Estado de la lista luego de reemplazar la página 1 por la 3.

Paginación anticipada

El concepto de **paginación anticipada** parte de identificar el conjunto de páginas que una aplicación utilizará en el futuro cercano y procurar que el conjunto de esas páginas se encuentre completo en memoria antes de proseguir con la ejecución del proceso, con lo que se evitarán interrupciones y cambios de contexto, y es posible mejorar el rendimiento del sistema.

Conjunto de trabajo

El concepto de vecindad se desarrolló durante el proceso de probar diversos algoritmos para intercambio automático de páginas, mientras se estudiaba el conjunto de páginas que un proceso utilizaba durante un periodo.

Al aplicar este concepto para aproximarnos al algoritmo ideal, podemos buscar el conjunto de páginas que el proceso está utilizando, al que llamaremos **conjunto de trabajo**.

De este modo, podemos afirmar que las páginas que no pertenecen a ese conjunto son las que se usarán durante más tiempo. Incluso si no podemos asegurar cuál de las páginas que no se han utilizado será la que se emplee el mayor tiempo, como indica el algoritmo ideal, esta aproximación reducirá de manera importante los fallos de página, lo cual resulta de gran utilidad.

El conjunto de trabajo se define formalmente como el conjunto de páginas que han sido utilizadas durante un tiempo no mayor a un umbral llamado τ (tau), previo al momento actual.

Para implementar un algoritmo de reemplazo de páginas que contemple el conjunto de trabajo, se requiere registrar el momento en el que se realizan las referencias a las páginas y posteriormente asegurar que todas las páginas del proceso que pertenezcan al conjunto de trabajo se encuentren cargadas en memoria principal.

Esto hace posible que, en caso de que haya un fallo de página para un proceso, podamos postergar la reanudación de este hasta que hayamos cargado todo su conjunto de trabajo y no solo la página que falló; eso tenderá a reducir el número de fallos de página y los cambios de contexto.

Sin embargo, su implementación puede ser muy costosa, ya que se requiere considerar información adicional por cada entrada en la tabla de páginas para copiar el valor del registro del temporizador (Time, o T para abreviar) del procesador que contabiliza los milisegundos, conforme a la medición estándar del tiempo, lo que suele ser 64 bits por cada página. Además, para verificar cuál es el conjunto de trabajo se requeriría revisar de manera continua los valores de muchos registros, lo que también con-

sumiría ciclos de la CPU. Por esta razón, muchos algoritmos basados en el conjunto de trabajo no han sido usados extensamente en sistemas operativos, por lo que su tratamiento en este libro solo se limita a lo que se ha expuesto hasta el momento.

Conjunto de trabajo de reloj (WSClock)

Un algoritmo que sí ha sido implementado con algunas optimizaciones adicionales en diversos sistemas operativos de gran aceptación es el conjunto de trabajo de reloj o WSClock. Dicho algoritmo parte de los conceptos del NRU con lista circular (reloj), que consiste en una lista circular con las páginas en memoria principal, bits R y W para determinar la actividad que ha tenido la página y un cursor para indicar cuál es la página que debemos revisar a continuación, sin necesidad de recorrer toda la lista. A estos se agrega un campo de momento de última utilización donde se copia el valor del registro Time cuando se emplea la página.

Para realizar un reemplazo de página, este algoritmo comienza a revisar en la ubicación del cursor, en la lista circular, si el bit R está encendido. Así, si está encendido, entonces sabremos que esta página ha tenido actividad reciente, por lo que se apaga el bit R y se pasa a revisar la página siguiente.

Cuando el bit R está apagado deberemos verificar si el momento de última utilización se encuentra dentro del periodo T; en caso de estarlo, pertenece al conjunto de trabajo y se procede a verificar la siguiente página. En caso de no pertenecer al conjunto de trabajo, deberemos verificar el bit W para determinar si existe una copia de la información en el almacenamiento secundario. Si el bit W está encendido, no tenemos una copia y debe escribirse la información de la página al almacenamiento secundario antes de liberar la página. Entonces se programa la escritura, pero se prosigue a revisar la página siguiente.

Cuando los bits W y R están apagados y la página no está en el conjunto de trabajo, podemos proceder a liberar la página y cargar la página que falló en esa ubicación de memoria.

Puede suceder el caso de que se recorra toda la lista sin encontrar una página en esa situación; cuando esto ocurre se revisa lo siguiente:

- Si alguna escritura de página fue programada, sabemos que no todas las páginas pertenecen al conjunto de trabajo, entonces el proceso se bloquea, y espera a que la operación de escritura de página termine y se reemplaza la página que haya sido escrita.
- En caso de que no se haya programado ninguna escritura, sabemos que todas las páginas pertenecían al conjunto de trabajo; debido a que no se tienen más elemen-

tos para discriminar, entonces se reemplaza la primera página a la que apunte el cursor que tenga su bit W apagado.

Hiperpaginación

También conocido como *trashing*, es un problema en la memoria virtual en la que, al incrementar el número de procesos en ejecución, en vez de elevar el uso de la CPU este disminuye rápidamente, al tiempo que el sistema satura sus operaciones de lectura y escritura al almacenamiento secundario. Su estudio y solución llevó al modelo de vecindad, ya que se descubrió que se presenta cuando el conjunto de trabajo de los procesos activos excede a la memoria RAM disponible.

Lo que sucede es que cuando un proceso se activa, este no tiene disponibles todas las páginas que usará en su turno de ejecución, por lo que debe reemplazar páginas del conjunto de trabajo de otros procesos para cargar las propias, pero se bloquea en el proceso. Cuando se activa otro proceso debe hacer lo mismo, y eso ocasiona que los otros procesos a su vez tengan que reemplazar páginas para completar su conjunto de trabajo, lo que resulta en un decremento considerable del rendimiento del equipo.

Usando el concepto de vecindad, y en particular del conjunto de trabajo, se puede resolver la hiperpaginación rehusando la atención de la CPU a procesos nuevos si no hay memoria suficiente para cargar los conjuntos de trabajo de todos los procesos actuales más el conjunto de trabajo del nuevo proceso.

En sistemas de tiempo compartido, a menudo se deja al usuario la responsabilidad de detectar la hiperpaginación y terminar los procesos que considere pertinente, aunque usualmente se termina por recomendar ampliar la capacidad de memoria del equipo en caso de ser posible.

Intercambio (Swap)

En los sistemas operativos que usan la segmentación en vez de la paginación, también se puede hacer uso del almacenamiento secundario para ampliar la capacidad de almacenamiento de información para los procesos; sin embargo, no tenemos el mismo nivel de granularidad en cuanto a los elementos que podemos mover.

El intercambio parte de la noción de mover segmentos completos al almacenamiento secundario. Sin embargo, esto solo debe hacerse cuando se espera que el proceso que los utiliza no tenga ninguna actividad por un periodo considerable. Por tanto, el proceso de intercambio suele utilizarse solo para procesos que se encuentran inactivos durante periodos considerables y se activan solo para responder a diversos eventos. Por

ejemplo, diversos servicios de Internet son atendidos por procesos que están bloqueados en espera de que se reciban paquetes de red en un puerto en particular.

Cuando los segmentos de un proceso se mueven al almacenamiento secundario, se modifican sus entradas en la tabla de segmentos para indicar la ubicación en el área de disco donde se escribió su información. Debido a que no están disponibles en memoria principal cuando se requiere usarlos, se deben reservar rangos de memoria RAM del tamaño correspondiente y realizar los ajustes propios de una relocalización en las tablas de símbolos y la tabla de segmentos.

Sistemas híbridos

Desde el diseño del sistema operativo Mutlics se planteó combinar las virtudes de la segmentación con las de la paginación, con el fin de lograr exponer las interfaces convenientes de programación de la primera con la flexibilidad y la memoria virtual de la segunda. Además, los estudios e ideas que se plantearon durante la concepción de este sistema operativo fueron fundamentales para el desarrollo del concepto de vecindad, y son la base de la mayor parte de los sistemas de administración de memoria actuales en sistemas de tiempo compartido, los cuales combinan ambos esquemas para lograr una funcionalidad accesible, eficiente y de grandes prestaciones para el desarrollo de aplicaciones.

El concepto básico es emplear las interfaces de programación de segmentación para las aplicaciones mediante la implementación de los segmentos a través de conjuntos de páginas en vez de rangos de memoria contigua.

A continuación se describen algunos de los ejemplos más destacados de sistemas híbridos.

Hoy día, la mayoría de los sistemas que emplean Internet usan tanto la paginación como la segmentación. Se trata de dispositivos móviles basados en procesadores de la familia ARMv7-A. ARM, los cuales dominan más de 90% del mercado de procesadores móviles y son muy eficientes en lo que se refiere al consumo de energía. Estos también tienen versiones para uso de segmentación pura, para aplicaciones de tiempo real, y sin protección de memoria, para aplicaciones embebidas de propósito particular. En la familia A de estos procesadores es posible encontrar registros de segmentos y de desplazamiento, junto con un modelo de paginación de tamaño fijo de 4 KB,

Importante

♥ **Docente.** El uso de la memoria en las aplicaciones parece engañosamente simple: se solicitan los rangos de memoria deseada, se manejan las excepciones en caso de no obtenerse y se liberan al terminar de usarlas (o simplemente se dejan de usar si se cuenta con un recolector de basura); sin embargo, en realidad presenta una cierta complejidad que debe ser comprendida. De este modo, el esfuerzo de desarrollo debe contemplar las previsiones necesarias tanto en el diseño como en las pruebas, la interpretación y la solución de problemas que pueden surgir durante la operación.

♠ **Líder.** Al usar sistemas híbridos, los proyectos de desarrollo de aplicaciones deben tomar medidas para vigilar que no se

aunque soporta páginas de 16 KB, 1 MB y 16 MB, así como una tabla de páginas de dos niveles con una TLB y otras prestaciones para el manejo óptimo de la paginación, análogas a las de los procesadores Intel. Para administrar la memoria, se asigna a cada proceso su propia tabla de páginas y se definen dos niveles de protección de las páginas de memoria: sistema y aplicación de usuario.

Otra de las implementaciones que goza de gran popularidad es la empleada por los procesadores Intel de la familia x86, ya que estos han logrado mantener la compatibilidad desde sus modelos 286 hasta los actuales procesadores de 64 bits de la serie i (i3 a i7). Algunas de las características principales de la arquitectura de la memoria virtual implementada por los procesadores Intel ya fueron mencionados con anterioridad, por lo que solo resta mencionar que las API de memoria de los sistemas operativos actualmente en uso en estos procesadores (es decir, Linux y Windows) realizan la asignación de segmentos usando directamente direcciones de paginación y considerando toda la memoria disponible como un único segmento que parte de la base de la memoria y que emplea el modelo de paginación para controlar las asignaciones.

incurra en fugas de memoria, ya que la memoria virtual ayuda a que ese problema tenga un impacto mínimo en los cortos tiempos de ejecución de las pruebas, enmascarando situaciones que pueden convertirse en críticas en ambientes de producción. Comprender de manera adecuada cómo el lenguaje y el sistema operativo manejan la memoria y el uso de herramientas estáticas de análisis de código son dos medidas muy recomendables.

Amenazas y protección

El problema más común para la administración de memoria es la fuga de memoria, debido a malas prácticas de desarrollo, por lo que hay una serie de herramientas para atacar el problema, entre las que se incluyen herramientas eficientes para monitorear el consumo de memoria a lo largo del tiempo de los procesos y del sistema en general. El administrador debe mantenerse atento a los niveles de utilización de la memoria y tomar medidas especiales en caso de que note que el consumo de esta comienza a crecer sin que el número de las aplicaciones o los datos que manejan también crezcan. Aunque suele ser un error sin malicia, las fugas de memoria también pueden usarse en contra de un sistema, al proporcionar un medio al atacante para detener la operación del sistema, lo que genera carga de trabajo aparentemente correcta.

Otra amenaza que se ha ido popularizando a últimas fechas son los desbordamientos de buffer. En estos ataques, por lo general se

Importante

♠ **Arquitecto.** Como desarrolladores es muy importante validar todas las entradas que se reciban por parte de los usuarios o de otros sistemas de información, si desde el diseño no podemos asegurar que dicha entrada ya ha sido validada, no solo por la longitud que puede tener sino también porque pueda contener otras formas de información que perjudiquen al sistema.

♥ **Usuario.** Este tipo de ataques son particularmente serios cuando se orientan a programas que

se ejecutan con privilegios de administración y atienden peticiones de usuarios normales, ya que mediante estos se pueden tomar acciones desde una cuenta de usuario sin privilegios, que logren acceso a los privilegios de administración. A estos ataques en particular se les conoce como Exploits, debido a que explotan defectos conocidos en aplicaciones del sistema operativo y son una de las causas por las cuales se deben instalar de manera regular las actualizaciones de seguridad del sistema operativo, ya que con estas se corrigen los defectos conocidos al mejorar las validaciones de las entradas en los programas.

abusa de una entrada de información de un programa para sobreescribir partes del proceso y hacerlo fallar o modificar su comportamiento. Esto se debe al hecho de que en el uso actual de los esquemas híbridos de paginación y de segmentación no se emplean los registros de la CPU para limitar el tamaño de las páginas asignadas (cuando están disponibles) y a que los datos a menudo comparten los segmentos y las páginas con otros elementos como las tablas de símbolos.

Así pues, el ataque consiste en entregar, durante una operación del proceso, una cantidad de información mucho mayor a la que se espera, en especial con cadenas de caracteres, y si el proceso no verifica la magnitud del dato recibido, puede intentar copiarlo a una de sus variables. Lo que sucede es que el dato rebasa los límites de la variable (o búfer) y prosigue sobreescribiendo valores en el segmento de datos que la alberga. Con un estudio cuidadoso de la memoria ocupada por el proceso, el atacante puede incluso determinar el tamaño y contenido de la cadena proporcionada para alcanzar y modificar una tabla de símbolos del proceso, con lo que puede modificar su comportamiento de forma efectiva.

Administración de memoria en diversos sistemas operativos

Linux

El desarrollo de Linux está íntimamente relacionado con la disponibilidad de computadoras personales, en particular con la familia Intel. Como ya se señaló, los procesadores basados en el 80268 y posteriores incluyen funcionalidad para dar soporte a la paginación, de modo que el sistema operativo se apoya en este para realizar la implementación de su administración de memoria (en este caso, híbrida con segmentación).

En cuanto a su tabla de segmentos, Linux permite, entre otras cosas, generar segmentos de memoria contigua con páginas grandes, lo que le ayuda a que áreas del sistema operativo ocupen memoria contigua sin preocuparse de los límites de tamaño de página. Además, en la tabla de segmentos se tiene el campo de granularidad, que indica si el segmento está implementado con páginas, como todos los segmentos de aplicaciones de usuario, o si se trata de un rango contiguo continuo.

Linux también tiene atributos para indicar el tipo de segmento, de código, de datos o sistema, y con base en ello saber cuál de los registros de direcciones lógicas usar y sus

derechos de acceso (lectura, escritura, ejecución, etc.) al momento de cargar sus direcciones base en el procesador.

Adicionalmente, Linux tiene un esquema de niveles de privilegio que usa para proteger a los segmentos de memoria de aplicaciones del sistema operativo, con objeto de que no sean modificados por aplicaciones de usuario. Dicho esquema se almacena en el campo de Descriptor Privilege Level de la tabla de segmentos y tiene cuatro niveles, que van desde el 0, para el núcleo del sistema operativo, al 3, que permite que cualquier aplicación lo acceda; por ejemplo, una librería compartida de carga dinámica con código que cualquier proceso podrá ejecutar, incluso si es del sistema operativo, y no podrá modificar.

La tabla de páginas es de tres niveles en arquitecturas de 32 bits, con la dirección lógica dividida en tres partes: Directorio, con los 10 bits más significativos; Tabla, con los 10 bits intermedios, y Offset, con los 12 bits menos significativos.

Para resolver las direcciones lógicas se usan dos tipos de tablas de páginas, el Directorio de páginas y la Tabla de páginas, para los primeros dos niveles, y posteriormente se aplica el Offset para determinar la posición relativa a la ubicación indicada en la tabla de páginas de la página particular que se está buscando.

La paginación tiene su propio modelo de protección que complementa al de los segmentos. Este modelo consta de dos niveles de privilegios: uno que solo permite acceso a las páginas en modo de *kernel* y otro que permite el acceso en general y depende del detalle proporcionado por el segmento.

En cuanto a los tipos de operaciones permitidas para la página, Linux solo limita a que sean de solo lectura o de lectura y escritura.

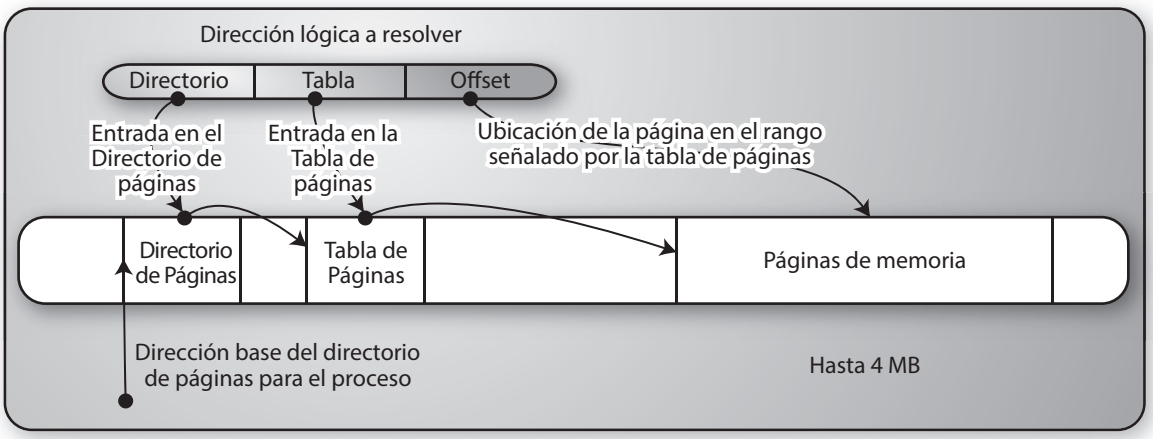


Figura 4.9 Obtención de una dirección física en Linux.

En arquitecturas de 64 bits, se genera un mayor número de niveles dependiendo de la arquitectura y del tamaño de las direcciones que soporta el procesador; por ejemplo, para la familia x86_64, en la que se usan 48 bits por dirección, se generan cuatro niveles de la tabla de páginas, dividiendo la dirección en cuatro segmentos de 9 bits y al offset de 12 bits.

Para atender reemplazos de páginas, mantener cierta proporción de páginas libres e incluso para ingresar en estado de hibernación, Linux usa algoritmos que son optimizaciones de los algoritmos de reemplazo de páginas que ya se han estudiado antes; en particular, usa una versión del NRU de segunda oportunidad con múltiples filas.

Asimismo, Linux mantiene dos listas de páginas, una con las páginas que han sido utilizadas recientemente o activas y una segunda lista con las demás páginas, que se conoce como lista de páginas inactivas. Para mover una página de la lista activa a la inactiva, revisa pequeños conjuntos de páginas periódicamente, y si una página resulta como candidata a ser desplazada en dos ocasiones, la agrega a una lista de candidatas y las mueve de forma efectiva hasta que esta lista se llena. Igualmente, para mover una página de la lista inactiva a la activa, requiere que se utilice esa página en al menos dos ocasiones y usa el mismo tipo de lista de candidatas antes de moverla. Para mover páginas de la lista de inactivas a la memoria secundaria sigue un proceso análogo.

Adicionalmente, cuando se detecta que una página debe copiarse a almacenamiento secundario, se agrega a una lista para ir realizando estas operaciones de escritura. Como los mecanismos de optimización de operaciones del disco duro pueden modificar el orden de las operaciones de escritura, las operaciones son solicitadas y las páginas se van considerando como disponibles hasta que se confirma que han sido escritas al almacenamiento secundario. Incluso entonces, la información de la página se conserva para que en caso de que se presenten referencias a la página antes de que se cargue la información de otra en su marco de página, no sea necesario volver a cargarla y simplemente se reintegre a la lista de páginas inactivas.

Importante

♥ **Arquitecto.** Es notable que las API de programación para el desarrollo de las aplicaciones en sistemas de tiempo real no son tan diferentes de los correspondientes a los de las versiones de tiempo compartido de Linux. Debido a los estándares como POSIX y a la gran cantidad de trabajo invertido en automatizar y formalizar la administración de la memoria, un desarrollador puede generar aplicaciones con un mínimo de conocimiento acerca de la operación interna de la administración de memoria. Sin embargo, es recomendable que al menos las áreas de arquitectura y seguridad comprendan las características de los siste-

RT Linux

En la versión de Linux para sistemas de tiempo real, la principal variante consiste en retirar la funcionalidad de memoria virtual para evitar que la imprevisibilidad de los fallos de página impida el cumplimiento de los compromisos de atención de los procesos.

En sus versiones para sistemas embebidos, incluso elimina los niveles de protección y ejecuta los procesos y el *kernel* en un único espacio de direcciones. Asimismo, intenta evitar la asignación de memoria durante la ejecución del proceso procurando reservar toda la memoria necesaria al iniciarse estos. Todas estas consideraciones parten del objetivo de eliminar consumo de recursos por la administración de memoria que pudieran restar predictibilidad al sistema y poner en riesgo los compromisos de atención y liberar recursos para las aplicaciones.

En el caso de los sistemas embebidos, se considera que las aplicaciones en este tipo de sistemas por lo regular se conocen en su totalidad y pueden ser probadas en conjunto desde que se diseña el sistema de información, además de que evita que el usuario instale aplicaciones nuevas, lo que alivia las presiones acerca de la protección en estos sistemas y permite que estos modelos simplificados operen de manera satisfactoria.

Windows

Windows, desde su versión Vista hasta sus más actuales versiones 7 y 8, también emplea una administración de memoria híbrida para que los procesos perciban un rango de memoria contigua, que posteriormente es implementado en páginas no contiguas.

Es importante destacar que, por defecto, Windows toma los 2 GB de memoria con direcciones más altas para el uso del propio sistema operativo, aunque permite que aplicaciones con privilegios especiales reclamen más memoria, reduciendo el sistema operativo a solo 1 GB.

Asimismo, Windows usa una tabla de páginas de dos niveles; el nivel más alto es el Directorio de páginas y el segundo es conocido como la Tabla de páginas. Las direcciones lógicas constan de tres elementos: 1) el índice para el directorio de páginas (10 bits), 2) el índice para la tabla de páginas (10 bits) y 3) el índice de bytes (12 bits) que se utiliza como el Offset de la dirección física.

Como las operaciones de memoria tienen posibilidad de operar de forma concurrente, Windows emplea mecanismos de sincronización para evitar condiciones de competencia sobre las modificaciones de la tabla de páginas y de la información de las páginas que se están car-

mas operativos en que operan para desempeñarse de modo adecuado.

♠ **Líder.** En la actualidad persiste una tendencia de mercado muy importante, que utiliza la gran penetración que ha tenido Internet como un medio de conexión de dispositivos embebidos, ahorrando la necesidad de establecer infraestructuras de comunicaciones especializadas. Esto es particularmente conveniente dado el bajo costo y la facilidad de acceso a las redes inalámbricas WiFi. Sin embargo, como acabamos de ver, a menudo las previsiones de seguridad de los sistemas embebidos están diseñadas con ambientes controlados en mente, por lo que hay un creciente número de incidentes de seguridad asociados a la explotación de funcionalidad en dispositivos de propósito particular conectados a Internet, tendencia que es muy probable que crezca en muy poco tiempo. Por tanto, al integrar soluciones con este tipo de sistemas, se impone sopesar con cuidado la economía de los dispositivos contra los riesgos de seguridad asociados, incluso más allá de las necesidades de la aplicación inmediata, ya que nuestra infraestructura puede pasar a ser parte de un ataque más amplio, dirigido hacia otros sistemas, también conectados a Internet.

gando o modificando. Sin embargo, en Windows Vista usaba una sola región crítica para todos los procesadores sobre todas las operaciones, lo que perjudicaba el desempeño al serializar operaciones de procesos distintos sobre rangos de memoria separados.

Desde la versión 7, Windows adoptó la estrategia conocida como NUMA (Non Uniform Memory Access) para asociar a los procesadores, a los procesos en ejecución en estos y a los elementos de administración de memoria en uso por estos a regiones críticas independientes, lo que reducía la serialización. Esto limita a las aplicaciones a operar en "ventanas" de memoria (por ejemplo de 256 MB), las cuales pueden ser direccionadas para acceder a distintas partes de la memoria reservada para la aplicación, que puede abarcar hasta 2 GB en sistemas de 32 bits y de 4 GB en sistemas de 64 bits; sin embargo, cada una de las aplicaciones debe especificar de manera explícita el momento y la dirección de memoria a donde desean dirigir la ventana para poder aprovechar la memoria asignada cuando los requerimientos de memoria de la aplicación llegan a estas magnitudes.

En Windows se permite el uso de páginas no solo de 8 KB, sino también de páginas grandes, de hasta 16 MB. Sin embargo, el uso de páginas grandes está restringido para áreas del sistema operativo, así como para los segmentos de código, los segmentos de memoria privada y los segmentos respaldados a disco de las aplicaciones (estos últimos se programan para escribir a almacenamiento secundario en cuanto se modifican). También se permite que los controladores de dispositivos se carguen en páginas grandes.

Por lo general, Windows emplea dos niveles de protección de la memoria. Originalmente, los niveles de protección se limitaron a solo dos para soportar procesadores que solo reconocían dos niveles, aunque procesadores como los de Intel soportan cuatro niveles: el modo de *kernel*, en el que se ejecutan los servicios de sistema, y los controladores de dispositivos y el modo de aplicación, donde se ejecutan las aplicaciones de usuario y que no tiene acceso a los segmentos de memoria de modo *kernel*.

Durante la ejecución de un proceso es común que el sistema alterne entre el modo *kernel* y el modo de usuario, ya que, mediante llamadas al sistema, las aplicaciones pueden hacer uso de los recursos del *kernel* del sistema operativo.

En sus versiones de 32 bits, no se proporciona protección de memoria de distintos segmentos que se encuentren dentro del espacio

Importante

- **Líder.** Para evitar que los controladores de dispositivos de entrada y de salida atenten contra la seguridad del sistema en Windows de 32 bits, Microsoft ha optado por un esquema de certificaciones de software y de programas de prueba, que estos deben completar para obtener el aval de la compañía; asimismo, empezó a exigir que el código ejecutable fuera firmado con algoritmos criptográficos para establecer mecanismos que puedan verificar si el controlador que el usuario está instalando proviene del fabricante y ha pasado por estos procesos. En Windows 7 y versiones posteriores, Microsoft ha extendido el uso de firmas criptográficas a casi todas las rutinas de los programas, lo que le ha permitido fortalecer su recaudación por la venta de los servicios de emisión de firmas, aunque no ha demostrado ser suficiente para evitar que se distribuyan y usen tanto programas realizados sin certificar como software malicioso que se incluya en aplicaciones auténticas.

de memoria de *kernel* usado por el sistema, lo que hace que efectivamente todos los controladores y el *kernel* del sistema operativo compartan un único espacio en memoria y se carezca de protección en la operación de estos.

La protección de la memoria para las páginas de las aplicaciones se vale del control de la traducción de las direcciones lógicas en direcciones físicas para validar que la dirección que se está traduciendo sea accesible para el tipo de operación solicitada para el proceso que la realiza.

En sus tablas de página, los procesadores también reconocen los atributos de permisos para permitir la lectura, escritura y ejecución de las páginas, y Windows también aprovecha esta funcionalidad.

La memoria compartida se maneja con listas de control de acceso (ACL, Access Control Lists), que registran los privilegios de cada proceso sobre los rangos de memoria compartida y permiten verificarlos de nuevo, al realizar las traducciones de direcciones lógicas.

Android

El sistema operativo Android se basa en un *kernel* de Linux, por lo que su administración básica de memoria es la misma; sin embargo, de cara a las aplicaciones, este sistema operativo hace consideraciones para la administración de la memoria que es importante destacar.

La primera es considerar que todas las aplicaciones son persistentes. Por ello, en vez de iniciar un proceso reservando su memoria y liberarla al terminarlo, se considera que las aplicaciones son utilizadas por el usuario una y otra vez, por lo que para ahorrar energía se mantienen cargadas en memoria el mayor tiempo posible.

En vez de usar el intercambio o usar memoria virtual para liberar memoria de las aplicaciones inactivas, Android da control explícito de las aplicaciones ante eventos de pérdida de foco y de reclamación de la memoria para que estas conserven su información de la manera más eficiente, considerando las necesidades de la aplicación. De igual manera, cuando las aplicaciones se reactivan, se generan eventos para que las aplicaciones implementen los mecanismos para recuperar el estado de almacenamiento secundario, que por lo regular es memoria Flash.

Android también hace uso de los niveles de protección de memoria presentes tanto en los procesadores Intel como ARM, con lo que provoca que las páginas de memoria del sistema operativo se usen en modo de *kernel* y toda la máquina virtual que ejecuta las aplicaciones

Importante

♠ **Arquitecto.** A pesar del uso de las facilidades de protección de memoria en Android, se han detectado pérdidas de información en el uso de servicios de sistema que emplean cachés que se comparten entre las aplicaciones, en un esfuerzo por acelerar, por ejemplo, el despliegue de información en

la pantalla. Esto puede ser importante cuando la información es confidencial. Sin duda, este es un antecedente muy importante, ya que muchos sistemas operativos manejan búfers internos de información que podrían ser vulnerables a ataques similares.

de usuario se mantengan con un mínimo nivel de privilegio. Tanto el uso de las rutinas, como la funcionalidad del sistema operativo de Android se hacen mediante llamadas al sistema que cambian el modo de ejecución, lo que ha demostrado dar un muy buen nivel de protección a las áreas de sistema. De manera adicional, cada aplicación se ejecuta como si fuera un usuario diferente para fines de protección de memoria, de modo que para acceder a los segmentos de aplicaciones distintas se requieren cambios de contexto que dan plena oportunidad de verificar los rangos de memoria.

EVALUACIÓN

4.1 ¿El modelo de computadora de Vonn Newman es inválido por el uso de diversos tipos de memoria en las computadoras actuales? Argumenta tu respuesta.

4.2 ¿Los datos contenidos en la memoria afectan el comportamiento de la computadora o solo son procesados por la funcionalidad previamente desarrollada? Argumenta tu respuesta.

4.3 En ciencia ficción se ha planteado el hecho de que cuando las computadoras puedan modificar su propia programación y diseñar nuevas computadoras de forma autónoma, llegarán rápidamente a tener capacidades casi ilimitadas. Desde la perspectiva de capacidad y administración de la memoria, ¿es posible que una retroalimentación de mejoras lleve a la memoria ideal (ilimitada)?

4.4 ¿Bajo qué condiciones es conveniente que el sistema operativo omita los mecanismos de protección de la administración de la memoria?

4.5 Las máquinas virtuales utilizan múltiples hilos de ejecución que simulan procesos. Si cada proceso, simulado o real, tiene su propio conjunto de trabajo, ¿qué efecto tiene eso sobre el conjunto de trabajo total de las máquinas virtuales?

4.6 ¿Qué comportamiento tienen las máquinas virtuales en el uso de memoria virtual, por ejemplo, en un sistema operativo como Linux, que busca mantener libre una parte de sus páginas?

4.7 ¿Los procesadores proporcionan más facilidades de protección de memoria que las que utilizan los sistemas operativos? ¿O son estos los que se ven obligados a complementar las facilidades que proporcionan los procesadores por ser insuficientes?

4.8 Explica la diferencia entre fragmentación interna y fragmentación externa.

4.9 Enumera las características básicas de la paginación y de la segmentación.

4.10 ¿Por qué la paginación se asoció originalmente con los procesadores CISC?

4.11 ¿Qué relación hay entre la TLB, cada procesador, cada proceso y el conjunto de trabajo?

4.12 ¿En qué ayudan las tablas de página multinivel a los sistemas donde la memoria disponible está dispersa en el mapa de memoria direccionable?

4.13 ¿Cuál es el inconveniente del algoritmo ideal de reemplazo de páginas que impiden que se emplee?

4.14 ¿Qué es un fallo de página?

4.15 ¿Qué síntomas presenta un sistema que cae en hiperpaginación?

Referencias bibliográficas

Denning, Peter J., *The Locality Principle, The profession of IT, Communications of the ACM*, Vol 48, Núm. 7, 2005.

Dhamdhere, D. M., *Sistemas operativos. Un enfoque basado en conceptos*, 2a ed., 2008.

O'Reilly, *Understanding the Linux Kernel*, 3a ed., 2009.

Russinovich, Mark E. y David A. Solomon, *Windows Internals*, 5a ed., 2009.

Stallings, William, *Operating Systems Internals and Design Principles*, 5a ed., 2008.

Tanenbaum, Andrew S., *Modern Operating Systems*, 3a ed., 2009.

Referencias electrónicas

Wikipedia, x86 memory segmentation: http://en.wikipedia.org/wiki/X86_memory_segmentation

Memory Management Reference: Compendio de información sobre la administración de la memoria que incluye glosario, lecciones básicas y bibliografía: http://www.memorymanagement.org/

COMPETENCIAS A DESARROLLAR

- El alumno podrá describir qué es un dispositivo de entrada y salida en una computadora, desde la perspectiva del sistema operativo.
- Diferenciará los diversos tipos de dispositivos desde el punto de vista de las arquitecturas usadas para transferir información hacia el procesador y la memoria.
- Distinguirá las capas del software de control de dispositivos de entrada y salida, así como los objetivos y funciones generales de cada una de ellas.
- Identificará los principales mecanismos de programación usados por las aplicaciones para interactuar con el software de control de dispositivos de entrada y salida.

Administración de los dispositivos de entrada y salida

5

¿QUÉ SABES...?

1. Mientras usas una computadora, ¿cuáles son las partes de esta que manipulas con más frecuencia o a las que prestas atención directa?
2. Cuando utilizas un equipo de cómputo diferente al habitual, ¿te resulta incómodo tener un teclado o mouse diferente o que no funcione correctamente?
3. ¿En qué situaciones es más conveniente utilizar un mouse, un `touchpad`, una pantalla sensible al tacto (`touchscreen`) o una tableta gráfica para interactuar con una aplicación?
4. ¿Cuáles son los principales dispositivos mediante los cuales "alimentamos" u obtenemos información de la computadora?
5. ¿Cómo se conectan estos dispositivos a la computadora?
6. ¿Por qué necesitamos instalar controladores para algunos dispositivos?

Importante

♥ **Docente.** La interfaz de puerto serie RS-232 fue publicada por primera vez en 1969 por el Comité de Estándares (en la actualidad Asociación de Industrias Electrónicas), la cual se basa en diseños originales donados por IBM. En esta se describen los conectores y la forma en que se transfiere información. Se considera uno de los primeros estándares de puertos adoptados por la mayoría de los fabricantes de dispositivos de hardware y de computadoras. No obstante, hoy día este importante estándar se está reemplazando por el más rápido y más complejo estándar de USB en sus diversas versiones.

5.1 Introducción

Una de las tareas fundamentales de las computadoras es recibir y entregar información, ya sea a sus usuarios, otras computadoras o actuar para afectar el medio que las rodea.

Para ello se requiere manipular elementos externos al procesador, como pueden ser las impresiones en papel, las señales luminosas enviadas por un monitor, las señales eléctricas generadas por un teclado, un mouse o un enlace de red. Los dispositivos especializados en convertir los datos manejados por el procesador en fenómenos en el exterior de la computadora, o de captar los diversos estímulos y convertirlos en datos interpretables por el procesador, se conocen como dispositivos de salida o entrada, respectivamente.

Para la plataforma de computadoras personales, particularmente las IBM PC compatibles, ha resultado más conveniente basar los dispositivos de entrada y salida en estándares que rigen la conexión con el equipo, de uso libre; esto favorece la existencia de gran número y variedad de fabricantes, así como la producción de múltiples dispositivos de distintas características y precios. Aunque otras plataformas suelen ser más selectivas en el control de los derechos de autor, para

limitar el número de fabricantes y mejorar la calidad, aun estas suelen tener una importante variedad de dispositivos, con diversos rangos de precios. Por lo descrito antes, como regla general debemos estar preparados para mantener la operación de las aplicaciones y de los sistemas ante una gran variedad de dispositivos, los cuales incluso pueden experimentar cambios a lo largo del tiempo. Esto hace que el principio de independencia física sea en especial relevante en el tema del control de los dispositivos de entrada y salida.

Los dispositivos basados en ambos estándares son un ejemplo de la gran variedad de dispositivos de entrada y salida disponibles.

- **Usuario.** En México, el desarrollo de software tradicionalmente se entiende como la implementación de aplicaciones en computadoras de propósito general; sin embargo también existe una considerable fuerza de trabajo que desarrolla software para sistemas de control y dispositivos de cómputo embebido de manera cotidiana. Precisamente es en este segundo mercado que la implementación de sistemas de control de dispositivos de entrada y salida tiene una mayor actividad en el país.
- **Líder.** En el caso de las PC de escritorio, solo el fabricante de algún dispositivo con características especiales es quien debe preocuparse por desarrollar un controlador para este y cumplir las normas que los sistemas operativos usados por sus clientes potenciales requiera, a fin de asegurar su base de usuarios.

5.2 Arquitecturas físicas

Con objeto de delimitar el reto para el software que controle los dispositivos de entrada y salida, lo primero que debemos hacer es revisar las características principales de los mecanismos de funcionamiento que deberán tener no solo los dispositivos en sí mismos, sino también las formas en las que la computadora podrá comunicarse con estos.

Familias de dispositivos

Es importante destacar que cada dispositivo debe tener una tarea clara y reconocible para sus usuarios. Incluso si cada modelo de dispositivo de entrada y salida tiene funciones y capacidades distintas a los de su competencia, es necesario que tanto el usuario como el software puedan reconocer y aprovechar todas estas características.

Esto ha generado una serie de nichos de mercado para los dispositivos, donde estos buscan satisfacer conjuntos muy bien delimitados de necesidades; esto les permite tener variedad, sin arriesgarse a perder la utilidad reconocible que ofrecen a los usuarios y a las aplicaciones.

Algunos nichos ampliamente reconocidos son los que se describen a continuación.

Recursos del sistema

Para operar una computadora es indispensable una serie de dispositivos, que a últimas fechas suelen integrarse en la tarjeta madre. Dichos dispositivos permiten realizar una serie de tareas especializadas que van desde medir el tiempo en unidades comunes

como horas, minutos y segundos, hasta ampliar las capacidades de la propia tarjeta madre, con conexiones de muy alta velocidad, mediante las ranuras de expansión (PCI, PCI express, etc.). Con frecuencia, todos estos dispositivos de tareas especializadas resultan transparentes a las aplicaciones; sin embargo, deben ser utilizados de manera adecuada por el sistema operativo para mantener al sistema funcionando en óptimas condiciones.

Controladores de puertos

Uno de los recursos del sistema que merece una mención especial son los controladores de puertos. Desde la aparición de las primeras computadoras comerciales, los dispositivos de entrada y salida se han comercializado de forma independiente del procesador, lo que permite diferentes alternativas en las adquisiciones y un flujo de capital más estable. Para que esto fuera posible se requirió que estos dispositivos pudieran ser conectados a la computadora después de su instalación original, sin necesidad de modificar la construcción de esta. De ahí surgió la necesidad de incluir conectores que se pudieran utilizar de forma opcional para enlazar los dispositivos a la computadora, a los que se conoce como "puertos", debido a su habilidad de "atracar" en las interfaces de los dispositivos. Tiempo después, IBM incursionó en el establecimiento de normas con objeto de que todos sus dispositivos pudieran conectarse mediante puertos construidos de la misma manera y con el mismo tipo de señalización. A la fecha, la empresa aún participa de manera activa en comités de estándares, con el fin de que todos los fabricantes que lo deseen produzcan dispositivos compatibles con sus equipos.

La circuitería que implementa el puerto de comunicación se conoce como controlador de puertos (*port controller*); sin embargo, este no debe confundirse con el software que conoce las características de algún dispositivo y que también suele llamarse controlador (*driver*). Para distinguir a ambos debe prestarse atención a que el contexto sea de hardware o de software.

Dispositivos de almacenamiento masivo

Mantener la información que se procesa se considera una de las tareas fundamentales de la computadora. Constituye la parte de conservar información durante largos periodos o en grandes cantidades en los sistemas de archivos o en memoria virtual. Es responsabilidad de una categoría especializada de dispositivos que pueden representar la información en algún medio físico que resulte de bajo costo y luego recuperarla. Aunque resulta deseable que la velocidad de transferencia sea elevada, el costo por MB suele ser el factor principal en este mercado.

Potencia

Como se dijo antes, las primeras computadoras requerían la construcción de instalaciones especiales para adaptar sus necesidades de alimentación y control de temperatura; sin embargo, conforme sus características se han mejorado, se ha logrado que a últimas fechas operen virtualmente en cualquier lugar donde los usuarios se encuentren, sin necesidad de una conexión a la red eléctrica durante periodos de operación cada vez más largos. Esto ha generado la necesidad de que el sistema operativo esté al tanto del tipo de alimentación de potencia que tiene; además, en caso de operar a baterías, debe detectar el estado y progreso de la carga de estas con el fin de limitar el uso de algunos recursos que disminuirían su capacidad de seguir operando.

Interfaces con el usuario

Dispositivos que al ser manipulados por el usuario permiten recibir instrucciones. Desde las propuestas del PARC (Palo Alto Research Center), originadas durante la década de 1970, se dejó de considerar al teclado de manera exclusiva, para empezar a contemplar al mouse (veáse figura 5.1) y a la pluma óptica para interactuar con elementos gráficos presentados en una pantalla. En la actualidad existe una amplia categoría de **dispositivos señaladores** en la que se incluye a los ratones (mouse), palancas de juego

Figura 5.1 Mouse original de Douglas Engelbart, creado en la década de 1970.

(joysticks), diversos controladores, pantallas sensibles al tacto, *touchpads* y muchos otros dispositivos, cada uno de los cuales tiene su propia variedad de implementaciones y características para adecuarse a las necesidades específicas de los usuarios, o simplemente para minimizar el costo.

Impresión

La función de las impresoras es generar respaldos de la información en papel, con el fin de que esta pueda ser consultada posteriormente o para dejar constancia de requisitos legales en diversos tipos de operaciones comerciales, financieras, etc. A pesar de la creciente confiablidad y el bajo costo de almacenamiento digital de la información en computadoras y dispositivos electrónicos, y de que el papel con frecuencia presenta mayores riesgos en su almacenamiento a largo plazo, este constituye el medio tradicional de almacenar documentos y tiene la ventaja de no requerir de una computadora para usarlo, por lo que la tendencia es continuar su uso. Esto ha impulsado la construcción de gran variedad de dispositivos para generar imágenes en papel en una enorme gama de capacidades y propósitos, desde el uso doméstico hasta aplicaciones industriales, que rivalizan con las imprentas tradicionales.

Red

Se refiere a los dispositivos que permiten que una computadora envíe y reciba información hacia o por parte de otras computadoras. Originalmente se usaban los puertos de conexión de dispositivos, con cables construidos a modo para permitir que la información enviada por una computadora llegase a otra cercana; sin embargo, hoy día ha progresado en sus capacidades de velocidad de transferencia de información, en su alcance geográfico y en las aplicaciones que soporta, por lo que constituye una parte indispensable del soporte de buena parte de las operaciones comerciales en las que se utilizan sistemas informáticos.

Video

La transmisión de imágenes a los usuarios en tiempo real ha permitido que la relación y el control de la computadora hayan pasado de los operadores del equipo a los usuarios finales, en especial en sistemas de cómputo personal y sistemas de tiempo compartido. Además de las recomendaciones de interfaz de usuario del PARC, el uso de ventanas, botones, etc., han permeado la concepción misma del público acerca de la forma en que se debe usar una computadora. La posibilidad de usar imágenes de alta

resolución, aunado a toda la gama de colores que el ojo humano es capaz de distinguir, han abierto la puerta a un rango importante de aplicaciones de video, donde la estimulación de los sentidos del usuario es el componente fundamental, entre las que destacan los sistemas multimedia, los juegos de video, el cine en casa, los videos bajo demanda, las videoconferencias y la realidad virtual, por citar solo algunas.

Sonido

En la actualidad no es posible referirse a la evolución del video sin una relación directa con la evolución del sonido y los efectos sonoros por computadora. Así, podemos afirmar que la emisión intencional de sonidos por computadora también ha experimentado un progreso similar al del video. Aunque la forma en que percibimos el sonido es mucho más subjetiva que como lo hacemos con la información visual, lo que resta importancia a la fidelidad, también es de uso común y constituye un componente indispensable de las aplicaciones basadas en la estimulación de los sentidos del usuario, como aquellas relacionadas con las telecomunicaciones y el entretenimiento.

Dada la evolución de estos elementos tan importantes en la transmisión de información auditiva y audiovisual, es razonable esperar que este conjunto de funciones forme parte de un solo dispositivo, independientemente de las características e incluso de la forma en que opere este. Más aún, las aplicaciones que emplee esta funcionalidad se verán beneficiadas en la medida en que no requieran conocer las características de los dispositivos y puedan usar tan solo las funciones generales.

Todas estas constituyen solo las familias de dispositivos comunes para computadoras personales, pues los dispositivos de propósito general y de cómputo embebido suelen contar con familias propias o dispositivos especiales que requieren ser controlados de manera directa.

5.3 Comunicación interna

Todos los diversos tipos de dispositivos que hoy día están disponibles en el mercado tienen necesidades muy variadas en lo que respecta a los medios que usan para comunicarse con el procesador e intercambiar datos con la memoria de la computadora, pues la velocidad de transferencia, el volumen de datos a transferir e incluso la sincronicidad que requieren, varían por muchos órdenes de magnitud de un tipo de dispositivo a otro. Algunos, como las tarjetas de video, con frecuencia son instalados de origen, desde que la CPU y la tarjeta madre son adquiridos por el usuario final, por lo que todos estos se acompañan a lo largo del ciclo de vida del equipo o simplemente están integrados en la

propia tarjeta madre. En cambio otros, como las memorias flash USB, se conectan y desconectan, incluso varias veces, durante la operación de una sola aplicación en el equipo.

Para acomodar esta variedad de requerimientos, el equipo implementa diversos mecanismos de conexión y comunicación que revisaremos a continuación.

Integrado al procesador

Muchos microcontroladores y algunos procesadores incluyen en su arquitectura y construcción registros especiales para recibir información de dispositivos de entrada y salida, los cuales se integran de origen a estos. Para acceder a dichos registros, por lo general se requieren instrucciones normales de lectura y copia de valores a la memoria, ya sea con respecto a un espacio de direcciones exclusivo para estos dispositivos o a diferentes secciones especiales de las direcciones reconocidas por la CPU.

Por ejemplo, el microcontrolador 68HC11 de Motorola incluye un convertidor analógico digital de 8 canales y usa los registros ADR1 a ADR4 para recopilar los resultados de la conversión. Dado que los dispositivos de este tipo deben considerarse desde que se diseña el procesador, existe muy poca variedad y se carece de flexibilidad en estos.

Mapeo a memoria

En arquitecturas sencillas, el mecanismo más utilizado para conectar los dispositivos con la CPU se basa en aprovechar los buses de memoria y los mecanismos que permiten que diversos bancos de memoria sean utilizados según la dirección que se activa.

Así, para acceder a voluntad a un conjunto de circuitos de memoria, que constituyan una palabra de 32 o de 64 bits según la arquitectura, y no a todos los demás circuitos presentes, se utilizan circuitos de memoria que tienen tres estados en sus conectores de salida:

- Encendido
- Apagado
- Alta impedancia

Este último estado permite que, aunque todos los circuitos de memoria estén conectados físicamente al Memory Buffer Register (Registro del Búfer de Memoria, MBR), responsable de transmitir las señales entre la memoria y la CPU, solo los circuitos que reciben una señal mediante su conector de **habilitar** (Enable), en efecto actuarán como si estuviesen conectados, pues todos los demás colocan sus transistores en corte y actúan como si se hubiesen desconectado del bus.

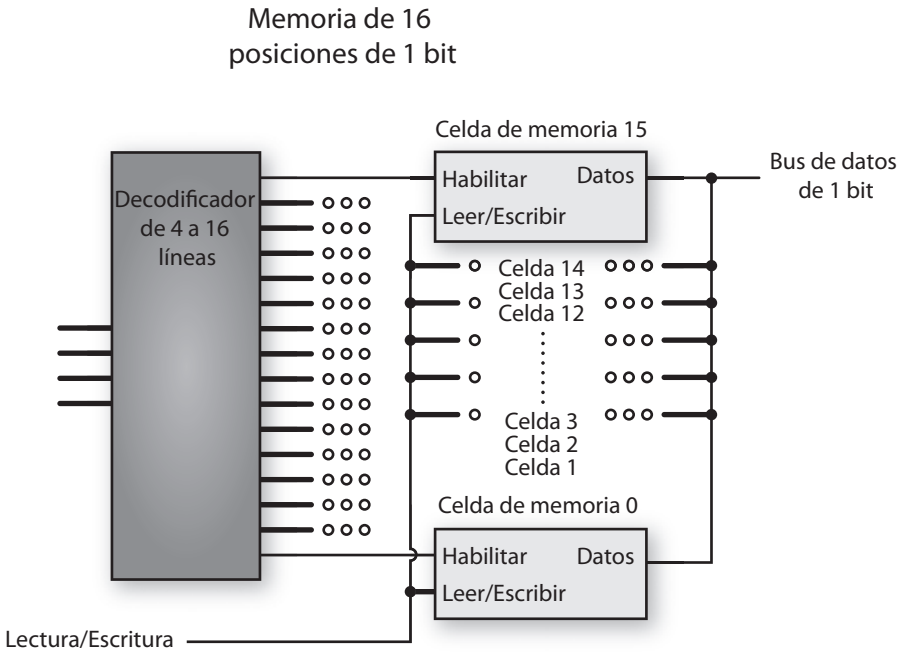


Figura 5.2 Selección de la celda de memoria mediante un decodificador y líneas para habilitar.

Para determinar cuáles circuitos de memoria deben recibir las señales de habilitar se emplea un conjunto de decodificadores que reciben la dirección, a partir de la cual generan las correspondientes señales de Enable o habilitar.

Por su parte, para mapear los dispositivos a memoria se elige un rango de valores de dirección que no vaya a ser utilizado por la memoria RAM en la arquitectura. Con frecuencia se emplean valores muy bajos o muy altos y se colocan decodificadores que, ante direcciones específicas en ese rango, activen los circuitos de los búferes de valores del dispositivo que queremos utilizar en vez de los circuitos de memoria.

Desde la perspectiva de la CPU, para recuperar valores del dispositivo se usa una posición de memoria como cualquier otra y se recupera el valor sin necesidad de tomar consideraciones adicionales. Sin embargo, a menudo estos búferes de dispositivo son más lentos que el resto de la memoria RAM, por lo que esta operación puede tomar un tiempo aún más largo que un acceso a memoria convencional; asimismo, deben tomarse precauciones para que la memoria caché de la CPU no intente recuperar los valores de estas posiciones, ya que no tendría medios para saber cuándo requiere actualizarlo y terminaría por distorsionar la comunicación con el dispositivo.

Una vez que la CPU puede intercambiar valores con el dispositivo mediante el mapeo a memoria, debe generarse un mecanismo por el que se sincronice la operación, con objeto de evitar que la CPU tome el valor antes de que el dispositivo y su búfer

terminen de reflejar de manera correcta el resultado de la operación realizada. Para esto, se puede realizar una verificación reiterada de algunos bits de control y programas que reconozcan su uso, o bien utilizar el mecanismo de interrupciones.

A continuación se revisamos estas alternativas con más detalle.

Controlado por software (Programmed I/O)

En estos casos, el procesador entra en un ciclo en el cual debe verificar muchas veces el estado de la palabra o el estado del dispositivo, esperando que el bit correspondiente indique que la operación está lista, es decir que cambie de valor, y una vez que se detecte este cambio pueda proceder a recuperar los valores obtenidos; a esto se le conoce como espera activa (Busy Waiting). Pero si lo que se desea es transmitir un valor al dispositivo, primero debe realizarse la escritura y enseguida llevar a cabo el ciclo para validar que la operación se concluya con éxito.

En el caso del convertidor analógico digital del 68Hc11, este utiliza el bit 7 del registro de control ADCTL, que está asociado a dicho dispositivo. Sin embargo, este mecanismo no es exclusivo de dispositivos integrados al procesador, por lo que también puede usarse por dispositivos mapeados a memoria. Una rutina en ensamblador para realizar una conversión sería como la siguiente:

```
*-------------------------------------------------------------------
* subroutine:   AD_READ
* descripcion:  Esta subrutina lee el canal del
*               convertidor A/D en el registro A y retorna el valor en A
*-------------------------------------------------------------------
AD_READ
      STAA    ADCTL           ; Inicia la conversion
ADL00   LDAA    ADCTL           ; Obten el status
      BPL     ADL00           ; Espera a que termine, bit 7 encendido
      LDAA    ADR4            ; Retorna resultado en A
      RTS                     ; Retorno de la subrutina
```

Usando interrupciones

Si se desea evitar la espera activa, especialmente porque el tiempo de espera por las operaciones será considerable y, en consecuencia, porque mantener el procesador ocu-

pado en espera implicaría un impacto serio al desempeño del sistema, es posible utilizar las líneas de interrupción de la propia CPU para recibir la notificación en el momento en que la operación termine. Cuando la CPU recibe dicha señal, de inmediato este suspende la ejecución del proceso o rutina que atiende en ese momento y procede a ejecutar una rutina que bien recupera los valores de las posiciones de memoria correspondientes o bien genera las notificaciones apropiadas al detectar que la operación a la que se enviaron valores ha terminado.

La mecánica detallada de atención de interrupciones se revisa más adelante, cuando se estudian las rutinas de atención de interrupciones como una de las capas del software de control de dispositivos de E/S.

Direct Memory Access (DMA)

Un aspecto importante del mapeo a memoria en general consiste en tomar en cuenta que la transferencia de la información del búfer del dispositivo a la memoria RAM debe realizarse con la participación de la CPU, primero cargando cada una de las palabras del búfer del dispositivo a un registro y luego escribiendo estos valores a la memoria RAM, en especial para dispositivos en los que se transfieren grandes volúmenes de información; por ejemplo, al leer un archivo de varios GB desde una memoria flash que se encuentra conectada mediante un puerto USB se generará una carga muy importante de trabajo para el procesador. Además, los controladores de dispositivo suelen ejecutarse con privilegios de superusuario (root), por lo que dicha carga de trabajo tendrá prioridad sobre los procesos de usuario, incrementando el impacto en el desempeño del equipo.

Por lo anterior, para evitar esta carga a la CPU se usa el Acceso Directo a Memoria (DMA, siglas de Direct Memory Access). El controlador del dispositivo de entrada y salida con DMA no solo es un dispositivo mapeado a memoria sino que tiene la capacidad de procesamiento y de tomar el control de los buses de memoria de forma similar al procesador, con el objetivo expreso de copiar los valores del búfer de los dispositivos que controla a la memoria RAM.

No obstante, para el logro de esto se requiere un mecanismo adicional para determinar qué dispositivo estará controlando la memoria, ya sea la MMU (Memory Management Unit) de la CPU o el controlador DMA, para lo cual se emplean líneas de control adicionales en el procesador, conocidas como líneas DMA.

Al llevar a cabo una operación de entrada o salida mediante un controlador DMA, lo primero que hace la CPU es utilizar el controlador como un dispositivo mapeado a memoria para escribir en sus registros los detalles y las instrucciones que describen la operación que se va a realizar.

Acto seguido, el controlador determina el orden en el que debe transferir la información y realizar la operación con los dispositivos; sin embargo, por regla general debe pasar algún tiempo, a fin de que se complete la operación en el dispositivo, durante el cual no se afecta la operación de la CPU, además de que se requiere un tiempo para tomar el control de la memoria RAM y comenzar a realizar la transferencia de la información, dejándole a la CPU exclusivamente la capacidad de proseguir su operación sobre los valores de las posiciones de memoria contenidas en su memoria caché y en los registros.

Una vez concluida la operación, el controlador DMA genera una interrupción para notificar a la CPU que terminó la operación y se pueda empezar a proceder con los procesos que se habían bloqueado en espera de esta y solicitar las operaciones siguientes, en caso de haberlas.

Como se puede observar, los diversos mecanismos de comunicación interna imponen necesidades al software para que utilice los dispositivos, con el objetivo de que se realicen las operaciones correspondientes y se reconozcan los tiempos y los mecanismos de coordinación, no solo los de los del dispositivo que habrán de usarse sino de todos los mecanismos que la arquitectura del equipo utilice para comunicar al procesador con este.

5.4 Mecanismos y funciones de los manejadores de dispositivos

Tanto el panorama de los dispositivos a controlar como el de las arquitecturas que los controlan son en extremo complicados debido a la variedad de requerimientos y mecanismos que debemos adecuar a las necesidades de las aplicaciones, las cuales son las encargadas de capturar las peticiones y necesidades de los usuarios y de generar las operaciones para los dispositivos.

Ante esta situación, el desarrollo de aplicaciones se enfrenta a costos muy elevados y condiciones muy difíciles de controlar, en especial cuando en cada aplicación se intenta incluir la funcionalidad necesaria para aprovechar esta, incluso aun cuando una parte de los dispositivos de entrada y salida pudiera ser más importante para el propósito de la aplicación. Sin embargo, una vez que un sistema de cómputo tiene un dispositivo de entrada y salida, a menudo puede atender a muchas de las aplicaciones y no solo a una de manera exclusiva.

Por ello es muy conveniente que la funcionalidad que permite aprovechar las características de un dispositivo en particular sea implementada como parte del sistema

operativo y que este, en su rol de extender la funcionalidad de la computadora, presente una interfaz de programación y servicios que todas las aplicaciones del sistema puedan utilizar para aprovechar los dispositivos. Con lo anterior se reduce la redundancia de trabajo en que se cae si cada aplicación trata de implementar este control de los dispositivos de forma independiente, además de que también se pueden aprovechar los servicios del sistema operativo para desarrollar la funcionalidad requerida partiendo de una plataforma que simplifique este esfuerzo.

Objetivos del software de E/S

Las necesidades del software responsable de controlar los dispositivos, integrado en el sistema operativo, se pueden resumir con base en los siguientes objetivos generales:

- **Independencia lógica.** La funcionalidad que controla el dispositivo y las aplicaciones puede modificarse de manera independiente a los propios cambios de las aplicaciones, sin comprometer la capacidad de operar del conjunto.
- **Independencia de dispositivo.** Con base en sus características específicas, el dispositivo que se emplee debe poder cambiar, sin comprometer la operación de las aplicaciones que lo usan.
- **API y nomenclatura uniforme.** Las interfaces de programación que empleen las aplicaciones para aprovechar las prestaciones de los dispositivos, así como la nomenclatura usada para identificarlos, deben ser uniformes en todo el rango de dispositivos del sistema de información. Esto reduce la curva de aprendizaje y la complejidad de las aplicaciones.
- **Rendimiento.** Las operaciones con dispositivos de entrada y salida generan carga para la CPU, la memoria e incluso para el consumo de potencia del equipo. De igual modo, tanto la estructura lógica del sistema operativo como la de las aplicaciones también generan carga adicional. Por tanto, es indispensable ser muy cuidadosos en la implementación que se realice, a fin de preservar el nivel de rendimiento y minimizar el impacto a las aplicaciones. Esto incluye el almacenamiento temporal de la información en la memoria (buffering) y la exposición de las interfaces de programación que simplifiquen el desarrollo de aplicaciones.
- **Balance.** Siempre es indispensable administrar la carga de trabajo de los dispositivos de entrada y salida, de modo que todos los dispositivos y procesos que tengan tareas pendientes se mantengan ocupados atendiéndolas. Con esto se mejora el rendimiento del sistema de información y el aprovechamiento de los dispositivos.

- **Manejo de errores.** Entendidos los errores como cualquier desviación de la operación normal, es muy común que en los dispositivos de entrada y salida ocurra de manera rutinaria una gran variedad de situaciones que impidan la operación. Muchos de estos errores suelen ser temporales o pueden compensarse con facilidad; por ello, la funcionalidad asociada al control de cada dispositivo debe estar al tanto de estas situaciones e implementar los procedimientos de recuperación apropiados, sin necesidad de que las aplicaciones intervengan. Pero en caso de que los errores resulten más serios, entonces se necesitarán otros mecanismos para notificar a las aplicaciones acerca de los errores que han ocurrido y que no se han podido recuperar mediante mecanismos que sean uniformes para todos los errores y dispositivos.
- **Protección.** Muchos dispositivos carecen de la capacidad de atender más de una tarea a la vez (`non-preemptive`), por lo que se requieren mecanismos que permitan proteger a lo largo del tiempo el uso de los dispositivos de forma exclusiva para un conjunto delimitado de procesos o solo uno de estos. También es indispensable considerar que una buena parte del software del control de dispositivos de entrada y salida necesitará usar rangos de memoria accesibles solo en modo *kernel*, por lo que se requieren mecanismos de protección de las interfaces de programación asociadas para que las aplicaciones no puedan abusar de estas y sea posible acceder a otras partes del *kernel* del sistema operativo.
- **Sincronización.** Como es sabido, las aplicaciones dependen del sistema operativo para controlar los tiempos de espera durante la realización de las operaciones de entrada y salida, lo que evita que requieran conocer o implementar los mecanismos para verificar el estado de los dispositivos o que respondan de manera directa a las interrupciones.

Importante

♥ **Arquitecto.** En la mayor parte de la bibliografía, al consultar los detalles de cada sistema operativo se puede observar que, a menudo, el énfasis se poneen el ambiente para el desarrollo de los controladores de dispositivo. Esto obedece a la necesidad de los múltiples fabricantes de dispositivos de E/S

Organización de las capas del software de E/S

Para contender con la variedad de responsabilidades que el software de control de dispositivos de entrada y salida debe atacar, e incluso con la variedad de fuentes de donde debe emerger, su funcionalidad se divide en múltiples capas, con el fin de que cada una de estas pueda atacar partes independientes de la complejidad y proporcionela flexibilidad requerida al implementarse con independencia lógica.

Cada sistema operativo tiene su propia arquitectura y forma particular de implementar las soluciones y de dividirla en capas. Sin embargo, se puede observar una división general común que se man-

tiene al menos en los sistemas de tiempo compartido y que tiene cuatro divisiones principales, las cuales se describen a continuación.

- **Interfaz con aplicaciones.** El sistema operativo soporta el desarrollo de las aplicaciones por la presencia de una interfaz de programación de aplicaciones (API), a través de la cual estas puedan acceder a la funcionalidad que los dispositivos presentan. Dichas funciones se ejecutan con los privilegios de los usuarios que emplean estas aplicaciones, por lo que con frecuencia se les conoce como **software de área de usuario**, por el contexto y privilegios en que opera.
- **Administración de entradas y salidas.** Para poder aprovechar al máximo la funcionalidad de los dispositivos se requiere reconocer cierto nivel de detalle de las ventajas y prestaciones que ofrecen estos; sin embargo, por lo general se manejan abstracciones que impiden que las aplicaciones generen dependencia con respecto al dispositivo particular que se encuentra presente. Por tanto, en esta capa se maneja la funcionalidad común a cada familia de dispositivos y se procura la independencia del dispositivo. Por este manejo de categorías genéricas, a esta capa se le conoce a menudo como **controladores independientes de dispositivo**.
- **Controladores de dispositivos.** En esta capa se implementa la funcionalidad que permite aprovechar las características específicas de cada dispositivo y manejar de manera adecuada la codificación propia que emplea la información de operación, de los errores que pudieran presentarse y de los procedimientos de recuperación de estos.
- **Interfaz con el hardware.** Un mismo dispositivo, con la misma aplicación, puede emplearse en una gran variedad de arquitecturas y configuraciones de equipos, por lo que conviene proporcionar cierto nivel de abstracción para el desarrollo de los controladores, que les otorgue suficiente independencia física de todos los factores que lo rodean, a fin de reducir el número de versiones a desarrollar para situaciones particulares. Adicionalmente, el sistema operativo debe manejar los mecanismos comunes de interacción, como las interrupciones genéricas, para que todos los controladores puedan aprovecharlos.

de generar ellos mismos los controladores para sus dispositivos. Por otra parte, también se pone mucha atención a las facilidades prestadas a las aplicaciones para aprovechar los sistemas operativos; no obstante, por quedar oculto a ambas, las características de la administración y las de los mecanismos para la interacción con el hardware suelen revisarse solo de forma general.

♥ **Líder.** Es recomendable que las aplicaciones y los controladores se restrinjan a usar los mecanismos recomendados de la interfaz para las aplicaciones (API) en vez de utilizar las características específicas de una implementación en particular, ya que estas últimas pueden variar entre versiones del sistema operativo y tendrían un efecto negativo en el mantenimiento de controladores y aplicaciones.

♣ **Docente.** Ya enunciamos y revisamos las capas en función de su proximidad a los usuarios finales. Por tanto, incluso si en este caso la interacción siempre tuviera que realizarse mediante las aplicaciones, una buena manera de presentar el contenido del tema es hilvanar la serie de operaciones que se llevan a cabo a través de las capas desde la perspectiva de ejemplos de operaciones que hacemos como usuarios.

Figura 5.3 Capas del software de central de dispositivos de E/S.

5.4 Interfaz con aplicaciones

El objetivo fundamental de esta capa es el soporte para el desarrollo de aplicaciones, por lo que se presta especial atención a los siguientes aspectos:

- Presentar una interfaz de programación de aplicaciones (API) uniforme para el uso de todos los dispositivos de E/S, que incluya el uso de una notación uniforme para todos los dispositivos.
- Proporcionar mecanismos estándar para recuperar información sobre los errores que se presenten en los dispositivos de E/S, o de hecho en cualquier parte del sistema.
- Manejar el almacenamiento temporal de la información recibida al enviar la a los dispositivos.

Interfaz de programación

La interfaz de programación de aplicaciones (API) que habrá de exponerse siempre deberá implementar una serie de funciones que las aplicaciones puedan utilizar. Para ello estas funciones suelen ser ligadas en tiempo de compilación a las propias aplicaciones y operan bajo el contexto de la aplicación de usuario. Para utilizar los dispositivos de E/S, con frecuencia se requiere efectuar operaciones en el contexto del *kernel* del sistema operativo, por lo que estas bibliotecas tendrán la implementación para realizar las llamadas al sistema necesarias para acceder al modo de *kernel*, así como para llevar a cabo las operaciones específicas con los dispositivos de entrada y salida requeridas. Es importante que este mecanismo de llamada sea controlado con el fin de que las aplicaciones no se encuentren en libertad de implementar cualquier actividad en modo de *kernel*, lo que pondría en riesgo la estabilidad del sistema y el respeto a los privilegios de usuario.

Control de errores

El mecanismo para el control de los errores de las operaciones que se realicen en el sistema operativo debe contener dispositivos estándar para que las aplicaciones puedan obtener la información que les permita notificar a los usuarios que ha ocurrido un error y la naturaleza del mismo, en especial cuando es necesario que los mismos usuarios participen en la solución. Sin embargo, es muy probable que los errores que puedan ser atendidos exitosamente por el controlador del dispositivo no requieran ser atendidos o requerir alguna actividad por parte del usuario y, por tanto, no reportarse (error mudo) o reportarse solo con fines informativos, sin que las aplicaciones estén obligadas a atenderlos.

Por lo general, existen tres mecanismos para manejar la información y el flujo de control de los errores, como se expone a continuación.

Interrupciones del flujo

Se trata de un mecanismo que interrumpe la atención de procesos para reaccionar ante eventos inusuales. En la interfaz de programación de aplicaciones (API) debe exponerse la funcionalidad necesaria para que las aplicaciones envíen señales o generen excepciones al detectar situaciones anómalas y definan las tareas que la aplicación llevará a cabo cuando se presenten excepciones y señales.

Las señales son enviadas desde aplicaciones o servicios de sistema y permiten que el código, al detectar una situación anómala, interrumpa su flujo habitual de ejecución y proceda a la rutina de atención asociada a la señal.

El mecanismo de las excepciones es similar al de las interrupciones de flujo respecto a que también interrumpe el flujo de la ejecución del programa; sin embargo, a diferencia de las señales, este último mecanismo no procede a una rutina de atención específica, sino que se propaga de acuerdo con el contexto de ejecución del programa en que se encuentra en ese momento. Estas interrupciones son más adecuadas para que diversos módulos reaccionen ante sus propias situaciones de excepción o para permitir que el manejo por defecto del sistema operativo reciba la excepción cuando estos no implementan mecanismos para atrapar y lidiar con dichas situaciones.

En la actualidad, las excepciones y señales son comunes a casi todos los sistemas operativos, incluyendo la funcionalidad que el sistema operativo empleará para manejar las excepciones que las aplicaciones no atrapen, lo que comprende tanto a los sistemas de tiempo compartido como los de tiempo real y embebidos.

Salida estándar de errores

Una de las propuestas más exitosas del sistema operativo UNIX la constituye el entubado de los flujos de información de los programas. Uno de los tres flujos de información

que todo proceso tiene es el de mensajes de error, conocido como **salida estándar de errores**. En este las aplicaciones pueden colocar todos los mensajes y códigos de error asociados a situaciones anómalas durante su ejecución, en especial si estos no impiden que el proceso continúe con su operación. Este flujo de información puede ser retomado por otras aplicaciones para, a su vez, ser procesado por el sistema operativo, usualmente para presentarse en la consola del usuario.

Códigos de retorno

Otra de las propuestas exitosas de UNIX se relaciona con el hecho de que todas las aplicaciones pueden terminar su ejecución entregando un código numérico al proceso que las inició, el cual represente el estado final con el que terminó la aplicación. Se considera que UNIX es el primer sistema operativo en implementar un catálogo de códigos estándar de terminación que define una serie de estados erróneos de terminación, donde se usa el código 0 para la terminación sin errores. A partir de este catálogo, la mayoría de las llamadas al sistema respetan estos códigos de terminación para reportar el resultado de su invocación a las aplicaciones que las emplean. En Windows también se emplean los códigos de terminación; sin embargo, no se tiene un catálogo de errores estandarizados.

Bitácoras

Tradicionalmente, las bitácoras han estado a cargo de la aplicación. Se consideran una extensión del concepto de la salida estándar de errores que utiliza un archivo persistente y que permite la recolección y la posterior revisión de los mensajes que describen la operación y los fallos que se hayan presentado durante la operación. En el caso de los sistemas operativos móviles, surge un requerimiento adicional: el desarrollador desea tener acceso a la información de diagnóstico de una falla sin tener acceso al equipo en el que se está ejecutando la aplicación, para lo cual se utilizan sistemas de bitácoras orientadas a sistemas distribuidos que permiten recopilar esta información sin poner en peligro la confidencialidad de los datos del usuario final.

5.5 Administración de entradas y salidas

La capa de administración de entradas y salidas tiene la responsabilidad de atender las peticiones enviadas por las aplicaciones mediante la capa de interfaz con aplicaciones, por lo que debe implementar varias interfaces de programación y funcionalidad propia.

Exponer funcionalidad común de la familia de dispositivos

Las principales interfaces de programación que debe implementar y exponer esta capa son aquellas que permitan a las aplicaciones aprovechar las funciones específicas de una familia de dispositivos, más allá de las funciones de lectura y escritura por bloques que pueden aplicarse a cualquier dispositivo. Estas API permiten aprovechar funciones específicas que requieren las aplicaciones, como solicitar la impresión de un trabajo y recabar eventos de la interfaz de usuario, entre muchas otras. Por la funcionalidad genérica que exponen autores como Tanenbaum se refieren a estas API como controladores independientes de dispositivo.

Hoy día, la mayoría de los sistemas de archivos modernos ha retomado el concepto originalmente usado en UNIX de representar la API básica de los dispositivos de E/S, manejando dichos dispositivos como si fueran archivos. Debido a esta condición cobra una importancia relevante, pues mediante esta se suelen resolver los aspectos básicos de privilegios, identificación de los recursos, inicialización, entrega y recuperación de la información.

Más allá de esto, también se proporcionan las API específicas para intercambios de información, por ejemplo, que aporten control sobre el almacenamiento temporal de la información, comunicación síncrona o detalles de la comunicación asíncrona, así como comunicación directa con el controlador específico de dispositivo, pero con el uso de una interfaz controlada. Debido a que operan en modo de *kernel*, también se manejan las interfaces necesarias para que un controlador de dispositivo pueda invocar a otros para atender operaciones que requieran una atención por múltiples capas de controladores.

De manera adicional, algunas familias de dispositivos tienen una API de programación específica para su operación, ajena a la uniformidad de los demás dispositivos. Específicamente, la capa de red y el control de los dispositivos de video reciben este tratamiento debido a que en ambos casos la funcionalidad que las aplicaciones necesitan explotar de estos dispositivos tiene complejos niveles de abstracción, que no se desea implementar en cada aplicación. En el caso de la red, los flujos (`Streams`) y la implementación de `Sockets` constituyen la representación habitual del soporte a los protocolos de Red Ethernet, proporcionada por el sistema operativo para acceder a redes como Internet. Por otra parte, el video cuenta con protocolos especializados, como X11 y DirectX.

Implementar la interfaz hacia los controladores de dispositivos

La administración de entradas y salidas también define las normas que cada sistema operativo impondrá a los controladores de dispositivos que operarán en el sistema, los

cuales deberán apegarse a estas para ser cargados, iniciados e invocados con las operaciones que habrán de atenderse y entregar resultados.

Los controladores de video y de red también suelen recibir un tratamiento especial en este aspecto, lo que se debe primordialmente al nivel de optimización en una serie de operaciones que resultan comunes a este tipo de dispositivos.

La administración de dispositivos, en su interacción con los controladores, también debe atender la configuración y vigilar el desempeño de los dispositivos, aunque, de igual modo, los parámetros varían en cada familia de dispositivos.

Almacenamiento temporal de resultados y adecuación del tamaño del bloque

El mecanismo genérico para manejar los dispositivos de entrada y salida contempla que se intercambien paquetes de información y que los procesos se bloquen en espera de que terminen las operaciones solicitadas. Los dispositivos pueden manejar la información palabra por palabra, o bien en bloques de datos de diversas longitudes. Asimismo, como ya se señaló, también existen diversos mecanismos para manejar la sincronización. Esta abstracción proporciona un mecanismo uniforme que puede simplificar el desarrollo de las aplicaciones, permitiéndoles ignorar los mecanismos particulares del dispositivo en virtud de esta mecánica uniforme.

Pero para que esto sea posible debe proporcionarse el mecanismo de almacenamiento temporal de la información enviada por las aplicaciones, el cual, independientemente de la cantidad de información y del ritmo al que la aplicación lo genere, deberá almacenar la información en memoria, generar una cola de peticiones que se enviarán al dispositivo, optimizar el orden en que las operaciones se realizarán y, por último, solicitar las operaciones con los paquetes de datos adecuados a la operación del dispositivo y en el orden optimizado.

Dispositivos de tipo no reentrantes (`non-preemptive`) pueden usar el software en la interfaz con las aplicaciones, primero para almacenar las peticiones que se reciben de manera concurrente en estructuras de memoria cuyo uso sea reentrante (`preemptive`) y luego para atender las peticiones de manera secuencial, a través del dispositivo no reentrante.

Las áreas de memoria en las que se almacena la información en lo que se generan las operaciones en el dispositivo para lograr un mejor desempeño y adecuar el tamaño del paquete se conocen como **búfer**, y las listas de peticiones o trabajos completos que están en espera de ser ordenadas para resolver problemas de concurrencia se conocen como **spool**.

Protección y asignación de recursos

Otra responsabilidad de esta capa consiste en manejar el control de acceso a los dispositivos, algunos de los cuales pueden requerir privilegios especiales en el usuario para recibir peticiones, o por ser no penetrantes requerir que, una vez que se asigne el dispositivo a un proceso, no deban admitirse las solicitudes de atención de otros procesos.

Según el modelo de UNIX, la representación de los dispositivos como archivos proporciona un esquema de permisos que puede aprovecharse para solventar la mayoría de estos requerimientos.

Carga dinámica de los controladores

Por último, la administración de entradas y salidas es la responsable de identificar, con base en los identificadores usados por la interfaz de aplicaciones, los dispositivos, el tipo de dispositivo y el controlador específico que debe atender las peticiones.

De este modo, puede cargar los controladores específicos cuando se recibe la petición o la respuesta a eventos de Plug And Play, donde el puerto o el BIOS detecta que se conecta un nuevo dispositivo al sistema y recupera la identificación del dispositivo que su controlador requiere que sea identificado, cargado a memoria y mediante el cual se realicen los procesos de inicio de la conexión propios del dispositivo.

Administración de potencia

En sistemas embebidos, principalmente móviles (como smartphones y otros dispositivos que operan con batería), y sistemas de tiempo compartido, que también suelen operar con baterías, es necesario que la administración de dispositivos de entrada y salida esté al tanto de la situación de alimentación de potencia del sistema, con el fin de que administre la funcionalidad de los dispositivos de entrar en modos de bajo consumo de potencia y de alto rendimiento, en función de la disponibilidad, por ejemplo, de la conexión a la línea de energía eléctrica o al nivel de carga de la batería.

5.6 Controladores de dispositivos

La tarea fundamental del controlador de un dispositivo es aprovechar al máximo la funcionalidad que ese dispositivo ofrece al sistema, para lo cual es necesario que sepa

Importante

♥ **Usuario.** Siempre debemos considerar que como muchos controladores de dispositivos operan en modo de *kernel*, con un conjunto ampliado de privilegios, son blanco perfecto para los fabricantes de software malicioso, que pudiera intentar convencer a los usuarios de instalarlo en sus equipos como caballos de Troya o inyectándolos en controladores de fabricantes de dispositivos reales, con los cuales tengan comprometido su proceso de generación o distribución de software.

♠ **Arquitecto.** Como desarrolladores de un controlador de dispositivo, es importante comprender a cabalidad las normas impuestas por el sistema operativo al que orientaremos el mercado de nuestro dispositivo. Incluso si detectamos atajos que nos permitan simplificar nuestro desarrollo omitiendo diversos aspectos de las interfaces de programación normales, debemos evitarlos, ya que por lo regular las funciones no documentadas o recomendadas son aquellas partes de la implementación interna de las capas vecinas que el fabricante del sistema operativo puede modificar en versiones posteriores y que no documentó públicamente, debido a que precisamente no es necesario que sean soportadas en versiones

interpretar los códigos de operación que el dispositivo entregue, traducir las peticiones que se reciban a los códigos de operación propios del dispositivo e interpretar los códigos de error o las situaciones anómalas de operación, para generar la información de reporte de errores.

Dado el íntimo conocimiento que requiere esta capa de controladores de dispositivos acerca de las características de los dispositivos, es conveniente que no sean los desarrolladores del sistema operativo, sino los desarrolladores de los dispositivos, quienes creen los controladores. Sin embargo, esto plantea algunos conflictos de interés, ya que los fabricantes, por lo regular, no son compañías dedicadas al desarrollo de software y, por tanto, su interés fundamental es la buena operación de su dispositivo y no de otros dispositivos que pudieran estar presentes en la enorme variedad de configuraciones posibles. Esto hace que den prioridad a la simplicidad y que no siempre respeten todas las normas o buenas prácticas de desarrollo impuestas por el sistema operativo.

Una de las principales funciones del software de control de dispositivos de entrada y salida es que sepa interpretar los códigos de error y detecte las situaciones anómalas de operación. Además,también se le considera el responsable de corregir muchas de las situaciones anómalas que se presenten, implementando los procedimientos de recuperación automática propios del dispositivo y que en aquellas situaciones donde no se pueda recuperar la operación sin intervención del usuario propague, con las interfaces estándar, la información apropiada, para que las aplicaciones y los usuarios puedan tomar las medidas correctivas necesarias para restaurar la operación del sistema de información.

En sistemas operativos sencillos, como en aquellos orientados a diversos sistemas embebidos, donde el desarrollo del sistema operativo se realiza para arquitecturas específicas, se permite que los controladores de dispositivos se comuniquen directamente con estos. Sin embargo, en sistemas de tiempo compartido o de tiempo real, donde se tienen más recursos pero se desea dar más flexibilidad a las plataformas de hardware en los que el sistema operativo y sus dispositivos podrán operar, se utilizan servicios que abstraigan la comunicación con el hardware, con el fin de que los controladores no dependan de

las características de la plataforma en que se está usando el dispositivo, por ejemplo si se trata de una plataforma de 32 o de 64 bits.

posteriores. Eso puede dejar inoperante nuestros controladores, perjudicando a nuestros usuarios y generando costos adicionales en la necesidad de generar nuevos controladores.

5.7 Interfaz con el hardware

El sistema operativo suele ser responsable de manejar la interacción con el hardware mediante dos tipos de mecanismos principales: las interrupciones y la abstracción del hardware.

Manejo de interrupciones

Las interrupciones son un mecanismo muy útil para evitar acaparar la atención de la CPU en espera de operaciones de entrada y salida, aunque constituyen una carga poco predecible que perturba la planificación de los procesos. Por lo anterior, resulta indispensable tomar todas las precauciones necesarias para minimizar su impacto en el desempeño.

De este modo, las rutinas que atienden a las interrupciones deben ser necesariamente rápidas en su ejecución, para lo cual se debe reducir su complejidad y su tamaño. Asimismo, también deben omitirse del modelo de procesos, para evitar la carga administrativa asociada, y ejecutarse en el contexto del *kernel* del sistema operativo bajo un estado inicial común a todas estas.

A continuación conviene revisar aquello que sucede cuando la CPU debe suspender la atención de un proceso para reaccionar ante una interrupción, a fin de comprender el impacto que pueden tener y los detalles de su implementación.

1. El circuito controlador del dispositivo usa las líneas de interrupción (IRQ) para notificar al procesador que requiere atención.
2. Al inicio del próximo ciclo de ejecución de una instrucción, el procesador detecta la interrupción y responde con la señal de `Acknowledge`, para que el dispositivo pueda liberar las líneas de interrupción.
3. Se almacenan los elementos mínimos del estado del proceso que se estaba ejecutando cuando ocurrió la interrupción; el contador de programa (PC) y la palabra de estado se almacenan en una pila de control.
4. El procesador carga un estado inicial para la rutina de atención de interrupciones en la palabra de control y, en función del número de interrupción, determina cuál de las rutinas de atención de interrupciones debe emplearse. Como todas estas rutinas son del mismo tamaño, puede determinar la dirección a cargar en el PC con la di-

rección base del vector de rutinas de interrupciones y el `offset` con el número de interrupción multiplicado por el tamaño de las rutinas individuales. En el proceso, verifica que el nivel de privilegios del proceso o del dispositivo que generó la interrupción sea adecuado para la interrupción que genera. Esta protección evita que procesos de usuario tengan acceso a ciertas rutinas de atención de interrupciones.

5. Inicia la ejecución de la rutina de atención de interrupciones. Algunas de estas podrían ser tan sencillas que no requieran modificar los demás registros del procesador, pero por lo regular es necesario resguardar el estado de todos los registros para que cuando el control se regrese al proceso, este pueda continuar su ejecución y la rutina pueda comenzar por almacenar esta información en la pila de control.
6. La rutina de atención de interrupciones empieza a atender la operación; para ello, puede comenzar por cargar sus propias direcciones de segmentos de pila y de datos que se manejan en áreas de memoria del kernel en registros de la CPU. Asimismo, procede a interactuar con el dispositivo, ya sea mediante su mapeo a memoria, sus registros especiales o cualquiera que sea el mecanismo propio del dispositivo; por esta razón, las rutinas de atención de interrupciones deben ser específicas para el dispositivo que levanta la interrupción.
7. Una vez concluida la ejecución de la tarea de la rutina, esta recupera el valor de los registros de la CPU de la pila de control en la CPU, en caso de que hubiera requerido usarlos.
8. Por último, recupera de la pila de control la palabra de control y la dirección del PC para reanudar la ejecución del proceso original.

En el caso de los procesadores Intel, se distinguen además dos tipos de interrupciones:

- **Síncronas.** Estas son producidas por la propia unidad de control de la CPU a petición expresa de los procesos o por encontrar situaciones anómalas en el procesamiento. Dichas interrupciones corresponden con las excepciones y reciben el nombre de síncronas porque necesariamente se generan al terminar la ejecución de una instrucción de la CPU y, por tanto, en sincronía con el reloj del procesador.
- **Asíncronas.** Estas son generadas por otros dispositivos de hardware que operan de forma independiente al reloj de la CPU.

Un aspecto muy importante que debe considerarse al diseñar las rutinas de atención de interrupciones es el manejo que se hará de interrupciones que ocurran durante la ejecución de otra rutina de atención de interrupciones. En principio, podría evitarse que una interrupción afectara la ejecución de otra; sin embargo, esto podría deteriorar

las condiciones de operación de dispositivos que requieran una atención muy rápida y, en general, podría ir en detrimento del balance del sistema de información, ya que generaría mayores tiempos de espera en la operación de los dispositivos de E/S, razón por la que suele permitirse la atención anidada de interrupciones, permitiendo que la rutina de atención de una sea interrumpida por la señal de otra.

Cuando una de las rutinas de interrupción se encuentra en una región crítica, puede instruir a la CPU para no recibir nuevas interrupciones, como se abordó en el tema de solución de condiciones de competencia, por lo que la región crítica debe ser lo más pequeña posible.

En el caso de los procesadores Intel, a las interrupciones que no se les prestará atención se les llama **interrupciones enmascarables** (`Masked`). Adicionalmente, los circuitos en la tarjeta principal que codifican las señales de cada dispositivo en el valor que se envía a la CPU, llamado Programmable Interrupt Controller (PIC), y en los sistemas con múltiples procesadores, se agregan a circuitos llamados I/O Advanced Programmable Interrupt Controller (I/O APIC), los cuales pueden ser programados para suspender de manera temporal la generación de interrupciones.

Abstracción de hardware

El objetivo del software de tener la mayor independencia posible del hardware específico con el que se opera también afecta a los controladores específicos de dispositivos. Aunque los controladores implementen la funcionalidad específica, es bueno que no requieran conocer las características de todos los demás dispositivos y de la arquitectura específica en la que se esté operando. Para ello, diversos elementos de la comunicación con el hardware tienen capas de software asociadas que ocultan sus características específicas y permiten que los controladores se desarrollen sobre estas abstracciones.

En el caso de Windows, se emplea el Hardware Abstraction Layer (HAL), que expone una única interfaz para el *kernel*. En este la administración de dispositivos de E/S y los controladores de dispositivos cuenta con tres implementaciones:

1. **Halacpi.dll.** Para arquitecturas de 32 bits en sistemas de un único procesador.
2. **Halmacpi.dll.** Para sistemas que usan APIC en arquitecturas de múltiples procesadores.
3. **Hal.dll.** Para arquitecturas de 64 bits, ya que estas siempre usan APIC.

En el caso de Linux, el acceso a los dispositivos se uniforma tratando a todos como si estuviesen conectados a puertos de E/S y usando los mecanismos y estructuras de datos correspondientes.

5.8 Operaciones de dispositivos de entrada/salida

A continuación se describen las operaciones de dispositivos de entrada/salida.

Operaciones comunes

El mecanismo que proporciona el mayor grado de independencia de dispositivo a las aplicaciones utiliza las mismas API de programación de archivos para manejar los dispositivos. Las operaciones más comunes que se realizan sobre los archivos y sobre los dispositivos de entrada y salida que explotan este mecanismo de representación son:

- **Apertura (reserva) del dispositivo de E/S.** Como se señaló antes, en los sistemas operativos UNIX y Windows es posible identificar los dispositivos de entrada y salida mediante nombres de archivo. Por ejemplo, con el uso de la función `fopen` podemos abrir un canal de comunicación hacia los dispositivos de entrada y salida mediante el cual podemos realizar las operaciones siguientes, y que queda representado en la estructura de datos conocida como manejador de archivo (File Handler).
- **Escritura.** Para enviar información al dispositivo basta con usar las funciones estándar para empezar a utilizar los manejadores de archivo, como `fprintf`, `fputs` o `fwrite`. Estas funciones están preparadas para utilizar los dispositivos de E/S, aprovechando la compatibilidad de la capa de administración de los mismos. Incluso algunos dispositivos con interfaces específicas, como la comunicación de red, establecen mecanismos estándar (en este caso, los flujos) para recibir la información que se enviará por medio de un `socket` de manera uniforme a otras aplicaciones usando las funciones antes mencionadas, las cuales solo usan su API particular para establecer el flujo usando un `socket` y no mediante un manejador de archivo.
- **Lectura.** Para leer información basta usar las correspondientes operaciones de lectura de información, típicamente usadas con archivos mediante funciones como `fscanf`, `fgetc` o `fgets`. Incluso se cuenta con la función `fread` para realizar la lectura por bloques a búfer de memoria.
- **Cierre.** Empleada hasta el final. Por lo general, cuando se ha terminado de utilizar el archivo o el dispositivo de E/S, esta operación da por terminado el canal de comunicación, instruyendo al sistema operativo para que termine la escritura de la información contenida en los búferes a los dispositivos o archivos con la función `fclose`.

Además de estas operaciones comunes, usualmente también se tienen interfaces que permiten a las aplicaciones tomar mayor control de la comunicación que tienen con el sistema de archivos o con las aplicaciones. Así, cuando el aspecto que tratan de controlar estas interfaces es la sincronización de la solicitud de las operaciones con las operaciones en el dispositivo, hablamos de operaciones síncronas y asíncronas.

Operaciones síncronas y asíncronas

Desde la perspectiva de las aplicaciones, la forma más sencilla de solicitar una operación de entrada y salida implica que cuando la operación haya realizado la petición, bloqueará el proceso o hilo que la realizó hasta que la operación termine, con lo que ya no requerirá hacer ninguna implementación adicional para seguir utilizando el dispositivo de entrada y salida para la siguiente operación, sea de lectura o de escritura.

Se conoce como síncrona la invocación de una operación en un dispositivo de entrada y salida que bloquea al solicitante mientras realiza la operación, debido a que coordina la interacción del proceso que solicita con las operaciones.

Por otra parte, si el proceso no desea que el proceso o hilo se bloquee para enviar una serie de peticiones del mismo tipo, sean lecturas o escrituras, y aprovechar la habilidad de optimizar el balance y la atención de las peticiones que pudieran tener las capas de administración de dispositivos de entrada y salida y de los controladores de dispositivos, puede utilizar interfaces de programación asíncronas, las cuales no bloquearán el proceso o hilo que realizará las peticiones. Al usar peticiones asíncronas es muy importante que antes de cambiar el tipo de peticiones, por ejemplo, para recuperar información que hayamos enviado a un dispositivo, nos aseguremos de que las peticiones anteriores hayan terminado, para lo cual será necesario usar funciones adicionales en la API correspondiente.

5.9 Estructuras de datos para manejo de dispositivos

A continuación hacemos una revisión de algunas particularidades de la implementación existente en algunos sistemas operativos, resaltando solo sus generalidades, ya que por cuestiones de espacio no es posible ahondar en sus interfaces de programación y características específicas. En especial nos referimos a las características actuales de los sistemas Windows 7 y Linux con *kernels* 2.4 y posteriores, debido

♠ **Arquitecto.** Es importante señalar que muchos otros sistemas operativos tienen implementaciones diferentes, por lo que para realizar desarrollos en arquitecturas específicas primero es necesario verificar cuáles son los lineamientos de di-

seño específicos para la plataforma y sistema operativo objetivo de nuestro software de control o para que se aprovechen los dispositivos de E/S. No obstante, confiamos en que encontrarás que hay muchos puntos en común con los aquí revisados.

- ♦ **Líder.** La decisión de utilizar la funcionalidad específica de un controlador o de una arquitectura para ganar control o desempeño en las operaciones al final depende de los requerimientos de la aplicación; sin embargo, un buen punto de partida consiste en tomar la independencia de dispositivo como un objetivo general y evitar el desarrollo a nivel de arquitectura o controlador particular, siempre que sea posible.
- ♠ **Usuario.** Cuando se desea incursionar en el desarrollo de controladores, siempre es necesario contemplar la adquisición de kits de desarrollo, libros y servicios de certificación y asesoría adecuados; asimismo, debe considerarse que el esfuerzo de pruebas, por ejemplo, requerirá realizarse en múltiples plataformas y versiones del sistema operativo, con el fin de poder asegurarnos que nuestros productos funcionarán adecuadamente para nuestro mercado objetivo.

a que estos son de los más representativos, a pesar de que ambos son sistemas de tiempo compartido. Por ejemplo, Linux cuenta con implementaciones muy influyentes para sistemas de tiempo real y opera en sistemas embebidos como *kernel* del sistema operativo Android. En ambos casos, la implementación de los controladores de dispositivos de entrada y salida es la misma, aunque los mecanismos para cargar esta funcionalidad en el *kernel* puedan presentar algunas variaciones. Por su parte, Windows tiene una gran popularidad, lo que por sí mismo lo hace digno de estudio.

Empaquetado de las solicitudes de entrada y salida

El software de administración de entradas y salidas del sistema operativo requiere de mecanismos para representar las peticiones que recibe de las aplicaciones durante la atención de estas, con lo que puede mantener el estado de la petición a lo largo de la atención que esta reciba por las diversas capas. Aunque algunos mecanismos específicos permitan acceder a los controladores o a funcionalidades específicas, la mayoría de las peticiones y todas aquellas que usen la abstracción de dispositivo como archivo utilizarán este tipo de estructuras para representar la petición.

En el caso de Windows, una vez que la interfaz de archivos ha recibido la petición, la parte de la administración de dispositivos de entrada y salida la codifica en un Input/Output Request Packet (`IRP`). En tanto, en el caso de Linux, la representación de una operación en un dispositivo de entrada y salida de bloques se realiza en la estructura Block Input/Output Operation (`bio`).

En cuanto a la información que se maneja de cada petición, esta varía, en gran medida, entre los diversos sistemas operativos. En caso de que se desee hacer el desarrollo de un controlador de dispositivos o una aplicación que trascienda la representación uniforme de archivos y se comunique directamente con un controlador para acceder a operaciones especiales o mejorar el rendimiento, se recomienda revisar el detalle de las estructuras, las funciones y la arquitectura específica del sistema operativo objetivo.

Ambos empaquetados de la operación tienen dos componentes fundamentales, el primero de los cuales consta de una serie de atribu-

tos que describen el estado y la funcionalidad de la petición (al que podríamos llamar encabezado o descriptor de la operación), y el segundo, que consta de los rangos de memoria utilizados para intercambiar los datos con el dispositivo de E/S.

Información sobre la petición

- **Listado de peticiones.** A menudo los dispositivos de E/S reciben variados conjuntos de peticiones a lo largo del tiempo, por lo que requieren construir una fila para atender todas estas peticiones, ya que el tiempo de atención suele ser mayor al tiempo que toma a los procesos generarlas. Una vez construida la lista, los controladores de dispositivos quedan libres para atender las peticiones en el orden en que se pueda lograr una mejor eficiencia. Esta lista de peticiones se implementa como una lista doblemente ligada de `bio` en Linux y mediante un `IRP` maestro en el caso de Windows; no obstante, en ambos casos, corresponde a la parte de la estructura de la petición representar la lista.
- **Estado de la petición.** Toda aquella información que permita detectar de manera rápida y uniforme el estado en el que se encuentra la petición siempre debe almacenarse, incluso si se ha solicitado que la operación se cancele, lo que no garantiza que la operación haya terminado.
- **Tipo, tamaño y ubicación de los búferes.** Siempre se debe tener la referencia a los búferes de memoria que se están empleando para intercambiar información con el dispositivo. Estos búferes pueden corresponder a direcciones mapeadas a memoria de los dispositivos, rangos de memoria en el espacio del *kernel* del sistema operativo o segmentos de memoria de las aplicaciones que los controladores utilizarán para intercambiar información directamente con el dispositivo, con el objeto de ahorrar la copia de memoria a áreas de usuario a áreas del *kernel* y luego a los dispositivos.
- **Estado de las transferencias de memoria.** Es indispensable registrar el progreso que se tiene en la transferencia de la información a los búferes de memoria. Sin embargo, hay que poner especial cuidado en los casos en que la información recibida del dispositivo exceda la longitud del búfer o la petición se divida en varias partes que no excedan el tamaño del búfer, para evitar desbordar las estructuras en memoria.
- **Control de referencias.** Para determinar el momento en que la operación puede concluir, es necesario llevar un registro de cuántos procesos o hilos de ejecución se encuentran utilizando el controlador, con el fin de conocer cuándo se ha terminado de utilizar; ello ocurre en el momento en que ningún proceso o hilo sigue usándolo. Por ejemplo, en el caso de las tuberías con nombre (Named Pipes), que se usan

como archivos que existen solo en memoria, cuando están abiertos por múltiples procesos permiten intercambiar información entre estos.

- **Operaciones asociadas a la operación.** También se requiere almacenar las direcciones de las operaciones que deben llevarse a cabo cuando las operaciones en curso terminen, cuando se deba destruir la estructura de datos o el objeto que representa la operación o en algunos otros momentos.
- **Información para el controlador.** De igual modo, se suelen almacenar apuntadores a segmentos, banderas de estados y otros datos que serán de utilidad al controlador en lo que se atiende la petición. El principal detalle de estos datos es que son completamente dependientes de la API para controladores de cada sistema operativo, e incluso varían entre versiones de los mismos.

Búfer de memoria

Una parte fundamental del empaquetado de la petición es el control de los rangos de memoria que representen tanto la interfaz con el dispositivo que usará el controlador, como los segmentos de memoria en los que se intercambiará la información en la memoria RAM. En Windows se emplean los objetos **IO_STACK_LOCATION** y en Linux las estructuras **`bio_vec`**. En general, estos objetos contienen:

- **Ubicación en la memoria.** Siempre se debe almacenar la estructura adecuada para indicar la ubicación en memoria en que comienza el búfer.
- **Longitud.** En todo momento es necesario controlar el tamaño del búfer para verificar que las operaciones que realicemos no excedan de la memoria asignada al búfer en cuestión.
- **Descriptor.** Depende de cada sistema operativo, pero en general se refiere a la información de la administración de memoria (página) o a la información acerca de la operación, las banderas de control, los parámetros para la operación y los detalles del dispositivo al que se asoció este rango de memoria.

Jerarquías de representación de los dispositivos

Con base en el seguimiento de la mecánica de la conexión de muchos dispositivos a la computadora, que se realiza mediante puertos manejados por controladores, con frecuencia se requiere establecer una jerarquía de los controladores de los dispositivos que refleje la organización de las conexiones entre los distintos dispositivos y sus respectivos circuitos de control e interfaz.

Para ello los sistemas operativos definen diversos tipos de representaciones lógicas que ayudan a agrupar los dispositivos y establecer capas de controladores, que permiten una interacción adecuada con los dispositivos en estas situaciones.

En el caso de Windows, la clasificación considera cuatro tipos de objetos:

- **Class drivers.** Operaciones de E/S para una familia de dispositivos, como los recursos de la tarjeta principal, los dispositivos de almacenamiento masivo, los dispositivos de interfaz con el usuario, etcétera. Los `class drivers` exponen la funcionalidad genérica que se espera de este tipo de dispositivos e incluso pueden controlar directamente dispositivos que se apeguen a normas de operación comunes, como los módems Hayes y diversos tipos de teclados o ratones que no requieren controladores específicos.
- **Miniclass drivers.** Operaciones y mecanismos de comunicación específicos para un dispositivo en particular, que por lo regular son implementados por el fabricante del dispositivo. Es importante señalar que estos se cargarán en el *kernel* y no deberán atender IRP directamente de parte de las aplicaciones, sino recibir sus peticiones mediante los `class drivers`.
- **Port drivers.** Operaciones específicas a un tipo de puerto de entrada y salida, como el USB o el SATA. Estas operaciones también se implementan en el *kernel* y dan soporte a la funcionalidad de los controladores de los dispositivos específicos que se conectan mediante estos puertos. Por lo común, los `port drivers` siempre son generados por la propia compañía Microsoft con el fin de garantizar que los fabricantes de dispositivos tengan acceso a la misma funcionalidad para desarrollar sus controladores. Solo en el caso de hardware muy especializado se generan controladores de puertos específicos.
- **Miniport drivers.** Peticiones genéricas recibidas mediante un controlador de puerto que dirigen a un adaptador de conexión específico. Por ejemplo, entre los puertos de conexión de red existen minipuertos específicos para las diversas interfaces de comunicación con los dispositivos que manejan Wi-Fi, donde cada dispositivo tiene su propio controlador. A menudo es posible dirigir adecuadamente, con `miniclass drivers` y `miniport drivers`, las peticiones mediante los puertos USB o PCI, dependiendo del tipo de conexión del dispositivo al equipo. Esta estructura le permite aprovechar la funcionalidad ya existente para controlar el puerto (por ejemplo, USB), logrando una adecuada división de responsabilidades entre los diversos controladores.

  En el caso de Linux 2.6 y versiones posteriores, se busca uniformar el modelo y la organización de los controladores con el llamado `device driver model`, mediante el cual los dispositivos siempre se modelan como conectados a puertos y

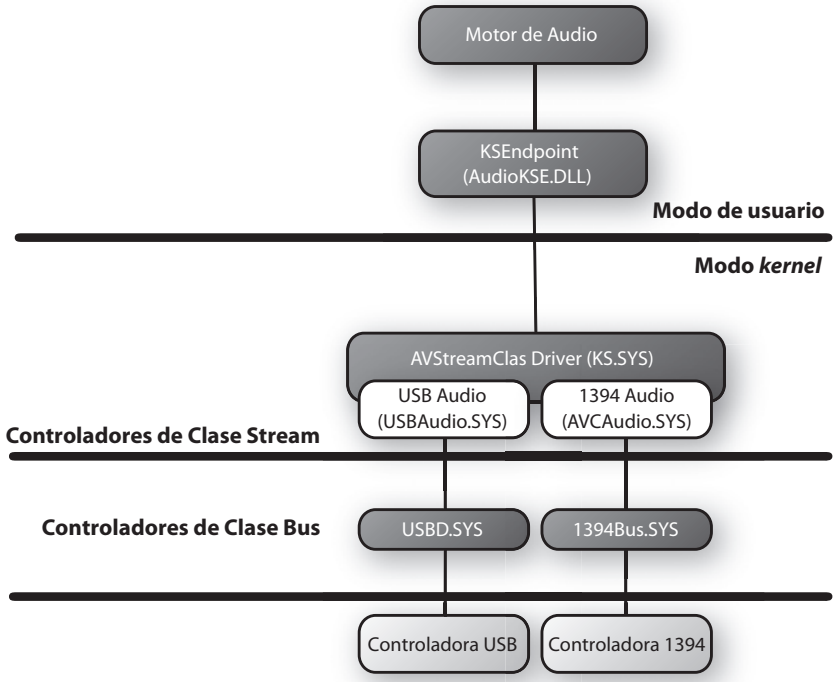


Figura 5.4 Estructura de controladores de audio típica a Windows.

se representan como archivos bajo el directorio/sysfs, donde se establece una jerarquía de estructuras de datos para la representación de las distintas etapas de la conexión, típicas de un dispositivo que se encuentra conectado a un puerto y que conforma la representación general para los demás tipos de dispositivos. Esta jerarquía tiene tres formas de representación en estructuras de la API: `subsistemas`, `kset` y `kobject`.

- **Subsistemas.** Agrupaciones de estructuras `kset` (adicionalmente al `kset` que contienen) que implementan un semáforo para proteger el acceso a los miembros.
- `kset`. Colecciones de `kobject` con el mismo tipo de contenedor; es decir, corresponden a dispositivos del mismo tipo. Tienen una referencia al subsistema al que pertenecen, al `kobject` que describe el tipo al que pertenecerán todos los miembros, a la lista doblemente ligada de los `kobject` que agrupa y a un `kobject` que se considera del mismo nivel que el `kset`, que es el encargado de representar a toda la colección y de implementar las operaciones de administración de las referencias a los dispositivos; incluso, gracias a este se puede agregar el `kset` a otro `kset`, ya que para ello basta agregar el `kobject` asociado al conjunto en el conjunto que ha de contener la colección.

- `kobject`. Núcleo del modelo de controladores de dispositivos. Se trata de una estructura genérica para representar cada directorio en el sistema de archivos bajo `sysfs`; por tanto, representará a los dispositivos y sus agrupaciones, manteniendo las referencias a los demás elementos en la organización.

Una vez descrita la jerarquía de los elementos con los elementos anteriores, ahora describimos la funcionalidad mediante las siguientes estructuras de datos:

- `device`. Representa a cada dispositivo. Para ello mantiene un registro de referencias a los demás dispositivos conectados al mismo nivel que este, a su dispositivo padre y a los dispositivos en el mismo tipo de bus que esté. También mantiene la referencia al controlador para el dispositivo, a los datos privados asociados al control de este dispositivo, a la información para administración de potencia, a la información para controlar el DMA del dispositivo (en caso de que utilice una controladora DMA) y a una función que se emplee cuando el dispositivo deje de utilizarse.

  En este caso, los dispositivos se organizan jerárquicamente con ayuda de las estructuras `kobject`, para llevar el control de los buses, los controladores y el resto de los mecanismos de comunicación por los que las operaciones deberán pasar para llegar a los dispositivos puntuales. Como cada una de estas etapas constituyen en sí mismas dispositivos de distintos tipos, resulta recomendable mantener la información de la jerarquía separada de la información para indicar el controlador de cada dispositivo, de modo que esta última se pueda utilizar por todas las rutas que la requieran.
- `device_driver`. Representa a los controladores de dispositivos. Incluye un nombre, una referencia al bus donde se conectan los dispositivos, un semáforo para prevenir que el controlador se descargue de memoria mientras está siendo utilizado por alguna aplicación, el `kobject` que representa su lugar en la jerarquía, una lista con todos los dispositivos controlados por este y las referencias a métodos del controlador que se invocará en caso de que se retire o se apague el dispositivo, se ponga en un estado de bajo consumo de energía o en el estado normal de operación y consumo de energía.

  Esta estructura también se asocia con tres funciones que deben implementar los controladores y que están orientadas a manejar la conexión de un dispositivo durante la operación del sistema de información (hot-plug): 1) determinar los controladores adecuados de un dispositivo cuando se inicia su operación (plug and play), 2) administrar la potencia, y 3) detectar si el controlador es adecuado para

manejar un dispositivo cuya identificación sugiere que podría ser atendido por un controlador en particular (`probe`).

El `kobject` asociado al controlador se emplea para llevar el control del número de usos que el controlador está teniendo, con objeto de conocer el momento en que estas terminen y el controlador pueda ser liberado.

Por lo regular, los controladores se incluyen de forma estática en descriptores de un nivel más elevado; por ejemplo, los controladores de dispositivos que se conectan mediante el bus PCI suelen incluirse en estructuras de datos de tipo `pci_driver`, que complementa la información con campos específicos para controladores PCI.

- `bus_type`. Representan cada tipo de bus de conexión de dispositivos presente en el sistema. Incluyen un `kobject` asociado al bus, el conjunto de `kobject` de los controladores asociados a los dispositivos conectados, los `kobject` de los propios dispositivos, referencias a estructuras de datos con los atributos propios de un bus, del controlador del bus y los atributos de este, y métodos para controlar el reconocimiento, el registro, la potencia y el contexto de los dispositivos conectados a este.
- `class`. Representa cada clase de dispositivos de entrada y salida. Es importante resaltar que todas las clases pertenecen al subsistema `class_subsys` y que en sí mismos estos constituyen un subsistema que corresponde al tipo de dispositivos que representan.

  Cada clase tiene un conjunto de estructuras `device` que representa los dispositivos lógicos, cada uno de los cuales puede atender peticiones de operaciones del tipo de dispositivos representada por la clase. Los dispositivos lógicos son ge-

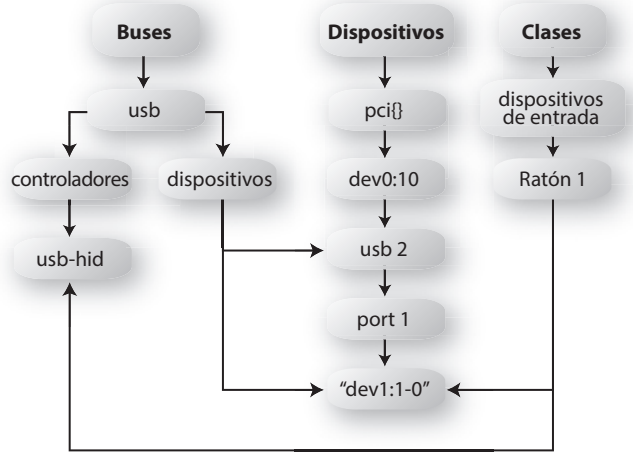


Figura 5.5 Jerarquía de elementos típica de Linux.

nerados por los dispositivos físicos, pero se distinguen de estos porque un solo dispositivo físico puede exponer funcionalidad para varias clases de dispositivos distintos; por ejemplo, un ratón con más de tres botones podría exponer la funcionalidad de los botones adicionales como eventos de teclado, por lo que presentaría dispositivos de clase USB para ratón y para teclado.

El objetivo de las clases es proporcionar un método uniforme para que las aplicaciones puedan obtener las interfaces a los dispositivos lógicos que requieren.

Interacción con el controlador del dispositivo

En Linux, la organización de los dispositivos y de los controladores de dispositivos, como parte de las clases de dispositivos, está expuesta a las aplicaciones. Así, en caso de que dichas aplicaciones deseen revisarla a detalle e interactuar con los dispositivos específicos, solo tendrán que revisar la estructura de `kobject` y ubicar los controladores que les interesan.

Por su parte, Windows cuenta con las API de programación Fast I/O y Scatter/Gather I/O, las cuales no utilizan el mecanismo de IRP para encapsular las peticiones a los controladores.

En Windows, fast I/O permite comunicarse directamente con el controlador de dispositivo, sin generar IRP. Esta interfaz se usa con frecuencia para recuperar información que los controladores han obtenido del dispositivo y colocado en memoria caché. De este modo, en caso de que la información no esté disponible en el caché, se revierte a una solicitud empleando IRP; sin embargo, dispositivos que requieran mejorar su desempeño controlando el estado de sus cachés también pueden beneficiarse de esta. No obstante, esta no se recomienda para comunicarse con controladores de dispositivos de E/S que no manejen sistemas de archivos.

Por su parte, Scatter/Gather I/O permite especificar, en una sola operación, conjuntos de operaciones de lectura o de escritura que utilicen memoria contigua para el intercambio de información de todas las operaciones, de modo que el controlador pueda determinar el mejor orden o incluso atender todas las peticiones en paralelo. A menudo se usa para leer conjuntos de bloques de disco, en cuyo caso las peticiones deben referirse además a bloques contiguos en el disco.

Dispositivos virtuales

Hay un caso especial en la implementación de los controladores de dispositivos de entrada y salida en la que el software del controlador, en vez de presentar la funciona-

lidad del dispositivo, implementa la funcionalidad que la clase de dispositivo a la que pertenece espera, realizando operaciones de una naturaleza completamente diferente en otro tipo de dispositivo o incluso en otro subsistema.

Estos dispositivos que reciben una implementación radicalmente diferente de la que exponen a las aplicaciones se conocen como dispositivos virtuales.

Como ejemplos de estos podemos mencionar las impresoras virtuales, que pueden almacenar contenido del trabajo de impresión en diversos tipos de archivo, como PDF, PS, etc., o incluso enviarlo mediante la red a una impresora conectada a otro equipo, o convertirse en un centro de impresión.

EVALUACIÓN

5.1 ¿Qué destino puede tener la información procesada en un sistema informático que no presente los resultados directamente a ninguna persona? Expón un ejemplo.

5.2 ¿Qué compañía comenzó la tendencia de usar normas abiertas para ser utilizadas por cualquier compañía en materia de interfaces entre computadoras y dispositivos de E/S?

5.3 ¿Qué influencia tuvo el Palo Alto Research Center en las interfaces de usuario modernas?

5.4 En un sistema operativo moderno, una aplicación que desea usar una impresora conectada en red, ¿qué tipo de dispositivo deberá usar al implementar la utilización?

5.5 ¿Por qué es importante la independencia de dispositivos?

5.6 Si el procesador no puede usar la memoria RAM al proseguir la atención de procesos en lo que una controladora DMA está realizando la transferencia de información desde un dispositivo, ¿qué caso tiene el uso del DMA?

5.7 ¿Qué es la espera activa y cómo se aplica a los dispositivos de E/S?

5.8 ¿Por qué se evita que el controlador de un dispositivo en particular se comunique directamente con este, en favor de rutinas de atención de interrupciones y capas de abstracción del hardware?

5.9 ¿Cuál es la diferencia entre una excepción y una señal?

5.10 ¿Qué capa se encarga de identificar un dispositivo para cargar los controladores correspondientes?

5.11 ¿En qué tipo de sistemas operativos es más importante la administración de potencia?

5.12 ¿Cuál es la relación entre la atención de interrupciones y el manejo de excepciones?

5.13 ¿Cuáles son las operaciones comunes de los dispositivos de E/S y qué familias no las proporcionan?

Referencias bibliográficas

Bovet, Daniel P. y Cesati, Marco, *Understanding the Linux Kernel,* 3a ed., 2005.
Silberschatz, Abraham, Baer Galvin Peter y Gagne Greg, *Operating System Concepts,* 7a ed., 2005.
Solomon, David y Russinovich Mark, *Windows Internals,* 6a ed., 2012.
Stallings, William, *Operating Systems Internals and Design Principles,* 5a ed., 2008.
Tannenbaum, Andrew S., *Modern Operating Systems,* 3a ed., 2009.

Referencias electrónicas

EMICROS How-To-Guide; Using the 68hc11 Analog to Digital Converter: http://www.emicros.com/a2d.htm

All About Circuits: http://www.allaboutcircuits.com

Understanding Error Messages: http://www-numi.fnal.gov/offline_software/srt_public_context/WebDocs/Errors/index.html

COMPETENCIAS A DESARROLLAR

- El lector aprenderá a identificar los elementos de un sistema de archivos, y a este de los demás elementos del sistema operativo.
- Identificará los objetivos generales de los sistemas de archivos y sus componentes.
- Conocerá los elementos, tipos y mecanismos de utilización de archivos.
- Comprenderá cómo utiliza el sistema de archivos los dispositivos de almacenamiento y los mecanismos que se construyen para brindar acceso rápido a cualquier archivo.
- Conocerá los principales aspectos de seguridad de la información y tolerancia a fallos presentes en los sistemas de archivos.
- Conocerá las principales interfaces de programación empleadas para desarrollar aplicaciones que emplean el sistema de archivos.

Administración de sistemas de archivos

6

¿QUÉ SABES...?

1. ¿Cómo se almacena información en la computadora?
2. Si se va a cambiar de computadora, ¿qué se requiere conservar del equipo que se dejará de utilizar?
3. ¿Cómo se define un archivo de computación?
4. ¿Qué tipos de archivos existen?
5. ¿Qué define y qué diferencia a los diferentes tipos de archivos?
6. ¿Cómo organizas los archivos para encontrarlos con facilidad?
7. ¿Cómo puedes controlar el acceso a la información contenida en un archivo?
8. ¿Cómo puedes reorganizar o eliminar archivos?
9. Cuando una aplicación usa archivos, ¿qué operaciones puede realizar?, ¿qué debemos hacer como usuarios cuando dicha aplicación termina de utilizarlos?
10. ¿Qué tipos de fallos has experimentado en tus sistemas de archivos?, ¿cómo los resolviste?, ¿qué impactos generaron con tu información?

Importante

♥ **Usuario.** Los archivos surgieron como abstracciones de las colecciones de información almacenadas en los dispositivos de almacenamiento secundario, debido a que resulta muy conveniente utilizar la misma mecánica y organización en todas las aplicaciones. Sin embargo, una vez que se establecieron los archivos, las aplicaciones encontraron rápidamente otras formas de implementar el almacenamiento de la información contenida en el archivo, que aportaban soluciones a un gran espectro de necesidades. Hoy día, los archivos

6.1 Introducción

Los sistemas de archivos constituyen uno de los aspectos más visibles y de fácil acceso del sistema operativo para los usuarios finales debido a que el almacenamiento de información relevante realizado en dichos sistemas requiere identificar cada uno de los conjuntos de datos de forma independiente a las aplicaciones, lo que resulta de gran importancia para que el usuario aproveche el sistema de información lo mejor posible a través del tiempo.

Para una mejor comprensión del tema central de este capítulo, más adelante se hace un recordatorio de las definiciones de conceptos como **dato**, **información** y **conocimiento**, expuestas con anterioridad.

Por lo regular, los archivos construyen su contexto a través de las aplicaciones que los utilizan, por lo que se dice que estos contienen información. Sin embargo, el archivo, en sí mismo y sin mecanismos que lo interpreten, solo contiene un conjunto de datos, debido a que estos datos, aunque reducen incertidumbre, no están asociados a un contexto por sí mismos.

A continuación se definen de manera formal los conceptos que tratamos en este capítulo, pues aunque la mayoría de las veces estos forman parte de las tareas cotidianas de los lectores, los conceptos espontáneos o populares pueden no corresponder con el planteamiento que se maneja en el presente, además de que resulta muy importante aclarar la perspectiva que vamos a usar, con el fin de evitar confusión al proseguir el tratamiento del tema.

- **Archivo.** Abstracción que representa una colección de datos con identificación, que en general se desea conservar más tiempo que el que duran los procesos que la crean. Un archivo siempre tiene un propósito particular, posee una estructura regular y sigue un código con reglas regulares de representación.
- **Directorio.** Contenedor de archivos o de referencias a archivos. En los sistemas de archivos modernos, los directorios también se consideran archivos. De este modo, es sencillo definir estructuras de directorios, ya que un directorio puede contener a otros en su calidad de archivos.
- **Metadatos.** Conjunto de información que describe un conjunto de datos. En el caso de los archivos, se trata de sus atributos e información de administración, en contraste con los datos contenidos en el archivo.
- **Sistema de archivos.** Aplicación encargada de implementar las colecciones de información organizadas en archivos y las aplicaciones que se pueden realizar sobre estos.
- **Atributos de archivo.** Metadatos asociados a cada archivo contenidos como parte de un sistema de archivos.

6.2 Objetivos de un sistema de archivos

En la actualidad existe gran variedad de sistemas de archivos que atienden un número importante de necesidades diversas, por lo que los objetivos también deben ser planteados de forma general y considerar las aplicaciones específicas como implementaciones particulares. A continuación revisaremos los objetivos generales de un sistema de archivos.

representan dispositivos de entrada y salida, mecanismos de comunicación entre procesos, áreas de memoria compartida, colecciones de información en todo tipo de dispositivos de almacenamiento no volátil e incluso información acerca de procesos. Por tanto, el concepto central con el que debemos trabajar al utilizar los archivos debe ser su habilidad de dar a las aplicaciones un mecanismo estándar de acceder a información, la cual sea identificada con nombres y se organice en colecciones en nuestro sistema de archivos, por lo que no debe sorprendernos la continua evolución en los mecanismos y aplicaciones de estos.

♠ **Docente.** En sistemas operativos resulta muy útil considerar al archivo como una abstracción, en vez de la acepción común de concebirlo como el contenido específico que podemos almacenar en una computadora. Así, al revisar la variedad de implementaciones, funciones y propósitos que atiende un archivo, no necesitamos una definición de archivo diferente para cada aplicación, sino que podamos reconocer los elementos comunes y la utilidad de las características específicas de cada implementación.

Almacenar grandes volúmenes de información

Los sistemas de archivos originalmente se empleaban para aprovechar los dispositivos de almacenamiento secundario, con el objetivo de almacenar de forma estructurada la información que usarían las aplicaciones, más allá de las capacidades de la memoria y de la duración de una sola sesión en la computadora, pues en general los sistemas de archivos deben almacenar más información de la que resulta relevante para la ejecución de un proceso en particular o en una ocasión en particular, y en cambio almacenan información por periodos prolongados y que pueden emplearse por múltiples aplicaciones.

Así, no solo tendremos los archivos empleados por los procesos en esa ocasión en particular, sino también los que se empleen en otros momentos. Incluso, cuando un archivo se utiliza como mecanismo de comunicación y no de almacenamiento a largo plazo, debemos almacenar la información en tránsito durante el tiempo que tome completar la transferencia, lo que a menudo deriva en la necesidad de mantener una cantidad considerable de información almacenada, que si se acumula en el tiempo fácilmente puede alcanzar grandes volúmenes de información. Esto implica que los mecanismos de implementación deben ser eficientes en el uso de la capacidad de almacenamiento de los dispositivos.

La información debe sobrevivir al proceso que la creó

Con base en la capacidad de almacenamiento actual de discos duros y otros dispositivos de almacenamiento secundario de tener almacenamiento no volátil de la información, tenemos que considerar que los datos almacenados en los sistemas de archivos deben sobrevivir a los procesos y a los reinicios del sistema. Por tanto, al implementar nuestro sistema de archivos en memoria volátil hay que distinguir la memoria asignada al proceso de la empleada por los archivos que usa. La capacidad de los archivos de existir y operar de forma independiente de los procesos que los crearon así lo requiere, tanto para compartir información entre múltiples procesos como para dar continuidad a la operación de un sistema en las ejecuciones sucesivas de un programa en múltiples procesos.

Asimismo, debe prestarse atención a salvaguardar la información ante fallos y durante la operación del sistema.

Múltiples procesos deben poder acceder la información de manera concurrente

La información contenida en los archivos debe tener independencia de los procesos que la generan, modifican o leen, no solo en su persistencia sino también en cuanto a

los procesos que tengan acceso a esta. En ese sentido, no solo se requiere que la información sea utilizada por varios procesos sucesivamente sino que también se pueda utilizar de manera concurrente por varios procesos, ya sea para que todos estos recuperen la misma información o como mecanismo para que la información escrita en el archivo por un proceso sea recuperada de modo correcto por los otros.

No obstante, como se aclaró en el capítulo 3, *Administración de procesos*, esto puede generar problemas de condiciones de competencia, por lo que el sistema de archivos deberá estar preparado para resolver dichas condiciones a fin de conservar la integridad de la información.

Proporcionar acceso a la información

El sistema de archivos debe implementar la API y los mecanismos necesarios para que las aplicaciones puedan usar una serie de operaciones sobre los archivos, a fin de que permitan generar y utilizar la información contenida en estos. Por ello suele incluir al menos operaciones para almacenar, recuperar, organizar y borrar la información. Estas operaciones deberán ser rápidas, realizarse de la misma manera para todos los tipos de archivos y lo más sencillas que resulte posible a fin de facilitar el desarrollo de aplicaciones. También debe vigilarse que las funciones para el mantenimiento del propio sistema de archivos sean tan accesibles y sencillas como se pueda.

Control de acceso

La información siempre debe protegerse con el fin de que esta se mantenga íntegra y sea utilizada de manera correcta y por los usuarios autorizados.

No solo deben vigilarse las posibles condiciones de competencia sino que se deben tomar medidas para reaccionar ante errores en las operaciones, en las aplicaciones y en los dispositivos, con el objetivo de salvaguardar la información de los archivos.

Debido a que la información contenida en los archivos se modificará a lo largo del tiempo, también es importante vigilar que los cambios no se realicen de manera parcial o incoherente, a fin de mantener la integridad de la información.

Un aspecto que a últimas fechas ha cobrado mayor importancia es el control de cuáles usuarios pueden acceder a la información de los archivos mediante los procesos que inician estos, así como las operaciones que pueden realizar en dichos archivos. Aunque este hecho podría revisarse como un objetivo independiente, está relacionado con las precauciones que se deben considerar al implementar las medidas que se toman para garantizar que los mecanismos de acceso y los refinamientos orientados a

mejorar el desempeño no solo garanticen la integridad, sino que también vigilen las restricciones y políticas de acceso a la información.

6.3 Componentes de un sistema de archivos

La estructura lógica del software que controla el sistema de archivos a su vez tiene su origen en la estructura lógica del software de control de dispositivos de entrada y salida; debido a que la mayoría de los dispositivos que albergarán los sistemas de archivos constituyen dispositivos de E/S orientados al almacenamiento masivo de información, esto resulta lógico. En este caso, el sistema de archivos se posiciona entre las aplicaciones y los controladores de dispositivos, generando la API estándar de utilización e implementando las operaciones para el almacenamiento de la información en los diversos dispositivos, dejando a la administración de los dispositivos de E/S la parte de la realización de las operaciones puntuales en los dispositivos.

Además, muchos sistemas de archivos cuentan con una serie de programas para dar mantenimiento al sistema, algunos de los cuales pueden ser iniciados al comenzar el uso del sistema de archivos. Esta serie es comúnmente conocida como **montaje del sistema de archivos** y su objetivo principal es la realización de actividades de mantenimiento de manera continua, como procesos de baja prioridad.

Organización lógica

Tipos de archivos

A continuación definimos todos aquellos tipos de archivos que tienen diferentes implementaciones entre sí, como los mecanismos de operación, pero que se representan como archivos.

Esto hace que la aplicación con la que se trabaja la información contenida no sea relevante (y por tanto tampoco la extensión) y que prestemos más atención a la mecánica de la implementación de cada uno.

Bajo este criterio, los tipos de archivos son:

- **Archivos planos.** Contienen información generada por una aplicación, la cual será recuperada después por esta misma u otras aplicaciones que implementen los mecanismos para interpretar el código en el que se guardó originalmente la información. Pueden contener código ejecutable, código fuente, datos organizados en

registros de estructura continua y longitud constante o de estructura variable y longitud variable, pero se distinguen porque las aplicaciones requerirán mecanismos para almacenar información en el archivo, la cual recuperarán después de la misma forma en que fue almacenada. En el caso de este tipo de archivos, son las aplicaciones mismas las que determinan el formato en el que la información será representada.

Aunque en algunos sistemas de archivos se soportan múltiples subtipos de archivos planos con diversas facilidades para el acceso a la información, su manejo puede ser agrupado, ya que todos estos tipos de archivos siguen almacenando información de las aplicaciones en dispositivos de almacenamiento masivo.

- **Directorios.** Contenedores de archivos que se usan para construir una organización jerárquica en el sistema de archivos. Una vez constituido el directorio, las aplicaciones pueden modificar los archivos para incluirlos o retirarlos del contenedor; sin embargo, el formato y el contenido del directorio es determinado por el sistema de archivos y no por las aplicaciones.
- **Archivos mapeados a dispositivos síncronos.** Como se trató en el capítulo 5, *Administración de los dispositivos de entrada y salida*, los dispositivos de E/S se representan con archivos. En este caso, la implementación de las operaciones de lectura y escritura deben derivar en la funcionalidad del software que controla el dispositivo. Así, el resultado de las operaciones de lectura y escritura serán definidas por la funcionalidad del dispositivo en cuestión. Las operaciones específicas que se pueden realizar con los dispositivos de E/S síncronos y que transfieren la información palabra por palabra hacen necesario que se genere un tipo determinado de archivo que reconozca las solicitudes de dichas operaciones y las encamine hacia los dispositivos apropiados.
- **Archivos mapeados a dispositivos asíncronos.** Archivos implementados por dispositivos de E/S. Los dispositivos asíncronos que transfieren la información en bloques son la forma genérica en que se puede acceder a cualquier dispositivo de E/S, y son los responsables de implementar el conjunto de operaciones asíncronas y de bloques correspondientes.

Importante

♥ **Docente.** A la clasificación del contenido de un archivo según las aplicaciones que pueden procesarlo, e identificado por la parte final de su nombre, conocida como "extensión", también se le conoce como **tipo de archivo**. Este concepto es relevante para la capacitación de los usuarios finales, quienes deben conocer las convenciones de nombres de archivos, con el fin de identificar las aplicaciones correctas para procesarlos, por lo que constituye la forma más popular en la bibliografía. Como esta acepción de tipo de archivo no es relevante para los mecanismos que el sistema operativo usa para manejar la información contenida en estos, tampoco resulta relevante para nuestro estudio.

Atributos de archivos

Con el fin de permitir una correcta administración de los archivos, resulta necesario mantener un conjunto de metadatos que dependen de la implementación del sistema de archivos; no obstante, hay diversos elementos comunes que vale la pena revisar.

- **Nombre del archivo.** Aunque en algunos sistemas de archivos el nombre por el cual el usuario identifica al archivo es una característica del directorio que hace referencia a este y no del archivo en sí mismo, es necesario asociar los archivos a nombres simbólicos, con el objetivo de que los usuarios y las aplicaciones puedan referirse a estos para usarlos.
- **Identificación.** Siempre debe proporcionarse algún dato que identifique al archivo de forma biunívoca.
- **Ubicación del archivo.** En todos los casos, siempre debe señalarse al menos dónde inicia el archivo e idealmente dónde se encuentran almacenadas todas las partes de este para facilitar el acceso aleatorio.
- **Tamaño real.** En todo momento es conveniente tener un registro del número de bytes de datos almacenados hasta entonces en el archivo.
- **Tamaño máximo.** Como los datos se escriben en bloques que deben asignarse completos a un archivo, es muy probable que parte de la capacidad de los bloques asignados al archivo (específicamente al último) no se haya ocupado en su totalidad, por lo que se debe registrar la capacidad de almacenamiento total reservada para el archivo, la cual será igual o excederá al tamaño real de los datos contenidos.
- **Registro de la marca de tiempo en que el archivo se creó, modificó y abrió por última vez.** Todos los archivos suelen guardar copias del registro **time**, el cual permite conocer la fecha y hora (también conocida como marca de tiempo) en que se realizaron dichas acciones por última vez en el archivo.
- **Información de control de acceso.** De acuerdo con la implementación de cada sistema de archivos, en general se requiere información para determinar cuáles usuarios tienen autorizado realizar determinadas operaciones en cada archivo.
- **Usuario creador del archivo.** Registro que identifica al usuario que originalmente crea el archivo.
- **Usuario dueño del archivo.** Originalmente, el dueño del archivo es el usuario que tiene la capacidad de modificar los privilegios sobre el archivo sin necesidad de tener privilegios de administrador.
- **Usuarios con privilegios específicos.** Algunos sistemas de archivos permiten especificar ciertos permisos para usuarios individuales adicionales al dueño del archivo.

En la sección de protección de archivos de este capítulo se revisan diversos esquemas de permisos.

- **Grupo(s) con privilegios específicos.** Además de los privilegios que se especifiquen para usuarios individuales, el archivo puede tener determinados permisos asociados a uno o varios grupos, dependiendo del sistema de archivos que se trate.
- **Banderas que indican características del archivo.** Las banderas son bits de información que permiten determinar si el archivo tiene o no una característica particular.
- **Solo lectura.** El contenido del archivo no debe ser modificado.
- **Sistema.** Utilizado por el sistema operativo.
- **Escondido.** Archivo que normalmente no se presenta en el contenido del directorio.
- **Archivo.** Se enciende cuando se modifica la información y se apaga cuando esta se respalda. Es usado por aplicaciones de respaldos para llevar el control de los archivos que deben ser incluidos en respaldos incrementales.
- **Temporal.** Se usa para destacar aquellos archivos que almacenan información pertinente solo para los procesos activos y que no deberían estar presentes luego de reiniciar el sistema.
- **Protegido.** Se utilizan para controlar el acceso concurrente.

Sistemas de archivos jerárquicos

En un principio, cuando la capacidad de almacenamiento de los dispositivos era muy limitada, a menudo solo era posible almacenar unos cuantos archivos en el dispositivo, por lo que el listado era corto y no se requería tener una estructura de orden para su almacenamiento; por esa razón, muchos sistemas de archivos para este tipo de dispositivos solo contemplaban un directorio por dispositivo, donde solo era posible almacenar un máximo de 256 archivos. Pero conforme la capacidad de los dispositivos creció, pronto se detectó que mantener un único contenedor no era conveniente para un número grande de archivos, por lo que se buscó que se pudieran crear contenedores en un segundo nivel y de manera eventual crear estructuras de directorios y subdirectorios en un árbol de profundidad definida solo por las limitaciones de almacenamiento del dispositivo.

A las organizaciones donde los directorios pueden contener directorios comunes hoy día se les conoce como sistemas de archivos jerárquicos.

Para lograr representar estas estructuras basta con considerar en la implementación que los directorios son en sí mismos archivos, por lo que tienen la capacidad de generar listados de directorios.

Cuando los archivos registran sus atributos de manera independiente de su registro en los directorios, se puede agregar un archivo en múltiples directorios sin generar un nivel de redundancia que perjudique el rendimiento del sistema; así, en sistemas como los basados en nodos-i tenemos lo que se conoce como **ligas**, que pueden ser duras o suaves y que se definen a continuación.

- **Ligas duras.** Incluyen un archivo en un directorio; ligan el archivo al directorio y para ello le asignan un nombre. Se considera que estas ligas identifican al archivo, por lo que si este se elimina del directorio, se buscan todas las demás ligas duras que se refieran al archivo para eliminarlas también.
- **Ligas suaves.** Incluyen el archivo en el directorio y le asignan un nombre, pero no se considera que identifiquen al archivo, por lo que si la liga es eliminada del directorio el archivo no se verá afectado. Una liga suave en un directorio puede señalar a un archivo que ya no existe o no se encuentre en la ruta donde se encontraba originalmente al crear la liga suave.

En sistemas operativos donde los atributos están ligados al directorio, y donde incluir un archivo en varios directorios resulta impráctico, se suele usar **ligas simbólicas**, que son pequeños archivos (usualmente de texto) en los que se especifica la ruta en la que se puede encontrar el archivo que se está ligando y se deja a cargo de las aplicaciones la capacidad de interpretarlos para acceder a dichos archivos. Esta mecánica proporciona un máximo de flexibilidad, pero es menos eficiente que los otros tipos de ligas.

Directorio raíz

Un directorio que merece especial atención es el directorio de donde parten todos los demás elementos del sistema de archivos, llamado directorio raíz, en todas las variaciones de UNIX. El sistema operativo genera un sistema virtual de archivos en el que se tiene un único directorio raíz, dentro del cual se incluyen o montan, en subdirectorios, todos los contenidos de los sistemas de archivos que se utilizarán. En el caso de Windows, hay un directorio raíz por cada dispositivo, y todos estos se identifican con letras.

Todos los archivos pueden ubicarse indicando la ruta absoluta, que es el conjunto ordenado de todos los directorios y subdirectorios que se deben seguir para llegar al archivo en cuestión.

En UNIX las rutas absolutas suelen ser de la forma mostrada en el diagrama de la figura 6.1.

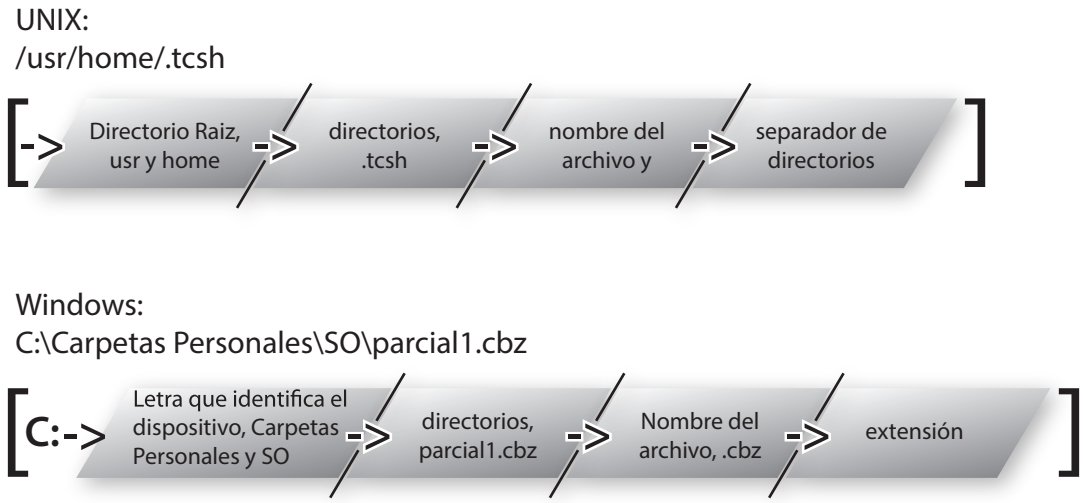


Figura 6.1 Estructura de rutas de archivos.

En los sistemas operativos en los que se construye un directorio virtual con todos los sistemas de archivos montados, se puede mejorar la protección para que una determinada aplicación no pueda acceder a los archivos, limitando el alcance que el proceso tiene a un subdirectorio y trasladando el directorio raíz al directorio donde comienza la información que el proceso requiere utilizar. Esto se conoce como cambio de directorio raíz o `chroot`.

Lo anterior es habitual en procesos que reciben peticiones directamente de los usuarios o de otros sistemas y se hace con el fin de que, aunque el proceso tenga una vulnerabilidad que termine siendo explotada en un intento por poseer información privada del sistema, el proceso no logre acceder a los datos protegidos debido a que se encuentra fuera del sistema de archivos que puede ver. Por ejemplo, los procesos que han de atender peticiones HTTP habitualmente trasladan la raíz del sistema de archivos a un directorio donde colocan solo su configuración en los acervos a entregar a los usuarios; de esta manera, si algún ataque logra obligar al proceso a entregar archivos que no necesariamente se quería publicar, el resto de los archivos del sistema estarán a salvo porque no se encuentran bajo el directorio raíz del proceso y, por tanto, no son accesibles a él.

Directorio actual

Otro directorio que desempeña una función especial para los procesos es el directorio activo o actual, el cual será asociado a cada instancia de ejecución de procesos y servirá de base para interpretar rutas relativas. Estas son rutas de archivos que en lugar de des-

cribir la ubicación de un archivo a partir del directorio raíz, describen la ubicación a partir del directorio actual usando símbolos como "." para el directorio actual, ".." para el directorio que contiene al directorio actual y los nombres de los directorios para explorar sus contenidos.

Para cambiar el directorio actual, la mayoría de los sistemas operativos proporcionan instrucciones como "cd" para indicar el nuevo directorio que un proceso (por ejemplo, una sesión de usuario) deberá usar como directorio actual. Los procesos hijos suelen iniciar en el mismo directorio actual que tenía el proceso padre al iniciar el proceso.

6.4 Organización física de archivos

A continuación revisaremos la forma en que la información almacenada en los dispositivos de almacenamiento secundario se organiza para soportar una recuperación eficiente de los contenidos.

La manera de organizar la información se basa en las características y limitaciones de los dispositivos de almacenamiento masivo disponibles y se conservan muchos conceptos para anexar los dispositivos novedosos, aunque tengan características muy diferentes a los esquemas existentes, los cuales de todas formas a menudo aportan funcionalidad valiosa.

Cintas magnéticas

Uno de los primeros medios de almacenamiento masivo fueron las cintas magnéticas. Cuando se comenzaron a usar en cómputo ya llevaban varios años empleándose para grabación de audio, lo que facilitó su adopción ya que muchos de los componentes electrónicos, mecánicos y las propias cintas se fabricaban a gran escala y fueron antecedentes importantes de los equipos dedicados a almacenar datos (véase figura 6.2).

Figura 6.2 Carrete de cinta magnética.

Para representar información binaria en una cinta magnética se han usado diversos sistemas, pero en principio la cinta, con una superficie que es capaz de conservar una polaridad magnética, debe pasar sobre una cabeza de lectura con inductores que traducen los cambios de polaridad en la cinta en señales eléctricas

que son amplificadas y traducidas en datos binarios para aplicaciones informáticas o señales analógicas en aplicaciones de audio y video.

Este mecanismo puede perder la sincronicidad de la cinta con la información, que se genera por pequeñas variaciones en la velocidad de dicha cinta, y por ello requiere de mecanismos para afinarla contra un oscilador de frecuencia conocida. Esto se puede lograr con circuitos similares a los que se usan para sintonizar frecuencia modulada en transmisiones radiales, pero para que estos funcionen de manera correcta se requieren cambios frecuentes en las polaridades de la cinta. Para evitar que cadenas de ceros o de unos a representar generen tramos largos sin cambios de polaridad es necesario transformar la información mediante operaciones matemáticas a fin de garantizar que habrá frecuentes cambios de polaridad en la superficie.

Además, los segmentos adyacentes de polaridades contrarias en las cintas no dejan de afectarse entre sí. A lo largo del tiempo, a temperatura ambiente y con los materiales disponibles, las polaridades contrarias tienden a cancelarse entre sí, haciendo más débil las señales recuperadas y generando errores en la lectura.

Para detectar y corregir estos errores se añadieron mecanismos de codificación adicionales, que usan información redundante sobre conjuntos de longitud constante de información. Estos conjuntos de longitud constante se conocen como **sectores**, los cuales, al separarse a lo largo de las cintas, con segmentos señalados por patrones específicos de cambios de secuencias, también permiten identificar el número de sectores transcurridos al pasar la cinta a alta velocidad, lo que facilita llegar a secciones de la cinta sin necesidad de leer toda la información desde el inicio.

Sin embargo, es necesario recorrer las cintas de forma secuencial, incluso si se cuenta con la funcionalidad de avanzarla y retrocederla a una velocidad mayor a la de lectura de la información.

ACTIVIDAD PROPUESTA

En acción

Como las operaciones de lectura y escritura a menudo no se realizaban en una sola secuencia ininterrumpida, las unidades necesitaban detener el paso de la cinta y luego reiniciarlo para escribir el sector siguiente, lo que implicaba someter la cinta a mayor esfuerzo. Para resolver esto, muchas cintas usaban columnas de vacío como resortes para limitar la tensión.

En equipo consulten el video de una unidad temprana de cinta basada en columnas de vacío: https://www.youtube.com/watch?v=5hSaMLqvr_g. Elaboren una tabla con las ventajas y desventajas de este proceso y compártanlo con sus compañeros.

La codificación de los datos para escribirlos en el medio magnético, así como los sectores y la redundancia para detección y corrección de errores, son técnicas que se iniciaron en las cintas, pero que se han conservado en todos los medios posteriores de almacenamiento masivo y, por tanto, es importante tenerlos en mente.

Discos magnéticos

Para solventar la limitación de la lectura secuencial de las cintas se generaron los discos magnéticos; en estos, el material donde se graba la información se encuentra en una superficie, que suele ser giratoria y puede ser alcanzada por cabezas de lectura capaces de moverse sobre ella en dos ejes y no solo de manera longitudinal. Los primeros dispositivos usaban una superficie cilíndrica, pero los modelos comerciales exitosos usaban una serie de platos circulares con cabezas de lectura montadas en brazos que se desplazaban para trazar círculos concéntricos en diversas longitudes de radio (véase figura 6.3).

Tiempo después se incrementó la densidad de cristales ferromagnéticos en la superficie, lo que ha permitido almacenar más información por centímetro cuadrado; además, los detalles de la construcción y el aprovechamiento de la densidad del material siguen mejorando a fin de permitir unidades con capacidades cada vez mayores. Para que las cabezas magnéticas logren detectar estas densidades de información necesitan estar a muy poca distancia de la superficie, y para conseguirlo emplean las diferencias de presión del aire a diversas distancias de la superficie logrando una separación estable a unos pocos nanómetros de distancia, sostenidas por la presión de las moléculas de aire aceleradas por la superficie y empujadas con suavidad hacia la superficie por los brazos que las sostienen. Esto hace que los dispositivos sean vulnerables a la vibración y que las cabezas deban ser muy ligeras.

Figura 6.3 Modelo comercial de IBM de disco duro conocido como Direct Access Storage Facility.

La organización de la información en los discos también se modifica. Aunque los sectores se conservan, estos ya no se colocan en una sola secuencia a lo largo de la cinta sino en una serie de secuencias en los diversos círculos trazados por la cabeza inmóvil sobre la superficie que gira del disco. Cada uno de estos círculos se conoce como pista o *track.* Hoy día, la mayoría de los discos duros usan las dos caras de una serie de platos, y resulta más lento mover el conjunto de cabezas de lectura que dejar girar el disco.

Usualmente, el conjunto de pistas sobre las que pasan las cabezas para una posición determinada del peine se conoce como cilindro. Se usan cilindros completos para almacenar tantos bloques como cabezas se tengan. En un principio, los dispositivos presentaban menos bloques en los cilindros cercanos al centro de los discos, que son de una longitud mucho menor. Pero con la creciente potencia de los microcontroladores incluidos en el dispositivo, la capacidad de sus *búferes* y el nivel de funcionalidad presente en sus sistemas internos de control, los discos actuales simulan para las controladoras de discos que todos los cilindros son de la misma capacidad e incluso analizan su propio desempeño para detectar, corregir y prevenir errores de forma estandarizada con la tecnología S.M.A.R.T. incluida en la norma de discos duros ATA (véase figura 6.4).

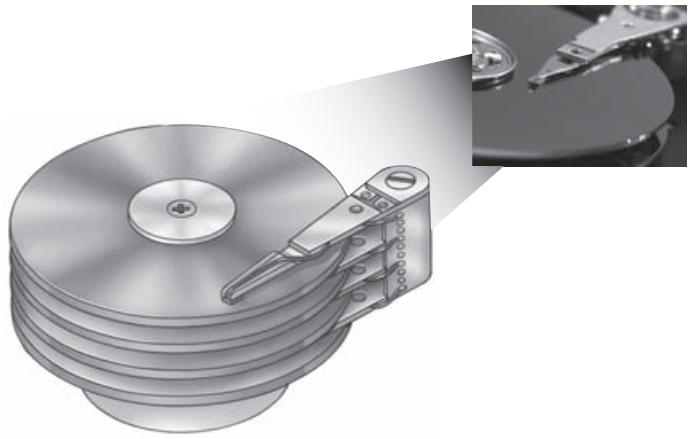


Figura 6.4 Peine de cabezas de lectura en un disco duro.

Los bloques de un mismo archivo tienden a dispersarse a lo largo de la superficie del disco conforme se van reservando más bloques, eliminando algunos archivos y reutilizando los bloques así liberados para almacenar información nueva. Como los bloques ya no estarán en secuencia en la superficie, se necesita generar listas doblemente ligadas con los bloques de modo que sea factible realizar las lecturas secuenciales de la información.

Para aprovechar el acceso aleatorio a los bloques de discos se requiere conocer el conjunto de bloques que constituye un archivo, de preferencia sin tener que leerlo en su totalidad. Para ello debemos construir un listado de todos los bloques del archivo que se pueda leer al iniciar las operaciones sobre el archivo, de modo que cuando se busque una ubicación aleatoria en este, el sistema de archivos pueda identificar muy rápido el bloque en el que se encuentren esos datos y sea capaz de recuperarlo con un mínimo de operaciones de disco. Diversos sistemas de archivos generan representaciones distintas de estas tablas de localización de los bloques, y estas consideran una de sus diferencias principales.

Discos ópticos

A partir de la funcionalidad y de las normas generadas para los discos duros (como Serial ATA), los medios de almacenamiento óptico removible pueden exponer la infor-

mación que almacenan de forma muy parecida a los controladores de dispositivos de almacenamiento a la de los discos duros. Sin embargo, la mecánica con que operan tiene diversas diferencias importantes.

El principio de operación de los discos compactos (CD), discos compactos grabables (CD-R) y reescribibles (CD-RW), discos de video digital (DVD) y blue ray (BR) en sus diversas versiones es muy similar y varían primordialmente en el mecanismo de escritura y las características ópticas de los medios.

La forma como representan la información es a través de cambios en el índice de refracción de la luz en una superficie del medio, mediante dispositivos ópticos que enfocan un haz de luz láser en un punto muy pequeño de la superficie, prácticamente de una sola longitud de onda de ancho, la cual, al desplazarse, traza una trayectoria en espiral partiendo del centro del disco hacia la parte exterior. El mecanismo no tiene contacto con la superficie y las características del rayo de luz reflejado en la superficie es todo lo que utiliza para seguir la pista, mantener el haz enfocado y detectar en la superficie las zonas que reflejan más o menos luz en fase con el rayo que se emitió originalmente; con esto se pueden diferenciar las zonas que reflejan mejor la luz (valles), de las que no la reflejan o lo hacen fuera de fase, y que se conocen como pozos. Los CD originales usaban auténticos pozos en la superficie del disco de un cuarto de longitud de onda para hacer que el rayo reflejado saliera de fase con el emitido, lo que al reunirse con el haz de referencia daba una intensidad mínima en el sensor. Medios grabables usan tintes y materiales que pueden ser alterados por un láser de alta potencia para cambiar la cantidad o polaridad de la luz que reflejan sin modificarse con las emisiones de baja potencia para lectura. Los tintes usados en los medios regrabables pueden incluso regresar a su estado original al ser afectados por rayos de una potencia específica.

ACTIVIDAD PROPUESTA

En acción

Consulta el video promocional para los LaserDisc, que constituye una descripción detallada y sencilla del funcionamiento de los dispositivos de almacenamiento óptico. Este video lo puedes encontrar en:

https://www.youtube.com/watch?v=5Y8zodehLVE

Ahora con tus propias palabras describe el funcionamiento de los dispositivos de almacenamiento.

Es importante destacar que a pesar de que la operación de un medio óptico sea radicalmente diferente de la de un medio magnético, ambas tienen algunas restriccio-

nes similares. Para que los mecanismos que siguen la pista en el disco se mantengan en ella, debe haber una frecuente alternancia de pozos y valles, por lo que se requiere también de algoritmos que transformen la información que habrá de grabarse garantizando que no se presenten largas secuencias de pozo o de valle ininterrumpidas. Los algoritmos usados en los CD y demás medios ópticos son estándar y están optimizados para las características del medio, por lo que no son los mismos que se emplean para los medios magnéticos. Como las primeras aplicaciones comerciales de estos medios fueron para audio y video, el tamaño del sector está optimizado para almacenar celdas de archivos codificados según la norma de MPEG y algoritmos relacionados de compresión de audio y video.

En las aplicaciones de datos, retomamos los formatos de bajo nivel para obtener sectores de datos en los que es factible almacenar información de archivos. Los medios ópticos tienen sus propias normas sobre la organización de información escrita en momentos diferentes, que son requeridas porque, a diferencia de los medios magnéticos, las series de pozos y valles no deben ser interrumpidas durante la grabación ya que el medio no cuenta con una estructura preexistente, como los discos magnéticos. Por esta razón, las diversas sesiones de grabado constituyen cintas diferentes en la superficie que se conocen como "sesiones" y que se aproximan al concepto de particiones. Para permitir que los dispositivos se utilicen con los mismos mecanismos que otros medios de almacenamiento, las interfaces ATA de las unidades ópticas ocultan esta funcionalidad del BIOS y, por tanto, de las aplicaciones; sin embargo, esto implica que los distintos medios están limitados en la forma en que representan la información en el nivel físico y solo pueden usar los mecanismos contemplados en las normas de construcción de las unidades definidas por la industria.

Discos RAM

En ocasiones se tienen archivos que se requiere utilizar con mucha frecuencia y que resultan críticos para el desempeño del sistema. Cuando hay una presión muy grande para optimizar un sistema con este tipo de archivos, que en ocasiones están dando soporte a bases de datos, es común que se busque implementar los sistemas de archivos que los contienen en dispositivos lo más rápidos que sea posible.

Siguiendo esa tendencia, si se configura una computadora con una gran cantidad de memoria RAM y se genera un dispositivo de almacenamiento masivo virtual que use un segmento de memoria para almacenar la información, que de otra forma iría directamente a un dispositivo de almacenamiento secundario, se puede crear un sistema de archivos con un desempeño muy superior a lo que sería posible para un disco

duro o incluso para una unidad flash, ya que la RAM es mucho más rápida que estos dispositivos, aunque tendrá la desventaja de que, al menos en principio, ese sistema de archivos tendrá que volverse a crear cada vez que se inicie el equipo.

En varias distribuciones de Linux, como Ubuntu, se tiene un disco RAM conocido como `tmpfs`, el cual permite que la información se mueva al disco duro como memoria virtual, recuperando algo de memoria para las aplicaciones, a costa de disminuir un poco el desempeño. Se suele montar bajo `/mnt/tmpfs` y, mediante la configuración en el montaje, se puede atender solo a usuarios con permisos de administración o a algún grupo o usuario específico, y es posible controlar el tamaño máximo del sistema de archivos que contenga. Una implementación anterior es `ramfs`. En este no se puede limitar el tamaño del sistema de archivos, por lo que si las aplicaciones que lo usan no implementan algún otro mecanismo de control y continúan guardando información en él, saturarán la memoria disponible y dejarán a la máquina en la imposibilidad de continuar la operación. Esta tampoco permite que los datos pasen a almacenamiento secundario, por lo que tiene un rendimiento superior en términos de velocidad y se suele montar en `/mnt/ramfs`.

En Windows se tienen productos comerciales que pueden generar dispositivos virtuales que alberguen en RAM los discos que simulan. Algunas de las características que agregan es el respaldo automático de la información al disco duro cuando se detiene el sistema, para dar persistencia a la información y el soporte a particionado del dispositivo virtual.

Memorias flash

La memoria flash se basa en un tipo especial de transistores que conservan una carga eléctrica en una base adicional (conocida como floating gate), lo que altera el voltaje de entrada que requieren para pasar a su estado conductor. Para almacenar información se colocan varios de estos transistores en serie y se alimenta un voltaje que ponga en saturación a todos los transistores de la serie, menos a uno que recibe un voltaje intermedio; mediante dicho voltaje intermedio se puede detectar el estado de la carga que tiene el transistor en esa posición. Se colocan varias series de transistores en paralelo para construir una palabra y así generar bloques de palabras, donde el número de las mismas se determina por la cantidad de transistores que se incluyen en cada serie, y la longitud de la palabra por el número de series que se coloquen en paralelo.

Las primeras memorias flash que se comercializaron usaban solo dos estados para la carga de cada transistor, con lo que se almacenaba un bit; estas memorias se cono-

cen como SLC (Single Level Cell). Después se desarrolló la tecnología para detectar varios niveles de carga y, por tanto, varios bits por transistor. Esta tecnología se conoce como MLC (Multi Level Cell) y presenta mayor sensibilidad a fugas de la carga eléctrica y al ruido electromagnético, pero permite mayor densidad de almacenamiento de información.

Otro aspecto importante a considerar de las memorias flash es que para inducir la carga en el floating gate se requiere aplicar una corriente importante en la base, y para eliminar la carga antes de reescribirlos también se necesita aplicar una corriente importante. Estos ciclos fatigan de manera progresiva los materiales del transistor y van acumulando electrones de carga que no se eliminan de manera correcta, lo que termina por inutilizar el transistor. Los circuitos SLC suelen tener vidas útiles del orden de 100 000 ciclos de escritura y borrado y los MLC del orden de 10 000 ciclos.

Las operaciones de lectura no se realizan sobre las palabras de memoria individuales sino sobre conjuntos de palabras llamadas páginas, que contienen un múltiplo de 2 KB de palabras, además de información adicional para corrección de errores y para almacenar metadatos.

Para las operaciones de escritura se definen conjuntos aún más grandes que reúnen múltiples páginas denominadas bloques. Las operaciones de borrado, que eliminan la carga de los transistores, solo pueden realizarse a nivel de bloque. Por su parte, las operaciones de escritura se desarrollan en página completa.

Los bloques se agrupan en planos (planes) y estos en unidades lógicas (LUN – Logical Unit). Las direcciones físicas en las memorias flash se construyen entonces con los cuatro elementos mostrados en el diagrama de la figura 6.5.

La mayoría de los dispositivos que usan las computadoras contienen circuitos controladores de memoria capaces de traducir las direcciones lógicas (adecuadas a las características de los sistemas de archivos) por lo regular a las direcciones físicas del dispositivo de forma transparente para el sistema operativo. En el proceso de traducción de las direcciones, a la lectura y escritura de la información se suelen añadir algunos de los siguientes servicios (en ocasiones todos), pues son necesarios si se desea usar el dispositivo con un sistema de archivos de propósito general como FAT el que se emplea en Windows:

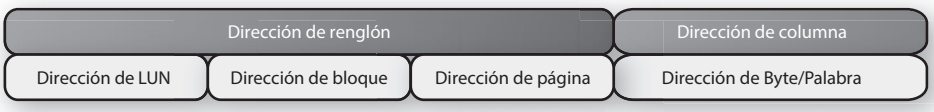


Figura 6.5 Estructura de la dirección física en una memoria flash.

- **Corrección de errores.** Debido a la sensibilidad en relación con fugas de carga y fatiga de los materiales, es indispensable incluir, incluso desde la manufactura, información redundante para tolerar las fallas y mantener el dispositivo en operación, lo que es de especial importancia con dispositivos MLC. Hoy día se prefiere el uso de algoritmos como Reed-Salomon y BCH (Bose, Ray-Chaudhuri and Hocquenghem), por lo regular con apoyo del hardware del dispositivo flash. Sin embargo, algunos dispositivos de bajo costo podrían delegar estas tareas a la CPU de la máquina a la que se conecte el dispositivo.
- **Control de bloques malos y balanceo del desgaste.** Para limitar los costos de fabricación se permite que un número reducido de circuitos de memoria defectuosos estén presentes en los dispositivos que se comercializan. Cuando estos circuitos defectuosos en un solo bloque exceden la capacidad del algoritmo de corrección de errores, el bloque se marca como defectuoso y no se emplea en la operación. Cada bloque en la memoria flash deberá registrar si se trata de un bloque defectuoso. Esa información se copia en una tabla de bloques defectuosos por dispositivo. Para que la fatiga de los materiales y la acumulación de carga no incrementen muy rápido el número de bloques defectuosos si los sistemas de archivos concentran sus operaciones de escritura en determinadas direcciones (lo que sería típico en sistemas populares como FAT), se implementan algoritmos para distribuir las modificaciones entre los bloques del dispositivo.

  Estos algoritmos suelen seguir dos estrategias determinadas, conocidas como estática y dinámica. Los dinámicos solo balancean la carga sobre los bloques que están siendo modificados con frecuencia; en cambio, los dinámicos abarcan todos los bloques de la memoria para repartir la carga de trabajo. Para lo anterior se incluye en los metadatos de cada bloque un contador con el número de ciclos de borrado que ha pasado. Cuando este número pasa de ciertos umbrales predefinidos, la información del bloque se traslada a otra ubicación con menor carga de trabajo acumulada, con lo que se logra incrementar la vida útil del dispositivo en su conjunto a costa de un consumo adicional de procesamiento en la relocalización de los bloques.
- **Recolección de basura.** Conforme se realizan las operaciones de lectura y escritura, la información de algunas páginas queda invalidada sin que se borren los bloques de inmediato para evitar desgaste. Estas páginas se escriben en otros bloques que el dispositivo usa como *caché* de cambios, lo que genera fragmentación en la memoria. La recolección de basura revisa con periodicidad la memoria, usualmente cuando se detecta que quedan pocos bloques libres, y reúne las páginas fragmentadas de los bloques liberando estos con páginas invalidadas y borrándolos a fin de que queden listos para recibir nueva información.

Algunos sistemas de archivos especializados en dispositivos de almacenamiento flash pueden ocupar dispositivos que no cuenten con todos estos servicios implementados en el controlador incluido en el dispositivo. Ejemplos de estos son JFFS2 y Safe-FLASH.

6.5 Mecanismos de acceso a los archivos

Considerando la variedad de implementaciones de los dispositivos de entrada y salida, es responsabilidad de los sistemas de archivos proporcionar mecanismos que logren cumplir sus objetivos, para lo cual se requiere implementar una serie de mecanismos que revisaremos a continuación.

Tablas de ubicación de archivos

Uno de los principales retos para dar un buen desempeño al utilizar medios es encontrar las ubicaciones de elementos específicos de un archivo. Al respecto, los medios magnéticos de almacenamiento a menudo son el punto de referencia debido a la importancia que tienen hoy en día.

En principio tenemos listas doblemente ligadas de los bloques que constituyen los archivos, las que junto con referencias en los directorios del inicio del archivo nos permitirían leer el archivo de manera secuencial; sin embargo, esto no es muy conveniente para recuperar solo algunos datos de archivos muy grandes, ya que para recuperar un dato en particular, en promedio necesitaríamos leer la mitad del archivo.

Una forma de acelerar la recuperación de la información que funciona para datos de muchos tipos es generar una tabla con las ubicaciones de todos los bloques del archivo; así, cuando sepamos cuál bloque del archivo queremos recuperar, solo tendremos que buscarlo en la tabla para obtener de nuevo la dirección, lo que reduciría en gran medida las operaciones de lectura, esto debido a que la tabla podrá almacenar muchas direcciones de bloques en poco espacio.

FAT

Una de las implementaciones más populares de las tablas de ubicación de archivos, debido a la popularidad de los S.O. de Microsoft, es la File Allocation Table (FAT) en sus implementaciones de 16 y de 32 bits usadas en las diversas versiones de Windows.

La tabla con las ubicaciones de los bloques de todos los archivos se coloca al principio de la partición, es decir, en los primeros bloques que usará en el dispositivo el sistema de archivos, después del MBR (Master Boot Record). Como FAT busca soportar sistemas de archivos muy grandes, toma precauciones para minimizar el tamaño de la propia FAT, de modo que la capacidad de almacenamiento del dispositivo pueda aprovecharse en información de las aplicaciones y para que la lectura de la FAT tarde lo menos posible.

Para minimizar su tamaño, la FAT identifica los registros de la tabla por posición, con lo que omite poner en el registro el número de bloque al que corresponde. Después tiene un código que puede representar, según su valor, el número del bloque siguiente, o un código que indica si el bloque está defectuoso o libre. Los directorios señalan al primer bloque del archivo, con lo que solo se requiere leer la FAT para determinar la ubicación de todos los bloques del disco. Esta información se carga en memoria cuando se abre un archivo, y se actualiza de manera periódica o después de que las aplicaciones cierran el archivo.

Una vez que se ha concentrado la información de todos los archivos en una sola tabla, cualquier error al escribirla podría dejar inutilizado todo el sistema de archivos. Para reducir la posibilidad de un error catastrófico, la FAT se almacena por duplicado y se vigila que se haya terminado de escribir la primera copia antes de escribir la segunda. Si se detecta un error, se verifican las dos copias, y en caso de que los códigos de corrección de errores de una de ellas verifiquen que su información está íntegra, se reescribe la copia dañada con la que no tenía errores. Esto puede hacer que algunas modificaciones en los archivos no se reflejen a la FAT, pero una revisión del archivo puede reconstruir su lista de bloques para actualizar la FAT.

En caso de que ninguna de las copias de la FAT esté correcta, se puede revisar toda la superficie del disco para detectar, mediante los directorios y las listas doblemente ligadas de bloques, cuáles son los archivos presentes. Sin embargo, este proceso implica mucho tiempo, puede corromper algunos archivos y tiende a detectar series de bloques de archivos que se eliminaron en el pasado como potenciales archivos, a los que se les conoce como cadenas perdidas y que son muy difíciles de aprovechar.

Nodos-i

Creados con los sistemas UNIX y empleados en diversos sistemas de archivos, los nodos-i (por Index Node o Information Node) son bloques que se reservan para almacenar metadatos de los archivos, primordialmente sus atributos y una serie de direcciones de bloques de datos del archivo.

Como los nodos-i solo ocupan un bloque, el número de bloques que pueden señalar es limitado. En algunos sistemas, como EXT3, esto establece un tamaño máximo del archivo; en otros se utiliza una serie de nodos-i secundarios dedicados a almacenar solo referencias de bloques para completar el listado de todos los bloques de un archivo de cualquier tamaño.

Los directorios solamente contienen listados de referencias a los nodos-i principales de los archivos asociándolos a un nombre propio del directorio. Esto facilita que los nodos-i puedan incluirse en múltiples directorios mediante ligas.

Cuando se abre un archivo, los nodos-i son cargados a memoria y en ella se lleva el registro de todas las modificaciones necesarias por cambios al archivo. Al igual que FAT, el sistema de archivos los escribe de manera periódica o cuando se cierra el archivo.

Gracias a que la información de diversos archivos se almacena por separado, los sistemas de nodos-i son menos sensibles a fallos o interrupciones en la operación, ya que un fallo solo afecta a los archivos que se encuentran en proceso de ser modificados.

Como el número de nodos-i indirectos crece de manera proporcional al número de bloques de archivos muy grandes, a menudo se construyen árboles de dos o tres niveles, los cuales se conocen como sistemas de nodos-i doble y triple indirectos, y se ha incursionado en establecer árboles binarios de nodos-i secundarios para tener una velocidad cercana al óptimo en sistemas de archivos, por ejemplo, Reiser4, EXT4 y NTFS (en sus directorios).

6.6 Seguridad en sistemas de archivos

Es muy importante para la seguridad informática salvaguardar que la información almacenada se mantenga íntegra y que solo sea utilizada por los usuarios y aplicaciones apropiados; por tanto, los sistemas de archivos deben considerar en su diseño las características necesarias para satisfacer dichas necesidades.

En la sección siguiente revisaremos con más detalle los mecanismos de protección de los sistemas de archivos que se emplean para solventar una serie de necesidades, entre las que se encuentran vigilar que los usuarios y sus procesos utilicen solo la parte del sistema de archivos para la que tienen autorización, supervisar que la concurrencia no ponga en peligro la información y cuidar que los programas almacenados en el sistema de archivos no sean corrompidos o ejecutados en situaciones inadecuadas.

Como ya comentamos con anterioridad, salvaguardar la integridad de la información almacenada es uno de los objetivos fundamentales de los sistemas de archivos, por lo que solo nos resta revisar las medidas que deben tomarse para proteger la

información de los errores causados por los propios usuarios; por lo regular, esto se consigue mediante mecanismos de respaldo y recuperación en caso de falla.

Mecanismos de protección

El control de acceso que los sistemas de archivos deben soportar ha cambiado mucho a partir de la masificación de Internet, pues ha quedado de manifiesto que las computadoras personales o dispositivos con un propósito particular están con frecuencia expuestos a amenazas desde el exterior, y el mero control de quién tiene acceso físico a estos equipos tiende a generar una falsa impresión de seguridad, lo que en realidad es relativamente sencillo de vulnerar debido al alto nivel de interacción de algunos sistemas.

Control de acceso

Debido a la necesidad de controlar el acceso a los archivos, la mayoría de los sistemas de archivos de uso generalizado implementan al menos un nivel rudimentario de control sobre las operaciones y partes del sistema de archivos que serán accesibles a los procesos, dependiendo de la identidad de los usuarios que los inician. Esta estrategia tiene la ventaja de que no requiere implementar mecanismos de autenticación al sistema de archivos, pues confía en que los ya implementados por la administración de procesos para reconocer a los usuarios sean suficientes. Cuando se requiere una certeza adicional se recurre a técnicas de cifrado para que, aunque se tenga acceso al archivo, la información no pueda ser recuperada sin un mecanismo adicional de identificación.

Dominios de protección

Un dominio de protección es la relación entre un conjunto de operaciones factibles de realizar (lectura, escritura, eliminación, renombrado, etc.), los usuarios o grupos que pueden intentar realizarlas y los archivos sobre las que aplican. El conjunto de asociaciones entre los usuarios y los archivos para una operación se considera un dominio de protección, y no es exclusivo de los archivos, ya que el concepto puede aplicarse a muchas otras entidades del sistema.

En los sistemas de archivos, los dominios de protección especifican qué es lo que se pretende controlar y la forma en que se darán las autorizaciones a los usuarios o a los grupos que contienen a los usuarios. Para ello se suele usar el principio de menor

privilegio, el cual estipula que un usuario solo recibirá los privilegios mínimos necesarios para realizar las operaciones que requiera al hacer uso del sistema y nada más. Por ello, todos los privilegios que no le sean otorgados se consideran denegados y con ello se evita que un uso inadecuado de los recursos ponga en riesgo de forma innecesaria la integridad de la información.

La representación de los dominios se lleva a cabo mediante varios mecanismos. Uno de los más sencillos es UNIX, donde se consideran 12 bits de protección para cada archivo, y estos definen los privilegios que recibirán los usuarios en tres niveles diferentes:

- **Dueño.** Define los privilegios para leer, modificar y ejecutar el contenido del archivo para el usuario registrado como dueño del archivo.
- **Grupo.** Define los mismos tres privilegios para los usuarios, quienes sin ser los dueños pertenezcan al grupo al que se asoció el archivo.
- **Otros.** Estos serán los tres privilegios a aplicar a todos los usuarios que no sean el dueño ni pertenezcan al grupo asociado al archivo.

Los siguientes tres bits adicionales se usan para determinar conductas especiales:

1. **Set UID** (Set User ID). En programas ejecutables hacen que el proceso que ejecute el programa emplee como usuario y grupo efectivo aquellos que están asociados al archivo del programa, en vez de los del usuario que ejecuta el programa; con ello el sistema permite que algunas aplicaciones den acceso temporal a los usuarios para realizar operaciones privilegiadas sobre archivos o sobre el sistema. En cambio, en directorios ocasiona que los nuevos archivos creados en el directorio hereden el User ID del directorio y no los del usuario que los crea.
2. **Set GID** (Set Group ID). Análogo a Set UID pero empleado para especificar el grupo.
3. **Sticky bit.** Se usa para que en directorios donde múltiples usuarios pueden crear y modificar archivos, sin ser dueños del directorio, cada uno solo pueda modificar y eliminar los archivos que él mismo ha creado y no los de los demás usuarios.

Estos privilegios, basados en 12 bits, son muy flexibles y permiten controlar gran variedad de esquemas de privilegios; sin embargo, si muchos usuarios requieren definiciones detalladas de privilegios, resulta inconveniente definir un grupo para cada combinación de privilegios que se desee. Para ello, UNIX también soporta una segunda mecánica de asignación de privilegios que complementa a la anterior, la cual es conocida como **Lista de control de acceso**.

La lista de control de acceso (ACL) permite especificar un conjunto de tres privilegios (leer, escribir y ejecutar) para todos los usuarios o grupos que se desee.

En los sistemas EXT3 usados por lo común en Linux, ACL se encuentra deshabilitada, pero se puede activar a voluntad en las particiones para comenzar a usarlas. Cuando se usan las ACL, los privilegios especificados con el esquema básico se toman como punto de partida de los privilegios a otorgar, a fin de evitar que al comenzar a usar las ACL los archivos comiencen a permitir más operaciones de las que autorizaban antes. Se considera que el privilegio de nivel **Otros** deberá limitar los privilegios otorgados por las ACL a modo de máscara; es decir, los privilegios que otorgue la ACL que no hayan sido otorgados en **Otros** serán ignorados.

Las ACL permiten definir los tres privilegios para el dueño, para el grupo al que se asocia el archivo, para usuarios y grupos individuales, para los demás usuarios y máscaras (límites adicionales a los privilegios otorgados), así como privilegios a otorgar al crear archivos nuevos dentro del directorio en el que se definan estos privilegios, a los que se llama "defaults".

Por su parte, Windows en NTFS usa un modelo de control de acceso que tiene dos partes fundamentales:

- **Tokens de acceso.** Contienen información sobre el usuario que ha iniciado sesión en el equipo.
- **Descriptores de seguridad.** Con la información de protección relevante para un objeto.

Los objetos sujetos a protección en Windows incluyen archivos, directorios, tuberías con nombre, tuberías sin nombre, procesos, hilos, *tokens* de acceso, sistemas enteros, servicios de Windows, impresoras, etcétera.

Los *tokens* de acceso se crean cuando un usuario inicia su sesión y se distribuyen copias de él a todos los procesos que inicia. Contienen identificadores de seguridad que permiten reconocer al usuario y a todos los grupos a los que pertenece. Además, incluye la lista de privilegios que le han sido proporcionados al usuario en cada uno de los grupos y para objetos específicos.

Al crear instancias de objetos sujetos a protección, Windows les asigna un descriptor de seguridad, el cual identifica al usuario que actuará como dueño del objeto, además de dos listas de controles de acceso:

- Lista de control de acceso discrecional (Discretionary access control list, DACL). Identifica a los usuarios y grupos a quienes se aprueba o rechaza mediante una serie de registros de control de acceso (Access control entries, ACE).

- Lista de control de acceso de sistema (System access control list, SACL). Especifica los privilegios requeridos para realizar las operaciones de monitoreo que la administración del sistema tendrá sobre el objeto, también mediante un conjunto de ACE.

Cada registro de control de acceso (ACE) contiene una lista de privilegios de acceso asociados a un identificador de sistema para el agente. Los privilegios pueden indicar que una operación se permita, rechace o monitoree. Los agentes pueden ser usuarios, grupos o sesiones de usuario.

Criptografía

Otra alternativa para prevenir que usuarios no autorizados vean la información en un archivo es utilizar algoritmos que transformen la información en un archivo mediante un procedimiento matemático que modifique la información de forma aleatoria basada en, al menos, un número muy difícil de adivinar, llamado llave, el cual suele ser muy grande y se fundamenta en números primos también grandes. Este algoritmo puede ser revertido si se conocen las llaves originales.

El estudio de los algoritmos y los tipos de llaves es tarea de la criptografía y resulta un tema muy interesante. Al elegir los algoritmos que se emplearán para asegurar la información deben compararse con seriedad las características contra el nivel de protección que deseamos, ya que es fácil subestimar los riesgos que se corren o elegir algoritmos que a la larga resultan gravosos para la operación, con lo que aumenta el costo de utilizarlos incluso más allá de lo conveniente.

Algunos sistemas de archivos incluyen algoritmos de cifrado para sus archivos; también existen productos que permiten cifrar toda la información de un dispositivo de almacenamiento masivo y luego utilizan dispositivos virtuales para hacer el proceso inverso al vuelo durante la operación del equipo mediante un dispositivo virtual.

Control de concurrencia

Los procesos pueden utilizar archivos de manera concurrente. Cuando se trata solo de operaciones de lectura no se presentan inconvenientes; al contrario, el *caché* en memoria de las partes del archivo que se han utilizado puede acelerar las operaciones. Sin embargo, cuando uno o más procesos deben realizar modificaciones a la información, es necesario tomar medidas para evitar condiciones de competencia con la información y así impedir que las lecturas de la información recuperen datos que ya han sido modificados.

Protección de código ejecutable

Los programas a ejecutar suelen almacenarse en los sistemas de archivos en espera de ser ejecutados. Esto plantea el problema de protección que radica en la mezcla de información de control con datos. En este caso, los datos contenidos en los archivos con programas se encuentran en los mismos directorios que albergan los archivos con datos, bitácoras y configuraciones, y los usuarios necesitan privilegios de lectura sobre todos estos elementos para poder ejecutar con éxito las aplicaciones.

El primer desafío consiste en evitar la modificación no apropiada de los programas. En principio, proporcionar accesos de solo lectura a los archivos correspondientes puede ser suficiente para evitar la modificación de los programas existentes, pero los privilegios sobre los directorios donde se almacenen archivos generados por la aplicación hacen necesario que los usuarios puedan crear y eliminar archivos de esos directorios. Por esta razón, hoy día casi todos los sistemas crean directorios en los que las aplicaciones generan nuevos archivos para uso exclusivo de cada usuario, separados de los directorios donde se almacenan los programas, e incluso se crean directorios específicos para que cada usuario genere sus propios archivos con la configuración de las aplicaciones.

Formato de programas, código ELF, PE y Mach-O

Ante el hecho de que los usuarios puedan crear archivos nuevos, existe la posibilidad de que el archivo que creen contenga un código ejecutable o trate de ser empleado como tal, cuando no sea parte de la aplicación.

Para evitar que sean ejecutados los archivos que no contienen código ejecutable, lo cual generaría efectos difíciles de predecir que podrían poner en riesgo la integridad del sistema, se emplean códigos al principio de los programas ejecutables auténticos que permiten al sistema operativo verificar si el contenido en realidad corresponde a un programa ejecutable.

Por ejemplo, en sistemas UNIX y Linux se usa el formato ELF (Executable and Linking Format), que es flexible y define reglas específicas para la representación de los programas ejecutables. Invariablemente inicia con un registro de 52 bytes, conocido como el encabezado principal del archivo ELF, y sigue con un código conocido como **número mágico** que indica el tipo de información o código que contiene el resto del archivo. Por ejemplo, puede mostrar si se trata de un archivo ejecutable, una biblioteca de objetos que se pueden compartir entre varias aplicaciones, si el código permite que se relocalice en la memoria, si es específico a un modelo de CPU, etc. Después, el registro especifica detalles tales como la arquitectura a la que está dirigido, si requiere un

procesador de 32 o de 64 bits, la versión del código ELF que está empleando y el sistema operativo que requiere, entre otras características.

Los sistemas Windows usan un formato análogo propietario llamado Portable Executable (PE), y Mac OS usa otro llamado Mach-O; este último tiene la peculiaridad de permitir que un solo archivo contenga componentes para múltiples arquitecturas, de modo que al momento de ejecutar el programa se determinen cuáles deben ser utilizadas.

Firmas de componentes

Estos formatos de programas ejecutables permiten garantizar que los elementos de las aplicaciones serán manejados de manera correcta; sin embargo, persiste el riesgo de usar el componente equivocado cuando una aplicación usa componentes compartidos que no se incluyen en el código del programa, sino que son cargados a memoria conforme se requieren.

Lo anterior podría producirse por actualizaciones de software que se lleven a cabo de forma parcial, por errores en la configuración de las rutas de archivos donde se buscarán las bibliotecas a cargar para un usuario en particular, e incluso por la acción de un usuario malicioso que, habiendo adquirido acceso a privilegios de administración, reemplace una biblioteca por una versión que altere la funcionalidad.

Para evitar este tipo de problemas se ha popularizado el firmado por medios criptográficos de las bibliotecas a usar por las aplicaciones y los correspondientes mecanismos de verificación en tiempo de ejecución.

Cabe resaltar que para ambientes de desarrollo de Microsoft se implementó el Strong-Name Signing for Managed Applications, y en UNIX y plataformas como Java se está impulsando la Service Component Architecture (SCA). Ambas tecnologías están orientadas a definir reglas que permitan integrar aplicaciones compuestas de elementos de diversas fuentes con mejores prestaciones de seguridad, estabilidad y soporte al desarrollo respecto a los métodos tradicionales.

MBR

En el sistema de archivos se encuentra también el código del propio sistema operativo. Uno de los componentes críticos de este es el programa encargado de coordinar la carga a memoria y el inicio del propio sistema operativo. En el caso de las computadoras IBM PC compatibles, el BIOS busca la información de los sectores de arranque en los dispositivos de almacenamiento masivo disponibles, de acuerdo con su configuración. El primero que revisa es el Master Boot Record (MBR), en el cual se almacena

información que indica cómo están colocadas las particiones en el dispositivo y el código ejecutable empleado para iniciar el sistema operativo.

Esto hace que corra el riesgo de que algún programa con acceso directo al dispositivo de almacenamiento reemplace dicha funcionalidad de carga por un programa malicioso, o simplemente la destruya inutilizando los sistemas de archivos que contiene. Muchos virus informáticos aprovechan esto como un mecanismo para propagarse, ya que el MBR debe leerse siempre que se comienza a usar un dispositivo.

Para evitar el daño, la mayoría de los controladores de discos duros permiten proteger los MBR directamente mediante modos de operación que prohíben modificar el sector de arranque de las unidades de discos y que son determinados por la configuración del BIOS. Se espera que el administrador del equipo restrinja durante la operación normal y solo la habilite cuando se requiera instalar un nuevo sistema operativo en el equipo.

6.7 Manejo de espacio en memoria secundaria

Como se mencionó en el capítulo 4, *Administración de memoria*, cuando se emplea swap o memoria virtual, se requiere almacenar parte de la información que los procesos contienen en la memoria RAM en el almacenamiento secundario cuando esta no se usará durante algún tiempo, y así liberar memoria RAM que podría destinarse a otros procesos. En su momento se dejó de lado el mecanismo que se emplearía en el almacenamiento secundario, en particular en el sistema de archivos, por lo que se revisará en esta sección, retomando los conceptos de sistemas de archivos que hemos revisado.

Las dos estrategias principales para almacenar de manera temporal la información que sacamos de la memoria RAM en el dispositivo de almacenamiento secundario son una partición de swap y el uso de archivos de intercambio. Sistemas como Linux abstraen los dos mecanismos en áreas de swap (conocidas como swap áreas), que usarán de forma indistinta después.

En el caso de las particiones de swaps, se genera una partición especial en el dispositivo para almacenar la información de la memoria. En esta partición se usa un sistema de archivos destinado a dar un acceso rápido a los bloques que contienen la información de los segmentos o de las páginas. Para ello dividen la partición en una serie de espacios para página (*page slots*) del mismo tamaño que las páginas en memoria (usualmente 4 KB).

En particiones de swap, la información contenida queda inutilizada al reiniciar el sistema ya que todos los procesos que la usaban terminan, de modo que solo se almacena un mínimo de información de administración en ellas. El primer espacio para

página se usa para almacenar la información de todos los espacios para página inutilizables por estar en bloques dañados del dispositivo y un arreglo en el que se almacenan las posiciones de las páginas que almacena. Si el sistema requiere mover un conjunto de páginas de la memoria principal al almacenamiento secundario, solicita un conjunto contiguo de espacios para página para ello; esto es muy conveniente para preparar el swap de un segmento completo. Estos conjuntos de espacios para página se conocen como extensiones de páginas (swap extents).

En el primer espacio para página de la partición swap también se almacena la dirección de inicio y el tamaño de las extensiones de páginas que se emplean en esta área de swap.

Si se requiere una nueva extensión de páginas, se puede buscar en el área de swap ya sea desde el principio o a partir de la última extensión de páginas, dependiendo de una serie de condiciones. Al mantener las extensiones de páginas contiguas se acelera la lectura, pues se evita fragmentar los datos de los segmentos a leer a costa de tener más fragmentación externa entre las extensiones de página y, en potencia, requerir más almacenamiento secundario, pero este inconveniente es cada vez menor debido a que la tendencia actual es incrementar la cantidad de memoria RAM en las computadoras, lo cual alivia las presiones sobre la memoria virtual.

Cuando se usa un archivo para almacenar las áreas de swap, se tiene la sobrecarga de pasar por el sistema de archivos que lo alberga para identificar el archivo correcto y su ubicación en el directorio. Sin embargo, el impacto es mínimo, pues esto solo se realiza al abrir el archivo y se hace por lo regular una sola vez al iniciar el sistema, de ahí que este enfoque ha ganado popularidad en fechas recientes. Una vez que se tiene el archivo con el área de swap, el manejo es prácticamente igual que cuando se tiene una partición de swap. La única diferencia importante es que la ubicación de los bloques en el disco está sujeta a la fragmentación natural de los archivos en el sistema de archivos que la alberga, lo que puede tener efectos adversos en el desempeño. De nuevo, debido a la tendencia de incluir más memoria RAM en sistemas actuales, esta presión es mínima y el efecto es de poca importancia.

Respaldo y mecanismos de recuperación en caso de falla

Mecanismos de recuperación

Los sistemas de archivos están sujetos a pérdidas de información ocasionada por dos tipos principales de problemas. El primer tipo son los fallos en los dispositivos, tanto temporales como permanentes; estos fallos son atendidos en su mayoría por la codifi-

cación y por los mecanismos de corrección de errores revisados en la parte de organización de archivos, y será optimizado para las características de cada dispositivo.

El segundo motivo que pone en riesgo la información es la demora entre el término de las operaciones de archivo y el momento en que los datos almacenados en memoria son realmente escritos en el dispositivo de una forma no volátil. Recordemos que hay múltiples niveles de búferes y reorganización de las operaciones que implican que los datos a menudo tomen tiempos largos en ser escritos en el dispositivo, y aun cuando son enviados, con frecuencia el propio dispositivo agrega un segundo nivel de *caché*. En especial ante errores que obligan al equipo a reiniciar o que suspenden la alimentación de potencia de manera súbita, resulta imposible garantizar que toda la información se escriba en el dispositivo, así como predecir cuáles elementos se perderán.

Revisiones de consistencia

Ante los problemas mencionados con anterioridad, es conveniente tener programas de mantenimiento que comparen la información de los metadatos (directorios o nodos-i) con la información contenida en los archivos para detectar inconsistencias y, en caso de encontrarlas, corregirlas.

Por ejemplo, si las modificaciones del archivo lograron ser registradas pero los metadatos en el directorio no se actualizaron, los atributos se pueden reconstruir a partir de los bloques del archivo y actualizar el directorio (en el caso de FAT). En EXT3 puede ocurrir lo contrario, ya que podría garantizarse que los metadatos se escriban antes que los datos del archivo; en caso de encontrar una inconsistencia, es factible regresar a la versión anterior del nodo-i o reconstruir la información desde los bloques, que, al igual que FAT, están organizados en una lista doblemente ligada.

También existen utilerías que permiten revisar todos los bloques del dispositivo y no solo los de la partición, a fin de diagnosticar e intentar reparar errores en la tabla de particiones; sin embargo, estos a menudo están limitados a unos cuantos tipos de sistemas de archivos en las particiones y, en general, implican un riesgo mucho mayor de corromper la información presente en el disco.

También debe considerarse el tiempo que tomará una revisión de todos los bloques del disco, considerando los dispositivos actuales que cuentan con varios miles de millones de bloques. Esta revisión toma un tiempo considerable y debe evitarse.

Bitácoras (Log File Systems)

Para evitar que los errores en la operación del sistema pongan en riesgo la información que originalmente se tenía almacenada en el sistema de archivos, se pueden imple-

mentar mecanismos que mantengan la información original mientras se escribe la nueva información, y solo cuando esta ha sido escrita con éxito en su totalidad se marquen como inactivos los bloques que almacenaban la versión anterior o modificada del archivo.

Estos mecanismos se conocen como bitácoras o Log File Systems, por su parecido con las bitácoras náuticas, en las que no se permiten las modificaciones a las anotaciones pasadas y las correcciones siempre deben realizarse con nuevas anotaciones. También toman conceptos de transaccionalidad por su capacidad de recuperar el estado inicial de un archivo en caso de que la modificación sea interrumpida o falle.

Tanto EXT3 y 4 como NTFS implementan sistemas de bitácora que, a grandes rasgos, definen un área especial en el sistema de archivos donde se almacenan las descripciones de las operaciones que deben escribirse al disco. Esta área se usa como una lista circular con el propósito de acelerar las operaciones de acceso, ya que al tener un apuntador (también almacenado en disco) a las operaciones que están pendientes, se evita tener que recorrer toda la lista para encontrar la operación siguiente.

El objetivo es permitir que el dispositivo organice las operaciones a realizar en la forma en que pueda proporcionar mejor rendimiento, pero que en caso de que haya una interrupción repentina en la operación se logre recuperar el listado de las operaciones que estaban en proceso o pendientes y simplemente se puedan aplicar las operaciones en la lista para evitar las pérdidas de información.

Esto conlleva el costo asociado de que las operaciones a disco terminan por realizarse dos veces, una en la bitácora y otra en la superficie normal del sistema de archivos. Para reducir la sobrecarga, EXT3 permite configurar el tipo de modificaciones que se escribirán en la bitácora; por defecto, solo las modificaciones a metadatos se escriben en la bitácora, aunque se permite configurar la partición para garantizar que todas las operaciones se escriben primero en la bitácora o para mejorar el rendimiento, permitiendo que las operaciones a la bitácora se realicen después de las operaciones en los bloques comunes de la partición, según los algoritmos de optimización del dispositivo.

Las operaciones que se registran en la bitácora deben poder aplicarse múltiples veces sin efectos adversos, característica que se conoce como idempotencia. Así, si la operación en cuestión ya se había realizado al momento de producirse la falla, pero no estaba marcada así en la bitácora, simplemente puede aplicarse de nuevo al inicio del sistema para vaciar la bitácora sin que se corrompa la información. Como las operaciones se centran en grabar determinado contenido en un bloque, el hecho de repetir estas operaciones en efecto no genera consecuencias adversas.

RAID

Los dispositivos de almacenamiento masivo tienen gran demanda en el mercado por lo que deben ofrecer el mejor costo por megabyte de almacenamiento posible; sin embargo, esto hace que los componentes sean de mediana calidad, por lo que los tiempos esperados de vida de los dispositivos y los tiempos promedio entre fallas no pueden mejorar de manera continua, sino que tienden a permanecer en torno a una media de la industria con tal de no incrementar los costos.

El disco duro mecánico común suele tener un tiempo esperado de vida aproximado de 2000 horas. Esto resulta suficiente para la mayoría de las aplicaciones domésticas, pero para servidores de datos (en especial si se desea tener el disco duro en operación prácticamente sin pausa) significaría que el dispositivo dejaría de operar poco más de un año. Perder la información del sistema cada año no es permisible para muchas aplicaciones, por lo que se requieren mecanismos que permitan prolongar los tiempos de vida de los dispositivos de almacenamiento masivo sin elevar mucho el costo y que reemplacen los dispositivos que fallen sin perder información.

Para atacar estas problemáticas se han construido diversas soluciones que utilizan un conjunto de discos duros de bajo costo y que reparten la carga de trabajo mediante software para ofrecer prestaciones superiores a lo que una sola unidad, incluso de costo superior, podría ofrecer.

El estándar RAID es una norma que indica diversos niveles de capacidades que las colecciones de dispositivos pueden exponer, aunque no especifica cómo deben implementarlas, por lo que a menudo una solución RAID no es compatible y no puede utilizar los datos en dispositivos que se encontraban en otra solución RAID.

RAID significa Arreglo redundante de discos de bajo costo (Redundant Array of Inexpensive Disks, o en una versión posterior Redundant Array of Independent Disks); fue publicada por SNIA y hoy en día se encuentra en la versión 2 del Formato de Datos en Disco Común para RAID.

Esta norma reunió las capacidades de diversas implementaciones previas de arreglos de discos que usaban la redundancia y las operaciones paralelas en diversas formas; cada uno de estos esquemas de operación se organizaron en niveles diferentes identificados por números, los cuales se describen a continuación.

RAID 0

Conocido como Solo un montón de discos (Just a Bunch of Disks, o JBOD), este esquema distribuye prácticamente por igual el almacenamiento de la información en todos los dispositivos disponibles (spanning), con lo que casi se puede asegurar que

la paralelización de las operaciones de conjuntos de bloques se distribuirá entre los dispositivos y se tendrá un incremento en las velocidades de lectura y de escritura proporcional al número de dispositivos que se empleen. Sin embargo, no implementa esquemas de redundancia, por lo que los fallos en alguno de los dispositivos no podrán recuperarse y generarán pérdidas de información en los sistemas de archivos que albergan (véase figura 6.6).

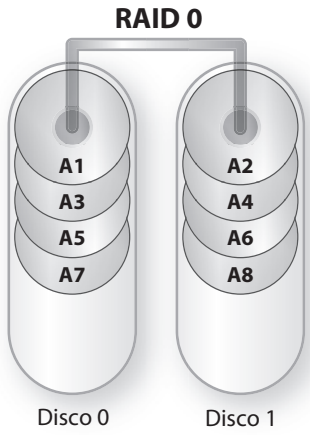


Figura 6.6 Organización de información en RAID 0.

RAID 1

En este esquema, denominado espejo (mirror), los dispositivos se organizan en parejas con las mismas características. Todas las operaciones realizadas en el primer dispositivo, al que se conoce como Maestro, se replican en el segundo, al que se denomina Esclavo. Esto genera una redundancia completa de la información, por lo que no se mejora la velocidad con que se realizan las operaciones, pero en caso de fallar uno de los dispositivos, la información estará a salvo en el otro. Si el dispositivo que falla es el Maestro, el control se transmite de manera automática al Esclavo, el cual pasa a ser el Maestro, y se notifica al usuario para que se reemplace el dispositivo que ha fallado.

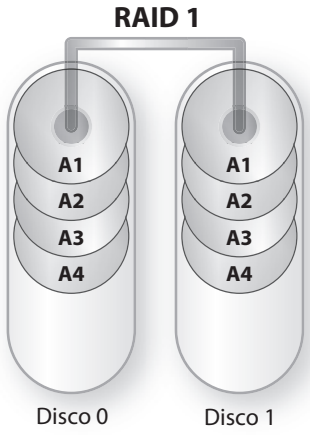


Figura 6.7 Organización de información en RAID 1.

En algunos sistemas se permite que el dispositivo se reemplace sin suspender la operación del sistema. A esta capacidad se le conoce como Hot Swap y requiere que el controlador RAID pueda proceder a copiar la información al nuevo disco esclavo cuando esté disponible y sin interferir con las operaciones normales (véase figura 6.7).

RAID 2

Se trata de un esquema en desuso que distribuye la información bit por bit entre todos los dispositivos usando un código Hamming de corrección de errores (el cual permite detectar hasta dos bits de error en la serie y corregir uno). Hoy día, los discos duros implementan por sí mismos códigos Hamming de corrección de errores, por lo que implementarlos en el controlador RAID resultaría complicado e improductivo (véase figura 6.8).

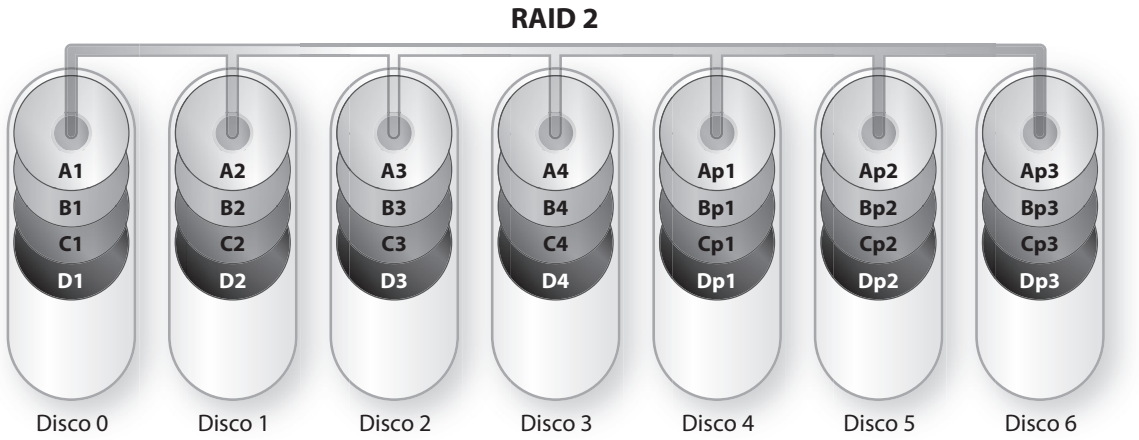


Figura 6.8 Organización de información en RAID 2.

RAID 3

Este es un esquema basado en una distribución de datos byte por byte (Byte stripign) en todos los dispositivos del conjunto, con un disco dedicado de paridad. Solo mejora la velocidad de transferencia cuando se requiere hacer lecturas de datos secuenciales de muchos datos, ya que en este caso la información recuperada en paralelo en todos los dispositivos será utilizada. Por el contrario, si se requieren múltiples consultas a datos en posiciones aleatorias, cada una de estas también usará todos los dispositivos, evitando que se atiendan en paralelo, y recibirá solo el rendimiento equivalente de un dispositivo (véase figura 6.9).

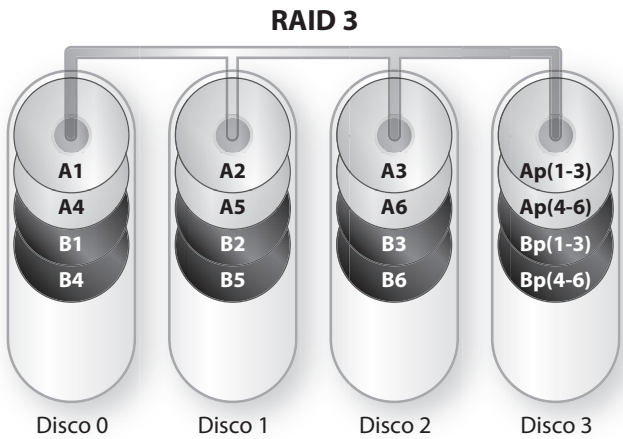


Figura 6.9 Organización de información en RAID 3.

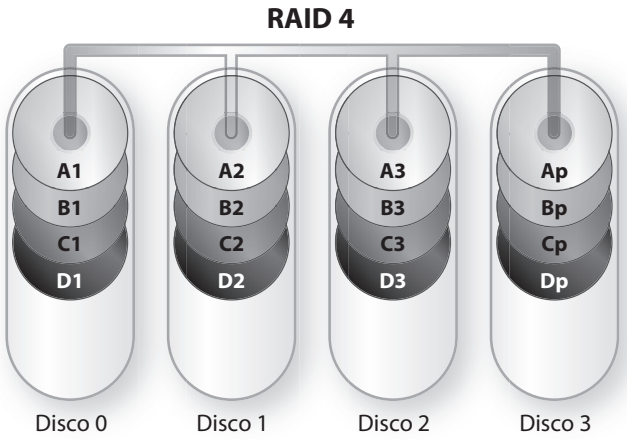


Figura 6.10 Organización de información en RAID 4.

RAID 4

Acceso independiente con discos dedicados a paridad. Distribuye los datos a nivel de bloque y no de byte, como el nivel 3, y almacena todos los bloques de paridad en un dispositivo dedicado. Necesita un mínimo de tres dispositivos físicos. Al almacenar los bloques completos, para acceder a un dato permite que se active un dispositivo independiente e incluso que se atiendan varias peticiones de lectura en dispositivos diferentes de manera simultánea. Las operaciones de escritura no se pueden paralelizar de esta forma porque todas involucran al dispositivo dedicado a paridad, lo que generaría un cuello de botella.

RAID 5

Distribuido con paridad, genera la información de paridad por bloques, pero distribuye el almacenamiento de estos entre todos los dispositivos del conjunto. Ha logrado po-

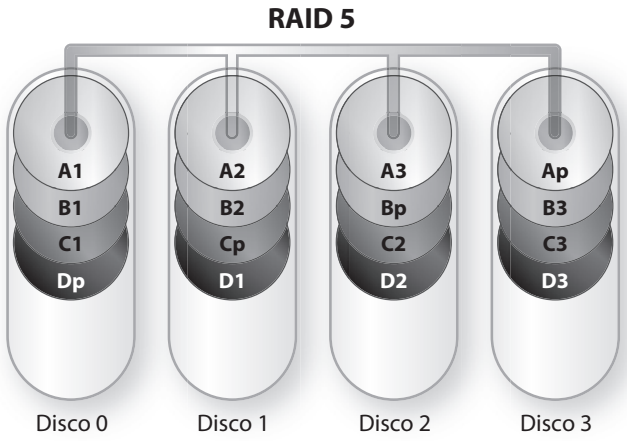


Figura 6.11 Organización de información en RAID 5.

Importante

- La norma de los niveles de RAID no especifica los formatos o algoritmos que cada fabricante use para implementar la funcionalidad que requiere cada nivel. Esto brinda libertad a los fabricantes de usar variantes que los diferencien de los demás, pero limita la capacidad de operar los dispositivos que almacenaron información con un controlador RAID en un segundo controlador de características diferentes. Esta limitación ha dado alguna ventaja a las implementaciones de software de RAID debido a que estas tienen facilidades para emplear la misma implementación bajo distintas plataformas, con lo que se puede asegurar el soporte a la información más allá de los tiempos en que se fabriquen los dispositivos del tipo usado originalmente.

pularidad debido a que proporciona buena protección con un bajo costo asociado a la redundancia. Necesita un mínimo de tres dispositivos para ser implementado, y a menudo se encuentra en el hardware de controladoras de disco incluso de bajo costo para cómputo personal. Cuando se modifica la información de un bloque, se deben leer los demás bloques del conjunto para recalcular la información que habrá de escribirse en el bloque de paridad, lo que hace que las operaciones de escritura en RAID 5 sean costosas en tiempo y comunicación entre los dispositivos cuando las modificaciones no se realizan para los conjuntos completos de bloques o stripes.

RAID 6

Muy similar a RAID 5, pero agregando un segundo bloque de paridad que también se distribuye entre los dispositivos. Este nivel de redundancia adicional brinda protección para los casos en que se presenten fallas en más de un dispositivo en un momento determinado (por ejemplo, si hay un fallo mientras se está reconstruyendo la información de un dispositivo que falló con anterioridad). Este nivel de RAID no era parte de la especificación original y, al igual que otras medidas, fue añadida posteriormente a la norma.

Algunas otras variantes recientes son RAID 5E y RAID 6E, que contemplan que los conjuntos de discos tengan dispositivos inactivos en reserva para entrar en operación de forma automática en caso de

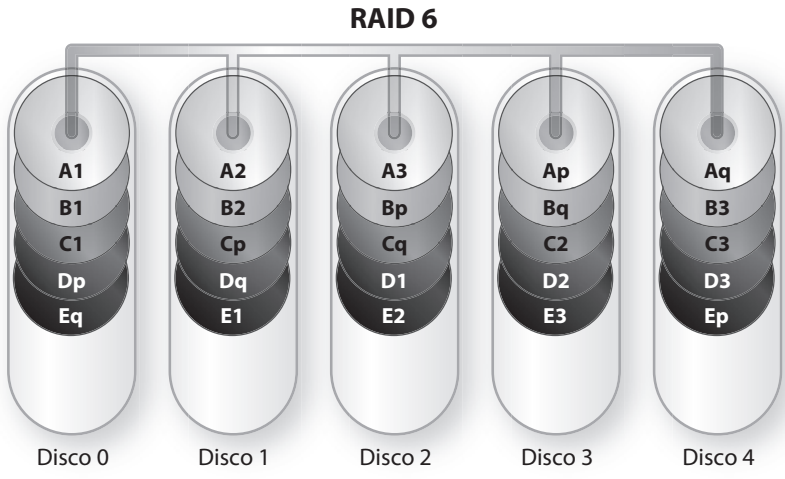


Figura 6.12 Organización de información en RAID 6.

que alguno de los dispositivos del conjunto falle. También se han popularizado los arreglos anidados, por ejemplo para hacer un RAID 0 JBOD que cuente con un segundo RAID 1 en espejo, que se denomina RAID 0+1.

Respaldos

Como ya se mencionó, los medios de almacenamiento invariablemente están sujetos a desgaste y a errores en la operación que ponen en riesgo la información que albergan. Por lo anterior es necesario considerar que los dispositivos dejarán de operar de manera eventual y que la información contenida en ellos no podrá ser recuperada.

Perder la información es contrario al objetivo de los sistemas de archivos y, dependiendo de la aplicación, generará un costo ya sea por el esfuerzo realizado para volver a generarla o por carecer de ella.

Para reducir el costo asociado a los fallos en los dispositivos y en la operación que ocasiona pérdidas de información, se pueden generar copias de respaldo. Para que el costo de tener la información duplicada se reduzca, es necesario considerar los siguientes aspectos:

- El dispositivo con las copias de respaldo debe ser diferente al dispositivo con la información original.
- La copia de respaldo debe resguardarse físicamente, evitando que se dañe o extravíe, aunque usualmente no se esté utilizando.
- Los dispositivos y medios de almacenamiento usados tienen un costo, y debe minimizarse la inversión usando compresión y respaldos incrementales.
- Recuperar la información de la copia de respaldo también requiere tiempo y esfuerzo, y el costo de esto debe considerarse al diseñar el plan de respaldos.
- Los respaldos deben generarse periódica y consistentemente si deseamos contar con información disponible para hacerle frente a cualquier fallo que se presente.
- Cuando los respaldos dejen de ser útiles, deben destruirse o reutilizarse los medios, de forma que se prevenga una fuga de información en caso de que alguien revise los desperdicios.

Para considerar todos estos puntos se requiere elaborar un plan de respaldos donde se analice, en función de los costos esperados en caso de pérdida de información, la inversión y el tipo de respaldos que resultan más convenientes, donde el costo de los respaldos más el costo esperado de la recuperación en caso de un fallo debe ser menor que el costo original por la pérdida de información.

Los planes de respaldos deben practicarse de forma regular (incluso si no se presentan fallos), deben mantenerse documentados y actualizados, y deben considerar la seguridad de la información que albergan.

6.8 Programación para usar sistemas de archivos

Para interactuar con los sistemas de archivos, las aplicaciones cuentan con una serie de interfaces de programación proporcionadas por el sistema de archivos, o por el *middleware* en el caso de algunas arquitecturas de sistemas distribuidos. Estas permiten interactuar con los archivos para aprovechar sus prestaciones.

Tipos de interfaz

Hay dos tipos principales de interfaces de programación para interactuar con los archivos y, por extensión, con los dispositivos de entrada y salida que emplean las mismas interfaces. La primera es la **API de programación para entradas y salidas**, que, mediante bibliotecas de funciones que se incluyen en las aplicaciones, está diseñada para optimizar y simplificar el desarrollo de aplicaciones que empleen la funcionalidad normal de los sistemas de archivos. La segunda, conocida como **llamadas de sistema**, es aquella empleada por los controladores de los dispositivos para exponer su funcionalidad de manera consistente para ser explotada por el sistema operativo y que puede utilizarse directamente por las aplicaciones para tener un control fino sobre las operaciones que se realizan en ellos.

API de programación

Estas bibliotecas, o Interface de Programación de Aplicaciones (Application Programming Interface), suelen ser bibliotecas de carga dinámica que se incluyen en el contexto de los procesos de usuario y se ejecutan con los privilegios normales. Corresponden con el software de área de usuario revisado en el capítulo 5, *Administración de los dispositivos de entrada y salida*, e implementan los mecanismos, búfer y optimizaciones pertinentes a fin de que las llamadas de sistema que se realicen para interactuar con los dispositivos se aprovechen al máximo en las tareas más comunes.

En la mayoría de las variantes de UNIX, la biblioteca donde se encuentran estas es stdio.h, y eso afecta a la mayoría de los sistemas operativos basados en UNIX y Win-

dows; por ejemplo, en C# define métodos equivalentes en el namespace: System.IO. StreamReader.

Las funciones básicas para usar archivos incluyen, por lo regular, las siguientes tareas:

- **Comenzar a usar un archivo**
  Con esta tarea se crearán las estructuras de datos para utilizar el archivo, ya sea para recuperar información o para modificarla. En este último caso puede hacerse creando un nuevo archivo o agregando información al final.
- **Finalizar el uso de un archivo**
  Permite liberar la memoria usada en las estructuras de datos que registran el estado y los búferes asociados a un archivo. Es muy importante llamarla, ya que también garantiza que los cambios realizados pasarán a escribirse en los dispositivos de almacenamiento.
- **Leer**
  Consiste en recuperar información del archivo. Conviene recordar que no necesariamente se trata de un archivo plano, por lo que solo recupera la información disponible en el archivo recibida mediante un dispositivo de cualquier tipo.
- **Escribir**
  Tarea consistente en enviar información al archivo, ya sea escribiéndola en un dispositivo de almacenamiento, entregándola a otras aplicaciones mediante una tubería (*pipe*) o entregándola a un dispositivo de entrada y salida.
- **Buscar**
  Solicita que el flujo de información sostenido mediante las operaciones de leer y escribir pase a un punto diferente en el archivo, sobre todo para archivos planos que estén manipulando información en dispositivos de almacenamiento masivo.

Llamadas al sistema (system calls)

Se puede acceder a la funcionalidad de los dispositivos y de los sistemas de archivos de manera directa, mediante funcionalidad proporcionada por el sistema operativo en funciones implementadas en sus controladores y que operan en modo de *kernel*. Estas son conocidas como llamadas de sistema (o system calls), y aunque permiten tener un control más íntimo de los elementos, tienen una penalización en rendimiento debido en parte al cambio de contexto que se debe realizar para que la petición hecha por un proceso de usuario acceda a la funcionalidad del *kernel*.

Las llamadas a sistema implementadas tradicionalmente incluyen:

- **Open.** Para iniciar el uso de un archivo o dispositivo.
- **Read.** Para recuperar información.
- **Write.** Para enviar información.
- **Close.** Para terminar una sesión de uso, recuperar la memoria y dejar los elementos en un estado seguro.
- **Comunicación al controlador.** Estas operaciones permiten comunicarse directamente con el controlador de un dispositivo para realizar operaciones específicas. En Linux estas se realizan con llamadas a ioctl y en Windows se tiene la API de los servicios del sistema de E/S (I/O System Services).

Estas llamadas a sistema son parte de los requerimientos que cada sistema operativo impone a los controladores de dispositivos, ya que todos ellos deben ser capaces de atender estas llamadas para ser utilizados por el sistema y las aplicaciones. Al ser invocadas de la misma manera para todos, ayudan a concretar el objetivo de la independencia lógica.

Flujos de información. Streams y pipelines

UNIX introdujo el concepto de entubamiento, o *pipelines*, como una forma de generar comportamientos complejos en los sistemas conjuntando la funcionalidad de múltiples programas sencillos, evitando así desarrollar programas que, de manera explícita, representaran esa complejidad.

Para lograrlo reconoce ciertos flujos de información que entran y salen de todos los procesos, sobre todo uno de entrada que podría corresponder a la terminal en la que suele encontrarse el usuario que ejecuta el proceso; otro en donde el proceso va entregando los resultados que genera normalmente, y un tercero donde se coloca la información de estado, notificaciones de errores e información de diagnóstico para corregirlos. La capacidad de organizar los procesos en funcionalidades más complejas se basa en que podemos redirigir estos flujos de información de la consola a otros tipos de entidades (en principio, a archivos o hacia otros procesos).

De lo anterior surgen los dos conceptos siguientes:

- **Stream.** Abstracción del flujo de información de entrada o de salida de un proceso que puede corresponder a un archivo, una consola o una comunicación mediante el protocolo de red o algún otro proceso.
- **Pipe.** Operación por la que se redirige un flujo de información a un nuevo stream, ya sea hacia un proceso, un archivo, etcétera.

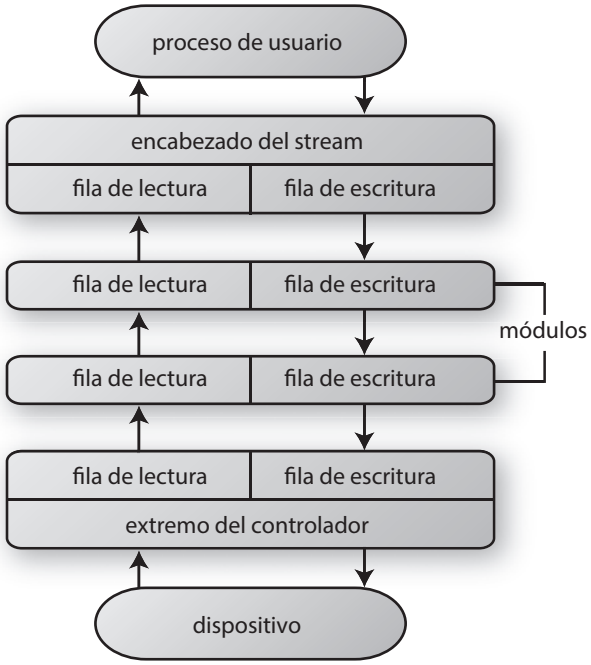


Figura 6.13 Flujo de una petición de operación de entrada o salida.

La característica de los streams radica en que sus filas de entrada pueden alimentarse de los contenidos de la fila de entrada de otro stream y, a su vez, alimentar la entrada del módulo siguiente, con lo que los controladores de los diversos niveles de controladores, protocolos de red y aplicaciones que requieren compartir el procesamiento de la información pueden usar un mecanismo común de representación y control del flujo de la información.

Cada stream tiene tres elementos principales. El primero es el encabezado o header, que expone la funcionalidad básica y las operaciones para entregar o retirar información. Adicionalmente, se implementan dos colas (FIFO) de información: una para las entradas en el stream y otra para la información de salida. Cada una de estas colas se encarga de almacenar la información en tránsito en una dirección (véase figura 6.13).

EVALUACIÓN

6.1 ¿En los archivos se almacenan datos o información?

6.2 ¿Qué tipo de información almacenan los metadatos?

6.3 ¿Todos los archivos deben almacenar su información en un dispositivo de almacenamiento masivo?

6.4 ¿En qué situación requieren las aplicaciones proteger con exclusión mutua el uso de archivos?

6.5 ¿Cuáles son los tipos de archivos tratados?

6.6 ¿Qué significa que un archivo tenga la extensión **.txt**?

6.7 ¿Cuál es la diferencia entre el tamaño máximo y el tamaño de un archivo?

6.8 ¿Por qué no se pueden hacer ligas de archivos en sistemas como NTFS que almacenan los atributos en los directorios?

6.9 ¿Qué diferencia al directorio raíz de los demás directorios?

6.10 ¿En qué tipo de dispositivos se comenzaron a usar las listas doblemente ligadas de bloques en los archivos? ¿Por qué surgió esta necesidad?

6.11 ¿Cuál es la diferencia entre la información almacenada en un disco RAM y la almacenada en un segmento de memoria asignado a un proceso?

6.12 ¿Se debería considerar a la tabla de ubicación de archivos como metadatos?

6.13 Adicionalmente al acceso físico a los equipos en que se almacenan los archivos, ¿qué medidas se toman para garantizar la seguridad de los datos?

6.14 ¿Qué significan las letras "rwx" en los permisos de archivos en UNIX? ¿Cómo funciona el sticky bit?

6.15 ¿Qué dicta el principio de mínimos privilegios para los sistemas de archivos?

6.16 ¿Cómo se emplea el atributo de archivo en la generación de respaldos?

6.17 ¿Cuál es una ventaja de la implementación de RAID mediante software con miras al tiempo de vida de los componentes?

6.18 ¿Qué consideraciones deben tomarse para un plan de respaldos doméstico? ¿Qué información se debe respaldar y cómo puede evaluarse el costo esperado por pérdidas de información?

6.19 ¿Qué parte de las API de sistemas de archivos suele usar el desarrollo de aplicaciones?

Referencias bibliográficas

Matthew, Neil y Richard Stones, *Beginning Linux Programming,* 4ª ed., John Wiley and Sons, 2007.
Silberschatz, Abraham *et al.*, *Operating Systems Concepts,* 7ª ed., John Wiley and Sons, 2005.
Stallings, William, *Operating Systems Internals and Design Principles,* 5ª ed, Pearson.
Tanenbaum, Andrew S., *Modern Operating Systems,* 3ª ed., Pearson, 2009.

Referencias electrónicas

All About Circuits
http://www.allaboutcircuits.com

Características de operación de diversas arquitecturas de memoria Flash:
http://www.design-reuse.com/articles/24503/nand-flash-memory-embedded-systems.html

Un video explicativo amateur pero detallado del funcionamiento de la memoria Flash:
https://www.youtube.com/watch?v=s7JLXs5es7I

JFFS2: The Journalling Flash File System, version 2
https://sourceware.org/jffs2/

SafeFLASH de HCC embedded
http://www.hcc-embedded.com/products/file-sytems/flash-file-systems

Sobre los códigos ELF:
http://www.sco.com/developers/gabi/latest/ch4.eheader.html#elfidhttp://www.ouah.org/RevEng/x430.htm

Sobre la firma de componentes empleada en assemblies de Visual Studio:
http://msdn.microsoft.com/en-us/library/ms247066.aspxhttp://msdn.microsoft.com/en-us/library/h4fa028b(v=vs.80).aspx

Sobre la Service Component Architecture (SCA):
http://www.oasis-opencsa.org/sca

Norma Common RAID Disk Data Format (DDF) en SNIA:
http://www.snia.org/tech_activities/standards/curr_standards/ddf

Revisión de las características de RAID y sus niveles:
http://en.wikipedia.org/wiki/RAIDhttp://en.wikipedia.org/wiki/Standard_RAID_levels

7200-RPM Drive Specification Comparison de Seagate:
http://www.seagate.com/files/www-content/product-content/_cross-product/en-us/docs/7200rpm-drive-spec-mb578-7-1201us.pdf

COMPETENCIAS A DESARROLLAR

- El alumno podrá diferenciar los sistemas distribuidos de los sistemas centralizados y reconocer la importancia de los primeros en las aplicaciones de mayor importancia económica de la actualidad.
- Diferenciará las características principales de diversos estilos arquitectónicos de sistemas distribuidos y los motivos principales que los moldearon.
- Identificará los principales mecanismos implementados con apoyo del sistema operativo orientados a soportar el desarrollo de sistemas distribuidos, la forma en que funcionan y las características de estos.

Sistemas distribuidos

7

¿QUÉ SABES...?

1. Contesta las siguientes preguntas:
   - Considerando todos los dispositivos de cómputo que empleas, ¿cuáles aplicaciones dependen para su funcionamiento de intercambiar información con otros dispositivos, aunque sea de forma esporádica?
   - Teniendo en cuenta aquellas aplicaciones que funcionan de manera autónoma y no requieren este intercambio de información, ¿conoces alternativas que sí empleen ese tipo de comunicación para atender las mismas necesidades con ventajas adicionales?
2. Revisa diversos artículos y publicaciones recientes y prepara una lista de algunas aplicaciones con mayores ingresos en el último año. Responde las siguientes preguntas.
   - ¿Qué modelos de comercialización emplean?
   - ¿El uso de esas aplicaciones requiere una conexión a Internet o a algún otro tipo de redes?

Importante

♥ **Docente.** Debido a que este es un tema reciente, es posible que la notación empleada en esta obra no corresponda con la de otros autores, pero como no hay todavía una autoridad o norma al respecto, queda como responsabilidad del profesional de cómputo adoptar la notación que le parezca más adecuada y usarla de forma consistente.

♠ **Arquitecto.** Siempre que se aborde el diseño de un sistema, debe prestarse atención a las lecciones recién aprendidas en el tipo de sistema que queremos realizar. Aunque mu-

7.1 Introducción

Gracias a las sucesivas mejoras en las prestaciones del equipo que poseen los sistemas y a los medios de comunicación, el desarrollo de sistemas cuenta ahora con gran variedad de elementos de diferentes capacidades y propósitos que se localizan a diversas distancias físicas de los usuarios finales.

Para aprovechar estos elementos y brindar mejores prestaciones o reducir costos, se han probado varios esquemas de distribución del trabajo entre los dispositivos a nuestra disposición. También se han implementado arreglos en estos dispositivos, y las capacidades de los mejores arreglos han llegado a ser parte fundamental de los sistemas. Por ello es necesario conocer la forma en que el software distribuye las actividades a realizar en el hardware disponible; esta es una de las tareas que el diseño de software debe llevar a cabo.

En nuestro estudio de los sistemas distribuidos nos centraremos en las facilidades que el sistema operativo requiere implementar para apoyar el desarrollo de este tipo de sistemas.

A continuación daremos un breve repaso de arquitecturas de sistemas distribuidos para ubicar las tareas y las necesidades de estas, las cuales revisaremos después con más detalle debido a su influencia en los sistemas operativos.

7.2 Definición y conceptos

Una **arquitectura de software** es el conjunto de estructuras necesarias para razonar acerca del sistema, incluyendo los elementos de software, las relaciones entre estos y las propiedades de ambos.

Para reutilizar el conocimiento y la información de las arquitecturas más exitosas se han creado conceptos adicionales para definir niveles de abstracción útiles en el diseño de sistemas:

- **Estilos o tipos de arquitecturas**, que se ocupan de las características que buscamos al diseñar o elegir la arquitectura adecuada para una solución o sistema.
- **Patrones de diseño**, que delimitan las soluciones a problemas específicos y recopilan las prácticas que han dado mejores resultados, así como las que deben evitarse en los detalles de la construcción de las implementaciones.

chos aspectos son en general aceptados, estos son susceptibles de ser modificados conforme las tecnologías empleadas avancen y nuevas prácticas se desarrollen, prueben y ganen aceptación general.

♦ **Líder de proyecto.** La elección de una arquitectura adecuada para los requerimientos de un sistema repercute en menores costos de mantenimiento, incluso para aquellas arquitecturas que pudieran generar complejidad adicional y, en consecuencia, un mayor costo inicial. También debe tenerse en cuenta que una vez realizado un sistema, cambiar su arquitectura representa un esfuerzo muy grande.

7.3 Repaso de arquitecturas de sistemas distribuidos

Hay tres aspectos fundamentales en las aplicaciones que son atendidos de varias maneras por las diversas arquitecturas y que usaremos como punto de comparación para su estudio:

1. **La interacción con los usuarios**

   Sin duda alguna, los sistemas, atienden las necesidades de sus usuarios finales, para lo cual deben contar con un mecanismo que les permita interactuar con sus usuarios entregando la información que estos requieren y recibir sus instrucciones. Al respecto, tenemos algunas tareas específicas:

   - La presentación de la información del sistema al usuario.

- La funcionalidad de la interfaz de usuario que permite la interacción de este con el sistema mediante controles gráficos, consola, dispositivos de E/S, etcétera.
- La validación de los datos que son introducidos por el usuario. Para mayor claridad, al revisar las arquitecturas lo separaremos de las verificaciones que podemos hacer para que la información tenga sentido respecto a los datos previamente contenidos en el sistema. Asimismo, nos referimos exclusivamente a la validación de que la forma, el tamaño y las características de la entrada se apeguen a las reglas que definen la entrada esperada, aun si esa información formulada de manera correcta pudiera ser incoherente en relación con el resto del estado del sistema y, por tanto, rechazada al final.

2. El almacenamiento de la información

Conservar la información procesada en el sistema conforme a un conjunto de reglas propias de la aplicación, y a lo largo del tiempo, es fundamental para la mayoría de los sistemas. Esta información contiene el estado de los diversos procesos que se deben seguir, los datos que se requiere procesar y las instrucciones que dictan cómo deben llevarse a cabo las operaciones y los procesos. De hecho, hay que mantener separada la información de control, que indica cómo deben realizarse las operaciones, de la información que contiene los datos que serán procesados. Para preservar esta información el sistema debe cuidar una serie de aspectos que incluyen:

- Almacenar y recuperar la información. Implementar mecanismos para permitir que la información se conserve a lo largo del tiempo.
- Vigilar la integridad de la información para evitar contradicciones e incoherencias debidas a operaciones incorrectas o que hayan presentado errores.
- Controlar el acceso a la información para tener control sobre las operaciones y las colecciones de datos disponibles a distintos usuarios o tipos de usuarios.

3. La implementación de las políticas del sistema

A partir de los requerimientos funcionales de cada sistema, los cuales indican lo que debemos realizar en cada operación implementada, existe una serie de políticas y reglas que afectan múltiples operaciones y cuyo cumplimiento debe vigilarse en todo momento. Estas políticas suelen incluir la información que debe registrarse en bitácoras, los límites que se deben poner a los privilegios de diversos grupos de usuarios, la información que debe almacenarse en formatos cifrados para mejo-

rar el grado de protección que tiene, etc. Las responsabilidades principales que tenemos en este renglón son:

- **Autenticidad de usuarios.** Implementar mecanismos para verificar al usuario del sistema.
- **Control de acceso.** Verificar que quien realice operaciones en el sistema tenga los privilegios necesarios para ello, y, en caso contrario, rechazar las solicitudes.
- **Operatividad.** Implementar las operaciones del sistema con el objetivo de garantizar que estas se realicen cabalmente, o utilizar los mecanismos para recuperar la operatividad del sistema en caso de que se presenten errores durante la operación.
- **Control del flujo de la operación.** Las operaciones individuales de un sistema suelen ser parte de los procesos que definen reglas para la transición y validez de los resultados de una operación que han de conducir las acciones de los usuarios a lo largo de una serie de actividades para lograr sus objetivos de manera cabal y ordenada. Hoy día se implementa la funcionalidad que observa y dirige la secuencia de las operaciones de forma separada de las propias operaciones y de la interfaz de usuario para facilitar el mantenimiento de todos estos elementos. Esta práctica se encuentra descrita en el patrón de diseño conocido como *Model View Controller*.
- **Manejo de bitácoras.** Se debe generar un registro de las operaciones realizadas en el sistema que ayude a posteriores esfuerzos de diagnóstico, detección y corrección del sistema.
- **Implementación de protocolos de comunicación.** La información que viajará por la red deberá ser codificada y transmitida de forma adecuada para garantizar que llegue a su destino. También es importante verificar que la transmisión sea correcta y que se recupere el flujo de información ante fallas. Además, se suelen emplear protocolos de comunicación estándar para facilitar la reutilización de desarrollos anteriores y la interconexión entre sistemas a futuro.

Cómputo centralizado

En esta arquitectura tenemos toda la funcionalidad en un solo equipo. La comunicación por red es mínima y la interacción con los usuarios es indirecta o se realiza mediante dispositivos de entrada y salida que son conectados directamente al equipo.

Figura 7.1 Ken Thompson (sentado) y Dennis Ritchie trabajando juntos en una computadora PDP-11.

Esta no es una arquitectura de sistemas distribuidos. Al contrario, es la arquitectura previa que usaremos como punto de partida y contraste con las demás. Incluye los sistemas que se desarrollaron antes de las redes de computadoras pero que no ha dejado de emplearse. Todos los sistemas que se desarrollen con el objetivo de operar en un único equipo sin usar la comunicación en red con otros sistemas se pueden considerar en esta misma arquitectura.

La característica de las arquitecturas de cómputo centralizado es que todas las tareas de los aspectos se realizan en este único equipo; la interacción con los usuarios, el almacenamiento de la información y la implementación de las políticas están por completo contenidas en el propio equipo (véase figura 7.1).

Clúster de computadoras

Para incrementar la capacidad de procesamiento de las computadoras se buscó no solo aumentar la potencia del procesador sino emplear múltiples procesadores para una sola aplicación; esto dio lugar a las arquitecturas de supercomputadoras que mediante

interfaces especiales permitían la rápida comunicación entre los procesadores y tenían la capacidad de compartir recursos como la memoria o los dispositivos de E/S. Años después se comenzó a explorar el empleo de las técnicas de paso de mensajes y diseño de algoritmos que aprovechan el procesamiento en paralelo usando computadoras personales de bajo costo o servidores autónomos, los cuales, al estar conectados a una red de área local, pudieran intercambiar información a alta velocidad.

Una de las arquitecturas más populares por su versatilidad, bajo costo y flexibilidad es la de los *clústeres bewolf*. Esta arquitectura no impone casi ninguna restricción al tipo de equipos que se empleen y solo requiere una computadora central que pueda enviar trabajos para ser procesados en otras arquitecturas mediante paso de mensajes, ejecución de una terminal remota y, de preferencia, facilidades para reportar errores y administrar recursos. Tiempo después se desarrollaron otras arquitecturas que no requieren un nodo central y que son capaces de distribuir la carga de manera uniforme, evitando la necesidad de tener un nodo central del *clúster* que distribuya el trabajo.

La implementación emplea servicios y bibliotecas de programación que al integrarse en el desarrollo de las aplicaciones permiten la distribución de información para que los distintos nodos de procesamiento reciban tareas que deben ejecutarse y retornen los resultados a un nodo central que compila dichos resultados y, en su caso, reacciona ante fallas de los nodos.

Los *clústeres* de computadoras procuran operar más como una sola computadora con múltiples paquetes de procesador y memoria que como una auténtica colección de computadoras autónomas.

Clústeres asimétricos

Los aspectos del sistema serán atendidos por el *clúster* en su conjunto, pero algunos nodos se dedican a tareas específicas, como las que se mencionan a continuación.

La interacción con los usuarios suele dejarse a cargo de un nodo único, por lo regular el nodo que administra todo el *clúster*.

Se acostumbra almacenar la información mediante sistemas de archivos distribuidos y, en general, se deja la integridad de la información a cargo de las políticas de negocio. También es frecuente dedicar un nodo al manejo de los dispositivos de almacenamiento masivo que darán servicio a las operaciones de sistemas de archivos solicitados por los demás nodos. Como se considera que todos los nodos son parte de una sola aplicación, el control de acceso se limita a trabajar con una única instancia como usuario, lo que minimiza su trabajo al punto de que se implemente simplemente con los privilegios de acceso a los archivos.

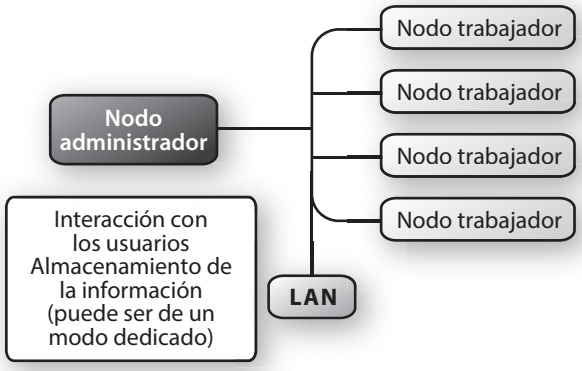


Figura 7.2 Organización de un *clúster* asimétrico.

En cuanto a la implementación de las políticas del sistema, también suele reducirse la parte de autenticidad y de control de acceso bajo la noción de actuar como una única aplicación en un solo equipo. La ejecución del control del flujo de trabajo de la aplicación, la operatividad, el manejo de bitácoras y la implementación de los protocolos de comunicación se encuentran en las tareas que son distribuidas a los nodos de procesamiento del *clúster*.

Cabe mencionar que es posible utilizar un *clúster* como un componente de una arquitectura más compleja, como un sistema federado, o simplemente diseñar aplicaciones que trasciendan las características típicas con nueva funcionalidad y técnicas más recientes. Lo anterior no resta relevancia a las aportaciones y características de esta arquitectura en particular (véase figura 7.2).

Clústeres simétricos

En otras arquitecturas de *clústeres*, las tareas se distribuyen por igual entre todos los nodos. Con frecuencia esto impacta el desempeño pero permite resistir las fallas en cualquiera de los nodos, a diferencia de los *clústeres* asimétricos donde la falla del nodo administrador no puede recuperarse por los nodos trabajadores.

Una arquitectura de este tipo es la soportada por Kerrighed. Este proyecto proporciona una capa de software que simula una supercomputadora debido a su capacidad de procesamiento ya que se ejecuta en todos los nodos del *clúster*. Las aplicaciones que la aprovechan se ejecutan dentro de esta máquina virtual y así tienen acceso a la memoria y a las capacidades de procesamiento conjuntas de todos los nodos (véase figura 7.3).

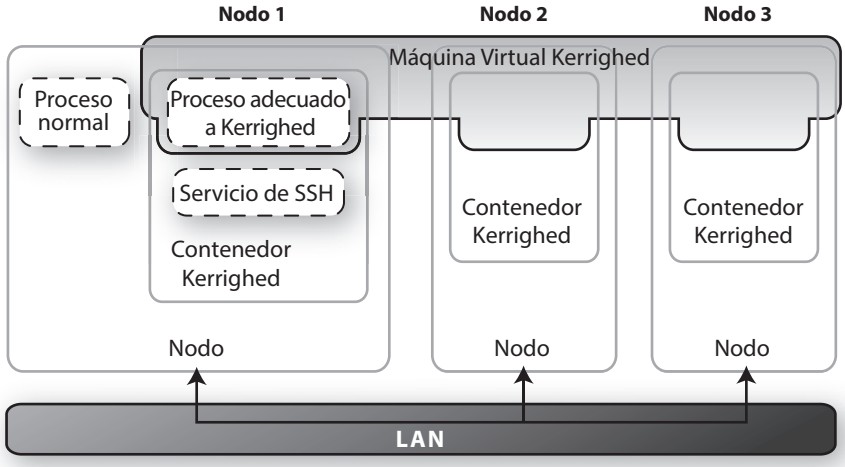


Figura 7.3 Organización de un sistema de máquinas virtuales Kerrighed.

GRID

La arquitectura GRID surge con el propósito de apoyar la colaboración entre universidades, e incluso de estas con empresas, al reunir los recursos de cómputo de diversas instituciones, en sitios remotos, para atender así las aplicaciones con un corto tiempo de vida y aprovechar los recursos disponibles en el momento.

Por regla general, las aplicaciones GRID se construyen con una capa de software que reemplaza y complementa diversas bibliotecas de programación estándar del sistema operativo; estas permiten a las aplicaciones aprovechar los recursos de las computadoras que participan en el GRID con muy pocas modificaciones respecto a una aplicación de procesamiento paralelo que opere en un solo equipo. La funcionalidad que ofrecen estas API no es interactiva y los sistemas se limitan a hacer procesamientos paralelos de datos obtenidos de un repositorio central o de un sistema de archivos distribuido, lo que reduce la capacidad de intercambio de datos entre nodos respecto a los esquemas de memoria compartida o de interfaces dedicadas de equipos de supercómputo. Se recomienda usar GRID cuando los nodos tienen un alto nivel de autonomía y puede entregárseles un paquete de datos o información que procesarán con un mínimo de interacciones con los otros nodos.

Como las redes GRID carecen de una administración central, no es posible garantizar que todos los nodos participantes respeten los compromisos que adquieren, por lo que en las aplicaciones que emplearán recursos de otros dominios se recomienda implementar mecanismos con el objetivo de verificar que los resultados entregados sean correctos y para recuperarse de las fallas ocasionadas por nodos que abandonen el GRID durante el proceso.

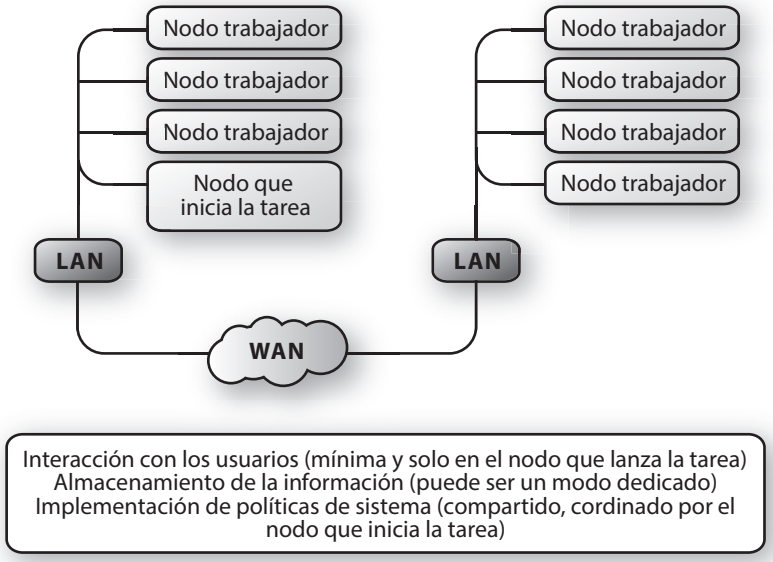


Figura 7.4 Organización de computadoras en un sistema de información GRID.

Cabe destacar que la seguridad del sistema debe considerar que en caso de emplear nodos de otras redes, algunos de los participantes podrían ser incluso desarrollos realizados para suplantar a nodos auténticos y para perturbar la operación, por lo que, de acuerdo con las características del sistema, debe considerarse limitar la colaboración solo a nodos de confianza, usar mecanismos de cifrado y firma para los nodos participantes, o mantener solo un nivel de redundancia que permita verificar que los resultados que entregue cualquier nodo sean correctos, al coincidir con los resultados generados por otro nodo independiente.

Algunas compañías, como IBM y Oracle, proporcionan el apoyo comercial de equipos de cómputo en redes GRID para atender aplicaciones desarrolladas por otros. A esto se le denomina ***cómputo como servicio*** (Utility computing), y aunque está siendo reemplazado por el término ***software como servicio*** (SaaS, Software as a service), sigue siendo un esquema de fácil acceso al cómputo paralelo para las empresas y universidades que requieren este tipo de multiprocesamiento (véase figura 7.4).

Cliente-servidor

La arquitectura cliente-servidor se basa en el concepto de tener una máquina encargada de salvaguardar la información y las políticas de negocio. A esta máquina la lla-

maremos servidor, y a diferencia de los *mainframes,* utiliza un conjunto de máquinas de menores capacidades que se encargan de manera exclusiva de la interfaz e interacción con los usuarios.

Gracias a su simplicidad y al soporte que recibe de los fabricantes de bases de datos relacionales, es uno de los modelos de sistemas distribuidos más comunes. Tiene dos tipos de nodos que se reparten las responsabilidades, como se muestra en la figura 7.5.

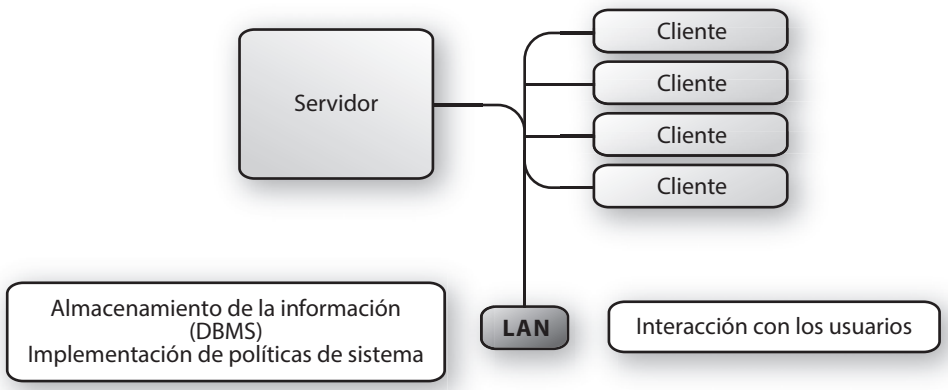


Figura 7.5 Organización de una arquitectura cliente-servidor.

Con la adopción de las interfaces gráficas de usuario (ventanas, botones y demás elementos gráficos) se incrementó de manera importante el consumo de la CPU y los recursos de cómputo para la atención de la interfaz de usuario. Aunado a la popularidad de las redes de computadoras, así como a la disponibilidad y al bajo costo de las computadoras personales, se impuso el uso de un conjunto de computadoras personales conectadas por una red de área local a una computadora de mayores prestaciones sobre el esquema del *mainframe* con múltiples terminales, esto debido no solo al costo menor que implicaba sino porque permite mayor funcionalidad para el cliente.

Además, los productos comerciales de bases de datos relacionales que se popularizaron en la misma época incluyen diversas capacidades que ayudan a atender las necesidades sustantivas asignadas al servidor, sobre todo almacenar la información, vigilar la integridad e implementar las políticas de negocio mediante sus procedimientos almacenados, todo ello considerando la posible concurrencia en el uso de la información por diversos clientes con las facilidades de transaccionalidad y de exclusión mutua necesarias para resolver conflictos y condiciones de competencia.

Sin embargo, en principio no se requiere que el modelo cliente-servidor use un gestor de bases de datos relacional o de algún otro tipo, pero sí debe asumir la responsabilidad de salvaguardar la información y aplicar las políticas de negocio.

La principal desventaja de los sistemas basados en el modelo de comunicación distribuida cliente-servidor radica en el mantenimiento de las computadoras donde se ejecutan las aplicaciones que implementan a los clientes del modelo de comunicación, pues estas suelen ser computadoras personales a cargo de los diversos usuarios de la aplicación, pero cada una está sujeta a fallas y variaciones en su configuración que afectan a la aplicación, por lo que la administración de esta debe lidiar con gran cantidad de aspectos relacionados con el soporte de todas las máquinas que ejecuten a las aplicaciones cliente. Esto representa esfuerzos y, en consecuencia, costos que crecen en proporción al número de clientes.

Cliente-abierto

Como una alternativa a la arquitectura cliente-servidor, en la que se aprovechan los avances logrados con los servicios de Internet, como es World Wide Web (Web), la arquitectura de cliente-abierto construye un esquema análogo al cliente-servidor donde la comunicación se realiza mediante los protocolos estándar de Internet como el HTTP, usado en Web, y en vez de desarrollar un programa cliente específico para la aplicación se emplea un navegador de red (Web Browser), o el cliente del estándar que se haya elegido.

Esto tiene la ventaja de que ahorra el desarrollo del cliente, pero también desplaza la responsabilidad de la operación del cliente de la administración del sistema al usuario individual, ya que el contar con un cliente adecuado para el protocolo en cuestión, pasa a ser parte de la administración de la computadora en cuestión y no de la aplicación en particular. Por su parte, la administración de la aplicación debe vigilar que esta sea compatible con los clientes que tienen sus usuarios, a menudo limitando el soporte a un conjunto de estos (por ejemplo, solo a las versiones actuales de los navegadores Web más populares, como Internet Explorer de Microsoft y Firefox, ambos incluidos en los sistemas operativos correspondientes), lo que minimiza la dificultad para los usuarios de obtenerlos y mantenerlos en operación.

Una vez que se emplea la funcionalidad estándar de comunicación, a menudo es conveniente utilizar también software que se haya generado para este en el lado del servidor. A estas aplicaciones con frecuencia se les conoce como servicios y suelen tener la funcionalidad necesaria para ser integrados a aplicaciones con arquitectura cliente-

abierto de modo que entreguen las peticiones a los programas o procesos con la implementación de las políticas de negocio, mientras mantienen la comunicación con los clientes mediante el estándar para el que fueron elaborados (véase figura 7.6).

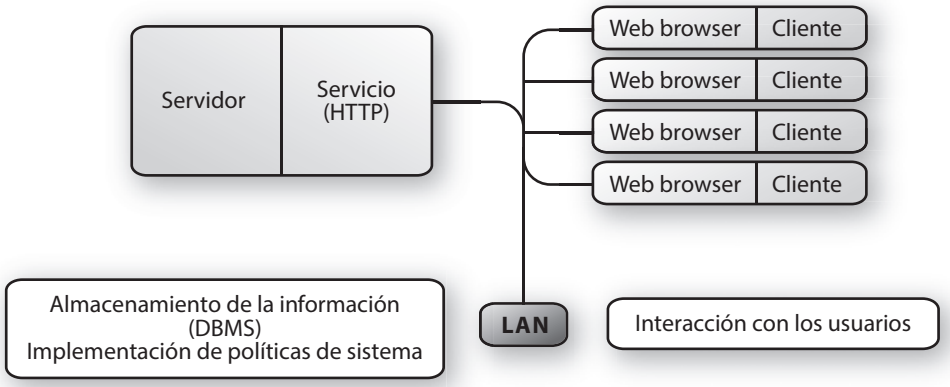


Figura 7.6 Organización de una arquitectura cliente-abierto para aplicaciones World Wide Web.

Debido a que es la alternativa dominante, nos referiremos en lo sucesivo a la arquitectura cliente-abierto basada en el protocolo HTTP empleado por la Web con servidores Apache o IIS.

Una peculiaridad de esta arquitectura consiste en que la funcionalidad de la interfaz de usuario y de validación de información, aunque se ejecutan en el cliente, no se almacenan en él, pues el Web browser es de propósito general, sino que se conservan en el servidor y este lo envía a los clientes para su presentación y ejecución, tanto en HTML como en código interpretable por los navegadores como Javascript.

El hecho de que la información de control y la que contiene los datos se mezclen en el flujo informativo entre el cliente y el servidor es la base de diversas vulnerabilidades de seguridad en estas arquitecturas, por lo que el desarrollo debe ser muy cuidadoso al interactuar con otros sitios Web y es importante vigilar celosamente la forma en la que se realizan las operaciones. La mayoría de las iniciativas que han tratado de almacenar el código de la interfaz de usuario en el lado del cliente, o de incrementar la funcionalidad para estos más allá de lo que ofrece el navegador de red, se han encontrado con severos problemas de seguridad, lo que ha impedido que reemplacen a *cliente-abierto*. A esta propuesta suele llamársele **cliente enriquecido** (Rich client o Fat client) como Flex y Java Web Start.

ACTIVIDAD PROPUESTA

En acción

Realicemos un ejercicio cotidiano e interesante. Usando una computadora conectada a Internet realiza las siguientes acciones:

1. Identifica los navegadores Web que tenga; por lo regular, hay al menos uno incluido en el paquete del sistema operativo, y es común que posteriormente se instale otro. Si ese equipo solo tiene un navegador, instala una segunda alternativa, idealmente gratuita y que sea apropiada a tu equipo.
2. Accede a una página Web sencilla (por ejemplo, la de Grupo Editorial Patria http://www.editorialpatria.com.mx/) con ambos navegadores y observa con cada uno de ellos y usando el monitor de recursos y procesos de tu equipo:
   a. El tiempo que tarda en realizar esta acción desde que se hace la petición hasta que se termina de cargar la página.
   b. La carga de red que genera cada uno de ellos en la primera ocasión en que lees la página, y en las ocasiones posteriores.
3. Reflexiona, de preferencia contrastando tus opiniones con algún compañero, acerca del comportamiento que observes, y responde las siguientes preguntas:
   a. ¿Por qué el consumo de CPU es diferente para los dos navegadores si la página es la misma?
   b. ¿Por qué la carga en la red es menor en las ocasiones posteriores a la primera en que se carga la página o justo después de limpiar el caché del navegador?
   c. ¿De dónde se obtuvo la información que presenta el navegador? ¿Qué tipo de sistema se usó y dónde está ubicado físicamente? (Pista: Basta con que navegues por el sitio, prestando atención a los URL y realizando consultas en el NIC respecto a la dirección del servidor, para que obtengas toda la información relevante.)

Importante

♠ **Arquitecto.** La arquitectura de los sistemas federados se ocupa de coordinar los enlaces entre las diversas aplicaciones presentes, vigilar que la información se mantenga íntegra y controlar la redundancia y la autoridad que los diversos sis-

Sistemas federados

Conforme las instituciones fueron acumulando sistemas informáticos diversos que ayudaban a mejorar su rentabilidad, se comenzó a buscar maximizar la utilidad que estos sistemas proveen. Una alternativa para obtener más valor y reducir costos se basa en integrar diversos sistemas en una sola aplicación con un alcance mayor. En otras ocasiones se logran estos objetivos implementando interacciones entre los sistemas de diversas compañías, con lo que se automatizan las relaciones de negocio.

En todos estos casos tenemos una colección de sistemas autónomos que interactúan entre sí desempeñando roles de clientes los unos de otros, según la operación. Aunque algunas funciones del sistema en su conjunto dependan de que todos estos sistemas estén operando de manera adecuada, en principio cada uno de los sistemas es autónomo, se administra por separado e implementa sus propias interfaces para colaborar con los demás. Otra de sus características es que los diversos sistemas son responsabilidad de instituciones diferentes o de distintas áreas dentro de la institución, y suelen tener implementaciones variadas en redes distribuidas de manera geográfica con cualquier arquitectura propia; incluso, suelen incluirse sistemas tipo *mainframe* con productos especiales que simulan ser terminales.

Este tipo de sistemas suele emplear tecnologías que permiten establecer los enlaces entre pares de sistemas, con protocolos comunes que dependen del lenguaje y tecnologías como la invocación remota de métodos (RMI), normas de comunicación como las de paso de mensajes (SOAP, B2B, EDI, etc.) o productos de mensajería específicos (MQ de IBM, SWIFT, etc.).

Es característico que los enlaces entre sistemas se implementan con todos los detalles y toda la funcionalidad en cada uno de ellos para que puedan interactuar entre sí (véase figura 7.7).

La división de las tareas entre clientes y servidores obedece a las reglas de la arquitectura particular de cada aplicación perteneciente a la federación, donde algunas podrían carecer de clientes auténticos y entonces delegan toda su interacción con usuarios a otras aplicaciones. Además, tratan sus enlaces con ellas como si fueran los enlaces a sus clientes.

temas tienen sobre diferentes segmentos de ella, así como sobre la forma de manejar la división de trabajo de modo que se mantenga la redundancia al mínimo y se proporcione un buen rendimiento.

♥ **Usuario.** Por lo regular, cada uno de los sistemas mantendrá la arquitectura con que se desarrolló originalmente y solo se realizarán pocas modificaciones para que opere de manera armónica con el resto de las aplicaciones del sistema. Esto permite aprovechar aplicaciones en sistemas nuevos y prolongar su tiempo de vida, por lo que resulta muy importante tomar precauciones durante el desarrollo a fin de garantizar la habilidad futura de la institución de proporcionar el mantenimiento correcto.

♣ **Líder de proyecto.** Los sistemas federados se han usado con éxito por varios años y durante su operación han demostrado ser muy sensibles a los cam-

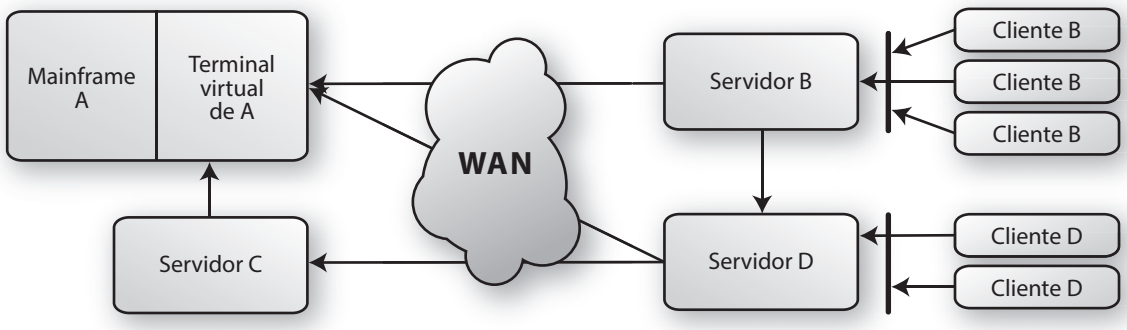


Figura 7.7 Organización ejemplo de sistema federado.

bios que de forma natural tienen los sistemas a lo largo de su vida. Cuando un servidor cambia su dirección de red o cuando se hacen pequeños cambios en la funcionalidad para responder a las necesidades de la institución, se vuelve muy complicado predecir los efectos que estas modificaciones tendrán sobre la totalidad de la federación de sistemas; de hecho, a menudo no se puede prever y se debe absorber el costo de adaptar los demás sistemas afectados hasta después del cambio, con las fallas, tiempos de suspensión de servicio y retrabajos que eso implica.

A la habilidad de una institución para concentrar la información sobre las relaciones entre sus sistemas y actuar en consecuencia se le conoce como **gobierno**, o **gobernance**, pero hasta hoy no ha recibido una atención formal por parte de muchas empresas y sigue siendo una fuente de fallas y costos inesperados muy importante.

Importante

♥ **Docente.** Uno de los protocolos estándar más populares para describir e implementar los servicios es el llamado SOAP (Simple Object Access

Service Oriented Architecture

Esta arquitectura surge debido al éxito y la fragilidad de los sistemas federados, así como de la búsqueda de estrategias tecnológicas que permitan la interoperabilidad de sistemas basados en plataformas distintas que reduzcan la fragilidad de los enlaces punto a punto desarrollados en los sistemas que participan en la federación ante los cambios constantes que requieren dichos sistemas.

SOA (*Service Oriented Architecture*) parte del concepto de servicio de negocio, que se define de la siguiente manera:

> **Servicio de negocio.** Implementación de un conjunto de operaciones con un alcance bien definido que aporta valor a los clientes finales, ya que realiza tareas que son autocontenidas en su alcance, son reutilizables y operan con base en estándares como aquellos de Servicios Web.
>
> La arquitectura SOA se encarga de establecer las técnicas y herramientas necesarias para definir, describir, enlazar e integrar dichos servicios.

Al igual que las arquitecturas federadas, la división de responsabilidades entre los servidores y sus clientes obedece a la arquitectura particular de cada aplicación, aunque el uso de protocolos de comunicación estándar propicia que los servidores se desarrollen con los objetivos de cliente-abierto en mente.

La restricción adicional que SOA impone a estos servicios radica en que los grupos de operaciones que se incluyan en un servicio de negocio deben estar contenidas en él, garantizando que en caso de emplearse otros servicios de negocio de la aplicación o de usarse las mismas operaciones en otra sesión, no deberá haber conflictos entre dichas operaciones. Para ello, no podrán hacer suposiciones sobre el tipo de "clientes" que está realizando las peticiones más allá de lo especificado en la definición de las propias operaciones.

Peer to peer

Las redes *peer to peer* (entre pares) son una federación de procesos que actúan como clientes y también como servidores para los demás miembros de la federación. Estas redes han ganado popularidad en especial para aplicaciones de manejo y distribución de contenidos, pero también han sido útiles en servicios de mensajería, telefonía y otras.

El principio de las redes *peer to peer* consiste en reconocer a cada proceso participante de la federación, llamado nodo, como un proveedor potencial de servicio para los otros, mientras que busca y a su vez obtiene el servicio que requiere de los demás procesos. Con ello se construye un grafo en donde cada proceso representa un nodo y los servicios que se prestan entre sí constituyen las aristas.

Protocol), el cual se logró como parte de un acuerdo informal entre Sun y Microsoft, debido al mutuo interés de ambos fabricantes de sistemas en que los servidores basados en UNIX y Java pudieran interactuar con los clientes y servidores de Windows. Esto fue adoptado muy pronto por IBM, con lo que se ha constituido como un estándar *de facto* en la industria.

Hay dos tipos principales de redes *peer to peer* que se distinguen por la forma en la que uno de los miembros encuentra a los demás y por los servicios que proporcionan:

- **Simétricas.** Todos los nodos participan proporcionando el servicio de la aplicación y ayudando a los demás miembros a ubicar a los proveedores presentes en ese momento.
- **Asimétricas.** Algunos nodos se especializan en el servicio de descubrimiento (en ocasiones realizándolo de manera exclusiva) y otros en proporcionar y consumir el servicio en cuestión.

Por ejemplo, las redes Gnutella, en su primera versión, para compartir archivos tenían a cada proceso proporcionando el servicio de transferencia de los archivos almacenados en el disco de la computadora en la que operaban de forma que pudieran ser copiados a los equipos de quien estuviese interesado en ellos.

Para obtener nuevos archivos y permitir que otros equipos encontraran los presentes para ese nodo, se dependía de lograr conectarse con otros nodos que ya estuviesen en la red Gnutella; al encontrar alguno (por ejemplo, en la red de área local), se solicita un listado de otros clientes para definir un número predeterminado de conexiones estables por las que se intercambian las peticiones de búsqueda que han de propagarse por toda la red. Con esto se accede casi a todo el acervo de los nodos conectados en ese momento. Ante las peticiones de búsqueda que se reciben, se responde con los elementos del acervo local que puedan coincidir con el criterio y se propaga la búsqueda a los demás nodos con los que se tiene conexión; si se reciben búsquedas repetidas, se descartan las posteriores ocurrencias. La red Gnutella es una aplicación *peer to peer* simétrica pues opera de la manera aquí descrita.

En una versión posterior, cuando un nodo detecta que tiene una capacidad de ancho de banda superior a los pares que lo rodean se denomina a sí mismo como un hipernodo y toma un rol diferente. En presencia de hipernodos, los nodos en la red Gnutella no buscan enlazarse con otros nodos, sino exclusivamente con otros hipernodos y no propagan los criterios de búsqueda entre sí, con lo que, en efecto, delegan las tareas de descubrimiento de otros nodos en los hipernodos. Así, estos logran administrar de mejor manera la propagación de búsquedas, el almacenamiento temporal de los resultados y reducen el reenvío redundante de mensajes.

Retomando las responsabilidades que deben repartirse entre los participantes de una arquitectura, tenemos que en las redes *peer to peer* simétricas todos los nodos realizan todas las tareas, dividiéndolas por la parte del acervo o carga de trabajo que llevan y no por las responsabilidades.

En el caso de las redes *peer to peer* asimétricas pasa algo muy parecido, pero puede especializarse a algunos nodos en los servicios de descubrimiento, los que no son una necesidad común de todas las arquitecturas.

Importante

♠ **Líder de proyecto.** Las redes *peer to peer* ofrecen una flexibilidad y capacidad de crecimiento sobresalientes; sin embargo, la falta de control sobre los nodos genera una serie de posibles fallas de seguridad, las cuales han evitado que este tipo de arquitectura tenga una mayor aceptación en sistemas con fines comerciales.

♥ **Docente.** El cómputo masivamente paralelo ha atraído la atención de diversos autores de ficción debido a que comienza a aproximarse en complejidad y tamaño a los cerebros humanos. En razón de la tendencia natural de considerar a fenómenos tecnológicos complejos como si fueran personas, es natural preguntarse si estas aplicaciones e Internet pudieran exhibir conductas y cualidades humanas. Es correcto plantear estas interrogantes y disfrutar la ficción en torno a temas como "la singularidad" o los

Masivamente paralelo

Es una plataforma desarrollada con el objetivo de aprovechar los recursos de procesamiento presentes en las computadoras personales comercializadas de manera masiva y con acceso a Internet.

Para estimar la capacidad de procesamiento accesible por Internet podemos mencionar que la cantidad de servidores conectados a esta red para enero del año 2000 fue de 72 millones, número que siguió creciendo hasta llegar a 732 millones en 2010. A partir de estas cifras, proporcionadas por el Internet Sistems Consortium, podemos estimar que el número de computadoras utilizando servicios de este tipo se encuentra alrededor de los cientos de millones y está creciendo de manera vertiginosa, luego de saturar el mercado de las computadoras personales gracias a la popularidad de los smartphones, los cuales son conectados a Internet mediante redes inalámbricas y cuentan con una considerable capacidad de procesamiento a pesar de sus limitaciones de ancho de banda, potencia y capacidad de almacenamiento.

Una de las primeras aplicaciones en intentar aprovechar estas capacidades fue SETI@Home en que se dio a la tarea de analizar señales

captadas con radiotelescopios en busca de las que pudieran provenir de civilizaciones extraterrestres. Hoy día, cuenta con más de millón y medio de usuarios registrados y se unió a una plataforma llamada BOINC, que permite a quienes tienen acceso a Internet participar de manera voluntaria en múltiples proyectos de investigación, incluyendo a SETI@Home.

Las aplicaciones que buscan aprovechar este tipo de plataformas deben asegurarse de proteger la información procesada, mediante redundancia y diversas salvaguardas de la intermitencia de los nodos, e incluso de nodos que usen un software desarrollado para perturbar la operación del sistema en su conjunto.

También existen plataformas de cómputo masivamente paralelo orientadas a propósitos ilegales y criminales que representan una de las amenazas de seguridad más serias en la actualidad. Estas se conocen como *bot nets* y aprovechan los recursos de los equipos de cómputo de millones de personas para insertarse en dichos equipos sin la autorización de los usuarios, mediante diversos mecanismos que suelen aprovechar la vulnerabilidad de seguridad en los equipos. Una vez instalados, los programas de un *bot net* a menudo quedan latentes o dedicados solo a propagarse hasta que reciben instrucciones sobre las tareas que deben realizar. Revisaremos con más detalle este tipo de sistemas en el tema de seguridad.

avances de las inteligencias artificiales, sobre todo para despertar el interés acerca de los auténticos temas técnicos y científicos relacionados. Pero hay que asegurarnos de separar el terreno de la ficción y de la especulación de las capacidades de la tecnología actual y, en especial, marcar los límites prácticos en cuestiones como consumo de energía o almacenamiento de la información, así como destacar la gran diferencia que hay entre la forma en que opera una computadora digital y un cerebro.

♣ **Usuario.** Dirigir el desarrollo de sistemas lejos de las especulaciones es importante tanto para hacer un manejo adecuado de las expectativas de los usuarios como para evitar caer en temores infundados por la falta de comprensión que en general rodea un tema complejo como este.

Cloud computing

Los sistemas alojados en "la nube" son aquellos en los que la infraestructura, sobre todo de los servidores, no se encuentra en un conjunto de computadoras específicas que requieran ser administradas por quien construyó el sistema, sino en plataformas de servidores contratadas a enormes centros de cómputo según la demanda. Estos centros ofrecen un número variable de servidores compartidos, de capacidades y software de servidor en paquetes estandarizados para albergar a las aplicaciones, cuando lo requieran los clientes y cobrando según la carga atendida.

Tiene su origen en los modelos de comunicación distribuida de cliente-servidor y cliente-abierto, en las que la escalabilidad de los sistemas siempre ha sido problemática cuando la carga de trabajo y el número de usuarios comienza a incrementarse.

Al llegar a determinados niveles de carga, la estrategia de distribuir la carga de procesamiento de los clientes funciona bien; sin embargo, cuando un solo servidor no

es capaz de manejar todas las peticiones de los clientes es necesario resolver diversos problemas adicionales, como los siguientes:

- Mantener un estado único y coherente de la información mientras esta es afectada por las peticiones de los clientes en servidores distintos.
- Distribuir la carga de trabajo.
- Incrementar de manera acorde la capacidad de los servidores, del ancho de banda y otros recursos consumidos cuando el número de clientes va en aumento.
- Mecanismos implementar para minimizar el costo en periodos de poca carga de trabajo.
- Mecanismos implementar para mejorar la atención de las peticiones de usuarios que se encuentren en posiciones geográficas remotas.

Servicios como Amazon o Google permiten a sus clientes utilizar un número variable de servidores, alojados en centros de cómputo ubicados por todo el mundo. Con esto se logra mejorar los tiempos de respuesta para aplicaciones que requieren atender millones de usuarios, ubicados en distintos países y que emplean muchos tipos de dispositivos distintos para conectarse a Internet y emplear la aplicación. Estos servicios aprovechan las capacidades de los servidores para atender múltiples aplicaciones de forma simultánea, al contar con mecanismos para establecer o retirar los componentes del sistema de información en nuevos servidores de forma rápida y a menudo automatizada en respuesta a los niveles de carga de trabajo y a los esquemas de cobro. Para quienes quieren usar estos servicios para alojar sus sistemas de información, el cobro se realiza en función de mediciones de los recursos consumidos como CPU, ancho de banda, etc. Así, el cobro puede realizarse en proporción de los recursos que realmente fueron consumidos por las aplicaciones.

La organización y las características de los servicios que se contratan en un sistema en la nube determinan el tipo de modelo que se emplea. Por ejemplo, un servidor, como una computadora que se desea administrar y que proporcione acceso a la administración del sistema operativo, se conoce como un modelo de Infraestructura como servicio (IaaS).

Por el contrario, si se desea acceso al software de servidores estándar de servicios de Internet, como podría ser un servidor HTTP, y de bases de datos para alojar una aplicación de cliente-abierto, sin requerir control del servidor en los que se alojen, hablamos de un Software como servicio (SaaS), en el que se pueden implementar los modelos de pago más dinámicos basados en el consumo de recursos.

Importante

♦ **Líder de proyecto.** La flexibilidad de los sistemas en la nube ha permitido que aplicaciones orientadas a teléfonos inteligentes puedan escalar en poco tiempo para tener una cobertura global sobre millo-

Además, se debe considerar seriamente el rango en el que es permisible manejar la transferencia de los datos de la aplicación. Con frecuencia hay requerimientos prácticos, de seguridad o legales, para que la información de un sistema no abandone, por ejemplo, el país en el que se utiliza la aplicación, sobre todo para limitar el impacto que pueda tener una falla de seguridad. Esto ha llevado a la creación de centros de cómputo capaces de dar los servicios de la nube, pero que se encuentran dedicados y contenidos dentro de empresas individuales y que son administrados por proveedores ajenos a la empresa que atienden, y que se conocen como nubes privadas.

Una de las herramientas que ha ayudado a la flexibilidad de los servicios de sistemas en la nube es la virtualización. Esta consiste en simular el hardware de una computadora con la finalidad de ejecutar un sistema operativo, junto con las aplicaciones que contenga, mediante un proceso en un servidor con capacidades mayores a las de la máquina que simula. Esto conlleva un consumo adicional de recursos de modo que los servidores así simulados, o virtuales, no pueden tener las capacidades del servidor anfitrión incluso si consumen toda su capacidad. Sin embargo, permite que los servidores virtuales puedan ser activados, desactivados, migrados y, en general, controlados a un nivel de velocidad y automatización que no es posible lograr con las máquinas físicas. Esto ha resultado en un elemento muy importante, en especial en el esquema de IaaS.

nes de dispositivos sin necesidad de que estas aplicaciones hagan enormes inversiones en infraestructura de servidores en sus inicios, cuando tienen pocos ingresos y están apenas dándose a conocer.

♠ **Arquitecto.** Para poder aprovechar la escalabilidad que la nube puede ofrecer, es vital que el diseño de las arquitecturas de las aplicaciones considere seriamente la paralización y el manejo distribuido de la información que va a realizarse; de lo contrario, aunque pueda acceder a grandes cantidades de recursos no será capaz de atender las necesidades de sus usuarios por requerir concentrar toda la carga de trabajo en nodos de procesamiento, almacenamiento de información o mediante pasos particulares en la secuencia que no toleren el uso paralelo y deban ser serializados.

7.4 Soporte a sistemas distribuidos

Las arquitecturas antes mencionadas comparten una serie de tareas que requieren del apoyo del sistema operativo. A continuación revisaremos las facilidades más comunes, las cuales son conocidas como mecanismos de comunicación distribuida, que se implementan para ello.

Comunicación entre procesos remotos

Una de las necesidades principales que surgen con modelos de comunicación distribuida como cliente-servidor es la de poder utilizar la funcionalidad que se encuentra en otro equipo para transformar información que es introducida por el usuario en el caso de un cliente, o para presentar resultados y obtener indicaciones en el caso de los ser-

vidores. Para ello hay diversos mecanismos que suelen emplearse con frecuencia y que parten de brindar una interfaz uniforme para establecer comunicación entre los procesos localizados en equipos distintos.

Invocación remota de métodos (RMI)

La invocación remota de métodos tiene por objetivo permitir que un proceso que actúa como cliente utilice funcionalidad que se encuentra definida en otro proceso remoto, al que llamaremos sirviente, y que hace las funciones de servidor.

En el caso de los sistemas federados, puede tratarse de un equipo de cómputo de grandes prestaciones que no da servicio a los usuarios finales y que implementa una interfaz de cliente para dirigir consultas a otro servidor. En lo que concierne a RMI, el propósito del proceso no importa, por lo que solo resulta relevante para determinar quién es cliente y quién es el servidor, pues el servidor define la funcionalidad y el cliente genera las peticiones para utilizarla.

En su forma más sencilla, las relaciones cliente-servidor se implementan con las características específicas de un servidor y de un cliente en un enlace punto a punto. Como esto resulta muy frágil debido a la tendencia de los sistemas de modificar algunos parámetros, como la dirección IP, con el tiempo a menudo se genera un tercer elemento que permite verificar la información para ayudar a los clientes a actualizarla y ubicar al servidor. Esto se conoce como servicios de descubrimiento o de directorio.

Una de las interfaces de programación y utilerías más influyentes en lo que se refiere a la implementación de RMI es la de CORBA. Esta fue desarrollada para proporcionar servicios de invocación remota de métodos en la plataforma C y, conforme a la filosofía de este lenguaje, permite la invocación de funciones simples que no pertenecen a clases ni objetos en muchas de sus implementaciones.

CORBA utiliza un proceso por cada nodo, conocido como el ORB (Object Request Broker), que se encarga de toda la comunicación de los objetos locales con los objetos que se encuentran en los demás nodos, cada uno de los cuales tendrá también un ORB. Así, cada cliente CORBA solo requiere implementar en los programas la API que comunica al cliente con el ORB local, evitando que cada uno deba implementar los mecanismos de comunicación por red y serialización de los parámetros.

Los programas que implementan la funcionalidad que se encuentra a disposición de los clientes, realizando las tareas de servidor, se conocen como sirvientes (servants) porque cada proceso solo requiere implementar un conjunto de operaciones es autocontenido, convive con otros procesos en el mismo equipo y deja al ORB local la tarea de comunicación y serialización. Como tal, solo toma la parte de implementación de

políticas específicas de un servicio y no se puede decir que tenga toda la implementación del servidor.

Para verificar durante la compilación que el cliente invoque la funcionalidad del sirviente con las firmas adecuadas, incluyendo los nombres de las funciones y sus parámetros con los tipos de todos ellos, se utilizan archivos que contienen la declaración de las funciones sin el código de la definición de la funcionalidad. Estos archivos se conocen como esqueletos (skeletons) y deben estar disponibles y consistentes en los proyectos del cliente y de los sirvientes (véase figura 7.8).

En la figura 7.8 se muestra el flujo de una petición CORBA. El proceso del cliente entrega los parámetros y el tipo de petición que debe realizarse al ORB mediante la API

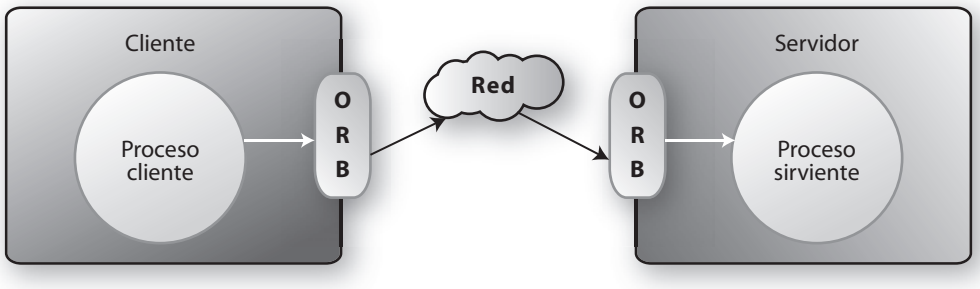


Figura 7.8 Organización de una aplicación CORBA con un único sirviente y solo un cliente.

local. El ORB entonces realiza la conexión con el ORB del servidor que recupera los parámetros y el tipo de petición de la serialización que transportaron los ORB y hace la invocación al proceso del sirviente. Después se recupera el resultado, lo serializa y se transmite de retorno en dirección contraria a la del envío de la petición.

El servicio de directorio permite a diversos sirvientes registrar las características del servicio que proporcionan para que los clientes puedan recuperar la información necesaria para establecer conexiones con ellos. Este directorio se construye con una red de sirvientes que intercambian la información de los servicios registrados y hace que sus clientes locales recuperen la información de todos los nodos del directorio que configuraron para cooperar entre sí.

Las versiones más modernas consideran que C++, la versión que incluye la orientación a objetos de C, contiene la notación de programación orientada a objetos, y por ello requiere cada vez más el uso de clases para agrupar los métodos que serán utilizados de modo remoto.

También han surgido otras plataformas de desarrollo que buscan simplificar la labor del programador, en especial respecto a la consistencia entre las interfaces que los objetos a invocar exponen y a la administración de los servicios de directorio. Una de las más populares es la que Java incluye en su plataforma J2EE que tiene diversos niveles y mecanismos, pero para nuestro estudio es suficiente mencionar el llamado RMI (Remote Method Invocation), que es uno de los primeros y más simples.

En RMI, una clase puede exponer diversos métodos de un objeto remoto como si fuera local, pero al construir la aplicación no se requiere conocer con exactitud las características del objeto a emplear ya que la descripción de la clase a la que pertenece, la que define las propiedades de sus métodos y atributos estáticos, es transmitida durante la ejecución para permitir al cliente reconocer y explotar las características del objeto, o clase, que actúe como servidor sin importar que no corresponda con exactitud a las que tenía durante la construcción, o incluso en ejecuciones anteriores. Para obtener las características de las clases a menudo se les publica en un servidor HTTP.

Para dar certeza a la operación sin conocer las propiedades específicas de la clase servidor se utilizan *interfaces*, que actúan como los esqueletos (skeletons) para verificar que la solicitud del cliente sea atendida por el servidor; sin embargo, como ya se mencionó, RMI permite que la clase servidor varíe en versiones posteriores sin que se requiera compilar de nuevo el cliente, lo que amplía la flexibilidad a costa de posibles divergencias en el futuro.

A diferencia de CORBA, donde cada equipo tiene un ORB, en RMI cada máquina virtual Java (JVM es el proceso que define una máquina virtual donde se ejecutan los programas Java) tiene el equivalente de un ORB y un servicio de directorio locales a la JVM. Para utilizar un objeto en otro equipo se requiere especificar la dirección IP y el puerto en la que la JVM remota espera las peticiones a sus clases con RMI. Con esto se consigue un conjunto de servicios de directorio con poca coordinación pero que no requieren de una configuración previa.

El servicio de directorio local a una JVM se conoce como el Registro de RMI (RMI Registry), como se muestra en la figura 7.9.

En el caso de la plataforma de desarrollo para programas en Windows Visual Studio, se cuenta con facilidades similares a las de CORBA y a las de RMI; sin embargo, estas están siendo reemplazadas por servicios para dar mayor importancia a la documentación e interoperabilidad de los componentes de los sistemas.

Las plataformas móviles como Android también suelen hacer especial énfasis en el desarrollo de clientes, confiando el desarrollo de servidores a otras plataformas. Por ello, su API de programación define un aspecto particular para los componentes de una aplicación que se dedicarán a manejar la información del sistema para que la interfaz

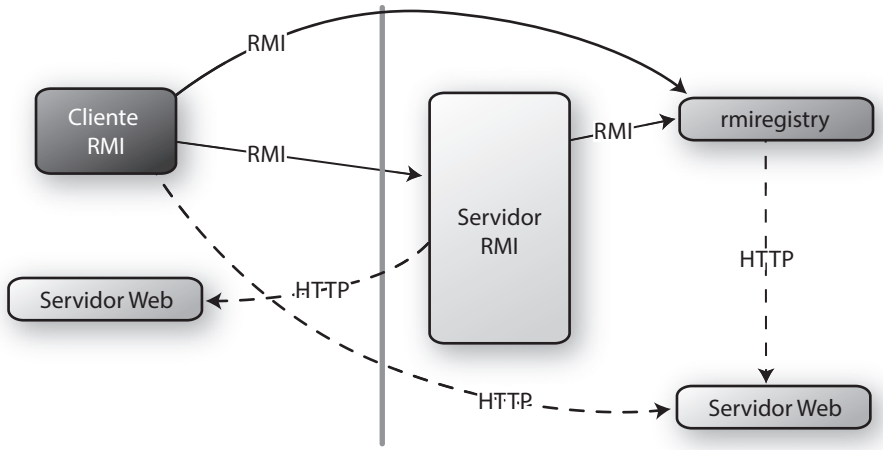


Figura 7.9 Llamadas en una operación RMI.

de usuario (otro aspecto) pueda explotarlo. La implementación también suele orientarse a servicios para aprovechar los estándares de comunicación y de documentación de los servicios presentes.

Paso distribuido de mensajes

Cuando tenemos múltiples sistemas que requieren intercambiar mensajes entre sí, resulta inconveniente pasarlos solo mediante enlaces de un sistema a otro. Resulta más factible usar implementaciones que permitan replicar la información contenida en los mensajes a una serie de instancias.

En su forma más sencilla, el paso de mensaje se establece entre dos sistemas diferentes que desean transferir información de uno de ellos de modo que el segundo pueda recibirlos, sin conocer de antemano cuándo serán enviados y logre procesar la información. El sistema que envía solo requiere saber que el destinatario ha recibido el mensaje, para lo cual se genera un mensaje de reconocimiento (conocido como Acknowledge, o Ack), y en ocasiones recibir algo de información en retorno.

El tipo de envío en el que solo se necesita confirmar que el mensaje fue recibido se conoce como **asíncrono**. En este envío se desconoce el momento en el que el mensaje se procesa, incluso si posteriormente recibe a su vez algún mensaje o alguna otra indicación de que el procesamiento ha terminado.

Por otro lado, la operación síncrona es aquella en la que se espera recibir información de retorno que indique el resultado del procesamiento (aunque sea parcial) del mensaje enviado, ya que se tendrá información sobre el momento en que se procesó el mensaje. Este esquema solo resulta conveniente para operaciones que no demoren mucho, pues

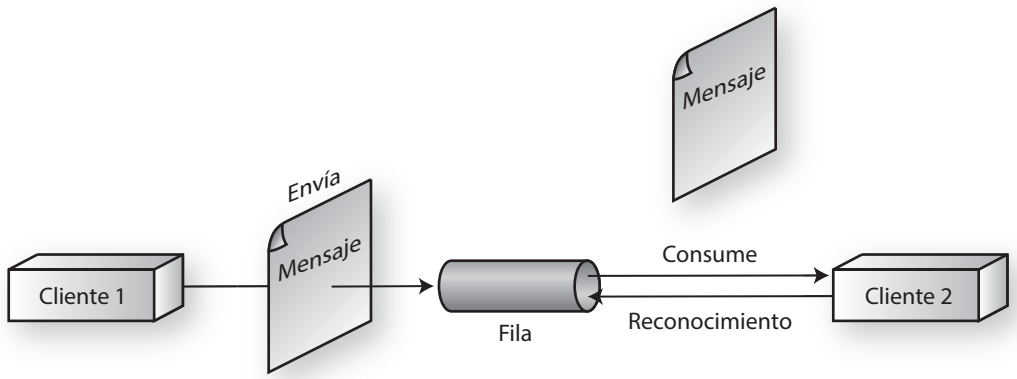


Figura 7.10 Flujo de mensajes en una fila.

de lo contrario el tiempo de espera puede afectar de manera negativa el rendimiento de las aplicaciones y el flujo de uso del sistema para el usuario final (véase figura 7.10).

Por otra parte, también podemos manejar el intercambio de mensajes de modo que la información llegue a más de un sistema destino. Una de las formas más comu-

ACTIVIDAD PROPUESTA

En acción

Enviemos unos mensajes. Toma el servicio de mensajería instantánea de tu preferencia, como por ejemplo WhatsApp, Viber, Line, BBM (Blackberry Messenger), SMS (Shot Message Service con su teléfono celular), mensajes en Skype, Google Hangouts (antes Google Talk), Facebook, o alguna otra similar.

Envía un mensaje y observa los siguientes aspectos:

1. ¿Qué servicios de telecomunicaciones utiliza esta aplicación para transferir la información?
2. ¿Cómo sabes, como usuario, en qué momento puedes o no enviar un mensaje?
3. ¿Puedes determinar si el mensaje se transmitió con éxito? ¿Si ha sido recibido por el destinatario? ¿Si el destinatario ya lo ha leído?
4. ¿Cómo puede el destinatario asegurarse de que el mensaje fue enviado por quien dice ser el remitente? ¿Qué tan seguro puede estar?
5. ¿Qué tan a menudo se demora un tiempo largo la entrega de un mensaje? ¿Qué causa estas demoras?

Por último, compara las respuestas que obtengas con la información respecto a diferentes servicios de mensajería instantánea. Observarás que las capacidades varían pero se mantienen fieles a los lineamientos del paso de mensajes entre aplicaciones de donde tomaron su inspiración original.

nes de esto es hacer llegar un mensaje a un conjunto de procesos para que cada uno lo procese de manera distinta o con un propósito diferente. Para esto se ha establecido lo que se conoce como **colas de mensajes** (message queue), las cuales deben almacenar los mensajes para replicar todo el tiempo que sea necesario a fin de asegurar que se han transmitido a la totalidad de los procesos interesados. Asimismo, cada uno de estos procesos debe iniciar una sesión e indicar que desea recibir los mensajes de esa cola en particular. A esto se le conoce como **suscripción**.

En el caso de JMS, las colas de mensajes se emplean incluso cuando se tiene un único subscriptor para aprovechar la implementación de persistencia y las garantías de entrega de los mensajes para ese único receptor. Si se requiere contar con múltiples destinatarios se tienen facilidades adicionales, como la de generar una suscripción que reserve los mensajes, incluso si el suscriptor está inactivo, de modo que cuando se reactive reciba los mensajes que de otra forma no se le habrían entregado. También se puede hacer que la cola de mensajes sea persistente para evitar que una interrupción en la máquina virtual que alberga la cola genere pérdidas de mensajes. Asimismo, es posible definir el tiempo de vida máximo que un mensaje debe persistir con miras a que aquellos que dejen de ser relevantes y no puedan ser entregados sigan consumiendo recursos (véase figura 7.11).

Otro caso interesante del paso de mensajes es aquel en el que se necesita que el estado de un conjunto de datos se mantenga sincroni-

Importante

♥ **Arquitecto.** Es interesante considerar las necesidades de almacenamiento requerida por las colas de mensajes (ya sea en memoria o en disco) para las persistentes. Si multiplicamos el tiempo esperado para concluir la entrega de los mensajes a todos los suscriptores de la lista, por el tamaño promedio del mensaje, tendremos un estimado de la cantidad de información promedio que nuestra cola deberá consumir para almacenar los mensajes en tránsito. Sin embargo, si queremos garantizar que la operación tenga pocas fallas hay que considerar tiempos máximos de entrega y de tamaño de mensajes, de modo que durante la operación no tengamos errores por saturación en el tamaño de la cola de mensajes. Debido a que en general pensamos en la transferencia de información como un fenómeno con dura-

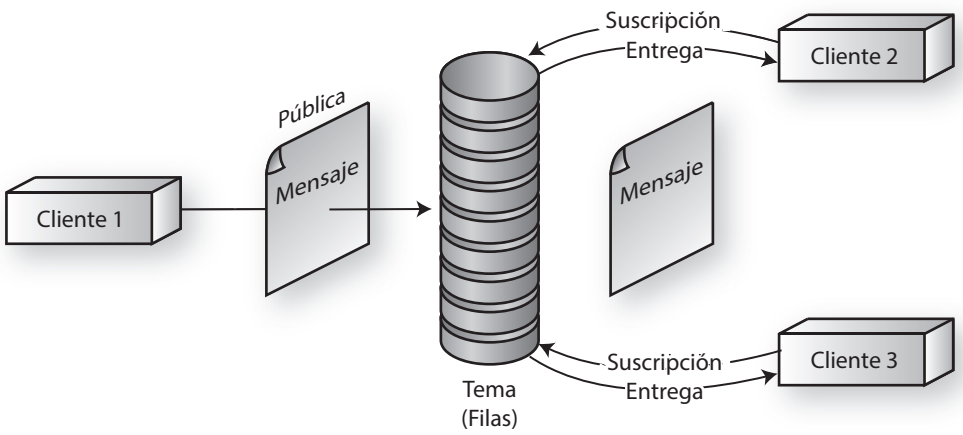


Figura 7.11 Flujo de mensajes en un tema.

ción casi instantánea, solemos perder de vista la inherente necesidad de almacenamiento de información que tiene todo proceso de transferencia de datos; sin embargo, en el caso de las colas de mensajes eso suele generar errores graves en la configuración de los sistemas, por lo que este aspecto debe ser atendido desde la fase de diseño.

♥ **Docente.** El paso de mensajes tiene diversos términos que se usan de formas particulares y que conviene conocer para comprender la documentación:

- **Cliente.** Procesos que participan enviando o recibiendo mensajes.
- **Servidor.** El o los procesos que reciben los mensajes y los almacenan mientras terminan de entregarlos a los clientes destinatarios.
- **Mensaje.** Conjunto de datos que siguen una estructura y tipo predeterminado que serán serializados para ser transferidos de un sistema a otro, pero que no están ligados a implementaciones específicas que los generen o procesen.
- **Fila.** Almacenamiento temporal de un conjunto de mensajes en tránsito.
- **Tema.** Similar a la cola, pero que envía cada mensaje a todos los clientes que se hayan suscrito.
- **Reconocimiento o acknowledge.** Tipo especial de mensaje

zado entre una serie de nodos, donde cada nodo tiene una copia de ese conjunto de datos y puede recibir modificaciones a ellos de forma concurrente. Este es el caso de las sesiones para sistemas Web, donde uno de un conjunto de servidores recibe la siguiente operación que habrá de realizarse en el marco de una sesión, pero se decide no ligar a cada sesión a ser atendida por un único servidor (sticky session).

En estos casos, los mensajes deben enviarse del servidor que recibió el cambio hacia todos los demás servidores, para lo cual hay varias estrategias: enviar un mensaje con la nueva información de sesión a todos los demás servidores mediante una suscripción a un tema de mensajes. Esta estrategia tiene la desventaja de que conforme crece el número de nodos participantes, el número de mensajes se incrementa de manera exponencial, lo que desemboca en una saturación de los recursos de transferencia entre los servidores.

Otra alternativa es establecer una lista circular con los servidores y mandar la actualización de cada uno al servidor siguiente mediante un mensaje y, sucesivamente, pasar el mensaje con la actualización hasta que llega al servidor que dio origen, en el que se desecha el mensaje. Esto puede implicar más tiempo para la propagación de los mensajes y se corre el riesgo de perder estos si el servidor que debería pasar un conjunto de mensajes falla. También debe implementarse algún mecanismo para reconstruir la lista circular y detectar fallas. Por ejemplo, puede hacerse circular un mensaje de verificación que no sea eliminado; en caso de que no se reciba en un determinado tiempo, se puede asegurar que se ha perdido y proceder a activar el mecanismo de recuperación para reconstruir la lista circular y regenerar el mensaje de verificación.

SOA

Como se puede apreciar en el tema anterior, es posible organizar las listas y los temas de mensajes para distribuir estos en una variedad de configuraciones. Sin embargo, el paso de mensajes no contempla facilidades para simplificar la reconfiguración de los enlaces punto a punto entre los servidores de mensajes y sus clientes ni para el procesamiento o transformación de mensajes en el tránsito para aliviar la necesidad de modificar los sistemas o para reunir la información de

diversos mensajes en uno solo en la integración de los sistemas federados.

Para solventar estas necesidades, junto con el apoyo a la documentación y estandarización de las características de los mensajes y operaciones sobre ellos, se utiliza la arquitectura de orientación a servicios (SOA).

que solo indica que un mensaje fue recibido con éxito, pero no informa si ha sido procesado.

La orientación de servicios encapsula la información que intercambian las operaciones en paquetes autocontenidos de información con todos los atributos y, en efecto, envían y reciben un conjunto de mensajes que pueden intercambiarse tanto de modo síncrono como asíncrono, en cuyo caso se conocen como operación de *fire and forget* (lanzar y olvidar), ya que en realidad no esperan por la respuesta.

También es fundamental para SOA emplear un mecanismo estándar que represente las características de los servicios que incluyen las firmas de sus operaciones y los mensajes que emplean. Esto ayuda a que se intercambien, utilicen y verifiquen las características de los servicios que un cliente pretende utilizar y disminuye el acoplamiento entre las distintas partes del sistema, dando mejores prestaciones para actualizar los enlaces.

El equivalente de las colas de mensajes en SOA se implementa con servidores especializados que pretenden ser intermediarios de un conjunto de servicios. Estos se conocen como concentradores de servicios (service bus). Como intermediarios, satisfacen la necesidad de hacer cambios en los servidores o en los clientes ante las modificaciones en los servicios; por ejemplo, si se modifica la dirección de un servidor, basta con hacer el ajuste correspondiente en el concentrador de mensajes.

Asimismo, los concentradores tienen mecanismos especializados para procesar los mensajes y realizar tareas sencillas de transformación, composición de varios mensajes en uno o descomposición de uno en varios, aun con secuencias de tareas y cierto grado de lógica. Están optimizados para que este procesamiento sea rápido con la finalidad de evitar que se requiera gran cantidad de memoria en grandes flujos de mensajes.

Sistemas de archivos distribuidos

Una de las primeras aplicaciones del intercambio de datos por redes fue compartir archivos entre dos equipos, reemplazando a medios de almacenamientos removibles. Conforme la velocidad de transferencia se ha incrementado, se ha hecho más conveniente transferir información por este medio.

Importante

♠ **Arquitecto.** Se mantienen prácticas de intercambio de medios físicos para casos de contingencia o cuando se requiere transferir gran cantidad de información. Estas técnicas en ocasiones se conocen como redes aviares (recordando a la mensajería hecha por palomas mensajeras, que incluso se utiliza en casos especiales). En ambientes urbanos se recurre principalmente a servicios de mensajería.

♣ **Líder de proyecto.** La transferencia de información por medio de una red de computadoras ha progresado mucho, hasta alcanzar velocidades que superan las de los dispositivos de almacenamiento masivo empleados en décadas pasadas, y con posibilidades muy bajas de errores en la transferencia. Sin embargo, no debemos olvidar que esos errores existen, por lo que es imperativo usar protocolos de corrección de errores para evitar alguna falla en la transferencia sobre grandes conjuntos de datos. Las arquitecturas y políticas de manejo de la información requieren considerar estas posibles fallas esporádicas.

Uno de los primeros servicios en proporcionar sistemas de archivos distribuidos, basado en un modelo de comunicación distribuida cliente-servidor, es NFS (Network File System). En este el almacenamiento de la información se realiza en un servidor, con acceso a grandes capacidades de almacenamiento en dispositivos de almacenamiento masivo conectados directamente a él. También cuenta con uno o varios procesos que permiten enviar y recibir información de dichos archivos con diversos clientes que se conecten por red en cualquier momento mediante la dirección de red del servidor y un puerto de comunicación predeterminado (usualmente el 2049).

Por su parte, los procesos cliente se encargan de mantener la comunicación por red para obtener y actualizar la información correspondiente, simulando en la capa de controladores específicos de dispositivo la funcionalidad necesaria para que se exponga a los usuarios y a los demás procesos a un nuevo dispositivo de almacenamiento masivo que en realidad está implementado mediante esta comunicación. Con ello se logra que las demás aplicaciones no requieran estar al tanto ni ser modificadas para aprovechar el sistema de archivos remoto, pues en los sistemas UNIX este se monta como parte del sistema de archivos virtual, al igual que cualquier otra partición activa del sistema.

Para proteger el acceso a la información, NFS soporta diversos mecanismos de autenticación, como la combinación de identificador de usuario y contraseña. Por sí mismo, carece de la capacidad de proteger la transferencia de información para que no pueda ser interpretada en tránsito por agentes ajenos, pero puede aprovechar facilidades de entubado (tunneling) de otros servicios como SSH (Secure Socket Shell).

En la actualidad se han popularizado otras plataformas para compartir sistemas de archivos. Por ejemplo, con frecuencia se usan los sistemas Windows para compartir directorios conocidos como CIFS. Originalmente se configura para solicitar una pareja de identificador de usuario y contraseña, pero también permite compartir carpetas de forma pública. En el caso de sistemas de archivos NFS propicia incluso el control de privilegios de acceso conforme a las normas de ese sistema de archivos. Además, este protocolo se ha implementado en UNIX con el paquete de aplicaciones llamado **samba**. Aunque también cuenta con un modelo de comunicación distribuida cliente-servidor, tiene la posibilidad de delegar la verificación de la autenticidad en un segundo tipo de

servidores de dominio (Active Directory) y permite que los diversos equipos actúen tanto como servidores de sus archivos locales como clientes de los archivos de otros equipos.

Estas primeras implementaciones se orientan sobre todo a compartir archivos dentro de la red de área local, conforme progresó la disponibilidad de enlaces de altas prestaciones entre cedes remotas se buscó que los servidores de almacenamiento pudieran estar más alejados de los clientes, lo que dio lugar a implementaciones como Ceph.

Hoy día se está trabajando en sistemas de archivos que distribuyan un acervo unificado con múltiples cedes de servidores de almacenamiento, pero que tengan la capacidad de usar el principio de localidad para mantener almacenes temporales de información cercanos a los clientes, lo que mejora el rendimiento en el estilo de los sistemas federados. Un ejemplo de estas arquitecturas de sistemas de almacenamiento distribuido es Ceph de IBM. Se orienta a tener una escalabilidad muy por encima de lo que podría conectarse como dispositivos de entrada y salida a un solo servidor, en la escala de petabytes, manteniendo un desempeño óptimo para grandes cargas de trabajo medidas en ancho de banda y operaciones por segundo junto con una gran estabilidad de operación.

En Ceph se divide la carga de trabajo en tres tipos de servidores que se pueden distribuir geográficamente para que queden más cerca de los clientes que deben atender y así mejorar el desempeño del sistema. Los clientes son similares a los de NFS en el sentido de que montan en el cliente un sistema de archivos implementado mediante los servicios de red de Ceph (véase figura 7.12).

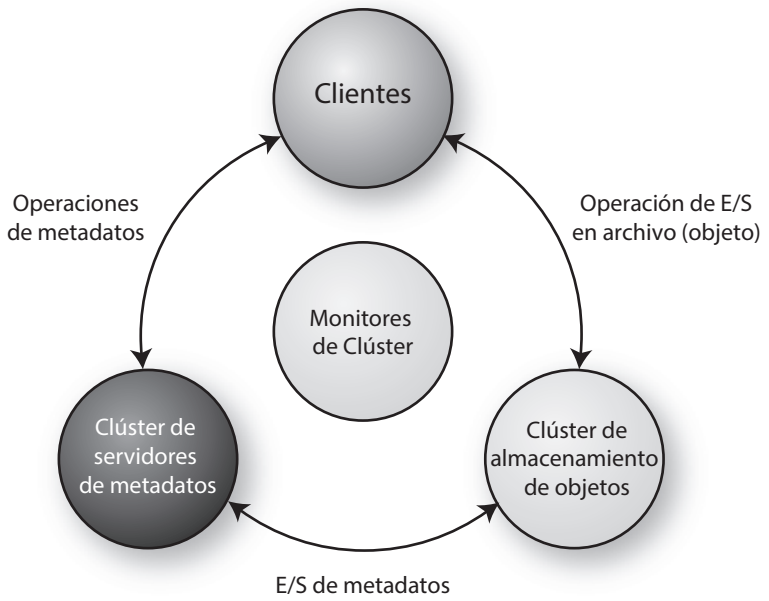


Figura 7.12 Elementos en un sistema de archivos distribuido Ceph.

Los servidores que emplea tienen tres funciones, de las cuales la primera es: almacenar la información, es decir, los archivos (objetos en este caso específico). A diferencia de otras arquitecturas, estos objetos no están almacenados en un único servidor, sino que pueden migrar entre diversos servidores de acuerdo con el uso que de ellos hagan los clientes. Para ubicar los objetos se tiene un segundo tipo de servidor, que permite acceder a la información de sus atributos (metadatos) y ubicación. Como solo manejan esta información, logran responder muy rápido y manejar una mayor parte del acervo de la que se puede almacenar localmente. También resguardan la información de cambios (log), como en los sistemas de archivos de bitácora. El tercer tipo de servidor, que es el de monitoreo, se encarga de validar la actividad de los servidores, en colaboración con los demás nodos de monitoreo, para reconfigurar la red a fin de considerar los servidores que se unen o que dejan de operar.

7.5 Gestión distribuida de procesos

Migración de procesos

Para distribuir la carga de trabajo, los sistemas recurren a una serie de estrategias. Podemos clasificar estas en tres grandes categorías, según el elemento que se transmite:

- **Migración de datos.** Como mencionamos en la sección de sistemas de archivos distribuidos, y en los casos más simples del paso de mensajes, a menudo se necesita transferir información, originalmente contenida en otro de los nodos del sistema, al lugar donde se requiere para su uso.
- **Migración de tareas.** Cuando resulta inconveniente transferir la información, ya sea por sus dimensiones o complejidad de relaciones con otros datos locales, es posible transferir las instrucciones codificadas de la operación que deseamos realizar y quedar en espera únicamente de los resultados. Esta forma de uso es la que usamos en RMI y en el paso de mensajes. Es importante señalar que la implementación de la funcionalidad que empleamos en estas operaciones se encuentra disponible en el servidor que responderá a la petición sin importar el ciclo de vida de la operación. Se requiere que todos los nodos capaces de atender este tipo de peticiones tengan instancias de los servidores con la funcionalidad completa de antemano.
- **Migración de procesos.** Consiste en extender la migración de la tarea. En esta estrategia tomamos todos los elementos que constituyen un proceso y los transferimos a otro nodo del sistema para ser procesados en él.

La estrategia más común en las aplicaciones comerciales suele ser la migración de tareas, pues se atienden con éxito los requerimientos de dos problemáticas críticas para la operación: el balanceo de carga y la alta disponibilidad. Dicha estrategia se basa en un conjunto de servidores con configuraciones muy similares que se reparten las peticiones mediante equipos que operen en la capa de red distribuyendo los paquetes entre ellos. De esa forma aprovechan los recursos de múltiples servidores. Se tiene excelente flexibilidad ya que se pueden agregar o retirar servidores del conjunto con relativa facilidad y resiste la falla de los servidores individuales sin mayores efectos adversos.

Para realizar la migración de un proceso se requiere serializar el estado del proceso, el código y manejar un esquema de identificación que permita administrar el ciclo de vida del proceso entre el nodo que solicitó su ejecución y el nodo que lo lleve a cabo.

Para reducir la complejidad, muchos esquemas de migración de procesos, antes de iniciar el proceso, determinan el nodo que resulte más adecuado para atender el proceso, transfieren el código del programa con la identidad correspondiente y evitan la serialización del estado del proceso, ya que ese solo se genera en el nodo donde será llevado a cabo.

EVALUACIÓN

7.1 ¿Qué es un sistema distribuido?

7.2 ¿Qué es una arquitectura de sistemas? ¿Para qué sirve documentarlas como tal?

7.3 ¿Alguna arquitectura revisada es inherentemente segura? ¿Por qué podemos asegurar eso?

7.4 ¿Qué ventajas ofrece para la administración el uso de clientes estándar como Firefox en cliente-abierto en comparación con el desarrollo de clientes a la medida del modelo de comunicación distribuida cliente-servidor?

7.5 ¿Cómo se incluyen los sistemas diseñados y construidos como centralizados en arquitecturas federadas?

7.6 ¿Cuál es la diferencia entre la relación entre clientes *peer to peer* y servidores en sistemas federados?

7.7 Plantea una aplicación que pudiera aprovechar las características de una arquitectura masiva paralela y luego verifica mediante una búsqueda en Internet si se ha realizado un intento de implementación y los problemas que enfrentó. Discute con tus compañeros las consideraciones de seguridad, costo-beneficio y potencial de convocatoria.

7.8 Verifica los costos actualizados de contratar servicios de IaaS y SaaS.

7.9 ¿En qué se diferencian los mensajes síncronos de los asíncronos?

7.10 ¿Qué capacidad de almacenamiento promedio requeriría una cola de mensajes que traslade 100 mensajes por segundo, donde cada uno es de 4 KB y la entrega toma en promedio 1 segundo?

7.11 Comparando NFS surgido en UNIX para compartir achivos y CIFS integrado en Windows con el mismo fin, ¿qué aspectos notas que recibieron más atención?

7.12 ¿Cuáles diferencias consideras más importantes entre las redes *peer to peer* para compartir archivos contra los sistemas de archivos distribuidos?

7.13 ¿Cuáles son las tres principales formas en que se distribuye el trabajo entre nodos de un sistema?

7.14 ¿Cuál es el trabajo de un balanceador de carga?

7.15 ¿Cómo puede evitar la migración de procesos serializar el estado de un proceso?

Referencias bibliográficas

Clements, P. *et al.*, *Documenting Software Architectures: Views and Beyond*, 3a. ed., Pearson Education, 2010.

Tanenbaum, Andrew S., *Modern Operating Systems*, 3a. edición, Pearson, 2009.

Referencias electrónicas

Java SE Application Design WithMoldel-View-Controller, Robert Eckstein, March 2007
http://www.oracle.com/technetwork/articles/javase/index-142890.html#

Resumen de características de Kerrighed
http://www.kerrighed.org/wiki/index.php/UserDoc

GRID Computing
http://en.wikipedia.org/wiki/Grid_computing

A New Era Of SOA: Sun and Microsoft
http://www.internetnews.com/ent-news/article.php/3357721/A+New+Era+Of+SOA+Sun+and+Microsoft.htmnews/article.php/3357721/A+New+Era+Of+SOA+Sun+and+Microsoft.htm

SOA Fundamentals in a Nutshell de IBM
https://www6.software.ibm.com/developerworks/education/ws-soa-ibmcertified/ws-soa-ibmcertified-pdf.pdf

ISC Domain Survey
https://www.isc.org/services/survey/

Guía de usuario de OmniOrb (implementación de Corba):
http://omniorb.sourceforge.net/omni42/omniORB/

Manual de configuración de NFS
http://www.tldp.org/HOWTO/NFS-HOWTO/index.html

Documentación sobre las utilerías de SAMBA
https://www.samba.org/

Ceph: A Linux petabyte-scale distributed file system
http://www.ibm.com/developerworks/library/l-ceph/

COMPETENCIAS A DESARROLLAR

- El alumno ubicará la situación general de la operación de sistemas de información en materia de seguridad informática en un marco de optimización del retorno de inversión y la atención a las leyes y reglamentos que tienen impacto directo sobre esta.
- Identificará las principales prácticas y principios rectores que permiten a la industria dar un tratamiento consistente y efectivo a las necesidades de seguridad de información, así como los principales tipos de atacantes.
- Identificará los principios de la seguridad informática vigentes, como líneas rectoras de los esfuerzos que se realicen en el desarrollo de aplicaciones.
- Identificará el rol de la criptografía como una herramienta para salvaguardar la integridad y la confidencialidad de la información sensible en los sistemas de información, así como las principales características de la legislatura vigente en la materia.
- Aprenderá a identificar un conjunto básico de amenazas de seguridad comunes de diversos tipos de sistemas informáticos para ayudarle a comprender las previsiones que se pueden tomar durante el ciclo de vida completo de las aplicaciones para que pueda ser un participante efectivo en este.
- Identificará las principales características de las técnicas de validación de los usuarios legítimos de un sistema de información, el control de acceso a la funcionalidad de estos y los mecanismos destinados a verificar el cumplimiento de las restricciones impuestas por estos mecanismos.
- Identificará el papel de las características de protección incluidas en los elementos del sistema operativo revisados en otros capítulos.

Panorama de seguridad informática

8

¿QUÉ SABES...?

El tema de la seguridad y la protección de los aspectos importantes de las actividades críticas de una sociedad no es reciente, y ha sido de interés y preocupación para los individuos y las naciones desde la prehistoria. En nuestra vida diaria hay gran cantidad de elementos que sirven para protegernos de numerosos factores físicos, sociales y económicos desarrollados a lo largo de milenios y que hoy día forman parte integral de la vida de casi todas las personas.

Al respecto, es necesario mencionar que hay tendencias en la forma en que las personas perciben su seguridad, así como en el riesgo y las amenazas potenciales que distorsionan su percepción de la situación que les rodea. En la actualidad se conocen varias de estas tendencias, llamadas "cognitive bias", las cuales permiten identificar y diseñar técnicas para corregir esta distorsión a fin de lograr ser más efectivos al perseguir objetivos de protección.

En particular, los sistemas de información cobran importancia en la realización de múltiples actividades económicas y socialmente importantes, motivo por el cual enfrentan problemas de seguridad y deben ajustar sus prácticas y características para funcionar de manera adecuada ante un mundo potencialmente hostil sin perder de vista los objetivos de los sistemas de información.

Importante

- **Docente.** Diversos grupos y organizaciones emplean definiciones diferentes que resultan más adecuadas para su labor, por lo que, a diferencia de otros temas, no debe parecer extraño al alumno que encuentre enfoques y definiciones distintos en diversas fuentes.
- **Líder de proyecto.** Las definiciones aquí presentes consideran el enfoque de la ingeniería de software, pues esta es una disciplina basada en la aplicación de una perspectiva sistémica, disciplinada y cuantificable para el desarrollo, operación y

8.1 Introducción

Al emplear sistemas de cómputo en tareas críticas de actividades con un impacto cada vez mayor, se necesita asegurar que estos sistemas operarán de manera adecuada para evitar que perjudiquen a las mismas actividades o a otras que estén asociadas.

Los sistemas de información constituyen un elemento reciente en el tema de la seguridad, de ahí que la disciplina de la seguridad en cómputo presente cambios rápidos, pues se carece de un sistema aceptado en general, como sucede con otros aspectos de la protección social.

Gracias a la masificación de Internet, la tendencia de los sistemas de información a interconectarse ha ayudado a aprovechar los recursos de las computadoras de maneras inesperadas, pero también ha generado nichos de oportunidad para explotar los mismos sistemas de información en beneficio de intereses ajenos a los que en principio tenían, y a menudo estos intereses resultan ilegales. Hoy en día se ha

observado que las amenazas de seguridad que enfrentan las compañías y los individuos son variadas y llegan a tener escala global. Algunas están bien financiadas y son extremadamente sofisticadas.

Definiciones

Es imperativo establecer ciertas definiciones para delimitar el estudio de la seguridad y prevenir así un grado de subjetividad en el enfoque que pudiera ser perjudicial para el logro de nuestros objetivos.

- **Seguridad en cómputo.** Disciplina encargada de diseñar e implementar las medidas de seguridad en los sistemas de información en función del estudio de las amenazas de seguridad y sus costos potenciales con el objetivo de minimizar el costo final producto de los incidentes de seguridad y las medidas que se tomaron para prevenirlos.
- **Medidas de seguridad.** Características de un sistema de información orientadas a prevenir la divulgación, la creación, el cambio, el borrado o la negación de acceso no autorizados a la información y otros recursos del sistema. Asimismo, son aquellas medidas orientadas a resistir ataques e incidentes de seguridad limitando el daño, restableciendo el servicio, acelerando la reparación y recuperación y manteniendo las propias medidas de seguridad durante la falla y la recuperación.
- **Amenaza de seguridad.** Acción potencial de un agente que violente las medidas de seguridad.
- **Incidente de seguridad.** Evento en el que se violan las medidas de seguridad y que genera un costo para la institución o persona que lo sufre.
- **Ataque.** Acción concreta que viola las medidas de seguridad (incluso si es accidental o intencional).

Conceptos

En general, definimos los siguientes objetivos particulares del enfoque de ingeniería de sistemas para la seguridad de cómputo:

mantenimiento de software; es decir, la ingeniería se aplica al software. Sin embargo, como la seguridad es un problema que debe resolverse prestando atención a la situación global y no a un solo aspecto del sistema de información por separado, a menudo se harán consideraciones a aspectos ajenos al software sin pretender abarcar la totalidad de las consideraciones de seguridad en todos los aspectos posibles.

♠ **Arquitecto.** Al considerar las medidas de seguridad de un sistema de información debe prestarse atención a la criticidad del rol que jugará, en especial donde una falla pudiera atentar contra la vida humana, la de otras criaturas, las estructuras físicas o el medio ambiente. A este tipo de sistemas de información se le conoce como de seguridad crítica, y son los que controlan servicios públicos como el drenaje, los sistemas de aparatos médicos, de transporte, etc. Cuando se trabaja con sistemas de este tipo la evaluación del costo potencial de las fallas es mucho más complicada, por lo que es necesario incluir a las disciplinas correspondientes las inversiones en el desarrollo de salvaguardas, así como las técnicas complementarias apropiadas para mitigar los riesgos asociados.

Importante

♥ **Líder de proyecto.** Al depender de múltiples factores externos, la disponibilidad del sistema de información es uno de los objetivos que enfrenta mayores retos. Eliminar puntos únicos de falla y las demás técnicas de los sistemas de *alta disponibilidad* es un inicio, pero debido a la creciente popularidad de ataques de negación de servicio basados en el uso de gran cantidad de computadoras que han sido vulneradas con anterioridad para saturar los recursos de red y de los sistemas de información, a menudo resulta muy caro para un sistema de información promedio establecer protecciones efectivas contra ataques que empleen este nivel de recursos. Por eso se requiere hacer planes de respuesta para escenarios que involucren este tipo de ataques.

- **Confidencialidad de datos.** Consiste en vigilar que los datos contenidos en el sistema de información solo sean accesibles, de acuerdo con las definiciones del propio sistema, para las personas que deban acceder a ellos y a los usuarios del sistema.
- **Integridad de la información.** La información solo debe ser modificada de acuerdo con la funcionalidad y las reglas definidas por el sistema de información y por los usuarios con los privilegios para hacerlo. Esto debe incluir medidas para evitar que se inserte, distorsione o elimine la información de formas o por personas inapropiadas.
- **Disponibilidad del sistema.** El sistema de información debe ser capaz de mantenerse en operación durante los periodos requeridos sin que las acciones de los usuarios o sistemas externos perturben o impidan dicha operación.
- **Privacidad.** Protección de los intereses de los usuarios ante la posibilidad de que la información personal contenida en el sistema de cómputo sea empleada en su perjuicio, en particular en lo relativo a su seguridad personal, atribuciones legales y derechos humanos.
- **Rentabilidad.** La seguridad de los sistemas de información debe velar por el cumplimiento de los objetivos de estos, incluida la rentabilidad y los beneficios que ofrece a sus usuarios legítimos. Es necesario hacer las consideraciones pertinentes para evitar que las medidas de seguridad excedan en costo al beneficio obtenido por el sistema.

8.2 Desarrollo y mantenimiento seguro de software

Como desarrolladores de sistemas e ingenieros de sistemas en general, debemos considerar los requerimientos de seguridad en nuestros sistemas en los momentos oportunos y con la seriedad suficiente para brindar el nivel correcto de protección por el costo adecuado. Esta es una tarea compleja, ya que las amenazas cambian de manera continua como respuesta a la protección que los sistemas emplean y por las limitaciones de costo involucradas. Además, a diferencia de otras formas de seguridad, no podemos depender de actores externos para proteger nuestros sistemas.

Por suerte, el estudio sistémico del desarrollo de software y de las amenazas de seguridad conocidas ha permitido definir técnicas y principios que todo profesional puede usar para llevar a buen término sus proyectos.

Uno de los principales principios es la "Security in depth" o **seguridad a profundidad**, que se refiere a la implementación de mecanismos de seguridad en todas las capas del sistema de información, de modo que si un ataque resulta exitoso en una capa, el impacto sea detenido por las protecciones de las demás capas. Este es un principio que se distingue en el diseño de muchos componentes de los sistemas operativos.

Otro principio por lo común aceptado se basa en que es mucho más sencillo y económico considerar las necesidades de seguridad en todas las etapas del desarrollo de sistemas de información nuevos que realizar cambios en el sistema de información. Es incluso pertinente mencionar las consideraciones de seguridad en las etapas de un ciclo de vida de cascada:

Seguridad en la definición de requerimientos

En el levantamiento de requerimientos deben considerarse los requisitos de seguridad del sistema de información para dar claridad y redactar de manera específica los contenidos de los objetivos y políticas en forma de requerimientos de software. Estos requerimientos serán la base de las consideraciones de seguridad que se realicen en las etapas posteriores. Debe prestarse atención tanto a las funciones específicas que la seguridad demanda como a las formas conocidas en que el sistema de información podría verse amenazado.

Seguridad en el diseño de software

En el diseño debe asegurarse que la interrelación de los módulos del sistema cumpla con los requerimientos de seguridad y facilite la funcionalidad de los diversos elementos. También es importante aclarar el desarrollo de los pasos en las demás etapas. Algunos factores a considerar incluyen las arquitecturas de seguridad y los módulos de control de acceso que se utilizarán y que rigen las estrategias y mecanismos de supervisión a seguir en la implementación.

Seguridad en la construcción de software

Debe definirse la forma de generar el código para las necesidades específicas del sistema de información de modo que se pueda asegurar el cumplimiento de los requisitos de seguridad. Esto tiene dos aspectos: que el código desarrollado sea seguro y que las medidas de seguridad indicadas en los requerimientos se incluyan en el desarrollo. Lo

anterior obliga a los desarrolladores a adquirir técnicas y disciplina en el desarrollo para lograr generar un código que sea seguro en sí mismo de forma consistente. También obliga al proceso de ingeniería de software a considerar las validaciones y el manejo de requerimientos adecuado para garantizar que, en efecto, el producto incluya los requerimientos apropiados.

Seguridad en las pruebas de software

Las técnicas actuales de desarrollo de software no garantizan que el producto estará libre de errores, por lo que es indispensable realizar pruebas que permitan detectar dichos errores para corregirlos lo antes posible y así limitar el costo asociado a ellos. La seguridad en el proceso de pruebas del software busca asegurar que los datos son protegidos y las políticas de seguridad se cumplen de manera efectiva por el sistema de información, de tal modo que en caso de haber una omisión o vulnerabilidad en el producto las pruebas lo detecten y permitan identificar la deficiencia para proceder a corregirla.

Actores de amenazas de seguridad

Uno de los aspectos de la seguridad informática que ha registrado mayores cambios en los inicios del siglo XXI es el de los actores principales, mediante los cuales es factible identificar mejor las características, el origen y los recursos con los que cuentan los ataques de seguridad a los que son sometidos nuestros sistemas de información.

En la actualidad podemos identificar cinco tipos principales de generadores de ataques en el panorama de la seguridad informática:

- **Aficionados.** Individuos con talento que con base en su estudio independiente o con muy poca organización logran un buen nivel técnico que emplean para atacar la seguridad de otros usuarios, organizaciones e incluso del gobierno. Fueron relevantes a fines del siglo XX. Hoy día carecen de los recursos o las condiciones para ser una amenaza importante; sin embargo, deben considerarse con seriedad, en especial porque puede tratarse de personas dentro de la organización que gocen de privilegios o que se encuentren en situaciones que faciliten lograr un ataque exitoso.
- **Criminales informáticos.** Grupos pequeños o medianos, a menudo con financiamiento de otras actividades ilícitas, cada vez con mayor especialización técnica, que simulan ser empresas legítimas

Importante

♥ **Docente.** El panorama de los principales actores en cada categoría cambia necesariamente y con frecuencia, por lo que resulta un ejercicio casi indispensable para el estudiante (y después para el profesional en cómputo) revisar con periodi-

y que establecen un conjunto de servicios conjugados para distribuir y ocultar la responsabilidad de ataques cada vez más peligrosos y elaborados. Se orientan a obtener ganancias financieras.

- **Hacktivistas.** Colectivos de muy diversos tamaños, a menudo espontáneos y sin una organización formal, pero que funcionan como aglutinantes para personas con talento que desean participar de manera esporádica, o presumiblemente continua, con diversos esfuerzos encaminados a apoyar causas políticas, religiosas o sociales tan variadas como los integrantes, siempre y cuando se logre el consenso sobre las acciones que el colectivo, o una parte de él, deba tomar. Aunque muchas de sus acciones son ilegales y son perseguidas de manera activa por los diversos organismos gubernamentales, su falta de estructura y la volatilidad de su membresía hace difícil que los identifiquen y detengan, e incluso cuando esto se logra, su importancia para el colectivo suele ser menor. Sobre todo, buscan notoriedad para sus causas mediante la disrupción de sistemas de información de quienes perciben como sus contrarios, sean empresas, sitios gubernamentales o incluso individuos.
- **Cyberterroristas.** Movimientos políticos o religiosos que destinan parte de su influencia y recursos a perturbar la operación de sistemas o a dañar los que consideran inconvenientes para ellos. Por ejemplo, es común que busquen alterar los contenidos de redes sociales mediante acoso y generación de mensajes masivos, o que intenten eliminar servicios que apoyen causas contrarias, tanto por medios legales como ilegales.
- **Organizaciones financiadas por naciones/estado.** Entidades gubernamentales, u organizaciones casi gubernamentales, de diversos territorios que reciben atención y parte de sus recursos del Estado y se dedican a la seguridad informática. Cuentan con protección legal o *de facto* del Estado, por lo que pueden actuar con casi total impunidad, aun cuando el alcance de sus acciones sea global. Intentan obtener propiedades intelectuales, información de inteligencia o ventajas estratégicas, aunque algunos grupos son notorios por combinar sus actividades con las de criminales informáticos, cyberterrorismo y hacktivismo, usando estos grupos como intermediarios o pantallas para ocultar su participación.

cidad las fuentes de noticias especializadas para identificar cada categoría, así como los tipos de ataques que están realizando, y con esta información enriquecer las decisiones en materia de seguridad informática que se realicen.

♦ **Líder de proyecto.** Desde el punto de vista de la planeación de proyectos, uno de los mayores retos en materia de seguridad es el estar sujetos a la acción de entidades gubernamentales que no están restringidas por su territorio o por las leyes locales y que cuentan con el personal, los recursos y la experiencia refinada en extremo para lograr sus objetivos. Incluso, cuando los sistemas de información a nuestro cargo no sean el blanco, a menudo el alcance de los ataques es global y afecta a otros como medios o colaterales de los verdaderos objetivos.

♠ **Arquitecto.** La cantidad de recursos disponibles para reforzar la seguridad de los sistemas de información es limitada, por lo que es indispensable preparar los mecanismos de detección, identificación y corrección apropiados para los casos en que sean vulnerados, ya que existen grupos con la capacidad y la motivación necesaria para explotar cualquier vulnerabilidad.

Principios de la seguridad en cómputo

Para orientar los esfuerzos en el diseño y construcción de medidas de seguridad en los sistemas de información es necesario tener un enfoque integral. Asimismo, para guiar este esfuerzo, y como parte de la cultura indispensable de todo profesional en cómputo, es conveniente retomar los lineamientos publicados por la OCDE, organismo que plantea nueve principios que deben acatarse como parte de un criterio integral para la seguridad de los sistemas de cómputo. Estos principios son los siguientes:

1. **Cultura.** Debe generarse una cultura que abarque los riesgos y las salvaguardas disponibles como la primera línea de defensa de la seguridad de los sistemas de cómputo y las redes de telecomunicaciones. Los sistemas de información están expuestos tanto a amenazas internas como externas, por lo que es importante comprender que las fallas de seguridad pueden ocasionar daños severos y emplearse para perjudicar a terceras personas debido al alto nivel de interconexión e interdependencia. Por lo anterior, hay que promover el conocimiento de las configuraciones, las actualizaciones, el rol dentro de las redes de computadoras, las mejores prácticas para implementar estas con el fin de mejorar la seguridad, así como las necesidades de los demás sistemas de información con los que se convive.
2. **Responsabilidad.** En el desarrollo de sistemas de información interconectados a redes de comunicación global (aunque sea de forma esporádica e indirecta) es necesario comprender la responsabilidad compartida en la seguridad de los sistemas. Otro punto importante a considerar es establecer tanto las responsabilidades pertinentes para los diversos roles de los usuarios como los mecanismos para ejercer y responder de formas consecuentes. Deben revisarse las políticas, prácticas, medidas y procedimientos de forma regular y evaluar si son apropiadas para las características actuales del ambiente. La información y las actualizaciones concernientes a la seguridad deben ser elaboradas por los desarrolladores y transmitirse a los interesados de forma oportuna para que los usuarios comprendan las funciones de seguridad de los productos y servicios, así como sus responsabilidades respecto a la seguridad.
3. **Respuesta.** Debido a la interconexión de los sistemas de información y su potencial para una afectación rápida y generalizada, es necesario actuar en respuesta a las amenazas de seguridad de manera pronta y cooperativa. Debe compartirse la información sobre las amenazas y las vulnerabilidades, siempre que sea apropiado, e implementar procedimientos para la cooperación rápida y eficiente para la prevención, detección y respuesta a los incidentes de seguridad, lo que implica comunicación y cooperación a nivel internacional.

4. **Ética.** Las acciones o la falta de acción puede perjudicar a otros. Por tanto, la ética en la conducta a seguir es crucial y deben llevarse a cabo esfuerzos para desarrollar y adoptar las mejores prácticas y promover la cultura que reconozca las necesidades de seguridad y respete los legítimos intereses de los demás.
5. **Democracia.** La seguridad debe implementarse de manera consistente con los valores reconocidos por las sociedades democráticas, incluyendo la libertad de intercambiar opiniones e ideas, el libre flujo de la información, la confidencialidad de la información y de las comunicaciones, la apropiada protección de la información personal, la apertura y la transparencia.
6. **Gestión de riesgos.** Al identificar las amenazas y vulnerabilidades, la perspectiva debe ser lo bastante amplia para considerar los factores clave tanto internos como externos (factores tecnológicos, físicos, humanos, políticas y servicios de terceros con implicaciones de seguridad). La gestión de riesgos permitirá determinar los niveles de riesgo aceptables y ayudará a la selección de los controles apropiados para manejar el riesgo de daños potenciales a los sistemas de información y redes en función de la naturaleza e importancia de los datos que se desea proteger. Debido a la creciente interconexión, la gestión de riesgos debe incluir consideraciones del daño potencial que podría originarse por otros o ser causados a otros.
7. **Diseño e implementación de medidas de seguridad.** Los sistemas de información, redes y políticas necesitan estar diseñadas, implementadas y coordinadas de forma apropiada para optimizar la seguridad. Uno de los principales objetivos, pero no exclusivo, es la adopción de soluciones y salvaguardas apropiadas para evitar o limitar el daño potencial producto de amenazas y vulnerabilidades conocidas. Estas soluciones y salvaguardas —de naturaleza técnica y no técnica— son necesarias y deben ser proporcionales al valor de la información en los sistemas de cómputo de la organización. La seguridad debe ser un elemento fundamental de todos los productos, servicios, sistemas y redes de telecomunicaciones y una parte integral de la arquitectura y diseño de sistemas de información. Para los usuarios finales, el diseño de la seguridad y su implementación consiste sobre todo en elegir y configurar los productos y servicios de manera apropiada.
8. **Administración de seguridad.** La administración de seguridad debe basarse en la estimación de riesgos y debe ser dinámica, abarcando todos los niveles de las actividades del sistema de información y todos los aspectos de sus operaciones. Debe incluir respuestas que prevengan amenazas emergentes y atender a la prevención, detección y respuesta a incidentes, recuperación de sistemas, mantenimiento sostenido, revisiones y auditoría. Las prácticas, las medidas y los procedimientos para los sistemas de información deben estar coordinados e integrados

para crear un conjunto coherente de medidas de seguridad. Los requerimientos de la administración de seguridad dependen del nivel de involucramiento, del rol del participante, del riesgo involucrado y de los requisitos del sistema de información.

9. **Reevaluación.** Continuamente se descubren nuevas y cambiantes amenazas y vulnerabilidades. Los participantes deben revisar, reevaluar y modificar de manera continua todos los aspectos de la seguridad para lidiar con estos riesgos emergentes.

Privacidad

Considerando la creciente importancia de las actividades económicas, sociales e intelectuales, así como de la información que las personas colocan en los sistemas de cómputo, las consideraciones de costo hechas para salvaguardar las inversiones realizadas en el desarrollo resultan ser solo un aspecto a salvaguardar.

La protección de los recursos que los usuarios invierten en el sistema de cómputo (sobre todo información) es otro aspecto de gran importancia que está ganando notoriedad, en especial debido a los abusos de los que han sido víctimas a manos de gobiernos y de particulares.

Un buen punto de referencia es la serie de principios promulgados por la OCDE en materia de privacidad por su compromiso con la democracia pluralista, el respeto a los derechos humanos y las economías de mercado abiertas. Estos principios no son prioritarios para todos los países, por lo que las legislaciones locales varían en gran medida y debe tenerse cuidado de verificar la legislación vigente al diseñar un sistema de cómputo que maneje información personal.

Principios de la OCDE

A continuación se presentan los principios de la OCDE:

- **Limitación en la recopilación.** Deben existir límites para la recolección de datos personales, los cuales deberán obtenerse por medios legales, justos y con el conocimiento del sujeto implicado.
- **Calidad de datos.** Los datos personales deberán ser relevantes para el propósito con el que se recabaron y, también en función de este, deberán ser exactos, completos y actuales.
- **Especificación de propósito.** El propósito de recopilar los datos deberá especificarse en el momento en que se realice la recolección. El uso de los datos debe limitarse al cumplimiento de los objetivos u otros que sean compatibles con el propósito original, especificando de manera puntual el cambio de objetivo, en caso de que este se realice.

- **Limitación de uso.** No se debe divulgar, poner a disposición o usar los datos personales para propósitos que no cumplan lo expuesto por la especificación del propósito, excepto:
  - Si se tiene consentimiento del sujeto implicado.
  - Por imposición legal o de las autoridades.
- **Salvaguarda de la seguridad.** Deben emplearse salvaguardas razonables de seguridad para proteger los datos personales contra riesgos, tales como pérdida, acceso no autorizado, destrucción, uso, modificación o divulgación de los mismos.
- **Transparencia.** Debe existir una política general sobre transparencia en cuanto a evolución, prácticas y políticas relativas a datos personales. Se deberá contar con medios ágiles para determinar la existencia y la naturaleza de datos personales, el propósito principal, la identidad y el lugar de residencia de quien controla esos datos.
- **Participación individual.** Todo individuo tiene derecho a:
  - Que quien controle los datos u otra fuente le confirme que tiene datos sobre su persona.
  - Que le comuniquen los datos relativos a su persona en un tiempo razonable, sin costo o a un precio que no sea excesivo, de forma razonable y de manera entendible.
  - Que se le expliquen las razones por las que una petición referente a los primeros dos puntos haya sido denegada, para poder cuestionar tal negación.
  - Expresar dudas sobre los datos relativos a su persona y, en caso de que proceda, conseguir que estos se eliminen, rectifiquen, completen o corrijan.
- **Responsabilidad.** Sobre todo aquel que controle datos personales recae la responsabilidad del cumplimiento de las medidas y regulaciones que se hagan efectivas de manera local y de acuerdo con los principios anteriores.

Ley Federal de Protección de Datos Personales

En el caso de México existen diversas leyes y reglamentos, ya que se toman consideraciones separadas para el manejo de los datos personales por instituciones financieras, por particulares con fines comerciales o de lucro y por el Estado. Por ejemplo, en la Ley Federal de Protección de Datos Personales en Posesión de Particulares se especifican principios semejantes a los de la OCDE, que son:

- **Licitud.** Los datos personales deben recabarse y tratarse conforme a las leyes aplicables. La obtención no debe realizarse por medios engañosos o fraudulentos, y en todo tratamiento de datos personales se presume que existe la expectativa razona-

ble de privacidad, entendida como la confianza que deposita cualquier persona en otra respecto de que los datos personales proporcionados entre ellos serán tratados conforme a lo acordado por las partes y en los términos que dicta la ley.

- **Consentimiento.** Todo tratamiento está sujeto a consentimiento de su titular y será expreso cuando se manifieste verbalmente, por escrito, por medios electrónicos, ópticos o cualquier otra tecnología o por signos inequívocos. También cuando de manera tácita no manifieste su oposición luego de poner a su disposición el Aviso de Privacidad. Tratándose de datos personales sensibles, se debe obtener el consentimiento expreso y por escrito del titular para su tratamiento, a través de su firma autógrafa, firma electrónica o algún mecanismo de autenticación de común acuerdo.
- **Información.** Se debe poner a disposición de los titulares el *Aviso de privacidad*, un documento que al menos tenga la identidad y el domicilio de quien recibe los datos, la finalidad que se dará a estos, las opciones y los medios que se ofrecen para limitar el uso o la divulgación de los datos, los medios para ejercer sus derechos de acceso, rectificación, cancelación u oposición, y en su caso las transferencias que se efectúan y el procedimiento y medio por el cual el responsable comunicará a los titulares de cambios al propio aviso.
- **Calidad.** El responsable debe procurar que los datos personales a su cuidado sean pertinentes, correctos y actualizados para los fines para los cuales fueron recabados.
- **Finalidad.** Cuando los datos dejen de ser necesarios para las finalidades previstas tendrán que cancelarse. Si se pretende tratar los datos para un fin distinto al originalmente pactado, se deberá recabar de nueva cuenta la autorización por parte del titular.
- **Lealtad.** El tratamiento de datos personales será el que resulte necesario, adecuado y relevante en relación con las finalidades previstas en el aviso de privacidad. En particular, para los datos personales sensibles deberán realizarse esfuerzos para limitar el periodo de tratamiento, de modo que sea el tiempo mínimo posible.
- **Proporcionalidad.** No deben adoptarse medidas de seguridad menores a aquella que se mantengan para la protección de la información propia y se debe tener en cuenta el riesgo existente, las posibles consecuencias para los titulares, la sensibilidad de los datos y el desarrollo tecnológico. Las vulneraciones de seguridad que al ocurrir afecten de forma significativa los derechos patrimoniales o morales de los titulares serán informadas de forma inmediata por el responsable al titular, a fin de que este pueda tomar las medidas correspondientes para defender sus derechos.
- **Responsabilidad.** El responsable velará por el cumplimiento de los principios de protección de datos personales establecidos por las leyes vigentes, debiendo adop-

tar las medidas necesarias para su aplicación, incluso cuando contrate a terceros para realizar el tratamiento de la información. Debe establecer y mantener medidas de seguridad administrativas, técnicas y físicas que permitan proteger los datos personales contra daño, pérdida, alteración, destrucción o uso, acceso o tratamiento no autorizado.

Adicionalmente, detalla una serie de derechos de los titulares de datos personales:

- **Acceso.** Acceder a los datos personales en poder del responsable, así como conocer el *Aviso de privacidad*.
- **Rectificación.** Corregir los datos cuando sean inexactos o incompletos.
- **Cancelación.** Bloquear el uso de los datos por un periodo tras el cual se proceda al borrado de estos. El responsable podrá conservarlos durante el bloqueo solo para efectos de las responsabilidades nacidas del tratamiento y el periodo será el mismo que el plazo de prescripción de las acciones de dichas responsabilidades. La cancelación no será obligatoria cuando los datos sean parte de un contrato, sean objeto de prevención o diagnóstico médico o deban ser tratados por disposición legal.
- **Oposición.** El titular puede oponerse al tratamiento de sus datos. Si la causa es legítima, el responsable deberá abstenerse de tratar los datos del titular.

Asimismo, se hace una distinción entre los datos personales y aquellos de especial relevancia por el potencial daño que se pueda realizar a la persona, para lo cual se definen de la siguiente manera:

- **Datos personales.** Cualquier información concerniente a una persona física identificada o identificable.
- **Datos personales sensibles.** Aquellos datos personales que afecten a la esfera más íntima de su titular, o cuya utilización indebida pueda dar origen a discriminación o conlleve un riesgo grave para este. En particular, se consideran sensibles aquellos que puedan revelar aspectos como origen racial o étnico, estado de salud presente y futuro, información genética, creencias religiosas, filosóficas y morales, afiliación sindical, opiniones políticas y preferencia sexual.

ACTIVIDAD PROPUESTA

En acción

En equipo de dos o tres personas elaboren un díptico pensado en usuarios sin conocimientos especializados en seguridad informática donde se presente la Ley Federal de Protección de Datos Personales. Expongan sus trabajos y seleccionen el mejor.

Criptografía

Esta disciplina reúne principios, medios y métodos para la transformación de datos con el propósito de ocultar la información que representan, establecer su autenticación, evitar modificaciones no detectadas, evitar su repudio y/o evitar su uso no autorizado. Es uno de los medios tecnológicos empleados para salvaguardar la seguridad de los datos en los sistemas de información y telecomunicaciones.

Sin embargo, el correcto uso de las técnicas criptográficas requiere de diversas consideraciones específicas que resulta conveniente conocer para evitar generar un falso sentido de seguridad en el caso de usos incorrectos o para evitar el abuso de estas técnicas incrementando los costos e incluso los riesgos.

Confianza del usuario

Los sistemas informáticos revisten gran importancia para que los individuos, las empresas y los gobiernos puedan funcionar de manera apropiada y sin interrupciones, de ahí la creciente necesidad de que los sistemas de cómputo sean confiables, estables y seguros, particularmente en lo que progresan aplicaciones como el comercio electrónico y los sistemas de pago electrónicos. La carencia de seguridad o de confianza en la seguridad de estos sistemas podría frenar el desarrollo y la adopción de estas tecnologías. Estos medios, al igual que los empleados con anterioridad no son del todo seguros y su uso implica la aceptación de un riesgo calculado. Los consumidores los adoptan cuando esa percepción de riesgo es superada por los beneficios que aportan. Para fomentar la confianza de los consumidores deben realizarse tres esfuerzos principales: desarrollar las tecnologías, educar al público sobre el uso de estas y planear medidas para evitar y corregir fallas en la tecnología.

Elección del usuario

Las soluciones para proteger la información de diversas amenazas pueden tomar muy diversas formas. Los usuarios tienen a su disposición una cantidad considerable de métodos de criptografía que resultan adecuados para gran variedad de requerimientos de sistema y de niveles de seguridad de los datos. Además, estas pueden integrarse en productos relacionados variando la fuerza de la protección y la complejidad de su uso. Aunque diversos gobiernos han implementado regulaciones sobre el uso de criptografía para limitar las exportaciones, el manejo de llaves o con requerimientos para los niveles mínimos de protección en diversas aplicaciones, estas pueden tener el efecto de favorecer algunos tipos e implementaciones sobre otras; sin embargo, es común acep-

tar que los usuarios tengan una amplia variedad de métodos criptográficos para elegir aquellos que se adapten mejor a sus necesidades, favoreciendo la existencia de numerosos productos y limitando el impacto de potenciales fallas en un mecanismo de uso generalizado.

Desarrollo guiado por el mercado

El sector privado es un participante crítico en el desarrollo y la construcción de la infraestructura de sistemas de información y redes de telecomunicaciones. Por lo anterior es común aceptar que la industria debe desarrollar productos y determinar los estándares con base en las necesidades de este sector. Aunque el sector público también tendrá sus necesidades y participará en el mercado influenciando el desarrollo, debe evitarse que guíe de manera exclusiva al mercado mediante regulaciones que pudieran resultar gravosas y que podrían terminar por frenar la adopción de la tecnología. En cambio, debe orientar su trabajo rector a cumplir sus responsabilidades de protección de la seguridad pública y la privacidad.

Estandarización

Para que la criptografía funcione de manera efectiva como una medida de seguridad para los sistemas de información es importante que sus métodos sean interoperables, móviles y portables a nivel global. La interoperabilidad es la capacidad técnica que poseen múltiples métodos criptográficos de funcionar juntos. Movilidad implica la capacidad de operar sobre diversas infraestructuras de telecomunicaciones. Portabilidad es la habilidad para adaptarse y funcionar en múltiples tipos de sistemas.

La estandarización es un ingrediente importante de los mecanismos de seguridad. Debido al rápido desarrollo de la infraestructura, los estándares para los mecanismos de seguridad (incluyendo los métodos criptográficos) emergen muy rápido, ya sean normas de *facto*, por dominio del mercado y por los cuerpos normativos nacionales o internacionales. Es importante para los gobiernos e industria trabajar en conjunto a fin de proveer la arquitectura necesaria y los estándares que permitan a los sistemas de cómputo alcanzar todo su potencial. Los esfuerzos para establecer estándares se describen como guiados por la industria, voluntarios, basados en consenso y de alcance internacional.

Importante

- ♣ **Usuario.** En la actualidad, en México hay muy poca regulación de los mecanismos para el acceso a información cifrada. En principio, se debe entregar la información solicitada por las cortes mexicanas, pero se considera una respuesta válida entregar información cifrada donde las llaves no estén a cargo de las empresas o individuos, impidiendo así el acceso a la información. Además, los recursos para auditorías y el análisis forense de la información son muy limitados, lo que reduce la efectividad de las dependencias para impartir justicia al usar los medios electrónicos como evidencia, y se limita la adopción de estos mecanismos para cuestiones legales y comerciales.
- ♥ **Docente.** La regulación sobre el acceso legal a la informa-

ción cifrada resulta en particular complicada en países como el nuestro, donde los órganos reguladores encargados de aplicar la ley tienen muy poco dominio de la tecnología y severos problemas de operación. Por lo regular se recurre a imitar sistemas de regulación de otros países y formalizar los estándares *de facto* en uso. Por estas razones, es de gran importancia que los desarrollos y las políticas privadas tengan cuidado de proteger estos aspectos alimentando al proceso de elaboración de las leyes con ejemplos bien fundamentados y que atiendan a las necesidades de todos los involucrados. En este aspecto la creciente profesionalización de los ingenieros en cómputo será un antecedente significativo.

Protección de la privacidad

El uso correcto de la criptografía en ambientes de red ayuda a proteger la privacidad de los datos personales y el secreto en información confidencial. Por otra parte, el uso de la criptografía para validar la integridad de los datos en las transacciones electrónicas genera gran cantidad de información factible de procesar para obtener pruebas de identidad y actividad que pueden delinear las actividades privadas y no comerciales relevantes para la privacidad de los usuarios. Por lo anterior es vital considerar la privacidad de los usuarios al diseñar las soluciones y el empleo de los mecanismos criptográficos.

Acceso legal

Una de las controversias más sonadas que rodea al uso de la criptografía es el conflicto aparente entre la confidencialidad de la información y la seguridad pública. En lo referente a la criptografía para proteger la privacidad se deben considerar las estrategias apropiadas para que los mecanismos legales tengan acceso a la información cifrada. Deben balancearse las necesidades de privacidad y de confidencialidad de la información de negocios e individuos con las necesidades de la aplicación de la ley y de seguridad nacional.

Responsabilidad

Los sistemas criptográficos pueden ser vulnerados ya sea por fallas o por error humano en la operación, en especial cuando las llaves criptográficas se ven comprometidas; en este caso todo el esquema de cifrado pierde su efectividad. Por ello es importante saber cuáles entidades participantes en la solución deben hacerse responsables y conocer los límites impuestos a esta responsabilidad de cara a los impactos que estas fallas en la seguridad generen. Deben realizarse las definiciones pertinentes de los niveles de responsabilidad que cada uno de los participantes adquiere, ya sea de forma contractual por previsiones de ley tanto a nivel personal como gubernamental. También deben considerarse las implicaciones nacionales e internacionales debido al alto nivel de interconexión actual.

Importante

♠ **Arquitecto.** Más adelante, en el tema *Elementos de cifrado para proteger información,* se revisarán los elementos clave de la implementación de soluciones de criptografía para compren-

Cooperación internacional

El creciente flujo global de información comercial y personal por las redes de comunicaciones genera una necesidad de enfocar los aspec-

tos de criptografía y estandarización mediante la cooperación y consulta internacional con miras a generar un ambiente sin barreras para el comercio internacional y de cara al impacto que las normas locales tendrán en actividades realizadas fuera de las fronteras de las organizaciones y de los países que las emiten.

der sus características y elementos principales. De momento, solo buscamos señalar su rol en el estudio de la seguridad informática en general.

8.3 Clasificación de las medidas de seguridad

Desde la antigüedad se ha observado que las medidas implementadas para proteger los recursos valiosos resultan inútiles si no se toman con un enfoque integral y coherente. Una puerta impenetrable es inútil si el intruso logra hacer un hoyo en la pared como en la saga del Rey Mono. Por esta razón, es inútil estudiar las medidas de protección del sistema operativo sin incursionar en temas de arquitectura de las aplicaciones, de seguridad de red e incluso de prácticas y cultura informática de los usuarios. En este sentido, todo esfuerzo de seguridad debe incluir consideraciones para todos los niveles. Las medidas se deberán clasificar con un enfoque basado en modelos que separen los diversos elementos y funcionalidad en capas que puedan estudiarse por separado. Un ejemplo de este tipo de modelos es el propuesto por la OSI (Open System Interconnection), la cual describe la funcionalidad de los sistemas de cómputo desde una perspectiva de redes de comunicaciones (véase figura 8.1).

Figura 8.1 Pila de capas de un sistema de información según la OSI.

La funcionalidad de las API proporcionadas por el SO, y de las aplicaciones que las emplean, terminan agrupadas en el nivel de la aplicación en el modelo de la OSI; sin embargo, la implementación de los protocolos de red hace que el rol del sistema operativo en la capa de presentación y sesión también sea muy importante.

A pesar de lo anterior, tanto los individuos como las organizaciones deben diseñar las medidas de seguridad para sus sistemas prestando atención a todas las capas. Hoy día, la tendencia comercial es ofrecer servicios integrales de seguridad; no obstante, a menudo se mencionan las siguientes categorías de seguridad:

- **Seguridad externa.** Aplicaciones externas al sistema operativo que proporcionan la implementación de las medi-

das de seguridad de una capa de servicios para las aplicaciones de un sistema de información.

- **Seguridad física.** Políticas, instalaciones y vigilancia hechas por personal dedicado a salvaguardar la integridad de las instalaciones y de los equipos.
- **Seguridad operativa.** Lo constituyen las políticas, las prácticas, las tecnologías y el uso de cifrado empleado para proteger la información sensible de una persona o compañía.

Validación y amenazas al sistema

Una vez que reconocemos la necesidad de proteger los sistemas, es conveniente identificar al menos los problemas más frecuentes y las contramedidas a tomar. A continuación presentamos una clasificación que también sirve como un panorama actual de las amenazas principales. Este panorama necesariamente cambia con rapidez ya que los atacantes adaptan sus esfuerzos de forma continua a las medidas de seguridad y estas buscan contrarrestar los cambiantes ataques. Las medidas de seguridad son efectivas si se realizan de manera consistente y formal, por lo que el tratamiento para prevenir que la amenaza ocurra debe seguir una serie de etapas bien definidas. Aunque cada organización diseña sus propios programas de seguridad, se deben seguir al menos las siguientes etapas:

- **Prevención.** Tomar medidas para mitigar el riesgo o para reducir la posibilidad de que ocurra la falla de seguridad.
- **Detección.** Descubrir el momento en que está ocurriendo la falla de seguridad.
- **Contención.** Minimizar el daño que puede ocasionar la falla una vez detectada mediante medidas rápidas.
- **Identificación.** Reconocimiento de las características específicas que tiene la falla y, de ser posible, el ataque, la vulnerabilidad o el programa específico.
- **Corrección.** Actuar para corregir la situación que originó la falla y para reducir el impacto.

Muchas organizaciones incluyen aspectos relacionados con el manejo de la comunicación y la coordinación de equipos de trabajo pero por ser asuntos específicos de las organizaciones y no de los ataques o vulnerabilidades de seguridad no haremos mención de ellos en nuestro estudio.

Clasificación de amenazas de seguridad

Podemos clasificar las amenazas de seguridad según los elementos del sistema a los que atacan. De esta manera tenemos tres grandes divisiones:

- **Ingeniería social.** La más antigua de las tres amenazas. Consiste en obtener acceso físico a las instalaciones o a los sistemas aprovechando la psicología de las personas que participan en los sistemas.
- ***Malware.*** Nombre con que se conoce al software que se desarrolla en particular para participar en ataques a sistemas de información. Debido a que muchas de las funciones del *malware* también están presentes en programas usados para dar mantenimiento a sistemas, es posible encontrar programas que puedan ser *malware* o herramientas legítimas en función solo del nombre que se le da y el propósito con el que se emplea.
- **Explotación de vulnerabilidades propias de los sistemas.** En esta categoría agruparemos las técnicas y medidas que un atacante puede tomar en función de las características de un sistema para lograr objetivos ajenos a los de este.

ACTIVIDAD PROPUESTA

En acción

Investiga en diferentes fuentes de información un caso real de amenazas de seguridad y sus consecuencias. Entrega por escrito tu trabajo.

Tácticas básicas de ingeniería social

A continuación se mencionan las tácticas de ingeniería social:

- **Transmitir confianza y control.** El lenguaje corporal, la actitud en general y la forma de comportarse de una persona son los principales elementos considerados por los instrumentos de vigilancia y por las personas para determinar su propia actitud y la confianza que pueden depositar en quienes no conocen. En la mayoría de las organizaciones las interacciones sociales no se limitan solo a personas que se conozcan bien, por lo que tener la actitud adecuada es muy efectivo para atraer la confianza de los demás. Por otra parte, todos tenemos una tendencia natural a sobrevalorar la información que parece provenir de una figura de autoridad; por eso, si el atacante puede fingir con éxito que es una figura de autoridad, es muy probable que logre convencer con facilidad a quienes crean que son sus subordinados y evite que se verifiquen las declaraciones que hace.
- **Ofrecer regalos y favores.** La reciprocidad es otra característica común en las personas que puede ser explotada. Si las personas sienten que han recibido un

favor o un regalo, incluso si tienen dudas razonables sobre la persona que los dio, se ven motivadas a corresponder de alguna manera siempre y cuando no lo identifiquen como un soborno o una medida de presión.

- **Usar el humor.** El humor permite a las personas buscar nuevos enfoques y aliviar la tensión en casi cualquier situación. El atacante usa el humor para modificar la actitud de las personas y conseguir así su confianza, o incluso para solucionar situaciones problemáticas.
- **Argumentos aparentemente razonables.** A través de investigaciones previas y del sentido común es posible formular de antemano razones en apariencia válidas que se puedan decir con confianza y prontitud para presentar un argumento razonable. Se tiende a confiar en los argumentos que se presentan de manera correcta, sin importar que entren en conflicto con conocimientos previos, siempre y cuando no atenten contra las creencias principales de la persona que las escucha.

Tipos de *malware*

En seguida se describen los tipos de *malware*:

- **Bot.** Este término es una abreviatura de robot, y se refiere a un tipo de programa que se coloca en un sistema preparado para tomar control de él y queda en espera de instrucciones. A menudo se combina con otros tipos de *malware* para obtener acceso a los sistemas. Una vez que se ha infiltrado a un gran número de sistemas y actúan en conjunto, se conocen como *Bot Nets* y se emplean para implementar ataques a gran escala o para proporcionar recursos de cómputo que incluso llegan a comercializarse.
- **Virus.** Los virus informáticos son programas desarrollados con el propósito de propagar copias de sí mismos y entregar una carga o *payload* al sistema que infectan. Suelen ser pequeños, eficientes y emplean alguna vulnerabilidad para propagarse. Por ejemplo, algunos virus se integran a programas del sistema de información de modo que se ejecutan cuando su programa anfitrión es ejecutado. Se desarrollan como programas independientes que se colocan entre los programas del sistema operativo y aprovechan características de la administración de procesos para ocultarse o evitar ser terminados. Ocupan rangos de memoria que evitan que el sistema operativo pueda ser recuperado; son capaces de capturar rutinas de atención de interrupciones o de señales para ser ejecutados con frecuencia. En el sector *boot* que se utiliza para iniciar la carga del sistema operativo desde algún dispositivo de almacenamiento masivo, como el BIOS, tiende a ejecu-

tar el código contenido en estos sectores, incluso en dispositivos removibles, lo que constituye una variante habitual. El código que se genera para alguna aplicación que permita incluir scripts en sus archivos (por ejemplo, aplicaciones de ofimática, conocidos como macrovirus) también goza de gran popularidad por la enorme cantidad de equipos que usan productos como el Office de Microsoft, que tienen las mismas vulnerabilidades en millones de equipos.

- **Caballo de Troya.** Toman su nombre de la leyenda del engaño que terminó la guerra de Troya, aunque por razones de brevedad en ocasiones se les llama solo "troyanos". Estos programas se hacen pasar por aplicaciones que el usuario desea, pero en realidad instalan *malware* cuando son activados. En ocasiones incluso proceden a instalar la aplicación que pretendían ser para ocultar sus actividades.
- **Gusano.** Se conoce de esta manera a los programas que son capaces de aprovechar un error en un servicio de red para transferirse al servidor que está siendo atacado y ejecutarse en él. A menudo incluyen la funcionalidad de usar el servidor así invadido para distribuirse a otros servicios similares desde dicho servidor.
- **Conejo.** Este es una versión especial de los gusanos que busca replicar el ataque a todos los servidores accesibles de manera indiscriminada y tiende a saturar los recursos de los servidores, ya que conforme se invade a más servidores, el número de intentos de invasión crece de manera exponencial.
- ***Spyware.*** Recibe este nombre el conjunto de programas que por diversas técnicas recopilan información del sistema y de las actividades del usuario, la preparan y eventualmente la extraen sin el conocimiento o autorización del usuario. En esta categoría se encuentran los programas que capturan las teclas que son oprimidas por el usuario (*keylogger*), los que capturan el contenido de la pantalla (*screen recorder*) y programas especializados en registrar la actividad de la navegación Web, correo electrónico o mensajería instantánea. El *spyware* a menudo es la carga de otro ataque de seguridad, por ejemplo un caballo de Troya o un virus.
- ***Rootkit.*** Estos son conjuntos de programas que se integran a otros elementos del sistema de información para ocultar su presencia y modificar la forma en que se realizan las operaciones para lograr sus objetivos. Pueden reemplazar, por ejemplo, la secuencia de inicio del sistema operativo para integrar su funcionalidad a la del *kernel* y alterar la administración de procesos a fin de ocultar la información que permitiría detectar procesos generados por él. Estos también suelen ser instalados por caballos de Troya y otras formas de *malware*.
- ***Exploit.*** Se denominan con este nombre los programas y algoritmos especialmente diseñados para atacar una vulnerabilidad en un sistema operativo, servicio o aplicación. Por lo regular buscan obtener la ejecución del código generado por el

atacante con privilegios superiores a los del usuario que introduce, voluntariamente o no, el *exploit* al sistema. Una clase especial de *exploits* son aquellos que se orientan a explotar vulnerabilidades de un sistema que no son del conocimiento del fabricante del producto víctima ni de la comunidad de especialistas en seguridad, por lo que no se están desarrollando aún contramedidas; a estos se les conoce como *exploits* de día 0 y tienen un alto valor comercial. A menudo se emplean en conjunto con otros programas de *malware* para potenciar una intrusión, entregar una carga o lograr tener acceso a un sistema.

- **Virus bélicos (Weaponized virus).** A diferencia de los virus convencionales, estos son aplicaciones complejas con gran cantidad de funcionalidad, incluyendo múltiples *exploits* de día 0. Son generados por organizaciones que cuentan con muchos recursos económicos y con la colaboración de profesionales de la seguridad usualmente con objetivos estratégicos a nivel nacional y al amparo de instituciones bélicas o de inteligencia. Aunque es poco probable que nuestros sistemas sean blancos de estos ataques, el mecanismo de propagación que emplean para llegar a sus objetivos suele ser indiscriminado, por lo que los sistemas de cualquier organización están expuestos a ser un paso intermedio, sin importar los efectos negativos que puedan sufrir y la postura de la institución víctima en el conflicto que dio origen al ataque. Hasta hoy no existe una legislación o tratados internacionales para controlar este tipo de ataques, por lo que debemos suponer que serán un elemento de estudio en el escenario de la seguridad por algún tiempo.

Importante

♥ **Docente.** Este tipo de ataques y vulnerabilidades es mucho más memorable si se expone mediante historias de casos famosos, por lo que es recomendable mantenerse atento a las publicaciones de instituciones dedicadas a la seguridad de cómputo para conocerlas y utilizarlas en el curso. Por ejemplo, uno de los primeros *malware* documentados fue el de "Walking disk", ataque que abusaba de la capacidad de estas unidades de recibir comandos desde los programas para activar los frenos de emergencia presentes en los primeros discos duros para evitar accidentes, ya que los medios magnéticos eran grandes, pesados y podían ser manipulados por los operadores. El objetivo de activarlos era desplazar físicamente la unidad de disco en el centro de cómputo al alternar la dirección de rotación y la conservación de momento angular al detener el disco rápidamente. Aunque en la

Vulnerabilidades propias de los sistemas

En el desarrollo de sistemas debe tenerse cuidado de no incurrir en una serie de errores o deficiencias que generan vulnerabilidades de seguridad factibles de ser explotadas por un atacante. A continuación revisamos un catálogo de este tipo de ataques a los sistemas.

- **Abuso de funcionalidad.** El atacante usa las características y la funcionalidad de un sistema para atacar al propio sistema o a otros, es decir, emplea las capacidades de un sistema para cumplir objetivos ajenos a los que originalmente tenía. Esto constituye un ataque porque suele consumir recursos, evitar el control de acceso o fil-

trar recursos incrementando los costos de los usuarios comunes del sistema. Como ejemplos podemos mencionar el uso de los servicios de correo electrónico para almacenar e intercambiar archivos, lo que genera una carga extra para el almacenamiento y la transferencia de información en los servidores en perjuicio del uso objetivo de transferir mensajes. Otros ejemplos son el empleo de los servicios de búsqueda o de traducción como *proxy* para acceder a materiales que de otra forma serían inaccesibles, y el abuso de los servicios de recuperación de contraseñas para obtener acceso a las cuentas de un usuario o para evitar que el usuario pueda continuar usando estas.

actualidad el hardware suele incluir salvaguardas contra usos inadecuados, este programa es un buen ejemplo de la dificultad de predecir todos los usos que una función puede tener y la continua necesidad de estar preparados para detectar y corregir abusos de funcionalidad sobre los sistemas en operación.

- **Fuerza bruta.** Este es un método para determinar un valor desconocido mediante un proceso automatizado que verifica gran número de posibles combinaciones de valores. Este tipo de ataques se aprovecha de que la entropía (cantidad de valores correctos frecuentes respecto a las posibles opciones) suele ser menor de lo que se cree. Por ejemplo, los números confidenciales, o PIN, de tarjetas de crédito de cuatro dígitos tienen 10 000 posibilidades; sin embargo, aproximadamente 10% de las personas usan "1234" como su PIN, y más de la cuarta parte emplean uno de los 20 PIN más frecuentes. Así, cuando se tiene acceso a una gran cantidad de tarjetas, es factible obtener una buena proporción de éxito al adivinar los PIN usando solo un conjunto reducido de valores frecuentes. Usar un conjunto de alternativas más frecuentes en vez de explorar todas las posibilidades se conoce como un ataque con diccionario. Este fenómeno aplica también a las contraseñas y a otros datos empleados en la autenticación de usuarios que dependen de valores proporcionados por estos. En casos en los que la verificación de una alternativa es rápida, sencilla o costeable para el atacante, también se pueden revisar grandes conjuntos de datos, llegando incluso al espacio de posibilidades completo.
- **Desbordamiento de búfer.** Esta es una falla que ocurre cuando la memoria que debe almacenar la información que ingresa al sistema permite que se le escriban más datos de los que es capaz de contener. Para explotar estos errores, el atacante aprovecha las limitaciones de la administración de memoria para alterar su comportamiento o el estado en memoria de la información. Este tipo de ataque se vale de que en la mayoría de los procesos la información de las tablas de símbolos (que definen la ubicación de las funciones) comparte el segmento de datos con los búferes para datos de entrada, y con esto se generan desbordamientos de búfer capaces de alterar la función que el proceso invoca en determinadas circunstancias. Así se logra que el proceso realice actividades bajo el control del atacante y diferentes a las

de su programación original. Esta es una técnica frecuente para implementar *exploits*.

- **Suplantación de contenido.** Estos ataques logran que información generada por el atacante parezca provenir del sistema afectado, lo que perjudica sobre todo a sistemas donde se publica contenido de una fuente formal, como un periódico o una revista, junto con información proporcionada por usuarios externos, como los comentarios. Cuando el ataque es exitoso, el mensaje que debía contener comentarios de un usuario malicioso logra introducir un código que es interpretado después por el sistema como si hubiera sido elaborado por la institución responsable del sistema.
- **Predicción de sesiones.** Es un método para suplantar a un usuario legítimo del sistema de información. En la actualidad es muy común que los mecanismos de autenticación generen un identificador de sesión cuando el usuario proporciona la información que permite identificarlo (por ejemplo, el nombre de usuario y la contraseña). Una vez que se tiene dicho identificador de sesión, las operaciones siguientes solo entregan el identificador para satisfacer los requerimientos de autenticación. Estos métodos logran predecir o reproducir los identificadores de la sesión que genera un usuario legítimo para utilizarlos con sus propios fines.
- **XSS Cross site scripting.** Programación entre sitios Web. Esta técnica presenta los datos introducidos por el atacante como un código que es ejecutado del lado del cliente. Es común en sistemas Web donde los navegadores son de propósito general y reciben el código a ejecutar en el cliente desde el servidor, pero también pueden afectar cualquier aplicación o sistema donde la información que reciben los clientes puede contener un código que se ejecuta de forma local. En fechas recientes estos ataques han logrado implementar funcionalidad similar a la de los gusanos y se han propagado a otras partes del servicio usando los clientes de los usuarios legítimos.
- **CSRF (Cross site request forgery).** Falsificación de peticiones entre sitios Web. En este ataque se obliga al programa cliente de la víctima a realizar una petición a un servidor sin su autorización ni conocimiento para obtener información confidencial a partir de la respuesta o para abusar de la confianza del servidor en el cliente y las sesiones que tenga abiertas. Estos son similares a los XSS, pero funcionan abusando de la confianza depositada en el cliente por el servidor en lugar de la confianza del cliente en el contenido que obtiene del servidor. En el caso de los sistemas que además son vulnerables al XSS se pueden conjugar ambas técnicas para lograr que en conjunto amplíen el rango de operaciones que el atacante realiza en el cliente.

- **DOS (Denial of service).** Negación de servicio. Satura alguno de los recursos del sistema generando carga de trabajo que no corresponde con los objetivos del sistema e ignorando la respuesta con el propósito de inhabilitarlo para atender las peticiones de sus usuarios legítimos. Al conjugarse con *Bot Nets*, ha demostrado ser muy difícil de evitar y ha afectado con éxito a empresas y organizaciones con muchos recursos a su disposición.
- **Fingerprinting.** Toma de huellas digitales. Obtiene información confidencial sobre la implementación del sistema. Abusando de la funcionalidad de diagnóstico o presente por defecto en los programas que se emplean como base de un servicio de información y que los administradores olvidaron o no son capaces de ocultar, un atacante puede obtener detalles sobre la configuración, versiones de productos, rutas de instalación y muchas otras que son de utilidad para orquestar otros ataques o para identificar otras vulnerabilidades.
- **Fragmentación de HTTP.** El protocolo de comunicación HTTP define el flujo de información entre clientes y servidores tanto para las peticiones como para las respuestas. Este flujo de información tiene un formato flexible para acomodar la gran variedad de materiales que los usuarios emplean al usar la World Wide Web. La fragmentación de HTTP aprovecha la riqueza de elementos del formato de las peticiones y las respuestas para incluir elementos que no son interpretados por aquellos encargados de verificar la seguridad de la comunicación como *Firewalls* y servidores *Proxy*; de esa manera pueden extraer o hacer llegar información entre los clientes y servidores externos a la institución que de otra forma serían bloqueados por los elementos de la red.
- **Inyección de código.** Esta familia de ataques se vale de la dualidad entre los datos de control y los datos de contenido en los sistemas de información para entregar información especialmente diseñada para que después el sistema la confunda con información de control logrando perturbar o alterar la operación del sistema.

  La información de control son las instrucciones que el sistema debe interpretar para dirigir su comportamiento, como los programas, los documentos con definiciones de estilo (por ejemplo, CSS), las cadenas que definen el formato de las entradas y salidas, señales que controlen el flujo de información, etc.

Importante

♥ **Docente.** Es interesante señalar que cuando Charles Babbage diseñó su máquina analítica consideró que la información de control que determinara las operaciones a realizar debería construirse de una forma diferente que los mecanismos que llevaran la información que debería ser procesada, y fue Ada Lovelace quien sugirió por primera vez que ambos tipos de información podrían manejarse por el mismo tipo de mecanismos, simplificando así la construcción de la máquina de procesamiento de información. Dado el incremento en la importancia de las medidas de seguridad, se analiza actualmente regresar a la idea de Babbage de implementar en dispositivos distintos el almacenamiento de los programas e información de control y los datos a procesar.

♠ **Arquitecto.** Con las computadoras y sistemas de información actuales es indispensable verificar que la información de control se distinga claramente de los datos de contenido y se implementen los mecanismos apropiados para evitar que estos lleguen a confundirse. Por

lo regular, esto implica el uso de API de programación específicos y el saneamiento de todos los datos que ingresan al sistema de información.

Los datos de contenido son la información que ha de ser procesada por el sistema de información. Los datos capturados por los usuarios al llenar un formulario, el registro de la fecha y la hora en que ocurre una transacción, el contenido de los mensajes que intercambiamos mediante los teléfonos celulares o el sonido de nuestra voz en una llamada son todos datos de contenido en diversas aplicaciones.

En este ataque se mezclan instrucciones y código que algún elemento del sistema de información pueda interpretar en los datos a procesar que introduce el usuario y se explotan las deficiencias presentes en la validación de dicha entrada y en los mecanismos que emplean la información para que se interprete como información de control al presentar o utilizar dichos datos. Ejemplos de estos ataques son incluir código HTML en mensajes que se presentarán luego en sitios Web, o enviar comandos SQL a bases de datos relacionales que usen ese lenguaje.

- **Null byte inyection (Inyección de carácter nulo).** Este ataque aprovecha la funcionalidad básica de muchos lenguajes de programación de alto nivel de utilizar un carácter nulo (el código con valor 0) para señalar el final de una cadena, para ocultar la verdadera longitud y contenido de una cadena de caracteres que contiene el carácter nulo pero no termina en él. Esto permite al atacante eludir algunos mecanismos de validación de la entrada de datos.
- **Predicción de ubicación de recursos.** La lógica de los sistemas de información debe guiar al usuario para realizar las operaciones en el orden definido por las políticas del sistema; sin embargo, las peticiones a menudo se realizan de forma independiente y, en especial en la arquitectura del cliente abierto, un usuario puede tener gran control sobre el momento y el contenido de las peticiones que envía al sistema. De esta forma, si el formato de las peticiones permite al atacante predecir el formato y contenido de una petición, este puede intentar realizarlas y mediante las respuestas tener acceso a información confidencial de recursos que estén protegidos de manera inadecuada o sobre la existencia y ubicación de recursos que después intentaría acceder mediante otros métodos.
- **Desviación de ruteo.** Dirigido de manera específica a los servicios que soportan la operación de Internet. Se puede modificar la configuración de los ruteadores o de los servicios de resolución de nombres mediante *malware* u otras técnicas, de modo que las peticiones de los usuarios sean desviadas a servidores preparados por los atacantes. A menudo el sitio del atacante simula además ser el sitio que en principio quería emplear el usuario; esta técnica, llamada *Phishing*, sirve para extraer información confidencial del usuario.

Técnicas de validación y monitoreo

A continuación revisaremos una serie de medidas comunes para garantizar la seguridad de las aplicaciones.

Autenticación

Según el glosario de términos de seguridad para Internet versión 2 de la IETF, la autenticación es el proceso de verificar la aseveración que un sistema, entidad o recurso de sistema hace de tener cierto atributo. Se refiriere usualmente a atributos que permiten identificar al usuario que hace uso del sistema y sus privilegios asociados.

Se reconocen cuatro mecanismos principales para que un usuario demuestre su identidad. De acuerdo con la criticidad del sistema se suelen usar varios de ellos en conjunto para disminuir la posibilidad de un atacante de suplantar al usuario con éxito.

Para verificar la identidad del usuario se puede usar:

- **Algo que el individuo sabe** y que no es del conocimiento público, como la contraseña, el número de identificación personal (PIN) o las respuestas a un conjunto previamente acordado de preguntas.

  La autenticación por medio de parejas de nombre de usuario y contraseña, o por contraseñas, es un mecanismo muy común que se suele combinar con el uso de algoritmos de cifrado con "sal"que impiden que un segundo sistema, con características similares, emplee el mismo algoritmo de cifrado para generar las mismas llaves cifradas con fines de comparación y adivinar así las contraseñas. Los atacantes pueden usar técnicas de fuerza bruta, ingeniería social o aprovechar la falta de capacitación de los usuarios para adivinar o adquirir de forma indebida la información que el usuario debió mantener en la confidencialidad.
- **Algo que el individuo tiene**, como diversos dispositivos que sean difíciles de duplicar o reemplazar, por ejemplo llaves, llaves electrónicas, tarjetas inteligentes (smart cards), etc. Estos dispositivos se conocen como *tokens*.

  Los dispositivos de almacenamiento conservan una pequeña cantidad de información y la distribuyen a los usuarios, como las bandas magnéticas de tarjetas de crédito, los códigos de barras en boletos impresos, la información almacenada en chips de memoria no volátil, etc. Son utilizados de manera rutinaria para identificar a los usuarios de diversos servicios que deben verificar su identidad para realizar múltiples operaciones.

  A menudo los atacantes usan dispositivos especializados para duplicar este tipo de dispositivos o simplemente los sustraen de sus legítimos usuarios como

parte de un ataque. También se emplean tarjetas inteligentes, que además de tener la capacidad de almacenar información pueden procesarla en virtud de tener un pequeño procesador integrado. Las tarjetas inteligentes pueden operar de forma estática, de modo que el usuario solo presenta el dispositivo ante un lector especializado y la operación de autenticación no requiere su participación, como sucede con las tarjetas LadaTel®.

Pueden ser generadores dinámicos de contraseñas, que emiten estas con base en un algoritmo de cifrado ligado al tiempo o al número de repeticiones y que puede sincronizar su operación con la del servicio que realizará la autenticación. Muchos servicios bancarios en línea emplean este tipo de dispositivos y desafío-respuesta; en estos el sistema de información genera un "reto", que puede ser una serie aleatoria de números, y entonces el *token* recibe estos números y genera una respuesta empleando un algoritmo de cifrado que el servidor puede verificar para confirmar que el *token* adecuado fue empleado.

- **Algo que el individuo es.** Se basa en biometría estática que verifica alguna característica constante y particular del usuario (por ejemplo, patrones en la retina, huella digital o en la cara). Los algoritmos de procesamiento digital de imágenes, junto con las características de los dispositivos que realizan la lectura de las características en cuestión, son altamente sensibles a factores del medio ambiente y a la forma en que se realiza la lectura. Esto hace que este tipo de medidas tengan altas posibilidades de falsos positivos y de falsos negativos; es decir, de reconocer por error a una persona diferente como el usuario, o de no reconocer al usuario legítimo por una mala lectura. Además, se han descubierto diversos medios de elaboración de maquetas capaces de engañar a los sensores a partir de elementos poco invasivos para los usuarios, lo que hace que deban conjuntarse con mecanismos de seguridad física significativos.

  Otra consideración muy importante para este tipo de sistemas es la invasión a la privacidad y lo sensible que resulta la información biométrica de los usuarios, lo que implica requerimientos de seguridad adicionales en los sistemas de información para salvaguardar la información biométrica de sus usuarios.
- **Algo que el individuo hace.** Es la biometría dinámica que verifica las características con las que el individuo realiza alguna acción (por ejemplo, el reconocimiento de los patrones de voz, las características de la escritura, el ritmo de escritura en el teclado, etc.). Es particularmente sensible a las condiciones del medio ambiente cuando se realiza la lectura. Tiene una alta posibilidad de generar falsos positivos y negativos y ocupar dispositivos de lectura relativamente caros, por lo que no suele usarse de forma individual.Sin embargo, representa un buen complemento

para otras medidas de autenticación debido a la facilidad de acceso para el usuario legítimo.

Control de acceso

Una política de control de acceso determina qué privilegios se pueden asignar a los usuarios, las circunstancias en que se le permite realizar esas operaciones y a qué usuarios se asocian. Se distinguen tres tipos principales de políticas de control de acceso:

- **DAC (Discretionary acceso control). Discrecional.** Define reglas de acceso o autorizaciones que indican qué solicitantes tienen permitido o denegado el uso del recurso. A esta política se le conoce como discrecional porque los usuarios que tienen permitido el acceso pueden habilitar dicho acceso para otros usuarios. Es el caso de los mensajes instantáneos o de correo electrónico, que son visibles solo para el destinatario hasta que este decide reenviarlo a otros.
- **MAC (Mandatory access control). Obligatorio.** Estos comparan denominaciones de niveles de seguridad asociadas a los recursos e indican qué tan críticas o sensibles son respecto del nivel de acceso de un usuario. Este nivel de acceso determina cuáles usuarios son candidatos a usar recursos sensibles. Se le conoce como obligatorio porque un usuario con autorización de acceder a un recurso no tiene autorización de propagar la información o el uso del recurso a otros usuarios que carezcan de acceso suficiente. Este modelo se desarrolló durante la Segunda Guerra Mundial para prevenir que información táctica se filtrara de los centros de operaciones militares y pudiera ser empleada por el enemigo.
- **RBAC (Role based access control). Basado en roles.** En estos se define la serie de operaciones que los recursos deben controlar como un conjunto de privilegios que pueden ser asignados. En vez de asignar de forma directa estos privilegios a los usuarios, se suelen formar colecciones de usuarios, o grupos, y de recursos o perfiles, y se realiza la asignación de perfiles a grupos de usuarios. Esto permite mantener grandes conjuntos de usuarios y de privilegios asignados de forma consistente. La mayoría de los sistemas de archivos emplean este modelo para determinar los permisos de archivos, como se revisó en la sección *Control de acceso* del capítulo 6, *Administración de sistemas de archivos*.

Detección de intrusiones

Es necesario explicar qué constituye una intrusión y a quiénes consideraremos intrusos.

El intruso es el individuo que accede a la información o a los recursos del sistema sin que tenga autorización para realizar dichos accesos. Las intrusiones pueden ser pasivas, si solo buscan acceder a información, o activas, si además buscan realizar modificaciones en los sistemas de información para ocultar sus actividades, perturbar la operación o para obtener algún otro beneficio.

Diversos estudios han demostrado que esta práctica es muy frecuente y suele tener un impacto mayor cuando el intruso es interno a la organización que está siendo atacada, ya que posee un nivel de acceso y conocimientos acerca de los recursos y mecanismos de seguridad como punto de partida. De acuerdo con la relación del atacante con la institución y sus privilegios, se le suele clasificar en uno de tres distintos grupos:

- **Impostor.** Individuo que no tiene autorización para utilizar los recursos, por lo que emplea los privilegios de otra persona que sí es un usuario legítimo sin su consentimiento.
- **Malhechor.** Usuario legítimo que accede a los recursos sin tener los privilegios necesarios, o que tiene los privilegios pero los emplea de forma inapropiada.
- **Usuario clandestino.** Individuo que se apropia de privilegios administrativos y los emplea para evadir los mecanismos de control y registro o suprimir los registros de sus actividades.

Como se ha mencionado a lo largo de este capítulo, los sistemas generan información de monitoreo que permite diagnosticar y ayudar en la corrección de diversos tipos de fallas. Un administrador puede revisar de forma manual y periódica la información de bitácoras y detectar en ellas patrones y eventos inusuales que le permitan identificar una intrusión a sus sistemas. Sin embargo, un sistema con unos pocos cientos de usuarios es capaz de generar enormes cantidades de información de bitácoras, por lo que la revisión manual de bitácoras no es práctica sino para sistemas pequeños o periodos muy cortos.

Para mantener una vigilancia constante sobre sistemas complejos o con grandes cargas de trabajo se requiere automatizar la instrumentación a fin de obtener información sobre las actividades que ocurren en el sistema y el análisis de esta información en busca de los patrones que permitan detectar las intrusiones. Los sistemas encargados de estas tareas se conocen como sistemas de detección y prevención de intrusiones (IDPS), y según el énfasis que se pongan en la detección o en la prevención pueden ser IDS o IPS en la literatura comercial.

Los cuatro tipos principales de IDPS se describen a continuación:

- **De red.** Monitorean el tráfico de red en segmentos o dispositivos particulares y realizan el análisis de la actividad de los protocolos de red activos en ellos en busca de patrones de actividad sospechosa.

- **Inalámbrico.** Orientados a monitorear el tráfico en redes inalámbricas y de analizarlo para identificar la actividad sospechosa propia de los protocolos de este tipo de redes.
- **Análisis de comportamiento de red (NBA Network bahavior analysis).** Examina el tráfico de red en un segmento en busca de patrones conocidos en flujos de información inusuales que sirvan como indicadores de ataques, como podrían ser negaciones de servicio (DOS), violaciones a las políticas o algunos tipos de *malware* como los *bot nets* o gusanos de red.
- **De servidor (Host based).** Revisan las características y la información de bitácoras de un solo servidor en busca de actividad sospechosa en él.

Estas herramientas buscan actividad inusual o que siga un patrón que correspondaal de un ataque observado en el pasado; sin embargo, a diferencia de un *firewall* que define reglas de antemano para impedir cierto tipo de tráfico en la red, los IDPS usan patrones que no se pueden garantizar *a priori* y que corresponden a fallas de seguridad. Por tanto, estos productos no suelen impedir las operaciones, a menos que se pasen umbrales específicos, y suelen limitarse a generar alertas para que un administrador logre orientar su atención a determinar si se trata de falsos positivos o de ataques auténticos. También se reconoce la posibilidad de que algunos ataques, incluso dentro de las categorías que el IDPS pretende detectar, puedan eludir la vigilancia mediante alguna táctica, por lo que el administrador también debe estar en busca de falsos negativos, es decir, ataques que hayan ocurrido sin generar alarmas, pero que pueden detectarse con información adicional.

Pruebas de penetración al sistema operativo

Una de las herramientas que podemos utilizar para mejorar la seguridad de los sistemas de información en general son las pruebas de penetración, también conocidas como intrusión ética (*Ethical hacking*).

Estas pruebas consisten en violar la seguridad del sistema mediante un esfuerzo sistémico, formal y de común acuerdo entre un profesional o grupo de expertos en seguridad y las instituciones que realizan los sistemas con el objetivo de detectar las vulnerabilidades presentes en el sistema de información.

Estas pruebas son valiosas porque ayudan a encontrar vulnerabilidades diversas:

- **Pruebas de regresión.** Permiten verificar que vulnerabilidades y problemas de seguridad detectados en el pasado no emerjan de nuevo como producto de problemas

en el manejo de versiones o por no realizar esfuerzos suficientes de corrección en el proceso de manufactura del software.

- **Evitar vulnerabilidades comunes.** Al usar diversos servicios, bibliotecas y productos con frecuencia utilizados en múltiples compañías, es muy probable que un sistema de información contenga vulnerabilidades presentes en estos productos o en formas incorrectas, pero comunes de emplearlas. Observar los problemas detectados en otros sistemas similares ayuda a diseñar pruebas que permitan detectar estas vulnerabilidades en caso de estar presentes.
- **Identificar vulnerabilidades que podrían ser difíciles de encontrar de forma manual.** Ya sea porque el análisis de grandes volúmenes de información o la secuencia de pasos a seguir sea excesivamente rápido o complejo para realizarse de forma manual, pero susceptible de ser automatizado.
- **Generar evidencia de las vulnerabilidades.** Generando ataques exitosos se puede obtener información vital para evaluar las características, la magnitud y el impacto que ocasionen las fallas, así como información concreta que guíe los posteriores esfuerzos de seguridad.
- **Evaluar la magnitud del impacto.** La detección de las vulnerabilidades permite identificar los sistemas y recursos comprometidos, y mediante esta información estimar el impacto que las fallas podrían tener, lo que a su vez será una guía muy importante para generar las estimaciones económicas que restrinjan los esfuerzos encaminados a mejorar la seguridad del sistema de información.

8.4 Concepto y objetivos de protección

Es recomendable diferenciar el problema general de la seguridad informática de las medidas particulares que un sistema operativo tome para evitar las vulnerabilidades y brindar facilidades al desarrollo de aplicaciones que permitan cumplir con sus requerimientos de seguridad. Por ello empleamos el término *mecanismos de protección* para referirnos en particular a los mecanismos implementados en el sistema operativo para salvaguardar la operación y la información de los sistemas de cómputo.

Funciones del sistema de protección

En el estudio formal de las medidas de protección que el sistema operativo requiere implementar, debemos separar las necesidades en temas particulares para facilitar su comprensión. En primer lugar tenemos las medidas de protección asociadas a las fun-

ciones de administración de recursos proporcionadas por el sistema operativo. Estas consideraciones se han mencionado en los temas Administración de procesos, Administración de la memoria, Administración del sistema de archivos, etc., pues las medidas de protección son una parte integral de los requerimientos y la funcionalidad de los diversos subsistemas del sistema operativo, y resultaría contraproducente estudiarlas como si se tratara de una consideración posterior o un aspecto separado.

Sin embargo, en la revisión contenida en esos temas no se han revisado con detalle las técnicas formales para estudiar los niveles de protección y el modelado de la seguridad, lo que haremos a continuación.

Implantación de matrices de acceso

Uno de los primeros aspectos que requiere un estudio formal es la definición de las actividades que deben estar disponibles para los diversos usuarios del sistema de información. En este rubro se distinguen en primer lugar las habilidades del administrador como usuario plenipotenciario de los usuarios de las aplicaciones que deben operar bajo el principio de menor privilegio, de modo que sus actividades cumplan con los requerimientos pero se minimice la posibilidad y el impacto en caso de comprometer la operación o seguridad del sistema de información.

El uso y definición de las matrices de acceso se revisó en el capítulo 6, *Administración de sistemas de archivos*, en el tema *Dominios de protección*.

Medidas de seguridad en el *kernel* (núcleo) del sistema operativo

Durante el desarrollo de los sistemas operativos existen diversas oportunidades para un atacante de introducir un código malicioso o de detectar vulnerabilidades que no comunica al resto del equipo de desarrollo. El sistema operativo en su conjunto es una colección muy grande de programas. Solo el proyecto del *kernel* Linux tenía 15 803 499 líneas de código para la versión 3.10 en agosto de 2013, por lo que el esfuerzo de revisar la implementación para asegurarse de que ningún usuario malicioso que haya colaborado en el desarrollo o alguna otra falla de seguridad haya insertado o deteriorado el código resulta complicado.

Para resolver la problemática de revisión del *kernel* del sistema operativo se busca de manera activa que toda la funcionalidad posible se desarrolle sin privilegios de administración ni acceso al modo protegido de la CPU (en la que se tiene acceso

irrestricto a cambios de contexto y consulta a tablas de páginas, como se mencionó en el capítulo 3, *Administración de procesos*, en el tema *Modo de usuario y modo de sistema* —o *protegido*—), la parte que ineludiblemente debe operar en modo protegido es solo un subconjunto reducido del código, y eso facilita en gran medida el esfuerzo de revisión de este.

La revisión de las aplicaciones que se distribuyen con el *kernel* se maneja por separado y con base en cada aplicación, y por lo regular es mucho más permisiva, ya que en principio no operará con privilegios elevados.

Para controlar los privilegios de los usuarios e implementar las limitaciones que los procesos deben respetar se utilizan mecanismos que deben implementarse en el *kernel* y que aprovechan estructuras de datos que se asocian con la información que se mantiene en memoria para administrar cada proceso. En el caso de Linux, los diversos atributos que permiten al sistema operativo determinar los tipos de operaciones a los que un proceso tiene acceso se conocen como capacidades (capabilities), y en el caso de Windows se abstraen mediante el concepto de "objetos" (Object), que incluyen estructuras con atributos para controlar esta información.

Importante

♥ **Docente.** Decidimos dar tratamiento al tema de la administración de permisos, discutiendo los aspectos de la protección e implementación de sus mecanismos y no en un tema independiente bajo la categoría de la seguridad informática para hacer énfasis en que la implementación de las consideraciones de seguridad es una parte integral del desarrollo de software y no una consideración que sea rentable realizar a futuro.

Protección basada en el lenguaje de programación

Muchas medidas de protección están implementadas en el *kernel* del sistema operativo; sin embargo, diversas reglas y verificaciones de seguridad que podrían aplicarse a cada operación son omitidas con el fin de reducir la carga de trabajo que esto generaría o porque la regla en cuestión no sea de aplicación general sino particular a un sistema de información.

Además, las arquitecturas de los sistemas incorporan cada vez más servicios y bibliotecas, cada una de las cuales debe definir sus propias reglas para salvaguardar la seguridad de la información que maneja. Estas reglas particulares no pueden incluirse de forma rentable, es decir, que los beneficios a obtener excedan al costo del esfuerzo de hacerlo en el sistema operativo, ya que no son comunes a la funcionalidad de todas las aplicaciones que operan en él y también porque el conjunto de arquitecturas y bibliotecas es demasiado grande para ser considerado en su totalidad.

Por estas razones, en la actualidad se considera que la implementación de los mecanismos de seguridad es una tarea compartida que impone actividades y re-

querimientos a la capa de aplicación del modelo de la OSI. En esta, los ambientes de desarrollo, lenguajes de programación y bibliotecas de funciones juegan un papel muy importante en la implementación de la seguridad.

En particular, los lenguajes de programación, sus compiladores e intérpretes incluyen declaraciones sobre las limitaciones de control de acceso que deben imponerse a los recursos, sean bibliotecas, entidades de datos, funciones, etc. Este tipo de declaraciones se integra a la gramática del lenguaje extendiendo las capacidades de la definición de tipos de datos. Lo anterior proporciona las siguientes ventajas:

- Las necesidades de protección se declaran, en vez de programarse como una secuencia de llamadas. Esto mejora la consistencia en la implementación, cuyo reúso queda a cargo del compilador y facilita el mantenimiento por ser una declaración más sencilla.
- Los requerimientos pueden especificarse de modo independiente de las prestaciones que ofrezca un sistema operativo en particular. Esto facilita la portabilidad entre sistemas y la interoperación en sistemas distribuidos.
- Los medios para garantizar el respeto a las políticas no necesitan ser desplegados por el desarrollador de la aplicación. Se pueden utilizar los mecanismos de protección proporcionados por el lenguaje de programación y la plataforma de desarrollo, con lo que se aprovecha mejor el esfuerzo y refinamiento presente en estos.
- La notación declarativa resulta sencilla de entender porque los privilegios de acceso a la información están conceptualmente relacionados con la declaración de los tipos de datos y componentes en los sistemas.

Uno de los ejemplos más importantes de esta tendencia se puede apreciar en la definición, apoyada por los principales fabricantes de software, de la Service Component Architecture, que está orientada a la generación de aplicaciones que se compongan de elementos dispares mediante una orientación a servicios. En su definición se especifica que las plataformas que la soporten deben sostener las siguientes ventajas para sus aplicaciones:

- Desacoplar la lógica de negocio de la aplicación de los detalles de las llamadas a servicios (independencia lógica).
- Desarrollo de servicios en múltiples lenguajes, incluyendo C++, Java, COBOL y PHP, y con lenguajes especializados como XML, BPEL y XSLT.
- La habilidad de trabajar de forma transparente con diversos mecanismos de comunicación, incluyendo unidireccionales, asíncronos, llamada-respuesta y notificaciones.

- La habilidad de enlazar componentes y servicios previamente desarrollados sin necesidad de realizar modificaciones significativas en ellos mediante tecnologías como Web services, EJB, JMS, JCA, RMI, RPC, CORBA y otras.
- La habilidad de declarar los requerimientos de seguridad, calidad de servicio, manejo de transacciones y el uso de *reliablemessaging*, independientemente de la lógica del sistema.
- Tener la capacidad de representar los datos en Service data objects de manera independiente a la plataforma de implementación y encapsulando los mecanismos de persistencia.

Como se puede ver, entre los objetivos fundamentales de esta arquitectura figura la declaración de los requerimientos de seguridad de los componentes, independientemente de los mecanismos de almacenamiento de la información o de implementación del resto de las políticas del sistema de información que se desarrolle en ella.

8.5 Elementos de cifrado para proteger información

Importante

♥ **Docente.** ¿Qué tan antiguo es el cifrado de información? Se han encontrado inscripciones en al menos una tumba egipcia de aproximadamente 1900 años a.C. En la cámara principal se encuentran jeroglíficos inusuales dispersos por la escritura, reemplazando a jeroglíficos normales. También se encuentran referencias en tratados militares antiguos de la India. Se conoce el uso alrededor del 500 a.C. del *scytale* para cambiar el orden de las letras de mensajes militares espartanos y se tiene bien documentado en la Roma clásica el uso de un cifrado por sustitución de letras, que en la actualidad se conoce como el cifrado César.

Una de las técnicas usadas desde la antigüedad para salvaguardar la información es el cifrado. Este es un método que consiste en esconder el significado de un mensaje para que solo los receptores deseados lo puedan interpretar. Se procura que el mensaje solo sea recibido por el legítimo destinatario y se mantenga en secreto, por lo que el cifrado brinda medios para impedir, o al menos retrasar, el acceso a la información en caso de que el mensaje se filtre. En esta técnica partimos de la suposición de que el atacante podría tener acceso a información que debería ser secreta, por lo que no resulta correcto que todo el esquema de cifrado dependa de que el atacante ignore el mecanismo de cifrado, ya que muchos de estos mecanismos pueden inferirse mediante el análisis de las características del mensaje cifrado a la luz del conocimiento del lenguaje y otras consideraciones similares.

En resumen, podemos decir que la criptografía es el estudio de los métodos para esconder la información en la escritura y para recuperar la información de ella. Es un campo de estudio con larga historia e intencional complejidad que rebasa por mucho el alcance del presente; sin embargo, es común considerar el uso de la criptografía en el diseño de la seguridad de los sistemas de información y, por tanto, debemos revisar los fundamentos de la criptografía.

A lo largo de la evolución de los métodos de criptografía se ha pasado de necesitar que el mecanismo de cifrado sea secreto a requerir solo que la llave se mantenga secreta. La llave es un complemento al mensaje que modifica el resultado que generará el cifrado, de modo que, aun conociendo el mensaje cifrado y el mecanismo usado, el mensaje original no se puede recuperar sin conocer la llave.

La ventaja de que el mensaje resista aunque solo se mantenga en secreto la llave radica en que cuando por fin la llave es conocida, ya sea porque se revele o se obtenga por fuerza bruta, el mecanismo de cifrado puede seguirse empleando con nuevos mensajes con solo generar llaves nuevas. De nueva cuenta, estas llaves serán seguras durante el periodo que tome al atacante averiguarlas, y en la medida en que hacerlo sea suficientemente complejo o tardado será un buen detractor para el atacante.

Conviene hacer hincapié en que ningún elemento de la comunicación se considera seguro ni se podrá mantener secreto por tiempo indefinido. Al diseñar esquemas de cifrado debe saberse que otorgando suficiente tiempo o recursos cualquier elemento podrá ser descubierto por el atacante, motivo por el cual deben planearse las medidas correspondientes para que la información tenga un tiempo de vida útil limitado o considerar el posible impacto a futuro.

Las llaves y los algoritmos de cifrado que las emplean pueden ser de dos tipos:

- **Llave secreta o llave simétrica.** Tiene la propiedad de que si se conoce la llave usada para cifrar un mensaje es fácil obtener la llave a usar para descifrarlo. La dificultad de obtener la llave por ataques de fuerza bruta a menudo crece de forma exponencial conforme se usen llaves más grandes, por lo que dadas las capacidades de cómputo actuales se recomiendan llaves de al menos 256 bits. Este tipo de algoritmos suelen tener muy buen rendimiento pero requieren que tanto el que envía como el que recibe el mensaje conozca la llave.
- **Llave pública.** Estos algoritmos tienen la propiedad de emplear llaves distintas para el cifrado y para el descifrado haciendo virtualmente imposible descubrir la llave de descifrado conociendo solo la llave de cifrado. En estas circunstancias, la llave de cifrado puede darse a conocer manteniendo secreta solo la llave de descifrado. Estos algoritmos son muy populares porque aprovechan la enorme carga de trabajo que representa intentar obtener una de estas llaves por ataques de fuerza bruta, incluso si el uso del cifrado con las llaves también resulta mucho más demandante que los algoritmos de llave simétrica.

Adicionalmente al cifrado basado en llaves, hay otras técnicas de cifrado que son empleadas con frecuencia para prevenir diversos problemas de seguridad.

Funciones no reversibles

Estas funciones tienen la característica de que ejecutarlas es relativamente sencillo para transformar un dato empleado como parámetro, pero si conocemos la función y el resultado obtenido, es prohibitivo el proceso necesario para determinar el parámetro que se empleó originalmente. A estas funciones se les conoce como **funciones criptográficas de dispersión** (Cryptographic Hash Function), ya que comparten con las funciones de dispersión o hash el que pequeñas modificaciones en el parámetro de entrada deben generar una salida con grandes diferencias. Esta carencia de patrones en la salida es parte integral de la dificultad de calcular el parámetro de entrada a partir de la salida obtenida.

La mayoría de los sistemas UNIX usan una función no reversible para almacenar las contraseñas de los usuarios. Para verificar que una solicitud de inicio de sesión está empleando la contraseña correcta, el sistema repite la aplicación de la función a la que ingresa el usuario y verifica si corresponde con la almacenada en la tabla correspondiente. Como solo se almacena el resultado de aplicar la función, incluso si un atacante logra obtener la tabla de contraseñas, no resulta sencillo determinar los datos originales que le dieron origen a la información ahí contenida. Además, para evitar que la contraseña de un sistema pueda reconocerse al ser utilizada en otro (por ejemplo, en un ataque de fuerza bruta), las funciones empleadas comúnmente en UNIX usan "sal", que consiste en agregar al parámetro de la contraseña un dato particular del sistema que lo está empleado, con lo que se obtiene un resultado diferente al que obtendría otro sistema para la misma contraseña.

Firmas digitales

A menudo también se requiere verificar que quien origina el mensaje sea quien dice ser, de forma que incluso el remitente deba reconocer su autoría aun si deja de convenir a sus intereses. Para ello se emplean las firmas digitales.

A *grosso modo*, las firmas digitales tienen dos pasos. El primero es obtener el resultado de aplicar una función hash que genere una salida de longitud limitada al mensaje o elemento a firmar. Esta función y su funcionamiento se conocen y aplican por ambas partes.

La salida de la función hash cambiará de manera significativa si el mensaje tiene cambios, por lo que es útil para detectar alteraciones al mensaje.

El segundo paso consiste en aplicar un cifrado de llave pública al código generado por la función hash, y el resultado se anexa al mensaje.

Cuando el mensaje debe validarse, se aplica de nueva cuenta la función hash sobre el mensaje y se compara el resultado cifrado contra el que se anexó con el mensaje.

A menudo se emplean algoritmos como RSA, en los que el resultado de aplicar el cifrado con la llave pública sobre el cifrado de la llave privada es igual a aplicar el cifrado de la llave privada sobre el resultado de la llave pública, lo que simplifica la comparación y permite únicamente aplicar el cifrado de llave pública sobre la firma que se anexa con el mensaje.

EVALUACIÓN

8.1 ¿Qué elemento medible y objetivo se emplea para limitar los recursos invertidos en medidas de seguridad?

8.2 ¿Qué diferencia a los sistemas de misión crítica de los convencionales?

8.3 ¿Cuál es la diferencia entre la confidencialidad y la privacidad de los datos?

8.4 ¿En qué consiste la seguridad a profundidad?

8.5 ¿Las instituciones gubernamentales siempre actúan en favor de la seguridad informática? ¿Cómo pueden sostenerse cuando no es el caso?

8.6 Explica la importancia del principio de la cultura en el criterio integral para la seguridad en sistemas de cómputo.

8.7 ¿En contra de qué derecho estipulado en la Ley Federal de Protección de Datos personales iría el que una compañía se negase a eliminar la cuenta y los datos asociados alegando que no tiene saldo ni costo, con tal de prolongar la relación con un cliente que ya no la desea?

8.8 ¿En qué consisten los datos personales sensibles y por qué se les da un tratamiento especial en la legislación?

8.9 ¿Sería benéfico o perjudicial para la seguridad de los sistemas promover que todos ellos se implementaran con el mecanismo de cifrado más resistente de manera exclusiva? ¿Por qué?

8.10 Explica las razones por las que se debe emplear la seguridad a profundidad aun si un proyecto de desarrollo solo tiene requerimientos para software.

8.11 ¿Cuáles son las etapas mínimas que debe tener la respuesta a un incidente o falla de seguridad?

8.12 ¿Cuáles son las tres categorías que se definieron para las amenazas de seguridad?

8.13 ¿Consideras que el tratamiento de las amenazas de seguridad es exclusivo para los sistemas de información o que podría emplearse en otros ámbitos de la actividad profesional de un ingeniero?

8.14 Al observar el PIN en el cajero automático de otro cuentahabiente, ¿cuál táctica básica de la ingeniería social se estaría aplicando?

8.15 Investiga un ejemplo documentado de un virus bélico como Stuxnet y evalúa la posibilidad de que un sistema en tu propia escuela o lugar de trabajo sea afectado basado en el número de equipos que hayan sido afectados por ese ataque en particular.

8.16 Revisa desarrollos recientes en los que hayas participado e identifica una vulnerabilidad propia del sistema, ya sea que la hayan detectado y corregido a tiempo o que haya permanecido en el sistema.

8.17 Menciona un ejemplo de autenticación que utilices hoy día de forma personal y que emplee más de un tipo de mecanismo de autenticación.

8.18 Explica la importancia de que los atacantes suelen modificar o eliminar las bitácoras de un sistema desde la perspectiva de la detección de intrusiones.

8.19 Explica las características principales que se buscan en el resultado de una función hash.

8.20 Menciona la diferencia entre una llave simétrica y un modelo de llave pública y privada.

8.21 Menciona o investiga un caso de uso de firmas digitales prestando atención a las características de las llaves empleadas.

Referencias bibliográficas

Tanenbaum, Andrew S, *Modern Operating Systems*, 3a ed., Pearson, 2009.

Secure Software Development, *Guide to the Software Engeneering Body of Knowledge*, versión 3.0, 2014.

Stallings, William, *Operating Systems, internals and design*, Aptara, 5a ed., Pearson, 2008

Referencias electrónicas

OECD Guidelines for the Security of Information Systems and Networks:
http://www.oecd.org/internet/ieconomy/15582260.pdf

Resumen de las directrices de la OCDE sobre protección de la privacidad y flujos transfronterizos de datos personales, OCDE 2002.
http://www.oecd.org-sti-ieconomy-15590267.pdf

OECD Guidelines for Cryptography Policy.
http://www.oecd.org-sti-ieconomy-guidelinesforcryptographypolicy.htm

WACS Threat clasification, versión2.0, Web Application Security Consortium 2010.
HTTP://WWW.WEBAPPSEC.ORG

Data genetics, PIN number. Análisis. Septiembre 2012:
http://www.datagenetics.com/blog/september32012/

RFC 4949, Internet Security Glossary, Version 2, IETF Trust 2007:
https://tools.ietf.org/html/rfc4949

Past, present, and future methods of cryptography and data encryption, Nicholas G. McDonald:
http://www.eng.utah.edu/~nmcdonal/Tutorials/EncryptionResearchReview.pdf

Guide to Intrusion Detection and Prevention Systems (IDPS), Karen Scarfone y Peter Mell, NIST, 2007:
http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf

Glosario

A

Actor de amenaza de seguridad. Se dice del agente, sea una persona o sistema, que toma un rol activo en una amenaza de seguridad.

Alarma. Son un tipo de señal con la que un proceso puede enviarse a sí mismo una notificación después de un periodo de espera.

Almacenamiento secundario. Colección de dispositivos de almacenamiento, por lo regular no volátil, donde se puede amasar la mayor cantidad de información almacenada por el menor costo por MB. Como los tiempos de acceso, lectura y escritura de esta información suelen ser de varios órdenes de magnitud más largos que los de la memoria RAM, constituye una segunda etapa de almacenamiento, ya que toda la información se maneja primero en los otros tipos de memoria.

Alta disponibilidad. Protocolo de diseño e implementación de sistemas que busca garantizar la habilidad del sistema para responder a las peticiones de sus usuarios por una proporción objetivo del tiempo en que se desee que esté en operación. Su principal objetivo es ofrecer redundancia en el sistema para evitar que el fallo de un solo componente pueda ocasionar la suspensión del servicio, así como reducir de manera drástica la posibilidad de este tipo de fallos.

API (*Application Programming Interface*). Conjunto de funciones y estructuras de datos (*clases*, *objetos* y *métodos* si se desarrolla en un lenguaje de programación orientado a objetos) que resuelven un grupo específico de requerimientos o dan acceso a determinada funcionalidad del sistema, por lo común reunidas en un paquete, que se distribuye con el objetivo de que se emplee en múltiples programas desarrollados posteriormente.

Aplicación o sistema legacy. Se dice de aquellos sistemas desarrollados con anterioridad que desempeñan funciones críticas en una empresa, lo que prolonga su tiempo de vida y suele conducir a dificultades para darles mantenimiento y desarrollar interfaces con tecnologías más modernas.

ATA. Conjunto de normas de operación de discos duros, unidades de CD-ROM, DVD, disquetes y unidades de cinta. En la actualidad, su versión más popular es el Serial-ATA, que hasta hoy día tiene tres versiones con velocidades de transferencia progresivamente más elevadas.

B

B2B (Business to Business). Norma de comunicación entre sistemas de negocios orientado a uniformar la implementación de interfaces de sistemas de clientes y proveedores para abastecer cadenas de producción.

Balanceo de carga. Técnicas empleadas en los sistemas distribuidos para distribuir la carga de trabajo entre los nodos participantes.

Bloqueo. Es el estado de un proceso en el que este deja de recibir atención de la CPU.

Bot net. Conjunto de computadoras que se encuentran afectadas por software malicioso, que pueden recibir diversas tareas o procesos a ejecutar por parte de un atacante para hacer que los equipos afectados participen en ataques de seguridad a gran escala, casi siempre sin conocimiento de sus usuarios legítimos.

Busy Waiting. Técnica de programación en la que durante el tiempo que tarda en ocurrir una condición, el programa entra en un ciclo en el que prueba la condición repetidas veces hasta que esta resulta verdadera. Notable por el consumo de recursos de procesamiento que realiza, hecho por el cual suele evitarse, en especial para periodos largos de espera.

C

Cognitive Bias. Tendencias a pensar de determinadas formas, que pueden llevar a desviaciones respecto de una norma de racionalidad o buen juicio, a menudo estudiadas por la psicología y economía conductual.

Colisión. Es el evento en el que se pierde o corrompe la información, también conocido como estado de un sistema, debido a una condición de carrera.

Concurrencia. Es la capacidad que tiene un sistema de realizar transiciones de estado aparentemente simultáneas mediante la rápida alternancia de atención de los procesos ejecutándose en un procesador. También contempla la capacidad de dichos procesos de comunicarse e interactuar entre sí.

Condición de competencia. Es el comportamiento de un sistema por el cual el estado final de una operación depende del momento en que se reciben diversas entradas, bajo circunstancias en las que la sincronía de estas entradas no puede predecirse de forma práctica. Esto genera variaciones impredecibles en la salida que a menudo constituyen errores en la operación y pueden corromper la información que se maneja. Esta también se conoce como race condition.

Contigüidad. Característica en la que todas las direcciones, o páginas, de memoria están adyacentes o juntas las unas de las otras en un único rango continuo y sin interrupciones.

CORBA (Common Object Request Broquer Architecture). Estándar definido por la Open Management Group que permite a diversos componentes de software trabajar juntos aun cuando se desarrollen en diversos lenguajes de programación y se

ejecuten en equipos distintos y remotos. Es uno de los primeros estándares para sistemas distribuidos. Se caracteriza porque soporta aplicaciones de misión crítica en múltiples empresas.

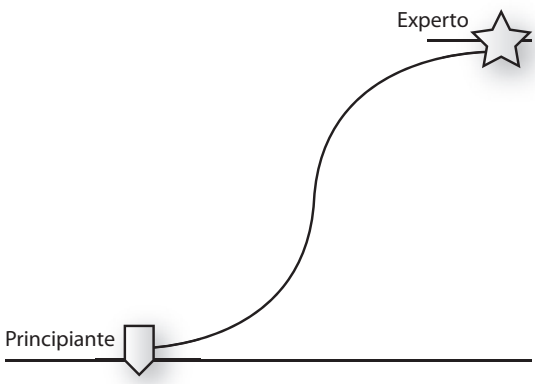


Figura G1

Curva de aprendizaje. Progresión de la habilidad adquirida de un conjunto de habilidades técnicas a lo largo del tiempo, aprendidas durante el inicio del aprendizaje del tema. Se considera que es una asíntota donde al principio se tiene la máxima ganancia de habilidad en el tiempo y se tiende a una asíntota de dominio del tema. Conforme un conocimiento sea más difícil de aprender, la pendiente inicial en la curva será menos pronunciada y el tiempo de productividad de dicho conocimiento será más largo.

D

Decodificador. Circuito electrónico que recibe un número código, a partir del que genera una señal en sus conectores de salida. Para la habilitación de dispositivos por direcciones de memoria se usa particularmente un *decodificador lineal*, que convierte un número binario de n dígitos en una señal en una de 2^n líneas de salida.

Democracia. Sistema político donde las decisiones importantes, o al menos la designación de representantes, involucra la participación directa de los ciudadanos.

E

EDI (Electronic Data Interchange). Intercambio de dato estructurados por medios electrónicos con el objetivo de dar soporte a las operaciones comerciales entre empresas.

EJB (Enterprise Java Beans). Parte de las especificaciones de J2EE (Java 2 Enterprise Edition) respecto a la representación de registros de información de relevancia para sistemas empresariales que incluyen consideraciones de encapsulamiento y serialización, entre otras.

Estructura de datos. Elemento de la gramática de ANSI C que representa agrupaciones de atributos como un tipo de datos nuevos, sin necesidad de asociarle un comportamiento. Desde una perspectiva de programación orientada a objetos, permite definir clases que solo contienen atributos y no métodos.

Ética. Rama de la filosofía que se encarga de sistematizar, defender y recomendar conceptos del bien y del mal. Respecto a la conducta de las personas, esta busca determinar qué es lo que una persona estaría obligada a hacer bajo diversas circunstancias.

F

Fallo de página. En la memoria virtual, es el intento de utilizar una página que en ese momento se encuentra solo en el almacenamiento secundario. Implica bloquear el proceso que solicitó la operación en lo que se puede cargar la página a memoria principal.

Fuga de memoria. Se dice de un error en un programa, que provoca que parte de la memoria que ocupa no se libere ni aún al terminar el proceso.

G

GNU (GNU is Not UNIX). Definición recursiva usada por Richard Stallman para denominar la colección de utilerías y elementos del sistema operativo desarrollados como software libre con el objetivo de generar una alternativa a los sistemas UNIX propietarios de finales del siglo XX.

H

Hack. Se dice del software desarrollado de forma informal y rápida pero que logra sus objetivos y a menudo consigue una funcionalidad extraordinaria. Su acepción original no es negativa y puede referirse a avances repentinos en el desarrollo de un sistema.

Hipervisor. Programa encargado de supervisar y controlar la ejecución de múltiples máquinas virtuales en una computadora.

J

JCA (Java Enterprise Edition Conector Architecture). Solución tecnológica orientada a la conectividad de sistemas Java con aplicaciones legacy, de aplicación empresarial o bases de datos.

JMS (Java Messaging Service). Protocolo para transferencia de mensajes entre máquinas virtuales Java mediante protocolos de red.

L

Librería de carga dinámica (DLL). Conjunto de rutinas que serán utilizadas por aplicaciones de usuario, pero que no se encuentran ligadas estáticamente a los

procesos. Estas rutinas se cargan de forma independiente y son ligadas al ejecutar los procesos.

M

Máquina virtual. Simulación de una computadora realizada por software que opera en el hardware de una computadora conocida como Anfitrión (Host). La computadora virtual, o invitada (Guest), puede tener características diferentes a las de su anfitrión; por lo regular, carga sistemas operativos diferentes o con configuraciones diferentes y expone solo una fracción de los recursos del anfitrión al software que opera dentro de la máquina virtual.

Memoria virtual. Simulación de una cantidad de memoria RAM mayor a la disponible físicamente en el sistema mediante el intercambio de páginas que no se estén usando en ese momento con el almacenamiento secundario.

MPEG (Moving Picture Expert Group). Grupo de trabajo de la ISO e IEC para establecer estándares de almacenaje y transmisión de audio y video. Algunas de las normas más populares que ha generado son el MP3 para archivos de audio, MPEG-2 para transmisión y almacenamiento de video digital, usado por las estaciones de TV digital por todo el mundo, y MP4 para empaquetar información de audio, video y medios de apoyo en un solo flujo de información o archivo.

MQ de IBM. De *Message Queue* o fila de mensajes. Producto principal para la implementación del paso de mensajes de IBM.

Multiprocesamiento. Es la capacidad de un sistema de asignar tareas o aprovechar más de un procesador en la atención de sus procesos.

N

Name space. En lenguajes de programación orientados a objetos como C# se refiere a la jerarquía de clases y paquetes en la que se encuentra la definición de una clase en particular. En el caso de las clases que están registradas en el proyecto o en el ambiente de programación, basta con conocer el **name space** de una clase para que el ambiente de desarrollo pueda incluir las librerías correspondientes.

O

OECD (Organization for Economic Cooperation and Development). Organismo internacional que tiene como objetivo promover las políticas que lleven al desarrollo sustentable, así como al desarrollo del comercio y la estabilidad económica de sus estados miembros y del resto del mundo.

P

Page Frame o Marco de página. Se refiere a la dirección física en la que se encuentra una página en la memoria RAM, por lo que a las páginas que se encuentran en la RAM, a menudo se les conoce así, en vez de llamarlas solo página. Por esta razón, un marco de página nunca estará en almacenamiento secundario.

Página. Rango de direcciones de memoria contigua de tamaño fijo y uniforme en el sistema. Se asignan tantas páginas a los procesos como se requiera para satisfacer cada petición de memoria. Tienen la característica de que el conjunto de páginas no requiere ser contiguo y pueden almacenar su información en la memoria RAM y en almacenamiento secundario, como una memoria Flash o un disco duro. Los procesos solo pueden usar las páginas que se encuentran en la RAM, por lo que en caso de requerir una que solo se encuentre en almacenamiento secundario, esta deberá cargarse primero a la RAM.

PCI (*Peripherial Component Interconnect*). También conocidas como *ranuras de expansión*, son un bus típico en las mother boards de computadoras IBM PC compatibles para agregar controladoras de diversos tipos de puertos y otros dispositivos que requieren acceso a líneas de control IRQ y rangos de memoria para mapear sus buses para intercambiar valores con la CPU y que no serán conectadas y desconectadas del sistema con frecuencia ni durante la operación.

Polimorfismo. En programación, característica orientada a objetos. Por tanto, los objetos generados a partir de clases que son derivadas de una clase "padre" genérica, pueden utilizarse como si fueran instancias de la clase genérica, independientemente de la clase derivada con la que fueron generados.

Política (empresarial). Sistema de principios creado para guiar decisiones y lograr resultados racionales.

Privacidad. Habilidad de una persona o grupo de segregarse a sí mismo o a la información de sí mismo y, por tanto, expresarse de modo selectivo.

R

RAM (Random Access Memory) Memoria de Acceso Aleatorio. Memoria implementada, por lo regular, mediante semiconductores. Ofrece una elevada velocidad de lectura y de escritura, pero que no requiere estar integrada al procesador. Asimismo, permite implementar una cantidad considerable de almacenamiento por un precio accesible. Se distingue por no requerir acceso secuencial a la información que almacena como muchos medios de almacenamiento secundario y por ser volátil.

Reemplazo de página. Es el proceso de liberar una página de la memoria principal, con el fin de contar con memoria que se usará para cargar la información de otra página de memoria, ya sea para satisfacer una solicitud de memoria adicional o un fallo de página.

Revisión estática de código. Tipo de pruebas donde no se ejecuta el programa, sino que se revisa con una serie de reglas gramaticales en búsqueda de patrones que indiquen errores o malas prácticas de desarrollo. Como estas búsquedas suelen arrojar algunos falsos positivos y consumir mucho más tempo que una compilación normal, es común que no se integren en su totalidad a los compiladores.

RMI (Remote Method Invocation). Solución tecnológica implementada de manera original en los estándares POSIX y después extendida en diversas plataformas de desarrollo que permite utilizar métodos o funciones en procesos remotos con la misma sintaxis con la que se invocan métodos locales.

RPC (Remote Procedure Call) - Llamada de procedimiento remoto. Mecanismo de comunicación que permite que un programa ejecute otro programa en un equipo remoto sin necesidad de implementar los protocolos de comunicación.

S

S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology). Parte de la especificación del standard modern para discos ATA y SCSI que define cómo los dispositivos deben vigilar su propio funcionamiento y desempeño, emitir notificaciones en caso de detectar problemas o señales de alarma e incluso evitar que siga operando bajo condiciones que pudieran destruir la información contenida.

Figura G2. Scytale. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Skytale.png#/media/File:Skytale.png

Scytale. Instrumento antiguo para cifrar mensajes. Para su uso de enrolla una tira de cuero o papel en un cilindro y se escribe el mensaje de forma transversal a la tira para cambiar el orden de las letras. Para descifrar el mensaje se utiliza un segundo cilindro del mismo diámetro (véase figura G2).

Segmento. Rango de direcciones de memoria contiguas que se asigna a un proceso para que opere. Son de la longitud que se especifique en el momento de la solicitud de memoria.

Señal. Es un mecanismo de notificación entre procesos mediante el cual se avisa a la planificación de procesos el hecho de que un proceso en particular debe recibir un código de operación; por lo general, este se representa por un número entero. Esto significa que si el proceso está bloqueado, deja de estarlo y retorna a esperar turno de atención por parte

de la CPU, para luego proceder a ejecutar el comportamiento por defecto asociado a la señal recibida o por una rutina de su propio programa que haya registrado como de atención a la señal (handler). Por defecto, se sabe que diversas señales tienen comportamientos distintos; por ejemplo, son ignoradas o terminan la ejecución del proceso.

Serialización. En el contexto de la concurrencia, se dice de las operaciones que se ordenan de forma seriada y deben realizarse en una secuencia estricta, por lo que no pueden ser concurrentes.

SOAP (Service Oriented Application Protocol). Protocolo de mensajería basado en el intercambio de mensajes codificados en XML mediante conexiones que empleen el protocolo de comunicación HTTP sobre redes TCP/IP. Se distingue porque incluso la definición de los mensajes, o metadatos, está codificado en XML mediante archivos WSDL en WSL.

Splint. De *Secure Programming Lint*. Realizado por el Inexpensive Program Analysis Group de la universidad de Virginia en Estados Unidos de América. Utilería de revisión estática de código para C y C++.

Stack – Pila. Estructura de datos que permite insertar y recuperar elementos de esta. Tiene la característica de que el último elemento en ser insertado es el primero en ser recuperado (Last In First Out). Se emplea comúnmente para llevar el registro la instrucción en la que se encuentra la ejecución de un programa, indicado por la dirección contenida en el Program Counter (PC) del procesador, cuando se bifurca para ejecutar una subrutina. Cuando la subrutina termina se recupera el valor del PC para retornar al punto en donde se hizo la bifurcación.

SWIFT (Society for Worldwide Interbank Financial Telecommunication). *Sociedad para las Comunicaciones Interbancarias y Financieras Mundiales* que mantiene una red internacional de comunicaciones por la que se intercambian mensajes que respaldan operaciones financieras entre bancos y otras entidades financieras. Se utiliza a nivel global.

T

Time. Se dice del registro del procesador que se incrementa conforme oscila un cristal de cuarzo sincronizado a una potencia de dos Hertz, conocido como Real Time Clock; por ejemplo, 32768 Hz para la mayoría de los PC, relojes digitales, GPS y otros. Empleado para hacer mediciones precisas de tiempo en las unidades estándar (milisegundos, segundos, minutos, etc.).

TLB. Table Lookaside Buffer. Memoria especializada en almacenar la información para traducir direcciones lógicas de tablas en direcciones físicas y donde se alberga di-

cha información en el procesador para acelerar la resolución; sin embargo, esto limita su tamaño y el número de páginas que puede recordar.

TLB Miss. Fallo de Tabla de Lookaside Buffer. Operación en la que se tiene que buscar la referencia de página en la tabla de páginas albergada en la RAM, por no encontrarla en el TLB.

U

UTP5 (*Unsheldeded Twisted Pair 5*). Especificación para la elaboración de cables de red sin blindaje, de acuerdo con la norma EIA/TIA 568B, habituales en las redes de área local Ethernet actuales.

V

Vecindad. De *Locality Principle*. Principio formulado por Peter J. Denning que plantea el concepto de vecindad definido como aquellos recursos que:

- Dada una función llamada D(x,t), que calcule la distancia del procesador a un recurso x en el tiempo t.
- El recurso x pertenecerá al conjunto de vecindad si para un tiempo t, su distancia es menor a un umbral T; es decir: $D(x, t) \leq T$.
- En la administración de memoria, el concepto de vecindad a menudo se aplica como una función de distancia definida con respecto al tiempo transcurrido desde la última referencia a las posiciones de memoria.
- Las posiciones pertenecientes a la vecindad presentan una mucho mayor probabilidad de ser usadas en el futuro próximo con respecto a las demás, por lo que son idóneas para utilizar los dispositivos de memoria más rápidos disponibles.

Virtualización. Es la simulación de un componente o recurso empleando un recurso diferente, a través de una capa de software intermedia.

W

Web service (Servicio Web). Tecnología para la interoperabilidad de sistemas. En particular, constituye un método de comunicación entre dos computadoras mediante una red de forma interoperable y donde la descripción del contenido de la comunicación (contrato) se presenta en un formato interpretable de forma automática. A menudo utiliza HTTP como protocolo de comunicación y XML para la codificación de los mensajes.

Working Set (Conjunto de trabajo). Es el conjunto de páginas W(t, T) referenciadas en el intervalo virtual de longitud *T* – precio al momento en el tiempo t. Según la definición formal de Peter J. Denning, en su artículo "The working set model for program behavior", en *Communications of the ACM*, Volumen 11, Número 5, Mayo 1968.