Sistemas Operativos:

una guía de estudios

Dr Luis Castellanos

2014

Sistemas Operativos: una guía de estudios

Dr Luis Castellanos

En Maracaibo, 2014.

Revisión 2015a.


http://luiscastellanos.wordpress.com


luiscastellanos@yahoo.com


@lrcastellanos

Imagen de Portada tomada de:

http://www.poderpda.com/wp-content/uploads/2012/12/sistemaasss-op.png

Prefacio

La finalidad del presente documento es facilitar a los estudiantes de la asignatura de "Sistemas Operativos", el material referencial y bibliográfico básico que debe manejar durante el desarrollo de la misma.

No se debe sustituir un buen libro de texto por una sencilla guía de estudios. Y una guía es simplemente una serie de indicaciones, con referencias adicionales que van a coadyuvar al estudiante. Particularmente recomiendo "Sistemas Operativos Modernos" de Andrew Tanembaum, sobre el cual he hecho este documento. Debo aclarar que la mayoría del material es tomado, con propósitos didácticos, del libro de Tanembaum.

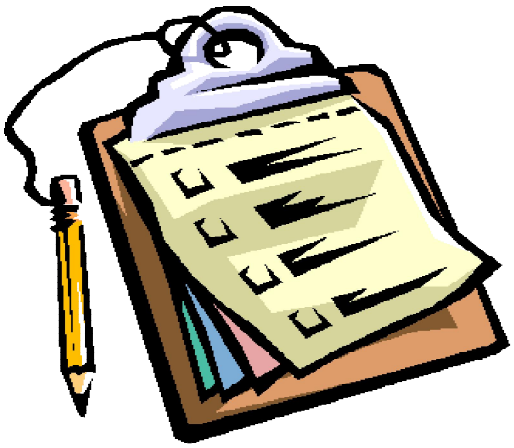
La estructura se hará de acuerdo con los siguientes temas, que se pueden detallar en la tabla de contenidos correspondiente.

- Conceptos Generales
  - Entrada y Salida
- Administración de Procesos
- Administración de Memoria
- Administración de Información
- Sistemas Operativos Modernos
  - Referencias Bibliográficas

Espero que sea de utilidad para todos los estudiantes que puedan tener acceso al documento presentado.

Dr Luis Castellanos

Contenido



1. Conceptos Generales

El sistema operativo es el programa que controla los diferentes trabajos que realiza la computadora. Un trabajo importante es la interpretación de los comandos que permiten al usuario comunicarse con la computadora. Algunos intérpretes de estos comandos están basados en texto y exigen que los comandos sean introducidos mediante el teclado. Otros están basados en gráficos, y permiten al usuario comunicarse señalando y haciendo clic en un icono. Por lo general, los intérpretes basados en gráficos son más sencillos de utilizar.

El sistema operativo tiene entre sus funciones: Coordinar y manipular el hardware de la computadora (como la memoria, las impresoras, las unidades de disco, el teclado o el ratón), organizar el almacenamiento de los archivos en diversos dispositivos (como discos flexibles,

Imagina que la computadora es el autobús. ¿Quién lo maneja? Si no hay quien lo maneje, no camina.

El Sistema Operativo es el chofer del autobús. Sin Sistema Operativo, la computadora no arranca.

discos duros, discos compactos o cintas magnéticas), y supervisar la ejecución de las diferentes tareas.

Los sistemas operativos pueden ser de tarea única o multitarea. Los sistemas operativos de tarea única, más primitivos, sólo pueden manejar una tarea en cada momento. Por ejemplo, cuando se está editando un documento la computadora no puede iniciar otra tarea ni responder a nuevas instrucciones hasta que se termine la edición del documento.

Todos los sistemas operativos modernos son multitarea y pueden ejecutar varias tareas simultáneamente. En la mayoría de las computadoras sólo hay una CPU, por lo que un sistema operativo multitarea debe compartir este CPU entre las distintas tareas que se ejecutan, creando la ilusión de que estas tareas se ejecutan simultáneamente en la CPU. El mecanismo que se emplea más a menudo para lograr esta ilusión es la multitarea por segmentación de tiempos, en la que cada tarea se ejecuta individualmente durante un periodo de tiempo determinado.

Si la tarea que se ejecuta en la CPU no finaliza en el tiempo asignado, ésta se suspende y se ejecuta otra tarea. Este intercambio de tareas se denomina conmutación de contexto. El sistema operativo se encarga de controlar el estado de las tareas suspendidas. También cuenta con un mecanismo llamado planificador que determina la siguiente tarea que debe ejecutarse. El planificador ejecuta las tareas basándose en su prioridad para minimizar el retraso percibido por el usuario. Las tareas parecen efectuarse simultáneamente por la alta velocidad de procesamiento que poseen los modernos procesadores o CPU, lo que hace que la conmutación de contexto entre las diferentes tareas tome muy poco tiempo.

Los sistemas operativos pueden emplear memoria virtual para ejecutar tareas que exigen más memoria principal de la realmente disponible. Con esta técnica se emplea espacio en el disco duro para simular la memoria adicional necesaria. Sin embargo, cuando el CPU requiere una tarea que ha sido pasada a memoria virtual (en disco duro), ésta debe ser llevada de nuevo a la memoria principal antes de poder ser procesada. El acceso al disco duro requiere más tiempo que el acceso a la memoria principal, por lo que el funcionamiento de la computadora cuando se utiliza la memoria virtual se hace más lento (Ramírez, I. s/f)

Tener en cuenta: el único Sistema Operativo que existe en el mundo no es MS Windows.

Gráfica 1 Sistemas Operativos para Computadores Personales.

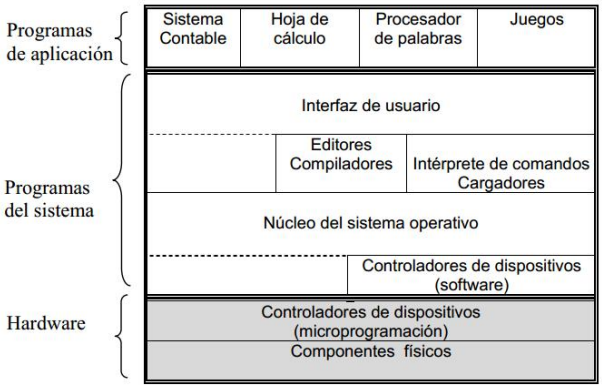
Gráfica 2 Gamas de Computadoras. Fuente: Elaboración propia.

Hoy en día, la computación está en todas partes, y tenemos una gran gama de ellas, desde las más grandes (Mainframes) hasta las más pequeñas (teléfonos inteligentes). Y en cada gama de computadoras, se van a encontrar distintos sistemas operativos. Para Mainframes hay una gama de sistemas operativos, distintos a los que se consiguen en las Estaciones de Trabajo, distintos a los que se instalan en las Computadoras Personales, distintos a los usados en las Tabletas, en los teléfonos inteligentes, y hasta en lavadoras, microondas y neveras.

1.1. Concepto de Sistema Operativo

Toda computadora está conformada por dos componentes, el hardware y el software. Siendo el software o programas la parte no física o lógica que hace funcionar a la computadora, los que a su vez se clasifican en programas del sistema y programas de aplicación. El software se ejecuta sobre la plataforma de hardware.

Los programas del sistema son los programas básicos e indispensables para poder utilizar la computadora, ya que manejan directamente la operación de la computadora (manejan y controlan el hardware de la misma). A los programas del sistema pertenecen los programas que conforman los sistemas operativos. Estos programas pueden ser desarrollados por el fabricante del equipo o por una casa de software independiente.

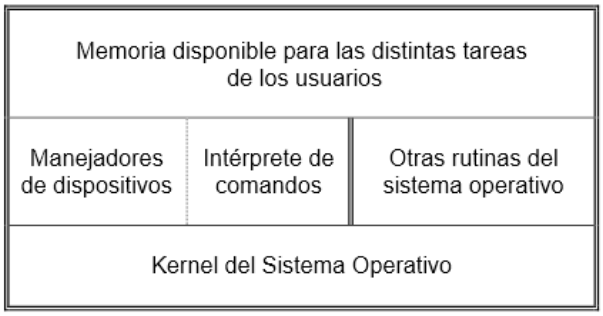


Gráfica 3 Componentes de un Computador. Fuente: Ramírez (s/f)

Ramírez (s/f) presenta la siguiente definición de Sistema Operativo:

> El Sistema Operativo es el programa maestro que controla todo el trabajo que realiza una computadora, (incluyendo el control de la misma computadora y la ejecución de los diferentes programas), que para ser funcional debe proporcionar al usuario una interfaz que le permita interactuar fácilmente con la computadora.

El sistema operativo, que está almacenado en algún medio de almacenamiento secundario, es el primer programa que se carga (copia), en la memoria principal (RAM) de la computadora después de que ésta es encendida, y el núcleo central (kernel) del mismo debe estar siempre en la memoria principal (por lo que sus rutinas pueden ser usadas por cualquier otro programa que las requiera) y se mantiene en ejecución cuando no se está procesando ninguna otra tarea,



Gráfica 4 Estructuración de la memoria principal. Fuente: Ramírez (s/f)

atento a procesar cualquier requerimiento del usuario.

Se debe tener presente que al apagarse la computadora los programas del sistema operativo, como cualquier programa, desaparecen de la memoria principal. La naturaleza del diseño y construcción de las computadoras, por lo general, hace que se requiera la presencia del sistema operativo cada vez que son utilizadas.

Algunas veces, erróneamente, se dice que el sistema operativo no realiza funciones útiles para los usuarios, sino que simplemente provee un ambiente dentro del cual otros programas hacen un trabajo útil, lo que no es cierto, ya que el sistema operativo es el gran administrador de los recursos del sistema, es el que controla la comunicación entre éstos y los programas de aplicación y, por tanto, es el que determina el ambiente general en que se realiza la actividad de programación y fija los estándares para los programas de aplicación que se podrán ejecutar, por lo que estos programas deben ser escritos para interactuar con el sistema operativo, dejando a un lado a la computadora misma.

1.2. Evolución

La evolución de los Sistemas Operativos ha ido siempre de la mano de la evolución de las Computadoras.

Por ello es relevante recordar las generaciones de Computadores (Tanembaum, 2009):

1. **Tubos al Vacío (1945-1955)**

Gráfica 5 Tubos al vacío de la ENIAC. Fuente: http://upload.wikimedia.org/wikipedia/commons/c/c9/ENIAC_Penn2.jpg

Después de los esfuerzos infructuosos de Babbage, no hubo muchos progresos en la construcción de computadoras digitales sino hasta la Segunda Guerra Mundial, que estimuló una explosión de esta actividad. El profesor John Atanasoff y su estudiante graduado Clifford Berry construyeron lo que ahora se conoce como la primera computadora digital funcional en Iowa State University. Utilizaba 300 tubos de vacío (bulbos). Aproximadamente al mismo tiempo,

Konrad Zuse en Berlín construyó la computadora Z3 a partir de relevadores. En 1944, la máquina Colossus fue construida por un equipo de trabajo en Bletchley Park, Inglaterra; la Mark I, por Howard Aiken en Harvard, y la ENIAC, por William Mauchley y su estudiante graduado J. Presper Eckert en la Universidad de Pennsylvania. Algunas fueron binarias, otras utilizaron bulbos, algunas eran programables, pero todas eran muy primitivas y tardaban segundos en realizar incluso hasta el cálculo más simple. A principios de la década de 1950, la rutina había mejorado un poco con la introducción de las tarjetas perforadas.

2. **Transistores (1955-1965)**

La introducción del transistor a mediados de la década de 1950 cambió radicalmente el panorama. Las computadoras se volvieron lo bastante confiables como para poder fabricarlas y venderlas a clientes dispuestos a pagar por ellas, con la expectativa de que seguirían funcionando el tiempo suficiente como para poder llevar a cabo una cantidad útil de trabajo. Por primera vez había una clara separación entre los diseñadores, constructores, operadores, programadores y el personal de mantenimiento.

Gráfica 6. IBM 1401. Fuente: http://ibm-1401.info/1401_ProcessingUnit.jpg

Estas máquinas, ahora conocidas como mainframes, estaban encerradas en cuartos especiales con aire acondicionado y grupos de operadores profesionales para manejarlas. Sólo las empresas grandes, universidades o agencias gubernamentales importantes podían financiar el costo multimillonario de operar estas máquinas. Para ejecutar un trabajo(es decir, un programa o conjunto de programas), el programador primero escribía el programa en papel (en FORTRAN o en ensamblador) y después lo pasaba a tarjetas perforadas. Luego llevaba el conjunto de tarjetas al cuarto de entrada de datos y lo entregaba a uno de los operadores; después se iba a tomar un café a esperar a que los resultados estuvieran listos. Cuando la computadora terminaba el trabajo que estaba ejecutando en un momento dado, un operador iba a la impresora y arrancaba las hojas de resultados para llevarlas al cuarto de salida de datos, para que el programador pudiera recogerlas posteriormente. Entonces, el operador tomaba uno de los conjuntos de tarjetas que se habían traído del cuarto de entrada y las introducía en la

máquina. Si se necesitaba el compilador FORTRAN, el operador tenía que obtenerlo de un gabinete de archivos e introducirlo a la máquina.

Gráfica 7. IBM 7094. Fuente: http://ed-thelen.org/comp-hist/vs-ibm-7094.jpg

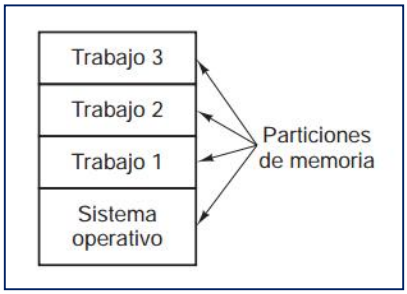
Se desperdiciaba mucho tiempo de la computadora mientras los operadores caminaban de un lado a otro del cuarto de la máquina. Dado el alto costo del equipo, no es sorprendente que las personas buscaran rápidamente formas de reducir el tiempo desperdiciado. La solución que se adoptó en forma general fue el sistema de **procesamiento por lotes**. La idea detrás de este concepto era recolectar una bandeja llena de trabajos en el cuarto de entrada de datos y luego pasarlos a una cinta magnética mediante el uso de una pequeña computadora relativamente económica, tal como la IBM 1401, que era muy adecuada para leer las tarjetas, copiar cintas e imprimir los resultados, pero no tan buena para los cálculos numéricos. Para llevar a cabo los cálculos numéricos se utilizaron otras máquinas mucho más costosas, como la IBM 7094.

3. **Circuitos Integrados (1965-1980)**

La IBM 360 fue la primera línea importante de computadoras en utilizar **circuitos integrados** (ICs) (a pequeña escala), con lo cual se pudo ofrecer una mayor ventaja de precio/rendimiento en comparación con las máquinas de segunda generación, las cuales fueron construidas a partir de transistores individuales. Su éxito fue inmediato y la idea de una familia de computadoras compatibles pronto fue adoptada por todos los demás fabricantes importantes. Los descendientes de estas máquinas se siguen utilizando hoy día en centros de cómputo. En la actualidad se utilizan con frecuencia para manejar bases de datos enormes (por ejemplo, para sistemas de reservaciones de aerolíneas) o como servidores para sitios de *World Wide Web* que deben procesar miles de solicitudes por segundo. La mayor fortaleza de la idea

Gráfica 8 IBM 360. Fuente: http://www.hoylen.com/photos/2009-west/06-computer-history/2009-09-25.3956.jpg

de “una sola familia” fue al mismo tiempo su mayor debilidad. La intención era que todo el software, incluyendo al sistema operativo OS/360, funcionara en todos los modelos. Debía ejecutarse en los sistemas pequeños, que por lo general sólo reemplazaban a la 1401s, que copiaba tarjetas a cinta, y en los sistemas muy grandes, que a menudo reemplazaban a la 7094s, que realizaba predicciones del clima y otros cálculos pesados. Tenía que ser bueno en sistemas con pocos dispositivos periféricos y en sistemas con muchos. Tenía que funcionar en ambos entornos comerciales y científicos. Por encima de todo, tenía que ser eficiente para todos estos usos distintos. No había forma en que IBM (o cualquier otra) pudiera escribir una pieza de software que cumpliera con todos estos requerimientos en conflicto. El resultado fue un enorme y extraordinariamente complejo sistema operativo, tal vez de dos a tres órdenes de magnitud más grande que el FMS. Consistía en millones de líneas de lenguaje ensamblador escrito por miles de programadores, con miles de errores, los cuales requerían un flujo continuo de nuevas versiones en un intento por corregirlos. Cada nueva versión corregía algunos errores e introducía otros, por lo que probablemente el número de errores permanecía constante en el tiempo. A pesar de su enorme tamaño y sus problemas, el OS/360 y los sistemas operativos similares de tercera generación producidos por otros fabricantes de computadoras en realidad dejaban razonablemente satisfechos a la mayoría de sus clientes. También popularizaron varias técnicas clave ausentes en los sistemas operativos de segunda generación.



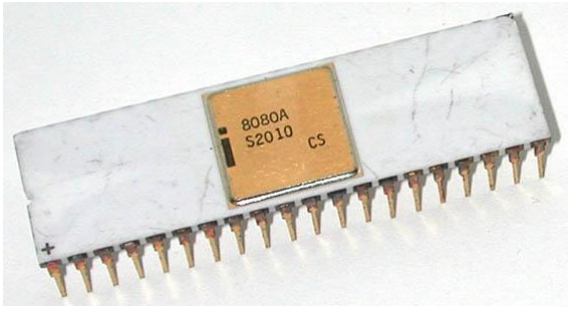
Gráfica 9. Sistema de multiprogramación con 3 trabajos en memoria. Fuente: Tanembaum (2009)

Quizá la más importante de éstas fue la multiprogramación. En la 7094, cuando el trabajo actual se detenía para esperar a que se completara una operación con cinta u otro dispositivo de E/S, la CPU simplemente permanecía inactiva hasta terminar la operación de E/S. Con los cálculos científicos que requieren un uso intensivo de la CPU, la E/S no es frecuente, por lo que este tiempo desperdiciado no es considerable. Con el procesamiento de datos comerciales, el tiempo de espera de las operaciones de E/S puede ser a menudo de 80 a 90 por ciento del tiempo total, por lo que debía hacerse algo para evitar que la (costosa) CPU esté inactiva por mucho tiempo.

La solución que surgió fue particionar la memoria en varias piezas, con un trabajo distinto en cada partición, como se muestra en la Gráfica 9. Mientras que un trabajo esperaba a que se completara una operación de E/S, otro podía estar usando la CPU. Si pudieran contenerse suficientes trabajos en memoria principal al mismo tiempo, la CPU podía estar ocupada casi 100 por ciento del tiempo. Para tener varios trabajos de forma segura en memoria a la vez, se requiere hardware especial para proteger cada trabajo y evitar que los otros se entrometan y lo malogren; el 360 y los demás sistemas de tercera generación estaban equipados con este hardware.

4. **Computadores Personales (1980-actual)**

Con el desarrollo de los circuitos LSI (*Large Scale Integration*, Integración a gran escala), que contienen miles de transistores en un centímetro cuadrado de silicio (chip), nació la era de la computadora personal. En términos de arquitectura, las computadoras personales (que al principio eran conocidas como microcomputadoras) no eran del todo distintas de las minicomputadoras de la clase PDP-11, pero en términos de precio sin duda eran distintas. Mientras que la minicomputadora hizo posible que un departamento en una compañía o universidad tuviera su propia computadora, el chip microprocesador logró que un individuo tuviera su propia computadora personal.



Gráfica 10. Chip Intel 8080. Fuente: arquiuct.blogspot.com

Gráfica 11. IBM PC. Fuente: http://pc-museum.com/gallery/rcm-001.jpg

Cuando Intel presentó el microprocesador 8080 en 1974 (la primera CPU de 8 bits de propósito general), deseaba un sistema operativo, en parte para poder probarlo. Intel pidió a uno de sus consultores, Gary Kildall, que escribiera uno. Kildall y un amigo construyeron primero un dispositivo controlador para el disco flexible de 8 pulgadas de Shugart Associates que recién había sido sacado al mercado, y conectaron el disco flexible con el 8080, con lo cual produjeron la primera microcomputadora con un disco. Después Kildall escribió un sistema operativo

basado en disco conocido como CP/M (Control Program for Microcomputers; Programa de Control para Microcomputadoras) para esta CPU. Como Intel no pensó que las microcomputadoras basadas en disco tuvieran mucho futuro, cuando Kildall pidió los derechos para CP/M, Intel le concedió su petición. Después Kildall formó una compañía llamada Digital Research para desarrollar y vender el CP/M.

En 1977, Digital Research rediseñó el CP/M para adaptarlo de manera que se pudiera ejecutar en todas las microcomputadoras que utilizaban los chips 8080, Zilog Z80 y otros. Se escribieron muchos programas de aplicación para ejecutarse en CP/M, lo cual le permitió dominar por completo el mundo de la microcomputación durante un tiempo aproximado de 5 años.

A principios de la década de 1980, IBM diseñó la IBM PC y buscó software para ejecutarlo en ella. La gente de IBM se puso en contacto con Bill Gates para obtener una licencia de uso de su intérprete de BASIC. También le preguntaron si sabía de un sistema operativo que se ejecutara en la PC. Gates sugirió a IBM que se pusiera en contacto con Digital Research, que en ese entonces era la compañía con dominio mundial en el área de sistemas operativos. Kildall rehusó a reunirse con IBM y envió a uno de sus subordinados, a lo cual se le considera sin duda la peor decisión de negocios de la historia. Para empeorar más aún las cosas, su abogado se rehusó a firmar el contrato de no divulgación de IBM sobre la PC, que no se había anunciado todavía. IBM regresó con Gates para ver si podía proveerles un sistema operativo.

Cuando IBM regresó, Gates se había enterado de que un fabricante local de computadoras, Seattle Computer Products, tenía un sistema operativo adecuado conocido como DOS (*Disk Operating System*; Sistema Operativo en Disco). Se acercó a ellos y les ofreció comprarlo (supuestamente por 75,000 dólares), a lo cual ellos accedieron de buena manera. Después Gates ofreció a IBM un paquete con DOS/BASIC, el cual aceptó. IBM quería ciertas modificaciones, por lo que Gates contrató a la persona que escribió el DOS, Tim Paterson, como empleado de su recién creada empresa de nombre Microsoft, para que las llevara a cabo. El sistema rediseñado cambió su nombre

Gráfica 12. Logo MS-DOS

a MS-DOS (Microsoft Disk Operating System; Sistema Operativo en Disco de Microsoft) y rápidamente llegó a dominar el mercado de la IBM PC. Un factor clave aquí fue la decisión de Gates (que en retrospectiva, fue en extremo inteligente) de vender MS-DOS a las empresas de computadoras para que lo incluyeran con su hardware, en comparación con el intento de Kildall por vender CP/M a los usuarios finales, uno a la vez (por lo menos al principio).

Después de que se supo todo esto, Kildall murió en forma repentina e inesperada debido a causas que aún no han sido reveladas por completo.

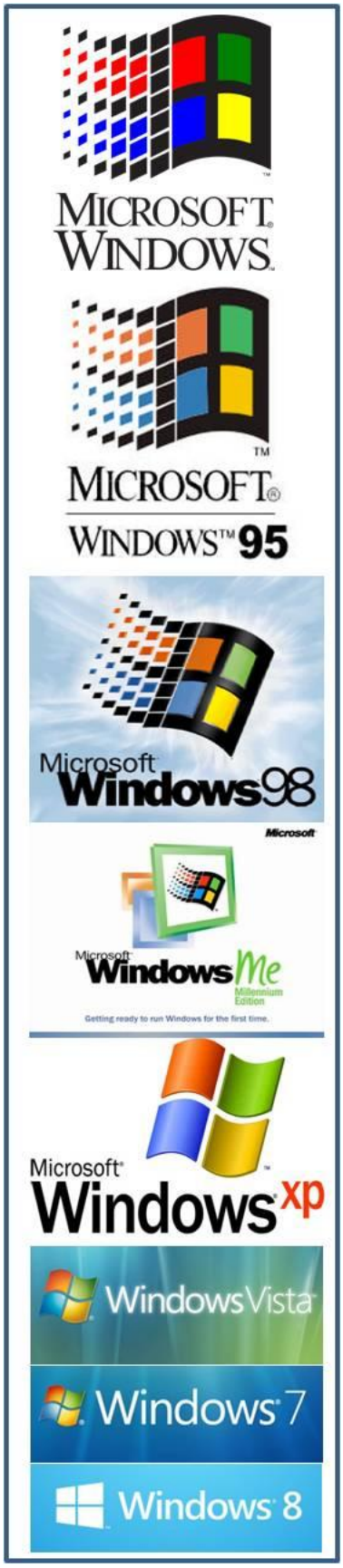
Gráfica 13. Logo UNIX

Para cuando salió al mercado en 1983 la IBM PC/AT, sucesora de la IBM PC, con la CPU Intel 80286, MS-DOS estaba muy afianzado y CP/M daba sus últimos suspiros. Más adelante, MS-DOS se utilizó ampliamente en el 80386 y 80486. Aunque la versión inicial de MS-DOS era bastante primitiva, las versiones siguientes tenían características más avanzadas, incluyendo muchas que se tomaron de UNIX. (Microsoft estaba muy al tanto de UNIX e inclusive vendía una versión de este sistema para microcomputadora, conocida como XENIX, durante los primeros años de la compañía).

CP/M, MS-DOS y otros sistemas operativos para las primeras microcomputadoras se basaban en que los usuarios escribieran los comandos mediante el teclado. Con el tiempo esto cambió debido a la investigación realizada por Doug Engelbart en el Stanford Research Institute en la década de 1960. Engelbart inventó la Interfaz Gráfica de Usuario GUI, completa con ventanas, iconos, menús y ratón. Los investigadores en Xerox PARC adoptaron estas ideas y las incorporaron en las máquinas que construyeron.


Gráfica 14. Apple Macintosh. Fuente: Apple Inc.

Un día, Steve Jobs, que fue co-inventor de la computadora Apple en su cochera, visitó PARC, vio una GUI y de inmediato se dio cuenta de su valor potencial, algo que la administración de Xerox no hizo. Posteriormente, Jobs emprendió el proyecto de construir una Apple con una GUI. Este proyecto culminó en Lisa, que era demasiado costosa y fracasó comercialmente. El segundo intento de Jobs, la Apple Macintosh, fue un enorme éxito, no



sólo debido a que era mucho más económica que Lisa, sino también porque era amigable para el usuario (*user friendly*), lo cual significaba que estaba diseñada para los usuarios que no sólo no sabían nada acerca de las computadoras, sino que además no tenían ninguna intención de aprender. En el mundo creativo del diseño gráfico, la fotografía digital profesional y la producción de video digital profesional, las Macintosh son ampliamente utilizadas y sus usuarios son muy entusiastas sobre ellas.

Cuando Microsoft decidió crear un sucesor para el MS-DOS estaba fuertemente influenciado por el éxito de la Macintosh. Produjo un sistema basado en GUI llamado Windows, el cual en un principio se ejecutaba encima del MS-DOS (es decir, era más como un *shell* que un verdadero sistema operativo). Durante cerca de 10 años, de 1985 a 1995, Windows fue sólo un entorno gráfico encima de MS-DOS. Sin embargo, a partir de 1995 se liberó una versión independiente de Windows, conocida como Windows 95, que incorporaba muchas características de los sistemas operativos y utilizaba el sistema MS-DOS subyacente sólo para iniciar y ejecutar programas de MS-DOS antiguos. En 1998, se liberó una versión ligeramente modificada de este sistema, conocida como Windows 98. Sin embargo, tanto Windows 95 como Windows 98 aún contenían una gran cantidad de lenguaje ensamblador para los procesadores Intel de 16 bits.

Otro de los sistemas operativos de Microsoft es Windows NT (NT significa Nueva Tecnología), que es compatible con Windows 95 en cierto nivel, pero fue completamente rediseñado en su interior. Es un sistema completo de 32

bits. El diseñador en jefe de Windows NT fue David Cutler, quien también fue uno de los diseñadores del sistema operativo VMS de VAX, por lo que hay algunas ideas de VMS presentes en NT. De hecho, había tantas ideas de VMS presentes que el propietario de VMS (DEC) demandó a Microsoft. El caso se resolvió en la corte por una cantidad de muchos dígitos. Microsoft esperaba que la primera versión de NT acabara con MS-DOS y todas las demás versiones de Windows, ya que era un sistema muy superior, pero fracasó. No fue sino hasta Windows NT 4.0 que finalmente empezó a tener éxito, en especial en las redes corporativas. La versión 5 de Windows NT cambió su nombre a Windows 2000 a principios de 1999. Estaba destinada a ser el sucesor de Windows 98 y de Windows NT 4.0. Esto tampoco funcionó como se esperaba, por lo que Microsoft preparó otra versión de Windows 98 conocida como Windows Me (Millennium edition). En el 2001 se liberó una versión ligeramente actualizada de Windows 2000, conocida como Windows XP. Esa versión duró mucho más en el mercado (6 años), reemplazando a casi todas las versiones anteriores de Windows. Después, en enero del 2007 Microsoft liberó el sucesor para Windows XP, conocido como Windows Vista. Tenía una interfaz gráfica nueva, Aero, y muchos programas de usuarios nuevos o actualizados. Microsoft esperaba que sustituya a Windows XP por completo, pero fue un rotundo fracaso.

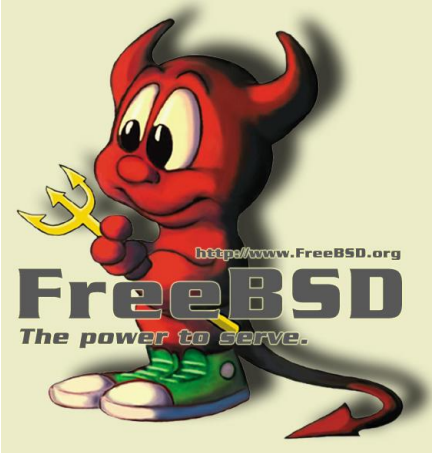
En el 2009, Microsoft lanzó Windows 7. A diferencia de su predecesor, Windows Vista, que introdujo a un gran número de nuevas características, Windows 7 pretendía ser una actualización incremental, enfocada a la línea de Windows, con el objetivo de ser compatible con aplicaciones y hardware que Windows Vista no era compatible. Windows 7 tiene soporte *multitouch*, un Windows shell rediseñado con una nueva barra de tareas, conocido como Superbar, un sistema red llamado HomeGroup, y mejoras en el rendimiento sobre todo en velocidad y en menor consumo de recursos.

Gráfica 15. Tux. Imagen de Linux.

En el 2012, Microsoft lanzó Windows 8. Por primera vez desde Windows 95, el botón Inicio ya no está disponible en la barra de tareas (lo cual se corrigió en la versión 8.1), aunque la pantalla de inicio está aún activa haciendo clic en la esquina inferior izquierda de la pantalla y presionando la tecla Inicio en el teclado. Presenta un Explorador de

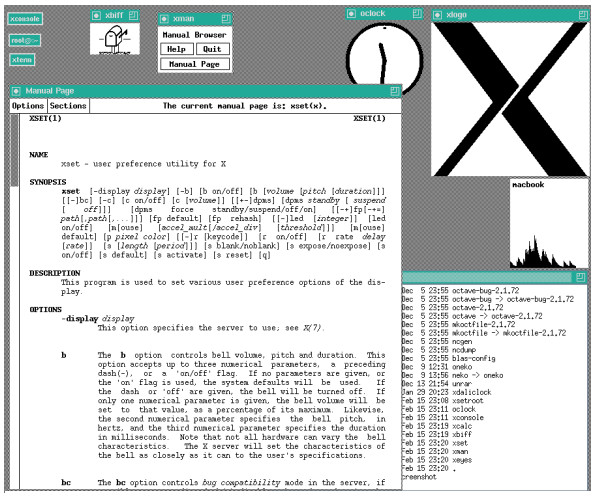
Windows rediseñado, con la famosa interfaz *ribbon* de Microsoft Office. Se conservan la gran mayoría de las características de su predecesor, Windows 7, con excepción de la nueva interfaz gráfica y algunos cambios menores.

El otro competidor importante en el mundo de las computadoras personales es UNIX (y todas sus variantes). UNIX es más fuerte en los servidores tanto de redes como empresariales, pero también está cada vez más presente en las computadoras de escritorio, en especial en los países que se desarrollan con rapidez, como India y China. En las computadoras basadas en Pentium, Linux se está convirtiendo en una alternativa popular para Windows entre los estudiantes y cada vez más usuarios corporativos. (A lo largo de este documento se usará el término "Pentium" para denotar al Pentium I, II, III y 4, así como sus sucesores tales como el Core 2 Duo.)



Gráfica 16. Logo FreeBSD

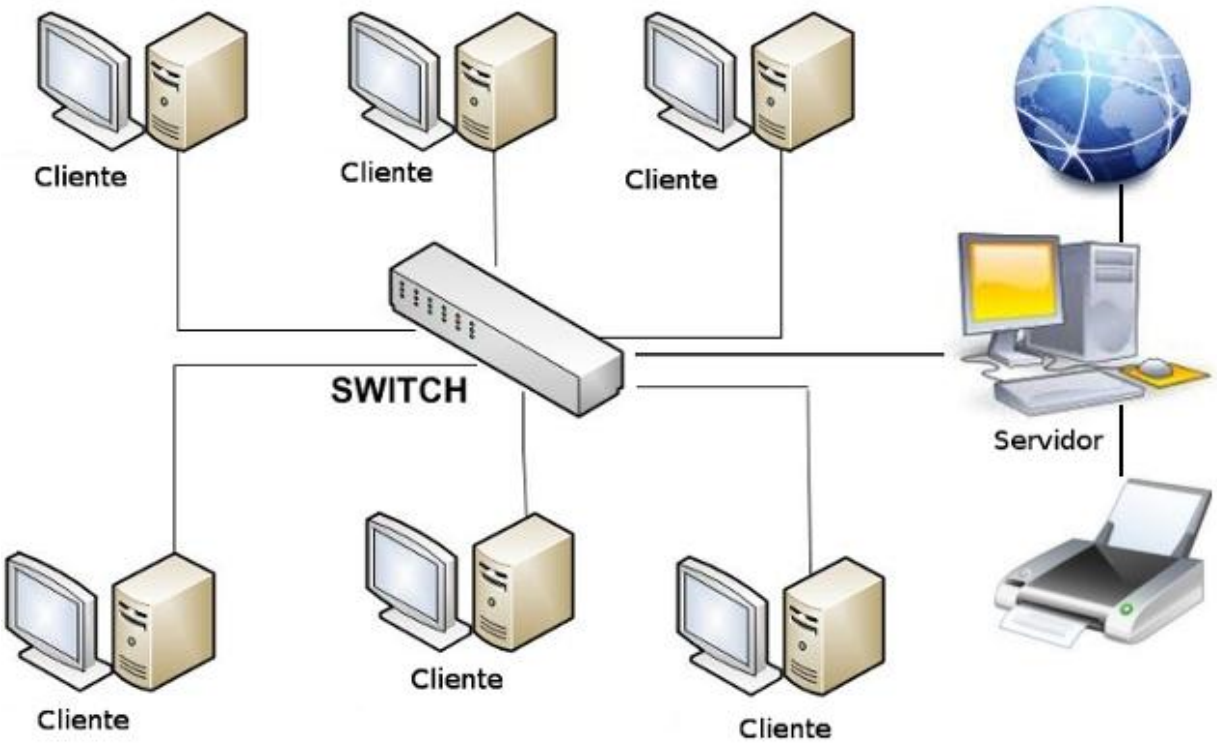
El término x86 también se utiliza algunas veces para indicar el rango completo de CPU Intel partiendo desde el 8086, mientras que utilizaremos "Pentium" para indicar todas las CPU desde el Pentium I. Admitimos que este término no es perfecto, pero no hay disponible uno mejor. Uno se pregunta qué genio de mercadotecnia en Intel desperdició una marca comercial (Pentium) que la mitad del mundo conocía bien y respetaba, sustituyéndola con términos como "Core 2 duo" que muy pocas personas comprenden; ¿qué significan "2" y "duo"? Tal vez "Pentium 5" (o "Pentium 5 dual core", etc.) eran demasiado difíciles de recordar. FreeBSD es también un derivado popular de UNIX, que se originó del proyecto BSD en Berkeley. Todas las computadoras modernas Macintosh utilizan una versión modificada de FreeBSD. UNIX también es estándar en las



Gráfica 17. Captura de pantalla de X Windows. Fuente: http://upload.wikimedia.org/wikipedia/commons/d/d4/X-Window-System.png

estaciones de trabajo operadas por chips RISC de alto rendimiento, como los que venden Hewlett-Packard y Sun Microsystems.

Muchos usuarios de UNIX, en especial los programadores experimentados, prefieren una interfaz de línea de comandos a una GUI, por lo que casi todos los sistemas UNIX presentan un sistema de ventanas llamado *X Window System* (también conocido como X11), producido en el M.I.T. Este sistema se encarga de la administración básica de las ventanas y permite a los usuarios crear, eliminar, desplazar y cambiar el tamaño de las ventanas mediante el uso de un ratón. Con frecuencia hay disponible una GUI completa, como Gnome o KDE, para ejecutarse encima de X11, lo cual proporciona a UNIX una apariencia parecida a la Macintosh o a Microsoft Windows, para aquellos usuarios de UNIX que desean algo así.



Gráfica 18. Sistema Operativo para una red. Fuente: http://i.imgur.com/ISjVP.jpg

Un interesante desarrollo que empezó a surgir a mediados de la década de 1980 es el crecimiento de las redes de computadoras personales que ejecutan sistemas operativos

en red y sistemas operativos distribuidos (Tanembaum y Van Steen, 2007). En un sistema operativo en red, los usuarios están conscientes de la existencia de varias computadoras, y pueden iniciar sesión en equipos remotos y copiar archivos de un equipo a otro. Cada equipo ejecuta su propio sistema operativo local y tiene su propio usuario (o usuarios) local.

Los sistemas operativos en red no son fundamentalmente distintos de los sistemas operativos con un solo procesador. Es obvio que necesitan un dispositivo controlador de interfaz de red y cierto software de bajo nivel para controlarlo, así como programas para lograr el inicio de una sesión remota y el acceso remoto a los archivos, pero estas adiciones no cambian la estructura esencial del sistema operativo.

En contraste, un sistema operativo distribuido se presenta a sus usuarios en forma de un sistema tradicional con un procesador, aun cuando en realidad está compuesto de varios procesadores. Los usuarios no tienen que saber en dónde se están ejecutando sus programas o en dónde se encuentran sus archivos; el sistema operativo se encarga de todo esto de manera automática y eficiente.

Los verdaderos sistemas operativos distribuidos requieren algo más que sólo agregar un poco de código a un sistema operativo con un solo procesador, ya que los sistemas distribuidos y los centralizados difieren en varios puntos críticos. Por ejemplo, los sistemas distribuidos permiten con frecuencia que las aplicaciones se ejecuten en varios procesadores al mismo tiempo, lo que requiere algoritmos de planificación del procesador más complejos para poder optimizar la cantidad de paralelismo. Los retrasos de comunicación dentro de la red implican a menudo que estos (y otros) algoritmos deban ejecutarse con información incompleta, obsoleta o incluso incorrecta. Esta situación es muy distinta a la de un sistema con un solo procesador, donde el sistema operativo tiene información completa acerca del estado del sistema.

1.3. Funciones de un Sistema Operativo

De acuerdo a Ramírez (s/f), las funciones principales de un Sistema Operativo son:

1. **Servir de intermediario en la comunicación entre los usuarios y el hardware de la computadora**: para realizar esta función, el sistema operativo debe proporcionar a los usuarios un ambiente de trabajo cómodo, accesible, eficiente y seguro. El sistema operativo el que se encarga de manejar el hardware de la computadora, lo que hace que los usuarios no requieran de conocimientos de electrónica para hacer uso de la misma (abstracción del hardware a los usuarios). Al utilizarse un sistema operativo es como si se colocara una capa de software sobre el hardware, con el objeto de que éste maneje todas las partes del sistema y presentar al usuario una interfaz o máquina virtual que es más fácil de entender y programar.

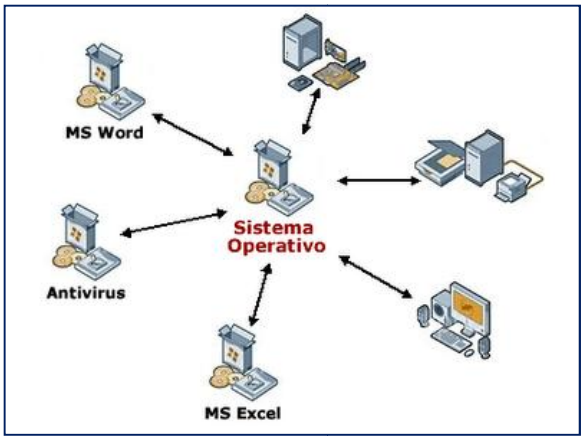
Gráfica 19. Interacción SO con el resto de las partes. Fuente: http://upload.wikimedia.org/wikipedia/commons/d/dc/Operating_system_placement-es.svg

2. **Administrar los recursos del sistema**: el sistema operativo proporciona un sistema lógico de comunicación y control (ordenado, seguro, consistente y eficiente) entre los distintos componentes que integran la computadora: el CPU, la memoria principal, las unidades de almacenamiento secundario y los dispositivos de entrada/salida. Además, se encarga de ofrecer una distribución ordenada y controlada de los recursos de que dispone el sistema entre los distintos programas que los requieren. Administrando los recursos de la computadora, el sistema operativo tiene control sobre el funcionamiento básico de la misma. Con el fin de poder cumplir con las funciones antes descritas, es necesario que el sistema operativo realice las actividades de administración de programas, administración de tareas, administración de dispositivos, administración de usuarios, administración de seguridad, etc.

a. **Administración de dispositivos:** Como se mencionó anteriormente, el sistema operativo debe ser capaz de controlar todos los componentes del sistema. Por ejemplo, debe manejar las entradas y las salidas de los datos a través de las unidades de entrada/salida, manteniendo los detalles del control de los dispositivos dentro del sistema operativo, pero al reemplazar o agregar un nuevo dispositivo, sólo debe cambiarse en el sistema operativo la rutina de control que se encarga de manejar este dispositivo.

b. **Administración de los sistemas de almacenamiento**: Debe proporcionar un sistema para el manejo de los archivos y las funciones necesarias para conocer como éstos quedan guardados en las unidades de almacenamiento secundario. Este sistema de manejo de archivos realizará todas las tareas que permitan el almacenamiento y recuperación de datos que sean requeridas por los usuarios. Los programas de aplicación no saben dónde se encuentran almacenados los datos o cómo recuperarlos, ya que estos conocimientos están contenidos en las rutinas de métodos de acceso del sistema o en los controladores de dispositivos. Cuando un programa requiere leer datos, le envía una orden al sistema operativo mediante un código de instrucción, éste busca el dato y lo entrega al programa. A la inversa, cuando el programa requiere guardar datos, los mismos son enviados al sistema operativo, quien es el que se encarga de ubicar espacio libre en el medio de almacenamiento y procesar su almacenamiento.

c. **Administración de trabajos**: el sistema operativo interpreta y responde a los comandos que ingresa el usuario, cargando en memoria principal, si es necesario, el programa correspondiente para su ejecución. En algunos casos, este proceso puede requerir la carga adicional de otros programas. Los sistemas operativos no son todos iguales,



algunos tienen características sobresalientes, tales como la habilidad de ejecutar más de una tarea a la vez (multitarea), soportar más de un usuario trabajando al mismo tiempo (multiusuario), proporcionar un sistema de seguridad que proteja el acceso a los equipos y los datos, etc. En el caso de los sistemas multiusuario, el sistema operativo debe decidir si acepta o no ejecutar el programa o trabajo requerido por un usuario, para lo cual debe verificar si el usuario está registrado y si el mismo tiene autorización para utilizar este programa.

d. **Administración de tareas**: En los sistemas monotarea, la administración de tareas es mínima, ya que para poder ejecutarse una nueva tarea tiene que haber finalizado la tarea previa. Pero en los sistemas multitarea, el sistema operativo es el responsable de la operación simultánea de uno o más programas (tareas), distribuyendo los recursos (CPU, memoria principal, etc.) entre las distintas tareas y coordinando su funcionamiento. Los sistemas operativos avanzados poseen la habilidad de asignar prioridades a las tareas de modo tal que se pueda cambiar el orden de ejecución de las mismas. El número de programas que pueden ser efectivamente ejecutados depende de la cantidad de memoria principal disponible, tipo y velocidad del CPU, así como también de la eficiencia y capacidades del mismo sistema operativo. La multitarea se realiza aprovechando las diferencias de velocidades de trabajo del CPU y de entrada/salida, mientras un programa está esperando una entrada, se pueden ejecutar instrucciones de otro programa. Cuando una computadora ejecuta simultáneamente varias tareas, surge la necesidad de administrar la asignación de los diferentes recursos requeridos por las mismas. El sistema operativo se encarga de asignar dinámicamente a cada tarea en ejecución los recursos que ésta requiere para su uso exclusivo durante el tiempo que sea necesario, siempre que estén disponibles y puedan ser utilizadas por el usuario a quien pertenece la tarea. Además se encarga de que no se presenten conflictos en la ejecución de las diferentes tareas.

e. **Administración de seguridad**: El sistema operativo debe proteger a la computadora del acceso o utilización por usuarios no autorizados, para lo cual debe proporcionar un sistema de creación y control de cuentas de usuarios, así como los mecanismos para el procesamiento de la identificación de los mismos cuando acceden al equipo. El sistema operativo debe mantener registro de la actividad del sistema y llevar la contabilidad de la utilización de los recursos por parte de los usuarios. También deben proveer los procedimientos para el respaldo de archivos y la recuperación del sistema en caso de presentarse fallos en el mismo.

1.4. Características de un Sistema Operativo

Los sistemas operativos presentan las siguientes características[1]:

1. Conveniencia: un sistema operativo hace más conveniente el uso de una computadora.
2. Eficiencia: el sistema operativo permite que los recursos de la computadora se usen de manera correcta y eficiente.
3. Habilidad para evolucionar: un sistema operativo debe de ser capaz de aceptar nuevas funciones sin que tenga problemas.
4. Encargado de administrar el hardware: el sistema operativo debe de ser eficaz.
5. Relacionar dispositivos
6. Algoritmos: un sistema operativo hace el uso de la computadora más racional

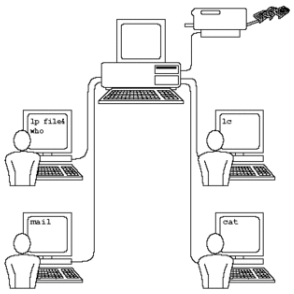
1.5. Tipos de Sistemas Operativos

Se presentan los siguientes tipos de Sistemas Operativos:

1. Según la administración de tareas[2]
    a. Monotarea: este tipo de sistemas operativos son capaces de manejar un programa o realizar una sola tarea a la vez. Son los más antiguos. Por ejemplo, si el usuario está escaneando, la computadora no responderá a nuevas indicaciones ni comenzará un proceso nuevo.

   b. Multitarea: esta característica es propia de los Sistemas Operativos más avanzados y permiten ejecutar varios procesos a la vez, desde uno o varios ordenadores, es decir que los pueden utilizar varios usuarios al mismo tiempo. Esto se puede realizar por medio de sesiones remotas una red o bien, a través de terminales conectadas a una computadora.

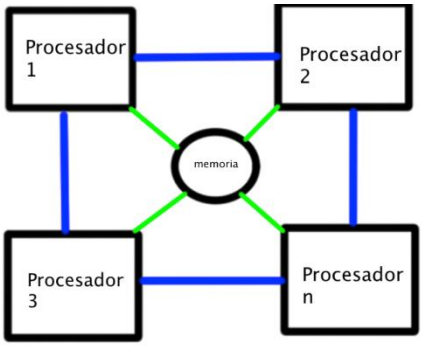
2. Según la administración de usuarios
   a. Monousuario: Sólo pueden responder a un usuario por vez. De esta manera, cualquier usuario tiene acceso a los datos del sistema. Existe un único usuario que puede realizar cualquier tipo de operación.
   b. Multiusuario: esta característica es propia de aquellos Sistemas Operativos en los que varios usuarios pueden acceder a sus servicios y procesamientos al mismo tiempo. De esta manera, satisfacen las necesidades de varios usuarios que estén utilizando los mismos recursos, ya sea memoria, programas, procesador, impresoras, scanners, entre otros.



Gráfica 20. Sistema Operativo Multiusuario. Fuente: http://wiki.inf.utfsm.cl/images/thumb/7/7f/Multiuse.gif/350px-Multiuse.gif

3. Según la administración de recursos[3]:
   a. Centralizado: permite usar los recursos de una sola computadora.
   b. Distribuido: permite utilizar los recursos (memoria, CPU, disco, periféricos, etc.) de más de una computadora al mismo tiempo

4. Según el número de procesadores[4]:
   a. Monoprocesador
      Trabajan con un solo procesador.
   b. Multiprocesador
      Pueden utilizar varios procesadores para distribuir el trabajo de cada uno. Pueden ser de dos tipos: Asimétrico (el sistema operativo selecciona un procesador maestro y los demás funcionan como esclavos) o Simétrico (se envía información o se trabaja con el procesador con menos carga y



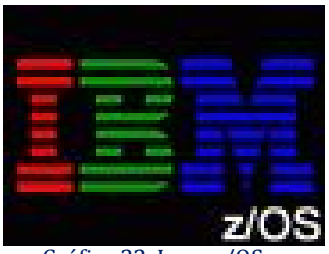
Gráfica 21. Multiprocesos.

así se distribuye mejor el trabajo, los procesos son enviados indistintamente a cual quiera de los procesadores disponibles).

5. Según el hardware (Tanembaum, 2009)

   En la sección 6 se describirán someramente algunos de los sistemas operativos más usados y más emblemáticos.

   a. De Mainframe



Gráfica 22. Logo z/OS.

En el extremo superior están los sistemas operativos para las mainframes, las computadoras del tamaño de un cuarto completo que aún se encuentran en los principales centros de datos corporativos. La diferencia entre estas computadoras y las personales está en su capacidad de E/S. Una mainframe con 1000 discos y millones de gigabytes de datos no es poco común; una computadora personal con estas especificaciones sería la envidia de los amigos del propietario. Las mainframes también están volviendo a figurar en el ámbito computacional como servidores Web de alto rendimiento, servidores para sitios de comercio electrónico a gran escala y servidores para transacciones de negocio a negocio. Los sistemas operativos para las mainframes están profundamente orientados hacia el procesamiento de muchos trabajos a la vez, de los cuales la mayor parte requiere muchas operaciones de E/S. Por lo general ofrecen tres tipos de servicios: procesamiento por lotes, procesamiento de transacciones y tiempo compartido. Un sistema de procesamiento por lotes procesa los trabajos de rutina sin que haya un usuario interactivo presente. El procesamiento de reclamaciones en una compañía de seguros o el reporte de ventas para una cadena de tiendas son actividades que se realizan comúnmente en modo de procesamiento por lotes. Los sistemas de procesamiento de transacciones manejan grandes cantidades de pequeñas peticiones, por ejemplo: el procesamiento de cheques en un banco o las reservaciones en una aerolínea. Cada unidad de trabajo es pequeña, pero el sistema debe manejar cientos o miles por segundo. Los sistemas de tiempo compartido permiten que varios usuarios remotos ejecuten trabajos en la computadora al mismo tiempo, como consultar

una gran base de datos. Estas funciones están íntimamente relacionadas; a menudo los sistemas operativos de las mainframes las realizan todas. Un ejemplo de sistema operativo de mainframe es el z/OS de IBM.

b. De Servidores

Gráfica 23. Logo Solaris

En el siguiente nivel hacia abajo se encuentran los sistemas operativos de servidores. Se ejecutan en servidores, que son computadoras personales muy grandes, estaciones de trabajo o incluso mainframes. Dan servicio a varios usuarios a la vez a través de una red y les permiten compartir los recursos de hardware y de software. Los servidores pueden proporcionar servicio de impresión, de archivos o Web. Los proveedores de Internet operan muchos equipos servidores para dar soporte a sus clientes y los sitios Web utilizan servidores para almacenar las páginas Web y hacerse cargo de las peticiones entrantes. Algunos sistemas operativos de servidores comunes son Solaris, FreeBSD, Linux y Windows Server 200x.

c. De Multiprocesadores

Una manera cada vez más común de obtener poder de cómputo de las grandes ligas es conectar varias CPU en un solo sistema. Dependiendo de la exactitud con la que se conecten y de lo que se comparta, estos sistemas se conocen como computadoras en paralelo, multicomputadoras o multiprocesadores. Necesitan sistemas operativos especiales, pero a menudo son variaciones de los sistemas operativos de servidores con características especiales para la comunicación, conectividad y consistencia.


Gráfica 24. Logo MS Windows

Con la reciente llegada de los chips multinúcleo para las computadoras personales, hasta los sistemas operativos de equipos de escritorio y portátiles convencionales están empezando a lidiar con multiprocesadores de al menos pequeña escala y es probable que el número de núcleos aumente con el tiempo. Por fortuna, se conoce mucho acerca de los sistemas operativos de multiprocesadores gracias a

los años de investigación previa, por lo que el uso de este conocimiento en los sistemas multinúcleo no debe presentar dificultades. La parte difícil será hacer que las aplicaciones hagan uso de todo este poder de cómputo. Muchos sistemas operativos populares (incluyendo Windows y Linux) se ejecutan en multiprocesadores.

d. De Computadores Personales

La siguiente categoría es el sistema operativo de computadora personal. Todos los sistemas operativos modernos soportan la multiprogramación, con frecuencia se inician docenas de programas al momento de arrancar el sistema. Su trabajo es proporcionar buen soporte para un solo usuario. Se utilizan ampliamente para el procesamiento de texto, las hojas de cálculo y el acceso a Internet. Algunos ejemplos comunes son Linux, FreeBSD, Windows y el sistema operativo Macintosh.

Gráfica 25. Logo MAC OS

Los sistemas operativos de computadora personal son tan conocidos que tal vez no sea necesario presentarlos con mucho detalle. De hecho, muchas personas ni siquiera están conscientes de que existen otros tipos de sistemas operativos.

e. De Computadores de Bolsillo

Continuando con los sistemas cada vez más pequeños, llegamos a las computadoras de bolsillo (*handheld*). Una computadora de bolsillo o PDA (*Personal Digital Assistant*, Asistente personal digital) es una computadora que cabe en los bolsillos y realiza una pequeña variedad de funciones, como libreta de direcciones electrónica y bloc de notas. Además, hay muchos teléfonos celulares muy similares a los PDAs, con la excepción de su teclado y pantalla. En efecto, los PDAs y los teléfonos celulares se han fusionado en esencia y sus principales diferencias se observan en el tamaño, el peso y la interfaz de usuario. Casi todos ellos se basan en CPUs de 32 bits con el modo protegido y ejecutan un sofisticado sistema operativo. Los


Gráfica 26. Logo Android

sistemas operativos que operan en estos dispositivos de bolsillo son cada vez más sofisticados, con la habilidad de proporcionar telefonía, fotografía digital y otras funciones. Muchos de ellos también ejecutan aplicaciones desarrolladas por terceros. De hecho, algunos están comenzando a asemejarse a los sistemas operativos de computadoras personales de hace una década. Una de las principales diferencias entre los dispositivos de bolsillo y las PCs es que los primeros no tienen discos duros de varios cientos de gigabytes, lo cual cambia rápidamente. Dos de los sistemas operativos más populares para los dispositivos de bolsillo son Android e iOS.

f. Integrados

Los sistemas integrados (*embedded*), que también se conocen como incrustados o embebidos, operan en las computadoras que controlan dispositivos que no se consideran generalmente como computadoras, ya que no aceptan software instalado por el usuario. Algunos ejemplos comunes son los hornos de microondas, las televisiones, los autos, los grabadores de DVDs, los teléfonos celulares y los reproductores de MP3. La propiedad principal que diferencia a los sistemas integrados de los dispositivos de bolsillo es la certeza de que nunca se podrá ejecutar software que no sea confiable. No se pueden descargar nuevas aplicaciones en el horno de microondas; todo el software se encuentra en ROM. Esto significa que no hay necesidad de protección en las aplicaciones, lo cual conlleva a cierta simplificación. Los sistemas como QNX y VxWorks son populares en este dominio.

Gráfica 27. Logo QNX.

g. De nodos sensores

Las redes de pequeños nodos sensores se están implementando para varios fines. Estos nodos son pequeñas computadoras que se comunican entre sí con una estación base, mediante el uso de comunicación inalámbrica. Estas redes de sensores se utilizan para proteger los perímetros de los edificios, resguardar las fronteras nacionales, detectar incendios en bosques, medir la temperatura y la

Sisten

Gráfica 28. Logo TinyOS

precipitación para el pronóstico del tiempo, deducir información acerca del movimiento de los enemigos en los campos de batalla y mucho más. Los sensores son pequeñas computadoras con radios integrados y alimentadas con baterías. Tienen energía limitada y deben trabajar durante largos periodos al exterior y desatendidas, con frecuencia en condiciones ambientales rudas. La red debe ser lo bastante robusta como para tolerar fallas en los nodos individuales, que ocurren con mayor frecuencia a medida que las baterías empiezan a agotarse. Cada nodo sensor es una verdadera computadora, con una CPU, RAM, ROM y uno o más sensores ambientales. Ejecuta un sistema operativo pequeño pero real, por lo general manejador de eventos, que responde a los eventos externos o realiza mediciones en forma periódica con base en un reloj interno. El sistema operativo tiene que ser pequeño y simple debido a que los nodos tienen poca RAM y el tiempo de vida de las baterías es una cuestión importante. Además, al igual que con los sistemas integrados, todos los programas se cargan por adelantado; los usuarios no inician repentinamente programas que descargaron de Internet, lo cual simplifica el diseño en forma considerable. TinyOS es un sistema operativo bien conocido para un nodo sensor.

h. En tiempo real

Gráfica 29. Logo e-Cos

Otro tipo de sistema operativo es el sistema en tiempo real. Estos sistemas se caracterizan por tener el tiempo como un parámetro clave. Por ejemplo, en los sistemas de control de procesos industriales, las computadoras en tiempo real tienen que recolectar datos acerca del proceso de producción y utilizarlos para controlar las máquinas en la fábrica. A menudo hay tiempos de entrega estrictos que se deben cumplir. Por ejemplo, si un auto se desplaza sobre una línea de ensamblaje, deben llevarse a cabo ciertas acciones en determinados instantes. Si un robot soldador realiza su trabajo de soldadura antes o después de tiempo, el auto se arruinará. Si la acción debe ocurrir sin excepción en cierto momento (o dentro de cierto rango), tenemos un sistema en tiempo real duro.

Muchos de estos sistemas se encuentran en el control de procesos industriales, en aeronáutica, en la milicia y en áreas de aplicación similares. Estos sistemas deben proveer garantías absolutas de que cierta acción ocurrirá en un instante determinado.

Otro tipo de sistema en tiempo real es el sistema en tiempo real suave, en el cual es aceptable que muy ocasionalmente se pueda fallar a un tiempo predeterminado. Los sistemas de audio digital o de multimedia están en esta categoría. Los teléfonos digitales también son ejemplos de sistema en tiempo real suave. Como en los sistemas en tiempo real es crucial cumplir con tiempos predeterminados para realizar una acción, algunas veces el sistema operativo es simplemente una biblioteca enlazada con los programas de aplicación, en donde todo está acoplado en forma estrecha y no hay protección entre cada una de las partes del sistema. Un ejemplo de este tipo de sistema en tiempo real es e-Cos. Las categorías de sistemas para computadoras de bolsillo, sistemas integrados y sistemas en tiempo real se traslapan en forma considerable. Casi todos ellos tienen por lo menos ciertos aspectos de tiempo real suave. Los sistemas integrados y de tiempo real sólo ejecutan software que colocan los diseñadores del sistema; los usuarios no pueden agregar su propio software, lo cual facilita la protección. Los sistemas de computadoras de bolsillo y los sistemas integrados están diseñados para los consumidores, mientras que los sistemas en tiempo real son más adecuados para el uso industrial. Sin embargo, tienen ciertas características en común.

i. De tarjetas inteligentes

Los sistemas operativos más pequeños operan en las tarjetas inteligentes, que son dispositivos del tamaño de una tarjeta de crédito que contienen un chip de CPU. Tienen varias severas restricciones de poder de procesamiento y memoria. Algunas se energizan mediante contactos en el lector en el que se insertan, pero las tarjetas inteligentes sin contactos se energizan mediante inducción, lo cual limita en forma considerable las cosas que

Gráfica 30. Logo de Java.

pueden hacer. Algunos sistemas de este tipo pueden realizar una sola función, como pagos electrónicos; otros pueden llevar a cabo varias funciones en la misma tarjeta inteligente. A menudo éstos son sistemas propietarios. Algunas tarjetas inteligentes funcionan con Java. Lo que esto significa es que la ROM en la tarjeta inteligente contiene un intérprete para la Máquina virtual de Java (JVM). Los *applets* de Java (pequeños programas) se descargan en la tarjeta y son interpretados por el intérprete de la JVM. Algunas de estas tarjetas pueden manejar varias *applets* de Java al mismo tiempo, lo cual conlleva a la multiprogramación y a la necesidad de planificarlos. La administración de los recursos y su protección también se convierten en un problema cuando hay dos o más *applets* presentes al mismo tiempo. El sistema operativo (que por lo general es en extremo primitivo) presente en la tarjeta es el encargado de manejar estas cuestiones.

1.6. Estructura de los Sistemas Operativos

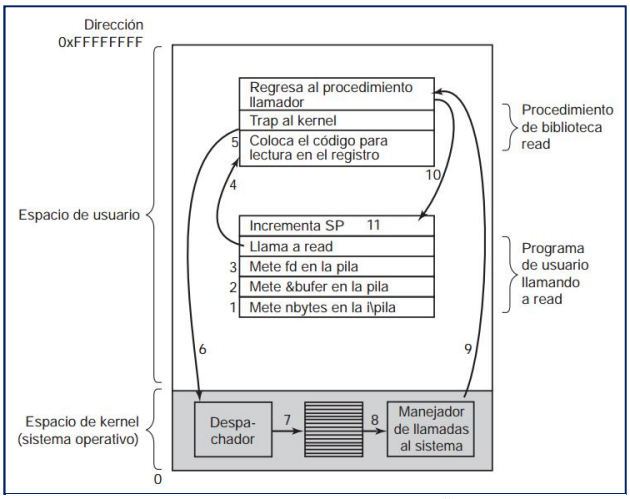
Los seis diseños de estructura de los Sistemas Operativos, de acuerdo a Tanembaum (2009) son: sistemas monolíticos, sistemas de capas, microkernels, sistemas cliente-servidor, máquinas virtuales y exokernels. A continuación se explicará cada uno de ellos.

1. **Sistemas Monolíticos**

   En este diseño, que hasta ahora se considera como la organización más común, todo el sistema operativo se ejecuta como un solo programa en modo kernel. El sistema operativo se escribe como una colección de procedimientos, enlazados entre sí en un solo programa binario ejecutable extenso. Cuando se utiliza esta técnica, cada procedimiento en el sistema tiene la libertad de llamar a cualquier otro, si éste proporciona cierto cómputo útil que el primero necesita. Al tener miles de procedimientos que se pueden llamar entre sí sin restricción, con frecuencia se produce un sistema poco manejable y difícil de comprender.

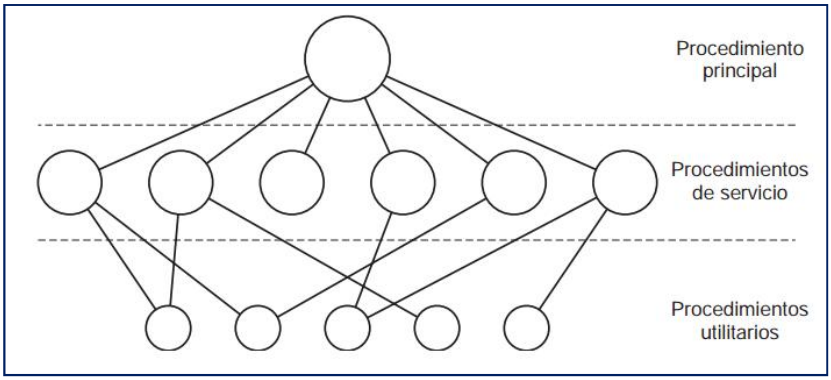
   Para construir el programa objeto actual del sistema operativo cuando se utiliza este diseño, primero se compilan todos los procedimientos individuales (o los archivos que contienen los procedimientos) y luego se vinculan en conjunto para formar un solo archivo ejecutable, usando el enlazador del sistema. En términos de ocultamiento de

información, en esencia no hay nada: todos los procedimientos son visibles para cualquier otro procedimiento (en contraste a una estructura que contenga módulos o paquetes, en donde la mayor parte de la información se oculta dentro de módulos y sólo los puntos de entrada designados de manera oficial se pueden llamar desde el exterior del módulo). Sin embargo, hasta en los sistemas monolíticos es posible tener cierta estructura. Para solicitar los servicios (llamadas al sistema) que proporciona el sistema operativo, los parámetros se colocan en un lugar bien definido (por ejemplo, en la pila) y luego se ejecuta una instrucción de trap. Esta instrucción cambia la máquina del modo usuario al modo kernel y transfiere el control al sistema operativo, lo cual se muestra como el paso 6 en la gráfica 31. Después el sistema operativo obtiene los parámetros y determina cuál es la llamada al sistema que se va a llevar a cabo. Después la indiza en una tabla que contiene en la ranura k un apuntador al procedimiento que lleva a cabo la llamada al sistema k (paso 7 en la gráfica 31). Esta organización sugiere una estructura básica para el sistema operativo:



Gráfica 31. Pasos para hacer la llamada "read". Fuente: Tanembaum (2009)

- Un programa principal que invoca el procedimiento de servicio solicitado.
- Un conjunto de procedimientos de servicio que llevan a cabo las llamadas al sistema.
- Un conjunto de procedimientos utilitarios que ayudan a los procedimientos de servicio.



Gráfica 32. Modelo de estructuración simple para un sistema monolítico. Fuente: Tanembaum (2009)

En este modelo, para cada llamada al sistema hay un procedimiento de servicio que se encarga de la llamada y la ejecuta. Los procedimientos utilitarios hacen cosas que necesitan varios procedimientos de servicio, como obtener datos de los programas de usuario. Esta división de los procedimientos en tres niveles se muestra en la gráfica 32.

Además del núcleo del sistema operativo que se carga al arrancar la computadora, muchos sistemas operativos soportan extensiones que se pueden cargar, como los drivers de dispositivos de E/S y sistemas de archivos. Estos componentes se cargan por demanda.

2. Sistemas de Capas

Una generalización del diseño de la gráfica 32 es organizar el sistema operativo como una jerarquía de capas, cada una construida encima de la que tiene abajo. El primer sistema construido de esta forma fue el sistema THE, construido en Technische Hogeschool Eindhoven en Holanda por E. W. Dijkstra (1968) y sus estudiantes. El sistema THE era un sistema simple de procesamiento por lotes para una computadora holandesa, la Electrologica X8, que tenía 32K de palabras de 27 bits (los bits eran costosos en aquel entonces). El sistema tenía seis capas, como se muestra en la gráfica 33.

Capa	Función
5	El operador
4	Programas de usuario
3	Administración de la entrada/salida
2	Comunicación operador-proceso
1	Administración de memoria y tambor
0	Asignación del procesador y multiprogramación

Gráfica 33. Estructura del Sistema Operativo THE. Fuente: Tanembaum (2009)

El nivel 0 se encargaba de la asignación del procesador, de cambiar entre un proceso y otro cuando ocurrían interrupciones o expiraban los temporizadores. Por encima del nivel 0, el sistema consistía en procesos secuenciales, cada uno de los cuales e podía

programar sin necesidad de preocuparse por el hecho de que había varios procesos en ejecución en un solo procesador. En otras palabras, el nivel 0 proporcionaba la multiprogramación básica de la CPU.

La capa 1 se encargaba de la administración de la memoria. Asignaba espacio para los procesos en la memoria principal y en un tambor de palabras de 512 K que se utilizaba para contener partes de procesos (páginas), para los que no había espacio en la memoria principal. Por encima de la capa 1, los procesos no tenían que preocuparse acerca de si estaban en memoria o en el tambor; el software de la capa 1 se encargaba de asegurar que las páginas se llevaran a memoria cuando se requerían.

La capa 2 se encargaba de la comunicación entre cada proceso y la consola del operador (es decir, el usuario). Encima de esta capa, cada proceso tenía en efecto su propia consola de operador.

La capa 3 se encargaba de administrar los dispositivos de E/S y de guardar en búferes los flujos de información dirigidos para y desde ellos. Encima de la capa 3, cada proceso podía trabajar con los dispositivos abstractos de E/S con excelentes propiedades, en vez de los dispositivos reales con muchas peculiaridades.

La capa 4 era en donde se encontraban los programas de usuario. No tenían que preocuparse por la administración de los procesos, la memoria, la consola o la E/S.

El proceso operador del sistema se encontraba en el nivel 5. Una mayor generalización del concepto de capas estaba presente en el sistema MULTICS. En vez de capa, MULTICS se describió como una serie de anillos concéntricos, en donde los interiores tenían más privilegios que los exteriores (que en efecto viene siendo lo mismo). Cuando un procedimiento en un anillo exterior quería llamar a un procedimiento en un anillo interior, tenía que hacer el equivalente de una llamada al sistema; es decir, una instrucción TRAP cuyos parámetros se comprobara cuidadosamente que fueran válidos antes de permitir que continuara la llamada. Aunque todo el sistema operativo era parte del espacio de direcciones de cada proceso de usuario en MULTICS, el hardware hizo posible que se designaran procedimientos individuales (en realidad, segmentos de memoria) como protegidos contra lectura, escritura o ejecución. Mientras que en realidad el esquema de capas de THE era sólo una ayuda de diseño, debido a que todas las partes del sistema estaban enlazadas entre sí en un solo programa ejecutable, en

MULTICS el mecanismo de los anillos estaba muy presente en tiempo de ejecución y el hardware se encargaba de implementarlo. La ventaja del mecanismo de los anillos es que se puede extender fácilmente para estructurar los subsistemas de usuario. Por ejemplo, un profesor podría escribir un programa para evaluar y calificar los programas de los estudiantes, ejecutando este programa en el anillo n, mientras que los programas de los estudiantes se ejecutaban en el anillo n1 y por ende no podían cambiar sus calificaciones.

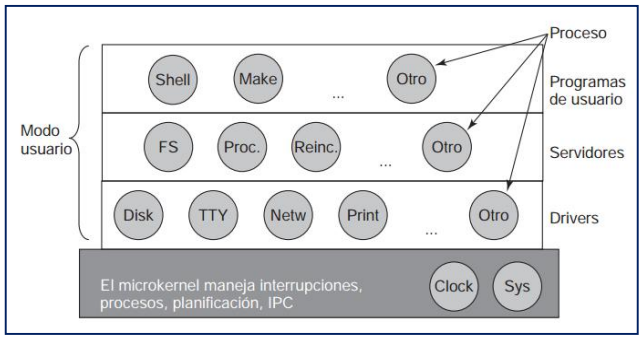
3. Microkernels

Con el diseño de capas, los diseñadores podían elegir en dónde dibujar el límite entre kernel y usuario. Tradicionalmente todas las capas iban al kernel, pero eso no es necesario. De hecho, puede tener mucho sentido poner lo menos que sea posible en modo kernel, debido a que los errores en el kernel pueden paralizar el sistema de inmediato. En contraste, los procesos de usuario se pueden configurar para que tengan menos poder, por lo que un error en ellos tal vez no sería fatal. Varios investigadores han estudiado el número de errores por cada 1000 líneas de código (por ejemplo, Basilli y Perricone, 1984; y Ostrand y Weyuker, 2002). La densidad de los errores depende del tamaño del módulo, su tiempo de vida y más, pero una cifra aproximada para los sistemas industriales formales es de diez errores por cada mil líneas de código. Esto significa que es probable que un sistema operativo monolítico de cinco millones de líneas de código contenga cerca de 50,000 errores en el kernel. Desde luego que no todos estos son fatales, ya que algunos errores pueden ser cosas tales como emitir un mensaje de error incorrecto en una situación que ocurre raras veces. Sin embargo, los sistemas operativos tienen tantos errores que los fabricantes de computadoras colocan botones de reinicio en ellas (a menudo en el panel frontal), algo que los fabricantes de televisiones, estéreos y autos no hacen, a pesar de la gran cantidad de software en estos dispositivos. La idea básica detrás del diseño de microkernel es lograr una alta confiabilidad al dividir el sistema operativo en módulos pequeños y bien definidos, sólo uno de los cuales (el microkernel) se ejecuta en modo kernel y el resto se ejecuta como procesos de usuario ordinarios, sin poder relativamente. En especial, al ejecutar cada driver de dispositivo y sistema de archivos como un proceso de usuario separado, un error en alguno de estos procesos puede hacer que falle ese componente, pero no puede

hacer que falle todo el sistema. Así, un error en el driver del dispositivo de audio hará que el sonido sea confuso o se detenga, pero la computadora no fallará. En contraste, en un sistema monolítico con todos los drivers en el kernel, un driver de audio con errores puede hacer fácilmente referencia a una dirección de memoria inválida y llevar a todo el sistema a un alto rotundo en un instante.

Se han implementado y desplegado muchos microkernels (Accetta y colaboradores, 1986; Haertig y colaboradores, 1997; Heiser y colaboradores, 2006; Herder y colaboradores, 2006; Hildebrand, 1992; Kirsch y colaboradores, 2005; Liedtke, 1993, 1995, 1996; Pike y colaboradores, 1992; y Zuberi y colaboradores, 1999). Son en especial comunes en las aplicaciones en tiempo real, industriales, aeronáuticas y militares que son de misión crítica y tienen requerimientos de confiabilidad muy altos. Algunos de los microkernels mejor conocidos son Integrity, K42, L4, PikeOS, QNX, Symbian y MINIX 3. Ahora veremos en forma breve las generalidades acerca de MINIX 3, que ha llevado la idea de la modularidad hasta el límite, dividiendo la mayor parte del sistema operativo en varios procesos independientes en modo usuario. MINIX 3 es un sistema de código fuente abierto en conformidad con POSIX, disponible sin costo en www.minix3.org (Herder y colaboradores, 2006a; Herder y colaboradores, 2006b).

El microkernel MINIX 3 sólo tiene cerca de 3200 líneas de C y 800 líneas de ensamblador para las funciones de muy bajo nivel, como las que se usan para atrapar interrupciones y conmutar proceso. El código de C administra y planifica los procesos, se encarga de la comunicación entre procesos (al pasar mensajes entre procesos) y ofrece un conjunto de aproximadamente 35 llamadas al kernel para permitir que el resto del sistema operativo realice su trabajo. Estas llamadas realizan funciones tales como asociar los drivers a las interrupciones, desplazar datos entre espacios de direcciones e instalar nuevos mapas de memoria para los procesos recién creados. La estructura de procesos de MINIX 3 se muestra en la gráfica 34, en donde los manejadores de las llamadas



Gráfica 34. Estructura Sistema Minix 3. Fuente: Tanembaum (2009)

al kernel se etiquetan como Sys. El manejador de dispositivo para el reloj también está en el kernel, debido a que el planificador interactúa de cerca con él. Todos los demás dispositivos controladores se ejecutan como procesos de usuario separados.

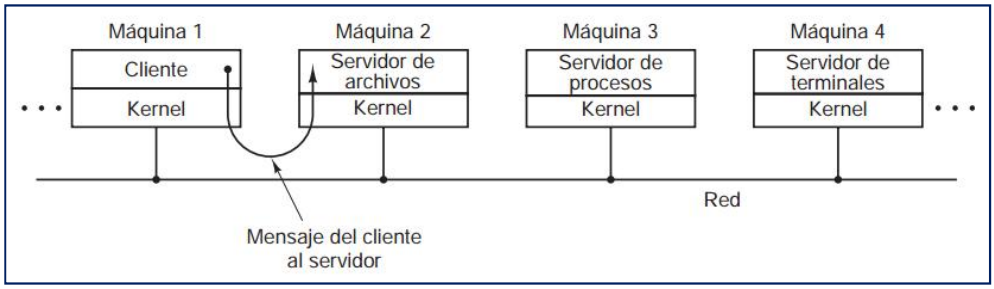
Fuera del kernel, el sistema se estructura como tres capas de procesos, todos se ejecutan en modo usuario. La capa más inferior contiene los drivers de dispositivos. Como todos se ejecutan en modo usuario, no tienen acceso físico al espacio de puertos de E/S y no pueden emitir comandos de E/S directamente. En vez de ello, para programar un dispositivo de E/S el driver crea una estructura para indicarle qué valores debe escribir en cuáles puertos de E/S y realiza una llamada al kernel para indicarle que realice la escritura. Esta metodología permite que el kernel compruebe que el driver esté escribiendo (o leyendo) de la E/S que está autorizado a utilizar. En consecuencia (y a diferencia de un diseño monolítico), un driver de audio defectuoso no puede escribir accidentalmente en el disco. Encima de los drivers hay otra capa en modo usuario que contiene los servidores, que realizan la mayor parte del trabajo del sistema operativo. Uno o más servidores de archivos administran el (los) sistema(s) de archivos, el administrador de procesos crea, destruye y administra los procesos y así sucesivamente. Los programas de usuario obtienen servicios del sistema operativo mediante el envío de mensajes cortos a los servidores, pidiéndoles las llamadas al sistema POSIX. Por ejemplo, un proceso que necesite realizar una llamada read envía un mensaje a uno de los servidores de archivos para indicarle qué debe leer. Un servidor interesante es el servidor de reencarnación, cuyo trabajo es comprobar si otros servidores y drivers están funcionando en forma correcta. En caso de que se detecte uno defectuoso, se reemplaza automáticamente sin intervención del usuario. De esta forma, el sistema es autocorregible y puede lograr una alta confiabilidad. El sistema tiene muchas restricciones que limitan el poder de cada proceso. Como dijimos antes, los drivers sólo pueden utilizar los puertos de E/S autorizados, pero el acceso a las llamadas al kernel también está controlado dependiendo del proceso, al igual que la habilidad de enviar mensajes a otros procesos. Además, los procesos pueden otorgar un permiso limitado a otros procesos para hacer que el kernel acceda a sus espacios de direcciones. Como ejemplo, un sistema de archivos puede otorgar permiso al dispositivo controlador de disco para dejar que el kernel coloque un bloque de disco recién leído en una dirección

específica dentro del espacio de direcciones del sistema de archivos. El resultado de todas estas restricciones es que cada driver y servidor tiene el poder exacto para realizar su trabajo y no más, con lo cual se limita en forma considerable el daño que puede ocasionar un componente defectuoso. Una idea que está en parte relacionada con tener un kernel mínimo es colocar el mecanismo para hacer algo en el kernel, pero no la directiva. Para aclarar mejor este punto, considere la planificación de los proceso. Un algoritmo de planificación relativamente simple sería asignar una prioridad a cada proceso y después hacer que el kernel ejecute el proceso de mayor prioridad que sea ejecutable. El mecanismo, en el kernel, es buscar el proceso de mayor prioridad y ejecutarlo. La directiva, asignar prioridades a los procesos, puede realizarse mediante los procesos en modo usuario. De esta forma, la directiva y el mecanismo se pueden desacoplar y el kernel puede reducir su tamaño.

4. **Cliente-Servidor**

Una ligera variación de la idea del microkernel es diferenciar dos clases de procesos: los servidores, cada uno de los cuales proporciona cierto servicio, y los clientes, que utilizan estos servicios. Este modelo se conoce como cliente-servidor. A menudo la capa inferior es un microkernel, pero eso no es requerido. La esencia es la presencia de procesos cliente y procesos servidor. La comunicación entre clientes y servidores se lleva a cabo comúnmente mediante el paso de mensajes. Para obtener un servicio, un proceso cliente construye un mensaje indicando lo que desea y lo envía al servicio apropiado. Después el servicio hace el trabajo y envía de vuelta la respuesta. Si el cliente y el servidor se ejecutan en el mismo equipo se pueden hacer ciertas optimizaciones, pero en concepto estamos hablando sobre el paso de mensajes. Una generalización obvia de esta idea es hacer que los clientes y los servidores se ejecuten en distintas computadoras, conectadas mediante una red de área local o amplia, como se describe en la gráfica 35. Como los clientes se comunican con los servidores mediante el envío de mensajes, no necesitan saber si los mensajes se manejan en forma local en sus propios equipos o si se envían a través de una red a servidores en un equipo remoto. En cuanto a lo que al cliente concierne, lo mismo ocurre en ambos casos: se envían las peticiones y se regresan las respuestas. Por ende, el modelo cliente-servidor es una abstracción que se puede utilizar para un solo equipo o para una red de equipos. Cada vez hay más sistemas

que involucran a los usuarios en sus PCs domésticas como clientes y equipos más grandes que operan en algún otro lado como servidores. De hecho, la mayor parte de la Web opera de esta forma. Una PC envía una petición de una página Web al servidor y la página Web se envía de vuelta. Éste es un uso común del modelo cliente-servidor en una red.



Gráfica 35. Modelo Cliente-Servidor. Fuente: Tanembaum (2009)

5. Máquinas Virtuales

Las versiones iniciales del OS/360 eran, en sentido estricto, sistemas de procesamiento por lotes. Sin embargo, muchos usuarios del 360 querían la capacidad de trabajar de manera interactiva en una terminal, por lo que varios grupos, tanto dentro como fuera de IBM, decidieron escribir sistemas de tiempo compartido para este sistema. El sistema de tiempo compartido oficial de IBM, conocido como TSS/360, se liberó después de tiempo y cuando por fin llegó era tan grande y lento que pocos sitios cambiaron a este sistema. En cierto momento fue abandonado, una vez que su desarrollo había consumido cerca de 50 millones de dólares (Graham, 1970). Pero un grupo en el *Scientific Center* de IBM en Cambridge, Massachusetts, produjo un sistema radicalmente distinto que IBM aceptó eventualmente como producto. Un descendiente lineal de este sistema, conocido como z/VM, se utiliza ampliamente en la actualidad, en las mainframes de IBM (zSeries) que se utilizan mucho en centros de datos corporativos extensos, por ejemplo, como servidores de comercio electrónico que manejan cientos o miles de transacciones por segundo y utilizan bases de datos cuyos tamaños llegan a ser hasta de varios millones de gigabytes.

VM/370

Gráfica 36. IBM VM/370. Fuente: http://ibm370.webs.com/370155.jpg

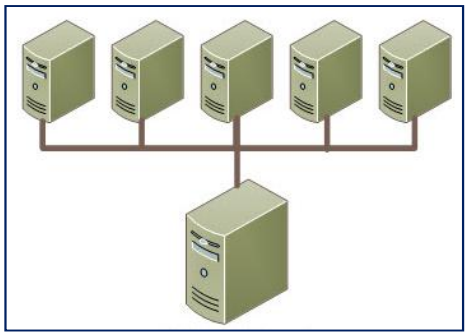
Este sistema, que en un principio se llamó CP/CMS y posteriormente cambió su nombre a VM/370 (Seawright y MacKinnon, 1979), estaba basado en una astuta observación: un sistema de tiempo compartido proporciona (1) multiprogramación y (2) una máquina extendida con una interfaz más conveniente que el hardware por sí solo. La esencia de VM/370 es separar por completo estas dos funciones.

El corazón del sistema, que se conoce como monitor de máquina virtual, se ejecuta en el hardware solamente y realiza la multiprogramación, proporcionando no una, sino varias máquinas virtuales a la siguiente capa hacia arriba. Sin embargo, a diferencia de otros sistemas operativos, estas máquinas virtuales no son máquinas extendidas, con archivos y otras características adecuadas. En vez de ello, son copias exactas del hardware, incluyendo el modo kernel/ usuario, la E/S, las interrupciones y todo lo demás que tiene la máquina real. Como cada máquina virtual es idéntica al verdadero hardware, cada una puede ejecutar cualquier sistema operativo que se ejecute directamente sólo en el hardware. Distintas máquinas virtuales pueden (y con frecuencia lo hacen) ejecutar distintos sistemas operativos. En el sistema VM/370 original, algunas ejecutaban OS/360 o uno de los otros sistemas operativos extensos de procesamiento por lotes o de procesamiento de transacciones, mientras que otros ejecutaban un sistema interactivo de un solo usuario llamado CMS (*Conversational Monitor System*; Sistema monitor conversacional) para los usuarios interactivos de tiempo compartido.

Redescubrimiento de las máquinas virtuales

Mientras que IBM ha tenido un producto de máquina virtual disponible durante cuatro décadas, y unas cuantas compañías más como Sun Microsystems y Hewlett-Packard han agregado recientemente el soporte de máquinas virtuales a sus servidores empresariales de alto rendimiento, la idea de la virtualización se había ignorado por mucho tiempo en el mundo de la PC, hasta hace poco. Pero en los últimos años, se han combinado nuevas necesidades, nuevo software y nuevas tecnologías para convertirla

en un tema de moda. Primero hablaremos sobre las necesidades. Muchas compañías han ejecutado tradicionalmente sus servidores de correo, servidores Web, servidores FTP y otros servidores en computadoras separadas, algunas veces con distintos sistemas operativos. Consideran la virtualización como una forma de ejecutarlos todos en la misma máquina, sin que una falla de un servidor haga que falle el resto.



Gráfica 37. Virtualización de servidores. Fuente: http://www.alojate.com/blog/wp-content/uploads/2011/07/vps.jpg

La virtualización también es popular en el mundo del hospedaje Web. Sin ella, los clientes de hospedaje Web se ven obligados a elegir entre el hospedaje compartido (que les ofrece sólo una cuenta de inicio de sesión en un servidor Web, pero ningún control sobre el software de servidor) y hospedaje dedicado (que les ofrece su propia máquina, lo cual es muy flexible pero no es costeable para los sitios Web de pequeños a medianos). Cuando una compañía de hospedaje Web ofrece la renta de máquinas virtuales, una sola máquina física puede ejecutar muchas máquinas virtuales, cada una de las cuales parece ser una máquina completa. Los clientes que rentan una máquina virtual pueden ejecutar cualesquier sistema operativo y software que deseen, pero a una fracción del costo de un servidor dedicado (debido a que la misma máquina física soporta muchas máquinas virtuales al mismo tiempo).

Otro uso de la virtualización es para los usuarios finales que desean poder ejecutar dos o más sistemas operativos al mismo tiempo, por decir Windows y Linux, debido a que algunos de sus paquetes de aplicaciones favoritos se ejecutan en el primero y algunos otros en el segundo.

Ahora pasemos al software. Aunque nadie disputa lo atractivo de las máquinas virtuales, el problema está en su implementación. Para poder ejecutar software de máquina virtual en una computadora, su CPU debe ser virtualizable (Popek y Goldberg, 1974). En síntesis, he aquí el problema. Cuando un sistema operativo que opera en una máquina virtual (en modo usuario) ejecuta una instrucción privilegiada, tal como para modificar el PSW o realizar una operación de E/S, es esencial que el hardware la atrape para el monitor de la máquina virtual, de manera que la instrucción se pueda emular en el

software. En algunas CPUs (como el Pentium, sus predecesores y sus clones) los intentos de ejecutar instrucciones privilegiadas en modo de usuario simplemente se ignoran. Esta propiedad hacía que fuera imposible tener máquinas virtuales en este hardware, lo cual explica la falta de interés en el mundo de la PC. Desde luego que había intérpretes para el Pentium que se ejecutaban en el Pentium, pero con una pérdida de rendimiento de por lo general 5x a 10x, no eran útiles para un trabajo serio.

Esta situación cambió como resultado de varios proyectos de investigación académicos en la década de 1990, como Disco en Stanford (Bugnion y colaboradores, 1997), que ocasionaron el surgimiento de productos comerciales (por ejemplo, VMware Workstation) y que reviviera el interés en las máquinas virtuales.

Gráfica 38. Logo VMWare

En contraste con los hipervisores de tipo 1, que se ejecutaban en el hardware directo, los hipervisores de tipo 2 se ejecutan como programas de aplicación encima de Windows, Linux o algún otro sistema operativo, conocido como sistema operativo anfitrión. Una vez que se inicia un hipervisor de tipo 2, lee el CD-ROM de instalación para el sistema operativo huésped elegido y lo instala en un disco virtual, que es tan sólo un gran archivo en el sistema de archivos del sistema operativo anfitrión. Cuando se arrancar el sistema operativo huésped, realiza lo mismo que en el hardware real; por lo general inicia algunos procesos en segundo plano y después una GUI. Algunos hipervisores traducen los programas binarios del sistema operativo huésped bloque por bloque, reemplazando ciertas instrucciones de control con llamadas al hipervisor. Después, los bloques traducidos se ejecutan y se colocan en caché para su uso posterior. Un enfoque distinto en cuanto al manejo de las instrucciones de control es el de modificar el sistema operativo para eliminarlas. Este enfoque no es una verdadera virtualización, sino paravirtualización.

La máquina virtual de Java

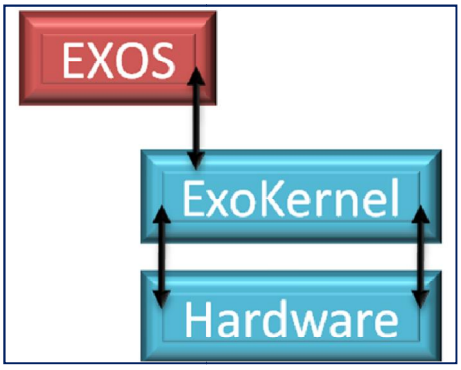
Otra área en donde se utilizan máquinas virtuales, pero de una manera algo distinta, es para ejecutar programas de Java. Cuando Sun Microsystems inventó el lenguaje de programación Java, también inventó una máquina virtual (es decir, una arquitectura de computadora) llamada JVM (*Java Virtual Machine*, Máquina virtual de Java). El

compilador de Java produce código para la JVM, que a su vez típicamente se ejecuta mediante un software intérprete de JVM. La ventaja de este método es que el código de la JVM se puede enviar a través de Internet, a cualquier computadora que tenga un intérprete de JVM y se ejecuta allí. Por ejemplo, si el compilador hubiera producido programas binarios para SPARC o Pentium, no se podrían haber enviado y ejecutado en cualquier parte con la misma facilidad. (Desde luego que Sun podría haber producido un compilador que produjera binarios de SPARC y después distribuir un intérprete de SPARC, pero JVM es una arquitectura mucho más simple de interpretar.) Otra ventaja del uso de la JVM es que, si el intérprete se implementa de manera apropiada, que no es algo completamente trivial, se puede comprobar que los programas de JVM entrantes sean seguros para después ejecutarlos en un entorno protegido, de manera que no puedan robar datos ni realizar algún otro daño.

Gráfica 39. Logo Java.

6. **Exokernels**

En vez de clonar la máquina actual, como se hace con las máquinas virtuales, otra estrategia es particionarla; en otras palabras, a cada usuario se le proporciona un subconjunto de los recursos. Así, una máquina virtual podría obtener los bloques de disco del 0 al 1023, la siguiente podría obtener los bloques de disco del 1024 al 2047 y así sucesivamente. En la capa inferior, que se ejecuta en el modo kernel, hay un programa llamado exokernel (Engler y colaboradores, 1995). Su trabajo es asignar recursos a las máquinas virtuales y después comprobar los intentos de utilizarlos, para asegurar que ninguna máquina trate de usar los recursos de otra. Cada máquina virtual de nivel de usuario puede ejecutar su propio sistema operativo, al igual que en la VM/370 y las Pentium 8086 virtuales, con la excepción de que cada una está restringida a utilizar sólo los recursos que ha pedido y que le han sido asignados. La ventaja del esquema del exokernel es que ahorra una capa de asignación. En los otros diseños, cada máquina virtual piensa que tiene su propio



Gráfica 40. Exokernel. Fuente: http://chsos20121909049.files.wordpress.com/2012/04/exo_kernel.png

disco, con bloques que van desde 0 hasta cierto valor máximo, por lo que el monitor de la máquina virtual debe mantener tablas para reasignar las direcciones del disco (y todos los demás recursos). Con el exokernel, esta reasignación no es necesaria. El exokernel sólo necesita llevar el registro para saber a cuál máquina virtual se le ha asignado cierto recurso. Este método sigue teniendo la ventaja de separar la multiprogramación (en el exokernel) del código del sistema operativo del usuario (en espacio de usuario), pero con menos sobrecarga, ya que todo lo que tiene que hacer el exokernel es mantener las máquinas virtuales separadas unas de las otras.

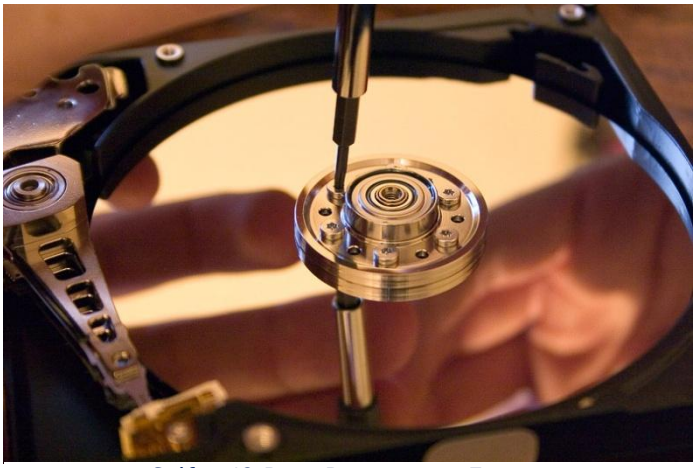
1.7. Complemento: Arranque de la Computadora. Caso Pentium

Gráfica 41. Procesador Pentium. Fuente: www.xataka.com

Cada Pentium contiene una tarjeta madre (*motherboard*). En la tarjeta madre o padre hay un programa conocido como BIOS (*Basic Input Output System*, Sistema básico de entrada y salida) del sistema. El BIOS contiene software de E/S de bajo nivel, incluyendo procedimientos para leer el teclado, escribir en la pantalla y realizar operaciones de E/S de disco, entre otras cosas. Hoy en día está contenido en una RAM tipo flash que es no volátil pero el sistema operativo puede actualizarla cuando se encuentran errores en el BIOS.

Cuando se arranca la computadora, el BIOS inicia su ejecución. Primero hace pruebas para ver cuánta RAM hay instalada y si el teclado junto con otros dispositivos básicos están instalados y responden en forma correcta. Empieza explorando los buses ISA y PCI para detectar todos los dispositivos conectados a ellos. Comúnmente, algunos de estos dispositivos son heredados (es decir, se diseñaron antes de inventar la tecnología *plug and play*), además de tener valores fijos para los niveles de interrupciones y las direcciones de E/S (que posiblemente se establecen mediante interruptores o puentes en la tarjeta de E/S, pero que el sistema operativo no puede modificar). Estos dispositivos se registran; y los dispositivos *plug and play* también. Si los dispositivos presentes son distintos de los que había cuando el sistema se inició por última vez, se configuran los nuevos dispositivos.

Después, el BIOS determina el dispositivo de arranque, para lo cual prueba una lista de dispositivos almacenada en la memoria CMOS. El usuario puede cambiar esta lista si entra a un programa de configuración del BIOS, justo después de iniciar el sistema. Por lo general, se hace un intento por arrancar del disco flexible, si hay uno presente. Si eso falla, se hace una consulta a la unidad de CD-ROM para ver si contiene un CD-ROM que se pueda arrancar. Si no hay disco flexible ni CD-ROM que puedan iniciarse, el sistema se arranca desde el disco duro. El primer sector del dispositivo de arranque se lee y se coloca en la memoria, para luego ejecutarse. Este sector contiene un programa que por lo general examina la tabla de particiones al final del sector de arranque, para determinar qué partición está activa. Después se lee un cargador de arranque secundario de esa partición. Este cargador lee el sistema operativo de la partición activa y lo inicia.



Gráfica 42. Disco Duro interno. Fuente: http://upload.wikimedia.org/wikipedia/commons/1/1d/Hard_disk_platter_reflection.jpg

Luego, el sistema operativo consulta al BIOS para obtener la información de configuración. Para cada dispositivo, comprueba si tiene el driver correspondiente. De no ser así, pide al usuario que inserte un CD-ROM que contenga el driver (suministrado por el fabricante del dispositivo). Una vez que tiene los drivers de todos los dispositivos, el sistema operativo los carga en el kernel. Después inicializa sus tablas, crea los procesos de segundo plano que se requieran, y arranca un programa de inicio de sesión o GUI.

2. Entrada y Salida

Gráfica 43. Periféricos de E/S. Fuente: sistemasoperativosroyer.blogspot.com

Además de proporcionar abstracciones como los procesos (e hilos), espacios de direcciones y archivos, un sistema operativo también controla todos los dispositivos de E/S (Entrada/Salida) de la computadora. Debe emitir comandos para los dispositivos, captar interrupciones y manejar errores.

Adicionalmente debe proporcionar una interfaz —simple y fácil de usar— entre los dispositivos y el resto del sistema. Hasta donde sea posible, la interfaz debe ser igual para todos los dispositivos (independencia de dispositivos). El código de E/S representa una fracción considerable del sistema operativo total. El tema de este capítulo es la forma en que el sistema operativo administra la E/S.

2.1. Subsistema de Entrada y Salida (Interfase)

Toda computadora de propósito general tiene un teclado y un monitor (y por lo general un ratón) para permitir que las personas interactúen con ella. Aunque el teclado y el monitor son dispositivos técnicamente separados, trabajan muy de cerca. En los mainframes con frecuencia hay muchos usuarios remotos, cada uno con un dispositivo que contiene un teclado y una pantalla conectados como una

Gráfica 44. Teclado Microsoft. Fuente: www.jetdicas.com

unidad. Estos dispositivos se conocen históricamente como terminales. Con frecuencia, las personas siguen utilizando ese término, aun cuando hablan sobre los teclados y monitores de las computadoras personales (en gran parte debido a que no hay otro mejor).

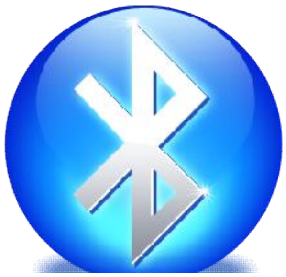
Software de Entrada

La entrada del usuario proviene principalmente del teclado y del ratón; analicemos estos dispositivos. En una computadora personal, el teclado contiene un microprocesador integrado que por lo general se comunica, a través de un puerto serial especializado, con un chip controlador en la tarjeta principal (aunque cada vez con más frecuencia, los teclados se conectan a un puerto USB). Se genera una interrupción cada vez que se oprime una tecla, y se genera una segunda interrupción cada vez que se suelta. En cada una de estas interrupciones de teclado, el software controlador del mismo extrae la información acerca de lo que ocurre desde el puerto de E/S asociado con el teclado. Todo lo demás ocurre en el software y es muy independiente del hardware.

- Software de teclado: El número en el puerto de E/S es el número de tecla, conocido como código de exploración, no el código ASCII. Los teclados tienen menos de 128 teclas, por lo que sólo se necesitan 7 bits para representar el número de tecla. El octavo bit se establece en 0 cuando se oprime una tecla, y en 1 cuando se suelta. Es responsabilidad del controlador llevar un registro del estado de cada tecla (oprimida o suelta).
- Software de ratón: La mayoría de las PCs tienen un ratón, o algunas veces un *trackball*, que sencillamente es un ratón boca arriba. Un tipo común de ratón tiene una bola de goma en su interior que se asoma por un hoyo en la parte inferior y gira, a medida que el ratón se desplaza por una superficie dura, frotándose contra unos rodillos posicionados en ejes ortogonales. El movimiento en la dirección este-oeste hace que gire el eje paralelo al eje y; el movimiento en la dirección norte-sur hace que gire el eje paralelo al eje x. Otro tipo popular de ratón es el óptico, que está equipado con uno o más diodos emisores de luz y

Gráfica 45. Mouse BENQ. Fuente: http://www.gizmos.es/files/2012/07/benq-e200.jpg

fotodetectores en su parte inferior. Los primeros tenían que operar sobre un tapete especial grabado con una rejilla rectangular, de manera que el ratón pudiera contar las líneas que cruzaba. Los ratones ópticos modernos tienen un chip de procesamiento de imágenes en ellos y sacan fotos continuas de baja resolución de la superficie debajo de ellos, buscando cambios de imagen en imagen. Cada vez que un ratón se ha desplazado cierta distancia mínima en cualquier dirección, o cuando se oprime o suelta un botón, se envía un mensaje a la computadora. La distancia mínima es de aproximadamente 0.1 mm (aunque se puede establecer mediante software). Algunas personas llaman a esta unidad *mickey*. Los ratones pueden tener uno, dos o tres botones, dependiendo de la estimación de los diseñadores en cuanto a la habilidad intelectual de los usuarios para llevar la cuenta de más de un botón. Algunos ratones tienen ruedas que pueden enviar datos adicionales a la computadora. Los ratones inalámbricos son iguales a los alámbricos, excepto que en vez de devolver sus datos a la computadora a través de un cable, utilizan radios de baja energía, por ejemplo mediante el uso del estándar Bluetooth. El mensaje para la computadora contiene tres elementos: Δx, Δy, botones. El primer elemento es el cambio en la posición x desde el último mensaje. Después viene el cambio en la posición y desde el último mensaje. Por último, se incluye el estado de los botones. El formato del mensaje depende del sistema y del número de botones que tenga el ratón. Por lo general, requiere de 3 bytes. La mayoría de los ratones se reportan con la computadora un máximo de 40 veces/seg, por lo que el ratón se puede haber desplazado varios *mickeys* desde el último reporte. Observe que el ratón indica sólo los cambios en la posición, no la posición absoluta en sí. Si se recoge el ratón y se coloca de nuevo en su posición gentilmente sin hacer que la bola gire, no se enviarán mensajes. Algunas GUIs diferencian un solo clic y un doble clic de un botón del ratón. Si dos clics están lo bastante cerca en el espacio (*mickeys*) y también en el tiempo (milisegundos), se señala un doble clic. El valor máximo para "lo bastante cerca" depende del software, y por lo general el usuario puede ajustar ambos parámetros.



Gráfica 46. Logo Bluetooth

Software de Salida

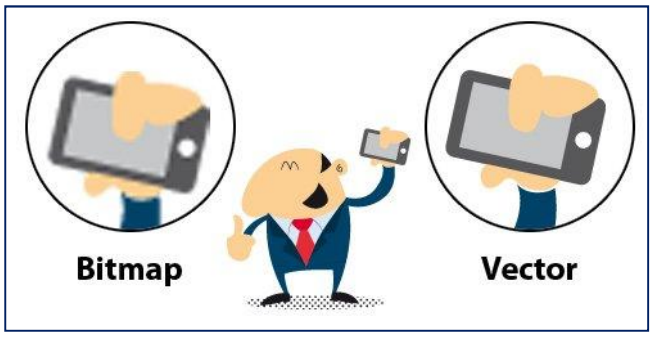
- Ventanas de texto: La salida es más simple que la entrada cuando se envía secuencialmente en un solo tipo de letra, tamaño y color. En su mayor parte, el programa envía caracteres a la ventana en uso y se muestran ahí. Por lo general, un bloque de caracteres (por ejemplo, una línea) se escribe en una llamada al sistema. Los editores de pantalla y muchos otros programas sofisticados necesitan la capacidad de actualizar la pantalla en formas complejas, como sustituir una línea a mitad de la pantalla. Para satisfacer esta necesidad, la mayoría de los controladores de software de salida proporcionan una serie de comandos para desplazar el cursor, insertar y eliminar caracteres o líneas en el cursor, entre otras tareas. A menudo estos comandos se conocen como secuencias de escape. En los días de la terminal "tonta" ASCII de 25 x 80 había cientos de tipos de terminales, cada una con sus propias secuencias de escape. Como consecuencia, era difícil escribir software que funcionara en más de un tipo terminal.
- El sistema X Window: Casi todos los sistemas UNIX basan su interfaz de usuario en el Sistema X Window (que a menudo sólo se le llama X), desarrollado en el M.I.T. como parte del proyecto Athena en la década de 1980. Es muy portátil y se ejecuta por completo en espacio de usuario. En un principio tenía como propósito principal conectar un gran número de terminales de usuario remotas con un servidor de cómputo central, por lo que está dividido lógicamente en software cliente y software servidor, que puede ejecutarse potencialmente en distintas computadoras. En la mayoría de las computadoras modernas, ambas partes se pueden ejecutar en el mismo equipo. En los sistemas Linux, los populares entornos de escritorio Gnome y KDE se ejecutan encima de X. Cuando X se ejecuta en un equipo, el software que recolecta la entrada del teclado y el ratón, y que escribe la salida en la pantalla, se llama servidor X. Este software tiene que llevar el registro de cuál ventana está seleccionada

Gráfica 47. Logo Gnome

en un momento dado (dónde se encuentra el ratón), para saber a qué cliente debe enviar cualquier entrada nueva del teclado. Se comunica con los programas en ejecución (posiblemente a través de una red), llamados clientes X. Les envía la entrada del ratón y del teclado, y acepta los comandos de pantalla de ellos.

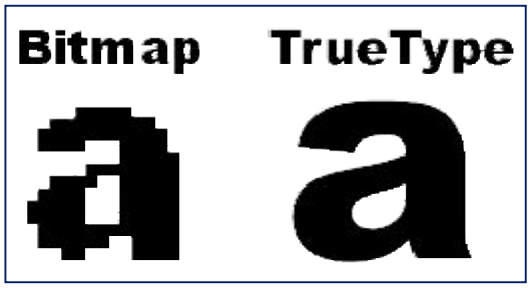
- Interfaces gráficas de usuario: La mayoría de las computadoras personales ofrecen una GUI (*Graphic User Interface*, Interfaz gráfica de usuario). Una GUI tiene cuatro elementos esenciales, denotados por los caracteres WIMP. Las letras representan ventanas (*Windows*), iconos (*Icons*), menús (*Menus*) y dispositivo señalador (*Pointing device*), respectivamente. Las ventanas son áreas rectangulares en la pantalla que se utilizan para ejecutar programas. Los iconos son pequeños símbolos en los que se puede hacer clic para que ocurra una acción. Los menús son listas de acciones, de las que se puede elegir una. Por último, un dispositivo señalador es un ratón, *trackball* u otro dispositivo de hardware utilizado para desplazar un cursor alrededor de la pantalla para seleccionar elementos. El software de GUI se puede implementar en código a nivel de usuario, como en los sistemas UNIX, o en el mismo sistema operativo, como en el caso de Windows. La entrada para los sistemas GUI sigue utilizando el teclado y el ratón, pero la salida casi siempre va a un hardware especial, conocido como adaptador de gráficos. Un adaptador de gráficos contiene una memoria especial conocida como RAM de video que contiene las imágenes que aparecen en la pantalla. Los adaptadores de gráficos de alto rendimiento a menudo tienen CPUs poderosas de 32 o 64 bits, y hasta 1 GB de su propia RAM, separada de la memoria principal de la computadora. La acción de dibujar en la pantalla se maneja mediante un paquete que consiste en cientos de procedimientos, que en conjunto forman la GDI (*Graphics Device Interface*, Interfaz de dispositivo gráfico). Puede manejar texto y todo tipo de gráficos, y está diseñada para ser independiente de la plataforma y del dispositivo. Antes de que un programa pueda dibujar (es decir, pintar) en una ventana, necesita adquirir un contexto de dispositivo, que es una estructura de datos interna que contiene propiedades de la ventana, como el tipo de letra actual, color de texto, color de fondo, etcétera. La mayoría de las llamadas a la GDI utilizan el contexto de dispositivo, ya sea para dibujar o para obtener o establecer las propiedades.

- Mapas de bits: Los procedimientos de la GDI son ejemplos de gráficos vectoriales. Se utilizan para colocar figuras geométricas y texto en la pantalla. Se pueden escalar con facilidad a pantallas más grandes o pequeñas (siempre y cuando el número de pixeles en la pantalla sea el mismo). También son relativamente independientes del dispositivo. Una colección de llamadas a procedimientos de la GDI se puede ensamblar en un archivo que describa un dibujo completo. A dicho archivo se le conoce como metarchivo de Windows, y es ampliamente utilizado para transmitir dibujos de un programa de Windows a otro. Dichos archivos tienen la extensión .wmf.



Gráfica 48. Bitmap Vs Vector. Fuente: http://l4c.me/uploads/sin-titulo-33-1301562150_full550.jpg

- Tipos de letras: En versiones de Windows anteriores a la 3.1, los caracteres se representaban como mapas de bits y se copiaban en la pantalla o en la impresora mediante BitBlt. El problema con eso, es que un mapa de bits que se ve bien en la pantalla es demasiado pequeño para la impresora. La solución fue la introducción de los tipos de letra TrueType, que no son mapas de bits sino contornos de los caracteres. Cada carácter TrueType se define mediante una secuencia de puntos alrededor de su perímetro. Todos los puntos son relativos al origen (0, 0). Mediante este sistema es fácil escalar los caracteres hacia arriba o hacia abajo. Todo lo que se tiene que hacer es multiplicar cada coordenada por el mismo factor de escala. De esta forma, un carácter TrueType se puede escalar hacia arriba o hacia abajo a cualquier tamaño de punto, incluso hasta tamaños de punto fraccionados.



Gráfica 49. Bitmap Vs TrueType. Fuente: http://static.ddmcdn.com/gif/question460-truetype.gif

2.2. Elementos Básicos del Hardware del Subsistema de E/S

Dispositivos de E/S

Los dispositivos de E/S se pueden dividir básicamente en dos categorías: dispositivos de bloque y dispositivos de carácter.

Gráfica 50. Dispositivo de Memoria USB. Fuente: aycarambablog.blogspot.com

- Un dispositivo de bloque almacena información en bloques de tamaño fijo, cada uno con su propia dirección. Los tamaños de bloque comunes varían desde 512 bytes hasta 32,768 bytes. Todas las transferencias se realizan en unidades de uno o más bloques completos (consecutivos). La propiedad esencial de un dispositivo de bloque es que es posible leer o escribir cada bloque de manera independiente de los demás. Los discos duros, CD-ROMs y memorias USBs son dispositivos de bloque comunes.
- Un dispositivo de carácter envía o acepta un flujo de caracteres, sin importar la estructura del bloque. No es direccionable y no tiene ninguna operación de búsqueda. Las impresoras, las interfaces de red, los ratones (para señalar), y la mayoría de los demás dispositivos que no son parecidos al disco se pueden considerar como dispositivos de carácter.

Los dispositivos de E/S cubren un amplio rango de velocidades, lo cual impone una presión considerable en el software para obtener un buen desempeño sobre muchos órdenes de magnitud en las velocidades de transferencia de datos. La siguiente gráfica muestra las velocidades de transferencia de datos de algunos dispositivos comunes. La mayoría de estos dispositivos tienden a hacerse más rápidos a medida que pasa el tiempo.

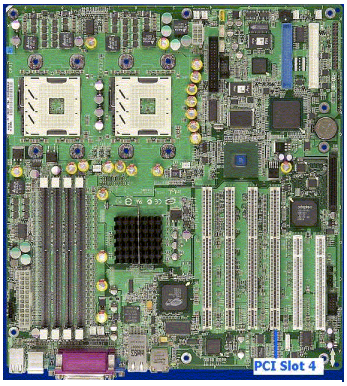
Dispositivo	Velocidad de transferencia de datos
Teclado	10 bytes/seg
Ratón	100 bytes/seg
Módem de 56K	7 KB/seg
Escáner	400 KB/seg
Cámara de video digital	3.5 MB/seg
802.11g inalámbrico	6.75 MB/seg
CD-ROM de 52X	7.8 MB/seg
Fast Ethernet	12.5 MB/seg
Tarjeta Compact Flash	40 MB/seg
FireWire (IEEE 1394)	50 MB/seg
USB 2.0	60 MB/seg
Red SONET OC-12	78 MB/seg
Disco SCSI Ultra 2	80 MB/seg
Gigabit Ethernet	125 MB/seg
Unidad de disco SATA	300 MB/seg
Cinta de Ultrium	320 MB/seg
Bus PCI	528 MB/seg

Gráfica 51. Velocidades de transferencia de datos comunes de algunos dispositivos, redes y buses. Fuente: Tanembaum (2009)

Controladores de Dispositivos:

Por lo general, las unidades de E/S consisten en un componente mecánico y un componente electrónico. A menudo es posible separar las dos porciones para proveer un diseño más modular y general. El componente electrónico se llama controlador de dispositivo o adaptador. En las computadoras personales, comúnmente tiene la forma de un chip en la tarjeta principal o una tarjeta de circuito integrado que se puede insertar en una ranura de expansión (PCI). El componente mecánico es el dispositivo en sí.

La tarjeta controladora por lo general contiene un conector, en el que se puede conectar un cable que conduce al dispositivo en sí. Muchos controladores pueden manejar dos, cuatro o inclusive ocho dispositivos idénticos. Si la interfaz entre el controlador y el dispositivo es estándar, ya sea un estándar oficial ANSI, IEEE o ISO, o un estándar de facto, entonces las empresas pueden fabricar controladores o dispositivos que se adapten a esa interfaz. Por ejemplo, muchas empresas fabrican unidades de disco que coinciden con la interfaz IDE, SATA, SCSI, USB o FireWire (IEEE 1394).



Gráfica 52. Tarjeta madre con Ranura PCI. Fuente: http://www.intel.com/support/pix/motherboards/server/pci_slot.gif

En los Sistemas con DMA, los dispositivos pueden acceder a la Memoria Principal sin intervención del CPU.

E/S por asignación de memoria

Cada controlador tiene unos cuantos registros que se utilizan para comunicarse con la CPU. Al escribir en ellos, el sistema operativo puede hacer que el dispositivo envíe o acepte datos, se encienda o se apague, o realice cualquier otra acción. Al leer de estos registros, el sistema operativo puede conocer el estado del dispositivo, si está preparado o no para aceptar un nuevo comando, y sigue procediendo de esa manera. Además de los registros de control, muchos dispositivos tienen un búfer de datos que el sistema operativo puede leer y escribir. Por ejemplo, una manera común para que las computadoras muestren píxeles en la pantalla es tener una RAM de video, la cual es básicamente sólo un búfer de datos disponible para que los programas o el sistema operativo escriban en él. De todo esto surge la cuestión acerca de cómo se comunica la CPU con los registros de control y los búferes de datos de los dispositivos. Existen dos alternativas. En el primer método, a cada registro de control se le asigna un número de puerto de E/S, un entero de 8 o 16 bits. El conjunto de todos los puertos de E/S forma el espacio de puertos de E/S y está protegido de manera que los programas de usuario ordinarios no puedan utilizarlo (sólo el sistema operativo puede).

Acceso directo a memoria (DMA)

Sin importar que una CPU tenga o no E/S por asignación de memoria, necesita direccionar los controladores de dispositivos para intercambiar datos con ellos. La CPU puede solicitar datos de un controlador de E/S un bit a la vez, pero al hacerlo se desperdicia el tiempo de la CPU, por lo que a menudo se utiliza un esquema distinto, conocido como DMA (Acceso Directo a Memoria). El sistema operativo sólo puede utilizar DMA si el hardware tiene un controlador de DMA, que la mayoría de los sistemas tienen. Algunas veces este controlador está integrado a los controladores de disco y otros controladores, pero dicho diseño requiere un controlador de DMA separado para cada dispositivo. Lo más común es que haya un solo controlador de DMA disponible (por ejemplo, en la tarjeta principal) para regular las transferencias a varios dispositivos, a menudo en forma concurrente. Sin importar cuál sea su ubicación física, el controlador de DMA tiene acceso al bus del sistema de manera independiente de la CPU.

Contiene varios registros en los que la CPU puede escribir y leer; éstos incluyen un registro de dirección de memoria, un registro contador de bytes y uno o más registros de control. Los registros de control especifican el puerto de E/S a utilizar, la dirección de la transferencia (si se va a leer del dispositivo de E/S o se va a escribir en él), la unidad de transferencia (un byte a la vez o una palabra a la vez), y el número de bytes a transferir en una ráfaga.

Los controladores de DMA varían considerablemente en cuanto a su sofisticación. Los más simples manejan una transferencia a la vez, como se describió antes. Los más complejos se pueden programar para manejar varias transferencias a la vez. Dichos controladores tienen varios conjuntos de registros internos, uno para cada canal. La CPU empieza por cargar cada conjunto de registros con los parámetros relevantes para su transferencia. Cada transferencia debe utilizar un controlador de dispositivo distinto.

2.3. Elementos Básicos del Software del Subsistema de E/S

Objetivos del software de E/S

Un concepto clave en el diseño del software de E/S se conoce como independencia de dispositivos. Lo que significa es que debe ser posible escribir programas que puedan acceder a cualquier dispositivo de E/S sin tener que especificar el dispositivo por adelantado. Por ejemplo, un programa que lee un archivo como entrada debe tener la capacidad de leer un archivo en el disco duro, un CD-ROM, un DVD o una memoria USB sin tener que modificar el programa para cada dispositivo distinto.

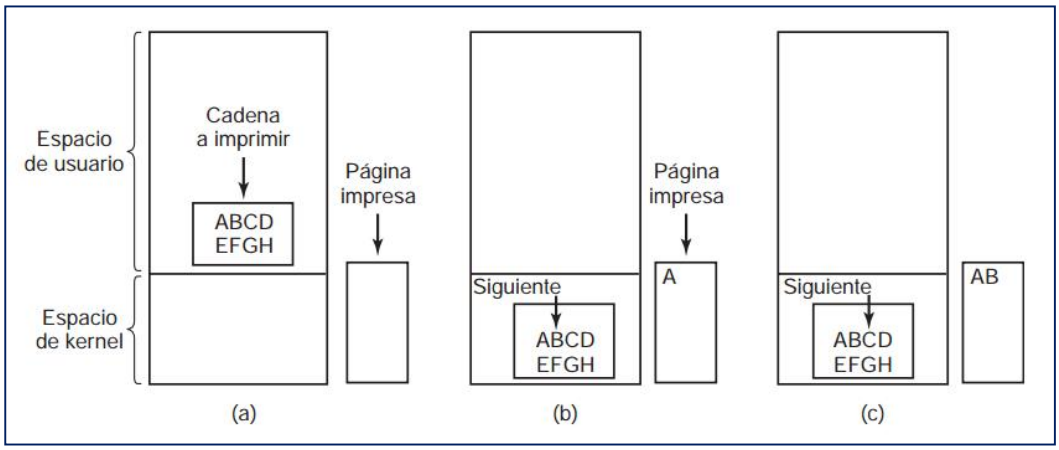
- Un objetivo muy relacionado con la independencia de los dispositivos es la **denominación uniforme**. El nombre de un archivo o dispositivo simplemente debe ser una cadena o un entero sin depender del dispositivo de ninguna forma. En UNIX, todos los discos se pueden integrar en la jerarquía del sistema de archivos de maneras arbitrarias, por lo que el usuario no necesita estar al tanto de cuál nombre corresponde a cuál dispositivo. Por ejemplo, una memoria USB se puede montar encima del directorio /usr/ast/respaldo, de manera que al copiar un archivo a /usr/ast/respaldo/lunes, este archivo se copie a la memoria USB. De esta forma, todos los archivos y dispositivos se direccionan de la misma forma: mediante el nombre de una ruta.

- Otra cuestión importante relacionada con el software de E/S es el **manejo de errores**. En general, los errores se deben manejar lo más cerca del hardware que sea posible. Si el controlador descubre un error de lectura, debe tratar de corregir el error por sí mismo. Si no puede, entonces el software controlador del dispositivo debe manejarlo, tal vez con sólo tratar de leer el bloque de nuevo. Muchos errores son transitorios, como los errores de lectura ocasionados por pizcas de polvo en la cabeza de lectura, y comúnmente desaparecen si se repite la operación. Sólo si los niveles inferiores no pueden lidiar con el problema, los niveles superiores deben saber acerca de ello. En muchos casos, la recuperación de los errores se puede hacer de manera transparente a un nivel bajo, sin que los niveles superiores se enteren siquiera sobre el error.
- Otra cuestión clave es la de las **transferencias síncronas** (de bloqueo) contra las asíncronas (controladas por interrupciones). La mayoría de las operaciones de E/S son asíncronas: la CPU inicia la transferencia y se va a hacer algo más hasta que llega la interrupción. Los programas de usuario son mucho más fáciles de escribir si las operaciones de E/S son de bloqueo: después de una llamada al sistema read, el programa se suspende de manera automática hasta que haya datos disponibles en el búfer. Depende del sistema operativo hacer que las operaciones que en realidad son controladas por interrupciones parezcan de bloqueo para los programas de usuario.
- Otra cuestión relacionada con el software de E/S es el **uso de búfer**. A menudo los datos que provienen de un dispositivo no se pueden almacenar directamente en su destino final. Por ejemplo, cuando un paquete llega de la red, el sistema operativo no sabe dónde colocarlo hasta que ha almacenado el paquete en alguna parte y lo examina. Además, algunos dispositivos tienen severas restricciones en tiempo real (por ejemplo, los dispositivos de audio digital), por lo que los datos se deben colocar en un búfer de salida por adelantado para desacoplar la velocidad a la que se llena el búfer, de la velocidad a la que se vacía, de manera que se eviten sub-desbordamientos de búfer. El uso de búfer involucra una cantidad considerable de copiado y a menudo tiene un importante impacto en el rendimiento de la E/S.
- El concepto final que mencionaremos aquí es la comparación entre los **dispositivos compartidos y los dispositivos dedicados**. Algunos dispositivos de E/S, como los discos, pueden ser utilizados por muchos usuarios a la vez. No se producen problemas debido a

que varios usuarios tengan archivos abiertos en el mismo disco al mismo tiempo. Otros dispositivos, como las unidades de cinta, tienen que estar dedicados a un solo usuario hasta que éste termine. Después, otro usuario puede tener la unidad de cinta. Si dos o más usuarios escriben bloques entremezclados al azar en la misma cinta, definitivamente no funcionará. Al introducir los dispositivos dedicados (no compartidos) también se introduce una variedad de problemas, como los interbloqueos. De nuevo, el sistema operativo debe ser capaz de manejar los dispositivos compartidos y dedicados de una manera que evite problemas.

E/S Programada

Hay tres maneras fundamentalmente distintas en que se puede llevar a cabo la E/S. La forma más simple de E/S es cuando la CPU hace todo el trabajo. A este método se le conoce como E/S programada. Es más simple ilustrar la E/S programada por medio de un ejemplo. Considere un proceso de usuario que desea imprimir la cadena de ocho caracteres "ABCDEFGH" en la impresora. Primero ensambla la cadena en un búfer en espacio de usuario, como se muestra en la gráfica 53(a). Después el proceso de usuario adquiere la impresora para escribir, haciendo una llamada al sistema para abrirla. Si la impresora está actualmente siendo utilizada por otro proceso, esta llamada fallará y devolverá un código de error o se bloqueará hasta que la impresora esté disponible, dependiendo del sistema operativo y los parámetros de la llamada. Una vez que obtiene la impresora, el proceso de usuario hace una llamada al sistema para indicar al sistema operativo que imprima la cadena en la impresora.



Gráfica 53. Pasos para imprimir una cadena de caracteres. Fuente: Tanenbaum (2009)

Después, el sistema operativo por lo general copia el búfer con la cadena a un arreglo, por ejemplo, p en espacio de kernel, donde se puede utilizar con más facilidad (debido a que el kernel tal vez tenga que modificar el mapa de memoria para tener acceso al espacio de usuario). Después comprueba si la impresora está disponible en ese momento. Si no lo está, espera hasta que lo esté. Tan pronto como la impresora está disponible, el sistema operativo copia el primer carácter al registro de datos de la impresora, en este ejemplo mediante el uso de E/S por asignación de memoria. Esta acción activa la impresora. El carácter tal vez no aparezca todavía, debido a que algunas impresoras colocan en búfer una línea o una página antes de imprimir algo. No obstante, en la gráfica 53(b) podemos ver que se ha impreso el primer carácter y que el sistema ha marcado a "B" como el siguiente carácter a imprimir.

Tan pronto como copia el primer carácter a la impresora, el sistema operativo comprueba si la impresora está lista para aceptar otro. En general la impresora tiene un segundo registro, que proporciona su estado. El acto de escribir en el registro de datos hace que el estado se convierta en "no está lista". Cuando el controlador de la impresora ha procesado el carácter actual, indica su disponibilidad estableciendo cierto bit en su registro de estado, o colocando algún valor en él. En este punto el sistema operativo espera a que la impresora vuelva a estar lista. Cuando eso ocurre, imprime el siguiente carácter, como se muestra en la gráfica 53(c). Este ciclo continúa hasta que se ha impreso toda la cadena. Después el control regresa al proceso de usuario.

En resumen: primero se copian los datos en el kernel. Después el sistema operativo entra a un ciclo estrecho, imprimiendo los caracteres uno a la vez. El aspecto esencial de la E/S programada, es que después de imprimir un carácter, la CPU sondea en forma continua el dispositivo para ver si está listo para aceptar otro. Este comportamiento se conoce comúnmente como sondeo u ocupado en espera.

La E/S programada es simple, pero tiene la desventaja de ocupar la CPU tiempo completo hasta que se completen todas las operaciones de E/S. Si el tiempo para "imprimir" un carácter es muy corto (debido a que todo lo que hace la impresora es copiar el nuevo carácter a un búfer interno), entonces está bien usar ocupado en espera. Además, en un sistema incrustado o embebido, donde la CPU no tiene nada más que hacer, ocupado en espera es razonable. Sin

embargo, en sistemas más complejos en donde la CPU tiene otros trabajos que realizar, ocupado en espera es ineficiente. Se necesita un mejor método de E/S.

E/S controlada por interrupciones

Ahora vamos a considerar el caso de imprimir en una impresora que no coloca los caracteres en un búfer, sino que imprime cada uno a medida que va llegando. Si la impresora puede imprimir (por ejemplo,) 100 caracteres/seg, cada carácter requiere 10 mseg para imprimirse. Esto significa que después de escribir cada carácter en el registro de datos de la impresora, la CPU estará en un ciclo de inactividad durante 10 mseg, esperando a que se le permita imprimir el siguiente carácter. Este tiempo es más que suficiente para realizar un cambio de contexto y ejecutar algún otro proceso durante los 10 mseg que, de otra manera, se desperdiciarían.

La forma de permitir que la CPU haga algo más mientras espera a que la impresora esté lista es utilizar interrupciones. Cuando se realiza la llamada al sistema para imprimir la cadena, el búfer se copia en espacio de kernel (como vimos antes) y el primer carácter se copia a la impresora, tan pronto como esté dispuesta para aceptar un carácter. En ese momento, la CPU llama al planificador y se ejecuta algún otro proceso. El proceso que pidió imprimir la cadena se bloquea hasta que se haya impreso toda la cadena.

Cuando la impresora ha impreso el carácter, y está preparada para aceptar el siguiente, genera una interrupción. Esta interrupción detiene el proceso actual y guarda su estado. Después se ejecuta el procedimiento de servicio de interrupciones de la impresora. Si no hay más caracteres por imprimir, el manejador de interrupciones realiza cierta acción para desbloquear al usuario. En caso contrario, imprime el siguiente carácter, reconoce la interrupción y regresa al proceso que se estaba ejecutando justo antes de la interrupción, que continúa desde donde se quedó.

E/S mediante el uso de DMA

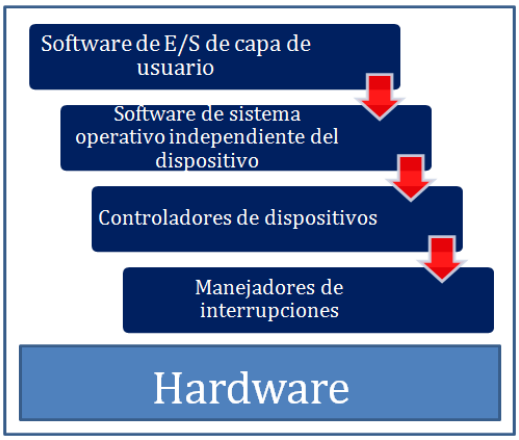
Una obvia desventaja de la E/S controlada por interrupciones es que ocurre una interrupción en cada carácter. Las interrupciones requieren tiempo, por lo que este esquema desperdicia cierta cantidad de tiempo de la CPU. Una solución es utilizar DMA. Aquí la idea es permitir que el controlador de DMA alimente los caracteres a la impresora uno a la vez, sin que la CPU se

moleste. En esencia, el DMA es E/S programada, sólo que el controlador de DMA realiza todo el trabajo en vez de la CPU principal. Esta estrategia requiere hardware especial (el controlador de DMA) pero libera la CPU durante la E/S para realizar otro trabajo.

La gran ganancia con DMA es reducir el número de interrupciones, de una por cada carácter a una por cada búfer impreso. Si hay muchos caracteres y las interrupciones son lentas, esto puede ser una gran mejora. Por otra parte, el controlador de DMA es comúnmente más lento que la CPU principal. Si el controlador de DMA no puede controlar el dispositivo a toda su velocidad, o si la CPU por lo general no tiene nada que hacer mientras espera la interrupción de DMA, entonces puede ser mejor utilizar la E/S controlada por interrupción o incluso la E/S programada. De todas formas, la mayor parte del tiempo vale la pena usar DMA.

Capas del Software de E/S

Por lo general, el software de E/S se organiza en cuatro capas, como se muestra en la gráfica 54. Cada capa tiene una función bien definida que realizar, y una interfaz bien definida para los niveles adyacentes. La funcionalidad y las interfaces difieren de un sistema a otro, por lo que el análisis que veremos a continuación, que examina todas las capas empezando desde el inferior, no es específico de una sola máquina.



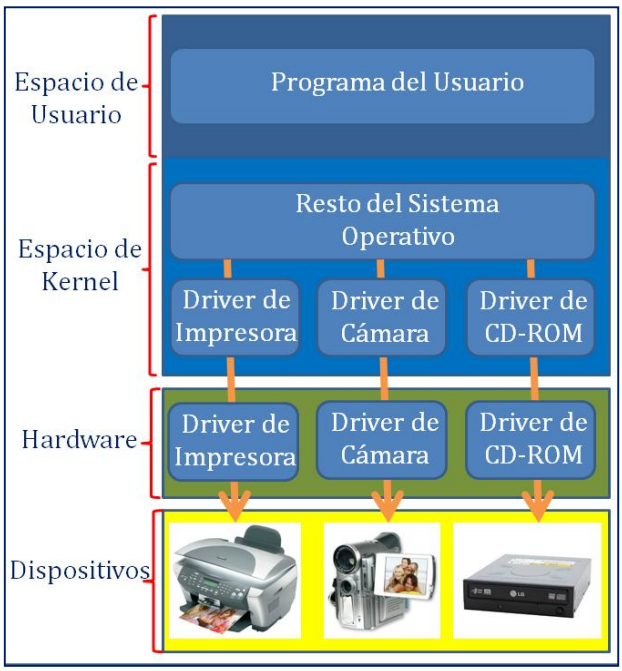
Gráfica 54. Capas del sistema de software de E/S. Fuente: elaboración propia.

- **Manejadores de Interrupciones**: Aunque la E/S programada es útil algunas veces, para la mayor parte de las operaciones de E/S las interrupciones son un hecho incómodo de la vida y no se pueden evitar. Deben ocultarse en la profundidad de las entrañas del

sistema operativo, de manera que éste sepa lo menos posible de ellas. La mejor manera de ocultarlas es hacer que el controlador que inicia una operación de E/S se bloquee hasta que se haya completado la E/S y ocurra la interrupción. El controlador se puede bloquear a sí mismo realizando una llamada a *down* en un semáforo, una llamada a *wait* en una variable de condición, una llamada a *receive* en un mensaje o algo similar, por ejemplo. Cuando ocurre la interrupción, el procedimiento de interrupciones hace todo lo necesario para poder manejarla. Después puede desbloquear el controlador que la inició. En algunos casos sólo completará *up* en un semáforo. En otros casos realizará una llamada a *signal* en una variable de condición en un monitor. En otros más enviará un mensaje al controlador bloqueado. En todos los casos, el efecto neto de la interrupción será que un controlador que estaba bloqueado podrá ejecutarse ahora. Este modelo funciona mejor si los controladores están estructurados como procesos del kernel, con sus propios estados, pilas y contadores del programa. Desde luego que en realidad esto no es tan simple. Procesar una interrupción no es cuestión de sólo tomar la interrupción, llamar a *up* en algún semáforo y después ejecutar una instrucción *IRET* para regresar de la interrupción al proceso anterior. Hay mucho más trabajo involucrado para el sistema operativo.

- **Controladores (o Drivers) de Dispositivos**: cada controlador tiene ciertos registros de dispositivos que se utilizan para darle comandos o ciertos registros de dispositivos que se utilizan para leer su estado, o ambos. El número de registros de dispositivos y la naturaleza de los comandos varían radicalmente de un dispositivo a otro. Por ejemplo, un driver de ratón tiene que aceptar información del ratón que le indica qué tanto se ha desplazado y cuáles botones están oprimidos en un momento dado. Por el contrario, un driver de disco tal vez tenga que saber todo acerca de los sectores, pistas, cilindros, cabezas, movimiento del brazo, los propulsores del motor, los tiempos de asentamiento de las cabezas y todos los demás mecanismos para hacer que el disco funcione en forma apropiada. Obviamente, estos drivers serán muy distintos. Como consecuencia, cada dispositivo de E/S conectado a una computadora necesita cierto código específico para controlarlo. Este código, conocido como driver, es escrito por el fabricante del dispositivo y se incluye junto con el mismo. Como cada sistema operativo necesita sus propios drivers, los fabricantes de dispositivos por lo común los proporcionan para

varios sistemas operativos populares. Cada driver maneja un tipo de dispositivo o, a lo más, una clase de dispositivos estrechamente relacionados. Por ejemplo, un driver de disco SCSI puede manejar por lo general varios discos SCSI de distintos tamaños y velocidades, y tal vez un CD-ROM SCSI también. Por otro lado, un ratón y una palanca de mandos son tan distintos que por lo general se requieren controladores diferentes. Sin embargo, no hay una restricción técnica en cuanto a que un driver controle varios dispositivos no relacionados. Simplemente no es una buena idea. Para poder utilizar el hardware del dispositivo (es decir, los registros del controlador físico), el driver por lo general tiene que formar parte del kernel del sistema operativo, cuando menos en las arquitecturas actuales. En realidad es posible construir controladores que se ejecuten en el espacio de usuario, con llamadas al sistema para leer y escribir en los registros del dispositivo. Este diseño aísla al kernel de los controladores, y a un controlador de otro, eliminando una fuente importante de fallas en el sistema: controladores con errores que interfieren con el kernel de una manera u otra. Para construir sistemas altamente confiables, ésta es, en definitiva, la mejor manera de hacerlo. Un ejemplo de un sistema donde los controladores de dispositivos se ejecutan como procesos de usuario es MINIX 3. Sin embargo, como la mayoría de los demás sistemas operativos de escritorio esperan que los controladores se ejecuten en el kernel, éste es el modelo que consideraremos aquí. Como los diseñadores de cada sistema operativo saben qué piezas de código (*drivers*) escritas por terceros se instalarán en él, necesita tener una arquitectura que permita dicha instalación. Esto implica tener un modelo bien definido de lo que hace un *driver* y la forma en que interactúa con el resto del sistema operativo. Por lo general, los controladores de dispositivos se posicionan



Gráfica 55. Posicionamiento lógico del software controlador de dispositivos. Fuente: elaboración propia.

debajo del resto del sistema operativo. Generalmente los sistemas operativos clasifican los controladores en una de un pequeño número de categorías. Las categorías más comunes son los dispositivos de bloque como los discos, que contienen varios bloques de datos que se pueden direccionar de manera independiente, y los dispositivos de carácter como los teclados y las impresoras, que generan o aceptan un flujo de caracteres.

En un sistema con "conexión en caliente" es posible agregar o eliminar dispositivos mientras la computadora está en ejecución. Como resultado, mientras un controlador está ocupado leyendo de algún dispositivo, el sistema puede informarle que el usuario ha quitado de manera repentina ese dispositivo del sistema. No sólo se debe abortar la transferencia actual de E/S sin dañar ninguna estructura de datos del kernel, sino que cualquier petición pendiente del ahora desaparecido dispositivo debe eliminarse también, con cuidado, del sistema, avisando a los que hicieron la llamada. Además, la adición inesperada de nuevos dispositivos puede hacer que el kernel haga malabares con los recursos (por ejemplo, las líneas de petición de interrupciones), quitando los anteriores al controlador y dándole nuevos recursos en vez de los otros.

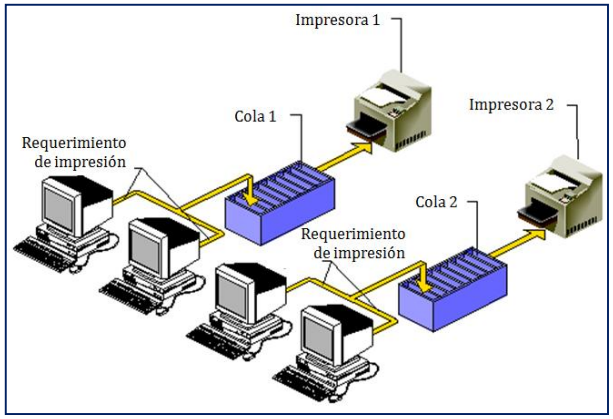
El controlador no puede hacer llamadas al sistema, pero a menudo necesita interactuar con el resto del kernel. Por lo general se permiten llamadas a ciertos procedimientos del kernel. Por ejemplo, comúnmente hay llamadas para asignar y desasignar páginas fijas de memoria para usarlas como búferes. Otras llamadas útiles se necesitan para administrar la MMU, los temporizadores, el controlador de DMA, el controlador de interrupciones, etc.

- **Software de Sistema Operativo independiente del dispositivo**: Aunque parte del software de E/S es específico para cada dispositivo, otras partes de éste son independientes de los dispositivos. El límite exacto entre los controladores y el software independiente del dispositivo depende del sistema (y del dispositivo), debido a que ciertas funciones que podrían realizarse de una manera independiente al dispositivo pueden realizarse en los controladores, por eficiencia u otras razones. Las siguientes funciones se realizan comúnmente en el software independiente del dispositivo:
    - Interfaz uniforme para controladores de dispositivos
    - Uso de búfer

- Reporte de errores
- Asignar y liberar dispositivos dedicados
- Proporcionar un tamaño de bloque independiente del dispositivo

La función básica del software independiente del dispositivo es realizar las funciones de E/S que son comunes para todos los dispositivos y proveer una interfaz uniforme para el software a nivel de usuario.

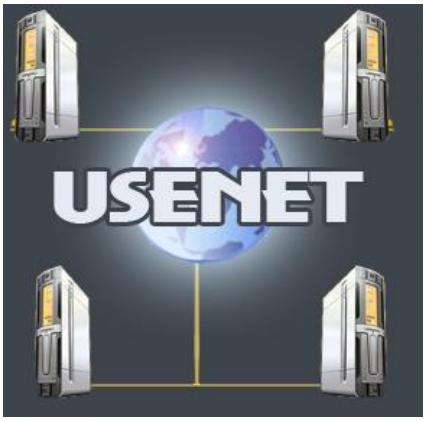
- **Software de E/S de capa de usuario**: Aunque la mayor parte del software de E/S está dentro del sistema operativo, una pequeña porción de éste consiste en bibliotecas vinculadas entre sí con programas de usuario, e incluso programas enteros que se ejecutan desde el exterior del kernel. Las llamadas al sistema, incluyendo las llamadas al sistema de E/S, se realizan comúnmente mediante procedimientos de biblioteca.



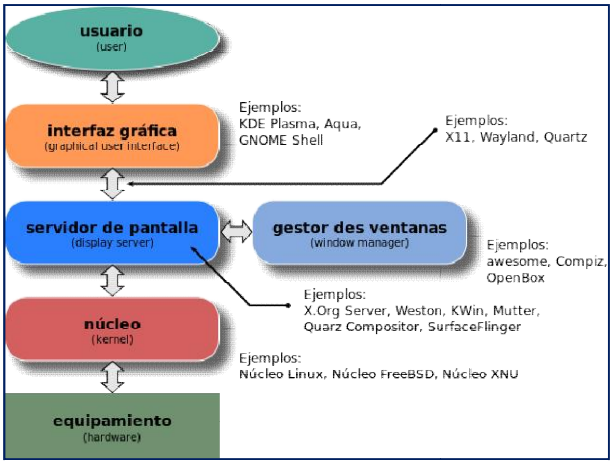
Gráfica 56. Colas de Impresión. Fuente: elaboración propia

En especial, el formato de la entrada y la salida se lleva a cabo mediante procedimientos de biblioteca. No todo el software de E/S de bajo nivel consiste en procedimientos de biblioteca. Otra categoría importante es el sistema de colas. El uso de colas (*spooling*) es una manera de lidiar con los dispositivos de E/S dedicados en un sistema de multiprogramación. Considere un dispositivo común que utiliza colas: una impresora. Aunque sería técnicamente sencillo dejar que cualquier proceso de usuario abriera el archivo de caracteres especial para la impresora, suponga que un proceso lo abriera y no hiciera nada durante horas. Ningún otro proceso podría imprimir nada. En vez de ello, lo que se hace es crear un proceso especial, conocido como demonio, y un directorio especial llamado directorio de cola de impresión. Para imprimir un archivo, un proceso genera primero todo el archivo que va a imprimir y lo coloca en el directorio de la cola de impresión. Es responsabilidad del demonio, que es el único proceso que tiene permiso para usar el archivo especial de la impresora, imprimir los archivos en el directorio. Al proteger el archivo especial contra el uso directo por parte de los usuarios, se elimina el problema de que alguien lo mantenga abierto por un tiempo

innecesariamente extenso. El uso de colas no es exclusivo de las impresoras. También se utiliza en otras situaciones de E/S. Por ejemplo, la transferencia de archivos a través de una red utiliza con frecuencia un demonio de red. Para enviar un archivo a cierta parte, un usuario lo coloca en un directorio de la cola de red. Más adelante, el demonio de red lo toma y lo transmite. Un uso específico de la transmisión de archivos mediante el uso de una cola es el sistema de noticias USENET. Esta red consiste en millones de máquinas en todo el mundo, que se comunican mediante Internet. Existen miles de grupos de noticias sobre muchos temas. Para publicar un mensaje, el usuario invoca a un programa de noticias, el cual acepta el mensaje a publicar y luego lo deposita en un directorio de cola para transmitirlo a las otras máquinas más adelante. Todo el sistema de noticias se ejecuta fuera del sistema operativo.



2.4. Complemento: GUI (*Graphical User Interface* – Interfaz Gráfica de Usuario)



Gráfica 57. Esquema de Capas GUI. Fuente: http://upload.wikimedia.org/wikipedia/commons/1/12/Esquema_de_las_capas_de_la_interfaz_gr%C3%A1fica_de_usuario.svg

La interfaz gráfica de usuario, conocida también como GUI (del inglés *graphical user interface*) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

Habitualmente las acciones se realizan mediante manipulación directa, para facilitar la interacción del usuario con la computadora. Surge como evolución de las interfaces de línea de comandos que se usaban para operar los primeros sistemas operativos y es pieza fundamental en un entorno gráfico. Como ejemplos de interfaz

gráfica de usuario, cabe citar los entornos de escritorio Windows, el X-Window de GNU/Linux o el de Mac OS X, Aqua.

En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático[5].

3. Administración de Procesos

3.1. Procesos

Un proceso es en esencia un programa en ejecución. Cada proceso tiene asociado un espacio de direcciones, una lista de ubicaciones de memoria que va desde algún mínimo (generalmente 0) hasta cierto valor máximo, donde el proceso puede leer y escribir información. El espacio de direcciones contiene el programa ejecutable, los datos del programa y su pila. También hay asociado a cada proceso un conjunto de recursos, que comúnmente incluye registros (el contador de programa y el apuntador de pila, entre ellos), una lista de archivos abiertos, alarmas pendientes, listas de procesos relacionados y toda la demás información necesaria para ejecutar el programa. En esencia, un proceso es un recipiente que guarda toda la información necesaria para ejecutar un programa.

Gráfica 58. Sesión con varios procesos abiertos y en ejecución. Fuente: elaboración propia.

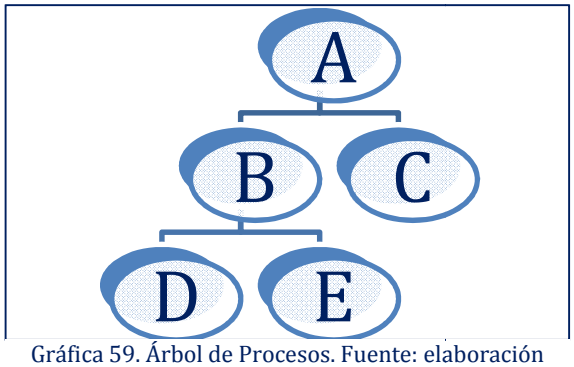
El usuario puede haber iniciado un programa de edición de video para convertir un video de una hora a un formato específico (algo que puede tardar horas) y después irse a navegar en la Web. Mientras tanto, un proceso en segundo plano que despierta en forma periódica para comprobar los mensajes entrantes puede haber empezado a ejecutarse. Así tenemos (cuando menos) tres procesos activos: el editor de video, el navegador Web y el lector de correo electrónico. Cada cierto tiempo, el sistema operativo decide detener la ejecución de un proceso

y empezar a ejecutar otro; por ejemplo, debido a que el primero ha utilizado más tiempo del que le correspondía de la CPU en el último segundo.

Cuando un proceso se suspende en forma temporal como en el ejemplo anterior, debe reiniciarse después exactamente en el mismo estado que tenía cuando se detuvo. Esto significa que toda la información acerca del proceso debe guardarse en forma explícita en alguna parte durante la suspensión. Por ejemplo, el proceso puede tener varios archivos abiertos para leerlos al mismo tiempo. Con cada uno de estos archivos hay un apuntador asociado que proporciona la posición actual (es decir, el número del byte o registro que se va a leer a continuación). Cuando un proceso se suspende en forma temporal, todos estos apuntadores deben guardarse de manera que una llamada a *read* que se ejecute después de reiniciar el proceso lea los datos apropiados. En muchos sistemas operativos, toda la información acerca de cada proceso (además del contenido de su propio espacio de direcciones) se almacena en una tabla del sistema operativo, conocida como la tabla de procesos, la cual es un arreglo (o lista enlazada) de estructuras, una para cada proceso que se encuentre actualmente en existencia.

Así, un proceso (suspendido) consiste en su espacio de direcciones, que se conoce comúnmente como imagen de núcleo (en honor de las memorias de núcleo magnético utilizadas antaño) y su entrada en la tabla de procesos, que guarda el contenido de sus registros y muchos otros elementos necesarios para reiniciar el proceso más adelante.

Las llamadas al sistema de administración de procesos clave son las que se encargan de la creación y la terminación de los procesos. Considere un ejemplo común. Un proceso llamado intérprete de comandos o *shell* lee comandos de una terminal. El usuario acaba de escribir un comando, solicitando la compilación de un programa. El *shell* debe entonces crear un proceso para ejecutar el compilador. Cuando ese proceso ha terminado la compilación, ejecuta una llamada al sistema para terminarse a sí mismo.



Gráfica 59. Árbol de Procesos. Fuente: elaboración propia

Si un proceso puede crear uno o más procesos aparte (conocidos como procesos hijos) y estos procesos a su vez pueden crear procesos hijos,

llegamos rápidamente la estructura de árbol de procesos de la gráfica 58. Los procesos relacionados que cooperan para realizar un cierto trabajo a menudo necesitan comunicarse entre sí y sincronizar sus actividades. A esta comunicación se le conoce como comunicación entre procesos.

Hay otras llamadas al sistema de procesos disponibles para solicitar más memoria (o liberar la memoria sin utilizar), esperar a que termine un proceso hijo y superponer su programa con uno distinto. En algunas ocasiones se tiene la necesidad de transmitir información a un proceso en ejecución que no está esperando esta información. Por ejemplo, un proceso que se comunica con otro, en una computadora distinta, envía los mensajes al proceso remoto a través de una red de computadoras. Para protegerse contra la posibilidad de que se pierda un mensaje o su contestación, el emisor puede solicitar que su propio sistema operativo le notifique después de cierto número de segundos para que pueda retransmitir el mensaje, si no se ha recibido aún la señal de aceptación. Después de asignar este temporizador, el programa puede continuar realizando otro trabajo. Cuando ha transcurrido el número especificado de segundos, el sistema operativo envía una señal de alarma al proceso. La señal provoca que el proceso suspenda en forma temporal lo que esté haciendo, almacene sus registros en la pila y empiece a ejecutar un procedimiento manejador de señales especial, por ejemplo, para retransmitir un mensaje que se considera perdido. Cuando termina el manejador de señales, el proceso en ejecución se reinicia en el estado en el que se encontraba justo antes de la señal. Las señales son la analogía en software de las interrupciones de hardware y se pueden generar mediante una variedad de causas además de la expiración de los temporizadores. Muchas *traps* detectadas por el hardware, como la ejecución de una instrucción ilegal o el uso de una dirección inválida, también se convierten en señales que se envían al proceso culpable.

Cada persona autorizada para utilizar un sistema recibe una UID (*User Identification*, Identificación de usuario) que el administrador del sistema le asigna. Cada proceso iniciado tiene el UID de la persona que lo inició. Un proceso hijo tiene el mismo UID que su padre. Los usuarios pueden ser miembros de grupos, cada uno de los cuales tiene una GID (*Group Identification*, Identificación de grupo).

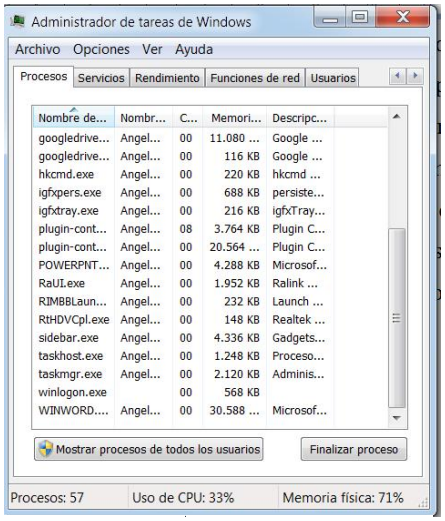
Una UID conocida como superusuario (*superuser* en UNIX) tiene poder especial y puede violar muchas de las reglas de protección. En instalaciones extensas, sólo el administrador del sistema conoce la contraseña requerida para convertirse en superusuario, pero muchos de los usuarios ordinarios (en especial los estudiantes) dedican un esfuerzo considerable para tratar de encontrar fallas en el sistema que les permitan convertirse en superusuario sin la contraseña.

Creación de un proceso

Los sistemas operativos necesitan cierta manera de crear procesos. En sistemas muy simples o sistemas diseñados para ejecutar sólo una aplicación (por ejemplo, el controlador en un horno de microondas), es posible tener presentes todos los procesos que se vayan a requerir cuando el sistema inicie. No obstante, en los sistemas de propósito general se necesita cierta forma de crear y terminar procesos según sea necesario durante la operación. Ahora analizaremos varias de estas cuestiones.

Hay cuatro eventos principales que provocan la creación de procesos:

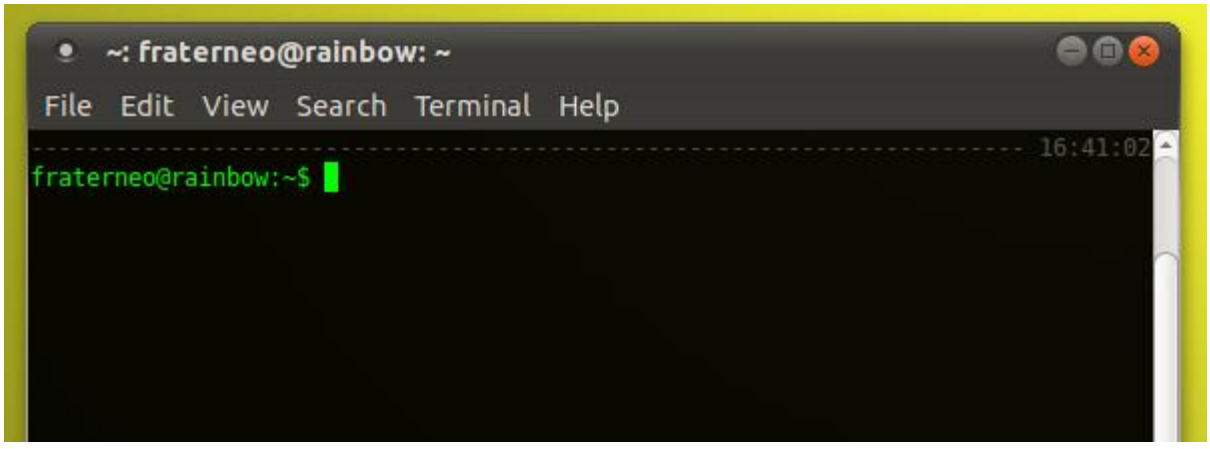
- El arranque del sistema.
- La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.
- Una petición de usuario para crear un proceso.
- El inicio de un trabajo por lotes.



Gráfica 60. Administrador de Tareas de MS Windows. Fuente: elaboración propia.

Generalmente, cuando se arranca un sistema operativo se crean varios procesos. Algunos de ellos son procesos en primer plano; es decir, procesos que interactúan con los usuarios (humanos) y realizan trabajo para ellos. Otros son procesos en segundo plano, que no están asociados con usuarios específicos sino con una función específica. Por ejemplo, se puede diseñar un proceso en segundo plano para

aceptar el correo electrónico entrante, que permanece inactivo la mayor parte del día pero que se activa cuando llega un mensaje. Se puede diseñar otro proceso en segundo plano para aceptar peticiones entrantes para las páginas Web hospedadas en ese equipo, que despierte cuando llegue una petición para darle servicio. Los procesos que permanecen en segundo plano para manejar ciertas actividades como correo electrónico, páginas Web, noticias, impresiones, etcétera, se conocen como demonios (*daemons*). Los sistemas grandes tienen comúnmente docenas de ellos. En UNIX podemos utilizar el programa *ps* para listar los procesos en ejecución. En Windows podemos usar el administrador de tareas. Además de los procesos que se crean al arranque, posteriormente se pueden crear otros. A menudo, un proceso en ejecución emitirá llamadas al sistema para crear uno o más procesos nuevos, para que le ayuden a realizar su trabajo. En especial, es útil crear procesos cuando el trabajo a realizar se puede formular fácilmente en términos de varios procesos interactivos relacionados entre sí, pero independientes en los demás aspectos. Por ejemplo, si se va a obtener una gran cantidad de datos a través de una red para su posterior procesamiento, puede ser conveniente crear un proceso para obtener los datos y colocarlos en un búfer compartido, mientras un segundo proceso remueve los elementos de datos y los procesa. En un multiprocesador, al permitir que cada proceso se ejecute en una CPU distinta también se puede hacer que el trabajo se realice con mayor rapidez.



Gráfica 61. Cónsola de UNIX. Fuente: http://k02.kn3.net/D2354849A.png

En los sistemas interactivos, los usuarios pueden iniciar un programa escribiendo un comando o haciendo (doble) clic en un icono. Cualquiera de las dos acciones inicia un proceso y ejecuta el programa seleccionado. En los sistemas UNIX basados en comandos que ejecutan X, el nuevo

proceso se hace cargo de la ventana en la que se inició. En Microsoft Windows, cuando se inicia un proceso no tiene una ventana, pero puede crear una (o más) y la mayoría lo hace. En ambos sistemas, los usuarios pueden tener varias ventanas abiertas a la vez, cada una ejecutando algún proceso. Mediante el ratón, el usuario puede seleccionar una ventana e interactuar con el proceso, por ejemplo para proveer datos cuando sea necesario.

La última situación en la que se crean los procesos se aplica sólo a los sistemas de procesamiento por lotes que se encuentran en las mainframes grandes. Aquí los usuarios pueden enviar trabajos de procesamiento por lotes al sistema (posiblemente en forma remota). Cuando el sistema operativo decide que tiene los recursos para ejecutar otro trabajo, crea un proceso y ejecuta el siguiente trabajo de la cola de entrada. Técnicamente, en todos estos casos, para crear un proceso hay que hacer que un proceso existente ejecute una llamada al sistema de creación de proceso. Ese proceso puede ser un proceso de usuario en ejecución, un proceso del sistema invocado mediante el teclado o ratón, o un proceso del administrador de procesamiento por lotes. Lo que hace en todo caso es ejecutar una llamada al sistema para crear el proceso. Esta llamada al sistema indica al sistema operativo que cree un proceso y le indica, directa o indirectamente, cuál programa debe ejecutarlo.

En UNIX sólo hay una llamada al sistema para crear un proceso: fork. Esta llamada crea un clon exacto del proceso que hizo la llamada. Después de fork, los dos procesos (padre e hijo) tienen la misma imagen de memoria, las mismas cadenas de entorno y los mismos archivos abiertos. Eso es todo. Por lo general, el proceso hijo ejecuta después a execveo una llamada al sistema similar para cambiar su imagen de memoria y ejecutar un nuevo programa. Por ejemplo, cuando un usuario escribe un comando tal como sort en el shell, éste crea un proceso hijo, que a su vez ejecuta a sort. La razón de este proceso de dos pasos es para permitir al hijo manipular sus descriptores de archivo después de fork, pero antes de execve, para poder lograr la redirección de la entrada estándar, la salida estándar y el error estándar. Por el contrario, en Windows una sola llamada a una función de Win32 (CreateProcess) maneja la creación de procesos y carga el programa correcto en el nuevo proceso. Esta llamada tiene 10 parámetros, que incluyen el programa a ejecutar, los parámetros de la línea de comandos para introducir datos a ese programa, varios atributos de seguridad, bits que controlan si los archivos abiertos se heredan, información de prioridad, una especificación de la ventana que se va a crear para el

proceso (si se va a crear una) y un apuntador a una estructura en donde se devuelve al proceso que hizo la llamada la información acerca del proceso recién creado. Además de CreateProcess, Win32 tiene cerca de 100 funciones más para administrar y sincronizar procesos y temas relacionados.

Tanto en UNIX como en Windows, una vez que se crea un proceso, el padre y el hijo tienen sus propios espacios de direcciones distintos. Si cualquiera de los procesos modifica una palabra en su espacio de direcciones, esta modificación no es visible para el otro proceso. En UNIX, el espacio de direcciones inicial del hijo es una copia del padre, pero en definitiva hay dos espacios de direcciones distintos involucrados; no se comparte memoria en la que se pueda escribir (algunas implementaciones de UNIX comparten el texto del programa entre los dos, debido a que no se puede modificar). Sin embargo, es posible para un proceso recién creado compartir algunos de los otros recursos de su creador, como los archivos abiertos. En Windows, los espacios de direcciones del hijo y del padre son distintos desde el principio.

Terminación de procesos

Una vez que se crea un proceso, empieza a ejecutarse y realiza el trabajo al que está destinado. Sin embargo, nada dura para siempre, ni siquiera los procesos. Tarde o temprano el nuevo proceso terminará, por lo general debido a una de las siguientes condiciones:

- Salida normal (voluntaria).
- Salida por error (voluntaria).
- Error fatal (involuntaria).
- Eliminado por otro proceso (involuntaria).

La mayoría de los procesos terminan debido a que han concluido su trabajo. Cuando un compilador ha compilado el programa que recibe, ejecuta una llamada al sistema para indicar al sistema operativo que ha terminado. Esta llamada es exit en UNIX y ExitProcess en Windows. Los programas orientados a pantalla también admiten la terminación voluntaria. Los procesadores de palabras, navegadores de Internet y programas similares siempre tienen un icono o elemento de menú en el que el usuario puede hacer clic para indicar al proceso que elimine todos los archivos temporales que tenga abiertos y después termine.

La segunda razón de terminación es que el proceso descubra un error. Por ejemplo, si un usuario escribe el comando *cc foo.c* para compilar el programa foo.c y no existe dicho archivo, el compilador simplemente termina. Los procesos interactivos orientados a pantalla por lo general no terminan cuando reciben parámetros incorrectos. En vez de ello, aparece un cuadro de diálogo y se le pide al usuario que intente de nuevo.

La tercera razón de terminación es un error fatal producido por el proceso, a menudo debido a un error en el programa. Algunos ejemplos incluyen el ejecutar una instrucción ilegal, hacer referencia a una parte de memoria no existente o la división entre cero. En algunos sistemas (como UNIX), un proceso puede indicar al sistema operativo que desea manejar ciertos errores por sí mismo, en cuyo caso el proceso recibe una señal (se interrumpe) en vez de terminar.

Gráfica 62. Aplicación de "kill" en UNIX. Fuente: http://www.tech-faq.com

La cuarta razón por la que un proceso podría terminar es que ejecute una llamada al sistema que indique al sistema operativo que elimine otros procesos. En UNIX esta llamada es kill. La función correspondiente en Win32 es TerminateProcess. En ambos casos, el proceso eliminador debe tener la autorización necesaria para realizar la eliminación. En algunos sistemas, cuando un proceso termina (ya sea en forma voluntaria o forzosa) todos los procesos que creó se eliminan de inmediato también. Sin embargo, ni Windows ni UNIX trabajan de esta forma.

3.2. Comunicación entre procesos

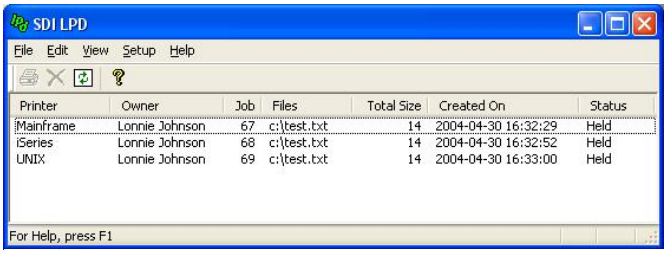
Con frecuencia, los procesos necesitan comunicarse con otros procesos. Por ejemplo, en una canalización del shell, la salida del primer proceso se debe pasar al segundo proceso y así sucesivamente. Por ende, existe una necesidad de comunicación entre procesos, de preferencia en una forma bien estructurada sin utilizar interrupciones.

En resumen, hay tres cuestiones aquí. La primera se alude a lo anterior: cómo un proceso puede pasar información a otro. La segunda está relacionada con hacer que dos o más procesos no se interpongan entre sí; por ejemplo, dos procesos en un sistema de reservaciones de una

aerolínea, cada uno de los cuales trata de obtener el último asiento en un avión para un cliente distinto. La tercera trata acerca de obtener la secuencia apropiada cuando hay dependencias presentes: si el proceso A produce datos y el proceso B los imprime, B tiene que esperar hasta que A haya producido algunos datos antes de empezar a imprimir. En la siguiente sección analizaremos las tres cuestiones.

También es importante mencionar que dos de estas cuestiones se aplican de igual forma a los hilos. La primera (el paso de información) es fácil para los hilos, ya que comparten un espacio de direcciones común (los hilos en distintos espacios de direcciones que necesitan comunicarse entran en la categoría de los procesos en comunicación). Sin embargo, las otras dos (evitar que un hilo entre en conflicto con los demás y obtener la secuencia apropiada) se aplican de igual forma a los hilos. Existen los mismos problemas y se aplican las mismas soluciones. A continuación veremos el problema en el contexto de los procesos, pero tenga en cuenta que también se aplican los mismos problemas y soluciones a los hilos.

Condiciones de Carrera



Gráfica 63. Cola de impresión. Fuente: http://sdisw.com/images/lpd_ss.png

En algunos sistemas operativos, los procesos que trabajan en conjunto pueden compartir cierto espacio de almacenamiento en el que pueden leer y escribir datos. El almacenamiento compartido puede estar en la memoria principal (posiblemente en una estructura de datos del kernel) o puede ser un archivo compartido; la ubicación de la memoria compartida no cambia la naturaleza de la comunicación o los problemas que surgen. Para ver cómo funciona la comunicación entre procesos en la práctica, consideremos un ejemplo simple pero común: un *spooler* de impresión. Cuando un proceso desea imprimir un archivo, introduce el nombre del archivo en un directorio de *spooler* especial. Otro proceso, el demonio de impresión, comprueba en forma periódica si hay archivos que deban imprimirse y si los hay, los imprime y luego elimina sus nombres del directorio.

Regiones críticas

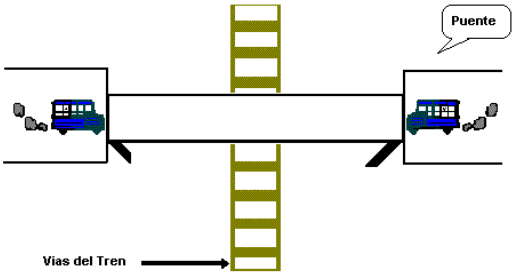
¿Cómo evitamos las condiciones de carrera? La clave para evitar problemas aquí y en muchas otras situaciones en las que se involucran la memoria compartida, los archivos compartidos y todo lo demás compartido es buscar alguna manera de prohibir que más de un proceso lea y escriba los datos compartidos al mismo tiempo. Dicho en otras palabras, lo que necesitamos es exclusión mutua, cierta forma de asegurar que si un proceso está utilizando una variable o archivo compartido, los demás procesos se excluirán de hacer lo mismo. La dificultad antes mencionada ocurrió debido a que el proceso B empezó a utilizar una de las variables compartidas antes de que el proceso A terminara con ella. La elección de operaciones primitivas apropiadas para lograr la exclusión mutua es una cuestión de diseño importante en cualquier sistema operativo y un tema que analizaremos con mayor detalle en las siguientes secciones.

El problema de evitar las condiciones de carrera también se puede formular de una manera abstracta. Parte del tiempo, un proceso está ocupado realizando cálculos internos y otras cosas que no producen condiciones de carrera. Sin embargo, algunas veces un proceso tiene que acceder a la memoria compartida o a archivos compartidos, o hacer otras cosas críticas que pueden producir carreras. Esa parte del programa en la que se accede a la memoria compartida se conoce como región crítica o sección crítica. Si pudiéramos ordenar las cosas de manera que dos procesos nunca estuvieran en sus regiones críticas al mismo tiempo, podríamos evitar las carreras.

Aunque este requerimiento evita las condiciones de carrera, no es suficiente para que los procesos en paralelo cooperen de la manera correcta y eficiente al utilizar datos compartidos. Necesitamos cumplir con cuatro condiciones para tener una buena solución:

- No puede haber dos procesos de manera simultánea dentro de sus regiones críticas.
- No pueden hacerse suposiciones acerca de las velocidades o el número de CPUs.
- Ningún proceso que se ejecute fuera de su región crítica puede bloquear otros procesos.
- Ningún proceso tiene que esperar para siempre para entrar a su región crítica.

Exclusión mutua con espera ocupada



Gráfica 64. Principio de exclusión mutua. Fuente: http://exa.unne.edu.ar

Se presentan a continuación varias proposiciones para lograr la exclusión mutua, de manera que mientras un proceso esté ocupado actualizando la memoria compartida en su región crítica, ningún otro proceso puede entrar a su región crítica y ocasionar problemas.

- **Deshabilitando interrupciones**: En un sistema con un solo procesador, la solución más simple es hacer que cada proceso deshabilite todas las interrupciones justo después de entrar a su región crítica y las rehabilite justo después de salir. Con las interrupciones deshabilitadas, no pueden ocurrir interrupciones de reloj. Después de todo, la CPU sólo se conmuta de un proceso a otro como resultado de una interrupción del reloj o de otro tipo, y con las interrupciones desactivadas la CPU no se conmutará a otro proceso. Por ende, una vez que un proceso ha deshabilitado las interrupciones, puede examinar y actualizar la memoria compartida sin temor de que algún otro proceso intervenga.

  Por lo general este método es poco atractivo, ya que no es conveniente dar a los procesos de usuario el poder para desactivar las interrupciones. Suponga que uno de ellos lo hiciera y nunca las volviera a activar. Ése podría ser el fin del sistema; aún más: si el sistema es un multiprocesador (con dos o posiblemente más CPUs), al deshabilitar las interrupciones sólo se ve afectada la CPU que ejecutó la instrucción *disable*. Las demás continuarán ejecutándose y pueden acceder a la memoria compartida.

  Por otro lado, con frecuencia es conveniente para el mismo kernel deshabilitar las interrupciones por unas cuantas instrucciones mientras actualiza variables o listas. Por ejemplo, si ocurriera una interrupción mientras la lista de procesos se encuentra en un estado inconsistente, podrían producirse condiciones de carrera. La conclusión es que a menudo deshabilitar interrupciones es una técnica útil dentro del mismo sistema operativo, pero no es apropiada como mecanismo de exclusión mutua general para los procesos de usuario.

La posibilidad de lograr la exclusión mutua al deshabilitar las interrupciones (incluso dentro del kernel) está disminuyendo día con día debido al creciente número de chips multinúcleo que se encuentran hasta en las PCs de bajo rendimiento. Ya es común que haya dos núcleos, las máquinas actuales de alto rendimiento tienen cuatro y dentro de poco habrá ocho o 16. En un multinúcleo (es decir, sistema con multiprocesadores) al deshabilitar las interrupciones de una CPU no se evita que las demás CPUs interfieran con las operaciones que la primera CPU está realizando. En consecuencia, se requieren esquemas más sofisticados.

- **Variables de candado**: Como segundo intento, busquemos una solución de software. Considere tener una sola variable compartida (de candado), que al principio es 0. Cuando un proceso desea entrar a su región crítica primero evalúa el candado. Si este candado es 0, el proceso lo fija en 1 y entra a la región crítica. Si el candado ya es 1 sólo espera hasta que el candado se haga 0. Por ende, un 0 significa que ningún proceso está en su región crítica y un 1 significa que algún proceso está en su región crítica.

Gráfica 65. Aviso en la puerta de baño en un avión. El usuario que entra cierra y se muestra aviso de "ocupado" (candado 1). Fuente: http://img.diariodelviajero.com/2010/02/banoavion.jpg

Por desgracia, esta idea contiene exactamente el mismo error fatal que vimos en el directorio de spooler. Suponga que un proceso lee el candado y ve que es 0. Antes de que pueda fijar el candado a 1, otro proceso se planifica para ejecutarse y fija el candado a 1. Cuando el primer proceso se ejecuta de nuevo, también fija el candado a 1 y por lo tanto dos procesos se encontrarán en sus regiones críticas al mismo tiempo.

- **Alternancia estricta**: esta solución requiere que los dos procesos se alternen de manera estricta al entrar en sus regiones críticas (por ejemplo, al poner archivos en la cola de impresión). Ninguno podría poner dos archivos en la cola al mismo tiempo. Aunque este algoritmo evita todas las condiciones de carrera, en realidad no es un candidato serio como solución.
- **Solución de Peterson**: Al combinar la idea de tomar turnos con la idea de las variables de candado y las variables de advertencia, un matemático holandés llamado T. Dekker fue

el primero en idear una solución de software para el problema de la exclusión mutua que no requiere de una alternancia estricta.

En 1981, G.L. Peterson descubrió una manera mucho más simple de lograr la exclusión mutua, con lo cual la solución de Dekker se hizo obsoleta. El algoritmo de Peterson consiste de dos procedimientos escritos en ANSI C.

Antes de utilizar las variables compartidas (es decir, antes de entrar a su región crítica), cada proceso llama a entrar_region con su propio número de proceso (0 o 1) como parámetro. Esta llamada hará que espere, si es necesario, hasta que sea seguro entrar. Una vez que haya terminado con las variables compartidas, el proceso llama a salir_region para indicar que ha terminado y permitir que los demás procesos entren, si así lo desea.

Al principio ningún proceso se encuentra en su región crítica. Ahora el proceso 0 llama a entrar_region. Indica su interés estableciendo su elemento del arreglo y fija turno a 0. Como el proceso 1 no está interesado, entrar_region regresa de inmediato. Si ahora el proceso 1 hace una llamada a entrar_region, se quedará ahí hasta que interesado[0] sea FALSE, un evento que sólo ocurre cuando el proceso 0 llama a salir_region para salir de la región crítica.

```
#define FALSE 0
#define TRUE 1
#define N   2                                   /* número de procesos */

int turno;                                      /* ¿de quién es el turno? */
int interesado[N];                              /* al principio todos los valores son 0 (FALSE) */

void entrar_region(int proceso);                /* el proceso es 0 o 1 */
{
    int otro;                                   /* número del otro proceso */

    otro = 1 – proceso;                         /* el opuesto del proceso */
    interesado[proceso] = TRUE;                 /* muestra que está interesado */
    turno = proceso;                            /* establece la bandera */
    while (turno == proceso && interesado[otro] == TRUE)  /* instrucción nula */;
}

void salir_region(int proceso)                  /* proceso: quién está saliendo */
{
    interesado[proceso] = FALSE;                /* indica que salió de la región crítica */
}
```

Gráfica 66. Solución de Peterson. Fuente: Tanembaum (2009)

Ahora considere el caso en el que ambos procesos llaman a entrar_region casi en forma simultánea. Ambos almacenarán su número de proceso en turno. Cualquier

almacenamiento que se haya realizado al último es el que cuenta; el primero se sobrescribe y se pierde. Suponga que el proceso 1 almacena al último, por lo que turno es 1. Cuando ambos procesos llegan a la instrucción while, el proceso 0 la ejecuta 0 veces y entra a su región crítica. El proceso 1 itera y no entra a su región crítica sino hasta que el proceso 0 sale de su región crítica.

Semáforos[6]

Un semáforo es una variable especial (o tipo abstracto de datos) que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento del sistema o variables del código fuente) en un entorno de multiprocesamiento (en el que se ejecutarán varios procesos concurrentemente). Fueron inventados por Edsger Dijkstra en 1965 y se usaron por primera vez en el sistema operativo THEOS.

Gráfica 67. Semáforo de tráfico

Los semáforos se emplean para permitir el acceso a diferentes partes de programas (llamados secciones críticas) donde se manipulan variables o recursos que deben ser accedidos de forma especial. Según el valor con que son inicializados se permiten a más o menos procesos utilizar el recurso de forma simultánea.

Un tipo simple de semáforo es el binario, que puede tomar solamente los valores 0 y 1. Se inicializan en 1 y son usados cuando sólo un proceso puede acceder a un recurso a la vez. Son esencialmente lo mismo que los mutex. Cuando el recurso está disponible, un proceso accede y decrementa el valor del semáforo con la operación P. El valor queda entonces en 0, lo que hace que si otro proceso intenta decrementarlo tenga que esperar. Cuando el proceso que decrementó el semáforo realiza una operación V, algún proceso que estaba esperando comienza a utilizar el recurso.

Para hacer que dos procesos se ejecuten en una secuencia predeterminada puede usarse un semáforo inicializado en 0. El proceso que debe ejecutar primero en la secuencia realiza la operación V sobre el semáforo antes del código que debe ser ejecutado después del otro proceso. Éste ejecuta la operación P. Si el segundo proceso en la secuencia es programado para

ejecutar antes que el otro, al hacer P dormirá hasta que el primer proceso de la secuencia pase por su operación V. Este modo de uso se denomina señalación (*signaling*), y se usa para que un proceso o hilo de ejecución le haga saber a otro que algo ha sucedido.

El otro uso de los semáforos es para la sincronización. Los semáforos vacíos y llenos se necesitan para garantizar que ciertas secuencias de eventos ocurran o no. En este caso, aseguran que el productor deje de ejecutarse cuando el búfer esté lleno y que el consumidor deje de ejecutarse cuando el búfer esté vacío. Este uso es distinto de la exclusión mutua.

Mutex

Cuando no se necesita la habilidad del semáforo de contar, algunas veces se utiliza una versión simplificada, llamada mutex. Los mutexes son buenos sólo para administrar la exclusión mutua para cierto recurso compartido o pieza de código. Se implementan con facilidad y eficiencia, lo cual hace que sean especialmente útiles en paquetes de hilos que se implementan en su totalidad en espacio de usuario.

Gráfica 68. Mutex actúa como un candado. Fuente: http://info.quadros.com/Portals/62908/images/Mutex.jpg

Un mutex es una variable que puede estar en uno de dos estados: abierto (desbloqueado) o cerrado (bloqueado). En consecuencia, se requiere sólo 1 bit para representarla, pero en la práctica se utiliza con frecuencia un entero, en donde 0 indica que está abierto y todos los demás valores indican que está cerrado. Se utilizan dos procedimientos con los mutexes. Cuando un hilo (o proceso) necesita acceso a una región crítica, llama a mutex_lock. Si el mutex está actualmente abierto (lo que significa que la región crítica está disponible), la llamada tiene éxito y entonces el hilo llamador puede entrar a la región crítica.

Por otro lado, si el mutex ya se encuentra cerrado, el hilo que hizo la llamada se bloquea hasta que el hilo que está en la región crítica termine y llame a mutex_unlock. Si se bloquean varios hilos por el mutex, se selecciona uno de ellos al azar y se permite que adquiera el mutex. Como

los mutexes son tan simples, se pueden implementar con facilidad en espacio de usuario, siempre y cuando haya una instrucción TSL o XCHG disponible.

Monitores

Los monitores tienen una importante propiedad que los hace útiles para lograr la exclusión mutua: sólo puede haber un proceso activo en un monitor en cualquier instante. Los monitores son una construcción del lenguaje de programación, por lo que el compilador sabe que son especiales y puede manejar las llamadas a los procedimientos del monitor en forma distinta a las llamadas a otros procedimientos. Por lo general, cuando un proceso llama a un procedimiento de monitor, las primeras instrucciones del procedimiento comprobarán si hay algún otro proceso activo en un momento dado dentro del monitor. De ser así, el proceso invocador se suspenderá hasta que el otro proceso haya dejado el monitor. Si no hay otro proceso utilizando el monitor, el proceso invocador puede entrar.

Es responsabilidad del compilador implementar la exclusión mutua en las entradas del monitor, pero una forma común es utilizar un mutex o semáforo binario. Como el compilador (y no el programador) está haciendo los arreglos para la exclusión mutua, es mucho menos probable que al o salga mal. En cualquier caso, la persona que escribe el monitor no tiene que saber acerca de cómo el compilador hace los arreglos para la exclusión mutua. Basta con saber que, al convertir todas las regiones críticas en procedimientos de monitor, nunca habrá dos procesos que ejecuten sus regiones críticas al mismo tiempo.

Un problema con los monitores (y también con los semáforos) es que están diseñados para resolver el problema de exclusión mutua en una o más CPUs que tengan acceso a una memoria común. Al colocar los semáforos en la memoria compartida y protegerlos con instrucciones TSL o XCHG, podemos evitar las condiciones de carrera. Si utilizamos un sistema distribuido que consista en varias CPUs, cada una con su propia memoria privada, conectadas por una red de área local, estas primitivas no se pueden aplicar. La conclusión es que los semáforos son de un nivel demasiado bajo y los monitores no pueden usarse, excepto en algunos lenguajes de programación. Además, ninguna de las primitivas permite el intercambio de información entre las máquinas. Se necesita algo más (el pasaje de mensajes).

Pasaje de Mensajes

Este método de comunicación entre procesos utiliza dos primitivas (send y receive) que, al igual que los semáforos y a diferencia de los monitores, son llamadas al sistema en vez de construcciones del lenguaje. Como tales, se pueden colocar con facilidad en procedimientos de biblioteca.

La primera llamada envía un mensaje a un destino especificado y la segunda recibe un mensaje de un origen especificado (o de cualquiera, si al receptor no le importa). Si no hay un mensaje disponible, el receptor se puede bloquear hasta que llegue uno. De manera alternativa, puede regresar de inmediato con un código de error.

Barreras

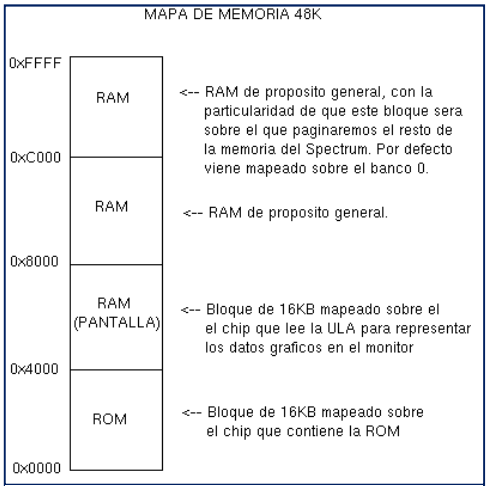
Gráfica 69. Barrera física. Fuente: http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/Barrier.jpg

Nuestro último mecanismo de sincronización está destinado a los grupos de procesos, en vez de las situaciones de tipo productor-consumidor de dos procesos. Algunas aplicaciones se dividen en fases y tienen la regla de que ningún proceso puede continuar a la siguiente fase sino hasta que todos los procesos estén listos para hacerlo. Para lograr este comportamiento, se coloca una barrera al final de cada fase. Cuando un proceso llega a la barrera, se bloquea hasta que todos los procesos han llegado a ella.

3.3. Algoritmos de planificación

Cuando una computadora se multiprograma, con frecuencia tiene varios procesos o hilos que compiten por la CPU al mismo tiempo. Esta situación ocurre cada vez que dos o más de estos procesos se encuentran al mismo tiempo en el estado listo. Si sólo hay una CPU disponible, hay que decidir cuál proceso se va a ejecutar a continuación. La parte del sistema operativo que realiza esa decisión se conoce como planificador de procesos y el algoritmo que utiliza se conoce como algoritmo de planificación.

En servidores en red, la situación cambia en forma considerable. Aquí varios procesos compiten a menudo por la CPU, por lo que la planificación retoma importancia. Por ejemplo, cuando la CPU tiene que elegir entre ejecutar un proceso que recopila las estadísticas diarias y uno que atiende las peticiones de los usuarios, habrá usuarios más contentos si el segundo tipo de procesos tiene prioridad sobre la CPU.



Gráfica 70. Mapa de Memoria. Fuente: http://magazinezx.speccy.org/17/img/mapa_memoria.png

Además de elegir el proceso correcto que se va a ejecutar a continuación, el planificador también tiene que preocuparse por hacer un uso eficiente de la CPU, debido a que la conmutación de procesos es cara. Para empezar, debe haber un cambio del modo de usuario al modo kernel. Después se debe guardar el estado del proceso actual, incluyendo el almacenamiento de sus registros en la tabla de procesos para que puedan volver a cargarse más adelante. En muchos sistemas, el mapa de memoria (por ejemplo, los bits de referencia de memoria en la tabla de páginas) se debe guardar también. Luego hay que seleccionar un nuevo proceso mediante la ejecución del algoritmo de planificación. Después de eso, se debe volver a cargar en la MMU el mapa de memoria del nuevo proceso.

Por último, se debe iniciar el nuevo proceso. Además de todo esto, generalmente la conmutación de procesos hace inválida toda la memoria caché, por lo que tiene que volver a cargarse en forma dinámica desde la memoria principal dos veces (al momento de entrar al kernel y al salir de éste). Con todo, si se realizan muchas conmutaciones de procesos por segundo, se puede llegar a consumir una cantidad considerable de tiempo de la CPU, por lo cual se aconseja tener precaución.

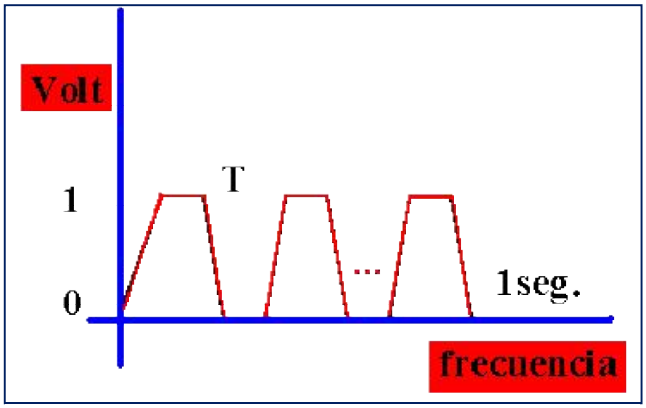
Cuándo planificar procesos

Una cuestión clave relacionada con la planificación es saber cuándo tomar decisiones de planificación. Resulta ser que hay una variedad de situaciones en las que se necesita la planificación.

- En primer lugar, cuando se crea un nuevo proceso se debe tomar una decisión en cuanto a si se debe ejecutar el proceso padre o el proceso hijo. Como ambos procesos se encuentran en el estado listo, es una decisión normal de programación y puede ejecutar cualquiera; es decir, el programador de procesos puede elegir ejecutar de manera legítima, ya sea el padre o el hijo.
- En segundo lugar, se debe tomar una decisión de planificación cuando un proceso termina. Ese proceso ya no se puede ejecutar (debido a que ya no existe), por lo que se debe elegir algún otro proceso del conjunto de procesos listos. Si no hay un proceso listo, por lo general se ejecuta un proceso inactivo suministrado por el sistema.
- En tercer lugar, cuando un proceso se bloquea por esperar una operación de E/S, un semáforo por alguna otra razón, hay que elegir otro proceso para ejecutarlo. Algunas veces la razón del bloqueo puede jugar un papel en la elección. Por ejemplo, si Aes un proceso importante y está esperando a que B salga de su región crítica, si dejamos que B se ejecute a continuación podrá salir de su región crítica y por ende, dejar que A continúe. Sin embargo, el problema es que el planificador comúnmente no tiene la información necesaria para tomar en cuenta esta dependencia.
- En cuarto lugar, cuando ocurre una interrupción de E/S tal vez haya que tomar una decisión de planificación. Si la interrupción proviene de un dispositivo de E/S que ha terminado su trabajo, tal vez ahora un proceso que haya estado bloqueado en espera de esa operación de E/S esté listo para ejecutarse. Es responsabilidad del planificador decidir si debe ejecutar el proceso que acaba de entrar al estado listo, el proceso que se estaba ejecutando al momento de la interrupción, o algún otro.

Si un reloj de hardware proporciona interrupciones periódicas a 50 o 60 Hz o cualquier otra frecuencia, se puede tomar una decisión de planificación en cada interrupción de reloj o en cada k-ésima interrupción de reloj. Los algoritmos de planificación se pueden dividir en dos categorías con respecto a la forma en que manejan las interrupciones del reloj. Un algoritmo de programación no apropiativo (nonpreemptive) selecciona un proceso para ejecutarlo y después sólo deja que se ejecute hasta que el mismo se bloquea (ya sea en espera de una operación de E/S o de algún otro proceso) o hasta que libera la CPU en forma voluntaria. Incluso aunque se ejecute durante horas, no se suspenderá de manera forzosa. En efecto, no se

toman decisiones de planificación durante las interrupciones de reloj. Una vez que se haya completado el procesamiento de la interrupción de reloj, se reanuda la ejecución del proceso que estaba antes de la interrupción, a menos que un proceso de mayor prioridad esté esperando por un tiempo libre que se acabe de cumplir.



Gráfica 71. Reloj de Interrupción. Fuente: http://lsi.vc.ehu.es/pablogn/docencia/manuales/SO

Por el contrario, un algoritmo de planificación apropiativa selecciona un proceso y deja que se ejecute por un máximo de tiempo fijo. Si sigue en ejecución al final del intervalo de tiempo, se suspende y el planificador selecciona otro proceso para ejecutarlo (si hay uno disponible). Para llevar a cabo la planificación apropiativa, es necesario que ocurra una interrupción de reloj al final del intervalo de tiempo para que la CPU regrese el control al planificador. Si no hay un reloj disponible, la planificación no apropiativa es la única opción.

Categorías de los algoritmos de planificación

No es sorprendente que distintos entornos requieran algoritmos de planificación diferentes. Esta situación se presenta debido a que las diferentes áreas de aplicación (y los distintos tipos de sistemas operativos) tienen diferentes objetivos. En otras palabras, lo que el planificador debe optimizar no es lo mismo en todos los sistemas. Tres de los entornos que vale la pena mencionar son:

- Procesamiento por lotes.
- Interactivo.
- De tiempo real.

Metas de los algoritmos de planificación

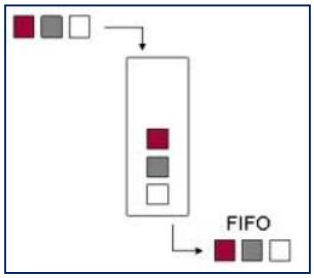
Para poder diseñar un algoritmo de programación, es necesario tener cierta idea de lo que debe hacer un buen algoritmo. Algunos objetivos dependen del entorno (procesamiento por lotes,

interactivo o de tiempo real), pero hay también algunos otros que son deseables en todos los casos.

- Todos los sistemas
  - Equidad - Otorgar a cada proceso una parte justa de la CPU
  - Aplicación de políticas - Verificar que se lleven a cabo las políticas establecidas
  - Balance - Mantener ocupadas todas las partes del sistema
- Sistemas de procesamiento por lotes
  - Rendimiento - Maximizar el número de trabajos por hora
  - Tiempo de retorno - Minimizar el tiempo entre la entrega y la terminación
  - Utilización de la CPU - Mantener ocupada la CPU todo el tiempo
- Sistemas interactivos
  - Tiempo de respuesta - Responder a las peticiones con rapidez
  - Proporcionalidad - Cumplir las expectativas de los usuarios
- Sistemas de tiempo real
  - Cumplir con los plazos - Evitar perder datos
  - Predictibilidad - Evitar la degradación de la calidad en los sistemas multimedia

Algoritmos de Planificación en sistemas de procesamiento por lotes

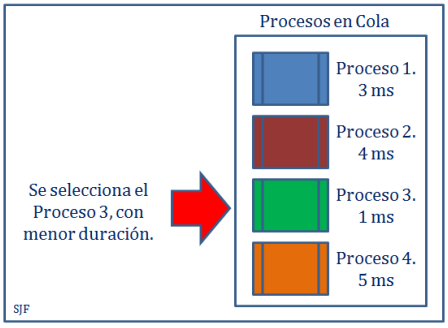
- **FIFO**: acrónimo de "*First in, first out*" (primero que entra, primero que sale). Con este algoritmo no apropiativo, la CPU se asigna a los procesos en el orden en el que la solicitan. En esencia hay una sola cola de procesos listos. Cuando el primer trabajo entra al sistema desde el exterior en la mañana, se inicia de inmediato y se le permite ejecutarse todo el tiempo que desee. No se interrumpe debido a que se ha ejecutado demasiado tiempo. A medida que van entrando otros trabajos, se colocan al final de la cola. Si el proceso en ejecución se bloquea, el primer proceso en la cola se ejecuta a continuación. Cuando un proceso bloqueado pasa al estado listo, al igual que un trabajo recién llegado, se coloca al final de la cola. La gran fuerza de este algoritmo es que es fácil de comprender e igualmente sencillo de programar. También es equitativo,



Gráfica 72. Método FIFO. Fuente:

en el mismo sentido en que es equitativo asignar los escasos boletos para eventos deportivos o conciertos a las personas que están dispuestas a permanecer en la línea desde las 2 a.m. Con este algoritmo, una sola lista ligada lleva la cuenta de todos los procesos listos. Para elegir un proceso a ejecutar sólo se requiere eliminar uno de la parte frontal de la cola. Para agregar un nuevo trabajo o desbloquear un proceso sólo hay que adjuntarlo a la parte final de la cola.

Por desgracia, este también tiene una importante desventaja. Suponga que hay un proceso ligado a los cálculos que se ejecuta durante 1 segundo en cierto momento, junto con muchos procesos limitados a E/S que utilizan poco tiempo de la CPU, pero cada uno de ellos tiene que realizar 1000 lecturas de disco para completarse. El proceso limitado a los cálculos se ejecuta durante 1 segundo y después lee un bloque de disco. Ahora se ejecutan todos los procesos de E/S e inician lecturas de disco. Cuando el proceso limitado a los cálculos obtiene su bloque de disco, se ejecuta por otro segundo, seguido de todos los procesos limitados a E/S en una rápida sucesión. El resultado neto es que cada proceso limitado a E/S llega a leer 1 bloque por segundo y requerirá 1000 segundos para completarse. Con un algoritmo de planificación que se apropió del proceso limitado a los cálculos cada 10 mseg, los procesos limitados a E/S terminarían en 10 segundos en vez de 1000 segundos y sin quitar mucha velocidad al proceso limitado a los cálculos.



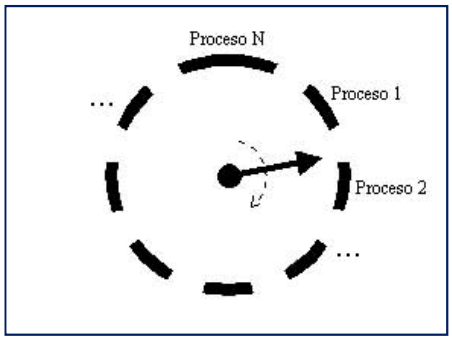
Gráfica 73. SJF. Elaboración propia.

- **SJF**: acrónimo de *Shortest Job First* (trabajo más corto primero) y algoritmo no apropiativo que supone que los tiempos de ejecución se conocen de antemano. Cuando hay varios trabajos de igual importancia esperando a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero
- **SRTN**: *Shortest Remaining Time Next* (menor tiempo restante a continuación). Algoritmo apropiativo donde el planificador siempre selecciona el proceso cuyo tiempo restante de ejecución sea el más corto. De nuevo, se debe conocer el tiempo de ejecución de antemano. Cuando llega un nuevo trabajo, su tiempo total se compara con el tiempo restante del proceso actual. Si el nuevo trabajo necesita menos tiempo para terminar

que el proceso actual, éste se suspende y el nuevo trabajo se inicia. Ese esquema permite que los trabajos cortos nuevos obtengan un buen servicio.

Algoritmos de Planificación en sistemas interactivos

- **Round Robin:** Uno de los algoritmos más antiguos, simples, equitativos y de mayor uso es el de turno circular (round-robin). A cada proceso se le asigna un intervalo de tiempo, conocido como quántum, durante el cual se le permite ejecutarse. Si el proceso se sigue ejecutando al final del cuanto, la CPU es apropiada para dársela a otro proceso. Si el proceso se bloquea o termina antes de que haya transcurrido el quántum, la conmutación de la CPU se realiza cuando el proceso se bloquea, desde luego. Es fácil implementar el algoritmo de turno circular.



Gráfica 74. Round Robin. Fuente: http://www-2.dc.uba.ar/proyinv/cso/obelix/ro_ro.jpg

  La conclusión se puede formular de la siguiente manera: si se establece el quántum demasiado corto se producen demasiadas conmutaciones de procesos y se reduce la eficiencia de la CPU, pero si se establece demasiado largo se puede producir una mala respuesta a las peticiones interactivas cortas. A menudo, un quántum con un valor entre 20 y 50 mseg constituye una solución razonable.
- **Por prioridad**: La planificación por turno circular hace la suposición implícita de que todos los procesos tienen igual importancia. Con frecuencia, las personas que poseen y operan computadoras multiusuario tienen diferentes ideas en cuanto a ese aspecto. Por ejemplo, en una universidad el orden jerárquico puede ser: primero los decanos, después los profesores, secretarias, y por último los estudiantes. La necesidad de tomar en cuenta los factores externos nos lleva a la planificación por prioridad. La idea básica es simple: a cada proceso se le asigna una prioridad y el proceso ejecutable con la prioridad más alta es el que se puede ejecutar.

  Incluso hasta en una PC con un solo propietario puede haber varios procesos, algunos de ellos más importantes que los demás. Por ejemplo, un proceso demonio que envía

correo electrónico en segundo plano debería recibir una menor prioridad que un proceso que muestra una película de video en la pantalla en tiempo real.

Para evitar que los procesos con alta prioridad se ejecuten de manera indefinida, el planificador puede reducir la prioridad del proceso actual en ejecución en cada pulso del reloj (es decir, en cada interrupción del reloj). Si esta acción hace que su prioridad se reduzca a un valor menor que la del proceso con la siguiente prioridad más alta, ocurre una conmutación de procesos. De manera alternativa, a cada proceso se le puede asignar un quántum de tiempo máximo que tiene permitido ejecutarse. Cuando este quántum se utiliza, el siguiente proceso con la prioridad más alta recibe la oportunidad de ejecutarse. A las prioridades se les pueden asignar procesos en forma estática o dinámica.

- **Múltiples colas**: Se han utilizado muchos otros algoritmos para asignar procesos a las clases de prioridades. Por ejemplo, el influyente sistema XDS 940 (Lampson, 1968) construido en Berkeley tenía cuatro clases de prioridad: terminal, E/S, quántum corto y quántum largo. Cuando un proceso que había estado esperando la entrada de terminal por fin se despertaba, pasaba a la clase de mayor prioridad (terminal). Cuando un proceso en espera de un bloque de disco pasaba al estado listo, se enviaba a la segunda clase. Cuando a un proceso que estaba todavía en ejecución se le agotaba su quántum, al principio se colocaba en la tercera clase. No obstante, si un proceso utilizaba todo su quántum demasiadas veces seguidas sin bloquearse en espera de la terminal o de otro tipo de E/S, se movía hacia abajo hasta la última cola. Muchos otros sistemas utilizan algo similar para favorecer a los usuarios y procesos interactivos en vez de los que se ejecutan en segundo plano.
- **El proceso más corto a continuación**: Como el algoritmo tipo el trabajo más corto primero siempre produce el tiempo de respuesta promedio mínimo para los sistemas de procesamiento por lotes, sería bueno si se pudiera utilizar para los procesos interactivos también. Hasta cierto grado, esto es posible. Por lo general, los procesos interactivos siguen el patrón de esperar un comando, ejecutarlo, esperar un comando, ejecutarlo, etcétera. Si consideramos la ejecución de cada comando como un “trabajo” separado, entonces podríamos minimizar el tiempo de respuesta total mediante la ejecución del

más corto primero. El único problema es averiguar cuál de los procesos actuales ejecutables es el más corto.

Un método es realizar estimaciones con base en el comportamiento anterior y ejecutar el proceso con el tiempo de ejecución estimado más corto. La técnica de estimar el siguiente valor en una serie mediante la obtención del promedio ponderado del valor actual medido y la estimación anterior se conoce algunas veces como envejecimiento. Se aplica en muchas situaciones en donde se debe realizar una predicción con base en valores anteriores.

- **Planificación garantizada**: Un método completamente distinto para la planificación es hacer promesas reales a los usuarios acerca del rendimiento y después cumplirlas. Una de ellas, que es realista y fácil de cumplir es: si hay n usuarios conectados mientras usted está trabajando, recibirá aproximadamente 1/n del poder de la CPU. De manera similar, en un sistema de un solo usuario con n procesos en ejecución, mientras no haya diferencias, cada usuario debe obtener 1/n de los ciclos de la CPU. Eso parece bastante justo.

  Para cumplir esta promesa, el sistema debe llevar la cuenta de cuánta potencia de CPU ha tenido cada proceso desde su creación. Después calcula cuánto poder de la CPU debe asignarse a cada proceso, a saber el tiempo desde que se creó dividido entre n. Como la cantidad de tiempo de CPU que ha tenido cada proceso también se conoce, es simple calcular la proporción de tiempo de CPU que se consumió con el tiempo de CPU al que cada proceso tiene derecho. Una proporción de 0.5 indica que un proceso solo ha tenido la mitad del tiempo que debería tener, y una proporción de 2.0 indica que un proceso ha tenido el doble de tiempo del que debería tener. Entonces, el algoritmo es para ejecutar el proceso con la menor proporción hasta que se haya desplazado por debajo de su competidor más cercano.

- **Planificación por sorteo**: Aunque hacer promesas a los usuarios y cumplirlas es una buena idea, es algo difícil de implementar. Sin embargo, se puede utilizar otro algoritmo para producir resultados similares con una implementación mucho más sencilla. Este algoritmo se conoce como planificación por sorteo (Waldspurger y Weihl, 1994).

  La idea básica es dar a los procesos boletos de lotería para diversos recursos del sistema, como el tiempo de la CPU. Cada vez que hay que tomar una decisión de

planificación, se selecciona un boleto de lotería al azar y el proceso que tiene ese boleto obtiene el recurso. Cuando se aplica a la planificación de la CPU, el sistema podría realizar un sorteo 50 veces por segundo y cada ganador obtendría 20 mseg de tiempo de la CPU como premio. Los procesos más importantes pueden recibir boletos adicionales, para incrementar su probabilidad de ganar.

- **Planificación por partes equitativas**: Hasta ahora hemos asumido que cada proceso se planifica por su cuenta, sin importar quién sea su propietario. Como resultado, si el usuario 1 inicia 9 procesos y el usuario 2 inicia 1 proceso, con la planificación por turno circular o por prioridades iguales, el usuario 1 obtendrá 90 por ciento del tiempo de la CPU y el usuario 2 sólo recibirá 10 por ciento. Para evitar esta situación, algunos sistemas toman en consideración quién es el propietario de un proceso antes de planificarlo. En este modelo, a cada usuario se le asigna cierta fracción de la CPU y el planificador selecciona procesos de tal forma que se cumpla con este modelo. Por ende, si a dos usuarios se les prometió 50 por ciento del tiempo de la CPU para cada uno, eso es lo que obtendrán sin importar cuántos procesos tengan en existencia.

Algoritmos de Planificación en sistemas de tiempo real

Gráfica 75. Sala de Terapia Intensiva. Fuente: http://static.diariomedico.com/images/2010/10/19/9_1.jpg

En un sistema de tiempo real, el tiempo desempeña un papel esencial. Por lo general, uno o más dispositivos físicos externos a la computadora generan estímulo y la computadora debe reaccionar de manera apropiada a ellos dentro de cierta cantidad fija de tiempo. Por ejemplo, la computadora en un reproductor de disco compacto recibe los bits a medida que provienen de la unidad y debe convertirlos en música, en un intervalo de tiempo muy estrecho. Si el cálculo tarda demasiado, la música tendrá un sonido peculiar. Otros sistemas de tiempo real son el monitoreo de pacientes en una unidad de cuidados intensivos de un hospital, el autopiloto en una aeronave y el control de robots en una fábrica automatizada. En todos estos casos, tener la respuesta correcta pero demasiado tarde es a menudo tan malo como no tenerla.

En general, los sistemas de tiempo real se categorizan como de tiempo real duro, lo cual significa que hay tiempos límite absolutos que se deben cumplir, y como de tiempo real suave, lo cual significa que no es conveniente fallar en un tiempo límite en ocasiones, pero sin embargo es tolerable. En ambos casos, el comportamiento en tiempo real se logra dividiendo el programa en varios procesos, donde el comportamiento de cada uno de éstos es predecible y se conoce de antemano. Por lo general, estos procesos tienen tiempos de vida cortos y pueden ejecutarse hasta completarse en mucho menos de 1 segundo. Cuando se detecta un evento externo, es responsabilidad del planificador planificar los procesos de tal forma que se cumpla con todos los tiempos límite.

Los algoritmos de planificación en tiempo real pueden ser estáticos o dinámicos. Los primeros toman sus decisiones de planificación antes de que el sistema empiece a ejecutarse. Los segundos lo hacen durante el tiempo de ejecución. La planificación estática sólo funciona cuando hay información perfecta disponible de antemano acerca del trabajo que se va a realizar y los tiempos límite que se tienen que cumplir. Los algoritmos de planificación dinámicos no tienen estas restricciones.

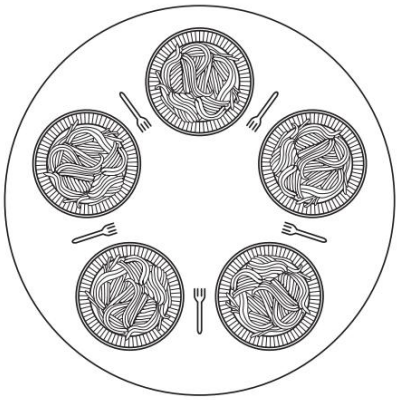
Problemas clásicos de Comunicaciones

- **El problema de los filósofos comelones**

  En 1965, Dijkstra propuso y resolvió un problema de sincronización al que llamó el problema de los filósofos comelones. Desde ese momento, todos los que inventaban otra primitiva de sincronización se sentían obligados a demostrar qué tan maravillosa era esa nueva primitiva, al mostrar con qué elegancia resolvía el problema de los filósofos comelones. Este problema se puede enunciar simplemente de la siguiente manera. Cinco filósofos están sentados alrededor de una mesa circular. Cada filósofo tiene un plato de espagueti. El espagueti es tan resbaloso, que un filósofo necesita dos tenedores para comerlo. Entre cada par de platos hay un tenedor. La distribución de la mesa se ilustra en la gráfica 76.

  La vida de un filósofo consiste en periodos alternos de comer y pensar (esto es algo así como una abstracción, incluso para los filósofos, pero las otras actividades son irrelevantes aquí). Cuando un filósofo tiene hambre, trata de adquirir sus tenedores

izquierdo y derecho, uno a la vez, en cualquier orden. Si tiene éxito al adquirir dos tenedores, come por un momento, después deja los tenedores y continúa pensando. La pregunta clave es: ¿puede usted escribir un programa para cada filósofo, que haga lo que se supone debe hacer y nunca se trabe? (Hemos recalcado que el requerimiento de los dos tenedores es algo artificial; tal vez deberíamos cambiar de comida italiana a comida china y sustituir el espagueti por arroz y los tenedores por palillos chinos).



Gráfica 76. Filósofos comelones. Fuente: Tanembaum (2009).

La solución obvia sería esperar hasta que el tenedor específico esté disponible y luego lo toma. Por desgracia, la solución obvia está mal. Suponga que los cinco filósofos toman sus tenedores izquierdos al mismo tiempo. Ninguno podrá tomar sus tenedores derechos y habrá un interbloqueo.

Ahora podríamos pensar que si los filósofos sólo esperan por un tiempo aleatorio en vez de esperar durante el mismo tiempo al no poder adquirir el tenedor derecho, la probabilidad de que todo continúe bloqueado durante incluso una hora es muy pequeña. Esta observación es verdad y en casi todas las aplicaciones intentar de nuevo en un tiempo posterior no representa un problema. Por ejemplo, en la popular red de área local Ethernet, si dos computadoras envían un paquete al mismo tiempo, cada una espera durante un tiempo aleatorio e intenta de nuevo; en la práctica esta solución funciona bien.

Una mejora que no tiene interbloqueo ni inanición es proteger las cinco instrucciones cada una con un semáforo binario. Antes de empezar a adquirir tenedores, un filósofo realizaría una operación down en mutex. Después de regresar los tenedores, realizaría una operación up en mutex. Desde un punto de vista teórico, esta solución es adecuada. Desde un punto de vista práctico, tiene un error de rendimiento: sólo puede haber un filósofo comiendo en cualquier instante. Con cinco tenedores disponibles, deberíamos poder permitir que dos filósofos coman al mismo tiempo.

Se podría proponer otra solución, usando un arreglo de semáforos, uno por cada filósofo, de manera que los filósofos hambrientos puedan bloquearse si los tenedores que necesitan están ocupados.

- **El problema de los lectores y escritores**

El problema de los filósofos comelones es útil para modelar procesos que compiten por el acceso exclusivo a un número limitado de recursos, como los dispositivos de E/S. Otro problema famoso es el de los lectores y escritores

(Courtois y colaboradores, 1971), que modela el acceso a una base de datos. Por ejemplo, imagine un sistema de reservación de aerolíneas, con muchos procesos en competencia que desean leer y escribir en él. Es aceptable tener varios procesos que lean la base de datos al mismo tiempo, pero si un proceso está actualizando (escribiendo) la base de datos, ningún otro proceso puede tener acceso a la base de datos, ni siquiera los lectores. La pregunta es, ¿cómo se programan los lectores y escritores?

Suponga que mientras un lector utiliza la base de datos, llega otro lector. Como no es un problema tener dos lectores al mismo tiempo, el segundo lector es admitido. También se pueden admitir más lectores, si es que llegan. Ahora suponga que aparece un escritor. Tal vez éste no sea admitido a la base de datos, ya que los escritores deben tener acceso exclusivo y por ende, el escritor se suspende. Más adelante aparecen lectores adicionales. Mientras que haya un lector activo, se admitirán los siguientes lectores. Como consecuencia de esta estrategia, mientras que haya un suministro continuo de lectores, todos entrarán tan pronto lleguen. El escritor estará suspendido hasta que no haya un lector presente. Si llega un nuevo lector, por decir cada 2 segundos y cada lector requiere 5 segundos para hacer su trabajo, el escritor nunca entrará. Para evitar esta situación, el programa se podría escribir de una manera ligeramente distinta: cuando llega un lector y hay un escritor en espera, el lector se suspende detrás del escritor, en vez de ser admitido de inmediato. De esta forma, un escritor tiene que esperar a que terminen los lectores que estaban activos cuando llegó, pero no tiene que esperar a los

lectores que llegaron después de él. La desventaja de esta solución es que logra una menor concurrencia y por ende, un menor rendimiento. Courtois y sus colaboradores presentan una solución que da prioridad a los escritores.

3.4. Interrupciones

Concepto[7]

En el contexto de la informática, una interrupción (del inglés *Interrupt Request*, también conocida como petición de interrupción) es una señal recibida por el procesador de un ordenador, indicando que debe "interrumpir" el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.

Una interrupción es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción, la cual, por lo general, no forma parte del programa, sino que pertenece al sistema operativo o al BIOS). Una vez finalizada dicha subrutina, se reanuda la ejecución del programa.

Las interrupciones surgen de la necesidad que tienen los dispositivos periféricos de enviar información al procesador principal de un sistema informático.

La primera técnica que se empleó para esto fue el *polling*, que consistía en que el propio procesador se encargara de sondear los dispositivos periféricos cada cierto tiempo para averiguar si tenía pendiente alguna comunicación para él. Este método presentaba el inconveniente de ser muy ineficiente, ya que el procesador consumía constantemente tiempo y recursos en realizar estas instrucciones de sondeo.

El mecanismo de interrupciones fue la solución que permitió al procesador desentenderse de esta problemática, y delegar en el dispositivo periférico la responsabilidad de comunicarse con él cuando lo necesitara. El procesador, en este caso, no sondea a ningún dispositivo, sino que queda a la espera de que estos le avisen (le "interrumpan") cuando tengan algo que comunicarle (ya sea un evento, una transferencia de información, una condición de error, etc.).

Procesamiento

- Terminar la ejecución de la instrucción máquina en curso.

- Salvar el valor del contador de programa, IP, en la pila, de manera que en la CPU, al terminar el proceso, pueda seguir ejecutando el programa a partir de la última instrucción.
- La CPU salta a la dirección donde está almacenada la rutina de servicio de interrupción (*Interrupt Service Routine*, o abreviado ISR) y ejecuta esa rutina que tiene como objetivo atender al dispositivo que generó la interrupción.
- Una vez que la rutina de la interrupción termina, el procesador restaura el estado que había guardado en la pila en el paso 2 y retorna al programa que se estaba usando anteriormente.

Clases

- **Interrupciones de hardware.**

  Estas son asíncronas a la ejecución del procesador, es decir, se pueden producir en cualquier momento independientemente de lo que esté haciendo el CPU en ese momento. Las causas que las producen son externas al procesador y a menudo suelen estar ligadas con los distintos dispositivos de E/S.

  Las interrupciones de hardware son aquellas interrupciones que se producen como resultado de, por lo general, una operación de E/S. No son producidas por ninguna instrucción de un programa sino por las señales que emiten los dispositivos periféricos para indicarle al procesador que necesitan ser atendidos.

  Cuando el microprocesador accede a un periférico (disco duro, puerto de comunicación...), puede transcurrir algún tiempo antes de que los datos sean obtenidos o transmitidos. La solución más simple es esperar hasta recibir los datos o hasta que se haya efectuado la transmisión (*polling*), pero esta solución bloquea todos los programas en ejecución, y eso no puede admitirse en un sistema multitarea. Por ello, en los sistemas modernos se prefiere un funcionamiento mediante interrupciones, ya que éstas permiten mejorar la productividad del procesador, de forma que este último puede ordenar una operación de E/S y, en lugar de tener que realizar una espera activa, se puede dedicar a atender a otro proceso o aplicación hasta que el dispositivo esté de nuevo disponible, siendo dicho dispositivo el encargado de notificar al procesador

mediante la línea de interrupción que ya está preparado para continuar/terminar la operación de E/S.

- **Excepciones.**

  Son aquellas que se producen de forma síncrona a la ejecución del procesador y por tanto podrían predecirse si se analiza con detenimiento la traza del programa que en ese momento estaba siendo ejecutado en la CPU. Normalmente son causadas al realizarse operaciones no permitidas tales como la división entre 0, el desbordamiento, el acceso a una posición de memoria no permitida, etc.

  Las excepciones son un tipo de interrupción sincrónica típicamente causada por una condición de error en un programa, como por ejemplo una división entre 0 o un acceso inválido a memoria en un proceso de usuario. Normalmente genera un cambio de contexto a modo supervisor para que el sistema operativo atienda el error. Así pues, las excepciones son un mecanismo de protección que permite garantizar la integridad de los datos almacenados tanto en el espacio de usuario como en el espacio kernel. Cuando el Sistema Operativo detecta una excepción intenta solucionarla, pero en caso de no poder simplemente notificará la condición de error a la aplicación/usuario y abortará la misma.

- **Interrupciones por software.**

  Las interrupciones por software son aquellas generadas por un programa en ejecución. Para generarlas, existen distintas instrucciones en el código máquina que permiten al programador producir una interrupción, las cuales suelen tener nemotécnicos tales como INT (por ejemplo, en DOS se realiza la instrucción INT 0x21 y en Unix se utiliza INT 0x80 para hacer llamadas de sistema).

  La interrupción por software, también denominadas llamadas al sistema, son aquellas generadas por un programa mientras este está ejecutándose. En general, actúan de la siguiente manera:

  - Un programa que se venía ejecutando luego de su instrucción I5 , llama al Sistema Operativo, por ejemplo para leer un archivo de disco (cuando un programa necesita un dato exterior , se detiene y pasa a cumplir con las tareas de recoger ese dato).

- A tal efecto, luego de I5 existe en el programa, la instrucción de código de máquina CD21, simbolizada INT 21 en Assembler, que realiza el requerimiento del paso 1. Puesto que no puede seguir le ejecución de la instrucción I6 y siguientes del programa hasta que no se haya leído el disco y esté en memoria principal dicho archivo, virtualmente el programa se ha interrumpido, siendo, además, que luego de INT 21, las instrucciones que se ejecutarán no serán del programa, sino del Sistema Operativo. Se detiene el programa y ordena en este caso mediante INT21 (interrupción predefinida) que recoge el dato solicitado, para poder seguir el programa que la ordenó).
- La ejecución de INT 21 permite hallar la subrutina del Sistema Operativo.
- Se ejecuta la subrutina del Sistema Operativo que prepara la lectura del disco.
- Luego de ejecutarse la subrutina del Sistema Operativo, y una vez que se haya leído el disco y verificado que la lectura es correcta, el Sistema Operativo ordenará reanudar la ejecución del programa autointerrumpido en espera.
- La ejecución del programa se reanuda.

3.5. Complemento: Procesadores

El "cerebro" de la computadora es la CPU, que obtiene las instrucciones de la memoria y las ejecuta. El ciclo básico de toda CPU es obtener la primera instrucción de memoria, decodificarla para determinar su tipo y operandos, ejecutarla y después obtener, decodificar y ejecutar las instrucciones subsiguientes. El ciclo se repite hasta que el programa termina. De esta forma se ejecutan los programas.

Gráfica 77. Procesador Sparc T5. Fuente: http://aodbc.files.wordpress.com/2013/04/sparc_t5_chip.jpg

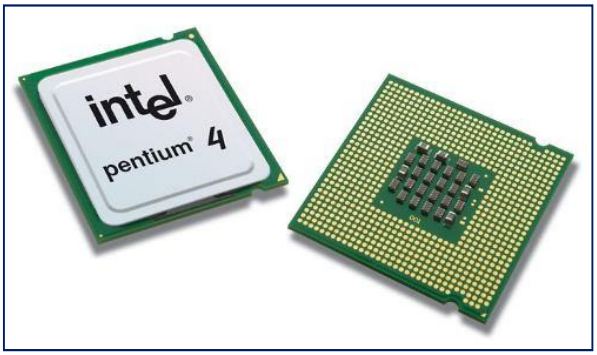
Cada CPU tiene un conjunto específico de instrucciones que puede ejecutar. Así, un Pentium no puede ejecutar programas de SPARC y un SPARC no puede ejecutar programas de Pentium. Como el acceso a la memoria para obtener una instrucción o palabra de datos requiere mucho más tiempo que ejecutar una instrucción, todas las CPU contienen ciertos registros en su interior para contener las variables clave y los resultados temporales. Debido a esto, el conjunto de instrucciones generalmente contiene instrucciones para cargar una palabra

de memoria en un registro y almacenar una palabra de un registro en la memoria. Otras instrucciones combinan dos operandos de los registros, la memoria o ambos en un solo resultado, como la operación de sumar dos palabras y almacenar el resultado en un registro o la memoria.

Además de los registros generales utilizados para contener variables y resultados temporales, la mayoría de las computadoras tienen varios registros especiales que están visibles para el programador. Uno de ellos es el contador de programa (program counter), el cual contiene la dirección de memoria de la siguiente instrucción a obtener. Una vez que se obtiene esa instrucción, el contador de programa se actualiza para apuntar a la siguiente.

Otro registro es el apuntador de pila (*stack pointer*), el cual apunta a la parte superior de la pila (*stack*) actual en la memoria. La pila contiene un conjunto de valores por cada procedimiento al que se ha entrado pero del que todavía no se ha salido. El conjunto de valores en la pila por procedimiento contiene los parámetros de entrada, las variables locales y las variables temporales que no se mantienen en los registros.

Otro de los registros es PSW (*Program Status Word*; Palabra de estado del programa). Este registro contiene los bits de código de condición, que se asignan cada vez que se ejecutan las instrucciones de comparación, la prioridad de la CPU, el modo (usuario o kernel) y varios otros bits de control. Los programas de usuario pueden leer normalmente todo el PSW pero por lo general sólo pueden escribir en algunos de sus campos. El PSW juega un papel importante en las llamadas al sistema y en las operaciones de E/S.



Gráfica 78. Procesador Intel Pentium 4. Fuente: lc.fie.umich.mx

El sistema operativo debe estar al tanto de todos los registros. Cuando la CPU se multiplexa en el tiempo, el sistema operativo detiene con frecuencia el programa en ejecución para (re)iniciar otro. Cada vez que detiene un programa en ejecución, el sistema operativo debe guardar todos los registros para poder restaurarlos cuando el programa continúe su ejecución.

Para mejorar el rendimiento, los diseñadores de CPUs abandonaron desde hace mucho tiempo el modelo de obtener, decodificar y ejecutar una instrucción a la vez. Muchas CPUs modernas cuentan con medios para ejecutar más de una instrucción al mismo tiempo. Por ejemplo, una CPU podría tener unidades separadas de obtención, decodificación y ejecución, de manera que mientras se encuentra ejecutando la instrucción n, también podría estar decodificando la instrucción n1 y obteniendo la instrucción n2. A dicha organización se le conoce como canalización (pipeline). El uso de canalizaciones más grandes es común. En la mayoría de los diseños de canalizaciones, una vez que se ha obtenido una instrucción y se coloca en la canalización, se debe ejecutar aún si la instrucción anterior era una bifurcación condicional que se tomó. Las canalizaciones producen grandes dolores de cabeza a los programadores de compiladores y de sistemas operativos, ya que quedan al descubierto las complejidades de la máquina subyacente.

La mayoría de las CPU, con excepción de las extremadamente simples que se utilizan en los sistemas integrados, tienen dos modos: modo kernel y modo usuario, como dijimos antes. Por lo general, un bit en el PSW controla el modo. Al operar en modo kernel, la CPU puede ejecutar cualquier instrucción de su conjunto de instrucciones y utilizar todas las características del hardware. El sistema operativo opera en modo kernel, lo cual le da acceso al hardware completo.

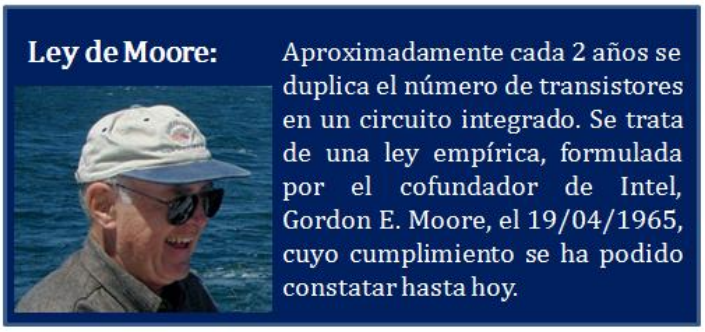
En contraste, los programas de usuario operan en modo de usuario, el cual les permite ejecutar sólo un subconjunto de las instrucciones y les da acceso sólo a un subconjunto de las características. En general, no se permiten las instrucciones que implican operaciones de E/S y protección de la memoria en el modo usuario. Desde luego que también está prohibido asignar el bit de modo del PSW para entrar al modo kernel.

Para obtener servicios del sistema operativo, un programa usuario debe lanzar una llamada al sistema (*system call*), la cual se atrapa en el kernel e invoca al sistema operativo. La instrucción TRAP cambia del modo usuario al modo kernel e inicia el sistema operativo. Cuando se ha completado el trabajo, el control se devuelve al programa de usuario en la instrucción que va después de la llamada al sistema. Más adelante en este capítulo explicaremos los detalles acerca del mecanismo de llamadas al sistema, pero por ahora piense en él como un tipo especial de

instrucción de llamada a procedimiento que tiene la propiedad adicional de cambiar del modo usuario al modo kernel.

Vale la pena indicar que las computadoras tienen otros traps aparte de la instrucción para ejecutar una llamada al sistema. La mayoría de los demás traps son producidos por el hardware para advertir acerca de una situación excepcional, tal como un intento de dividir entre 0 o un subdesbordamiento de punto flotante. En cualquier caso, el sistema operativo obtiene el control y debe decidir qué hacer. Algunas veces el programa debe terminarse con un error. Otras veces el error se puede ignorar (un número que provoque un subdesbordamiento puede establecerse en 0). Por último, cuando el programa anuncia por adelantado que desea manejar ciertos tipos de condiciones, puede devolvérsele el control para dejarlo resolver el problema.

Chips con multihilamiento y multinúcleo

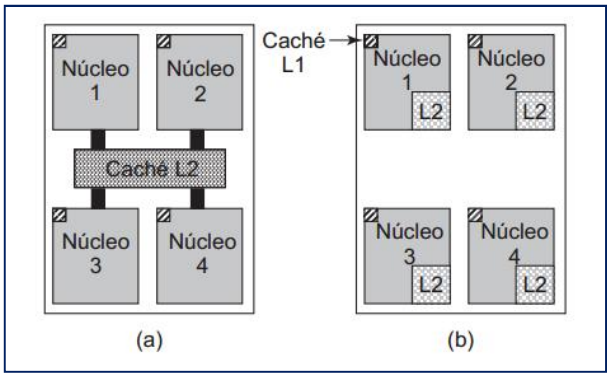


La ley de Moore establece que el número de transistores en un chip se duplica cada 18 meses. Esta "ley" no es ningún tipo de ley de física, como la de la conservación del momento, sino una observación hecha por Gordon Moore, cofundador de Intel, acerca de la velocidad con la que los ingenieros de procesos en las compañías de semiconductores pueden reducir sus transistores. La ley de Moore ha estado vigente durante tres décadas hasta hoy y se espera que siga así durante al menos una década más.

La abundancia de transistores está ocasionando un problema: ¿qué se debe hacer con todos ellos? En párrafos anteriores vimos una solución: las arquitecturas superescalares, con múltiples unidades funcionales. Pero a medida que se incrementa el número de transistores, se puede hacer todavía más. Algo obvio por hacer es colocar cachés más grandes en el chip de la CPU y eso está ocurriendo, pero en cierto momento se llega al punto de rendimiento decreciente.

El siguiente paso obvio es multiplicar no sólo las unidades funcionales, sino también parte de la lógica de control. El Pentium 4 y algunos otros chips de CPU tienen esta propiedad, conocida como multihilamiento (*multithreading*) o hiperhilamiento (*hyperthreading*) (el nombre que puso Intel al multihilamiento). Para una primera aproximación, lo que hace es permitir que la CPU contenga el estado de dos hilos de ejecución (*threads*) distintos y luego alterne entre uno y otro con una escala de tiempo en nanosegundos (un hilo de ejecución es algo así como un proceso ligero, que a su vez es un programa en ejecución). Por ejemplo, si uno de los procesos necesita leer una palabra de memoria (que requiere muchos ciclos de reloj), una CPU con multihilamiento puede cambiar a otro hilo. El multihilamiento no ofrece un verdadero paralelismo. Sólo hay un proceso en ejecución a la vez, pero el tiempo de cambio entre un hilo y otro se reduce al orden de un nanosegundo.

El multihilamiento tiene consecuencias para el sistema operativo, debido a que cada hilo aparece para el sistema operativo como una CPU separada. Considere un sistema con dos CPU reales, cada una con dos hilos. El sistema operativo verá esto como si hubiera cuatro CPU. Si hay suficiente trabajo sólo para mantener ocupadas dos CPU en cierto punto en el tiempo, podría planificar de manera inadvertida dos hilos en la misma CPU, mientras que la otra CPU estaría completamente inactiva. Esta elección es mucho menos eficiente que utilizar un hilo en cada CPU. El sucesor del Pentium 4, conocido como arquitectura Core (y también Core 2, Dual Core), no tiene hiperhilamiento, pero Intel ha anunciado que el sucesor del Core lo tendrá nuevamente.



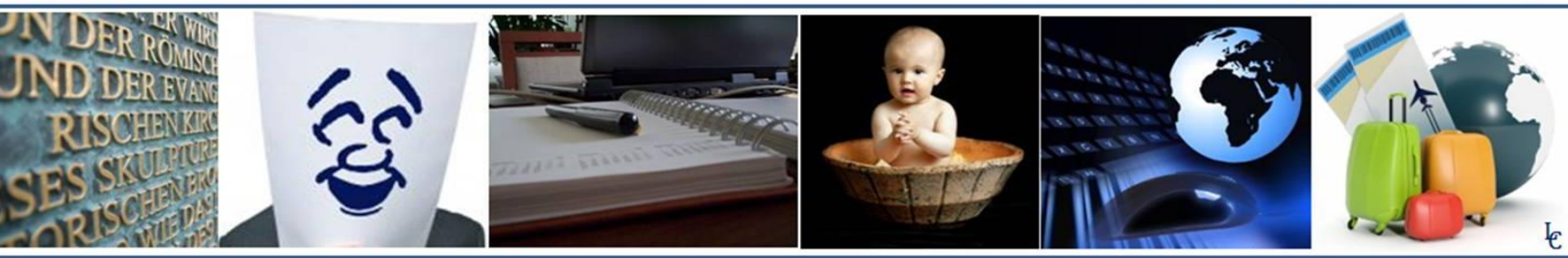
Gráfica 79. (a) Un chip de cuatro núcleos (quad-core) con una caché L2 compartida. (b) Un chip de cuatro núcleos con cachés L2 separadas. Fuente: Tanembaum (2009)

Más allá del multihilamiento, tenemos chips de CPU con dos, cuatro o más procesadores completos, o núcleos (cores) en su interior. Los chips de multinúcleo (multicore) de la figura 79 contienen efectivamente cuatro minichips en su interior, cada uno con su propia CPU independiente (más adelante hablaremos sobre las cachés). Para hacer uso de dicho chip multinúcleo se requiere en definitiva un sistema operativo multiprocesador.

Intel Core 2

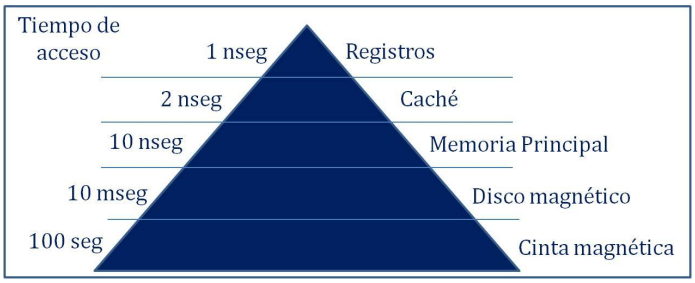
Gráfica 80. Intel Core 2 Duo. Fuente: www.notebookcheck.org

La marca Intel Core 2 se refiere a una gama de CPU comerciales de Intel de 64 bits de doble núcleo y CPU 2x2 MCM (Módulo Multi-Chip) de cuatro núcleos con el conjunto de instrucciones x86-64, basado en el *Core microarchitecture* de Intel, derivado del procesador portátil de doble núcleo de 32 bits Yonah. El CPU 2x2 MCM de cuatro núcleos tenía dos matrices separadas de dos núcleos (CPU) -uno junto al otro- en un paquete MCM de cuatro núcleos. El Core 2 relegó la marca Pentium a un mercado de gama medio-bajo, y reunificó las líneas de sobremesa y portátiles, las cuales previamente habían sido divididas en las marcas Pentium 4, D, y M[8].



4. Administración de Memoria

4.1. Memoria. Concepto y Tipos.



Gráfica 81. Jerarquía de Memoria. Fuente: elaboración propia.

El segundo componente importante en cualquier computadora, luego del procesador, es la memoria. En teoría, una memoria debe ser en extremo rápida (más rápida que la velocidad de ejecución de una instrucción, de manera que la memoria no detenga a la CPU), de gran tamaño y muy económica. Ninguna tecnología en la actualidad cumple con todos estos objetivos, por lo que se adopta una solución distinta. El sistema de memoria está construido como una jerarquía de capas, como se muestra en la gráfica 81. Las capas superiores tienen mayor velocidad, menor capacidad y mayor costo por bit que las capas inferiores, a menudo por factores de mil millones o más.

Registros

La capa superior consiste en los registros internos de la CPU. Están compuestos del mismo material que la CPU y, por ende, tienen la misma rapidez. En consecuencia no hay retraso a la hora de utilizarlos. La capacidad de almacenamiento disponible en estos registros es generalmente de 32 32 bits en una CPU de 32 bits y de 64 64 bits en una CPU de 64 bits. Menos de 1 KB en ambos casos. Los programas deben administrar los registros (es decir, decidir qué deben guardar en ellos) por su cuenta, en el software.

Caché

El siguiente nivel es la memoria caché, que el hardware controla de manera parcial. La memoria principal se divide en líneas de caché, que por lo general son de 64 bytes, con direcciones de 0 a 63 en la línea de caché 0, direcciones de 64 a 127 en la línea de caché 1 y así sucesivamente. Las líneas de caché que se utilizan con más frecuencia se mantienen en una caché de alta velocidad, ubicada dentro o muy cerca de la CPU. Cuando el programa necesita leer una palabra de memoria, el hardware de la caché comprueba si la línea que se requiere se encuentra en la caché. Si es así (a lo cual se le conoce como acierto de caché), la petición de la caché se cumple y no se envía una petición de memoria a través del bus hacia la memoria principal. Los aciertos de caché por lo general requieren un tiempo aproximado de dos ciclos de reloj. Los fallos de caché tienen que ir a memoria, con un castigo considerable de tiempo. La memoria caché está limitada en tamaño debido a su alto costo. Algunas máquinas tienen dos o incluso tres niveles de caché, cada una más lenta y más grande que la anterior.

El uso de cachés juega un papel importante en muchas áreas de las ciencias computacionales, no sólo en la caché de las líneas de RAM. Cada vez que hay un recurso extenso que se puede dividir en piezas, algunas de las cuales se utilizan con mucho más frecuencia que otras, a menudo se invoca a la caché para mejorar el rendimiento. Los sistemas operativos la utilizan todo el tiempo.

Por ejemplo, la mayoría de los sistemas operativos mantienen (piezas de) los archivos que se utilizan con frecuencia en la memoria principal para evitar tener que obtenerlos del disco en forma repetida. De manera similar, los resultados de convertir nombres de rutas extensas tales como

/home/ast/proyectos/minix3/src/kernel/reloj.c

a la dirección en disco donde se encuentra el archivo, se pueden colocar en la caché para evitar búsquedas repetidas. Por último, cuando una dirección de una página Web (URL) se convierte en una dirección de red (dirección IP), el resultado se puede poner en la caché para usarlo a futuro (existen muchos otros usos).

En cualquier sistema de caché surgen con rapidez varias preguntas, incluyendo:

- Cuándo se debe poner un nuevo elemento en la caché.
- En qué línea de caché se debe poner el nuevo elemento.
- Qué elemento se debe eliminar de la caché cuando se necesita una posición.
- Dónde se debe poner un elemento recién desalojado en la memoria de mayor tamaño.

No todas las preguntas son relevantes para cada situación de uso de la caché. Para poner líneas de la memoria principal en la caché de la CPU, por lo general se introduce un nuevo elemento en cada fallo de caché. La línea de caché a utilizar se calcula generalmente mediante el uso de algunos de los bits de mayor orden de la dirección de memoria a la que se hace referencia. Por ejemplo, con 4096 líneas de caché de 64 bytes y direcciones de 32 bits, los bits del 6 al 17 podrían utilizarse para especificar la línea de caché, siendo los bits del 0 al 5 el byte dentro de la línea de la caché. En este caso, el elemento a quitar es el mismo en el que se colocan los nuevos datos, pero en otros sistemas podría ser otro. Por último, cuando se vuelve a escribir una línea de caché en la memoria principal (si se ha modificado desde la última vez que se puso en caché), la posición en memoria donde se debe volver a escribir se determina únicamente por la dirección en cuestión.

Gráfica 82. Memoria Caché en Intel y AMD. Fuente: http://pcexpertos.com

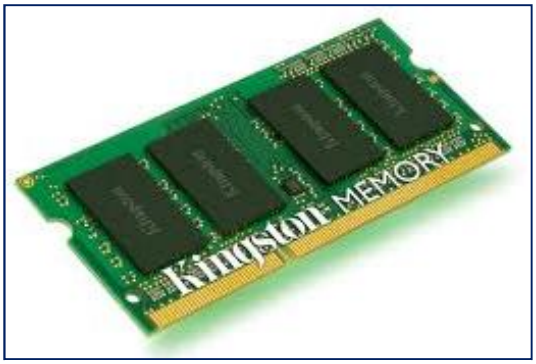
Las cachés son una idea tan útil que las CPUs modernas tienen dos de ellas. La caché L1 o de primer nivel está siempre dentro de la CPU, y por lo general alimenta las instrucciones decodificadas al motor de ejecución de la CPU. La mayoría de los chips tienen una segunda caché L1 para las palabras de datos que se utilizan con frecuencia. Por lo general, las cachés L1 son de 16 KB cada una. Además, a menudo hay una segunda caché, conocida como caché L2, que contiene varios megabytes de palabras de memoria utilizadas recientemente. La diferencia entre las cachés L1 y L2 está en la velocidad. El acceso a la caché L1 se realiza sin ningún retraso, mientras que el acceso a la caché L2 requiere un retraso de uno o dos ciclos de reloj.

En los chips multinúcleo, los diseñadores deben decidir dónde deben colocar las cachés. En la gráfica 79 (a) hay una sola caché L2 compartida por todos los núcleos; esta metodología se

utiliza en los chips multinúcleo de Intel. En contraste, en la gráfica 79 (b) cada núcleo tiene su propia caché L2; AMD utiliza esta metodología. Cada estrategia tiene sus pros y sus contras. Por ejemplo, la caché L2 compartida de Intel requiere un dispositivo controlador de caché más complicado, pero la manera en que AMD utiliza la caché hace más difícil la labor de mantener las cachés L2 consistentes.

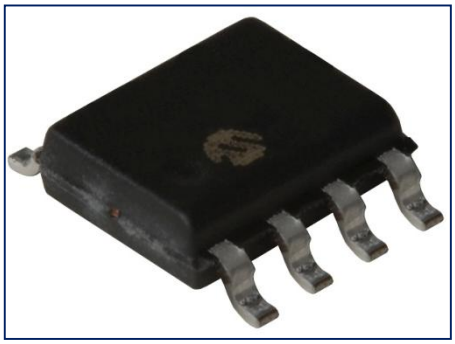
Memoria Principal o RAM

La memoria principal viene a continuación en la jerarquía de la gráfica 81. Es el "caballo de batalla" del sistema de memoria. Por lo general a la memoria principal se le conoce como RAM (*Random Access Memory*, Memoria de Acceso Aleatorio). Los usuarios de computadora antiguos algunas veces la llaman memoria de núcleo debido a que las computadoras en las décadas de 1950 y 1960 utilizaban pequeños núcleos de ferrita magnetizables para la memoria principal. En la actualidad, las memorias contienen desde cientos de megabytes hasta varios gigabytes y su tamaño aumenta con rapidez. Todas las peticiones de la CPU que no se puedan satisfacer desde la caché pasan a la memoria principal.



Gráfica 83. Memoria RAM DIMM DDR3 de 4 Gb. Fuente: http://www.irbit.com.ar/store/images/high_1925251-Kingston.jpg

Además de la memoria principal, muchas computadoras tienen una pequeña cantidad de memoria de acceso aleatorio no volátil. A diferencia de la RAM, la memoria no volátil no pierde su contenido cuando se desconecta la energía. La ROM (*Read Only Memory*, Memoria de sólo lectura) se programa en la fábrica y no puede modificarse después. Es rápida y económica. En algunas computadoras, el cargador de arranque (*bootstrap loader*) que se utiliza para iniciar la computadora está contenido en la ROM. Además, algunas tarjetas de E/S vienen con ROM para manejar el control de los dispositivos de bajo nivel. La EEPROM (*Electrically Erasable PROM*, PROM eléctricamente borrable) y la memoria flash son también no volátiles, pero


Gráfica 84. IC Serial EEPROM. Fuente: http://mot.theicstock.com/images_part/175/93C66-E_SN_840175.jpg

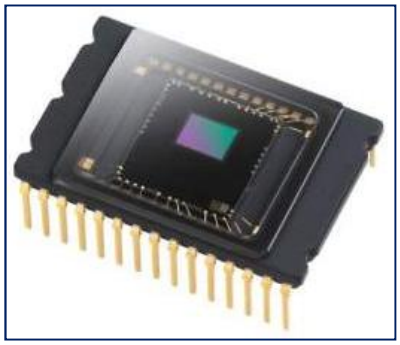
en contraste con la ROM se pueden borrar y volver a escribir datos en ellas. Sin embargo, para escribir en este tipo de memorias se requiere mucho más tiempo que para escribir en la RAM, por lo cual se utilizan en la misma forma que la ROM, sólo con la característica adicional de que ahora es posible corregir los errores en los programas que contienen, mediante la acción de volver a escribir datos en ellas en el campo de trabajo.

Memoria Flash

La memoria flash también se utiliza comúnmente como el medio de almacenamiento en los dispositivos electrónicos portátiles. Sirve como película en las cámaras digitales y como el disco en los reproductores de música portátiles, para nombrar sólo dos usos. La memoria flash se encuentra entre la RAM y el disco en cuanto a su velocidad. Además, a diferencia de la memoria en disco, si se borra demasiadas veces, se desgasta.

Gráfica 85. Memoria Flash. Fuente: http://www.gizmos.es/files/2012/07/microsd-flash.jpg

CMOS



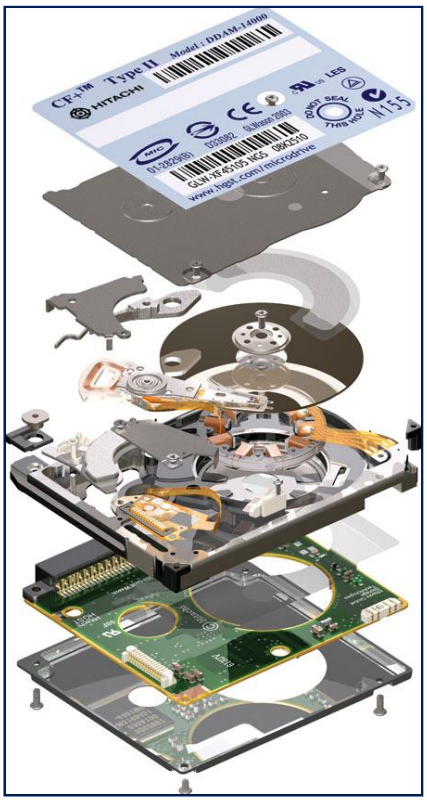
Gráfica 86. Chip CMOS. Fuente: http://www.creativeplanetnetwork.com/dcp/news/cmos-technology-primer/40995

Otro tipo más de memoria es CMOS, la cual es volátil. Muchas computadoras utilizan memoria CMOS para guardar la fecha y hora actuales. La memoria CMOS y el circuito de reloj que incrementa la hora en esta memoria están energizados por una pequeña batería, por lo que la hora se actualiza en forma correcta aun cuando la computadora se encuentre desconectada. La memoria CMOS también puede contener los parámetros de configuración, como el disco del cual se debe iniciar el sistema. Se utiliza CMOS debido a que consume tan poca energía que la batería instalada en la fábrica dura varios años. Sin embargo, cuando empieza a fallar puede parecer como si la computadora tuviera la enfermedad de Alzheimer,

olvidando cosas que ha sabido durante años, como desde cuál disco duro se debe iniciar el sistema.

Discos Magnéticos

El almacenamiento en disco es dos órdenes de magnitud más económico que la RAM por cada bit, y a menudo es dos órdenes de magnitud más grande en tamaño también. El único problema es que el tiempo para acceder en forma aleatoria a los datos en ella es de cerca de tres órdenes de magnitud más lento. Esta baja velocidad se debe al hecho de que un disco es un dispositivo mecánico.



Gráfica 87. Componentes de un Disco.
Fuente: http://laventanamuerta.net

Un disco consiste en uno o más platos que giran a 5400, 7200 o 10,800 rpm. Un brazo mecánico, con un punto de giro colocado en una esquina, se mueve sobre los platos de manera similar al brazo de la aguja en un viejo tocadiscos. La información se escribe en el disco en una serie de círculos concéntricos. En cualquier posición dada del brazo, cada una de las cabezas puede leer una región anular conocida como pista (*track*). En conjunto, todas las pistas para una posición dada del brazo forman un cilindro (*cylinder*).

Cada pista se divide en cierto número de sectores, por lo general de 512 bytes por sector. En los discos modernos, los cilindros exteriores contienen más sectores que los interiores. Para desplazar el brazo de un cilindro al siguiente se requiere aproximadamente 1 milisegundo. Para desplazar el brazo a un cilindro aleatoriamente se requieren por lo general de 5 a 10 milisegundos, dependiendo de la unidad. Una vez que el brazo se encuentra en la pista correcta, la unidad debe esperar a que el sector necesario gire hacia abajo de la cabeza, con un retraso adicional de 5 a 10 milisegundos, dependiendo de las rpm de la unidad. Una vez que el sector está bajo la cabeza, la lectura o escritura ocurre a una velocidad de 50 MB/seg en los discos de bajo rendimiento hasta de 160 MB/seg en los discos más rápidos.

Muchas computadoras presentan un esquema conocido como memoria virtual (*virtual memory*). Este esquema hace posible la ejecución de programas más grandes que la memoria física al colocarlos en el disco y utilizar la memoria principal como un tipo de caché para las partes que se ejecutan con más frecuencia. Este esquema requiere la reasignación de direcciones de memoria al instante, para convertir la dirección que el programa generó en la dirección física en la RAM en donde se encuentra la palabra. Esta asignación se realiza mediante una parte de la CPU conocida como MMU (*Memory Management Unit*, Unidad de Administración de Memoria).

La presencia de la caché y la MMU pueden tener un gran impacto en el rendimiento. En un sistema de multiprogramación, al cambiar de un programa a otro (lo que se conoce comúnmente como cambio de contexto o *context switch*), puede ser necesario vaciar todos los bloques modificados de la caché y modificar los registros de asignación en la MMU. Ambas operaciones son costosas y los programadores se esfuerzan bastante por evitarlas.

Cintas Magnéticas

Gráfica 88. Cinta IBM. Fuente: http://www.fayerwayer.com/up/2010/01/IBM-data-tape-660x350.jpg

La última capa de la jerarquía en la memoria es la cinta magnética. Este medio se utiliza con frecuencia como respaldo para el almacenamiento en disco y para contener conjuntos de datos muy extensos. Para acceder a una cinta, primero debe colocarse en un lector de cinta, ya sea que lo haga una persona o un robot (el manejo automatizado de las cintas es común en las instalaciones con bases de datos enormes). Después la cinta tal vez tenga que embobinarse hacia delante para llegar al bloque solicitado. En general, este proceso podría tardar varios minutos. La gran ventaja de la cinta es que es en extremo económica por bit y removible, lo cual es importante para las cintas de respaldo que se deben almacenar fuera del sitio de trabajo para que puedan sobrevivir a los incendios, inundaciones, terremotos y otros desastres.

IBM en conjunto con Fujifilm se encuentran desarrollando una tecnología que hace posible almacenar 35 TB de datos en una cinta magnética. En teoría esta cinta podría almacenar hasta 35 millones de libros, mide 800 metros de longitud por media pulgada de ancho. Para lograr tal grado de densidad (29.5 mil millones de bits por pulgada al cuadrado) fue necesario crear un nuevo tipo de material basado en Ferrita de Bario, con lo que se logra aumentar la densidad de almacenamiento sin disminuir las propiedades magnéticas de la superficie. El desarrollo de este tipo de cintas está orientado para ser utilizado principalmente en los grandes centros de datos[9].

Conclusión

La jerarquía de memoria descrita es la común, pero algunas instalaciones no tienen todas las capas o tienen unas cuantas capas distintas (como el disco óptico). Aún así, a medida que se desciende por todos las capas en la jerarquía, el tiempo de acceso aleatorio se incrementa en forma dramática, la capacidad aumenta de igual forma y el costo por bit baja considerablemente. En consecuencia, es probable que las jerarquías de memoria se utilicen por varios años más.

4.2. Técnicas de almacenamiento

Los programas y datos tienen que estar en la memoria principal para poder ejecutarse o ser referenciados, pero los programas y datos que no son necesarios de inmediato pueden mantenerse en el almacenamiento secundario. El almacenamiento principal es más costoso y menor que el secundario pero de acceso más rápido.

Los sistemas con varios niveles de almacenamiento requieren destinar recursos para administrar el movimiento de programas y datos entre los niveles.

Se contemplan diversas estrategias para el almacenamiento de los procesos en memoria:

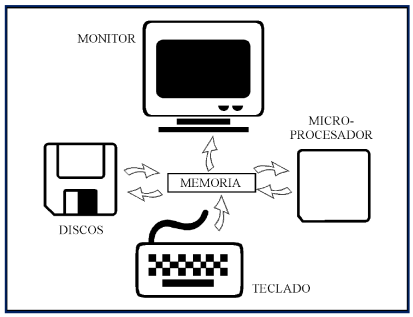
- **Estrategia de mejor ajuste**

  Un trabajo nuevo es colocado en el agujero en el cual quepa de forma más ajustada: debe dejarse el menor espacio sin usar.
- **Estrategia de primer ajuste**

  Un trabajo nuevo es colocado en el primer espacio disponible con tamaño suficiente para alojarlo.
- **Estrategia de peor ajuste**

  Consiste en colocar un programa en el espacio en el que quepa de la peor manera, es decir en el más grande posible. El espacio restante es también grande para poder alojar a un nuevo programa relativamente grande.

4.3. Esquemas de Administración de Memoria



Gráfica 89. Administrador de Memoria.
Fuente: selennadepaz.blogspot.com

La parte del sistema operativo que administra (parte de) la jerarquía de memoria se conoce como administrador de memoria. Su trabajo es administrar la memoria con eficiencia: llevar el registro de cuáles partes de la memoria están en uso, asignar memoria a los procesos cuando la necesiten y desasignarla cuando terminen.

Entre sus funciones[10] están:

- Control de que partes de la memoria están utilizadas o libres.
- Asignar memoria a procesos y liberarla cuando terminan.
- Administrar intercambio entre memoria y disco (Memoria Virtual)

Las herramientas básicas de la gestión de memoria son la paginación y la segmentación. En la paginación, cada proceso se divide en páginas de tamaño constante y relativamente pequeño. La segmentación permite el uso de partes de tamaño variable. También es posible combinar la segmentación y la paginación en un único esquema de gestión de memoria.

En un sistema monoprogramado[11], la memoria principal se divide en dos partes: una parte para el sistema operativo (monitor residente, núcleo) y otra parte para el programa que se ejecuta

en ese instante. En un sistema multiprogramado, la parte de "usuario" de la memoria debe subdividirse aún más para hacer sitio a varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el sistema operativo y se conoce como gestión de memoria.

En un sistema multiprogramado resulta vital una gestión efectiva de la memoria. Si sólo hay unos pocos procesos en memoria, entonces la mayor parte del tiempo estarán esperando a la E/S y el procesador estará desocupado. Por ello, hace falta repartir eficientemente la memoria para meter tantos procesos como sea posible.

4.4. Administración de Memoria contigua simple

Gráfica 90. Asignación de Memoria contigua. Fuente: elaboración propia.

En la "asignación contigua" cada programa ocupa un bloque contiguo y sencillo de localizaciones de almacenamiento.

En la "asignación no contigua" un programa se divide en varios bloques o "segmentos" que pueden almacenarse en direcciones que no tienen que ser necesariamente adyacentes, por lo que es más compleja pero más eficiente que la asignación continua.

El tamaño de los programas está limitado por la cantidad de memoria principal, pero se puede superar este límite con técnicas de "recubrimientos", con las siguientes características:

- Si una sección particular del programa ya no es necesaria, se carga otra sección desde el almacenamiento secundario ocupando las áreas de memoria liberadas por la sección que ya no se necesita.
- La administración manual por programa del recubrimiento es complicada y dificulta el desarrollo y el mantenimiento.

4.5. Administración de Memoria particional

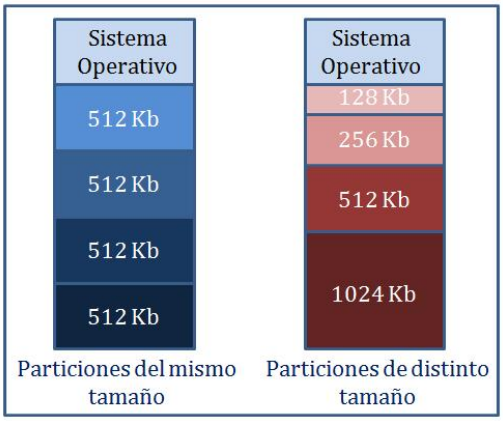
En la mayoría de los esquemas de gestión de memoria, se puede suponer que el sistema operativo ocupa una parte fija de la memoria principal y el resto de la memoria está disponible para ser usado por varios procesos.

El esquema más sencillo de gestión de la memoria disponible es dividirla en particiones con límites fijos. Las particiones pueden ser del mismo o de distinto tamaño, pero generalmente serán de distinto tamaño.

Si las particiones son del mismo tamaño, la ubicación de un proceso en la memoria es trivial, ya que mientras haya una partición libre, puede cargarse un proceso en esa partición. Cuando todas las particiones están ocupadas por procesos que no están listos para ejecutarse, uno de esos procesos debe sacarse y hacer sitio para un nuevo proceso.

Si las particiones son de distinto tamaño, hay dos maneras posibles de asignar los procesos a las particiones:

- La forma más simple es asignar cada proceso a la partición más pequeña en la que quepa, es decir, solo en la que mejor se adapte, para lo cual es preciso conocer la cantidad máxima de memoria, que necesitaría cada uno de los procesos y realmente esta información no siempre se conocerá. En este caso, será necesaria una cola de planificación para cada partición, que albergue los procesos listos para ejecutarse, cuyo destino es esa partición. La ventaja de este enfoque es que los procesos están siempre asignados de forma que se minimiza la memoria desaprovechada dentro de cada partición.



Gráfica 91. Particiones del mismo tamaño y distinto tamaño. Fuente: elaboración propia.

  Este enfoque que parece óptimo desde el punto de vista de una partición individual, no lo es, desde el punto de vista del sistema global, ya que si tenemos particiones grandes y los procesos que van llegando son todos pequeños, las colas de las particiones grandes permanecerán sin usar, incluso aunque algún proceso más pequeño pudiera haber sido asignado a las mismas.
- Una solución mejor sería emplear una sola cola para todos los procesos y cuando se va a cargar un proceso en memoria principal, se selecciona la partición más pequeña disponible que pueda albergar al proceso. Si todas las particiones están ocupadas, se

debe tomar una decisión de intercambio y puede darse preferencia al intercambio de la partición más pequeña, que pueda contener el proceso entrante. También es posible considerar otros factores, tales como, prioridades y preferencia para descargar procesos bloqueados antes que procesos listos.

El uso de particiones de distinto tamaño proporciona cierto grado de flexibilidad a las particiones fijas, además, ambos tipos de esquema de partición fija son relativamente simples y exigen un software del sistema operativo sencillo y una sobrecarga de procesamiento mínima.

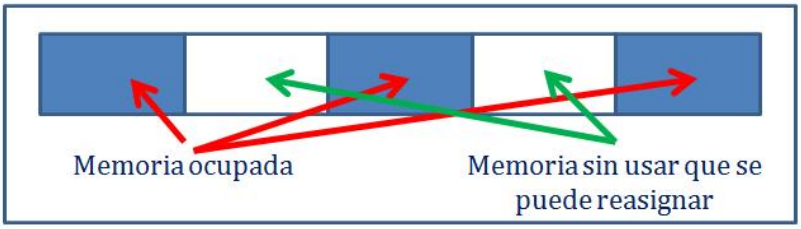
El particionamiento fijo consiste en la asignación estática de la memoria particionada, que es una forma de hacer posible la multiprogramación, dividiendo la memoria física disponible en varias particiones, cada una de las cuales puede ser asignada a diferentes procesos.

Dependiendo de cómo y cuándo son creadas y modificadas esas particiones, el particionamiento de la memoria, puede ser estático o dinámico.

4.6. Administración de Memoria particional re-asignable

Permite cambiar un proceso de partición en caso de ser necesario (reasignarlo).

Registro de reasignación: El registro de relocalización contiene la dirección física más pequeña; el registro limite contiene el rango de las direcciones lógicas. Cada dirección lógica debe ser menor al registro limite.



Gráfica 92. Memoria particional reasignable. Fuente: elaboración propia.

Cuando no se puede asignar ubicación contigua a un proceso por los espacios ocupados por los procesos que quedan en la memoria, se compactan por reasignación hacia arriba.

La desventaja es que no asegura que el programa siga funcionando en la nueva ubicación a menos que se modifiquen los componentes dependientes de dirección tales como:

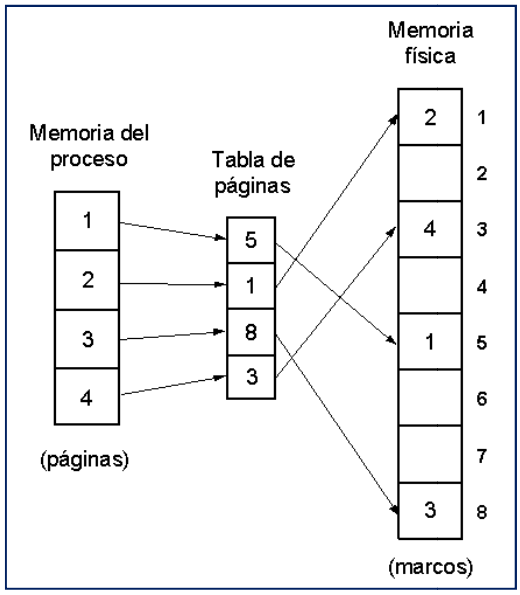
- instrucciones con referencia a memoria,

- lista de parámetros,
- estructura de datos.

La solución es recargar cada programa que deba ser reasignado y empezar desde el principio. Para solventar el problema de la fragmentación se puede:

- tolerar el desperdicio de memoria y degradación,
- aumentar la memoria hasta que la multiprogramación se mantenga siempre en nivel adecuado para el aprovechamiento del CPU,
- dar recursos adicionales de equipo para atacar las causas de la fragmentación.

4.7. Administración de Memoria paginada



Gráfica 93. Paginado. Fuente: http://lichtschein.com.ar/linux/paginado.png

En sistemas operativos de computadoras, los sistemas de paginación de memoria dividen los programas en pequeñas partes o páginas. Del mismo modo, la memoria es dividida en trozos del mismo tamaño que las páginas llamados marcos de página. De esta forma, la cantidad de memoria desperdiciada por un proceso es el final de su última página, lo que minimiza la fragmentación interna y evita la externa.

En un momento cualquiera, la memoria se encuentra ocupada con páginas de diferentes procesos, mientras que algunos marcos están disponibles para su uso. El sistema operativo mantiene una lista de estos últimos marcos, y una tabla por cada proceso, donde consta en qué marco se encuentra cada página del proceso. De esta forma, las páginas de un proceso pueden no estar contiguamente ubicadas en memoria, y pueden intercalarse con las páginas de otros procesos.

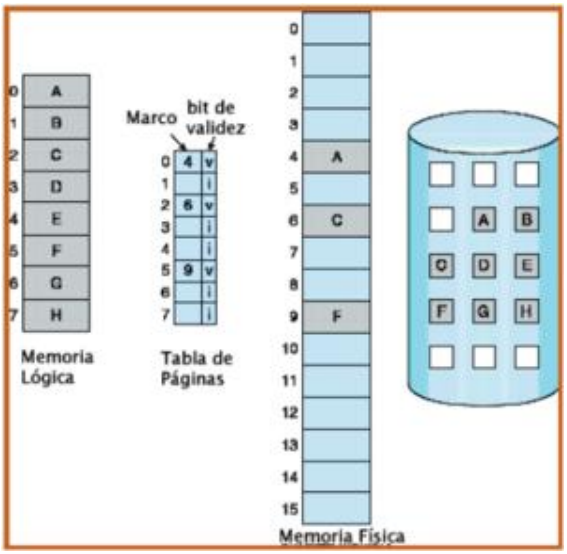
En la tabla de páginas de un proceso, se encuentra la ubicación del marco que contiene a cada una de sus páginas. Las direcciones lógicas ahora se forman como un número de página y de un desplazamiento dentro de esa página (conocido comúnmente como *offset*). El número de página es usado como un índice dentro de la tabla de páginas, y una vez obtenida la dirección

del marco de memoria, se utiliza el desplazamiento para componer la dirección real o dirección física. Este proceso se realiza en una parte del computador específicamente diseñada para esta tarea, es decir, es un proceso hardware y no software.

De esta forma, cuando un proceso es cargado en memoria, se cargan todas sus páginas en marcos libres y se completa su tabla de páginas[12].

4.8. Administración de Memoria paginada por demanda

Los procesos residen en memoria secundaria (en el disco). Cuando queremos ejecutar un proceso, lo alojamos en memoria principal. Sin embargo, en vez de intercambiar todo el proceso hacia la memoria, utilizamos un intercambiador perezoso. Un intercambiador perezoso nunca reincorpora una página a memoria a menos que se necesite. Como ahora consideramos un proceso como una secuencia de páginas en vez de un gran espacio contiguo de direcciones, el término intercambio es técnicamente incorrecto. Un intercambiador manipula procesos enteros, mientras que un paginador trata con las páginas individualmente de un proceso.



Gráfica 94. Paginación por demanda. Fuente: http://wiki.inf.utfsm.cl/index.php?title=Paginaci%C3%B3n_por_demanda_y_Fallos_de_P%C3%A1ginas

Cuando un proceso se reincorpora, el paginador lleva a memoria las páginas necesarias. Así evita colocar en la memoria páginas que no se utilizarán, reduciendo el tiempo de intercambio y la cantidad de memoria física necesaria.

Este esquema requiere apoyo del hardware. Generalmente se añade un bit más a cada entrada de la tabla de páginas: un bit válido-inválido. Cuando este bit está asignado como válido, indica que la página asociada se encuentra en memoria. Si el bit está como inválido, este valor indica que la página está en disco. Una página marcada como inválida no tendrá ningún efecto si el proceso nunca intenta acceder a esa página.

Ventajas[13]

- Al no cargar las páginas que no son utilizadas ahorra memoria para otras aplicaciones.
- Al mejorar el uso de la memoria, mejora el grado de multiprogramación.
- Carga inicial más rápida ya que solo lee del disco lo que se utilizará.
- Capacidad de hacer funcionar programas que ocupan más memoria que la poseída.
- Copia en escritura: Permite utilizar las mismas páginas para dos procesos (padre-hijo) hasta que uno de estos las modifique.

Desventajas[14]

- Debido a la sobre-asignación podemos quedarnos sin frames libres para agregar nuevas páginas, si esto sucede debemos recurrir a un reemplazo.
- Cada fallo de página requiere cargar a memoria una página a leer, si ocurren muchos fallos de página el rendimiento empeora notablemente.
- Las páginas que son sacadas de los *frames* por intercambio pueden volver a ser llamadas, lo que ocasiona que se lea en múltiples ocasiones la misma información.

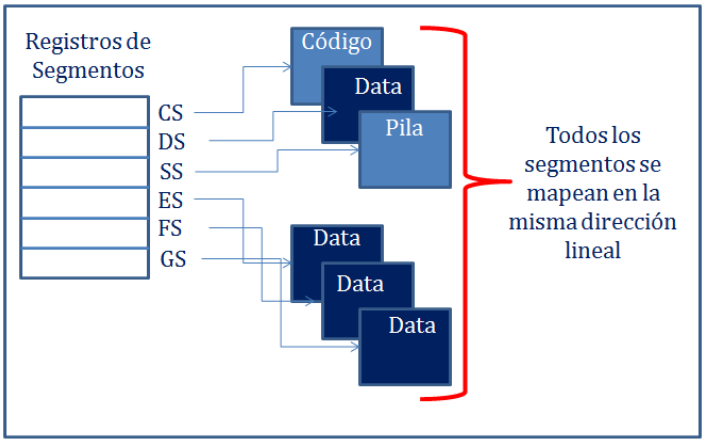
4.9. Administración de Memoria segmental

La segmentación[15] es una técnica de gestión de memoria que pretende acercarse más al punto de vista del usuario. Los programas se desarrollan, generalmente, en torno a un núcleo central (principal) desde el que se bifurca a otras partes (rutinas) o se accede a zonas de datos (tablas, pilas, etc).

Desde este punto de vista, un programa es un conjunto de componentes lógicos de tamaño variable o un conjunto de segmentos, es decir, el espacio lógico de direcciones se considera como un conjunto de segmentos, cada uno definido por un identificador, y consistente de un punto de inicio y el tamaño asignado.

La segmentación de un programa la realiza el compilador y en ella cada dirección lógica se expresará mediante dos valores: Número de segmento (s) y desplazamiento dentro del segmento (d).

Una de las implementaciones más obvias y directas de un espacio de memoria segmentado es asignar un segmento distinto a cada una de las secciones del espacio en memoria de un proceso.



Gráfica 95. Memoria segmentada. Fuente: elaboración propia.

La segmentación también ayuda a incrementar la modularidad de un programa: Es muy común que las bibliotecas enlazadas dinámicamente estén representadas en segmentos independientes.

En sí es un esquema de manejo de memoria mediante el cual la estructura del programa refleja su división lógica; llevándose a cabo una agrupación lógica de la información en bloques de tamaño variable denominados segmentos.

Permite al programador contemplar la memoria como si constara de varios espacios de direcciones o segmentos. Los segmentos pueden ser de distintos tamaños, incluso de forma dinámica. Se utilizan para encapsular regiones de memoria que tienen atributos comunes.

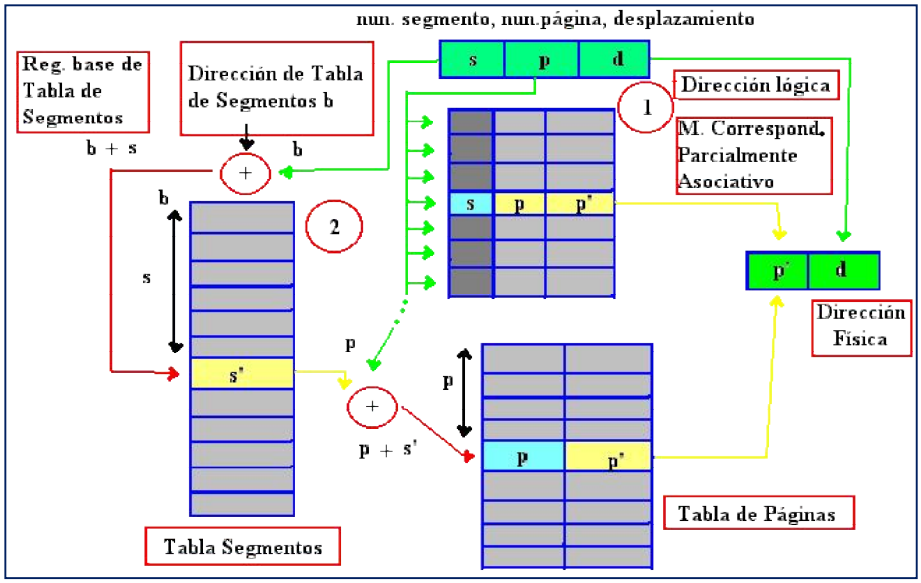
En este caso[16], el programa y sus datos asociados se dividen en un conjunto de segmentos. No es necesario que todos los segmentos de todos los programas tengan la misma longitud, aunque existe una longitud máxima de segmento. Como en la paginación, una dirección lógica segmentada consta de dos partes, en este caso un número de segmento y un desplazamiento.

Como consecuencia del empleo de segmentos de distinto tamaño, la segmentación resulta similar a la partición dinámica. En ausencia de un esquema de superposición o del uso de memoria virtual, sería necesario cargar en memoria todos los segmentos de un programa para su ejecución. La diferencia, en comparación con la partición dinámica, radica en que, con segmentación, un programa puede ocupar más de una partición y éstas no tienen por qué estar contiguas. La segmentación elimina la fragmentación interna, pero, como la partición dinámica, sufre de fragmentación externa. Sin embargo, debido a que los procesos se dividen en un conjunto de partes más pequeñas, la fragmentación externa será menor.

Mientras que la paginación es transparente al programador, la segmentación es generalmente visible y se proporciona como una comodidad para la organización de los programas y datos. Normalmente, el programador o el compilador asignan los programas y los datos a diferentes segmentos. En aras de la programación modular, el programa o los datos pueden ser divididos de nuevo en diferentes segmentos. El principal inconveniente de este servicio es que el programador debe ser consciente de la limitación de tamaño máximo de los segmentos.

4.10. Administración de Memoria segmental paginada

Consiste en dividir a los segmentos en varias páginas de igual tamaño. A cada proceso se le asigna una tabla de segmentos, donde cada segmento posee una tabla de páginas. En este sistema, el bit de presencia y modificado es innecesario en la tabla de segmentos, ya que se especifican en las tablas de páginas.

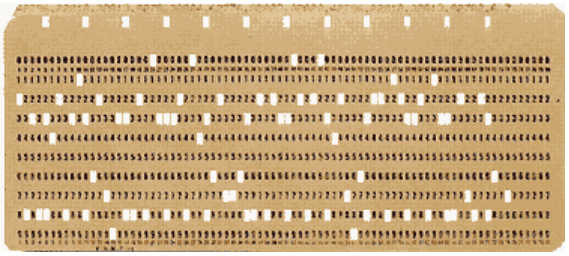


Gráfica 96. Memoria segmentada paginada. Fuente: http://lsi.vc.ehu.es/pablogn/docencia/manuales/SO/TemasSOuJaen/ADMINISTRACIONDELAMEMORIA/Image101.gif

4.11. Complemento: Tarjeta Perforada

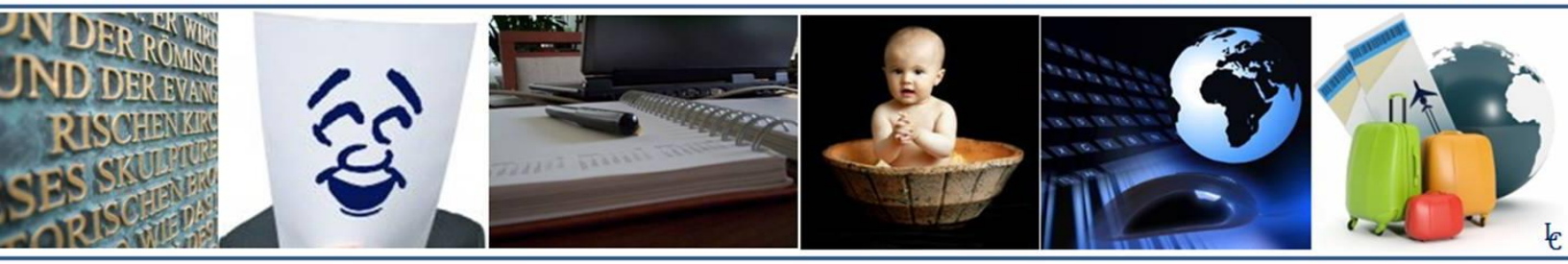
La tarjeta perforada o simplemente tarjeta es una lámina hecha de cartulina que contiene información en forma de perforaciones según un código binario. Estos fueron los primeros medios utilizados para ingresar información e instrucciones a un computador en los años 1960 y 1970. Las tarjetas perforadas fueron usadas con anterioridad por Joseph Marie Jacquard en

los telares de su invención, de donde pasó a las primeras computadoras electrónicas. Con la misma lógica se utilizaron las cintas perforadas.



Gráfica 97. Tarjeta perforada. Fuente: http://html.rincondelvago.com/000200270.png

Actualmente las tarjetas perforadas han sido reemplazadas por medios magnéticos y ópticos de ingreso de información. Sin embargo, muchos de los dispositivos de almacenamiento actuales, como por ejemplo el CD-ROM también se basa en un método similar al usado por las tarjetas perforadas, aunque por supuesto los tamaños, velocidades de acceso y capacidad de los medios actuales no admiten comparación con los antiguos medios[17].



5. Sistemas de Archivos

Todas las aplicaciones de computadora requieren almacenar y recuperar información. Mientras un proceso está en ejecución, puede almacenar una cantidad limitada de información dentro de su propio espacio de direcciones. Sin embargo, la capacidad de almacenamiento está restringida por el tamaño del espacio de direcciones virtuales. Para algunas aplicaciones este tamaño es adecuado; para otras, tales como las de reservaciones en aerolíneas, las bancarias o las de contabilidad corporativa, puede ser demasiado pequeño.

Gráfica 98. Archivador físico. Fuente: sistemasdearchivosdistribuidosgrupo4.blogspot.com

Un segundo problema relacionado con el mantenimiento de la información dentro del espacio de direcciones de un proceso es que cuando el proceso termina, la información se pierde. Para muchas aplicaciones (por ejemplo, una base de datos) la información se debe retener durante semanas, meses o incluso indefinidamente. Es inaceptable que esta información se desvanezca cuando el proceso que la utiliza termine. Además, no debe desaparecer si una falla en la computadora acaba con el proceso.

Un tercer problema es que frecuentemente es necesario que varios procesos accedan a (partes de) la información al mismo tiempo. Si tenemos un directorio telefónico en línea almacenado dentro del espacio de direcciones de un solo proceso, sólo ese proceso puede tener acceso al directorio. La manera de resolver este problema es hacer que la información en sí sea independiente de cualquier proceso.

En consecuencia, tenemos tres requerimientos esenciales para el almacenamiento de información a largo plazo:

- Debe ser posible almacenar una cantidad muy grande de información.
- La información debe sobrevivir a la terminación del proceso que la utilice.
- Múltiples procesos deben ser capaces de acceder a la información concurrentemente.

Durante muchos años se han utilizado discos magnéticos para este almacenamiento de largo plazo, así como cintas y discos ópticos, aunque con un rendimiento mucho menor.

Los archivos son unidades lógicas de información creada por los procesos. En general, un disco contiene miles o incluso millones de archivos independientes. De hecho, si concibe a cada archivo como un tipo de espacio de direcciones, no estará tan alejado de la verdad, excepto porque se utilizan para modelar el disco en vez de modelar la RAM.

Los procesos pueden leer los archivos existentes y crear otros si es necesario. La información que se almacena en los archivos debe ser persistente, es decir, no debe ser afectada por la creación y terminación de los procesos. Un archivo debe desaparecer sólo cuando su propietario lo remueve de manera explícita. Aunque las operaciones para leer y escribir en archivos son las más comunes, existen muchas otras, algunas de las cuales examinaremos a continuación.

Los archivos son administrados por el sistema operativo. La manera en que se estructuran, denominan, abren, utilizan, protegen, implementan y administran son tópicos fundamentales en el diseño de sistemas operativos. La parte del sistema operativo que trata con los archivos se conoce como sistema de archivos.

Desde el punto de vista del usuario, el aspecto más importante de un sistema de archivos es su apariencia; es decir, qué constituye un archivo, cómo se denominan y protegen los archivos qué operaciones se permiten con ellos, etcétera. Los detalles acerca de si se utilizan listas enlazadas (ligadas) o mapas de bits para llevar la cuenta del almacenamiento libre y cuántos sectores hay

en un bloque de disco lógico no son de interés, aunque sí de gran importancia para los diseñadores del sistema de archivos.

5.1. Archivos

Los archivos son un mecanismo de abstracción. Proporcionan una manera de almacenar información en el disco y leerla después. Esto se debe hacer de tal forma que se proteja al usuario de los detalles acerca de cómo y dónde se almacena la información y cómo funcionan los discos en realidad.

Probablemente, la característica más importante de cualquier mecanismo de abstracción sea la manera en que los objetos administrados son denominados, por lo que empezaremos nuestro examen de los sistemas de archivos con el tema de la nomenclatura de los archivos. Cuando un proceso crea un archivo le proporciona un nombre. Cuando el proceso termina, el archivo continúa existiendo y puede ser utilizado por otros procesos mediante su nombre.

Las reglas exactas para denominar archivos varían un poco de un sistema a otro, pero todos los sistemas operativos actuales permiten cadenas de una a ocho letras como nombres de archivos legales. Por ende, andrea, bruce y cathy son posibles nombres de archivos. Con frecuencia también se permiten dígitos y caracteres especiales, por lo que nombres como 2, urgente! y Fig.2-14 son a menudo válidos también. Muchos sistemas de archivos admiten nombres de hasta 255 caracteres.

Algunos sistemas de archivos diferencian las letras mayúsculas de las minúsculas, mientras que otros no. UNIX cae en la primera categoría; MS-DOS en la segunda. Así, un sistema UNIX puede tener los siguientes nombres como tres archivos distintos: maria, Maria y MARIA. En MS-DOS, todos estos nombres se refieren al mismo archivo. Tal vez sea adecuado hacer en este momento un paréntesis sobre sistemas de archivos. Windows 95 y Windows 98 utilizan el sistema de archivos de MS-DOS conocido como FAT-16 y por ende heredan muchas de sus propiedades, como la forma en que se construyen sus nombres. Windows 98 introdujo algunas extensiones a FAT-16, lo cual condujo a FAT-32, pero estos dos sistemas son bastante similares. Además,

Windows NT, Windows 2000, Windows XP y .WV admiten ambos sistemas de archivos FAT, que en realidad ya son obsoletos. Estos cuatro sistemas operativos basados en NT tienen un sistema de archivos nativo (NTFS) con diferentes propiedades (como los nombres de archivos en Unicode).

Gráfica 99. Algunos tipos de extensiones. Fuente: http://www.elflip.info/UserFiles/Image/filetypes.jpg

Muchos sistemas operativos aceptan nombres de archivos en dos partes, separadas por un punto, como en prog.c. La parte que va después del punto se conoce como la extensión del archivo y por lo general indica algo acerca de su naturaleza. Por ejemplo, en MS-DOS, los nombres de archivos son de 1 a 8 caracteres, más una extensión opcional de 1 a 3 caracteres. En UNIX el tamaño de la extensión (si la hay) es a elección del usuario y un archivo puede incluso tener dos o más extensiones, como en paginainicio.html.zip, donde .html indica una página Web en HTML y .zip indica que el archivo se ha comprimido mediante el programa zip.

En algunos sistemas (como UNIX) las extensiones de archivo son sólo convenciones y no son impuestas por los sistemas operativos. Un archivo llamado archivo.txt podría ser algún tipo de archivo de texto, pero ese nombre es más un recordatorio para el propietario que un medio para transportar información a la computadora. Por otro lado, un compilador de C podría insistir que los archivos que va a compilar terminen con .c y podría rehusarse a compilarlos si no tienen esa terminación.

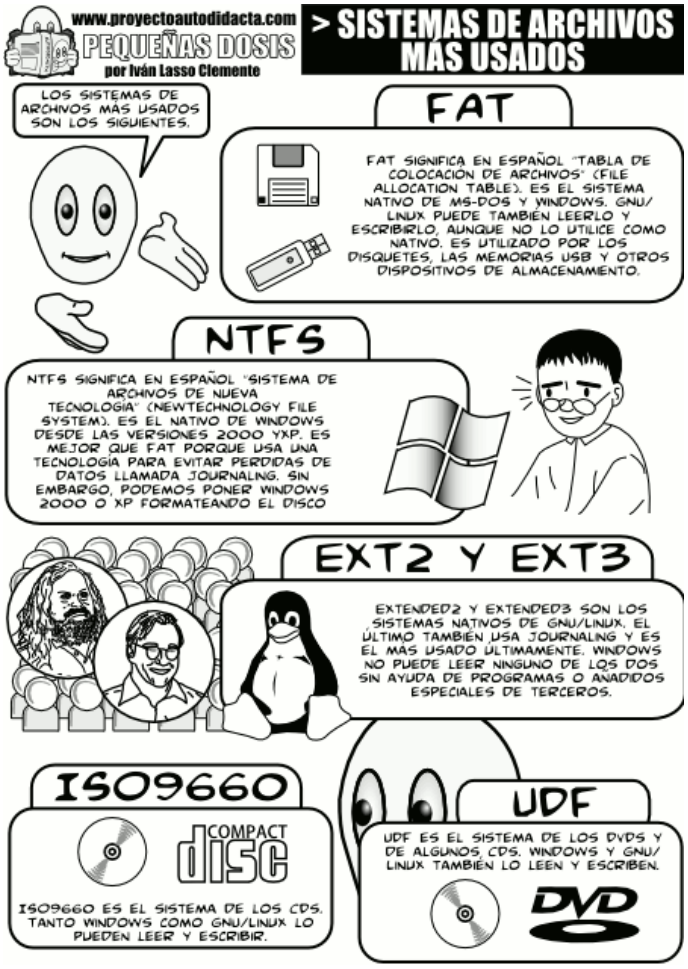
> En informática, una extensión de archivo o extensión de fichero, es una cadena de caracteres anexada al nombre de un archivo, usualmente precedida por un punto. Su función principal es diferenciar el contenido del archivo de modo que el sistema operativo disponga el procedimiento necesario para ejecutarlo o interpretarlo, sin embargo, la extensión es solamente parte del nombre del archivo y no representa ningún tipo de obligación respecto a su contenido.
>
> Tomado de Wikipedia

Las convenciones como ésta son especialmente útiles cuando el mismo programa puede manejar diferentes tipos de archivos. Por ejemplo, el compilador C puede recibir una lista de varios archivos para compilarlos y enlazarlos, algunos de ellos archivos de C y otros archivos de lenguaje

ensamblador. Entonces, la extensión se vuelve esencial para que el compilador sepa cuáles son archivos de C, cuáles son archivos de lenguaje ensamblador y cuáles son archivos de otro tipo.

Todas las extensiones de archivos, en orden alfabético: http://whatis.techtarget.com/file-extension-list/A

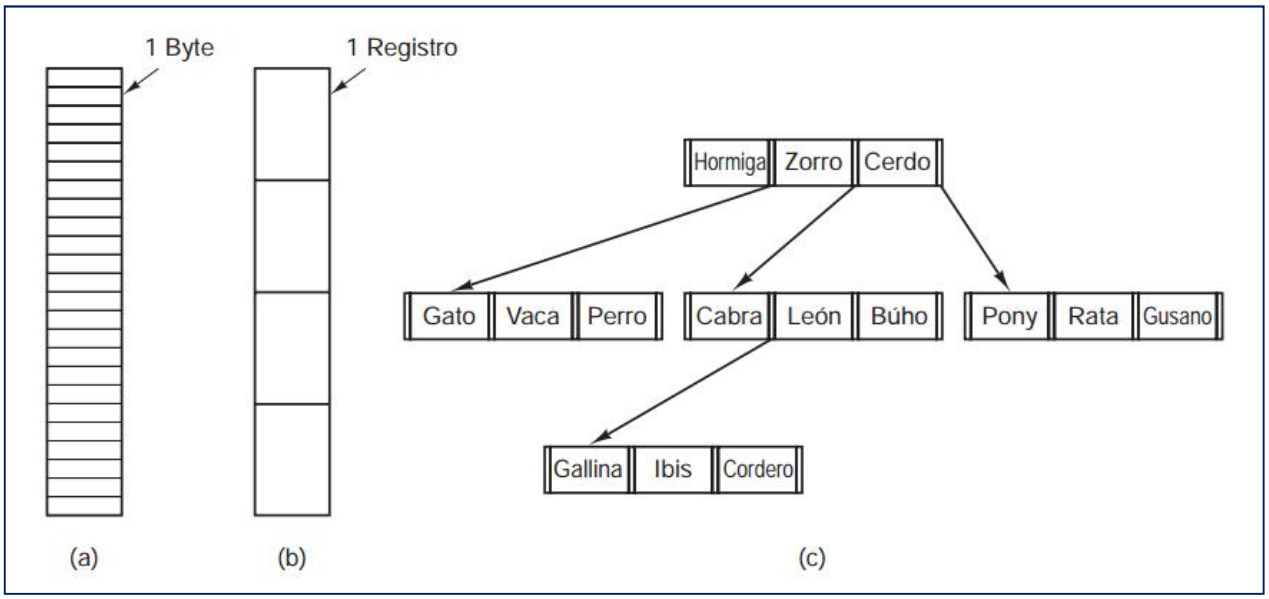
Por el contrario, Windows está consciente de las extensiones y les asigna significado. Los usuarios (o procesos) pueden registrar extensiones con el sistema operativo y especificar para cada una cuál programa "posee" esa extensión. Cuando un usuario hace doble clic sobre un nombre de archivo, el programa asignado a su extensión de archivo se inicia con el archivo como parámetro. Por ejemplo, al hacer doble clic en archivo.doc se inicia Microsoft Word con archivo.doc como el archivo inicial a editar.



Gráfica 100. Sistema de Archivos más usados. Fuente: http://www.proyectoautodidacta.com

Estructura de Archivos

Los archivos se pueden estructurar en una de varias formas. Tres posibilidades comunes se describen en la gráfica 101. El archivo en la gráfica 101 (a) es una secuencia de bytes sin estructura: el sistema operativo no sabe, ni le importa, qué hay en el archivo. Todo lo que ve son bytes. Cualquier significado debe ser impuesto por los programas a nivel usuario. Tanto UNIX como Windows utilizan esta metodología.



Gráfica 101. 3 tipos de archivo. Fuente: Tanembaum (2009)

Hacer que el sistema operativo considere los archivos sólo como secuencias de bytes provee la máxima flexibilidad. Los programas de usuario pueden colocar cualquier cosa que quieran en sus archivos y denominarlos de cualquier manera conveniente. El sistema operativo no ayuda, pero tampoco estorba. Para los usuarios que desean realizar cosas inusuales, esto último puede ser muy importante. Todas las versiones de UNIX, MS-DOS y Windows utilizan este modelo de archivos.

La primera configuración en la estructura se muestra en la gráfica 101 (b). En este modelo, un archivo es una secuencia de registros de longitud fija, cada uno con cierta estructura interna. El concepto central para la idea de que un archivo sea una secuencia de registros es la idea de que la operación de lectura devuelva un registro y la operación de escritura sobrescriba o agregue un registro. Como nota histórica, hace algunas décadas, cuando reinaba la tarjeta perforada de 80 columnas, muchos sistemas operativos de mainframes basaban sus sistemas de archivos en

archivos consistentes de registros de 80 caracteres; es decir, en imágenes de la tarjeta. Estos sistemas también admitían archivos con registros de 132 caracteres, que fueron destinados para la impresora de línea (que en esos días eran grandes impresoras de cadena con 132 columnas). Los programas leían la entrada en unidades de 80 caracteres y la escribían en unidades de 132 caracteres, aunque los últimos 52 podían ser espacios, desde luego. Ningún sistema de propósito general de la actualidad utiliza ya este modelo como su sistema de archivos primario, pero en aquellos días de las tarjetas perforadas de 80 columnas y del papel de impresora de línea de 132 caracteres, éste era un modelo común en las computadoras mainframe.

El tercer tipo de estructura de archivo se muestra en la gráfica 101 (c). En esta organización, un archivo consiste de un árbol de registros, donde no todos son necesariamente de la misma longitud; cada uno de ellos contiene un campo llave en una posición fija dentro del registro. El árbol se ordena con base en el campo llave para permitir una búsqueda rápida por una llave específica. La operación básica aquí no es obtener el "siguiente" registro, aunque eso también es posible, sino obtener el registro con una llave específica. Para el archivo del zoológico de la gráfica 101 (c), podríamos pedir al sistema que, por ejemplo, obtenga el registro cuya llave sea pony, sin preocuparnos acerca de su posición exacta en el archivo. Además, se pueden agregar nuevos registros al archivo, con el sistema operativo, y no el usuario, decidiendo dónde colocarlos. Evidentemente, este tipo de archivos es bastante distinto de los flujos de bytes sin estructura que se usan en UNIX y Windows, pero se utiliza de manera amplia en las grandes computadoras mainframe que aún se emplean en algún procesamiento de datos comerciales.

Tipos de archivos

Muchos sistemas operativos soportan varios tipos de archivos. Por ejemplo, UNIX y Windows tienen archivos y directorios regulares. UNIX también tiene archivos especiales de caracteres y de bloques. Los archivos regulares son los que contienen información del usuario. Todos los archivos de la gráfica 101 son archivos regulares. Los directorios son sistemas de archivos para mantener la estructura del sistema de archivos. Los archivos especiales de caracteres se relacionan con la entrada/salida y se utilizan para modelar dispositivos de E/S en serie, tales como terminales, impresoras y redes. Los archivos especiales de bloques se utilizan para

modelar discos. En este capítulo estaremos interesados principalmente en los archivos regulares.

Por lo general, los archivos regulares son archivos ASCII o binarios. Los archivos ASCII consisten en líneas de texto. En algunos sistemas, cada línea se termina con un carácter de retorno de carro. En otros se utiliza el carácter de avance de línea. Algunos sistemas (por ejemplo, MS-DOS) utilizan ambos. No todas las líneas necesitan ser de la misma longitud.

La gran ventaja de los archivos ASCII es que se pueden mostrar e imprimir como están, y se pueden editar con cualquier editor de texto. Además, si muchos programas utilizan archivos ASCII para entrada y salida, es fácil conectar la salida de un programa con la entrada de otro, como en las canalizaciones de shell. (La plomería entre procesos no es más fácil, pero la interpretación de la información lo es si una convención estándar, tal como ASCII, se utiliza para expresarla).

Otros archivos son binarios, lo cual sólo significa que no son archivos ASCII. Al listarlos en la impresora aparece un listado incomprensible de caracteres. Por lo general tienen cierta estructura interna conocida para los programas que los utilizan.

Acceso de Archivos

Los primeros sistemas operativos proporcionaban sólo un tipo de acceso: acceso secuencial. En estos sistemas, un proceso podía leer todos los bytes o registros en un archivo en orden, empezando desde el principio, pero no podía saltar algunos y leerlos fuera de orden. Sin embargo, los archivos secuenciales podían rebobinarse para poder leerlos todas las veces que fuera necesario. Los archivos secuenciales eran convenientes cuando el medio de almacenamiento era cinta magnética en vez de disco.

Cuando se empezó a usar discos para almacenar archivos, se hizo posible leer los bytes o registros de un archivo fuera de orden, pudiendo acceder a los registros por llave en vez de posición.

Los archivos cuyos bytes o registros se pueden leer en cualquier orden se llaman archivos de acceso aleatorio. Son requeridos por muchas aplicaciones.

Los archivos de acceso aleatorio son esenciales para muchas aplicaciones, como los sistemas de bases de datos. Si el cliente de una aerolínea llama y desea reservar un asiento en un vuelo específico, el programa de reservación debe poder tener acceso al registro para ese vuelo sin tener que leer primero los miles de registros de otros vuelos.

Es posible utilizar dos métodos para especificar dónde se debe empezar a leer. En el primero, cada operación read da la posición en el archivo en la que se va a empezar a leer. En el segundo se provee una operación especial (seek) para establecer la posición actual. Después de una operación seek, el archivo se puede leer de manera secuencial desde la posición actual. Este último método se utiliza en UNIX y Windows.

Atributos de Archivos

Atributo	Significado
Protección	Quién puede acceso al archivo y en qué forma
Contraseña	Contraseña necesaria para acceder al archivo
Creador	ID de la persona que creó el archivo
Propietario	El propietario actual
Bandera de sólo lectura	0 para lectura/escritura; 1 para sólo lectura
Bandera oculto	0 para normal; 1 para que no aparezca en los listados
Bandera del sistema	0 para archivos normales; 1 para archivo del sistema
Bandera de archivo	0 si ha sido respaldado; 1 si necesita respaldarse
Bandera ASCII/binario	0 para archivo ASCII; 1 para archivo binario
Bandera de acceso aleatorio	0 para sólo acceso secuencial; 1 para acceso aleatorio
Bandera temporal	0 para normal; 1 para eliminar archivo al salir del proceso
Banderas de bloqueo	0 para desbloqueado; distinto de cero para bloqueado
Longitud de registro	Número de bytes en un registro
Posición de la llave	Desplazamiento de la llave dentro de cada registro
Longitud de la llave	Número de bytes en el campo llave
Hora de creación	Fecha y hora en que se creó el archivo
Hora del último acceso	Fecha y hora en que se accedió al archivo por última vez
Hora de la última modificación	Fecha y hora en que se modificó por última vez el archivo
Tamaño actual	Número de bytes en el archivo
Tamaño máximo	Número de bytes hasta donde puede crecer el archivo

Gráfica 102. Atributos de Archivos. Fuente: Tanembaum (2009).

Todo archivo tiene un nombre y sus datos. Además, todos los sistemas operativos asocian otra información con cada archivo; por ejemplo, la fecha y hora de la última modificación del archivo y su tamaño. A estos elementos adicionales les llamaremos atributos del archivo. Algunas personas los llaman metadatos. La lista de atributos varía considerablemente de un sistema a otro. La tabla de la gráfica 102 muestra algunas de las posibilidades, pero existen otras. Ningún sistema existente tiene todos, pero cada uno de ellos está presente en algún sistema. Los primeros cuatro atributos se relacionan con la protección del archivo e indican quién puede acceder a él y quién no. Todos los tipos de esquemas son posibles, algunos de los cuales estudiaremos más adelante. En algunos sistemas, el usuario debe presentar una contraseña para acceder a un archivo, en cuyo caso la contraseña debe ser uno de los atributos.

Las banderas son bits o campos cortos que controlan o habilitan cierta propiedad específica. Por ejemplo, los archivos ocultos no aparecen en los listados de todos los archivos. La bandera de archivo es un bit que lleva el registro de si el archivo se ha respaldado recientemente. El programa de respaldo lo desactiva y el sistema operativo lo activa cada vez que se modifica un archivo. De esta forma, el programa de respaldo puede indicar qué archivos necesitan respaldarse. La bandera temporal permite marcar un archivo para la eliminación automática cuando el proceso que lo creó termina.

Los campos longitud de registro, posición de llave y longitud de llave sólo están presentes en los archivos en cuyos registros se pueden realizar búsquedas mediante el uso de una llave. Ellos proporcionan la información requerida para buscar las llaves.

Los diversos tiempos llevan la cuenta de cuándo se creó el archivo, su acceso y su modificación más recientes. Éstos son útiles para una variedad de propósitos. Por ejemplo, un archivo de código fuente que se ha modificado después de la creación del archivo de código objeto correspondiente necesita volver a compilarse. Estos campos proporcionan la información necesaria.

El tamaño actual indica qué tan grande es el archivo en el presente. Algunos sistemas operativos de computadoras mainframe antiguas requieren que se especifique el tamaño máximo a la hora de crear el archivo, para poder permitir que el sistema operativo reserve la cantidad máxima de almacenamiento de antemano. Los sistemas operativos de estaciones de trabajo y computadoras personales son lo bastante inteligentes como para arreglárselas sin esta característica.

Operaciones de archivos

Los archivos existen para almacenar información y permitir que se recupere posteriormente. Distintos sistemas proveen diferentes operaciones para permitir el almacenamiento y la recuperación.

A continuación se muestra un análisis de las llamadas al sistema más comunes relacionadas con los archivos.

- Create. El archivo se crea sin datos. El propósito de la llamada es anunciar la llegada del archivo y establecer algunos de sus atributos.
- Delete. Cuando el archivo ya no se necesita, se tiene que eliminar para liberar espacio en el disco. Siempre hay una llamada al sistema para este propósito.
- Open. Antes de usar un archivo, un proceso debe abrirlo. El propósito de la llamada a open es permitir que el sistema lleve los atributos y la lista de direcciones de disco a memoria principal para tener un acceso rápido a estos datos en llamadas posteriores.

- Close. Cuando terminan todos los accesos, los atributos y las direcciones de disco ya no son necesarias, por lo que el archivo se debe cerrar para liberar espacio en la tabla interna. Muchos sistemas fomentan esto al imponer un número máximo de archivos abiertos en los proceso. Un disco se escribe en bloques y al cerrar un archivo se obliga a escribir el último bloque del archivo, incluso aunque ese bloque no esté lleno todavía.
- Read. Los datos se leen del archivo. Por lo general, los bytes provienen de la posición actual. El llamador debe especificar cuántos datos se necesitan y también debe proporcionar un búfer para colocarlos.
- Write. Los datos se escriben en el archivo otra vez, por lo general en la posición actual. Si la posición actual es al final del archivo, aumenta su tamaño. Si la posición actual está en medio del archivo, los datos existentes se sobrescriben y se pierden para siempre.
- Append. Esta llamada es una forma restringida de write. Sólo puede agregar datos al final del archivo. Los sistemas que proveen un conjunto mínimo de llamadas al sistema por lo general no tienen append; otros muchos sistemas proveen varias formas de realizar la misma acción y algunas veces ésos tienen append.
- Seek. Para los archivos de acceso aleatorio, se necesita un método para especificar de dónde se van a tomar los datos. Una aproximación común es una llamada al sistema de nombre seek, la cual reposiciona el apuntador del archivo en una posición específica del archivo. Una vez que se completa esta llamada, se pueden leer o escribir datos en esa posición.
- Get attributes. A menudo, los procesos necesitan leer los atributos de un archivo para realizar su trabajo. Por ejemplo, el programa make de UNIX se utiliza con frecuencia para administrar proyectos de desarrollo de software que consisten en muchos archivos

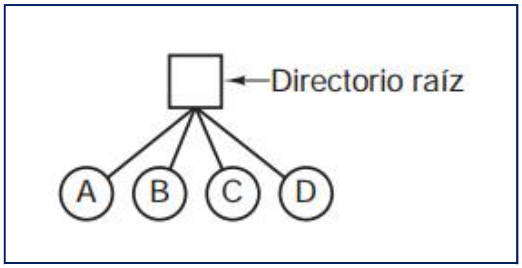
fuente. Cuando se llama a make, este programa examina los tiempos de modificación de todos los archivos fuente y objeto, con los que calcula el mínimo número de compilaciones requeridas para tener todo actualizado. Para hacer su trabajo, debe analizar los atributos, a saber, los tiempos de modificación.

- Set attributes. Algunos de los atributos puede establecerlos el usuario y se pueden modificar después de haber creado el archivo. Esta llamada al sistema hace eso posible. La información del modo de protección es un ejemplo obvio. La mayoría de las banderas también caen en esta categoría.
- Rename. Con frecuencia ocurre que un usuario necesita cambiar el nombre de un archivo existente. Esta llamada al sistema lo hace posible. No siempre es estrictamente necesaria, debido a que el archivo por lo general se puede copiar en un nuevo archivo con el nuevo nombre, eliminando después el archivo anterior.

5.2. Directorios o Carpetas

Para llevar el registro de los archivos, los sistemas de archivos por lo general tienen directorios o carpetas, que en muchos sistemas son también archivos. En esta sección hablaremos sobre los directorios, su organización, sus propiedades y las operaciones que pueden realizarse con ellos.

Sistemas de directorios de un solo nivel



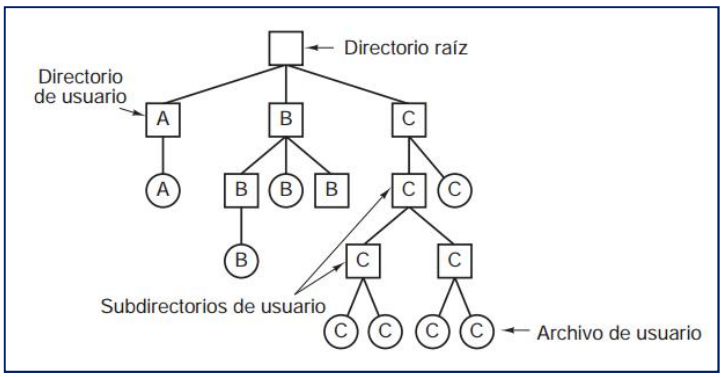
Gráfica 103. Directorio de un solo nivel. Fuente: Tanembaum (2009)

La forma más simple de un sistema de directorios es tener un directorio que contenga todos los archivos. Algunas veces se le llama directorio raíz, pero como es el único, el nombre no importa mucho. En las primeras computadoras personales, este sistema era común, en parte debido a que sólo había un usuario. Como dato interesante, la primera supercomputadora del mundo (CDC 6600) también tenía un solo directorio para todos los archivos, incluso cuando era utilizada por muchos usuarios a la vez. Esta decisión sin duda se hizo para mantener simple el diseño del software.

En la gráfica 103 se muestra un ejemplo de un sistema con un directorio. Aquí el directorio contiene cuatro archivos. Las ventajas de este esquema son su simpleza y la habilidad de localizar archivos con rapidez; después de todo, sólo hay un lugar en dónde buscar. A menudo se utiliza en dispositivos incrustados simples como teléfonos, cámaras digitales y algunos reproductores de música portátiles.

Sistemas de directorios jerárquicos

Tener un solo nivel es adecuado para aplicaciones dedicadas simples (e incluso se utilizaba en las primeras computadoras personales), pero para los usuarios modernos con miles de archivos, sería imposible encontrar algo si todos los archivos estuvieran en un solo directorio.

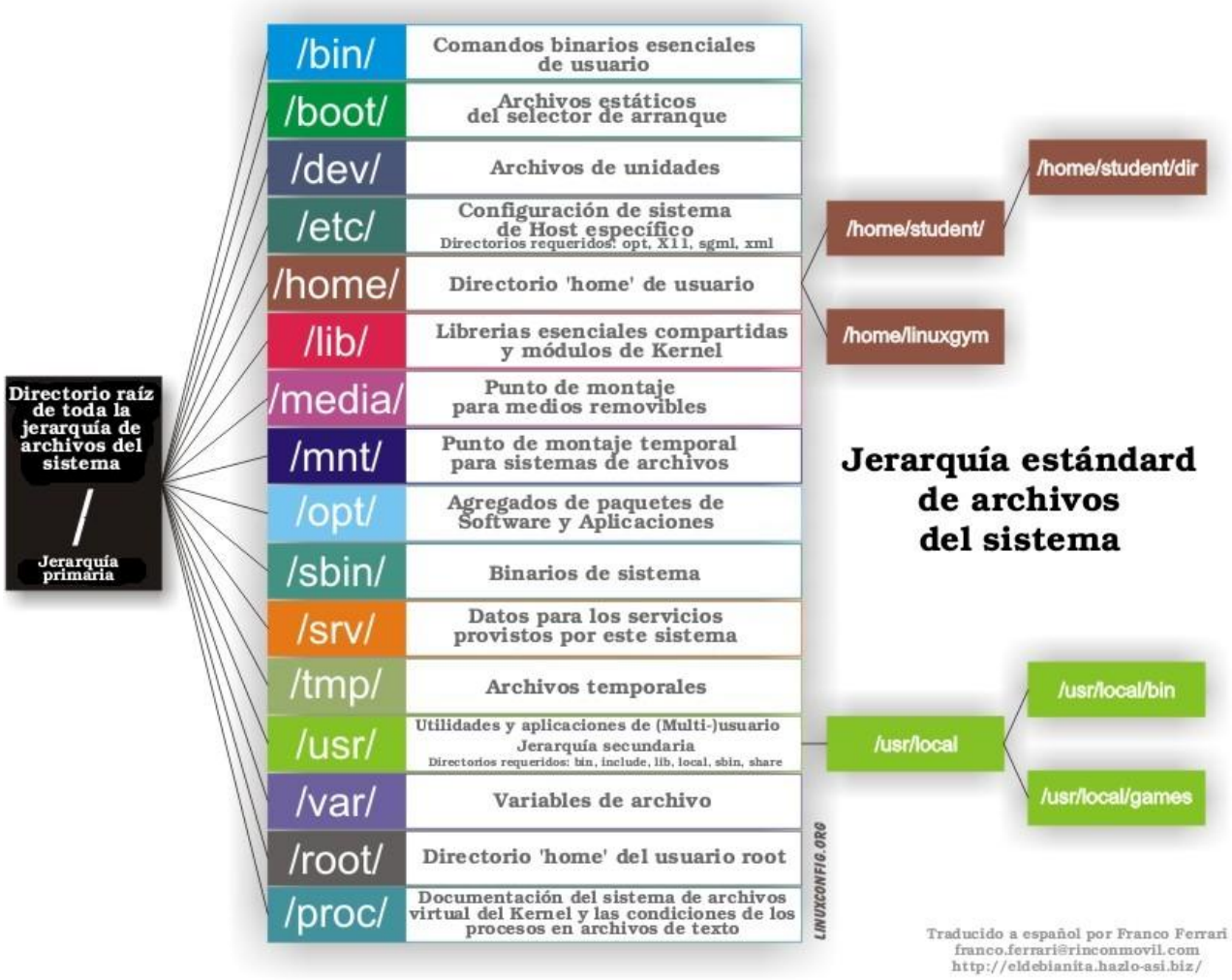


Gráfica 104. Directorios Jerárquicos. Fuente: Tanembaum (2009)

En consecuencia, se necesita una forma de agrupar los archivos relacionados. Por ejemplo, un profesor podría tener una colección de archivos que en conjunto formen un libro que está escribiendo para un curso, una segunda colección de archivos que contienen programas enviados por los estudiantes para otro curso, un tercer grupo de archivos que contenga el código de un sistema de escritura de compiladores avanzado que está construyendo, un cuarto grupo de archivos que contienen proposiciones de becas, así como otros archivos para correo electrónico, minutas de reuniones, artículos que está escribiendo, juegos, etcétera.

Lo que se necesita es una jerarquía (es decir, un árbol de directorios). Con este esquema, puede haber tantos directorios como se necesite para agrupar los archivos en formas naturales. Además, si varios usuarios comparten un servidor de archivos común, como se da el caso en muchas redes de empresas, cada usuario puede tener un directorio raíz privado para su propia jerarquía. Este esquema se muestra en la gráfica 104. Aquí, cada uno de los directorios A, B y C contenidos en el directorio raíz pertenecen a un usuario distinto, dos de los cuales han creado subdirectorios para proyectos en los que están trabajando.

La capacidad de los usuarios para crear un número arbitrario de subdirectorios provee una poderosa herramienta de estructuración para que los usuarios organicen su trabajo. Por esta razón, casi todos los sistemas de archivos modernos se organizan de esta manera.



Gráfica 105 . Jerarquía de directorios en GNU/Linux. Fuente: http:// eldebianita.hazlo-asi.biz

Nombres de rutas

Cuando el sistema de archivos está organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos. Por lo general se utilizan dos métodos distintos. En el primer método, cada archivo recibe un nombre de ruta absoluto que consiste en la ruta desde el directorio raíz al archivo. Como ejemplo, la ruta /usr/ast/mailbox significa que el directorio raíz contiene un subdirectorio llamado usr, que a su vez contiene un subdirectorio ast, el cual contiene el archivo *mailbox*. Los nombres de ruta absolutos siempre empiezan en el

directorio raíz y son únicos. En UNIX, los componentes de la ruta van separados por /. En Windows el separador es \. En MULTICS era >.

Sin importar cuál carácter se utilice, si el primer carácter del nombre de la ruta es el separador, entonces la ruta es absoluta.

El mismo nombre de ruta se escribiría de la siguiente manera en estos tres sistemas:

Windows \usr\ast\mailbox

UNIX /usr/ast/mailbox

MULTICS >usr>ast>mailbox

El otro tipo de nombre es el nombre de ruta relativa. Éste se utiliza en conjunto con el concepto del directorio de trabajo (también llamado directorio actual). Un usuario puede designar un directorio como el directorio de trabajo actual, en cuyo caso todos los nombres de las rutas que no empiecen en el directorio raíz se toman en forma relativa al directorio de trabajo. Por ejemplo, si el directorio de trabajo actual es /usr/ast, entonces el archivo cuya ruta absoluta sea /usr/ast/mailboxse puede referenciar simplemente como *mailbox*. En otras palabras, el comando de UNIX **cp /usr/ast/mailbox /usr/ast/mailbox.bak** y el comando **cp mailbox mailbox.bak** hacen exactamente lo mismo si el directorio de trabajo es /usr/ast. A menudo es más conveniente la forma relativa, pero hace lo mismo que la forma absoluta.

Operaciones de directorios

Las llamadas al sistema permitidas para administrar directorios exhiben más variación de un sistema a otro que las llamadas al sistema para los archivos. Para dar una impresión de lo que son y cómo funcionan, daremos un ejemplo (tomado de UNIX).

- Create. Se crea un directorio. Está vacío, excepto por punto y puntopunto, que el sistema coloca ahí de manera automática (o en unos cuantos casos lo hace el programa mkdir).
- Delete. Se elimina un directorio. Se puede eliminar sólo un directorio vacío. Un directorio que sólo contiene a punto y puntopunto se considera vacío, ya que por lo general éstos no se pueden eliminar.

- Opendir. Los directorios se pueden leer. Por ejemplo, para listar todos los archivos en un directorio, un programa de listado abre el directorio para leer los nombres de todos los archivos que contiene. Antes de poder leer un directorio se debe abrir, en forma análoga al proceso de abrir y leer un archivo.
- Closedir. Cuando se ha leído un directorio, se debe cerrar para liberar espacio en la tabla interna.
- Readdir. Esta llamada devuelve la siguiente entrada en un directorio abierto. Antes era posible leer directorios utilizando la llamada al sistema read común, pero ese método tiene la desventaja de forzar al programador a conocer y tratar con la estructura interna de los directorios. En contraste, readdir siempre devuelve una entrada en formato estándar, sin importar cuál de las posibles estructuras de directorio se utilice.
- Rename. En muchos aspectos, los directorios son sólo como archivos y se les puede cambiar el nombre de la misma forma que a los archivos.
- Link. La vinculación (ligado) es una técnica que permite a un archivo aparecer en más de un directorio. Esta llamada al sistema especifica un archivo existente y el nombre de una ruta, creando un vínculo desde el archivo existente hasta el nombre especificado por la ruta. De esta forma, el mismo archivo puede aparecer en varios directorios. A un vínculo de este tipo, que incrementa el contador en el nodo-i del archivo (para llevar la cuenta del número de entradas en el directorio que contienen el archivo), se le llama algunas veces vínculo duro(o liga dura).
- Unlink. Se elimina una entrada de directorio. Si el archivo que se va a desvincular sólo está presente en un directorio (el caso normal), se quita del sistema de archivos. Si está presente en varios directorios, se elimina sólo el nombre de ruta especificado. Los demás permanecen. En UNIX, la llamada al sistema para eliminar archivos (que vimos antes) es, de hecho, unlink.

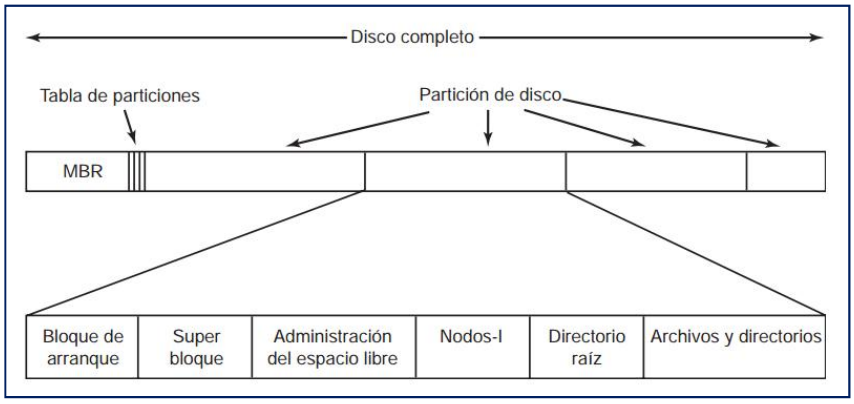
5.3. Implementación de Sistemas de Archivos

Distribución del sistema de archivos

Los sistemas de archivos se almacenan en discos. La mayoría de los discos se pueden dividir en una o más particiones, con sistemas de archivos independientes en cada partición. El sector 0 del disco se conoce como el MBR (*Master Boot Record*; Registro maestro de arranque) y se

utiliza para arrancar la computadora. El final del MBR contiene la tabla de particiones, la cual proporciona las direcciones de inicio y fin de cada partición. Una de las particiones en la tabla se marca como activa. Cuando se arranca la computadora, el BIOS lee y ejecuta el MBR. Lo primero que hace el programa MBR es localizar la partición activa, leer su primer bloque, conocido como bloque de arranque, y ejecutarlo. El programa en el bloque de arranque carga el sistema operativo contenido en esa partición. Por cuestión de uniformidad, cada partición inicia con un bloque de arranque no contenga un sistema operativo que se pueda arrancar. Además, podría contener uno en el futuro.

Además de empezar con un bloque de arranque, la distribución de una partición de disco varía mucho de un sistema de archivos a otro. A menudo el sistema de archivos contendrá algunos de los elementos que se muestran en la gráfica 106. El primero es el superbloque. Contiene todos los parámetros clave acerca del sistema de archivos y se lee en la memoria cuando se arranca la computadora o se entra en contacto con el sistema de archivos por primera vez. La información típica en el superbloque incluye un número mágico para identificar el tipo del sistema de archivos, el número de bloques que contiene el sistema de archivos y otra información administrativa clave.



Gráfica 106 Posible distribución de Sistema de Archivos. Fuente: Tanembaum(2009)

A continuación podría venir información acerca de los bloques libres en el sistema de archivos, por ejemplo en la forma de un mapa de bits o una lista de apuntadores. Ésta podría ir seguida de los nodos-i, un arreglo de estructuras de datos, uno por archivo, que indica todo acerca del archivo. Después de eso podría venir el directorio raíz, que contiene la parte superior del árbol

del sistema de archivos. Por último, el resto del disco contiene todos los otros directorios y archivos.

Implementación de archivos

Probablemente la cuestión más importante al implementar el almacenamiento de archivos sea mantener un registro acerca de qué bloques de disco van con cuál archivo. Se utilizan varios métodos en distintos sistemas operativos.

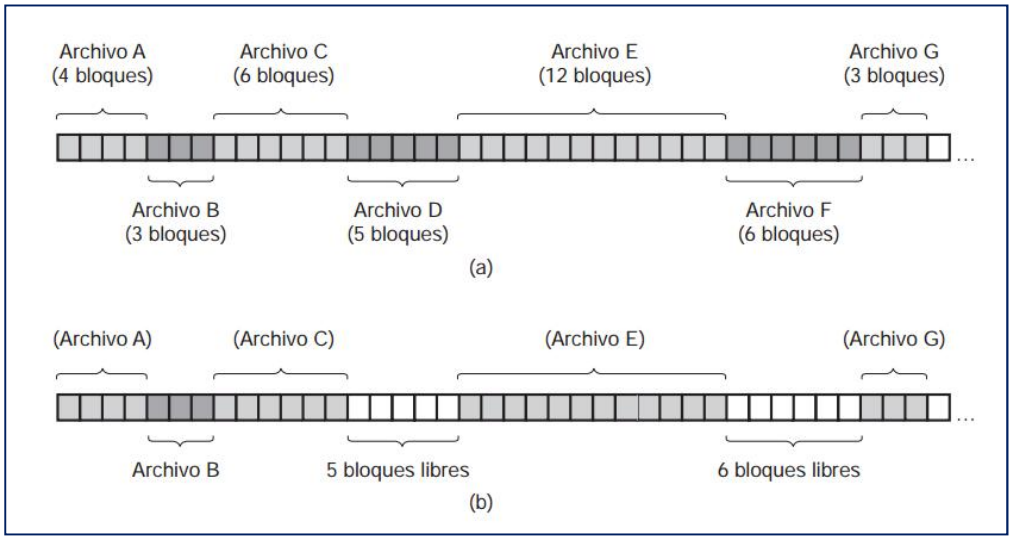
- **Asignación contigua**

  El esquema de asignación más simple es almacenar cada archivo como una serie contigua de bloques de disco. Así, en un disco con bloques de 1 KB, a un archivo de 50 KB se le asignarían 50 bloques consecutivos. Con bloques de 2 KB, se le asignarían 25 bloques consecutivos. En la gráfica 107 (a) podemos ver un ejemplo de asignación de almacenamiento contigua. Aquí se muestran los primeros 40 bloques de disco, empezando con el bloque 0, a la izquierda. Al principio el disco estaba vacío, después se escribió un archivo A de cuatro bloques de longitud al disco, empezando desde el principio (bloque 0). Posteriormente se escribió un archivo de seis bloques llamado B, empezando justo después del archivo A. Observe que cada archivo empieza al inicio de un nuevo bloque, por lo que si el archivo Afuera realmente de 3 ½ bloques, se desperdiciaría algo de espacio al final del último bloque. En la gráfica se muestra un total de siete archivos, cada uno empezando en el bloque que va después del final del archivo anterior. Se utiliza sombreado sólo para facilitar la distinción de cada archivo. No tiene un significado real en términos de almacenamiento. La asignación de espacio en disco contiguo tiene dos ventajas significativas.

  - En primer lugar es simple de implementar, ya que llevar un registro de la ubicación de los bloques de un archivo se reduce a recordar dos números: la dirección de disco del primer bloque y el número de bloques en el archivo. Dado el número del primer bloque, se puede encontrar el número de cualquier otro bloque con una simple suma.

- En segundo lugar, el rendimiento de lectura es excelente debido a que el archivo completo se puede leer del disco en una sola operación. Sólo se necesita una búsqueda (para el primer bloque).

Después de eso, no son necesarias más búsquedas ni retrasos por rotación, por lo que los datos llegan con el ancho de banda completa del disco. Por ende, la asignación contigua es simple de implementar y tiene un alto rendimiento.



Gráfica 107. Asignación contigua. Fuente: Tanembaum (2009)

Por desgracia, la asignación contigua también tiene una desventaja ligeramente significante: con el transcurso del tiempo, los discos se fragmentan. Para ver cómo ocurre esto, examine la gráfica 107 (b).

Aquí se han eliminado dos archivos, D y F. Cuando se quita un archivo, sus bloques se liberan naturalmente, dejando una serie de bloques libres en el disco. El disco no se compacta al momento para quitar el hueco, ya que eso implicaría tener que copiar todos los bloques que van después del hueco, que podrían ser millones. Como resultado, el disco al final consiste de archivos y huecos, como se ilustra en la gráfica.

Al principio esta fragmentación no es un problema, ya que cada nuevo archivo se puede escribir al final del disco, después del anterior. Sin embargo, en un momento dado el disco se llenará y será necesario compactarlo, lo cual es en extremo costoso o habrá que reutilizar el espacio libre de los huecos. Para reutilizar el espacio hay que mantener una lista de huecos, lo cual se puede hacer.

Sin embargo, cuando se cree un nuevo archivo será necesario conocer su tamaño final para poder elegir un hueco del tamaño correcto y colocarlo. Imagine las consecuencias de tal diseño. El usuario empieza un editor de texto o procesador de palabras para poder escribir un documento. Lo primero que pide el programa es cuántos bytes tendrá el documento final. Esta pregunta se debe responder o el programa no continuará. Si el número dado finalmente es demasiado pequeño, el programa tiene que terminar prematuramente debido a que el hueco de disco está lleno y no hay lugar para colocar el resto del archivo. Si el usuario trata de evitar este problema al proporcionar un número demasiado grande como tamaño final, por decir 100 MB, tal vez el editor no pueda encontrar un hueco tan grande y anuncie que el archivo no se puede crear. Desde luego que el usuario tiene la libertad de iniciar de nuevo el programa diciendo 50 MB esta vez y así en lo sucesivo hasta que se encuentre un hueco adecuado. Aún así, no es probable que este esquema haga que los usuarios estén felices.

Sin embargo, hay una situación en la que es factible la asignación contigua y de hecho, se utiliza ampliamente: en los CD-ROMs. Aquí todos los tamaños de los archivos se conocen de antemano y nunca cambiarán durante el uso subsiguiente del sistema de archivos del CD-ROM.

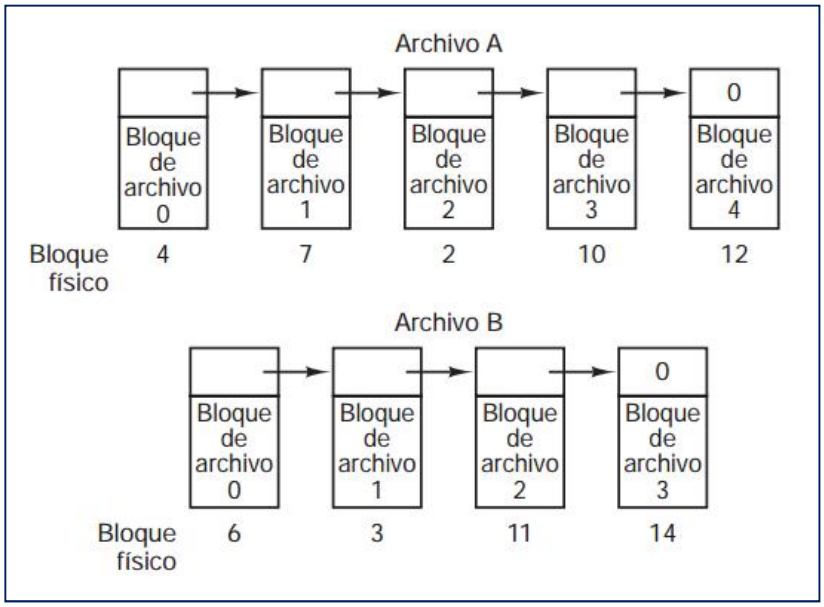
- **Asignación de lista enlazada** (ligada)

  El segundo método para almacenar archivos es mantener cada uno como una lista enlazada de bloques de disco, como se muestra en la gráfica 108. La primera palabra de cada bloque se utiliza como apuntador al siguiente. El resto del bloque es para los datos.

  A diferencia de la asignación contigua, en este método se puede utilizar cada bloque del disco. No se pierde espacio debido a la fragmentación del disco (excepto por la fragmentación interna en el último bloque). Además, para la entrada del directorio sólo le basta con almacenar la dirección de disco del primer bloque. El resto se puede encontrar a partir de ella.

  Por otro lado, aunque la lectura secuencial un archivo es directa, el acceso aleatorio es en extremo lento. Para llegar al bloque n, el sistema operativo tiene que empezar desde

el principio y leer los n– 1 bloques anteriores, uno a la vez. Es claro que tantas lecturas serán demasiado lentas.



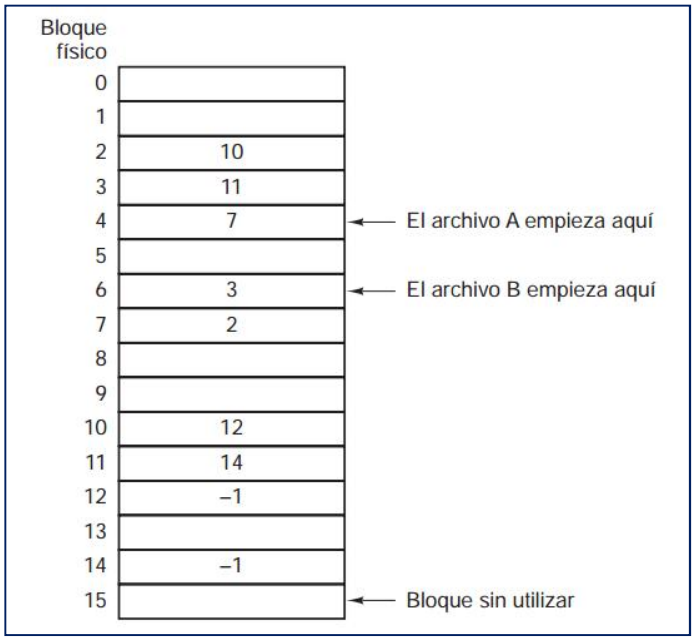
Gráfica 108. Lista enlazada. Fuente: Tanembaum (2009)

Además, la cantidad de almacenamiento de datos en un bloque ya no es una potencia de dos, debido a que el apuntador ocupa unos cuantos bytes. Aunque no es fatal, tener un tamaño peculiar es menos eficiente debido a que muchos programas leen y escriben en bloques, cuyo tamaño es una potencia de dos. Con los primeros bytes de cada bloque ocupados por un apuntador al siguiente bloque, leer el tamaño del bloque completo requiere adquirir y concatenar información de dos bloques de disco, lo cual genera un gasto adicional de procesamiento debido a la copia.

- **Asignación de lista enlazada utilizando una tabla en memoria**

  Ambas desventajas de la asignación de lista enlazada se pueden eliminar si tomamos la palabra del apuntador de cada bloque de disco y la colocamos en una tabla en memoria. La gráfica 109 muestra cuál es la apariencia de la tabla para el ejemplo de la figura 4-11. En ambas figuras tenemos dos archivos. El archivo A utiliza los bloques de disco 4, 7, 2, 10 y 12, en ese orden y el archivo B utiliza los bloques de disco 6, 3, 11 y 14, en ese orden. Utilizando la tabla de la gráfica 109, podemos empezar con el bloque 4 y seguir toda la cadena hasta el final. Lo mismo se puede hacer empezando con el bloque 6. Ambas cadenas se terminan con un marcador especial (por ejemplo, 1) que no sea un

número de bloque válido. Dicha tabla en memoria principal se conoce como FAT (*File Allocation Table*, Tabla de asignación de archivos).



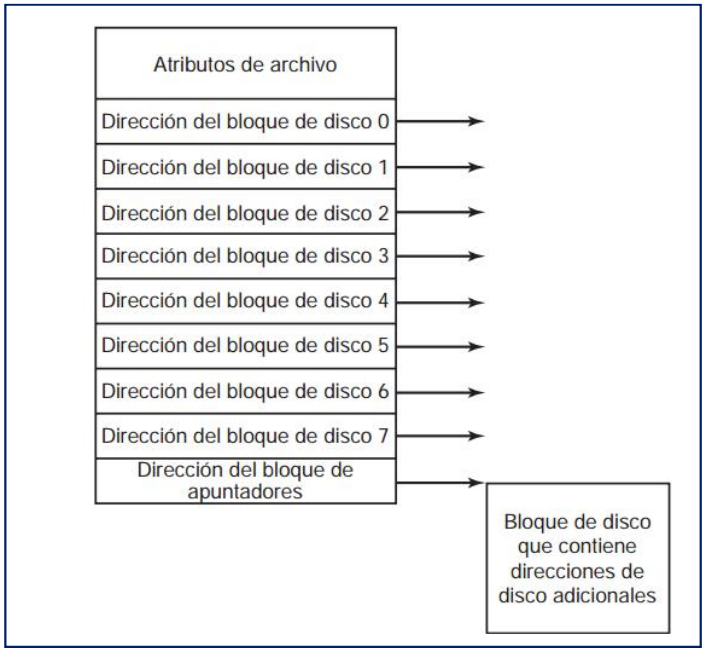
Gráfica 109. Lista enlazada con Tabla en Memoria. Fuente: Tanembaum (2009).

Utilizando esta organización, el bloque completo está disponible para los datos. Además, el acceso aleatorio es mucho más sencillo. Aunque aún se debe seguir la cadena para encontrar un desplazamiento dado dentro del archivo, la cadena está completamente en memoria y se puede seguir sin necesidad de hacer referencias al disco. Al igual que el método anterior, la entrada de directorio necesita mantener sólo un entero (el número de bloque inicial) y aún así puede localizar todos los bloques, sin importar qué tan grande sea el archivo.

La principal desventaja de este método es que toda la tabla debe estar en memoria todo el tiempo para que funcione. Con un disco de 200 GB y un tamaño de bloque de 1 KB, la tabla necesita 200 millones de entradas, una para cada uno de los 200 millones de bloques de disco. Cada entrada debe tener un mínimo de 3 bytes. Para que la búsqueda sea rápida, deben tener 4 bytes. Así, la tabla ocupará 600 MB u 800 MB de memoria principal todo el tiempo, dependiendo de si el sistema está optimizado para espacio o tiempo. Esto no es muy práctico. Es claro que la idea de la FAT no se escala muy bien en los discos grandes.

- **Nodos-i**

Nuestro último método para llevar un registro de qué bloques pertenecen a cuál archivo es asociar con cada archivo una estructura de datos conocida como nodo-i (nodo-índice), la cual lista los atributos y las direcciones de disco de los bloques del archivo. En la gráfica 110 se muestra un ejemplo simple. Dado el nodo-i, entonces es posible encontrar todos los bloques del archivo. La gran ventaja de este esquema, en comparación con los archivos vinculados que utilizan una tabla en memoria, es que el nodo-i necesita estar en memoria sólo cuando está abierto el archivo correspondiente. Si cada nodo-i ocupa n bytes y puede haber un máximo de k archivos abiertos a la vez, la memoria total ocupada por el arreglo que contiene los nodos-i para los archivos abiertos es de sólo kn bytes. Sólo hay que reservar este espacio por adelantado.



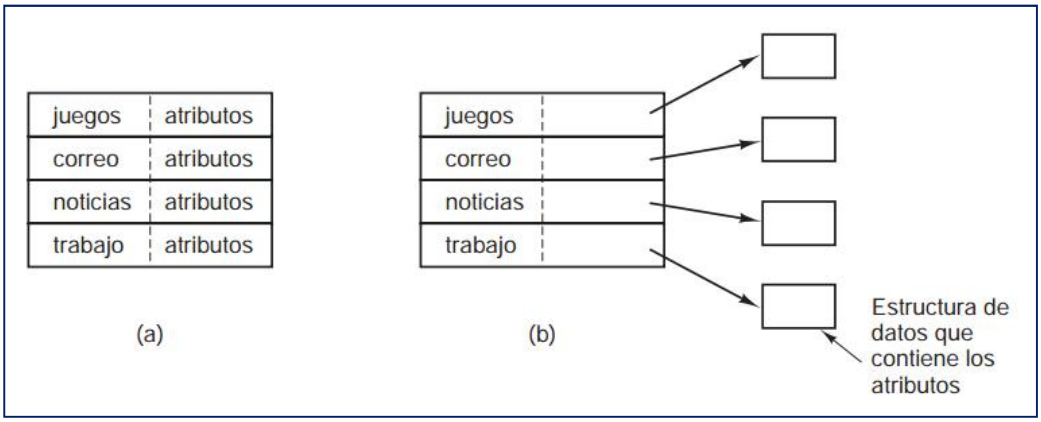
Gráfica 110. Nodo-i. Fuente: Tanembaum (2009)

Por lo general, este arreglo es mucho más pequeño que el espacio ocupado por la tabla de archivos descrita en la sección anterior. La razón es simple: la tabla para contener la lista enlazada de todos los bloques de disco es proporcional en tamaño al disco en sí. Si el disco tiene n bloques, la tabla necesita n entradas. A medida que aumenta el tamaño de los discos, esta tabla aumenta linealmente con ellos. En contraste, el esquema del nodo-i requiere un arreglo en memoria cuyo tamaño sea proporcional al número

máximo de archivos que pueden estar abiertos a la vez. No importa si el disco es de 10 GB, de 100 GB ó de 1000 GB. Un problema con los nodos-i es que si cada uno tiene espacio para un número fijo de direcciones de disco, ¿qué ocurre cuando un archivo crece más allá de este límite? Una solución es reservar la última dirección de disco no para un bloque de datos, sino para la dirección de un bloque que contenga más direcciones de bloques de disco, como se muestra en la gráfica 110. Algo aun más avanzado sería que dos o más de esos bloques contuvieran direcciones de disco o incluso bloques de disco apuntando a otros bloques de disco llenos de direcciones.

5.4. Implementación de Directorios

Antes de poder leer un archivo, éste debe abrirse. Cuando se abre un archivo, el sistema operativo utiliza el nombre de la ruta suministrado por el usuario para localizar la entrada de directorio. Esta entrada provee la información necesaria para encontrar los bloques de disco. Dependiendo del sistema, esta información puede ser la dirección de disco de todo el archivo (con asignación contigua), el número del primer bloque (ambos esquemas de lista enlazada) o el número del nodo-i. En todos los casos, la función principal del sistema de directorios es asociar el nombre ASCII del archivo a la información necesaria para localizar los datos.



Gráfica 111. (a) Directorio con Entradas Fijas (b) Directorio con i-nodos. Fuente: Tanembaum (2009).

Una cuestión muy relacionada es dónde deben almacenarse los atributos. Cada sistema de archivos mantiene atributos de archivo, como el propietario y la hora de creación de cada archivo, debiendo almacenarse en alguna parte. Una posibilidad obvia es almacenarlos directamente en la entrada de directorio. Muchos sistemas hacen eso. Esta opción se muestra en la gráfica 111 (a). En este diseño simple, un directorio consiste en una lista de entradas de

tamaño fijo, una por archivo, que contienen un nombre de archivo (de longitud fija), una estructura de los atributos del archivo y una o más direcciones de disco (hasta cierto máximo) que indique en dónde se encuentran los bloques de disco.

Para los sistemas que utilizan nodos-i, existe otra posibilidad para almacenar los atributos en los nodos-i, en vez de hacerlo en las entradas de directorio. En ese caso, la entrada de directorio puede ser más corta: sólo un nombre de archivo y un número de nodo-i. Este método se ilustra en la gráfica 111 (b). Este método tiene ciertas ventajas sobre el método de colocarlos en la entrada de directorio. Los dos esquemas que se muestran en la gráfica 111 corresponden a Windows y UNIX, respectivamente.

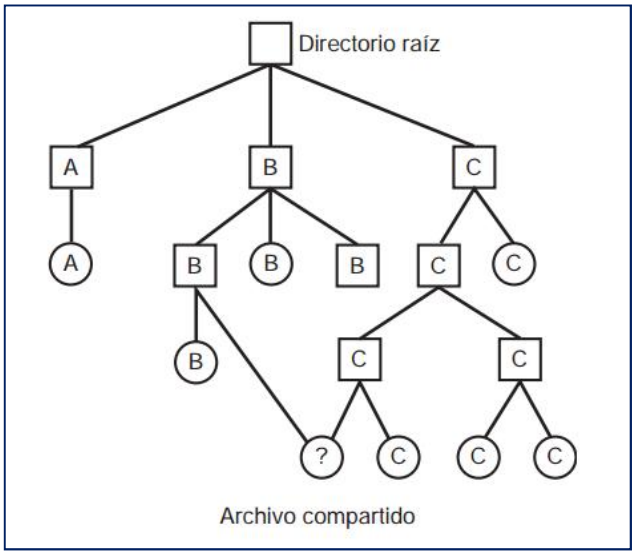
Hasta ahora hemos hecho la suposición de que los archivos tienen nombres cortos con longitud fija. En MS-DOS, los archivos tienen un nombre base de 1 a 8 caracteres y una extensión opcional de 1 a 3 caracteres. En UNIX Version 7, los nombres de los archivos eran de 1 a 14 caracteres, incluyendo cualquier extensión. Sin embargo, casi todos los sistemas operativos modernos aceptan nombres de archivos más largos, con longitud variable. ¿Cómo se pueden implementar éstos?

El esquema más simple es establecer un límite en la longitud del nombre de archivo, que por lo general es de 255 caracteres y después utilizar uno de los diseños de la gráfica 111 con 255 caracteres reservados para cada nombre de archivo. Este esquema es simple, pero desperdicia mucho espacio de directorio, ya que pocos archivos tienen nombres tan largos. Por cuestiones de eficiencia, es deseable una estructura distinta.

5.5. Archivos compartidos

Cuando hay varios usuarios trabajando en conjunto en un proyecto, a menudo necesitan compartir archivos. Como resultado, con frecuencia es conveniente que aparezca un archivo compartido en forma simultánea en distintos directorios que pertenezcan a distintos usuarios. La gráfica 112 muestra el sistema de archivos de la gráfica 104 de nuevo, sólo que con uno de los archivos de C ahora presentes en uno de los directorios de B también. La conexión entre el directorio de B y el archivo compartido se conoce como un vínculo (liga). El sistema de archivos en sí es ahora un Gráfico acíclico dirigido (*Directed Acyclic Graph*, DAG) en vez de un árbol.

Compartir archivos es conveniente, pero también introduce ciertos problemas. Para empezar, si los directorios en realidad contienen direcciones de disco, entonces habrá que realizar una copia de las direcciones de disco en el directorio de B cuando se ligue el archivo. Si B o C agregan posteriormente al archivo, los nuevos bloques se listarán sólo en el directorio del usuario que agregó los datos. Los cambios no estarán visibles para el otro usuario, con lo cual fracasa el propósito de la compartición.



Gráfica 112, Sistema de Archivos con Archivo compartido. Fuente: Tanembaum (2009)

Este problema se puede resolver de dos formas. En la primera solución, los bloques de disco no se listan en los directorios, sino en una pequeña estructura de datos asociada con el archivo en sí. Entonces, los directorios apuntarían sólo a la pequeña estructura de datos. Éste es el esquema que se utiliza en UNIX (donde la pequeña estructura de datos es el nodo-i).

En la segunda solución, B se vincula a uno de los archivos de C haciendo que el sistema cree un archivo, de tipo LINK e introduciendo ese archivo en el directorio de B. El nuevo archivo contiene sólo el nombre de la ruta del archivo al cual está vinculado. Cuando B lee del archivo vinculado, el sistema operativo ve que el archivo del que se están leyendo datos es de tipo LINK, busca el nombre del archivo y lee el archivo. A este esquema se le conoce como vínculo simbólico (liga simbólica), para contrastarlo con el tradicional vínculo (duro).

5.6. Sistema de Archivos por Bitácora

Aunque los sistemas de archivos estructurados por registro son una idea interesante, no se utilizan ampliamente, debido en parte a su alta incompatibilidad con los sistemas de archivos existentes. Sin embargo, una de las ideas inherentes en ellos, la robustez frente a las fallas, se puede aplicar con facilidad a sistemas de archivos más convencionales. La idea básica aquí es mantener un registro de lo que va a realizar el sistema de archivos antes de hacerlo, por lo que si el sistema falla antes de poder realizar su trabajo planeado, al momento de re-arrancar el

sistema puede buscar en el registro para ver lo que estaba ocurriendo al momento de la falla y terminar el trabajo. Dichos sistemas de archivos, conocidos como sistemas de archivos por bitácora (*Journaling files system*, JFS), se encuentran en uso actualmente. El sistema de archivos NTFS de Microsoft, así como los sistemas ext3 y ReiserFS de Linux son todos por bitácora.

Para ver la naturaleza del problema, considere una operación simple que ocurre todo el tiempo: remover un archivo. Esta operación (en UNIX) requiere tres pasos:

1. Quitar el archivo de su directorio.
2. Liberar el nodo-i y pasarlo a la reserva de nodos-i libres.
3. Devolver todos los bloques de disco a la reserva de bloques de disco libres.

En Windows se requieren pasos similares. En la ausencia de fallas del sistema, el orden en el que se realizan estos pasos no importa; en la presencia de fallas, sí. Suponga que se completa el primer paso y después el sistema falla. El nodo-i y los bloques de archivo no estarán accesibles desde ningún archivo, pero tampoco estarán disponibles para ser reasignados; sólo se encuentran en alguna parte del limbo, disminuyendo los recursos disponibles. Si la falla ocurre después del siguiente paso, sólo se pierden los bloques.

Si el orden de las operaciones se cambia y el nodo-i se libera primero, entonces después de rearrancar, el nodo-i se puede reasignar pero la entrada de directorio anterior seguirá apuntando a él y por ende al archivo incorrecto. Si los bloques se liberan primero, entonces una falla antes de limpiar el nodo-i indicará que una entrada de directorio válida apunta a un nodo-i que lista los bloques que ahora se encuentran en la reserva de almacenamiento libre y que probablemente se reutilicen en breve, produciendo dos o más archivos que compartan al azar los mismos bloques. Ninguno de estos resultados es bueno.

Lo que hace el sistema de archivos por bitácora es escribir primero una entrada de registro que liste las tres acciones a completar. Después la entrada de registro se escribe en el disco (y como

buena medida, posiblemente se lea otra vez del disco para verificar su integridad). Sólo hasta que se ha escrito la entrada de registro es cuando empiezan las diversas operaciones. Una vez que las operaciones se completan con éxito, se borra la entrada de registro. Si ahora el sistema falla, al momento de recuperarse el sistema de archivos puede verificar el registro para ver si había operaciones pendientes. De ser así, todas ellas se pueden volver a ejecutar (múltiples veces, en caso de fallas repetidas) hasta que el archivo se elimine en forma correcta.

Para que funcione el sistema por bitácora, las operaciones registradas deben ser idempotentes, lo cual significa que pueden repetirse todas las veces que sea necesario sin peligro. Las operaciones como “Actualizar el mapa de bits para marcar el nodo-i k o el bloque n como libre” se pueden repetir hasta que las todas las operaciones se completen sin peligro. De manera similar, las operaciones de buscar en un directorio y eliminar una entrada llamada foobar también son idempotentes.

Por otro lado, la operación de agregar los bloques recién liberados del nodo-i K al final de la lista libre no es idempotente, debido a que éstos tal vez ya se encuentren ahí. La operación más costosa “Buscar en la lista de bloques libres y agregarle el bloque n si no está ya presente”, también es idempotente. Los sistemas de archivos por bitácora tienen que organizar sus estructuras de datos y operaciones que pueden registrarse, de manera que todas ellas sean idempotentes. Bajo estas condiciones, la recuperación de errores puede ser rápida y segura.

Para una mayor confiabilidad, un sistema de archivos puede introducir el concepto de las bases de datos conocido como transacción atómica. Cuando se utiliza este concepto, varias acciones se pueden agrupar mediante las operaciones *begin transaction* y *end transaction*. Así, el sistema de archivos sabe que debe completar todas las operaciones agrupadas o ninguna de ellas, pero ninguna otra combinación.

5.7. Complemento: Software Libre

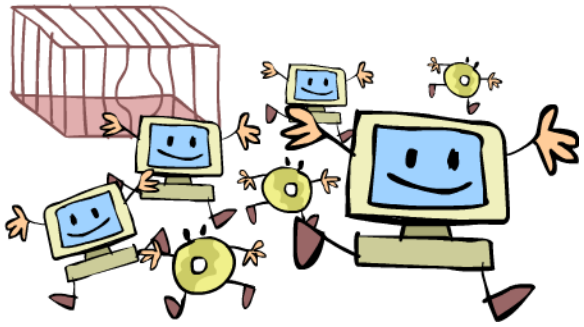
Concepto

El software libre (en inglés "free software", aunque esta denominación a veces se confunde con "gratis" por la ambigüedad del término "free" en el idioma inglés, por lo que también se usa "libre software") es la denominación del software que respeta la libertad de todos los usuarios que adquirieron el producto y, por tanto, una vez obtenido el mismo puede ser usado, copiado, estudiado, modificado, y redistribuido libremente de varias formas. Según la *Free Software Foundation*, el software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, y estudiar el mismo, e incluso modificar el software y distribuirlo modificado.

El software libre suele estar disponible gratuitamente, o al precio de costo de la distribución a través de otros medios; sin embargo no es obligatorio que sea así, por lo tanto no hay que asociar software libre a "software gratuito" (denominado usualmente *freeware*), ya que, conservando su carácter de libre, puede ser distribuido comercialmente ("software comercial"). Análogamente, el "software gratis" o "gratuito" incluye en ocasiones el código fuente; no obstante, este tipo de software no es libre en el mismo sentido que el software libre, a menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa.

Tampoco debe confundirse software libre con "software de dominio público". Éste último es aquel software que no requiere de licencia, pues sus derechos de explotación son para toda la humanidad, porque pertenece a todos por igual. Cualquiera puede hacer uso de él, siempre con fines legales y consignando su autoría original. Este software sería aquel cuyo autor lo dona a la humanidad o cuyos derechos de autor han expirado, tras un plazo contado desde la muerte de éste, habitualmente 70 años. Si un autor condiciona su uso bajo una licencia, por muy débil que sea, ya no es del dominio público[18]

Libertades del Software Libre:



- Libertad 0: Ejecutar el programa con cualquier propósito (privado, educativo, público, comercial, militar, etc.)
- Libertad 1: Estudiar y modificar el programa (para lo cual es necesario poder acceder al código fuente)
- Libertad 2: Copiar el programa de manera que se pueda ayudar al vecino o a cualquiera
- Libertad 3: Mejorar el programa y publicar las mejoras[19]

Ventajas Software Propietario

El Software Propietario se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), o cuyo código fuente no está disponible o el acceso a éste se encuentra restringido.

Un software sigue siendo no libre aún si el código fuente es hecho público, cuando se mantiene la reserva de derechos sobre el uso, modificación o distribución (por ejemplo, la versión comercial de SSH o el programa de licencias *shared source* de Microsoft)[20].

Ventajas:

- Propiedad y decisión de uso del software por parte de la empresa
- Soporte para todo tipo de hardware
- Mejor acabado de la mayoría de aplicaciones
- Las aplicaciones número uno son propietarias
- El ocio para ordenadores personales está destinado al mercado propietario
- Menor necesidad de técnicos especializados
- Mayor mercado laboral actual
- Mejor protección de las obras con copyright

- Unificación de productos

Ventajas Software Libre

- Económico
- Libertad de uso y redistribución
- Independencia tecnológica
- Fomento de la libre competencia al basarse en servicios y no licencias
- Soporte y compatibilidad a largo plazo
- Formatos estándar
- Sistemas sin puertas traseras y más seguros
- Corrección más rápida y eficiente de fallos
- Métodos simples y unificados de gestión de software
- Sistema en expansión

5.8.

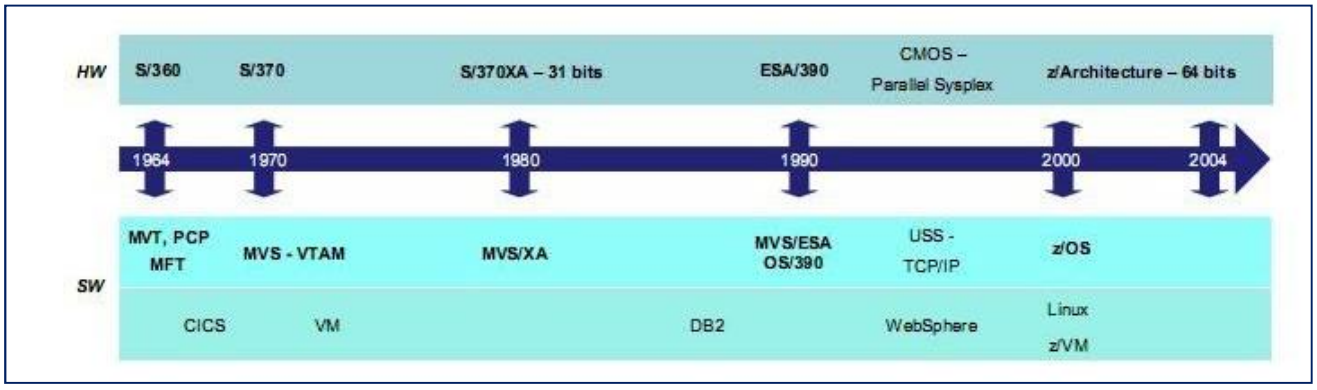
6. Sistemas Operativos Modernos

A continuación, se hará un breve resumen de los Sistemas Operativos más usados hoy en día, de acuerdo al Hardware que sirve.

6.1. Sistemas Operativos para Macrocomputadores

Gráfica 113 Mainframe

Un Mainframe o Macrocomputador es un sistema de computación utilizado en negocios para almacenar bases de datos comerciales, servidores de transacciones y aplicaciones, que requieren alta seguridad y disponibilidad que normalmente no se encuentra en máquinas de menor escala. El poder de un mainframe provee velocidad y capacidad de computación, permitiéndole desarrollar grandes volúmenes de procesamiento. Un mainframe puede procesar grandes cantidades de tareas de diferentes tipos y en distintas zonas horarias. Se debe tener en cuenta que la mayoría de las compañías de Fortune 1000 usan mainframes, y que el 60% de la información disponible en Internet está almacenada en computadoras mainframe.



Gráfica 114 Evolución de HW y SW en Mainframes. http://imageshack.us/photo/my-images/401/arquitec.jpg/

Gráfica 115 System z de IBM. http://www-03.ibm.com

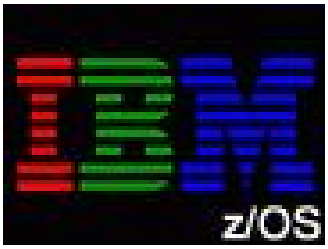
En los últimos años, la empresa IBM ha sido la que ha llevado la batuta en venta de equipos Mainframe, y sus respectivos Sistemas Operativos. Los Mainframes más usados son los de la familia de System z de IBM.

Entre los más usados hoy en día se pueden encontrar:

IBM i es un Sistema Operativo basado en EBCDIC que corre en IBM Systems Power y en IBM PureSystems. Es la actual versión del sistema operativo anteriormente denominado i5/OS y originalmente OS/400 cuando se introdujo en los computadores AS/400 en 1988[21].

Linux para System z es el término colectivo para el sistema operativo Linux compilado para correr en Mainframes de IBM, especialmente en máquinas de la familia de System z. Otras denominaciones incluyen Linux en zEntreprise 196, Linux en System z9, Linux en System z10, z/Linux, zLinux, etc.[22]

Unix (registrado oficialmente como UNIX®) es un sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969, por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie y Douglas McIlroy. El sistema, junto con todos los derechos fueron vendidos por AT&T a Novell, Inc. Esta vendió posteriormente el software a Santa Cruz Operation en 1995, y esta, a su vez, lo revendió a Caldera Software en 2001, empresa que después se convirtió en el grupo SCO. En 2010, y tras una larga batalla legal, ésta ha pasado nuevamente a ser propiedad de Novell[23].



z/OS es el sistema operativo actual de las computadoras centrales de IBM. Del sistema MVT (de 1967) se pasó al MVS en 1974 añadiéndole múltiples espacios de memoria virtual, agregándole a éste compatibilidad UNIX se pasó al OS/390 en 1995, y ampliando a éste el direccionamiento de 64 bits se pasó a z/OS en el año 2000.[24]

z/TPF es un sistema operative en tiempo real, de la empresa IBM para equipos mainframe, para las familias System z9 y zSeries. TPF deriva de "*Transaction Processing Facility*" (Lugar para procesar transacciones). z/TPF tiene un procesamiento rápido y de alto volumen, para manejar grandes cantidades de transacciones a través de redes distribuidas. El sistema TPF más avanzado puede procesar 10.000 transacciones por segundo.[25]

z/VM es la actual versión de la familia de Sistemas Operativos de máquinas virtuales. z/VM fue lanzado al público en octubre 2000 y permanece en uso activo y desarrollo hasta la fecha (diciembre 2013). Está basado en la tecnología y conceptos de los años 1960's, de los sistemas operatives CP/CMS de IBM, sobre el System/360-67. x/VM corre sobre la familia de computadores System z de IBM[26].

z/VSE (*Virtual Storage Extended* – Almacenamiento Virtual extendido) es un sistema operativo para los mainframes IBM, derivado del DOS/360. Es menos usado que el z/OS, y casi siempre en equipos más pequeños.[27]

6.2. Sistemas Operativos para Servidores

Un servidor es

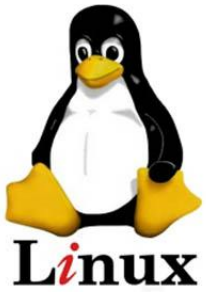
> Una computadora en la que se ejecuta un programa que realiza alguna tarea en beneficio de otras aplicaciones llamadas clientes, tanto si se trata de un ordenador central (mainframe), un miniordenador, una computadora personal, una PDA o un sistema embebido; sin embargo, hay

computadoras destinadas únicamente a proveer los servicios de estos programas: estos son los servidores por antonomasia.

Un servidor no es necesariamente una máquina de última generación de grandes proporciones, no es necesariamente un superordenador; un servidor puede ser desde una computadora de bajo recursos, hasta una máquina sumamente potente (ej.: servidores web, bases de datos grandes, etc. Procesadores especiales y hasta varios terabytes de memoria). Todo esto depende del uso que se le dé al servidor[28].

A continuación, algunos sistemas operativos usados en servidores:

FreeBSD es un sistema operativo libre para computadoras basado en las CPU de arquitectura Intel, incluyendo procesadores Intel 80386, Intel 80486 (versiones SX y DX), y Pentium. También funciona en procesadores compatibles con Intel como AMD y Cyrix. Actualmente también es posible utilizarlo hasta en once arquitecturas distintas como Alpha, AMD64, IA-64, MIPS, PowerPC y UltraSPARC. FreeBSD está basado en la versión 4.4 BSD-Lite del Computer Systems Research Group (CSRG) de la University of California, Berkeley siguiendo la tradición que ha distinguido el desarrollo de los sistemas BSD[29].



Linux es un núcleo libre de sistema operativo (también suele referirse al núcleo como kernel) basado en Unix. Es uno de los principales ejemplos de software libre y de código abierto. Linux está licenciado bajo la GPL v2 y está desarrollado por colaboradores de todo el mundo. El núcleo Linux fue concebido por el entonces estudiante de ciencias de la computación finlandés Linus Torvalds en 1991. Normalmente Linux se utiliza junto a un empaquetado de software, llamado distribución GNU/Linux y servidores.[30]

Mac OS X Server es un sistema operativo para servidores desarrollado por Apple Inc. basado en Unix. Es idéntico a su versión de escritorio, pero incluye además herramientas administrativas gráficas para la gestión de usuarios, redes, y servicios de red como LDAP, Servidor de correo, Servidor Samba, DNS, entre otros. También incorpora en sus versiones más recientes un número adicional de servicios y herramientas para configurarlos, tales como Servidor web, herramientas para crear una Wiki, Servidor iChat, y otros más[31].

Microsoft Servers (anteriormente llamado Windows Server System) es una marca que abarca una línea de productos de servidor de Microsoft. Esto incluye las ediciones de servidor de Microsoft Windows su propio sistema operativo, así como productos dirigidos al mercado más amplio de negocio. Algunas versiones: Windows 2000 Server, Windows Server 2003, Windows Server 2008, Windows HPC Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Small Business Server, Windows Essential Business Server, Windows Home Server[32].[33]

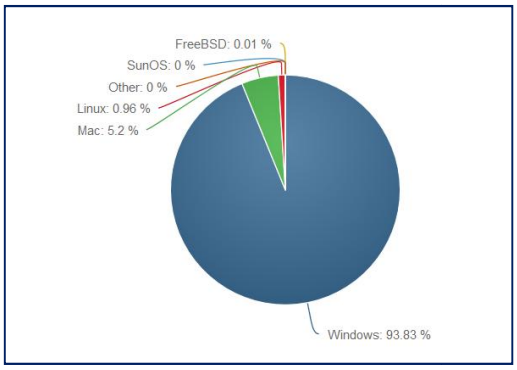
Novell Netware es un sistema operativo. Es una de las plataformas de servicio para ofrecer acceso a la red y los recursos de información, sobre todo en cuanto a servidores de archivos. Aunque el producto Windows de Microsoft nunca soportó una comparación con Netware, el retiro en 1995 de Ray Noorda junto al escaso marketing de Novell hicieron que el producto perdiera mercado, aunque no vigencia por lo que se ha anunciado soporte sobre este sistema operativo hasta el año 2015, por lo menos[34].

Solaris es un sistema operativo de tipo Unix desarrollado desde 1992 inicialmente por Sun Microsystems y actualmente por Oracle Corporation como sucesor de SunOS. Es un sistema certificado oficialmente como versión de Unix. Funciona en arquitecturas SPARC y x86 para servidores y estaciones de trabajo.[35]

Unix (registrado oficialmente como UNIX®) es un sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969, por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie y Douglas McIlroy. El sistema, junto con todos los derechos fueron vendidos por AT&T a Novell, Inc. Esta vendió posteriormente el software a Santa Cruz Operation en 1995, y esta, a su vez, lo revendió a Caldera Software en 2001, empresa que después se convirtió en el grupo SCO. En 2010, y tras una larga batalla legal, ésta ha pasado nuevamente a ser propiedad de Novell[36].

Windows NT es una familia de sistemas operativos producidos por Microsoft, de la cual la primera versión fue publicada en julio de 1993. Previamente a la aparición del Windows 95 la empresa Microsoft concibió una nueva línea de sistemas operativos orientados a estaciones de trabajo y servidor de red. Un sistema operativo con interfaz gráfica propia, estable y con características similares a los sistemas de red UNIX. Las letras NT provienen de la designación del producto como "Tecnología Nueva" (New Technology). Windows NT se distribuía en dos versiones, dependiendo de la utilidad que se le fuera a dar: Workstation para ser utilizado como estación de trabajo y Server para ser utilizado como servidor.[37]

6.3. Sistemas Operativos para Microcomputadores



Gráfica 116 Participación en el mercado de Sistemas Operativos de Escritorio. Diciembre 2013. http://www.netmarketshare.com/#

Cuando se revisan los Sistemas Operativos para Microcomputadores, Computadoras Personales, Laptops, y en algunos Netbooks, se consigue que casi un 94% de ellos son MS Windows, en sus distintas versiones.

Ya para el año 2014, **Windows** continúa dominando el mercado, pero su participación disminuyó a un 91,56%, **Mac OS X** con 7,11% y **Linux** con 1,34%.

A continuación, algunos Sistemas Operativos para Escritorio, o para Microcomputadores:

Linux es un núcleo libre de sistema operativo (también suele referirse al núcleo como kernel) basado en Unix. Es uno de los principales ejemplos de software libre y de código abierto. Linux está licenciado bajo la GPL v2 y está desarrollado por colaboradores de todo el mundo. El núcleo Linux fue concebido por el entonces estudiante de ciencias de la computación finlandés Linus Torvalds en 1991. Normalmente Linux se utiliza junto a un empaquetado de software, llamado distribución GNU/Linux y servidores.[38]

Una distribución Linux (distro) es un conjunto de software acompañado del núcleo Linux que se enfoca a satisfacer las necesidades de un grupo específico de usuarios. De este modo hay distribuciones para hogares, empresas y servidores.

DISTROS BASADAS EN DEBIAN:

- BACK TRACK, CANAIMA, ELIVE, GNEWSENSE, KNOPPIX, KUBUNTU, LINEX, LINUX MINT, MEPIS, UBUNTU, WEBCONVERGER, XBMC

DISTROS BASADAS EN GENTOO:

- FLASH LINUX, GESTOOX, JOLLIX, PENTOO, SABAYON, UTUTO, VIDALINUX, ZYNOT

DISTROS BASADAS EN RED HAT:

- CLEAR OS, FEDORA, RED HAT, CENTOS, MANDRAKE (MANDRIVA BASADA EN MANDRAKE), ORACLE ENTERPRISE

DISTROS BASADAS EN SUSE:

- JAVA DESKTOP SYSTEM, KURISO OS, OPEN SUSE, STRESSLINUX, SUSE

Mac OS (del inglés *Macintosh Operating System*, en español Sistema Operativo de Macintosh) es el nombre del sistema operativo creado por Apple para su línea de computadoras Macintosh. Es conocido por haber sido uno de los primeros sistemas dirigidos al gran público en contar con una interfaz gráfica compuesta por la interacción del mouse con ventanas, Icono y menús.[39] La versión actual es el OS X (Sistema Operativo X ó 10).

MS-DOS (siglas de *MicroSoft Disk Operating System*, Sistema operativo de disco de Microsoft) es un sistema operativo para computadoras basados en x86. Fue el miembro más popular de la familia de sistemas operativos DOS de Microsoft, y el principal sistema para computadoras personales compatible con IBM PC en la década de 1980 y mediados de 1990, hasta que fue sustituida gradualmente por sistemas operativos que ofrecían una interfaz gráfica de usuario, en particular por varias generaciones de Microsoft Windows.[40]

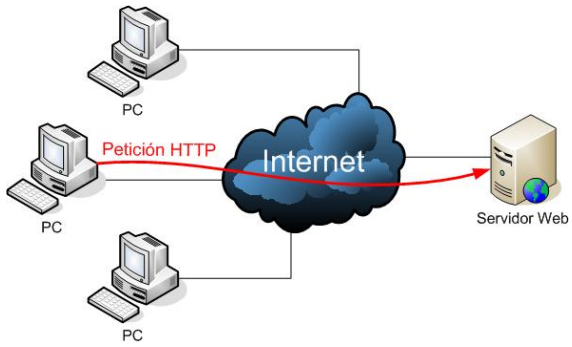
Microsoft Windows (conocido generalmente como Windows), es el nombre de una familia de sistemas operativos desarrollados y vendidos por Microsoft. Microsoft introdujo un entorno operativo denominado Windows el 25 de noviembre de 1985 como un complemento para MS-DOS en respuesta al creciente interés en las interfaces gráficas de usuario (GUI).[41]

Microsoft Windows tiene actualmente en el mercado varias versiones, con su respectiva participación de mercado:

- MS Windows 7 (2009) → 48%
- MS Windows 8 (2012) → 10%
- MS Windows Vista (2007) → 4%
- MS Windows XP (2001) → 28%
- Otros → 10%

6.4. Sistemas Operativos para Servidores Web

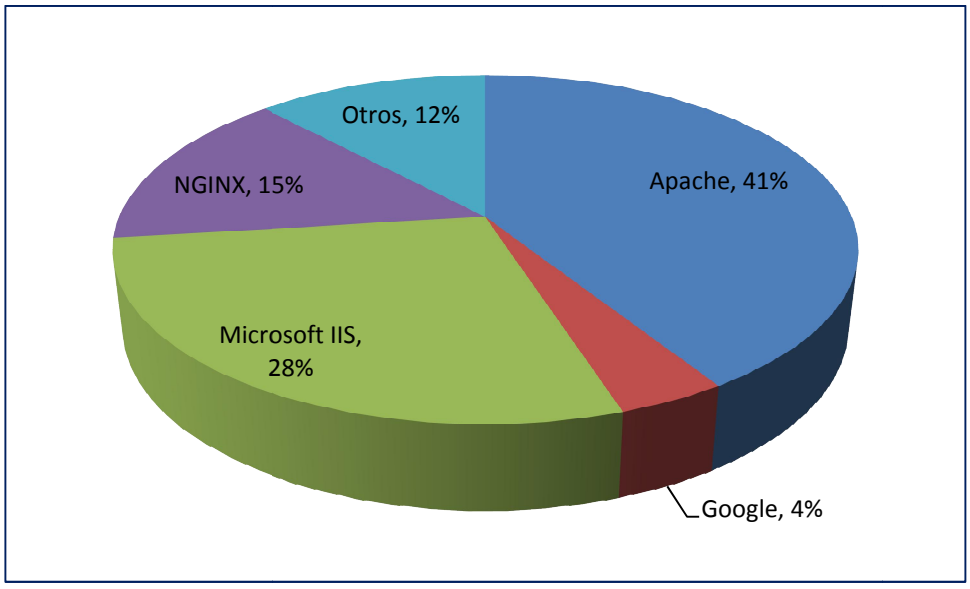
Un servidor web es



Gráfica 117 Servidor Web. http://ociotec.com/wp-content/uploads/2008/11/tunel-ssh-arquitectura-esperada.png

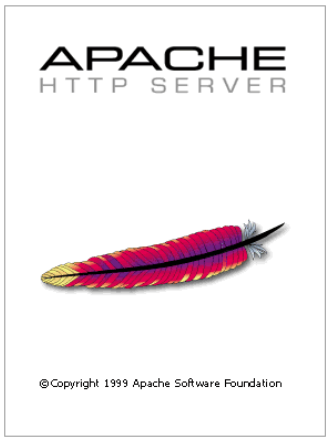
"un programa que sirve datos en forma de Páginas Web, hipertextos o páginas HTML (HyperText Markup Language): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de sonidos. La comunicación de estos datos entre cliente y servidor se hace por medio un protocolo, concretamente del protocolo Http. Con esto, un servidor Web se mantiene a la espera de peticiones HTTP, que son ejecutadas por un cliente HTTP; lo que solemos conocer como un Navegador Web. A modo de ejemplo: al teclear (http://www.cnice.mec.es) en un navegador, éste realizará una petición HTTP al servidor que tiene asociada dicha URL."[42]

De acuerdo a Netcraft, la participación de los Servidores Web en el mercado es:



Gráfica 118 Participación en el Mercado de Servidores Web. http://news.netcraft.com/archives/category/web-server-survey/ Diciembre 2013. Fuente: elaboración propia.

Para diciembre del 2014, los números quedaron liderados de nuevo por Apache con un 39%, Microsoft con 30%, Nginx con 14% y Google con 2%.



El servidor HTTP **Apache** es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo. Su nombre se debe a que Behelendorf quería que tuviese la connotación de algo que es firme y enérgico pero no agresivo, y la tribu Apache fue la última en rendirse al que pronto se convertiría en gobierno de EEUU, y en esos momentos la preocupación de su grupo era que llegasen las empresas y "civilizasen" el paisaje que habían creado los primeros ingenieros de internet.[43]

Cherokee es un servidor web multiplataforma. Su objetivo es ser rápido y completamente funcional, sin dejar de ser liviano comparado con otros servidores web. Está escrito completamente en C. Puede usarse como un sistema embebido y soporta complementos para aumentar sus funcionalidades. Es software libre, disponible bajo la Licencia Pública General de GNU[44].

Google Web Server (GWS) es el nombre del servidor web que utiliza Google en sus infraestructuras y servidores. Google es intencionadamente vago acerca de GWS, simplemente se limitó a decir que es un servidor personalizado de desarrollo propio que se ejecuta en sistemas UNIX como GNU/Linux. Adicionalmente existen especulaciones sobre que GWS es una versión modificada y adaptada de Apache HTTP Server que Google utiliza para su propia explotación.[45]

Internet Information Services (IIS) es un servidor web y un conjunto de servicios para el sistema operativo Microsoft Windows. Originalmente era parte del Option Pack para Windows NT. Luego fue integrado en otros sistemas operativos de Microsoft destinados a ofrecer servicios, como Windows 2000 o Windows Server 2003. Los servicios que ofrece son: FTP, SMTP, NNTP y HTTP/HTTPS. Este servicio convierte a una PC en un servidor web para Internet o una intranet, es decir que en las computadoras que tienen este servicio instalado se pueden publicar páginas web tanto local como remotamente.[46]

nginx (pronunciado en inglés "engine X") es un servidor web/proxy inverso ligero de alto rendimiento y un proxy para protocolos de correo electrónico (IMAP/POP3). Es software libre y de código abierto, licenciado bajo la Licencia BSD simplificada. Es multiplataforma, por lo que corre en sistemas tipo Unix (GNU/Linux, BSD, Solaris, Mac OS X, etc.) y Windows. El sistema es usado por una larga lista de sitios web conocidos, como: WordPress, Hulu, GitHub, Ohloh, SourceForge, TorrentReactor y partes de Facebook (como el servidor de descarga de archivos zip pesados).[47]

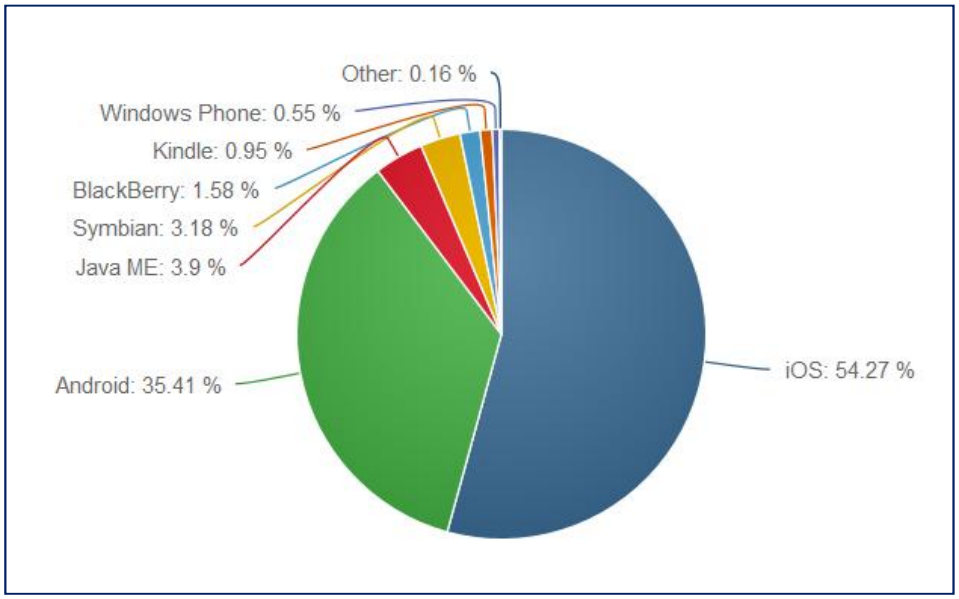
Oracle WebLogic es un servidor de aplicaciones Java EE y también un servidor web HTTP desarrollado por BEA Systems posteriormente adquirida por Oracle Corporation. Se ejecuta en Unix, Linux, Microsoft Windows, y otras plataformas. WebLogic puede utilizar Oracle, DB2, Microsoft SQL Server, y otras bases de datos que se ajusten al estándar JDBC. El servidor WebLogic es compatible con WS-Security y cumple con los estándares de J2EE 1.3 desde su versión 7 y con la J2EE 1.4 desde su versión 9 y Java EE para las versiones 9.2 y 10.x[48]

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems.[49]

6.5. Sistemas Operativos para Teléfonos Celulares y/o Tabletas

Existe una gran cantidad de Sistemas Operativos para Teléfonos Celulares Inteligentes (*Smartphones*) y Tabletas (*Tablets*), los cuales en su gran mayoría se derivan de versiones para Microcomputadores.

De acuerdo a estadísticas con fecha diciembre 2013, la distribución de sistemas operativos móviles fue la siguiente:



Gráfica 119. Participación en el mercado de Sistemas Operativos móviles. http://www.netmarketshare.com/# Diciembre 2013.

Para diciembre del 2014, los números cambiaron un poco, liderando la contienda Android con 47%, iOS 43% (lo que representa el 90% del mercado), Symbian 3%, Windows Phone 2% y BB OS 1%.

Entre los más usados hoy en día, y otros que se introducen con mucho potencial, se pueden mostrar:

Android es un sistema operativo basado en Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, inicialmente desarrollado por Android Inc. Google respaldó económicamente y más tarde compró esta empresa en 2005. El primer móvil con el sistema operativo Android fue el HTC Dream[50].

BlackBerry OS es un sistema operativo móvil desarrollado por RIM (ahora Blackberry) para los dispositivos BlackBerry. El sistema permite multitarea y tiene soporte para diferentes métodos de entrada adoptados para su uso en computadoras de mano, particularmente la trackwheel, trackball, touchpad y pantallas táctiles[51].

Firefox OS es un sistema operativo móvil, basado en HTML5 con núcleo Linux, de código abierto, para *smartphones* y tabletas. Es desarrollado por Mozilla Corporation. Este sistema operativo está enfocado especialmente en los dispositivos móviles, incluidos los de gama baja. Está diseñado para permitir a las aplicaciones HTML5 comunicarse directamente con el hardware del dispositivo usando JavaScript y Open Web APIs[52].

iOS es un sistema operativo móvil de la empresa Apple Inc. Originalmente desarrollado para el iPhone (iPhone OS), siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV. La interfaz de usuario de iOS está basada en el concepto de manipulación directa, usando gestos multitáctiles. Los elementos de control consisten de deslizadores, interruptores y botones[53].

Symbian fue un sistema operativo producto de la alianza de varias empresas de telefonía móvil, entre las que se encontraban Nokia, Sony Mobile Communications, Psion, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. Sus

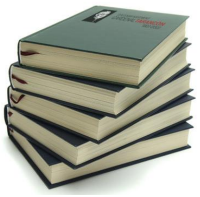
orígenes provenían de su antepasado EPOC32, utilizado en PDA's y *Handhelds* de PSION. El objetivo de Symbian fue crear un sistema operativo para terminales móviles que pudiera competir con el de Palm o el Windows Mobile 6.X de Microsoft y ahora Android de Google Inc. , iOS de Apple Inc. y BlackBerry OS de Blackberry[54].

Ubuntu Touch es un sistema operativo móvil basado en Linux. Es desarrollado por Canonical Ltd. Presentado el 2 de enero de 2013 al público mediante un anuncio en la web de Ubuntu, culmina el proceso de Canonical de desarrollar una interfaz que pueda utilizarse en ordenadores de sobremesa, portátiles, netbooks, tablets y teléfonos inteligentes[55].

Windows Phone es un sistema operativo móvil desarrollado por Microsoft, como sucesor de la plataforma Windows Mobile. A diferencia de su predecesor, está enfocado en el mercado de consumo generalista en lugar del mercado empresarial. Con Windows Phone, Microsoft ofrece una nueva interfaz de usuario que integra varios servicios propios como SkyDrive, Skype y Xbox Live en el sistema operativo[56].

7. Referencias Bibliográficas

Bibliografía:



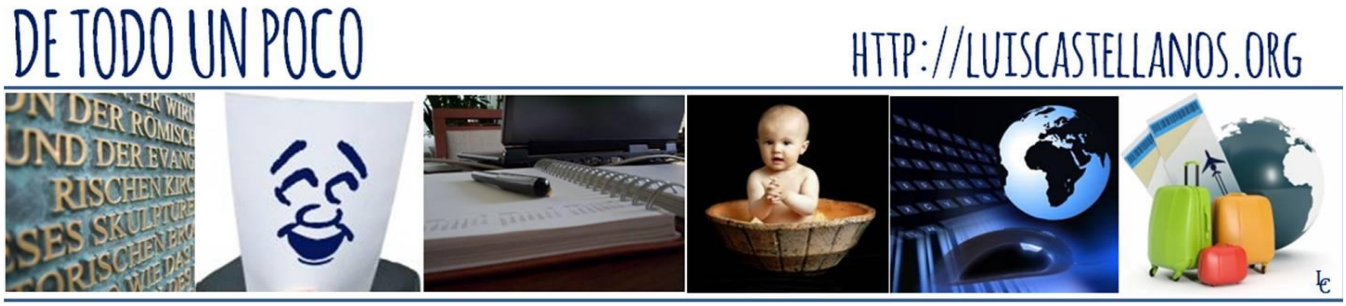
1. Glez del Alba, Angel: Teoría de los Sistemas Operativos. s/f. www.ieru.org/org/tmp/informatica/ssoo/Apuntes/Cap5so1_a.doc
2. Ramírez, Israel J: Los Sistemas Operativos. ULA-FACES. Mérida. s/f. http://isis.faces.ula.ve/COMPUTACION/Israel/Sistema_operat.PDF
3. Tanembaum, Andrew: Sistemas Operativos Modernos. Ed Pearson-Prentice Hall. México, 3ra Edición. 2009.

Webgrafía:

---

[1] http://es.wikibooks.org/wiki/Sistemas_operativos/Caracter%C3%ADsticas
[2] http://www.tiposde.org/informatica/15-tipos-de-sistemas-operativos/
[3] http://es.wikipedia.org/wiki/Sistema_operativo
[4] http://es.wikibooks.org/wiki/Sistemas_operativos/Por_servicios
[5] http://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario
[6] http://es.wikipedia.org/wiki/Sem%C3%A1foro_%28inform%C3%A1tica%29
[7] http://es.wikipedia.org/wiki/Interrupci%C3%B3n
[8] http://es.wikipedia.org/wiki/Intel_Core_2
[9] http://www.fayerwayer.com/2010/01/ibm-y-fujifilm-desarrollan-cinta-magnetica-para-almacenar-35-tb-de-informacion/
[10] http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/SO3.htm
[11] http://sistemasoperativos3000.blogspot.com/2009/07/administracion-de-la-memoria-principal.html
[12] http://es.wikipedia.org/wiki/Paginaci%C3%B3n_de_memoria
[13] http://wiki.inf.utfsm.cl/index.php?title=Paginaci%C3%B3n_por_demanda_y_Fallos_de_P%C3%A1ginas
[14] Idem.
[15] http://es.wikipedia.org/wiki/Segmentaci%C3%B3n_de_memoria
[16] http://sistemasoperativos3000.blogspot.com/2009/07/administracion-de-la-memoria-principal.html
[17] http://es.wikipedia.org/wiki/Tarjeta_perforada
[18] http://es.wikipedia.org/wiki/Software_libre
[19] http://luiscastellanos.wordpress.com/tecnologia/software-libre/libertades/
[20] http://es.wikipedia.org/wiki/Software_propietario
[21] http://en.wikipedia.org/wiki/IBM_i
[22] http://en.wikipedia.org/wiki/Linux_on_System_z
[23] http://es.wikipedia.org/wiki/Unix
[24] http://es.wikipedia.org/wiki/Z/OS
[25] http://en.wikipedia.org/wiki/Z/TPF
[26] http://en.wikipedia.org/wiki/Z/VM
[27] http://en.wikipedia.org/wiki/Z/VSE
[28] http://es.wikipedia.org/wiki/Servidor
[29] http://es.wikipedia.org/wiki/FreeBSD
[30] http://es.wikipedia.org/wiki/Linux
[31] http://es.wikipedia.org/wiki/Mac_OS_X_Server
[32] http://es.wikipedia.org/wiki/Microsoft_Servers

[33] http://es.wikipedia.org/wiki/Windows_Server
[34] http://es.wikipedia.org/wiki/Novell_Netware
[35] http://es.wikipedia.org/wiki/Solaris_%28sistema_operativo%29
[36] http://es.wikipedia.org/wiki/Unix
[37] http://es.wikipedia.org/wiki/Windows_NT
[38] http://es.wikipedia.org/wiki/Linux
[39] http://es.wikipedia.org/wiki/Mac_OS
[40] http://es.wikipedia.org/wiki/MS-DOS
[41] http://es.wikipedia.org/wiki/Microsoft_Windows
[42] http://www.ecured.cu/index.php/Servidores_Web
[43] http://es.wikipedia.org/wiki/Servidor_HTTP_Apache
[44] http://es.wikipedia.org/wiki/Cherokee_%28servidor_web%29
[45] http://es.wikipedia.org/wiki/Google_Web_Server
[46] http://es.wikipedia.org/wiki/Internet_Information_Services
[47] http://es.wikipedia.org/wiki/Nginx
[48] http://es.wikipedia.org/wiki/Oracle_WebLogic
[49] http://es.wikipedia.org/wiki/Tomcat
[50] http://es.wikipedia.org/wiki/Android
[51] http://es.wikipedia.org/wiki/BlackBerry_OS
[52] http://es.wikipedia.org/wiki/Firefox_OS
[53] http://es.wikipedia.org/wiki/IOS_%28sistema_operativo%29
[54] http://es.wikipedia.org/wiki/Symbian
[55] http://es.wikipedia.org/wiki/Ubuntu_Touch
[56] http://es.wikipedia.org/wiki/Windows_Phone



DEL AUTOR

Luis Castellanos es venezolano, nacido en Caracas. Ingeniero de Sistemas de profesión (IUPFAN), Experto en e-Learning (FATLA), con Maestría en Ingeniería de Sistemas (USB) y un Doctorado Honoris Causa en Educación (CIHCE).

Es Docente Universitario en varios Centros de Estudios, y bloguero. Además, es editor de la Revista Académica Digital "De Tecnología y Otras Cosas" (DTyOC).