Deming
EDITORIAL TECNOLÓGICA

FUNDAMENTOS DE LOS SISTEMAS OPERATIVOS





INSTITUTO SUPERIOR TECNOLÓGICO UNIVERSITARIO CORPORATIVO
EDWARDS DEMING®
saber qué hacer y cómo hacerlo

Autor:
PATRICIO RENÉ GARCÍA URBINA

Patricio René García Urbina

Fundamentos de los Sistemas Operativos

Prólogo escrito por Christian Vásquez

Editorial Deming

Fundamentos de los Sistemas Operativos
Autor: Patricio René García Urbina
Docente del Instituto Superior Tecnológico Universitario Corporativo Edwards Deming

Editorial Deming
Dirección: Calle Edmundo Chiriboga OE-3H y Jorge Páez, Quito, Ecuador

Primera edición – 2025

ISBN: 978-9942-692-54-2

Impreso por: GraficArte

Este libro ha sido debidamente examinado y valorado en la modalidad doble par ciego con el fin de garantizar la calidad de la publicación.

© Este libro es de uso científico, académico e investigativo. Prohibida su comercialización.

Para más información, escribir al correo:
editorial@deming.edu.ec

He utilizado y revisado muchos materiales sobre sistemas operativos y puedo decir que este texto destaca por su claridad conceptual y enfoque pedagógico. Este libro logra explicar con solidez los principios técnicos que rigen el funcionamiento de los sistemas informáticos, sin perder de vista las necesidades de quienes se están formando en esta área. A lo largo de sus capítulos, se desarrollan temas clave como la gestión de procesos, la memoria, los sistemas de archivos, la sincronización y la seguridad. Me ha resultado especialmente útil la manera en que el texto articula teoría con ejemplos y diagramas explicativos que facilitan la comprensión de conceptos complejos. El estilo es directo, bien estructurado y con una secuencia lógica que favorece tanto el autoestudio como su uso en el aula. Además, el enfoque no se limita a un sistema operativo específico, sino que permite al lector comprender los fundamentos comunes a diferentes entornos, lo que amplía su aplicabilidad.

Recomiendo este material como una base sólida para cualquier estudiante o profesional que desee entender cómo funcionan realmente los sistemas que usamos a diario. Una lectura indispensable para todo informático en formación.

Christian Vásquez

ÍNDICE DE CONTENIDO

INTRODUCCIÓN 17

CAPÍTULO I DESARROLLO DEL MODELO 18

¿Qué es un sistema operativo? 19

Sistema operativo como una maquina extendida 19

Sistema operativo como administrador de aplicaciones 20

Historia de los Sistemas Operativos 20

La primera generación (1945 a 1955): tubos al vacío 20

La segunda generación (1955 a 1965): transistores y sistemas de procesamiento por lotes 21

La tercera generación (1965 a 1980): circuitos integrados y multiprogramación 22

La cuarta generación (1980 a la fecha): las computadoras personales 23

Revisión del hardware de computadora 24

Procesador 24

Memoria 25

Disco 25

Cinta Magnética 26

Dispositivos de E/S 27

Buses 27

Arranque de la computadora 28

Los tipos de sistemas operativos 29

De Mainframe 29

De servidores 29

De Multiprocesadores 29

De computadores Personales 30

De computadoras de Bolsillo 30

Integrados 30

De Nodos Sensores 30

En tiempo real 31

De tarjetas Inteligentes 31

Conceptos de los sistemas operativos 31

Procesos 31

Espacios de direcciones 32

Archivos 33

Entrada/salida 34

Protección 34

El shell 35

Llamadas al sistema 36

Llamadas al sistema para la administración de procesos 37

Llamadas al sistema para la administración de archivos 39

Llamadas al sistema para la administración de directorios 39

Miscelánea de llamadas al sistema 40

Estructura de un sistema operativo 40

Sistemas Monolíticos 40

Sistemas de Capas 42

Microkernels 44

Cliente-Servidor 48

Máquinas Virtuales 49

Exokernels 52

CAPITULO II PROCESOS E HILOS 54

Proceso 54

El modelo del proceso 54

Creación de un proceso 55

Terminación de procesos 56

Jerarquías de procesos 56

Estados de un proceso 56

Implementación de los procesos 57
Modelación de la multiprogramación 58
Hilos 58
¿Qué es un hilo? 58
Ventajas de usar hilos 59
Hilos a nivel de usuario y de kernel 60
Cómo se relacionan los hilos a nivel de kernel y los de usuario 60
Modelo Mx1 (Many to one) 60
Modelo 1x1 (one to one) 61
Modelo MxN (many to many) 61
Comunicación entre procesos 62
Condiciones de carrera 63
Regiones críticas 63
Exclusión mutua con espera ocupada 65
Desactivación de Interrupciones 65
Variables de cerradura 66
Alternancia Estricta 66
Solución de Peterson 66
Instrucción TSL 67
Dormir y Despertar 67
Primitivas de comunicación 67
CAPITULO III PLANIFICACIÓN DE PROCESOS 69
Planificación tipo Roun Robin 69
Planificación por prioridad 69
Criterios del Planificador de Nivel Superior 71
Procesos disjuntos o NO interactuantes 72
Procesos Interactuantes 72
Tipos de Recursos 72

Bloqueos 73
Bloqueo Mutuo 73
Principios del bloqueo mutuo 73
Condiciones para el bloqueo mutuo 74
Modelación de bloqueos 74
Recuperación de bloqueos 74
Técnicas para recuperar un bloqueo 74
Problemas Clásicos de Comunicación entre Procesos 75
El problema de la cena de los filósofos 75
El problema de los lectores y los escritores 77
CAPÍTULO IV ADMINISTRACIÓN DE MEMORIA 78
Política y filosofía 78
Filosofía 78
Políticas 78
Round Robin: 79
SJF (Shortest job first - Prioridad del trabajo más corto): 79
SRT (Sortest remaining time scheduling? Tiempo restante más corto): 79
Memoria real 80
Administración de la memoria con mapas de bits 81
Administración de la memoria con listas ligadas 81
Organización de la memoria virtual 81
Memoria Virtual 81
Administración de la memoria virtual 82
Ficheros y directorios 84
Fichero 84
Directorio 84
En la jerarquía de directorios actual: 85
Sistema de ficheros 86

Definición 86

Sector vs Cluster..... 87

Cluster 87

Técnicas de asignación de bloques a ficheros..... 89

Sistema de ficheros nativo 89

Algunos sistemas de ficheros 90

FAT (File Allocation Table). FAT12, FAT16 y FAT32. 90

NTFS (New Technology File System). 90

CAPÍTULO V ENTRADA/SALIDA 92

Gestión de Entradas y Salidas..... 92

Problemas entre el Sistema Operativo y los Dispositivos. 93

Control/estado y datos de entrada y salida controladas por programas: 94

Entrada y salida sincrónica 95

Entrada y salida Sincrónica..... 95

Procesadores de entrada y salida 96

Buffer de entrada y salida. 97

CAPÍTULO VI INTERBLOQUEOS 98

Una historia basada en hechos reales..... 99

Detección y recuperación. 100

Prevención. 101

Predicción. 101

Escenarios de interbloqueos 103

Interbloqueo entre procesos independientes..... 105

Interbloqueo entre procesos o threads de la misma aplicación 108

Interbloqueo con espera activa 109

Auto-interbloqueo..... 110

Interbloqueo en el tratamiento de eventos asíncronos 112

Interbloqueo en la operación interna del sistema operativo..... 115

Interbloqueo por la competencia en el uso de un recurso 116

Interbloqueo en la comunicación de procesos 119

Livelock vs Deadlock 121

Definición del interbloqueo 123

Modelo básico del sistema 126

Representación mediante un grafo de asignación de recursos 127

Comprobación de la existencia de un interbloqueo 131

Modelo extendido del sistema 135

Tipos de recursos 135

Recursos compartidos o exclusivos 136

Recursos expropiables o no 137

Recursos con una única unidad o con múltiples 139

Recursos reutilizables o consumibles 140

GLOSARIO DE TÉRMINOS 142

REFERENCIAS BIBLIOGRÁFICAS 155

ÍNDICE DE FIGURAS

Figura 1. Posición del Sistema Operativo.....19
Figura 2. Tubos al Vacío.....21
Figura 3. Centro de Cómputo Segunda Generación.....22
Figura 4. Transistores Tercera Generación 22
Figura 5. Computadoras Personales.....23
Figura 6. Procesador Pentium.....24
Figura 7. Memoria RAM 25
Figura 8. Disco Duro 26
Figura 9. Cinta Magnética 26
Figura 10. Dispositivos de Entrada Salida.....27
Figura 11. Buses de Datos 28
Figura 12. Mainframe 29
Figura 13. Árbol de Procesos. 32
Figura 14. Archivos. Directorio 34
Figura 15. Llamada Read.....37
Figura 16. Una versión simplificada del shell.....39
Figura 17. Sistemas Monolíticos 41
Figura 18. Sistemas de Capas 42
Figura 19. Estructura MINIX 3 45
Figura 20. Modelo Cliente-Servidor.....48
Figura 21. Virtualización de servidores.....49
Figura 22. Logo VMWare 51
Figura 23. Exokernel.....52
Figura 24. Multiprocesos.....55
Figura 25. Estados de un proceso.....57
Figura 26. Información de un proceso 58

Figura 27. Modelo Mx1 (one to one) 61
Figura 28. Modelo 1x1 (one to one) 61
Figura 29. Modelo MxN 62
Figura 30. Spooler de impresión 63
Figura 31. Región Critica 64
Figura 32. Alternativa Estricta 66
Figura 33. Instrucción TSL 67
Figura 34. Cena de Filósofos 75
Figura 35. Política FIFO 79
Figura 36. Round Robin 79
Figura 37. Shortest job first 79
Figura 38. SRT 80
Figura 39. Memoria Virtual 81
Figura 40. Jerarquía de Archivos 85
Figura 41. Sector de un Disco 87
Figura 42. Cluster 88
Figura 43. Dispositivos de Entrada y Salida 93
Figura 44. Acceso Directo de la Memoria (DMA). 95
Figura 45. Procesadores de entrada y salida o canales 97
Figura 46. Atasco al cruzar un puente estrecho 100
Figura 47. Problema del pasillo 103
Figura 48. Interbloqueo entre procesos independientes 106
Figura 49. Interbloqueo entre procesos independientes 107
Figura 50. Asignación de recursos del primer ejemplo 129
Figura 51. Asignación de Recursos del Segundo Ejemplo 130
Figura 52. Asignación de Recursos del Tercer Ejemplo 131
Figura 53. Tercer ejercicio implementando en el sistema operativo 134
Figura 54. Espera del tercer ejemplo 134

INTRODUCCIÓN

Un sistema operativo (SO) es el programa que, después de ser cargado inicialmente en la computadora por un programa de arranque, administra todos los demás programas de aplicación en una computadora. Los programas de aplicación hacen uso del sistema operativo al realizar solicitudes de servicios a través de una interfaz de programa de aplicación (API) definida. Además, los usuarios pueden interactuar directamente con el sistema operativo a través de una interfaz de usuario, como una interfaz de línea de comandos (CLI) o una interfaz de usuario gráfica (GUI).

Aporta poderosos beneficios al software informático y al desarrollo de software. Sin un sistema operativo, cada aplicación debería incluir su propia interfaz de usuario, así como el código completo necesario para manejar todas las funciones de bajo nivel de la computadora subyacente, como almacenamiento en disco, interfaces de red, etc. Teniendo en cuenta la amplia gama de hardware subyacente disponible, esto aumentaría enormemente el tamaño de cada aplicación y haría que el desarrollo de software no fuera práctico.

En cambio, muchas tareas comunes, como enviar un paquete de red o mostrar texto en un dispositivo de salida estándar, como una pantalla, se pueden descargar al software del sistema que sirve como intermediario entre las aplicaciones y el hardware. El software del sistema proporciona una forma coherente y repetible para que las aplicaciones interactúen con el hardware sin que las aplicaciones necesiten conocer ningún detalle sobre el hardware.

CAPÍTULO I

DESARROLLO DEL MODELO

Una computadora consta de muchos periféricos, interfaces de red y más dispositivos de entrada y salida, es un manejo muy complicado la administración y sería un trabajo titánico.

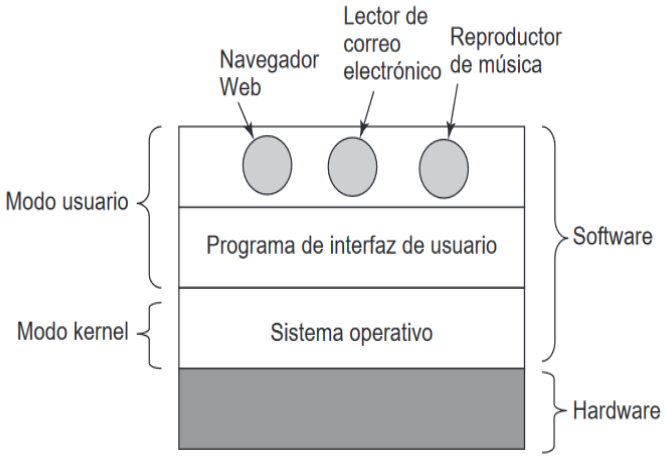
Se entrega esta responsabilidad a los Sistemas Operativos cuyo propósito es manejar todos los recursos disponibles del hardware.

La mayoría de los estudiantes tienen cierto grado de experiencia con los sistemas operativos Windows, Linux, etc. Lamentablemente esto puede ser una falsa apreciación, puesto que el estudiante ha interactuado con el SHELL (Interfase de texto) y (Graphical User Interface) (Interfaz gráfica de usuario).

La figura 1 presenta un esquema general de los componentes principales que aquí se analizan. En la parte inferior se muestra el hardware, que consiste en circuitos integrados (chips), tarjetas, discos, un teclado, un monitor y objetos físicos similares. Por encima del hardware se encuentra el software. La mayoría de las computadoras tienen dos modos de operación: modo kernel y modo usuario. El sistema operativo es la pieza fundamental del software y se ejecuta en modo kernel (también conocido como modo supervisor). En este modo, el sistema operativo tiene acceso completo a todo el hardware y puede ejecutar cualquier instrucción que la máquina sea capaz de ejecutar. El resto del software se ejecuta en modo usuario, en el cual sólo un subconjunto de las instrucciones de máquina es permitido.

En particular, las instrucciones que afectan el control de la máquina o que se encargan de la E/S (entrada/salida) están prohibidas para los programas en modo usuario.

Figura 1. *Posición del Sistema Operativo*



Fuente: (Andrew S)

¿Qué es un sistema operativo?

Los sistemas operativos son los programas encargados de administrar y gestionar de manera eficiente todos los recursos de un ordenador y otros dispositivos. También se les conoce como software de sistema, y su función comienza nada más encender el dispositivo en el que están instalados.

Sistema operativo como una maquina extendida

Este concepto principalmente se encuentra enfocado al hecho de que el sistema operativo realiza la función de convertir algo tan abstracto como el lenguaje maquina en algo que la gente común y corriente podamos entender ya sea en imágenes o diversas expresiones fáciles de entender, siendo el sistema operativo el que se encarga de manejar todas las instrucciones que la maquina internamente puede manejar, otra forma de verlo que es como se plantea en el libro es que oculta la parte oscura y lúgubre de la computadora mostrando un panorama más bello al usuario que lo utiliza.

Sistema operativo como administrador de aplicaciones

Este concepto está enfocado a la parte del sistema operativo que se encarga de administrar a las aplicaciones, esto quiere decir que es el que pone orden a todo lo que sucede dentro de él, por ejemplo cuando queremos imprimir cosas de distintas aplicaciones si el sistema operativo no hiciera su trabajo todo sería un caos ya que los distintos programas que solicitan el uso del dispositivo de salida se pelearan entre sí, a este concepto se le conoce como “race conditions”, pero afortunadamente el sistema operativo le asigna a cada uno su tiempo para utilizar el dispositivo, de esta manera con ayuda del multiplexaje que el sistema utiliza permite que las aplicaciones funcionen en armonía.

Historia de los Sistemas Operativos

Los sistemas operativos han ido evolucionando a través de los años, analizaremos brevemente algunos de los hitos más importantes. Como los sistemas operativos han estado estrechamente relacionados a través de la historia con la arquitectura de las computadoras en las que se ejecutan, analizaremos generaciones sucesivas de computadoras para ver cómo eran sus sistemas operativos.

La primera generación (1945 a 1955): tubos al vacío

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.

Figura 2. *Tubos al Vacío.*



Fuente: (Webcindario, s.f.)

La segunda generación (1955 a 1965): transistores y sistemas de procesamiento por lotes

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.

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.

Figura 3. *Centro de Cómputo Segunda Generación*

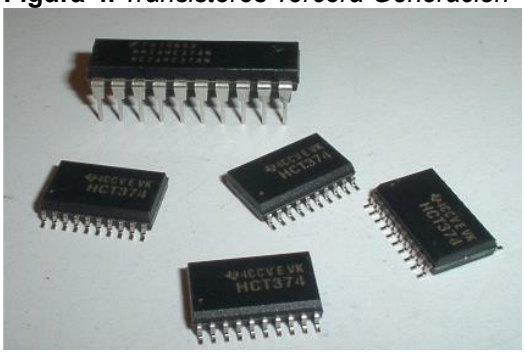


Fuente: (Webcindario, s.f.)

La tercera generación (1965 a 1980): circuitos integrados y multiprogramación

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.

Figura 4. *Transistores Tercera Generación*



Fuente: (Webcindario, s.f.)

La cuarta generación (1980 a la fecha): las computadoras personales

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.

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.

Figura 5. *Computadoras Personales*



Fuente: (Webcindario, s.f.)

Revisión del hardware de computadora

Un sistema operativo está íntimamente relacionado con el hardware de la computadora sobre la que se ejecuta. Extiende el conjunto de instrucciones de la computadora y administra sus recursos. Para trabajar debe conocer muy bien el hardware, por lo menos en lo que respecta a cómo aparece para el programador.

Procesador

Es el "cerebro" de la computadora, lógicamente es llamado CPU o micro. Sus funciones principales incluyen el manejo del sistema operativo, la ejecución de las aplicaciones y la coordinación de los diferentes dispositivos que componen el equipo.

Figura 6. *Procesador Pentium*



Fuente: (Paoguaman, s.f.)

En su aspecto físico, el procesador es una pequeña pastilla de silicio recubierta de lo que se conoce como encapsulado. En su interior se encuentran varios niveles de memoria, denominados cache, que aceleran el acceso a los datos de la memoria RAM. Aparte se encuentran registros del micro que es donde finalmente se ejecutan operaciones. La mayoría de las computadoras, además de los registros generales utilizados para contener variables y resultados temporales, tienen varios registros especiales que están visibles para el programador. Uno de ellos es el contador de programa (program counter), otro registro es el apuntador de pila (stack pointer), y otro es PSW (Program Status Word: Palabra de estado del programa).

Memoria

La memoria es uno de los elementos más importantes de un PC, suelen ser de rápido acceso. Las memorias suelen ser volátiles o no volátiles, la clasificación principal de memorias son RAM (Random Access Memory, Memoria de acceso aleatorio) y ROM (Read Only Memory, Memoria de sólo lectura), estas memorias son utilizadas para almacenamiento primario. Existen memorias de almacenamiento secundario como los discos duros. Por lo general a la memoria principal se le conoce como RAM, su escasez puede hacer que incluso el equipo más rápido se vuelva lento, ya que entre sus funciones se encuentra la de servir de almacén para los programas y datos con los que se trabaja en cada momento.

Figura 7. *Memoria RAM*



Fuente: (Paoguaman, s.f.)

Disco

Un disco es un dispositivo que permite almacenar datos de un ordenador como programas (software), y consiste en uno o más platos que giran a 5400, 7200 o 10,800 rpm. Cada plato del disco tiene 2 caras sobre las que se escriben y se leen datos. Cada cara se divide en varias pistas concéntricas y al conjunto formado por todas estas pistas se conoce como cilindro con la misma numeración que pertenece a cada uno de los platos del disco. A su vez cada pista se divide en varios sectores, los cuales contienen la unidad mínima de información que puede leer o escribir un disco duro. Un disco está dividido en miles de clústeres de igual tamaño y los archivos son repartidos y almacenados en distintos clústeres.

Figura 8. *Disco Duro*



Fuente: (Paoguaman, s.f.)

Cinta Magnética

Es un tipo de medio o soporte de almacenamiento de información que se graba en pistas sobre una banda plástica con un material magnetizado, generalmente óxido de hierro. El tipo de información que se puede almacenar en las cintas magnéticas es variado, como vídeo, audio y datos. Para acceder a una cinta, primero debe colocarse en un lector de cinta, después la cinta tal vez tenga que embobinarse hacia delante para llegar al bloque solicitado. 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.

Figura 9. *Cinta Magnética*



Fuente: (Paoguaman, s.f.)

Dispositivos de E/S

La CPU y la memoria no son los únicos recursos que el sistema operativo debe administrar. Los dispositivos de E/S también interactúan mucho con el sistema operativo, y generalmente constan de dos partes: un dispositivo controlador y el dispositivo en sí. El dispositivo controlador es un chip o conjunto de chips que controla físicamente el dispositivo. La otra pieza es el dispositivo en sí. Los dispositivos tienen interfaces bastante simples.

Como la interfaz real del dispositivo está oculta detrás del dispositivo controlador, todo lo que el sistema operativo ve es la interfaz para el dispositivo controlador, que puede ser bastante distinta de la interfaz para el dispositivo. Como cada tipo de dispositivo controlador es distinto, se requiere software diferente para controlar cada uno de ellos. El software que se comunica con un dispositivo controlador, que le proporciona comandos y acepta respuestas, se conoce como driver (controlador).

Figura 10. *Dispositivos de Entrada Salida*



Fuente: (Paoguaman, s.f.)

Buses

El sistema tiene ocho buses (caché, local, memoria, PCI, SCSI, USB, IDE e ISA), cada uno con una velocidad de transferencia y función distintas. El sistema operativo debe estar al tanto de todos estos buses para su configuración y administración. Los dos buses principales son el bus ISA (Industry Standard Architecture, Arquitectura estándar de la industria) de la IBM PC original y su sucesor, el bus PCI (Peripheral Component Interconnect, Interconexión de componentes periféricos). El sistema operativo tiene que saber qué dispositivos periféricos están conectados a la computadora y cómo configurarlos.

Figura 11. *Buses de Datos*



Fuente: (Paoguaman, s.f.)

Arranque de la computadora

En la tarjeta madre o padre hay un programa conocido como BIOS (Basic Input Output System, Sistema básico de entrada y salida) del sistema, que contiene software de E/S de bajo nivel.

Cuando se arranca la computadora, el BIOS inicia su ejecución. Primero hace pruebas para ver cuánta RAM hay instalada y si los dispositivos básicos están instalados y responden en forma correcta. Empieza explorando los buses ISA y PCI para detectar todos los dispositivos conectados a ellos. Estos dispositivos se registran. 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 primer sector del dispositivo de arranque se lee y se coloca en la memoria, para luego ejecutarse. Luego, el sistema operativo consulta al BIOS para obtener la información de configuración. Para cada dispositivo, comprueba si tiene el driver correspondiente. 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.

Los tipos de sistemas operativos

Los sistemas operativos han estado en funcionamiento durante más de medio siglo. Durante este tiempo se ha desarrollado una variedad bastante extensa de ellos, aunque no todos se conocen ampliamente

De Mainframe

Los sistemas operativos para las mainframes (computadoras del tamaño de un cuarto completo) están profundamente orientados hacia el procesamiento de muchos trabajos a la vez, de los cuales la mayor parte requiere muchas operaciones de E/S. Aunque, los sistemas operativos de mainframes están siendo reemplazados gradualmente por variantes de UNIX, como Linux.

Figura 12. *Mainframe*



Fuente: (Paoguaman, s.f.)

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. Algunos sistemas operativos de servidores comunes son: FreeBSD, Linux y Windows Server 200x.

De Multiprocesadores

Estos sistemas se conocen como computadoras en paralelo, multicomputadoras o multiprocesadores, debido a que la técnica de multiprocesamiento consiste en conectar varias CPU en un solo sistema. Necesitan sistemas operativos especiales, pero a menudo son

variaciones de los sistemas operativos de servidores con características especiales para la comunicación, conectividad y consistencia.

De computadores Personales

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 Vista y el sistema operativo Macintosh.

De computadoras de Bolsillo

Una computadora de bolsillo (handheld) 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. Hay celulares muy similares a los PDAs, con excepción de ciertas diferencias como el tamaño, el peso, la interfaz de usuario, teclado y pantalla. Casi todos ellos se basan en CPUs de 32 bits y ejecutan un sofisticado sistema operativo.

Integrados

Los sistemas integrados, conocidos también 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; debido a que todo el software se encuentra en ROM. Algunos ejemplos comunes son los hornos de microondas, las televisiones, los autos, los grabadores de DVDs, los teléfonos celulares y los reproductores de MP3.

De Nodos Sensores

Son pequeñas computadoras con radios integrados y alimentadas con baterías. Se comunican entre sí con una estación base, mediante el uso de comunicación inalámbrica. Tienen energía limitada y deben trabajar durante largos periodos al exterior y desatendidas, con frecuencia en condiciones ambientales rudas. El sistema operativo tiene que ser pequeño y simple debido a que los nodos tienen poca RAM. Al igual que con los sistemas integrados

y los de tiempo real, sólo se ejecuta software que colocan los diseñadores del sistema, es decir, los usuarios no pueden agregar su propio software.

En tiempo real

Se utilizan principalmente en la industria y son sistemas diseñados para funcionar en entornos con limitaciones de tiempo. Un sistema de tiempo real debe tener capacidad para operar en forma fiable según limitaciones de tiempo específicas y deben proveer garantías absolutas de que cierta acción ocurrirá en un instante determinado.

De tarjetas Inteligentes

Son dispositivos del tamaño de una tarjeta de crédito que contienen un chip de CPU. Tienen severas restricciones de poder de procesamiento y memoria. 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. Algunas tarjetas inteligentes funcionan con varios applets de Java al mismo tiempo, lo cual conlleva a la multiprogramación.

Conceptos de los sistemas operativos

Procesos

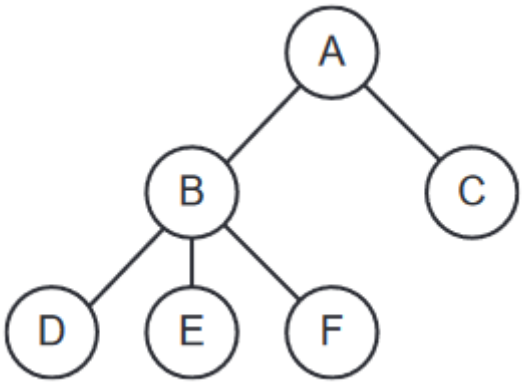
Un concepto clave en todos los sistemas operativos es el proceso. Un proceso es en esencia un programa en ejecución. Cada proceso tiene asociado un espacio de direcciones, una lista de ubicaciones de memoria que va desde algún mínimo (generalmente 0) hasta cierto valor máximo, donde el proceso puede 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.

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. 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.

Figura 13. *Árbol de Procesos.*



Fuente: (Andrew S)

Espacios de direcciones

Cada computadora tiene cierta memoria principal que utiliza para mantener los programas en ejecución.

En un sistema operativo muy simple sólo hay un programa a la vez en la memoria. Para ejecutar un segundo programa se tiene que quitar el primero y colocar el segundo en la memoria.

Los sistemas operativos más sofisticados permiten colocar varios programas en memoria al mismo tiempo. Para evitar que interfieran unos con otros (y con el sistema operativo), se necesita cierto mecanismo de protección. Aunque este mecanismo tiene que estar en el hardware, es controlado por el sistema operativo.

El anterior punto de vista se relaciona con la administración y protección de la memoria principal de la computadora. Aunque diferente, dado que la administración del espacio de direcciones de los procesos está relacionada con la memoria, es una actividad de igual importancia. Por lo general, cada proceso tiene cierto conjunto de direcciones que puede utilizar, que generalmente van desde 0 hasta cierto valor máximo. En el caso más simple, la máxima cantidad de espacio de direcciones que tiene un proceso es menor que la memoria

principal. De esta forma, un proceso puede llenar su espacio de direcciones y aun así habrá suficiente espacio en la memoria principal para contener todo lo necesario.

Sin embargo, en muchas computadoras las direcciones son de 32 o 64 bits, con lo cual se obtiene un espacio de direcciones de 2 32 o 264 bytes, respectivamente. ¿Qué ocurre si un proceso tiene más espacio de direcciones que la memoria principal de la computadora, y desea usarlo todo?

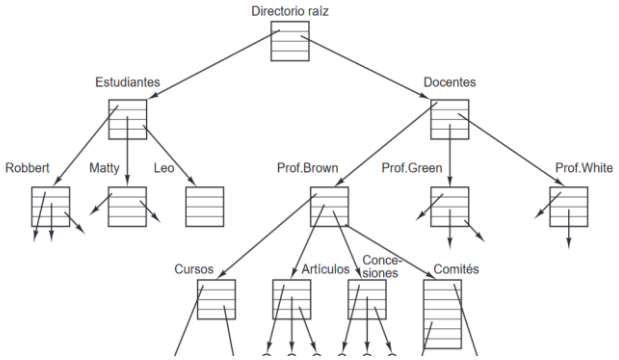
En las primeras computadoras, dicho proceso simplemente no podía hacer esto. Hoy en día existe una técnica llamada memoria virtual, como se mencionó antes, en la cual el sistema operativo mantiene una parte del espacio de direcciones en memoria principal y otra parte en el disco, moviendo pedazos de un lugar a otro según sea necesario. En esencia, el sistema operativo crea la abstracción de un espacio de direcciones como el conjunto de direcciones al que puede hacer referencia un proceso. El espacio de direcciones se desacopla de la memoria física de la máquina, pudiendo ser mayor o menor que la memoria física. La administración de los espacios de direcciones y la memoria física forman una parte importante de lo que hace un sistema operativo.

Archivos

Otro concepto clave de casi todos los sistemas operativos es el sistema de archivos. Como se dijo antes, una de las funciones principales del sistema operativo es ocultar las peculiaridades de los discos y demás dispositivos de E/S, presentando al programador un modelo abstracto limpio y agradable de archivos independientes del dispositivo. Sin duda se requieren las llamadas al sistema para crear los archivos, eliminarlos, leer y escribir en ellos. Antes de poder leer un archivo, debe localizarse en el disco para abrirse y una vez que se ha leído información del archivo debe cerrarse, por lo que se proporcionan llamadas para hacer estas cosas.

Para proveer un lugar en donde se puedan mantener los archivos, la mayoría de los sistemas operativos tienen el concepto de un directorio como una manera de agrupar archivos.

Figura 14. *Archivos. Directorio*



Fuente: (Andrew S)

Entrada/salida

Todas las computadoras tienen dispositivos físicos para adquirir entrada y producir salida. Después de todo, ¿qué tendría de bueno una computadora si los usuarios no pudieran indicarle qué debe hacer y no pudieran obtener los resultados una vez que realizara el trabajo solicitado? Existen muchos tipos de dispositivos de entrada y de salida, incluyendo teclados, monitores, impresoras, etcétera.

Es responsabilidad del sistema operativo administrar estos dispositivos.

En consecuencia, cada sistema operativo tiene un subsistema de E/S para administrar sus dispositivos de E/S. Parte del software de E/S es independiente de los dispositivos, es decir, se aplica a muchos o a todos los dispositivos de E/S por igual. Otras partes del software, como los drivers de dispositivos, son específicas para ciertos dispositivos de E/S.

Protección

Las computadoras contienen grandes cantidades de información que los usuarios comúnmente desean proteger y mantener de manera confidencial. Esta información puede incluir mensajes de correo electrónico, planes de negocios, declaraciones fiscales y mucho más. Es responsabilidad del sistema operativo administrar la seguridad del sistema de manera que los archivos, por ejemplo, sólo sean accesibles para los usuarios autorizados.

Como un ejemplo simple, sólo para tener una idea de cómo puede funcionar la seguridad, considere el sistema operativo UNIX. Los archivos en UNIX están protegidos

debido a que cada uno recibe un código de protección binario de 9 bits. El código de protección consiste en tres campos de 3 bits, uno para el propietario, uno para los demás miembros del grupo del propietario (el administrador del sistema divide a los usuarios en grupos) y uno para todos los demás. Cada campo tiene un bit para el acceso de lectura, un bit para el acceso de escritura y un bit para el acceso de ejecución. Estos 3 bits se conocen como los bits rwx.

El shell

El sistema operativo es el código que lleva a cabo las llamadas al sistema. Los editores, compiladores, ensambladores, enlazadores e intérpretes de comandos en definitiva no forman parte del sistema operativo, aun cuando son importantes y útiles. Con el riesgo de confundir un poco las cosas, en esta sección describiremos brevemente el intérprete de comandos de UNIX, conocido como shell.

Aunque no forma parte del sistema operativo, utiliza con frecuencia muchas características del mismo y, por ende, sirve como un buen ejemplo de la forma en que se pueden utilizar las llamadas al sistema. También es la interfaz principal entre un usuario sentado en su terminal y el sistema operativo, a menos que el usuario esté usando una interfaz gráfica de usuario. Existen muchos shells, incluyendo sh, csh, ksh y bash. Todos ellos soportan la funcionalidad antes descrita, que se deriva del shell original (sh).

Cuando cualquier usuario inicia sesión, se inicia un shell. El shell tiene la terminal como entrada estándar y salida estándar. Empieza por escribir el indicador de comandos (prompt), un carácter tal como un signo de dólar, que indica al usuario que el shell está **esperando** aceptar un comando. Por ejemplo, si el usuario escribe **date** el shell crea un proceso hijo y ejecuta el programa date como el hijo. Mientras se ejecuta el proceso hijo, el shell espera a que termine. Cuando el hijo termina, el shell escribe de nuevo el indicador y trata de leer la siguiente línea de entrada.

Llamadas al sistema

Hemos visto que los sistemas operativos tienen dos funciones principales: proveer abstracciones a los programas de usuario y administrar los recursos de la computadora. En su mayor parte, la interacción entre los programas de usuario y el sistema operativo se relaciona con la primera función: por ejemplo, crear, escribir, leer y eliminar archivos. La parte de la administración de los recursos es en gran parte transparente para los usuarios y se realiza de manera automática. Por ende, la interfaz entre los programas de usuario y el sistema operativo trata principalmente acerca de cómo lidiar con las abstracciones. Para comprender realmente qué hacen los sistemas operativos, debemos examinar esta interfaz con detalle.

Para hacer más entendible el mecanismo de llamadas al sistema, vamos a dar un vistazo rápido a la llamada al sistema read. Tiene tres parámetros: el primero especifica el archivo, el segundo apunta al búfer y el tercero proporciona el número de bytes a leer. Al igual que casi todas las llamadas al sistema, se invoca desde programas en C mediante una llamada a un procedimiento de la biblioteca con el mismo nombre que la llamada al sistema: read. Una llamada desde un programa en C podría tener la siguiente apariencia:

cuenta=read (fd, bufer, nbytes);

La llamada al sistema (y el procedimiento de biblioteca) devuelve el número de bytes que se leen en cuenta. Por lo general este valor es el mismo que nbytes, pero puede ser más pequeño si, por ejemplo, se encuentra el fin de archivo al estar leyendo.

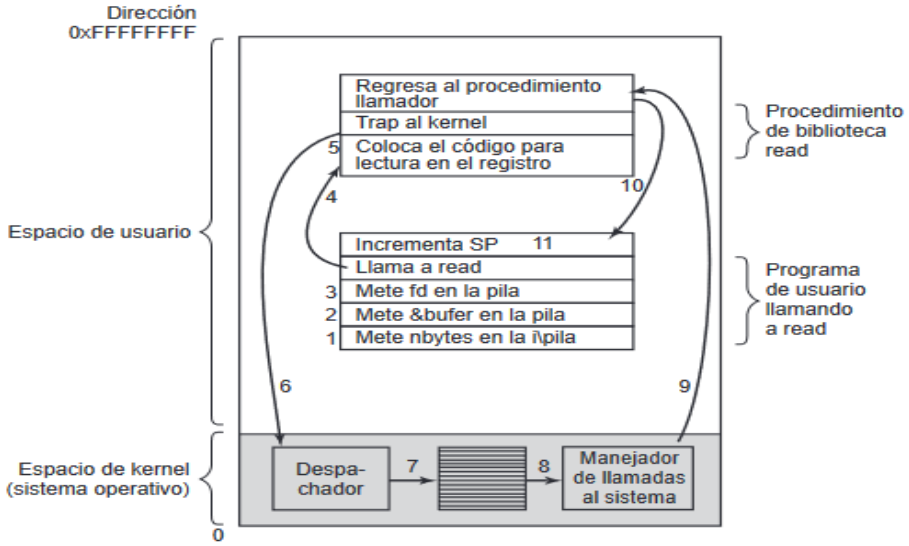
Si la llamada al sistema no se puede llevar a cabo, ya sea debido a un parámetro inválido o a un error del disco, cuenta se establece a 1 y el número de error se coloca en una variable global llamada errno. Los programas siempre deben comprobar los resultados de una llamada al sistema para ver si ocurrió un error.

Las llamadas al sistema se llevan a cabo en una serie de pasos. Para que este concepto quede más claro, vamos a examinar la llamada read antes descrita. En su

preparación para llamar al procedimiento de biblioteca read, que es quien realmente hace la llamada al sistema read, el programa llamador primero mete los parámetros en la pila, como se muestra en los pasos 1 a 3 de la siguiente figura.

Los compiladores de C y C++ meten los parámetros en la pila en orden inverso por razones históricas (esto tiene que ver con hacer que el primer parámetro para printf, la cadena del formato, aparezca en la parte superior de la pila). Los parámetros primero y tercero se pasan por valor, pero el segundo parámetro se pasa por referencia, lo cual significa que se pasa la dirección del búfer (lo cual se indica mediante &), no el contenido del mismo. Después viene la llamada al procedimiento de biblioteca (paso 4). Esta instrucción es la instrucción de llamada a procedimiento normal utilizada para llamar a todos los procedimientos.

Figura 15. *Llamada Read*



Fuente: (Andrew S)

Llamadas al sistema para la administración de procesos

El primer grupo de llamadas en la figura 1-18 se encarga de la administración de los procesos. Fork es un buen lugar para empezar este análisis. fork es la única manera de crear un nuevo proceso en POSIX. Crea un duplicado exacto del proceso original, incluyendo todos los descriptores de archivos, registros y todo lo demás. Después de fork, el proceso original y la copia (el padre y el hijo) se van por caminos separados.

Todas las variables tienen valores idénticos al momento de la llamada a fork, pero como los datos del padre se copian para crear al hijo, los posteriores cambios en uno de ellos no afectarán al otro (el texto del programa, que no se puede modificar, se comparte entre el padre y el hijo). La llamada a fork devuelve un valor, que es cero en el hijo e igual al identificador del proceso (PID) hijo en el padre. Mediante el uso del PID devuelto, los dos procesos pueden ver cuál es el proceso padre y cuál es el proceso hijo.

En la mayoría de los casos, después de una llamada a fork el hijo tendrá que ejecutar código distinto al del padre. Considere el caso del shell: lee un comando de la terminal, llama a fork para crear un proceso hijo, espera a que el hijo ejecute el comando y después lee el siguiente comando cuando el hijo termina.

Para esperar a que el hijo termine, el padre ejecuta una llamada al sistema waitpid, la cual sólo espera hasta que el hijo termine (cualquier hijo, si existe más de uno). Waitpid puede esperar a un hijo específico o a cualquier hijo anterior si establece el primer parámetro a 1. Cuando waitpid se completa, la dirección a la que apunta el segundo parámetro (statloc) se establece al estado de salida del hijo (terminación normal o anormal, con el valor de exit). También se proporcionan varias opciones, especificadas por el tercer parámetro.

Ahora considere la forma en que el shell utiliza a fork. Cuando se escribe un comando, el Shell crea un nuevo proceso usando fork. Este proceso hijo debe ejecutar el comando de usuario. Para ello utiliza la llamada al sistema execve, la cual hace que toda su imagen de núcleo completa se sustituya por el archivo nombrado en su primer parámetro. (En realidad, la llamada al sistema en sí es exec, pero varios procedimientos de biblioteca lo llaman con distintos parámetros y nombres ligeramente diferentes. Aquí trataremos a estas llamadas como si fueran llamadas al sistema.) En la figura 16 se muestra un shell muy simplificado que ilustra el uso de fork, waitpid y execve.

Figura 16. *Una versión simplificada del shell.*

```
#define TRUE 1

while (TRUE) {                                  /* se repite en forma indefinida */
        type_prompt();                          /* muestra el indicador de comando en la pantalla */
        read_command(command, parameters);      /* lee la entrada de la terminal */

        if (fork() !=0) {                       /* usa fork para el proceso hijo */
                /* Codigo del padre. */
                waitpid(-1, &status, 0);        /* espera a que el hijo termine */
        } else {
                /* Codigo del hijo. */
                execve(command, parameters, 0); /* ejecuta el comando */
        }
}
```

Administración de procesos

Llamada	Descripción
pid = fork()	Crea un proceso hijo, idéntico al padre
pid = waitpid(pid, &statloc, opciones)	Espera a que un hijo termine
s = execve(nombre, argv, entornp)	Reemplaza la imagen del núcleo de un proceso
exit(estado)	Termina la ejecución de un proceso y devuelve el estado

Fuente: (Miscelaneos, s.f.)

Llamadas al sistema para la administración de archivos

Administración de archivos

Llamada	Descripción
fd = open(archivo, como, ...)	Abre un archivo para lectura, escritura o ambas
s = close(fd)	Cierra un archivo abierto
n = read(fd, bufer, nbytes)	Lee datos de un archivo y los coloca en un búfer
n = write(fd, bufer, nbytes)	Escribe datos de un búfer a un archivo
posicion = lseek(fd, desplazamiento, dedonde)	Desplaza el apuntador del archivo
s = stat(nombre, &buf)	Obtiene la información de estado de un archivo

Llamadas al sistema para la administración de directorios

Administración del sistema de directorios y archivos

Llamada	Descripción
s = mkdir(nombre, modo)	Crea un nuevo directorio
s = rmdir(nombre)	Elimina un directorio vacío
s = link(nombre1, nombre2)	Crea una nueva entrada llamada nombre2, que apunta a nombre1
s = unlink(nombre)	Elimina una entrada de directorio
s = mount(especial, nombre, bandera)	Monta un sistema de archivos
s = umount(especial)	Desmonta un sistema de archivos

Miscelánea de llamadas al sistema

Llamadas varias

Llamada	Descripción
s = chdir(nombredir)	Cambia el directorio de trabajo
s = chmod(nombre, modo)	Cambia los bits de protección de un archivo
s = kill(pid, senial)	Envía una señal a un proceso
segundos = tiempo(&segundos)	Obtiene el tiempo transcurrido desde Ene 1, 1970

Estructura de un sistema operativo

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.

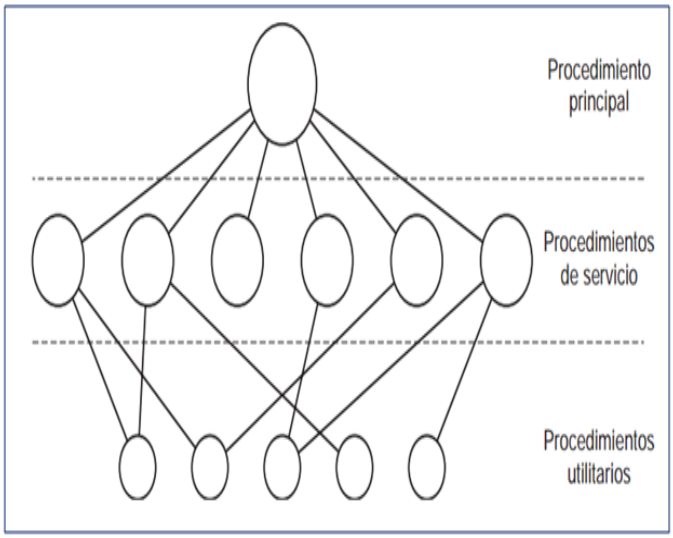
Para construir el programa objeto actual del sistema operativo cuando se utiliza este diseño, primero se compilan todos los procedimientos individuales (o los archivos que contienen los procedimientos) y luego se vinculan en conjunto para formar un solo archivo ejecutable, usando el enlazador del sistema. 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 Figura 15. 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 Figura 15). Esta organización sugiere una estructura básica para el sistema operativo:

- 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.

Figura 17. *Sistemas Monolíticos*



Fuente: (Wordpress, s.f.)

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 anterior.

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.

Sistemas de Capas

Una generalización del diseño de la gráfica 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.

Figura 18. *Sistemas de Capas*

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

Fuente: (Wordpress, s.f.)

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 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.

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.

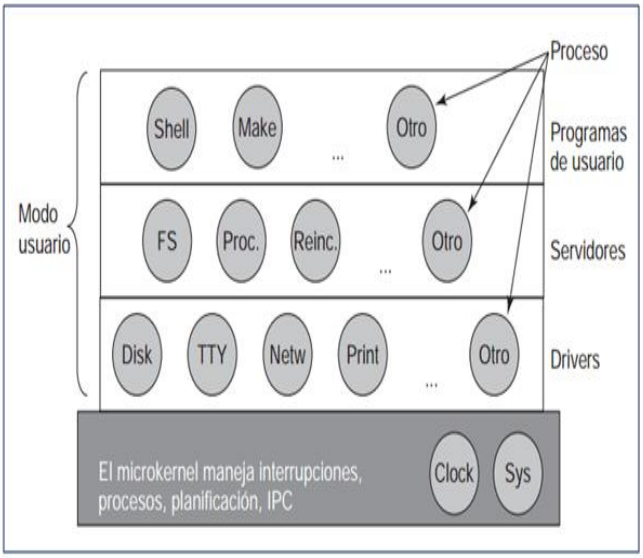
Sin embargo, los sistemas operativos tienen tantos errores que los fabricantes de computadoras colocan botones de reinicio en ellas (a menudo en el panel frontal), algo que los fabricantes de televisiones, estéreos y autos no hacen, a pesar de la gran cantidad de software en estos dispositivos. La idea 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 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 del 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 http://www.minix3.org (Herder y colaboradores, 2006a; Herder y colaboradores, 2006b).

Figura 19. *Estructura MINIX 3*



Fuente: (Wordpress, s.f.)

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, en donde los manejadores de las llamadas 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 procesos.

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.

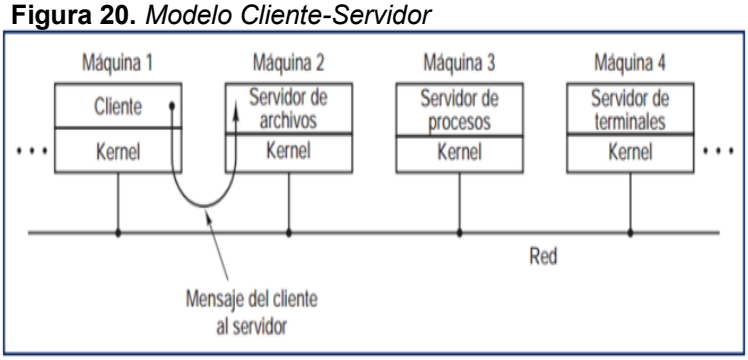
La directiva, asignar prioridades a los procesos, puede realizarse mediante los procesos en modo usuario. De esta forma, la directiva y el mecanismo se pueden desacoplar y el kernel puede reducir su tamaño.

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.

Figura 20. *Modelo Cliente-Servidor*



Fuente: (Wordpress, s.f.)

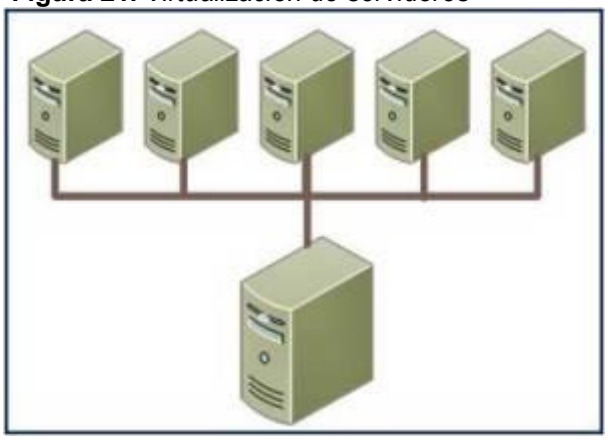
Una generalización obvia de esta idea es hacer que los clientes y los servidores se ejecuten en distintas computadoras, conectadas mediante una red de área local o amplia, como se describe en la gráfica. Como los clientes se comunican con los servidores mediante el envío de mensajes, no necesitan saber 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.

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.

Figura 21. *Virtualización de servidores*



Fuente: (Wordpress, s.f.)

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. 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.

Figura 22. *Logo VMWare*



Fuente: (Wordpress, s.f.)

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.

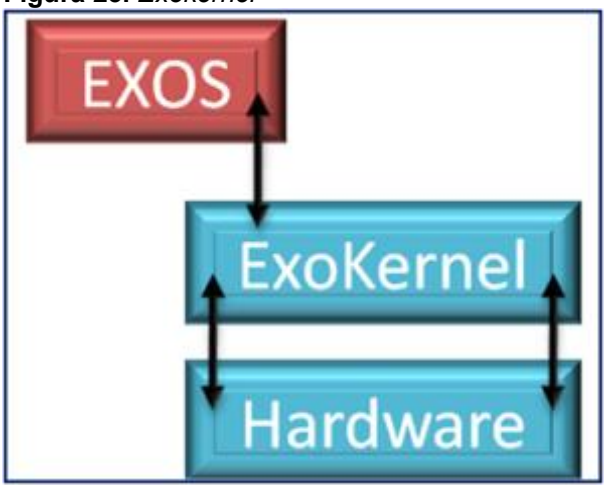
En contraste con los hipervisores de tipo 1, que se ejecutaban en el hardware directo, los hipervisores de tipo 2 se ejecutan como programas de aplicación encima de Windows, Linux o algún otro sistema operativo, conocido como sistema operativo anfitrión. Una vez que se inicia un hipervisor de tipo 2, 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 para virtualización.

Exokernels

Figura 23. *Exokernel*



Fuente: (Wordpress, s.f.)

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 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.

CAPITULO II

PROCESOS E HILOS

Proceso

Un proceso es un programa en ejecución. Se suele decir que el programa es la entidad pasiva y el proceso es la activa.

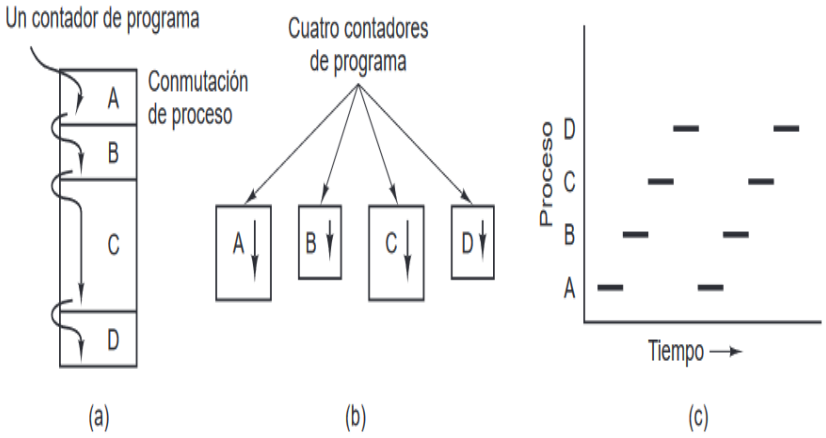
En cualquier sistema de multiprogramación, la CPU conmuta de un proceso a otro con rapidez, ejecutando cada uno durante décimas o centésimas de milisegundos: hablando en sentido estricto, en cualquier instante la CPU está ejecutando sólo un proceso, y en el transcurso de 1 segundo podría trabajar en varios de ellos, dando la apariencia de un paralelismo (o pseudoparalelismo, para distinguirlo del verdadero paralelismo de hardware de los sistemas multiprocesadores con dos o más CPUs que comparten la misma memoria física).

Es difícil para las personas llevar la cuenta de varias actividades en paralelo; por lo tanto, los diseñadores de sistemas operativos han evolucionado con el paso de los años a un modelo conceptual (procesos secuenciales) que facilita el trabajo con el paralelismo. Este modelo, sus usos y algunas de sus consecuencias conforman el tema de este capítulo.

El modelo del proceso

En este modelo, todo el software ejecutable en la computadora, que algunas veces incluye al sistema operativo, se organiza en varios procesos secuenciales (procesos, para abreviar). Un proceso no es más que una instancia de un programa en ejecución, incluyendo los valores actuales del contador de programa, los registros y las variables. En concepto, cada proceso tiene su propia CPU virtual; en la realidad, la CPU real conmuta de un proceso a otro, pero para entender el sistema es mucho más fácil pensar en una colección de procesos que se ejecutan en (pseudo) paralelo, en vez de tratar de llevar la cuenta de cómo la CPU conmuta de programa en programa. Esta conmutación rápida de un proceso a otro se conoce como multiprogramación

Figura 24. *Multiprocesos*



Fuente: (Andrew S)

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:

1. El arranque del sistema.

2. La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.

3. Una petición de usuario para crear un proceso.

4. El inicio de un trabajo por lotes

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.

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:

1. Salida normal (voluntaria).
2. Salida por error (voluntaria).
3. Error fatal (involuntaria).
4. Eliminado por otro proceso (involuntaria)

Jerarquías de procesos

En algunos sistemas, cuando un proceso crea otro, el proceso padre y el proceso hijo continúan asociados en ciertas formas. El proceso hijo puede crear por sí mismo más procesos, formando una jerarquía de procesos. Observe que, a diferencia de las plantas y los animales que utilizan la reproducción sexual, un proceso sólo tiene un padre (pero cero, uno, dos o más hijos).

Estados de un proceso

Aunque cada proceso es una entidad independiente, con su propio contador de programa y estado interno, a menudo los procesos necesitan interactuar con otros. Un proceso puede generar cierta salida que otro proceso utiliza como entrada. En el comando de shell

cat capitulo1 capitulo2 capitulo3 | grep arbol

el primer proceso, que ejecuta cat, concatena tres archivos y los envía como salida. El segundo proceso, que ejecuta grep, selecciona todas las líneas que contengan la palabra “arbol”. Dependiendo de la velocidad relativa de los dos procesos (que dependen tanto de la complejidad relativa de los programas, como de cuánto tiempo ha tenido cada uno la CPU),

puede ocurrir que grep esté listo para ejecutarse, pero que no haya una entrada esperándolo. Entonces debe bloquear hasta que haya una entrada disponible.

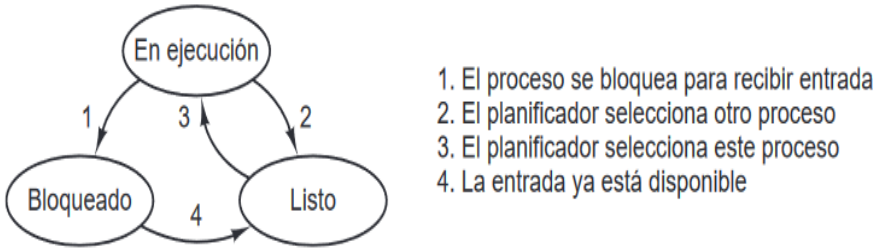
1. En ejecución (en realidad está usando la CPU en ese instante).

2. Listo (ejecutable; se detuvo temporalmente para dejar que se ejecute otro proceso).

3. Bloqueado (no puede ejecutarse sino hasta que ocurra cierto evento externo).

En sentido lógico, los primeros dos estados son similares. En ambos casos el proceso está deseoso de ejecutarse; sólo en el segundo no hay temporalmente una CPU para él. El tercer estado es distinto de los primeros dos en cuanto a que el proceso no se puede ejecutar, incluso aunque la CPU no tenga nada que hacer.

Figura 25. *Estados de un proceso*



Fuente (Sistop, s.f.)

Implementación de los procesos

Para implementar el modelo de procesos, el sistema operativo mantiene una tabla (un arreglo de estructuras) llamada tabla de procesos, con sólo una entrada por cada proceso (algunos autores llaman a estas entradas bloques de control de procesos).

Esta entrada contiene información importante acerca del estado del proceso, incluyendo su contador de programa, apuntador de pila, asignación de memoria, estado de sus archivos abiertos, información de contabilidad y planificación, y todo lo demás que debe guardarse acerca del proceso cuando éste cambia del estado en ejecución a listo o bloqueado, de manera que se pueda reiniciar posteriormente como si nunca se hubiera detenido.

Figura 26. *Información de un proceso*

Administración de procesos	Administración de memoria	Administración de archivos
Registros	Apuntador a la información del segmento de texto	Directorio raíz
Contador del programa	Apuntador a la información del segmento de datos	Directorio de trabajo
Palabra de estado del programa	Apuntador a la información del segmento de pila	Descripciones de archivos
Apuntador de la pila		ID de usuario
Estado del proceso		ID de grupo
Prioridad		
Parámetros de planificación		
ID del proceso		
Proceso padre		
Grupo de procesos		
Señales		
Tiempo de inicio del proceso		
Tiempo utilizado de la CPU		
Tiempo de la CPU utilizado por el hijo		
Hora de la siguiente alarma		

Fuente: (Sistop, s.f.)

Modelación de la multiprogramación

Cuando se utiliza la multiprogramación, el uso de la CPU se puede mejorar. Dicho en forma cruda: si el proceso promedio realiza cálculos sólo 20 por ciento del tiempo que está en la memoria, con cinco procesos en memoria a la vez la CPU deberá estar ocupada todo el tiempo. Sin embargo, este modelo es demasiado optimista, ya que supone que los cinco procesos nunca estarán esperando la E/S al mismo tiempo.

Hilos

En el capítulo anterior, se suponía que un proceso consistía únicamente en un hilo. Ahora, la mayoría de los sistemas operativos proporcionan características que permiten que un proceso tenga múltiples hilos de control. En este capítulo veremos que es un hilo, sus ventajas, y los diferentes modelos de implementación.

¿Qué es un hilo?

Un hilo es una unidad básica de utilización de CPU, la cual contiene un id de hilo, su propio program counter, un conjunto de registros, y una pila; que se representa a nivel del sistema operativo con una estructura llamada TCB (thread control block).

Los hilos comparten con otros hilos que pertenecen al mismo proceso la sección de código, la sección de datos, entre otras cosas. Si un proceso tiene múltiples hilos, puede realizar más de una tarea a la vez (esto es real cuando se posee más de un CPU).

Veamos un ejemplo para clarificar el concepto:

Un servidor web acepta solicitudes de los clientes que piden páginas web. Si este servidor tiene varios clientes y funcionara con un solo hilo de ejecución, solo podría dar servicio a un cliente por vez, y el tiempo que podría esperar un cliente para ser atendido podría ser muy grande.

Una posible solución sería que el servidor funcione de tal manera que acepte una solicitud por vez, y que cuando reciba otra solicitud, cree otro proceso para dar servicio a la nueva solicitud. Pero crear un proceso lleva tiempo y utiliza muchos recursos, entonces, si cada proceso realizará las mismas tareas ¿Por qué no utilizar hilos?

Generalmente es más eficiente usar un proceso que utilice múltiples hilos (un hilo para escuchar las solicitudes, y cuando llega una solicitud, el lugar de crear otro proceso, se crea otro hilo para procesar la solicitud)

Ventajas de usar hilos

Respuesta: el tiempo de respuesta mejora, ya que el programa puede continuar ejecutándose, aunque parte de él esté bloqueado.

Compartir recursos: los hilos comparten la memoria y los recursos del proceso al que pertenecen, por lo que se puede tener varios hilos de ejecución dentro del mismo espacio de direcciones.

Economía: Es más fácil la creación, cambio de contexto y gestión de hilos que de procesos.

Utilizaciones múltiples CPUs: permite que hilos de un mismo proceso ejecuten en diferentes CPUs a la vez. En un proceso mono-hilo, un proceso ejecuta en una única CPU, independientemente de cuantas tenga disponibles.

Hilos a nivel de usuario y de kernel

Hasta ahora hemos hablado de los hilos en sentido genérico, pero a nivel práctico los hilos pueden ser implementados a nivel de usuario o a nivel de kernel.

Hilos a nivel de usuario: son implementados en alguna librería. Estos hilos se gestionan sin soporte del SO, el cual solo reconoce un hilo de ejecución.

Hilos a nivel de kernel: el SO es quien crea, planifica y gestiona los hilos. Se reconocen tantos hilos como se hayan creado.

Los hilos a nivel de usuario tienen como beneficio que su cambio de contexto es más sencillo que el cambio de contexto entre hilos de kernel. A demás, se pueden implementar aún si el SO no utiliza hilos a nivel de kernel. Otro de los beneficios consiste en poder planificar diferente a la estrategia del SO.

Los hilos a nivel de kernel tienen como gran beneficio poder aprovechar mejor las arquitecturas multiprocesadores, y que proporcionan un mejor tiempo de respuesta, ya que, si un hilo se bloquea, los otros pueden seguir ejecutando.

Cómo se relacionan los hilos a nivel de kernel y los de usuario

Existen 3 formas para establecer la relación

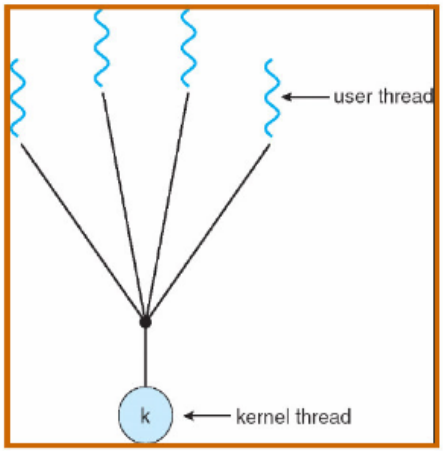
Modelo Mx1 (Many to one)

El modelo asigna múltiples hilos de usuario a un hilo del kernel.

Este caso se corresponde a los hilos implementados a nivel de usuario, ya que el sistema solo reconoce un hilo de control para el proceso.

Tiene como inconveniente que, si un hilo se bloquea, todo el proceso se bloquea. También, dado que solo un hilo puede acceder al kernel cada vez, no podrán ejecutarse varios hilos en paralelo en múltiples CPUs

Figura 27. *Modelo Mx1 (one to one)*



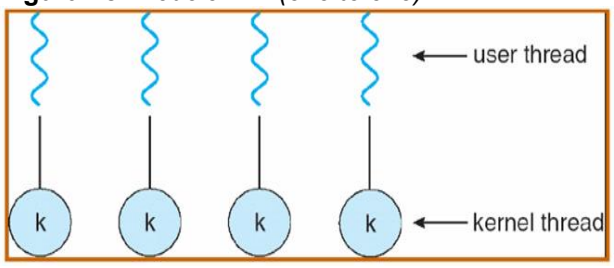
Fuente: (Fing, s.f.)

Modelo 1x1 (one to one)

El modelo asigna cada hilo de usuario a un hilo del kernel. Proporciona una mayor concurrencia que el modelo anterior, permitiendo que se ejecute otro hilo si uno se bloqueó.

Tiene como inconveniente que cada vez que se crea un hilo a nivel de usuario, se crea un hilo a nivel del kernel, y la cantidad de hilos a nivel del kernel están restringidos en la mayoría de los sistema.

Figura 28. *Modelo 1x1 (one to one)*



Fuente: (Fing, s.f.)

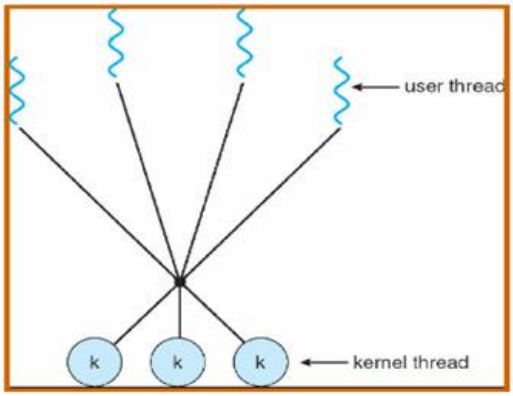
Modelo MxN (many to many)

El modelo multiplexa muchos hilos de usuario sobre un número menor o igual de hilos del kernel. Cada proceso tiene asignado un conjunto de hilos de kernel, independientemente de la cantidad de hilos de usuario que haya creado.

No posee ninguno de los inconvenientes de los dos modelos anteriores, ya que saca lo mejor de cada uno. El usuario puede crear tantos hilos como necesite y los hilos de kernel pueden ejecutar en paralelo. Asimismo, cuando un hilo se bloquea, el kernel puede planificar otro hilo para su ejecución.

Entonces, el planificador a nivel de usuario asigna los hilos de usuario a los hilos de kernel, y el planificador a nivel de kernel asigna los hilos de kernel a los procesadores.

Figura 29. *Modelo MxN*



Fuente: (Fing, s.f.)

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 las siguientes secciones analizaremos algunas de las cuestiones relacionadas con esta comunicación entre procesos o IPC.

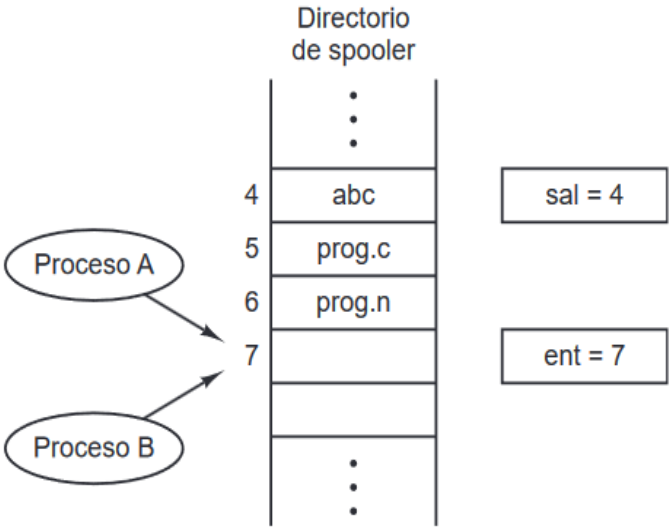
En resumen, hay tres cuestiones aquí. La primera se alude a lo anterior: cómo un proceso puede pasar información a otro. La segunda está relacionada con hacer que dos o más procesos no se interpongan entre sí; por ejemplo, dos procesos en un sistema de reservaciones de una aerolínea, cada uno de los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

Condiciones de carrera

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.

Figura 30. *Spooler de impresión*



Fuente: *(Profesores, s.f.)*

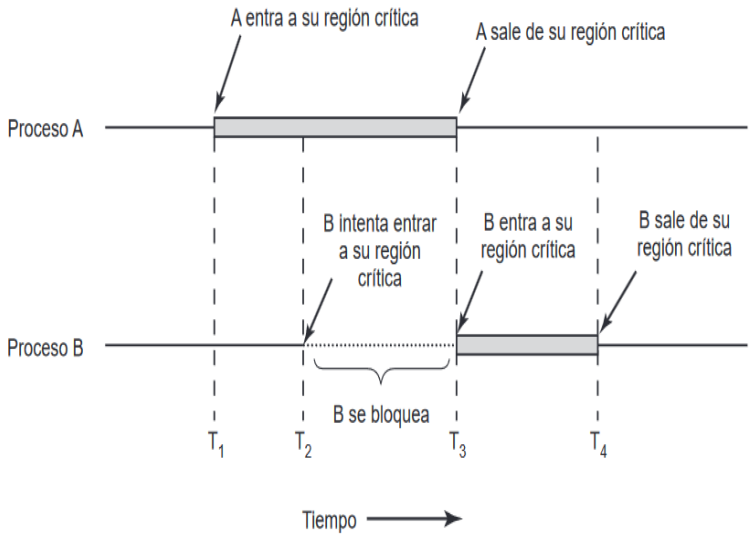
Regiones críticas

¿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.

Figura 31. *Región Critica*



Fuente: (Profesores, s.f.)

Las Condiciones de Competencia para que los procesos cooperen libremente son:

1.Dos procesos nunca pueden estar simultáneamente dentro de sus regiones críticas.

2. No puede suponerse nada acerca de las velocidades de la CPU.

3. Ninguno de los procesos fuera de la sección crítica puede ingresar en ella.

4. Ningún proceso puede esperar eternamente para ingresar en la sección crítica.

Si se cumplen las 4 condiciones anteriores, no hay bloqueo.

Exclusión mutua con espera ocupada

Existen varios métodos para lograr la exclusión mutua, de forma tal que, si un proceso está ocupado con la actualización de la memoria compartida en su región crítica, ningún otro proceso entre a su región crítica y provoque problemas

Exclusión mutua con espera ocupada

1. Desactivación de interrupciones
2. Variables de cerradura
3. Alternancia estricta (espera ocupada)
4. Solución de Peterson (enter_region, leave_region)
5. Instrucción TSL (Test and Set Lock)
6. Dormir y despertar (Sleep y Wakeup)

Desactivación de Interrupciones

Cada proceso desactiva todas sus interrupciones antes de entrar a su región crítica y los activa al salir de ella. Como la CPU alterna entre procesos a través de las interrupciones y del reloj, la misma no alternará con otros procesos.

Una vez desactivadas sus interrupciones, puede examinar y actualizar la memoria compartida sin intervención de otros procesos.

Inconvenientes

• Los procesos de usuarios no deben tener el poder de desactivar las interrupciones.

• La desactivación de las interrupciones sólo ocurre en una sola CPU por lo que otras CPU tendrían acceso a la memoria compartida

• Se deben evitar las condiciones de competencia por lo que es mejor que el núcleo sea quien desactive las interrupciones.

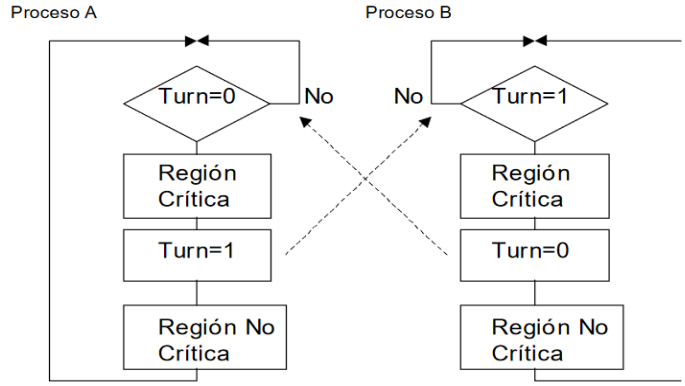
Variables de cerradura

Si se desea que un proceso entre en su región crítica, se analiza el valor de una variable de cerradura, si esta es cero, el proceso cambia el valor a uno y entra en la región crítica. Si el valor es uno, espera hasta que el valor sea cero y realiza el procedimiento anterior.

Inconveniente: posibilidad de que se modifique el valor de la variable justo antes de una alternancia entre procesos.

Alternancia Estricta

Figura 32. *Alternativa Estricta*



Fuente: (Profesores, s.f.)

Solución de Peterson

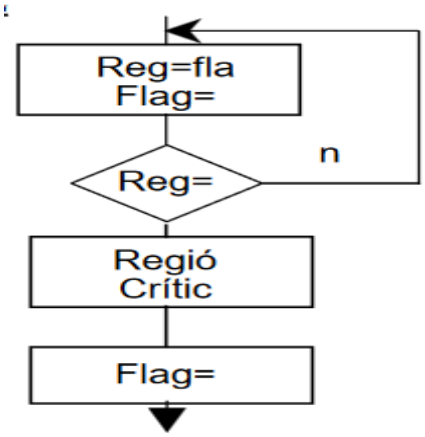
Antes de utilizar las variables compartidas, cada proceso llama a enter_region con su número de proceso, 0 ó 1 como parámetro. Esto provoca una espera hasta que pueda entrar.

Al terminar con ellas, el proceso llama a leave_region para indicar finalización y permitir el ingreso de otro proceso si es necesario.

La llamada enter_region hace que una variable (interested) sea 0 ó 1, lo que provocará la espera o no.

Instrucción TSL

Figura 33. *Instrucción TSL*



Fuente: (Profesores, s.f.)

Dormir y Despertar

• Dormir (sleep): llamada al sistema que provoca el bloqueo del proceso que hizo la llamada (que será suspendido hasta que otro proceso lo despierte).

• Despertar (Wakeup): tiene un parámetro que es el proceso por despertar

Los procesos se pueden comunicar entre sí mediante las primitivas de comunicación entre procesos, que se utilizan para garantizar que dos procesos no se encuentren jamás al mismo tiempo dentro de sus regiones críticas, es decir la excusión mutua.

Primitivas de comunicación

Semáforos: variable que puede tener un valor cero (indicando que no existen despertares) o positivo (si están pendientes uno o más despertares).

Contadores de eventos: variable E sobre la que se definen tres operaciones:

Read(E): regresa el valor actual de E.

Advance(E): incrementa en forma atómica el valor de E en 1.

Await(E): Espera hasta que E tenga el valor de v o mayor.

Monitores: es una colección de procedimientos, variables y estructuras de datos agrupados en cierto módulo o paquete; en donde sólo uno de los procesos puede estar activo

en un monitor en cada momento. Se introducen variables de condición, con dos operaciones sobre ellas (WAIT y SIGNAL).

Transferencia de Mensajes: este método de comunicación utiliza dos primitivas (SEND y RECEIVE) que son llamadas al sistema en vez de comandos del sistema (igual que los monitores y a diferencia de los semáforos).

Características diferenciales:

Confirmación de recepción

Dominios: agrupación de máquinas para reducir conflictos de máquinas con mismo nombre

Autentificación: identificación de clientes en un servidor.

CAPITULO III

PLANIFICACIÓN DE PROCESOS

Cuando hay más de un proceso ejecutable, el S.O. debe decidir cual se ejecutará primero.

La parte del S.O. que toma esta decisión se llama “Planificador”. La planificación de procesos es un intercambio de información entre la memoria principal y el disco y determina el tiempo de ejecución de cada proceso. Una complicación que deben asumir los planificadores es que cada proceso se ejecute durante demasiado tiempo. Casi todas las computadoras tienen incorporado un cronómetro o reloj electrónico que genera interrupciones periódicamente.

Los criterios de planificación son:

Equidad: Asegurarse de que cada proceso reciba una parte justa de tiempo de CPU.

Eficacia: Mantener la CPU ocupada todo el tiempo.

Tiempo de respuesta: Minimizar el tiempo de respuesta para usuarios interactivos.

Retorno: Minimizar el tiempo que los usuarios por lotes tienen que esperar sus salidas

Planificación tipo Roun Robin: A cada proceso se le asigna un intervalo de tiempo de ejecución llamado quantum. Si el proceso continúa en ejecución al final de su quantum, otro proceso se apropia de la C.P.U. Si el proceso está bloqueado o ha terminado antes de consumir su quantum, se alterna el uso de la CPU

Ahora el tema es determinar la longitud del quantum para la alternancia de procesos para mejorar la eficacia.

Planificación por prioridad: A cada proceso se le asigna una prioridad y se permite que se ejecute el proceso que tenga la prioridad más alta. A fin de evitar que los procesos de alta prioridad se ejecuten indefinidamente, el planificador puede reducir la prioridad de los

procesos que actualmente se ejecutan cada tic del reloj. Si esta acción hace que la prioridad se vuelva menor que la del siguiente proceso con más alta prioridad, ocurrirá una conmutación de procesos. Como alternativa se podría asignar a cada proceso un cuanto máximo en que se le permitiera tener la CPU continuamente, cuando se agota ese cuanto, se da oportunidad al proceso con la siguiente prioridad más alta de ejecutarse.

Colas múltiples: Se establecen clases de prioridad con diferente cantidad de quantums, los de la clase de mayor prioridad se ejecutan en un quantum, los de la siguiente en dos quantums y así sucesivamente. Cuando un proceso consume todos sus quantums, se lo mueve a la siguiente clase.

Primero el trabajo más corto: apropiado para las tares por lotes, donde se conocen los tiempos de ejecución previamente. Consiste en utilizar el criterio de ejecutar primero el trabajo más corto, cuando varios trabajos de similar importancia esperan en la lista para iniciar.

Tiempo promedio de regreso para cuatro tareas a,b,c,d: para a el tiempo a; para b, a+b; para c, a + b + c; para d, a + b + c + d; con lo cual queda: (4a+3b+2c+d)/4 produce el promedio mínimo de tiempo de respuesta.

Planificación Garantizada: Consiste en hacer promesas reales al usuario y después cumplirlas (en cuanto al rendimiento). El sistema calculará el tiempo de CPU al que tenía derecho cada proceso, es decir, el tiempo de creación dividido la cantidad de usuarios.

Puesto que también se conoce el tiempo de CPU del que cada proceso ha disfrutado realmente, es fácil calcular la relación entre el tiempo de CPU recibido y el tiempo al que tenía derecho. El algoritmo consiste en ejecutar el trabajo con la relación más baja hasta que su relación haya rebasado a la de su competidor más cercano.

Política vs Mecanismo: Ninguno de los planificadores que hemos visto aceptan entradas de los procesos de usuarios relacionados con las decisiones de planificación. Por tanto, el planificador casi nunca toma la mejor decisión. La solución a este problema consiste

en separar el mecanismo de planificación de la política de planificación. Esto significa que el algoritmo de planificación se regula de alguna manera mediante parámetros, y que estos parámetros pueden ser proporcionados por procesos de usuario.

Planificación de dos niveles: Si la memoria principal no es suficiente, alguno de los procesos ejecutables tendrá que mantenerse en el disco total o parcialmente. Esta situación tiene implicaciones importantes para la planificación, ya que el tiempo de conmutación de procesos cuando hay que traer procesos del disco es varias órdenes de magnitud mayor que cuando la conmutación es a un proceso que ya está en memoria. Primero se carga en la Memoria Principal un subconjunto de los procesos ejecutables, luego el planificador se limita a escoger procesos de ese subconjunto durante un cierto tiempo.

El “Planificador de Nivel Superior” se ocupa de eliminar los procesos que han estado en memoria suficiente tiempo y cargar procesos que han estado en disco demasiado tiempo.

El “Planificador de Nivel Inferior” se ocupa de escoger entre los procesos ejecutables que están en la memoria en ese momento, mientras que el “Planificador de Nivel Superior” se ocupa de trasladar procesos entre la memoria y el disco.

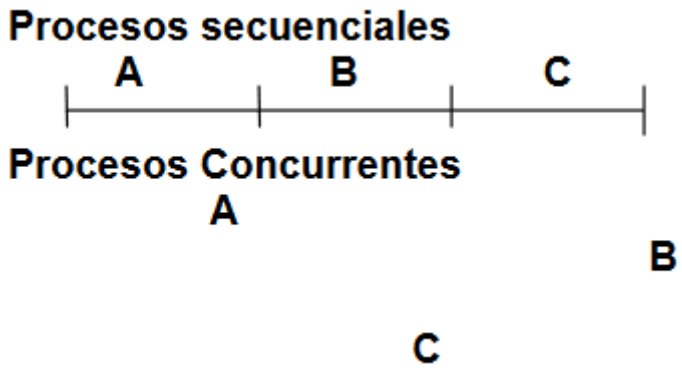
Criterios del Planificador de Nivel Superior

¿Cuánto tiempo hace que el proceso se intercambió desde o hacia el disco?

¿Cuánto tiempo de C.P.U. ha recibido recientemente?

¿Qué tan grande es el proceso?

¿Qué tan alta es la prioridad del proceso?



Dos procesos son concurrentes si su ejecución se superpone en el tiempo.

Dos procesos son concurrentes si la primera operación de uno de ellos comienza antes que se complete la última operación del otro.

Procesos disjuntos o NO interactuantes son procesos concurrentes que operan sobre conjuntos de variables disjuntas

Procesos Interactuantes. dependientes del tiempo

Proceso A **Proceso B**

Recursos Recurso es el elemento utilizado por un SOLO proceso en un momento determinado Puede ser un Dispositivo de hardware (disco, impresora, etc.…) o una parte de información (un registro, una base de datos, etc.…)

Tipos de Recursos

Apropiables: es un recurso que se puede arrebatar al proceso que lo tiene sin que haya efectos adversos; no hay bloqueo mutuo.

No apropiables: No puede quitárselo al proceso poseedor, sin que el cómputo falle. En general en los bloqueos mutuos intervienen recursos de este tipo. Los bloqueos mutuos ocasionados por recursos no expropiables, pueden resolverse reasignando los recursos de un proceso a otro.

Bloqueos

"Un conjunto de procesos se bloquea si cada proceso del conjunto espera un evento que sólo puede ser provocado por otro proceso del conjunto."

Bloqueo Mutuo

Los sistemas tienen muchos recursos que sólo pueden ser utilizados por un proceso a la vez.

Por lo anterior, los S.O. tienen la capacidad de conceder (temporalmente) a un proceso acceso exclusivo a ciertos recursos.

El bloqueo mutuo ocurre cuando dos procesos permanecen bloqueados porque esperan mutuamente un recurso que tiene el otro proceso.

Pueden ocurrir en muchas situaciones (dispositivos de E/S, recursos de hardware y software)

La secuencia de sucesos que se requiere para usar un recurso es:

Solicitar un recurso

Usar el recurso

Liberar el recurso

Si el recurso no está disponible cuando se solicita, el proceso que realiza la solicitud, tiene que esperar.

Principios del bloqueo mutuo

Un conjunto de procesos está en bloqueo mutuo si cada proceso del conjunto está esperando un evento que sólo otro proceso del conjunto puede causar.

Cada miembro del conjunto de procesos mutuamente bloqueado está esperando un recurso que está en poder de otro proceso en bloqueo.

Condiciones para el bloqueo mutuo

Exclusión mutua: Cada recurso está asignado a un único proceso o está disponible.

Posesión y espera: Los procesos que tienen, en un momento dado, recursos asignados con anterioridad, pueden solicitar nuevos recursos.

No apropiación: Los recursos otorgados con anterioridad no pueden ser forzados a dejar un proceso. El proceso que los posee debe liberarlos en forma explícita.

Espera circular: debe existir una cadena circular de dos o más procesos, cada uno de los cuales espera un recurso poseído por el siguiente miembro de la cadena.

Modelación de bloqueos

- Estrategias para el manejo del bloqueo mutuo
- Simplemente hacer caso omiso del problema (Algoritmo del avestruz – Ostrich).
- Detección y Recuperación.
- Evitarlo de manera dinámica, mediante una asignación cuidadosa de recursos.
- Prevención, negando estructuralmente una de las 4 condiciones de competencia.

Recuperación de bloqueos

Una vez detectado el bloqueo, es necesario retroceder.

Técnicas para recuperar un bloqueo

Apropiación se quita el recurso (difícil implantación)

Rollback se crean archivos de verificación

Eliminación de procesos del ciclo que NO son del ciclo

Problemas Clásicos de Comunicación entre Procesos

El problema de la cena de los filósofos

En 1965 Dijkstra planteó y resolvió un problema de sincronización llamado el problema de la cena de los filósofos, que se puede enunciar como sigue. Cinco filósofos se sientan a la mesa, cada uno con un plato de espaghetti. El espaghetti es tan escurridizo que un filósofo necesita dos tenedores para comerlo. Entre cada dos platos hay un tenedor. En la figura 4.15 se muestra la mesa La vida de un filósofo consta de periodos alternos de comer y pensar.

Cuando un filósofo tiene hambre, intenta obtener un tenedor para su mano derecha, y otro para su mano izquierda, cogiendo uno a la vez y en cualquier orden. Si logra obtener los dos tenedores, come un rato y después deja los tenedores y continúa pensando. La pregunta clave es: ¿ puede el lector escribir un programa para cada filósofo que permita comer equitativamente a los filósofos y no se interbloquee ?

Figura 34. Cena de Filósofos

```
#define N 5                                  /* número de filósofos */

void filósofo (int i)                        /* i: qué filósofo (desde 0 hasta N-1) */
{
while (1) {
    pensar ( );                             /* el filósofo está pensando */
    coger_tenedor (i);                       /* coge el tenedor izquierdo */
    coger_tenedor ((i + 1) % N); /* coge el tenedor derecho */
    comer ( );
    dejar_tenedor (i);                       /* deja el tenedor izquierdo en la mesa */
    dejar_tenedor ((i + 1) % N);  /* deja el tenedor derecho en la mesa */
}
}
```

Fuente: (TemasSOuJaen, s.f.)

La figura 30 muestra una solución obvia. El procedimiento coger_tenedor espera hasta que el tenedor especificado esté disponible y lo coge. Por desgracia la solución obvia es incorrecta. Supongamos que los cinco filósofos cogen sus tenedores izquierdos de forma simultánea. Ninguno podría coger su tenedor derecho, lo que produciría un interbloqueo.

Se podría modificar el programa de forma que después de coger el tenedor izquierdo, el programa verificara si el tenedor derecho está disponible. Si no lo está, el filósofo deja el izquierdo, espera cierto tiempo y vuelve a repetir el proceso. Esta propuesta también falla, aunque por razones distintas. Con un poco de mala suerte todos los filósofos podrían empezar el algoritmo de forma simultánea, por lo que cogerían sus tenedores izquierdos, verían que los derechos no están disponibles, esperarían, volverían a coger sus tenedores izquierdos simultáneamente, etc. eternamente. Esto implica un aplazamiento indefinido.

El lector podría pensar: "si los filósofos esperaran un tiempo arbitrario, en vez del mismo tiempo, después de que no pudieran coger el tenedor derecho, la probabilidad de que todo quedara bloqueado, incluso una hora, sería muy pequeña". Esta observación es correcta, pero en ciertas aplicaciones se desea una solución que funcione siempre, y no que pueda funcionar bien con gran probabilidad. (Piense en el control de seguridad de una planta nuclear).

Una mejora a la 30, que no tiene interbloqueos ni aplazamiento indefinido, es la protección de los cinco enunciados siguientes a la llamada al procedimiento pensar mediante un semáforo binario exmut. Antes de empezar a coger los tenedores, un filósofo haría un wait sobre exmut. Desde el punto de vista teórico esta solución es adecuada. Desde el punto de vista práctico presenta un error de eficiencia: en todo instante existirá a lo sumo un filósofo comiendo. Si se dispone de cinco tenedores, se debería permitir que dos filósofos comieran al mismo tiempo.

La solución es correcta, y permite el máximo paralelismo para un número arbitrario de filósofos. Utiliza un vector, estado, para llevar un registro de la actividad de un filósofo: si está comiendo, pensando o hambriento (estado que indica que quiere coger los tenedores). Un filósofo puede comer únicamente si los vecinos no están comiendo. Los vecinos del i-ésimo filósofo se definen en las macros IZQ y DER. En otras palabras, si i= 2, entonces IZQ = 1, y DER = 3.

El programa utiliza un vector de semáforos, uno por filósofo, de forma que los filósofos hambrientos puedan bloquearse si los tenedores necesarios están ocupados. Observe que cada proceso ejecuta el procedimiento filósofo como programa principal, pero los demás procedimientos, coger_tenedores, dejar_tenedores y prueba, son procedimientos ordinarios y no procesos separados.

El problema de los lectores y los escritores

El problema de la cena de los filósofos es útil para modelar procesos que compiten por el acceso exclusivo a un número limitado de recursos, como una unidad de cinta u otro dispositivo de E/S. Otro problema famoso es el de los lectores y escritores (Courtois et al., 1971), que modela el acceso a una base de datos.

Supóngase una base de datos, con muchos procesos que compiten por leer y escribir en ella. Se puede permitir que varios procesos lean de la base de datos al mismo tiempo, pero si uno de los procesos está escribiendo (es decir, modificando) la base de datos, ninguno de los demás debería tener acceso a ésta, ni siquiera los lectores. La pregunta es ¿cómo programaría los lectores y escritores?

En esta solución, el primer lector que obtiene el acceso a la base de datos realiza un wait sobre el semáforo bd. Los lectores siguientes sólo incrementan un contador, nl. Al salir los lectores, éstos decrementan el contador, y el último en salir realiza un signal sobre el semáforo, lo que permite entrar a un escritor bloqueado, si existe.

Una hipótesis implícita en esta solución es que los lectores tienen prioridad sobre los escritores. Si surge un escritor mientras varios lectores se encuentran en la base de datos el escritor debe esperar. Pero si aparecen nuevos lectores, y queda al menos un lector accediendo a la base de datos, el escritor deberá esperar hasta que no haya más lectores interesados en la base de datos.

CAPÍTULO IV

ADMINISTRACIÓN DE MEMORIA

Objetivo: Identificar la administración de memoria y sistemas de archivos por parte del sistema operativo, mediante el análisis de ejercicios de segmentación de memoria, para comprender el funcionamiento de la administración de recursos del computador.

Política y filosofía

Filosofía

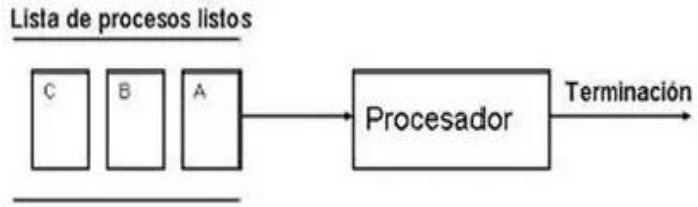
La memoria principal puede ser considerada como un arreglo lineal de localidades de almacenamiento de un byte de tamaño. Cada localidad de almacenamiento tiene asignada una dirección que la identifica. Una de las funciones básicas que debe implementar un SO es la Administración de la Memoria para tener un control sobre los lugares donde están almacenados los datos y procesos que actualmente se están utilizando.

Sea cual sea el esquema de organización del almacenamiento que se adopte para un sistema especifico es necesario decidir qué estrategias se deben utilizar para obtener un rendimiento óptimo.

Políticas

FIFO: Los procesos se despachan de acuerdo a su tiempo de llega a la cola de procesos listos, si un proceso llega al procesador sale hasta que termine. La política FIFO actualmente no se usa como el esquema principal de un sistema, pero si por ejemplo cuando se usa una política de prioridades y hay procesos con la misma prioridad, a estos se les podría aplicar FIFO.

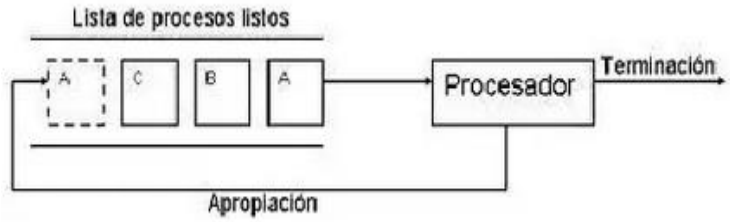
Figura 35. *Política FIFO*



Fuente: (Scribd, s.f.)

Round Robin: Los procesos se despachan en la forma que lo hace el FIFO, pero se les asigna una cantidad limitada de tiempo (CUANTUM) en el procesador, si no termina en ese lapso se manda al final de la lista de procesos listos.

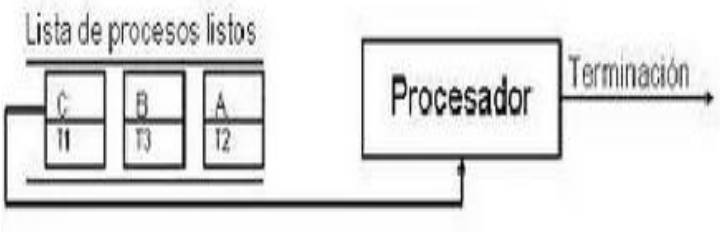
Figura 36. *Round Robin*



Fuente: (Scribd, s.f.)

SJF (Shortest job first - Prioridad del trabajo más corto): Se ejecuta primero el proceso en espera que tiene el menor tiempo estimado. SJF favorece a los procesos cortos, ya que los largos podrían llegar a rezagarse mucho tiempo e incluso nunca ejecutarse.

Figura 37. *Shortest job first*



Fuente: (Scribd, s.f.)

SRT (Sortest remaining time scheduling? Tiempo restante más corto): En SJF una vez que un proceso comienza su ejecución continua hasta terminar. En SRT, un proceso en ejecución puede ser desposeído por uno nuevo de menor tiempo de ejecución. HRN: (¿highest response ratio next? Prioridad de la tasa de respuesta más alta): Política no apropiativa que corrige el retraso excesivo de procesos grandes que produce el SJF, para así no caer en un favoritismo excesivo por los procesos cortos, lo logra usando una formula

basada en el tiempo de espera y el tiempo de servicio, con lo cual la prioridad de cada trabajo no solo está en función del tiempo de servicio sino también del tiempo que ha esperado para ser atendido.

Figura 38. *SRT*



Fuente: (Scribd, s.f.)

Memoria real

La memoria real o principal es en donde son ejecutados los programas y procesos de una computadora y es el espacio real que existe en memoria para que se ejecuten los procesos. Por lo general esta memoria es de mayor costo que la memoria secundaria, pero el acceso a la información contenida en ella es de más rápido acceso. Solo la memoria cache es más rápida que la principal, pero su costo es a su vez mayor.

Cuando no existe memoria virtual no hay diferenciación entre el espacio de direcciones y la memoria real; el espacio de direcciones que puede ser usado en los programas tiene idéntico tamaño al espacio de memoria real posible. Si se utiliza memoria virtual, el espacio de direcciones disponibles para los programas es aquel determinado por el tamaño de la memoria virtual implementada y no el espacio de direcciones provisto por la memoria real disponible (el espacio de la memoria virtual será mayor que el de la memoria real).

La organización y administración de la “memoria principal”, “memoria primaria” o “memoria real” de un sistema ha sido y es uno de los factores más importantes en el diseño de los S. O

Administración de la memoria con mapas de bits

Este tipo de administración divide la memoria en unidades de asignación, las cuales pueden ser tan pequeñas como unas cuantas palabras o tan grandes como varios kilobytes.

Administración de la memoria con listas ligadas

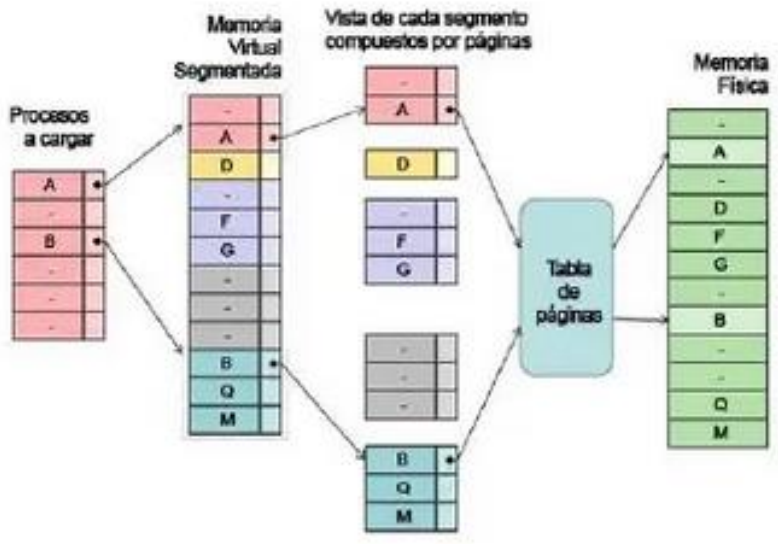
Otra forma de mantener un registro de la memoria es mediante una lista ligada de los segmentos de memoria asignados o libres, en donde un segmento puede ser un proceso o un hueco entre dos procesos

Organización de la memoria virtual

Memoria Virtual

La memoria virtual es una técnica para proporcionar la simulación de un espacio de memoria mucho mayor que la memoria física de una máquina. Esta ilusión permite que los programas se hagan sin tener en cuenta el tamaño exacto de la memoria física. La memoria virtual ha llegado a ser un componente esencial de la mayoría de los S.O actuales. Y como en un instante dado, en la memoria solo se tienen unos pocos fragmentos de un proceso dado, se pueden mantener más procesos en la memoria. La memoria virtual también simplifica la carga del programa para su ejecución llamado reubicación, este procedimiento permite que el mismo programa se ejecute en cualquier posición de la memoria física.

Figura 39. *Memoria Virtual*



Fuente: (Scribd, s.f.)

La ilusión de la memoria virtual está soportada por el mecanismo de traducción de memoria, junto con una gran cantidad de almacenamiento rápido en disco duro. Así en cualquier momento el espacio de direcciones virtual hace un seguimiento de tal forma que una pequeña parte de él, está en memoria real y el resto almacenado en el disco, y puede ser referenciado fácilmente.

Administración de la memoria virtual

Es un método mediante el cual, un sistema operativo simula tener más memoria principal que la que existe físicamente. Para implementar la memoria virtual se utiliza un medio de almacenamiento secundario de alta velocidad de acceso, generalmente en disco duro de la máquina. Un sistema de memoria virtual se implementa utilizando paginación como método de administración de memoria básica y algún mecanismo de intercambio (para descargar páginas de la memoria principal hacia el disco duro y para cargar esas páginas de nuevo a la memoria).

La memoria virtual es una técnica para proporcionar la simulación de un espacio de memoria mucho mayor que la memoria física de una máquina. Esta “ilusión” permite que los programas se hagan sin tener en cuenta el tamaño exacto de la memoria física. La ilusión de la memoria virtual esta soportada por el mecanismo de traducción de memoria, junto con una gran cantidad de almacenamiento rápido en disco duro. Así en cualquier momento el espacio de direcciones virtual hace un seguimiento de tal forma que una pequeña parte de él, está en memoria real y el resto almacenado en el disco, y puede ser referenciado fácilmente

Un trabajo que entre en el sistema se colocará en el espacio vació del almacenamiento principal en el que quepa mejor y que deje la menor cantidad posible de espacio sin utilizar

Un trabajo que entre en el sistema se colocara en el almacenamiento principal en el primer vació disponibles lo bastante grande para contenerlo.

Estrategias del peor ajuste: Consiste en colocar un programa en el almacenamiento primario en el espacio vació donde peor se ajusta es decir en el espacio más grande posible

Estrategias de Obtención. Determinan cuando se debe de transferir una página o un segmento de almacenamiento secundario al primario. Las estrategias de obtención por demanda esperan a que un proceso en ejecución haga referencia a una página o un segmento antes de traer la página o el segmento de almacenamiento primario. Los esquemas de obtención anticipada intentan determinar por adelantado a cuáles paginas o segmentos hará referencia un proceso.

Estrategias de Colocación. Determinan en qué lugar del almacenamiento primario se debe colocar una página o un segmento entrante. Una página entrante se puede colocar en cualquier marco de página disponible.

Estrategias de Reemplazo. Sirven para decidir cual página o segmento se debe desplazar para dar espacio a una página o segmento entrante cuando está completamente ocupado el almacenamiento primario.

Principio de Optimalidad. Para obtener un rendimiento óptimo, la página que se debe reemplazar es aquella que tardara más tiempo en ser utilizada. Esta estrategia óptima de reemplazo se denomina OPT o MIN.

Reemplazo de Páginas aleatorio. Es una técnica sencilla. Todas las páginas que se encuentran en el almacenamiento principal tienen la misma probabilidad de ser elegidas para el reemplazo. Esta estrategia podría seleccionar cualquier página para ser reemplazada, incluyendo la siguiente página la que se hará referencia. Las decisiones de reemplazo aleatorio se de páginas se pueden tomar rápidamente y con un número significativo de marcos de página. Es un esquema que casi no se utiliza por su enfoque aleatorio de acertar o errar.

Sistemas de Archivos

Los sistemas de ficheros son una parte esencial en el trabajo de los sistemas operativos. Gracias tanto a los propios sistemas de ficheros y a la parte del sistema operativo

que se encarga de trabajar con ellos podemos almacenar ficheros y directorios en particiones de nuestros discos duros y dispositivos de almacenamiento masivo en general.

Ficheros y directorios

Seguramente todos conozcamos que es un fichero y qué es un directorio. No obstante, vamos a tratar de definirlos.

Fichero

Un **fichero** es un contenedor de información relacionada entre sí. Es decir, información que tiene sentido en conjunto.

Los ficheros se crearon para poder **almacenar información de manera persistente** en una memoria secundaria o auxiliar.

Todo fichero tiene un **nombre** que sirve para identificarlo, sobre todo para que el usuario lo identifique, y tiene una **extensión** que da información sobre el tipo de información que contiene. El nombre del fichero es una cadena de texto y su extensión sería una cadena de texto corta (unos tres caracteres antiguamente) que aparece después del último punto del nombre del fichero.

Ojo, la extensión nos informa del tipo de información que se supone contiene el fichero, pero si cambiamos la extensión de un fichero no lo estamos transformando por arte de magia.

Directorio

Un directorio es un contenedor de ficheros y directorios. Debido a que el número de ficheros utilizados en un sistema informático es alto, se necesita poder organizarlos para facilitarnos su uso.

A medida que el uso de ficheros en el sistema aumentó se hizo patente la necesidad de utilizar algún mecanismo para organizarlos. Supongamos que no existieran los directorios, situación inicial. No podríamos tener dos ficheros con el mismo nombre y, además, nos encontraríamos con todos los ficheros de todos los usuarios del sistema en un mismo lugar.

La evolución fue crear el directorio, un contenedor de ficheros. Al principio, en la primera solución, tan solo existía un directorio para cada usuario del sistema. De esta forma, cada usuario tendría en su directorio sus propios ficheros. No obstante, cuando el número de ficheros de cada usuario creció de nuevo surgió el problema de organizar todos estos ficheros.

La solución, crear una jerarquía de directorios en forma de árbol. De esta manera se pueden crear directorios y subdirectorios a medida que sean necesarios para organizar toda la información.

Figura 40. *Jerarquía de Archivos*



Fuente: (Scribd, s.f.)

En la jerarquía de directorios actual:

Existe un directorio raíz que contiene a todos los ficheros y directorios de la «unidad».

Un directorio puede contener ficheros y otros directorios, que podemos denominar subdirectorios.

Todo fichero o directorio tiene un directorio padre, que es el directorio que lo contiene directamente, excepto el directorio raíz.

El directorio raíz no tiene directorio padre. Su directorio padre, por simplificar, sería él mismo.

De esta forma podemos crear tantos directorios sean necesarios para organizar los ficheros que contiene una memoria secundaria o auxiliar.

Cuando instalamos un sistema operativo, durante el proceso de instalación, el sistema operativo se encarga de crear una serie de directorios propios y copia en ellos los ficheros que necesita para poder funcionar. A medida que el sistema va funcionando tanto los usuarios, que crean fichero con información propia y directorios para organizarlos, como el sistema operativo, que añade nuevos ficheros y directorios para que el sistema pueda realizar su trabajo, hacen uso de la memoria secundaria.

Sabemos qué es un fichero y un directorio, pero no sabemos cómo se almacenan realmente en una memoria secundaria. Sabemos que las memorias secundarias y auxiliares son memorias de bloques, cuyo bloque básico es el sector que tiene una capacidad de 512 Bytes.

Pero, nos debería surgir las siguientes preguntas:

¿Cómo se almacena un fichero en los sectores de un disco duro (partición)?

¿Cómo puede saber el sistema operativo qué sectores pertenecen a cada fichero y en qué orden?

¿Cómo sabe qué contiene cada directorio y cuál es su directorio padre?

Para ello se utilizan los sistemas de ficheros…

Sistema de ficheros

Definición

Con el término de sistema de ficheros hay un tanto de ambigüedad, se puede hacer referencia a la parte del sistema operativo encargada de gestionar los archivos o a los sistema de ficheros que se almacenan en memorias de almacenamiento masivo. Vamos a tratar de clarificar los términos.

Para nosotros, un **sistema de ficheros** es una estructura de datos que se encarga de organizar los bloques y sectores en una partición de una memoria de almacenamiento masivo para almacenar archivos en ella.

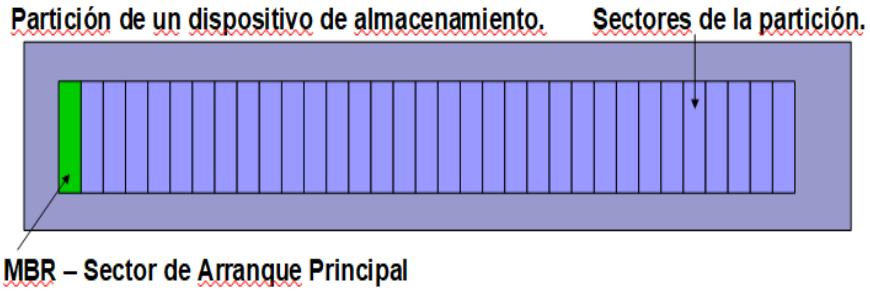
La parte del sistema operativo que se encarga de gestionar los sistemas de ficheros conectados a él lo denominaríamos como **gestor de sistema de ficheros.**

Sector vs Cluster

Los discos duros, y en general los dispositivos de almacenamiento masivo de memoria secundaria y auxiliar, están divididos en sectores. Un **sector** es un bloque de **512 Bytes** y es la **unidad mínima con la que trabaja un dispositivo de almacenamiento masivo** (de bloques).

De esta forma, si se desea escribir un solo byte de un fichero, lo mínimo con lo que va a trabajar el disco duro será con el sector.

Figura 41. *Sector de un Disco*



Fuente: (Educatica, s.f.)

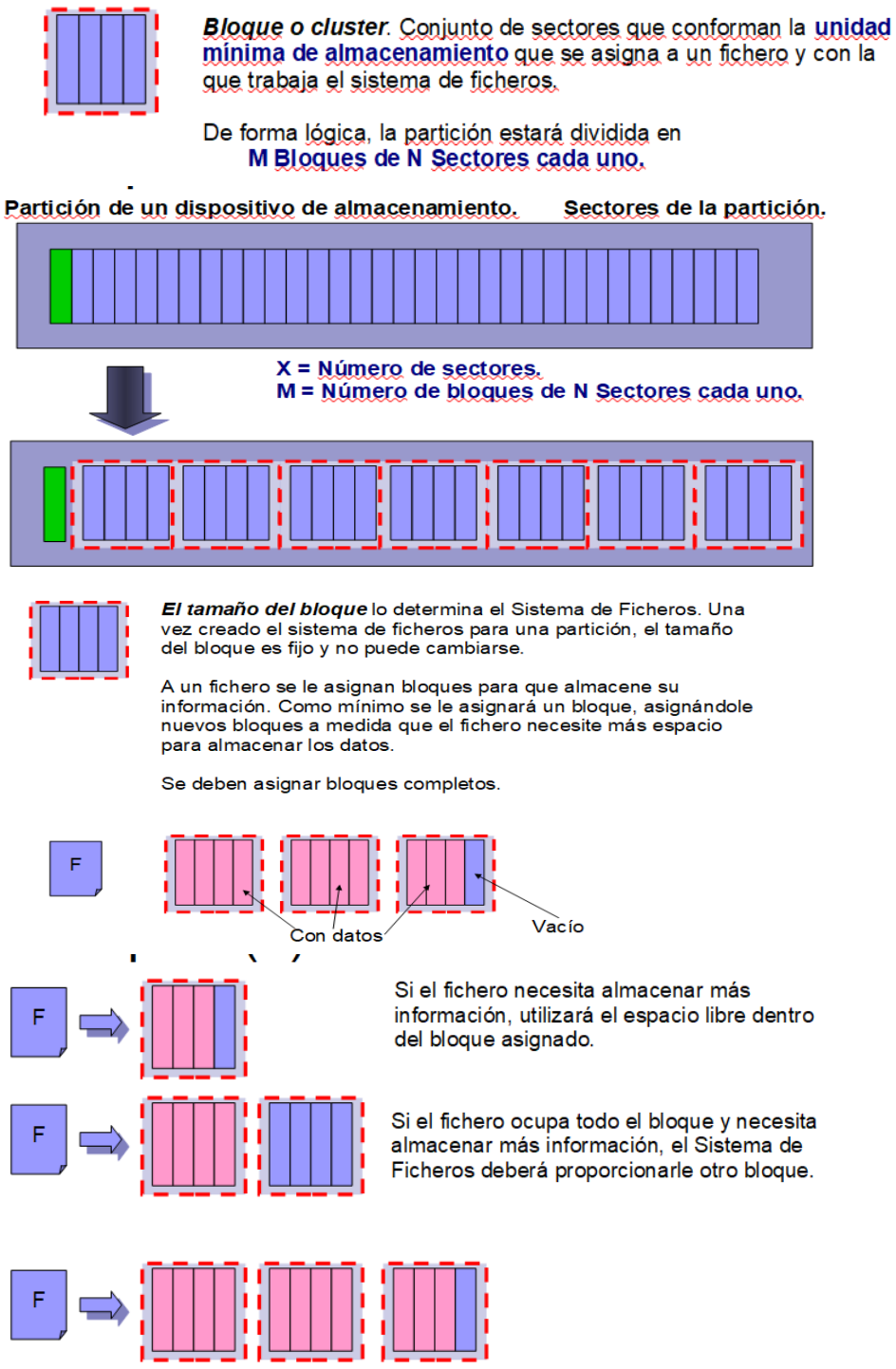
Cluster

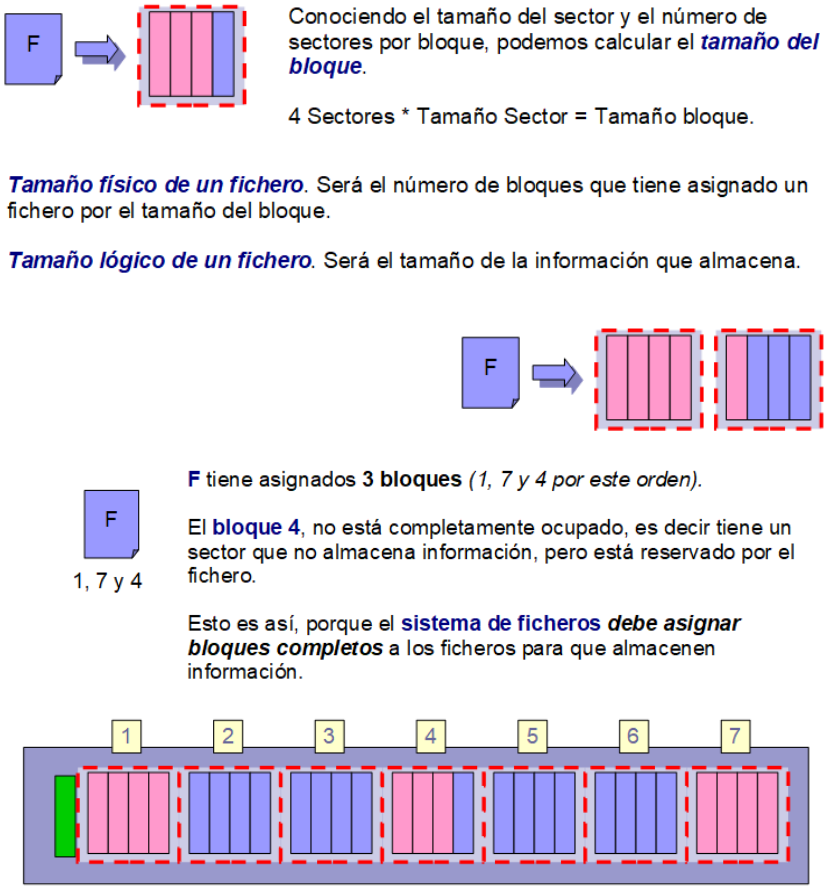
Un cluster, bloque o unidad de asignación consiste en un conjunto o bloque de sectores contiguos. Debido a dos factores es necesario trabajar con cluster o unidades de asignación:

– La cantidad de sectores que hay en los discos duros actuales y a los problemas de direccionamiento que generaría direccionarlos todos (cantidad de bits necesarios para direccionar tal cantidad de sectores).

– Gestionar los sectores ocupados y libres en el sistema de ficheros. Si no se utilizan bloques o clusters la cantidad de sectores es enorme e inmanejable.

Figura 42. *Cluster*





Fuente: (Educatica, s.f.)

Técnicas de asignación de bloques a ficheros

Sistema de ficheros nativo

Un sistema de ficheros se denomina nativo de un sistema operativo cuando fue desarrollado para ese tipo de sistema operativo. Por ejemplo, FAT32 es un sistema de ficheros nativo de Microsoft Windows, puesto que fue desarrollado para ser utilizado en los sistemas operativos de Microsoft Windows.

Sin embargo, que un sistema de ficheros sea nativo de un sistema operativo no significa, necesariamente, que no pueda ser utilizado en otros sistemas de operativos. Por ejemplo, aunque FAT32 o NTFS son sistemas de ficheros nativos de Microsoft Windows, se pueden utilizar en sistemas operativos GNU/Linux como Debian o Ubuntu.

Esto es así puesto que estos sistemas operativos integran en su gestor de sistemas de ficheros este tipo de sistemas de ficheros. Se podría entender como que integran un módulo que es capaz de entender y gestionar estos sistemas de ficheros.

El contra ejemplo son los sistemas de ficheros nativos de GNU/Linux como ext2/3/4 que no son accesibles directamente desde sistemas operativos de Microsoft Windows. Para acceder a este tipo de sistemas de ficheros sería necesario instalar un software especial a modo de módulo de acceso que permitiera acceder a esos sistemas de ficheros.

Algunos sistemas de ficheros

FAT (File Allocation Table). FAT12, FAT16 y FAT32. Son sistemas de ficheros de Microsoft. aunque en un sistema de ficheros que es limitado se sigue utilizando mucho en la versión de FAT 32 puesto que es un sistema de ficheros bien conocido por todos los fabricantes de dispositivos de almacenamiento masivo y electrónica general (por ejemplo, reproductores de DVD / multimedia, etc).

Algunas de las limitaciones más destables de su versión de 32 bits es la limitación de ficheros de hasta 4GB de capacidad y que no gestiona permisos de usuarios (no identifica el propietario del fichero ni los permisos de acceso que se dan a otros usuarios).

NTFS (New Technology File System). viene a suplir las carencias de FAT32, en cuanto a seguridad, tamaño de ficheros y gestión del propio sistema de ficheros por parte del Sistema Operativo.

Almacena la estructura de datos del sistema de ficheros en forma de árbol, mejorando el rendimiento de acceso a la información en disco. De esta forma no hace falta mantener toda la tabla de FAT en memoria para acceder a la información de los sistemas de ficheros.

Con NTFS, Microsoft Windows permite gestionar propietario y permisos de acceso de ficheros. Además, permite ficheros de un tamaño mayor a 4GB.

ext2 / ext3 / ext4

Sistemas de ficheros nativos de GNU/Linux. Desde sistemas de Microsoft Windows no se puede acceder a la información almacenada en este tipo de sistemas de ficheros directamente (se podría con un software especial).

Hay más sistemas de ficheros, como hpfs, sin embargo, a nosotros, por ahora, los que más nos interesan son los que acabamos de citar.

Un sistema de ficheros es una estructura de datos que permite gestionar los ficheros que se almacenan en una partición.

CAPÍTULO V

ENTRADA/SALIDA

Gestión de Entradas y Salidas

Gestión de Entrada y Salida: esto representa los intercambios de información desde el procesador o incluso al acceso directo de la memoria. Esta actividad de entrada y salida se realiza gracias a la relación que hay entre el sistema operativo con los diversos periféricos, así administrar y controlar los estados y recursos que tienen cada uno de estos mediante los controladores de dispositivos.

Dispositivos o periféricos: son aparatos solidos e independientes que van conectados a la unidad central de la computadora. Existen diversos tipos de dispositivos con funciones diferentes.

Existen tres tipos de dispositivos según su función:

Dispositivos de entrada: tienen la tarea de introducir información, órdenes, comandos e instrucciones para que la computadora lo procese con el sistema operativo.

Entre los más conocidos se encuentran:

El Teclado. Es el principal dispositivo para introducir órdenes e información al sistema.

El Mouse. Dispositivo fundamental que sirve para señalar y accionar los diversos elementos de la pantalla del computador.

El Escáner. Este dispositivo nos permite convertir en formato digital los textos, fotografías e imágenes impresas.

Dispositivos de salida: son los que se encargan de procesar los resultados de salida de los procesos de datos. Y mostrarlos al usuario.

Entre ellos tenemos:

El Monitor. Es el principal dispositivo en procesar los datos y mostrárselos al usuario.

La Impresora. Es el periférico encargado de hacer copias en papel de la información que nos muestra el computador.

Dispositivos de Bloques: Estos periféricos almacenan información en bloques de tamaño fijo, estos van desde 128 bytes hasta 1024 bytes.

Entren los dispositivos de bloques se encuentran:

Unidad de lectora de CD –ROM. Permite leer datos, los cuales se graban en el disco duro o pueden ser leídos para obtener información.

Disco Duro. Proporcionan un acceso más rápido a los datos que los discos flexibles y pueden almacenar mucha más información.

Problemas entre el Sistema Operativo y los Dispositivos.

Cada dispositivo tiene diferentes funciones, los dispositivos dependen de núcleo de sistema operativo, existe diferentes velocidades entre los dispositivos tanto como, de entrada, salida y bloque. El sistema operativo trata de buscar la mejor manera de realizar la función de entrada y salida.

Control de entrada y salida: mediante esta actividad el sistema operativo decide qué proceso hará uso del recurso, durante cuánto tiempo y en qué momento.

Entrada y salida Controladas por Programas e Interrupciones

Figura 43. *Dispositivos de Entrada y Salida*



Fuente: (unefa.blogspot, s.f.)

Por programas: es la forma más sencilla en realizar operaciones de entradas y salidas. Esto ocurre cuando el procesador y el dispositivo coinciden en el mismo tiempo definido; el procesador interactúa directamente con el dispositivo comprobando el estado o si tiene algún dato listo para así poder recibirlo.

Por interrupciones: en este caso el procesador espera un tiempo indefinido hasta que el módulo que es el encargado de proyectar la operación de entrada y salida esté preparado para ejecutar la operación.

Control/estado y datos de entrada y salida controladas por programas:

El procesador es el que tiene la mayor importancia y determina la actividad y el control sobre los procesos de entrada y salida y el programa indica en qué momento se ejecuta.

El microprocesador: Se encarga de decidir el estado de los dispositivos indicando información del dispositivo como: datos listos para transmitir, dispositivo ocupado, dispositivo no conectado o error, etc.

Procesamiento de la Interrupción:

a) El dispositivo le envía una señal de interrupción al procesador.

b) El procesador termina la instrucción que estaba ejecutando antes que comience a responder a la interrupción.

c) El procesador investiga si hay una interrupción, si encuentra dicha interrupción le informa al dispositivo que la causo.

d) El procesador se prepara para transferir control a la interrupción y guarda información necesaria para terminar con el proceso que se interrumpió.

e) El procesador carga en la computadora la dirección de inicio del programa de servicio que se interrumpió.

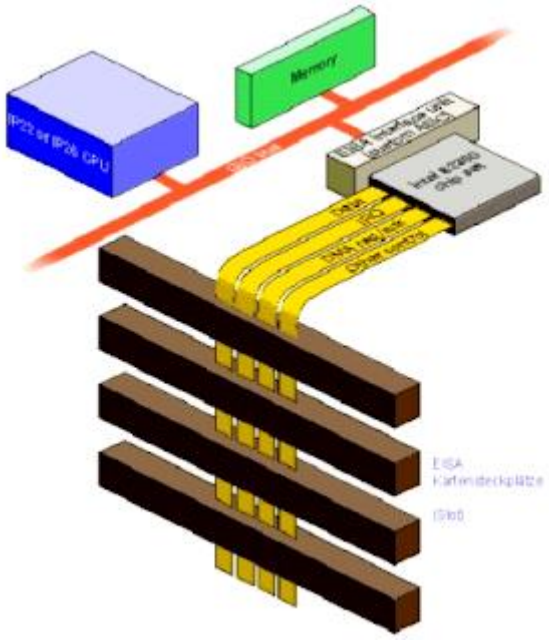
f) Una vez completado con la interrupción, el procesador continúa con el ciclo de instrucción siguiente.

g) Cuando se termina con el servicio de la interrupción los registros vuelven a restaurarse.

Entrada y salida sincrónica: cuando la operación de entrada y salida finaliza, el control es retornado al proceso que la generó.

Entrada y salida Sincrónica: retorna al programa usuario sin esperar que la operación de entrada y salida finalice. Se necesita una llamada al sistema que le permita al usuario esperar por la finalización de entrada y salida, si es requerido.

Figura 44. *Acceso Directo de la Memoria (DMA).*



Fuente: (unefa.blogspot, s.f.)

Tiene la habilidad de transferir directamente la información entre la memoria y los dispositivos de entrada o estos a la memoria sin la necesidad de un canal que intercambie datos.

Características del acceso directo a memoria: -Las computadoras modernas que tienen DMA, transfieren datos a los dispositivos de menos utilización que las que no tienen DMA.

El procesador transfiere cuando otras operaciones se terminan y dicha transferencia se encuentra en proceso, luego termina cuando recibe la interrupción del DMA.

Es útil en actividades de aplicaciones en tiempo real.

El DMA necesita:

- Registro contador: guarda la longitud de bloque a transferir.
- Bits: indica si la operación es de lectura o escritura.
- Bloque de control: controla el funcionamiento del sistema.

Existen 3 tipos de transferencia para el máximo aprovechamiento del bus:

- Por ráfagas: el DMA toma el control del bus y no lo suelta hasta terminar la transferencia.
- Por robo de ciclo: el DMA toma el bus durante un ciclo enviando una palabra cada vez.
- Transparente: se aprovechan los ciclos en que el procesador no usa el bus.

Procesadores de entrada y salida.

Revisión de métodos de entrada y salida: el procesador efectúa una sentencia de espera hasta que el dispositivo esté preparado. La CPU se encarga solo dispositivo durante la:

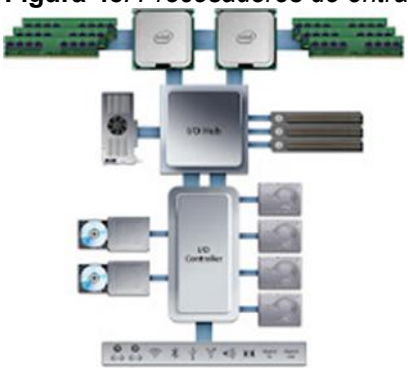
Entrada y salida por interrupciones: el procesador puede realizar otros procesos y solo atiende el dispositivo que produce la interrupción.

Acceso directo a memoria: el procesador solo se encarga de realizar el acceso a la memoria indicándole la cantidad de transferencias. Esta actividad solo se encarga de controlar las transferencias. Durante el proceso de entrada y salida el procesador puede realizar otras tareas.

Procesadores de entrada y salida o canales: El canal es un procesador auxiliar dedicado a la operación de entrada y salida. Esto tiene una lista de instrucciones con límites

y operaciones de entrada y salida, ya que son capaces de controlar las operaciones mediante las instrucciones.

Figura 45. *Procesadores de entrada y salida o canales*



Fuente: (unefa.blogspot, s.f.)

Las instrucciones se almacenan en la memoria principal de la computadora. El procesador de entrada y salida ejecuta estas instrucciones y controla la transferencia de datos.

Buffer de entrada y salida.

Estos son espacios de memoria principal que se mantienen solo para almacenar datos que envían de los dispositivos o destino a ellos.

Los buffers se clasifican en: buffer simple, que su transferencia de entrada e hace desde el dispositivo hacia el buffer que el sistema operativo guarda en la memoria principal a dicho proceso.

Buffer doble, es cuando un proceso transfiere sus datos a un buffer mientras que el sistema operativo vacía o llena el buffer secundario.

CAPÍTULO VI

INTERBLOQUEOS

En un sistema informático se ejecutan de forma concurrente múltiples procesos que, normalmente, no son independientes, sino que compiten en el uso exclusivo de recursos y se comunican y sincronizan entre sí. El sistema operativo debe encargarse de asegurar que estas interacciones se llevan a cabo apropiadamente, proporcionando la sincronización requerida por las mismas, tal como se analiza en el capítulo dedicado a ese aspecto. Sin embargo, generalmente, no basta con esto.

Las necesidades de algunos procesos pueden entrar en conflicto entre sí provocando que estos se bloqueen permanentemente. Esta situación se denomina interbloqueo (en inglés, se usa normalmente el término deadlock). En este capítulo se estudiará este problema y se analizarán sus posibles soluciones. Se trata de un problema general ya identificado en la década de los sesenta del siglo XX y estudiado ampliamente desde entonces.

En el campo de los sistemas operativos, como se explicará en este capítulo, se han adoptado algunas de las soluciones ideadas en el ámbito teórico para evitar que aparezca este problema en el funcionamiento interno del propio sistema operativo. Sin embargo, aunque pueda parecer sorprendente a priori, por razones que se comprenderán a lo largo de este tema, no ha sido así para los servicios que ofrecen los sistemas operativos. La mayoría de los sistemas operativos actuales no aseguran de forma completa que los procesos que usan sus servicios no se vean afectados por esta patología.

Los interbloqueos: Una historia basada en hechos reales.

Escenarios de interbloqueos en un sistema informático.

Definición del interbloqueo y modelo del sistema.

Tratamiento del interbloqueo.

Detección y recuperación del interbloqueo.

Prevención del interbloqueo.

Predicción del interbloqueo.

Tratamiento del interbloqueo en los sistemas operativos.

Diseño de aplicaciones concurrentes libres de interbloqueos.

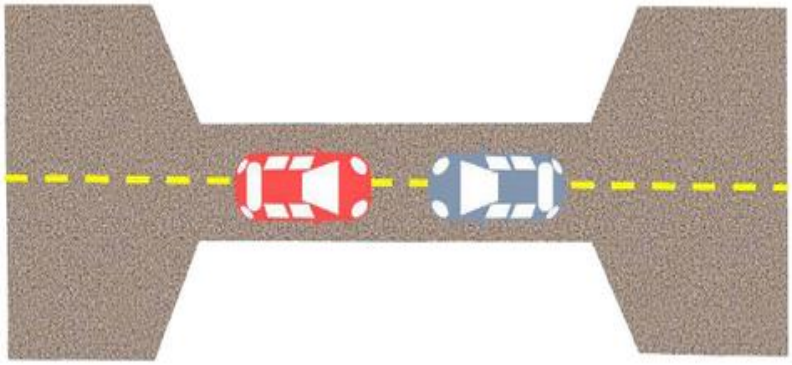
Una historia basada en hechos reales

El problema de los interbloqueos no se circunscribe únicamente al mundo de la informática, sino que aparece en muchos otros ámbitos, incluyendo el de la vida cotidiana. De hecho, algunos de los ejemplos utilizados por los investigadores en este tema están inspirados en situaciones cotidianas. Un ejemplo de ello es el conocido problema de los filósofos comensales, propuesto por Dijkstra en 1965, y ya estudiado en el capítulo dedicado a la comunicación y sincronización de procesos. En esta sección se presentará un ejemplo de interbloqueo extraído de la vida real para que sirva como una introducción intuitiva a este tema.

Uno de los ámbitos que proporciona más ejemplos es el del tráfico de vehículos. De hecho, uno de los objetivos de algunas señales de tráfico, e incluso de algunas normas de circulación, es resolver los posibles interbloqueos entre los vehículos. Nótese que, en este ámbito, el interbloqueo causaría la detención permanente de los vehículos implicados. Al fin y al cabo, un atasco de tráfico es un caso de interbloqueo.

Como ejemplo, considérese una carretera con dos sentidos de circulación que atraviesa un largo puente estrecho por el que solo cabe un vehículo. Como se aprecia en la figura, el interbloqueo se produciría en el momento en el que dos vehículos atravesando de forma simultánea el puente en sentido contrario se encontraran el uno frente al otro sin posibilidad, por tanto, de continuar.

Figura 46. *Atasco al cruzar un puente estrecho*



Fuente: (Studocu, s.f.)

Nótese que cada vehículo posee un recurso (el tramo del puente que ya ha cruzado hasta el momento), pero necesita otro recurso (el tramo que le queda por cruzar) para terminar su labor (cruzar el puente). El interbloqueo surge debido a que se produce un conflicto entre las necesidades de los dos vehículos: el recurso que necesita cada vehículo lo posee el otro. Hay que resaltar que otros vehículos que intentaran cruzar el puente en ese momento en cualquiera de los dos sentidos se quedarían detenidos detrás de ellos, viéndose, en consecuencia, implicados también en el interbloqueo. Sobre este ejemplo, se puede plantear una primera idea general de cuáles son las posibles estrategias para tratar este problema:

Detección y recuperación. En esta estrategia, se requiere algún elemento externo que actúe de detector del problema y gestor del mismo, que, en el ejemplo planteado, podría ser un agente de circulación. Una vez que el agente detecta la situación de interbloqueo, debe hacer que uno de los vehículos libere el recurso que posee para dejar que el otro lo utilice.

Una posible recuperación de esta situación consistiría en seleccionar uno de los sentidos de circulación, y ordenar que el vehículo o vehículos detenidos en ese sentido dieran marcha atrás hasta el principio del puente, liberando así el paso en el otro sentido (se está suponiendo que un vehículo tiene capacidad para avanzar marcha atrás; si no fuera así, habría que tomar una acción más drástica como tirarlo al río que pasa por debajo del puente). El agente debería utilizar algún criterio para determinar qué vehículos deben retroceder.

Para ello, se podrían tener en cuenta aspectos tan diversos como cuánta distancia ha recorrido cada vehículo dentro del puente, qué velocidad tiene cada vehículo, cuál de ellos tiene mayor prioridad (por ejemplo, en el caso de una ambulancia) o cuántos vehículos hay en cada sentido. Nótese que cada vehículo afectado incurriría en un gasto extra de combustible y de tiempo debido al primer intento frustrado de cruzar el puente, junto con el recorrido de vuelta hasta el principio del mismo. Esta estrategia, por tanto, tiene un carácter que se podría considerar destructivo, ya que se pierde parte del trabajo realizado por una entidad, e incurre en el coste asociado a la existencia de un elemento detector externo.

Prevención. Suponga que, en el ejemplo, se instala un sistema de señalización basado en semáforos (reales, no informáticos, claro), que cambian su estado periódicamente, en cada entrada del puente, de manera que se imposibilita que haya simultáneamente vehículos en distintos sentidos dentro del puente. Esta estrategia preventiva no tiene el carácter destructivo de la anterior, ni requiere el uso de un elemento activo detector, pero puede implicar una infrautilización de los recursos. En el ejemplo planteado, puede ocurrir que el uso de semáforos haga que un vehículo tenga que detenerse a la entrada del puente, aunque este esté libre.

Predicción. Un punto importante a resaltar en este ejemplo, y en general sobre los interbloqueos, es que, antes de producirse el interbloqueo propiamente dicho (los vehículos detenidos frente a frente), existe un “punto de no retorno”, a partir del cual el interbloqueo es inevitable. En el ejemplo, habiendo uno o más vehículos atravesando el puente en un determinado sentido, el punto de no retorno ocurriría en el momento en que un vehículo entrase en el puente en sentido contrario. A partir de ese momento, tarde o temprano, se produciría un interbloqueo. Las estrategias basadas en la predicción evitan el interbloqueo asegurándose de que no se llega a este punto de no retorno.

En el ejemplo, se podría usar para ello un sistema de semáforos inteligentes, de manera que, en vez de cambiar su estado de forma periódica, lo hagan según circulan los vehículos: estando los semáforos de ambas entradas en verde, cuando cruza un vehículo

una de las entradas, se pondrá en rojo el semáforo de la otra. En principio, esta solución parece la panacea, ya que no conlleva infrautilización de recursos ni pérdida de trabajo.

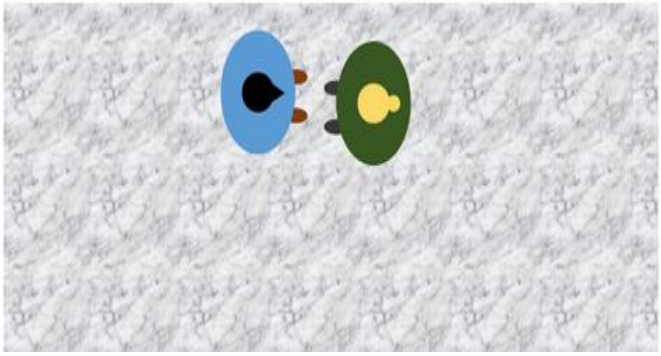
Sin embargo, como veremos más adelante, es la que menos se podrá aplicar al campo de los sistemas operativos, puesto que requiere que los procesos declaren por anticipado qué recursos van a usar (es como si, en el ejemplo, al iniciar un viaje, el conductor se viera obligado a declarar toda su ruta para así conocer a priori por qué puentes de un solo sentido va a pasar durante el trayecto).

Más adelante, se presentará un problema relacionado con el interbloqueo, el livelock, de una manera más formal, pero se considera conveniente introducirlo en este punto planteando un ejemplo cotidiano que permita desde el principio resaltar las similitudes y diferencias entre ambos problemas.

Considere que, como se muestra en la figura, dos personas se cruzan por un pasillo donde hay sitio suficiente para pasar simultáneamente, pero, sin embargo, se encuentran una frente a otra, por lo que no pueden progresar. De forma natural, ambas personas intentarían moverse aleatoriamente hacia algún lado para evitar esa colisión. Sin embargo, podría ocurrir que ambas se desplazaran lateralmente de la misma forma, con lo que se repetiría el problema.

Con muy mala suerte, estos reintentos podrían frustrarse repetidamente de manera que las dos personas nunca cruzarían el pasillo, aunque lo más probable es que en algún momento no haya colisión y puedan atravesarlo satisfactoriamente. Nótese que esta patología tiene similitudes con el interbloqueo, en el sentido de que hay entidades compitiendo por el uso exclusivo de recursos que se ven involucradas en una situación en la que no progresan. Sin embargo, la situación del interbloqueo es permanente, requiriendo siempre algún tipo de intervención externa, mientras que la del livelock no tiene por qué ser definitiva, pudiendo resolverse en cualquier reintento.

Figura 47. *Problema del pasillo*



Fuente: (Studocu, s.f.)

Escenarios de interbloqueos

Como se ha podido apreciar en la sección anterior, un escenario donde pueden aparecer interbloqueos se caracteriza por la existencia de un conjunto de entidades activas (los vehículos) que realizan acciones (entrar y salir del puente) sobre un conjunto de recursos de uso exclusivo (el puente estrecho) para llevar a cabo su labor. De manera similar, en un sistema informático existirán estos mismos elementos:

- Las entidades activas, que corresponden a los procesos existentes en el sistema. Es importante resaltar que en un sistema operativo que proporcione threads, estos representarán las entidades activas, ya que son la unidad de ejecución del sistema. Sin embargo, hay que tener en cuenta que en un sistema existen otro tipo de actividades no vinculadas directamente con el código de los procesos que pueden verse involucradas en un interbloqueo, como, por ejemplo, la ejecución de la rutina de tratamiento de una interrupción. Por tanto, para generalizar, a lo largo del tema se usará el término “proceso” para referirse a cualquier flujo de ejecución o actividad que pueda verse implicada en un interbloqueo.
- Los recursos de uso exclusivo existentes en el sistema, que serán utilizados por los procesos para realizar su labor.
- Las acciones u operaciones que llevan a cabo los procesos sobre los recursos, que les permiten obtener y liberar dichos recursos.

A la hora de analizar el problema del interbloqueo y sus repercusiones, es conveniente, desde el principio, tener en cuenta los distintos contextos en los que puede aparecer, y la responsabilidad de su tratamiento en cada caso:

- Procesos independientes que compiten en el uso de recursos. En este caso, el único responsable de afrontar el problema será el sistema operativo. Por tanto, si se produce un interbloqueo, se deberá a que, ya sea por error o conscientemente, el sistema operativo ofrece unos servicios que no lo impiden.
- Procesos, o threads, cooperantes que forman parte de una misma aplicación concurrente. En este caso, el diseño de la aplicación debería haber tenido en cuenta este problema y su aparición es un error en el mismo.
- Ejecución concurrente del código del sistema operativo que realiza el tratamiento de las llamadas al sistema que realizan los procesos y las excepciones que estos provocan, así como el de las interrupciones generadas por los dispositivos. Un problema de interbloqueo en este contexto corresponde a un error en el diseño del sistema operativo.

Como se analizará en la parte final del capítulo, en todos estos ámbitos se aplicarán algunas de las estrategias que se irán presentando a lo largo del mismo. Sin embargo, habrá ciertos matices en cómo aplicarlas y en las repercusiones que tienen los interbloqueos en cada caso.

El objetivo de esta sección es introducir el problema del interbloqueo mostrando situaciones de muy diversas características donde puede aparecer esta patología. La colección de escenarios planteados va a permitir apreciar las peculiaridades con las que se presenta este problema y va a servir como una clasificación a la que se hará referencia en el resto del tema cuando se analicen los distintos tipos de soluciones y la aplicación de las mismas. Asimismo, en la parte final de la sección, como un escenario adicional, se introducirá de manera más formal el problema del livelock.

Interbloqueo entre procesos independientes

En este apartado se plantea un ejemplo que afecta a dos procesos independientes que utilizan dos recursos de uso exclusivo. Supóngase que existen dos procesos, P1 y P2, tal que ambos durante su ejecución necesitan utilizar una cinta (C) y una impresora (I), que son, por razones obvias, recursos de uso exclusivo. Estos procesos usarán operaciones que les permitan solicitar el uso exclusivo del recurso y liberarlo cuando ya no lo requieran. Considere, asimismo, que estos procesos se comportan según la siguiente traza de ejecución, en la que en algunos momentos solo requieren uno de los recursos, mientras que en otros instantes necesitan usar ambos:

Proceso P_1	Proceso P_2
Solicita(C)	**Solicita**(I)
Uso del recurso C	Uso del recurso I
Solicita(I)	**Solicita**(C)
Uso de ambos recursos	Uso de ambos recursos
Libera(I)	**Libera**(C)
Uso del recurso C	Uso del recurso I
Libera(C)	**Libera**(I)

Durante la ejecución de estos procesos en un sistema multiprogramado, se puede producir un interbloqueo, ya que se puede llegar a una situación en la que el primer proceso tiene asignada la cinta y está bloqueado esperando por la impresora, mientras que el segundo tiene la impresora, pero está esperando por la cinta. A continuación, se muestra un posible orden de ejecución de los procesos que produciría el interbloqueo:

1. P1: Solicita(C)

2. P2: Solicita(I)

3. P2: Solicita(C) $\rightarrow$ se bloquea puesto que el recurso no está disponible

4. P1: Solicita(I) $\rightarrow$ se bloquea por recurso no disponible: hay interbloqueo

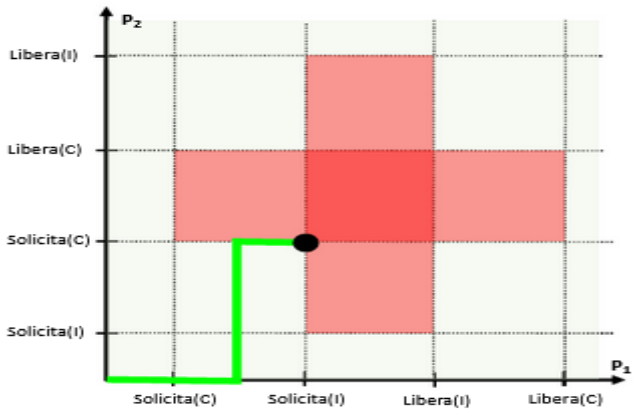
En un sistema con un único procesador, este orden de ejecución entrelazado, característico de los interbloqueos, se debería a que se producen cambios de contexto en los instantes correspondientes. En un multiprocesador, podría ocurrir debido a que cada proceso

ejecuta en un procesador diferente. De hecho, a este tipo de interbloqueos entre dos procesos se les suele denominar ABBA precisamente por ese orden de ejecución intercalado que los caracteriza.

Una manera de ilustrar este problema para el caso de solo dos procesos es usar un diagrama de la trayectoria de los mismos en el que se muestra gráficamente su ejecución como una línea que avanza hacia la derecha cuando ejecuta el primer proceso y hacia arriba cuando lo hace el segundo (en un multiprocesador podría avanzar de forma simultánea en ambos sentidos).

En la siguiente figura se muestra con una línea continua gruesa la traza de ejecución de los dos procesos del ejemplo. En ella se puede identificar el interbloqueo como un círculo a partir del cual no es posible que progrese ningún proceso puesto que la trayectoria no puede avanzar ni hacia la derecha ni hacia arriba, al no poderse atravesar la zona sombreada que representa escenarios no factibles debido al uso exclusivo de los recursos.

Figura 48. *Interbloqueo entre procesos independientes*



Fuente: (Studocu, s.f.)

Es interesante resaltar que no todos los posibles órdenes de ejecución de estos dos procesos causarían un interbloqueo. Así, por ejemplo, si el orden de ejecución fuera el siguiente:

1. P1: Solicita(C)

2. P1: Solicita(I)

3. P2: Solicita(I) → se bloquea puesto que el recurso no está disponible

4. P1: Libera(I) → se desbloquea P2 puesto que el recurso ya está disponible

5. P2: Solicita(C) → se bloquea puesto que el recurso no está disponible

6. P1: Libera(C) → se desbloquea P2 porque el recurso ya está disponible

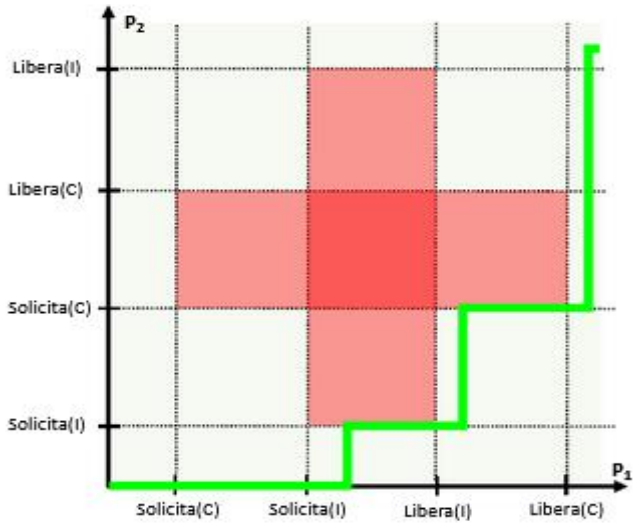
7. P2: Libera(C)

8. P2: Libera(I)

Los dos procesos habrían terminado realizando su labor sin producirse interbloqueos.

Nótese que hay que diferenciar claramente entre el bloqueo de un proceso debido a la falta de un recurso (como ocurre con P2 en los pasos 3 y 5 anteriores), que terminará cuando dicho recurso esté disponible, y el interbloqueo, que implica el bloqueo permanente de los procesos involucrados. La siguiente figura muestra el diagrama de trayectoria de los procesos para esta traza de ejecución libre de interbloqueo.

Figura 49. *Interbloqueo entre procesos independientes*



Fuente: (Studocu, s.f.)

Es interesante resaltar ya desde el primer ejemplo dos aspectos que caracterizan a las situaciones donde pueden aparecer interbloqueos:

- Los procesos involucrados piden los recursos en un orden diferente. Evidentemente, este orden viene determinado por las necesidades de cada proceso.

- Los procesos requieren utilizar en algún punto de su ejecución varios recursos de carácter exclusivo simultáneamente. Como se analizará más adelante, dado el carácter independiente de los procesos involucrados, el único responsable de tratar este problema es el sistema operativo.

Interbloqueo entre procesos o threads de la misma aplicación

A continuación, se plantea un ejemplo que se corresponde con un escenario donde los flujos de ejecución involucrados forman parte de una misma aplicación. Considere el siguiente programa con tres threads que usan tres cerrojos de tipo mutex para sincronizarse (en UNIX se usarían los servicios pthread_mutex_lock y pthread_mutex_unlock):

Thread P_1	Thread P_2	Thread P_3
lock(Ma)	**lock**(Mb)	**lock**(Mc)
$tarea_{11}$	$tarea_{21}$	$tarea_{31}$
lock(Mb)	**lock**(Mc)	**lock**(Ma)
$tarea_{12}$	$tarea_{22}$	$tarea_{32}$
unlock(Ma)	**unlock**(Mb)	**unlock**(Mc)
$tarea_{13}$	$tarea_{23}$	$tarea_{33}$
unlock(Mb)	**unlock**(Mc)	**unlock**(Ma)

El siguiente orden de solicitud de recursos causaría un interbloqueo que afectaría a los tres threads:

1. P1: lock (Ma)

2. P2: lock (Mb)

3. P3: lock (Mc)

4. P3: lock (Ma) $\rightarrow$ se bloquea puesto que el recurso no está disponible

5. P1: lock (Mb) $\rightarrow$ se bloquea puesto que el recurso no está disponible

6. P2: lock (Mc) $\rightarrow$ se bloquea por recurso no disponible: hay interbloque

Es importante resaltar en el ejemplo que el primer thread requiere ejecutar tanto el código correspondiente a tarea11 como a tarea12 manteniendo todo ese tiempo la posesión del mutex. Ma. No sería válida, por tanto, una solución que intente eliminar el interbloqueo haciendo que dicho thread perdiera momentáneamente la posesión de ese cerrojo al finalizar el primer fragmento de código para recuperarla inmediatamente justo antes de empezar el segundo.

En este tipo de escenarios, aunque el sistema operativo pueda ofrecer soporte para facilitar el tratamiento de este problema, la responsabilidad final corresponde a los desarrolladores del programa: el diseño de la aplicación debe tenerlo en cuenta y su aparición es un error en el mismo.

Interbloqueo con espera activa

En los ejemplos previos, los flujos de ejecución involucrados en un interbloqueo se quedaban bloqueados de forma permanente. Sin embargo, hay que remarcar que el hecho de que los procesos involucrados pasen a un estado de bloqueados no es un aspecto consustancial del interbloqueo, sino que también puede aparecer este problema en situaciones donde los procesos no se bloquean y usan espera activa para resolver los conflictos de acceso a los recursos.

Considere que, en el primer ejemplo planteado, la operación de petición de un recurso no es bloqueante y devuelve un valor de verdadero o falso dependiendo de si se ha podido asignar el recurso o no. En este caso, los procesos usarán un bucle de espera activa hasta que se les haya asignado el recurso solicitado:

Proceso P1	Proceso P2
while (Solicita(C)==false);	while (Solicita(I)==false);
Uso del recurso C	Uso del recurso I
while (Solicita(I)==false);	while (Solicita(C)==false);

Uso de ambos recursos	Uso de ambos recursos
Libera(I)	Libera(C)
Uso del recurso C	Uso del recurso I
Libera(C)	Libera(I)

A pesar de que los procesos no se bloquean, se puede dar la misma situación de interbloqueo que en el ejemplo bloqueante, pero encima con espera activa, que conlleva un gasto inútil del procesador1. P1: Solicita(C) → verdadero puesto que el recurso está disponible.

1. P1: Solicita(C) → verdadero puesto que el recurso está disponible

2. P2: Solicita(I) → verdadero puesto que el recurso está disponible

3. P2: Solicita(C) → falso ya que el recurso no está disponible

4. P1: Solicita(I) → falso ya que el recurso no está disponible: interbloqueo

Asimismo, en el segundo ejemplo, si se sustituye el uso de mutex por spinlocks, un mecanismo de sincronización similar pero que usa espera activa y se utiliza principalmente para la sincronización interna del sistema operativo (dentro del núcleo de Linux, correspondería a las funciones spin_lock y spin_unlock), se mantendrá la misma patología.

Auto-interbloqueo

El interbloqueo puede afectar a cualquier número de procesos, e incluso puede aparecer, aunque solo intervenga uno. Suponga que un proceso que está en posesión de un cerrojo de tipo mutex vuelve, por error, a intentar adquirirlo:

Proceso P

lock(M)

.....

lock(M)

En algunas implementaciones, el proceso se quedará en estado de interbloqueo (un auto-interbloqueo). Otras, sin embargo, detectarían este caso trivial de interbloqueo y devolverían un error en la segunda llamada. En principio, parecería que este escenario patológico es un error de programación obvio que es relativamente fácil de evitar.

Sin embargo, hay que tener en cuenta que en una aplicación multithread de una cierta entidad, en la que participan varios programadores y se usan bibliotecas de terceros, es frecuente que puedan producirse errores en los que un thread que ya tiene asignado un mutex, invoque una función que, sin que el programador que desarrolló el código que realiza la llamada sea consciente de ello, vuelve a intentar establecerlo. Aunque la implementación detecte este problema y se evite el interbloqueo del thread consigo mismo, el funcionamiento de la aplicación puede ser erróneo.

Para entenderlo, vamos a suponer que en el programa existe una determinada función F que durante su ejecución usa un mutex M:

```
F () {

.....

lock(M)

.....

unlock(M)

.....
```

Si un thread que ya tiene asignado ese mutex invoca esta función, o bien entra en un interbloqueo, o bien se producirá un error en la llamada lock de la función, en caso de que la implementación compruebe ese interbloqueo trivial. En ese último caso, si el código de la función no comprueba el error devuelto por lock y prosigue su ejecución, la llamada unlock dentro de la función no fallará, sino que liberará el mutex, por lo que, al retornar de la función, el thread que la invocó ha perdido, erróneamente, el uso del mutex.

Los mutex recursivos resuelven este problema, ya que en este nuevo tipo de mutex, una llamada lock de un proceso que ya tiene asignado el mutex no lo bloquea ni devuelve un error, sino que, simplemente, actualiza un contador asociado al mutex que refleja cuántas veces ha solicitado adquirir ese mutex el proceso que lo tiene asignado. El mutex solo se liberará cuando se produzcan el mismo número de llamadas unlock de ese mismo proceso. Sin embargo, hay que resaltar que este tipo de mutex también tiene sus detractores porque, además de ser menos eficientes, hacen que el programador pierda el control del ámbito y duración de la posesión de un cerrojo.

Interbloqueo en el tratamiento de eventos asíncronos

Un escenario de interbloqueo especialmente interesante es el que aparece en el tratamiento de eventos asíncronos, como son las señales o, internamente, las interrupciones. Como bien es conocido, el uso de asincronía en cualquier ámbito de la informática plantea siempre retos complejos. En este escenario puede haber problemas de sincronización entre el flujo interrumpido y la rutina que trata el evento asíncrono si ambos flujos comparten información, ya sea directamente o a través del uso de una función común, y en la resolución de estos problemas de sincronización pueden aparecer situaciones de interbloqueo.

A continuación, se plantea un ejemplo de un interbloqueo vinculado con la recepción de una señal. Considérese un programa que utiliza una biblioteca que gestiona internamente un cerrojo de tipo mutex para asegurar la coherencia de sus datos internos cuando se usa en un entorno con múltiples threads (gracias al uso del mutex, múltiples threads pueden trabajar con una biblioteca que de por sí no es reentrante). Suponga que tanto el thread principal del programa como la rutina de tratamiento de una señal invocan una determinada función de esa biblioteca:

Proceso	Rutina de tratamiento de la señal
.....	.....
f ();	f ();
.....	.....
f () {	
lock(M)	
tarea ←señal	
unlock(M)	
.....}	

Si en el ejemplo se genera una señal que interrumpe la llamada a la función por parte del proceso cuando está en posesión del mutex, se producirá un interbloqueo. En el momento que la rutina de tratamiento de la señal invoque la función y esta intente solicitar ese mismo cerrojo, no podrá continuar al no estar disponible; pero, al bloquearse, detendrá también al proceso, que es el único que puede liberar ese recurso, por estar ejecutándose ambos flujos en el mismo contexto, quedándose bloqueados permanentemente.

Dependiendo de las características del sistema, en vez de un interbloqueo, el sistema podría haber devuelto un error en la solicitud del mutex desde la rutina de tratamiento de la señal, al considerarlo un auto-interbloqueo (el tratamiento de la señal ejecuta en el mismo contexto), lo cual no arreglaría el problema. Tampoco lo haría definir el mutex como recursivo, ya que, en ese caso, aunque no se produzca un interbloqueo ni falle ninguna solicitud, la función f se ejecutará de forma concurrente, para lo que no está diseñada.

En el ejemplo, el conflicto aparece debido a que la rutina de tratamiento de la señal usa, indirectamente, el mismo cerrojo de tipo mutex que el flujo del thread interrumpido. Sin embargo, puede haber escenarios de interbloqueo más complejos y sutiles donde no se dé

esta circunstancia, como en el siguiente ejemplo, donde un programa usa dos bibliotecas, cada una protegida por su propio mutex, y cada thread es interrumpido por una señal cuya rutina de tratamiento no usa, indirectamente, el mismo mutex que el flujo interrumpido:

Thread P1	Thread P2	Señal S1
.....	..... .	.....
f ();	g ();	g ();
.....	.....	.....
f () {	g () {	Señal S2
lock (Ma)	lock (Mb)	.....
tarea1 ←S1	tarea2 ←S2	f ();
unlock (Ma)	unlock (MB)	.....
.....}	.....}	

Acto seguido, se muestra un posible orden de ejecución que causaría un interbloqueo de tipo ABBA:

1. P1: lock (Ma)

2. P2: lock (Mb)

3. S2: lock (Ma) → se bloquea P2 puesto que el recurso no está disponible

4. P1: tarea1

5. S1: lock (Mb) → se bloquea P1 por recurso no disponible: hay interbloqueo

Nuevamente, en estos escenarios sería el desarrollador del programa el responsable de evitar este tipo de problemas.

Interbloqueo en la operación interna del sistema operativo

Un escenario crítico para el interbloqueo es cuando este se produce dentro del código del propio sistema operativo, puesto que puede afectar al buen funcionamiento de todo el sistema. El sistema operativo debe tratar eventos tales como llamadas a sistema, excepciones e interrupciones y en el tratamiento de los mismos puede surgir el temido interbloqueo.

Considere, por ejemplo, que, como es habitual, el sistema operativo ofrece una llamada al sistema que permite renombrar un fichero, moviéndolo de directorio si el cambio de nombre así lo requiere. Para asegurar la correcta sincronización en una llamada de esta índole, en caso de que se precise mover el fichero de directorio, hay que bloquear el acceso a los dos directorios involucrados (en caso contrario, podrían ocurrir situaciones patológicas como, por ejemplo, que una vez eliminado el fichero en el directorio origen, pero antes de darlo de alta en el destino, se cree una nueva entrada con el mismo nombre en ese segundo directorio):

```
renombrar (rutaPrevia, rutaNueva) {
        dirOrg = directorio padre de rutaPrevia
        dirDst = directorio padre de rutaNueva
        if (dirOrg != dirDst) {
                Bloquea acceso a dirOrg
                Bloquea acceso a dirDst
                Elimina entrada rutaPrevia de dirOrg
                Añade entrada rutaNueva en dirDst
                Desbloquea acceso a dirOrg
                Desbloquea acceso a dirDst}
        else .....}
```

Suponga que dos procesos realizan las siguientes llamadas:

P1: renombrar (“/dir1/fA”, “/dir2/fB”);

P2: renombrar (“/dir2/fC”, “/dir1/fD”);

El siguiente orden de ejecución de las acciones internas asociadas a cada llamada causaría un interbloqueo que, además de los procesos que realizaron la llamada, bloquearía permanentemente el acceso a ambos directorios:

1. Llamada de P1: Bloquea acceso a “/dir1”

2. Llamada de P2: Bloquea acceso a “/dir2”

3. Llamada de P2: Bloquea acceso a “/dir1” $\rightarrow$ se bloquea

4. Llamada de P1: Bloquea acceso a “/dir2” $\rightarrow$ interbloqueo

Dentro del sistema operativo, también se producen escenarios de interbloqueos vinculados con el tratamiento de las interrupciones, que, dado su carácter asíncrono, presentan similitudes con los problemas analizados en el apartado anterior, y se estudiarán en la parte final del tema.

Evidentemente, en ese caso, el único responsable de evitar este problema es el equipo desarrollador del sistema operativo y la presencia del mismo es un error en su programación.

Interbloqueo por la competencia en el uso de un recurso

Este apartado presenta un escenario de interbloqueo un poco diferente a los mostrados previamente. En vez de estar directamente relacionado con problemas de sincronización, está vinculado con situaciones en las que se entra en conflicto a la hora de reservar de manera incremental unidades de un recurso limitado. Este escenario se caracteriza por lo siguiente:

- Existe un recurso del que se dispone de un número limitado de unidades (N).
- Cada proceso a lo largo de su vida irá reservando y liberando unidades según las vaya necesitando. Puntualmente, podría llegar a necesitar hasta un cierto número máximo de unidades (M, que, por simplificar, consideramos igual para todos los procesos y que, obviamente, debería ser menor o igual que N).
- Dado que los procesos solo requieren ocasionalmente usar ese número máximo de unidades, por economía, es conveniente que el número total de unidades disponibles sea inferior al máximo total requerido por todos los procesos existentes (si hay P procesos, P x M > N), para evitar que se infrautilicen las unidades de ese recurso.

Esta condición refleja precisamente la esencia del sistema operativo: repartir un número limitado de recursos (procesadores, memoria, dispositivos, etc.) entre los procesos existentes. Asimismo, esta característica aparece también en otros contextos afines. Así, por ejemplo, en un entorno de virtualización, la suma total de la memoria física que creen manejar las distintas máquinas virtuales activas puede ser mayor que la cantidad de memoria física realmente existente (memory overcommitment).

- Si un proceso solicita unidades de un recurso y no están disponibles en ese momento, se bloqueará hasta que otro proceso las libere.

En un escenario con estas características se produciría un interbloqueo si se diese una situación en la que todos los procesos, teniendo ya asignadas algunas unidades del recurso, solicitaran unidades adicionales y ninguna petición pudiera satisfacerse. Es interesante resaltar dos aspectos de esta situación patológica:

- El interbloqueo se produce porque los procesos van reservando incrementalmente las unidades de los recursos según las van requiriendo: tienen ya asignado un conjunto de unidades, pero necesitan más. Si cada proceso solicitase de golpe el número máximo de unidades que puede necesitar, no podría darse el interbloqueo: algunos procesos obtendrían todo lo que requieren, mientras que otros se quedarían bloqueados, pero solo hasta que los primeros liberasen las unidades que poseen.

- No es necesario disponer del máximo número total de unidades necesitadas por los procesos (P x M) para asegurarse de que no se puede producir un interbloqueo. Bastaría con que existan P x (M-1) + 1, puesto que con esa cantidad se garantizaría que, en el peor de los casos posibles (todos los procesos han solicitado, y tienen asignadas, sus necesidades máximas menos una unidad, P x (M-1), y ahora piden una adicional), al menos un proceso satisface sus necesidades máximas. Así, por ejemplo, con 10 procesos que usan un máximo de 5 unidades de un recurso, en vez de 50, sería suficiente con 41 unidades para garantizar que no puede haber un interbloqueo.

En un sistema operativo de propósito general, algunos recursos que responden a estas características (es decir, un recurso con un número limitado de unidades) son la memoria, el disco o los procesadores. A continuación, se muestra un ejemplo de interbloqueo en el uso de la memoria. Considere la ejecución de los dos procesos siguientes en un sistema que no usa memoria virtual y donde las solicitudes de memoria se bloquean si no hay espacio suficiente en ese momento, suponiendo que se dispone inicialmente de 450KB de memoria.

Proceso P1	Proceso P2
Solicita(100K)	Solicita(200K)
Solicita(100K)	Solicita(100K)

Si se produce un orden de ejecución tal que se satisfacen las dos primeras solicitudes del primer proceso y la primera del segundo, se produciría un interbloqueo, puesto que la cantidad de memoria libre (50KB) no es suficiente para cumplir con ninguna de las peticiones pendientes. En un sistema con memoria virtual, como se estudia en el capítulo correspondiente a esa materia, el sistema operativo resolvería este problema liberando parte de la memoria asignada a un proceso y copiándola a disco.

El sistema operativo, usando todas las técnicas requeridas por la multiprogramación que se estudian en los distintos capítulos del libro, hace que este tipo de problemas no afecten directamente a los procesos de los usuarios, tratándose más bien de un problema de gestión interno, como se analizará en la parte final del tema.

Por último, conviene resaltar que, exceptuando elementos como la memoria, el sistema operativo trata los recursos de forma individualizada. Así, por ejemplo, en un sistema en el que existan tres impresoras, no considera que se trata de un único recurso que dispone de tres unidades, sino de tres recursos independientes. Por ello, el tratamiento del interbloqueo para recursos compuestos de múltiples unidades se analiza únicamente en la sección que estudia los aspectos más avanzados de este campo

Interbloqueo en la comunicación de procesos

Un escenario donde también se pueden producir interbloqueos es en la comunicación de procesos. Generalmente, se trata de situaciones en las que, debido a un fallo en el diseño del protocolo de comunicación o en la implementación del mismo, todos los procesos involucrados se quedan a la espera de un mensaje que nunca llegará, puesto que lo debería enviar uno de esos procesos bloqueados.

Considere un sistema de comunicación que proporciona funciones para enviar y recibir mensajes en las que se especifica como primer parámetro el proceso al que se le quiere enviar o del que se quiere recibir, respectivamente, y como segundo el espacio de almacenamiento asociado al mensaje enviado o recibido, dependiendo del caso (observe cierta similitud con las operaciones de comunicación ofrecidas por el estándar MPI, así como una funcionalidad parecida a la de los sockets stream). Suponga que se ejecutan en este sistema los siguientes tres procesos:

Proceso P1	Proceso P2	Proceso P3
Envía (P3, A)	Recibe (P1, D)	Recibe (P2, F)
Recibe (P3, B)	Envía (P3, E)	Envía (P1, G)
Envía (P2, C)		Recibe (P1, H)

La ejecución de estos tres procesos produciría un interbloqueo de los mismos, con independencia de cuál sea su orden de ejecución. El proceso P3 se quedaría bloqueado indefinidamente esperando el mensaje de P2, ya que este no lo puede mandar hasta que no reciba un mensaje de P1, que, a su vez, no puede hacerlo porque debe antes recibir un

mensaje de P3. Cada proceso se queda bloqueado esperando un mensaje que solo puede enviarlo otro de los procesos implicados, lo que no puede ocurrir, ya que dicho proceso está también bloqueado esperando un mensaje.

Es importante resaltar que, en este ejemplo, a diferencia de los planteados previamente, se produce el interbloqueo con independencia del orden en que se ejecuten los procesos, al tratarse de un error en el diseño del patrón de comunicaciones entre los procesos. En una aplicación relativamente compleja que conste de múltiples procesos comunicándose entre sí, pueden darse errores de este tipo que sean bastante difíciles de detectar.

Un aspecto que conviene remarcar es cómo influyen las características del sistema de comunicación en la posibilidad de que se produzcan interbloqueos. Suponga que se usa un sistema donde la recepción del mensaje es bloqueante, pero el envío no puede causar el bloqueo del proceso, ya sea porque se utiliza una operación de envío asíncrona, que devuelve el control inmediatamente, o una con buffering, que, antes de hacerlo, copia sin bloquearse el mensaje en un buffer (en el caso del estándar MPI, correspondería a las operaciones MPI_Isend y MPI_Bsend, respectivamente). Considere que se ejecutan en este sistema los siguientes tres procesos:

Proceso P1	Proceso P2	Proceso P3
Envía (P2, A)	Envía (P3, B)	Envía (P1, C)
Recibe (P3, D)	Recibe (P1, E)	Recibe (P2, F)

Dadas las características del sistema de comunicación planteado, no se producirá interbloqueo. Sin embargo, si en este sistema el envío del mensaje es síncrono (en el caso de MPI correspondería a la operación MPI_Ssend), es decir, que la función de envío no se completa hasta que el destinatario invoque la operación de recepción, la ejecución de estos tres procesos produciría un interbloqueo de los mismos, ya que todos se quedarían

bloqueados en la operación de envío a la espera de que el destinatario correspondiente ejecute la operación de recepción, lo que no puede hacer al estar bloqueado.

Téngase en cuenta que en el caso de usar una operación de envío con buffering, pero tal que los buffers usados para la comunicación tengan un tamaño limitado (como ocurre, con los sockets stream), también podría ocurrir un interbloqueo en ese ejemplo si se llenan los buffers de comunicación debido a mensajes de tamaño grande (o muchos mensajes pequeños): al estar llenos los buffers de comunicación, para completarse el envío se necesita que el proceso destinatario reciba el mensaje, pero no lo puede hacer al estar también bloqueado porque, recíprocamente, en su envío se ha encontrado llenos los buffers asociados al sentido contrario de la comunicación.

Nótese que este escenario no solo ocurre cuando se produce comunicación remota, sino que aparecería también usando un recurso local como una tubería. En general, la gestión de los buffers de un sistema de comunicación de manera que se garantice que no se puede producir un interbloqueo en ninguna circunstancia es una labor compleja.

Si se quiere mantener el uso de una operación de envío síncrona, habría que cambiar el orden de las operaciones para que desaparezca el interbloqueo:

Proceso P1	Proceso P2	Proceso P3
Envía (P2, A)	Recibe (P3, B)	Recibe (P1, C)
Recibe (P3, D)	Envía (P1, E)	Envía (P2, F)

Dadas las dificultades específicas que conlleva el tratamiento de los interbloqueos en la comunicación de procesos, se estudiará únicamente en la sección que presenta los aspectos más avanzados de este tema.

Livelock vs Deadlock

Como se explicó previamente, el problema del livelock está muy relacionado con el del interbloqueo, aunque presenta ciertas diferencias. Esta patología se caracteriza porque los procesos involucrados no están progresando debido a un conflicto en el uso de recursos; pero, a diferencia del interbloqueo, esta situación no tiene por qué ser definitiva y podría

resolverse sin ninguna intervención externa si se dan las condiciones oportunas durante la ejecución de los procesos.

Tomando como punto de partida el ejemplo de interbloqueo con espera activa presentado previamente, se plantea, a continuación, una modificación del mismo donde, en vez de un interbloqueo, se produce un livelock:

Proceso P1	Proceso P2
while (Solicita(C)==false);	while (Solicita(I)==false);
Uso del recurso C	Uso del recurso I
if (Solicita (I)==false)	if (Solicita (C)==false)
Libera(C)	Libera(I)
Deshace trabajo hecho	Deshace trabajo hecho
Vuelve al inicio	Vuelve al inicio
Uso de ambos recursos	Uso de ambos recursos
Libera(I)	Libera(C)
Uso del recurso C	Uso del recurso I
Libera(C)	Libera(I)

Nótese que, en este caso, si no está disponible el segundo recurso, el proceso libera el que tiene asignado y vuelve a empezar solicitando de nuevo ese primer recurso. Con buena suerte, en el próximo reintento un proceso puede conseguir los dos recursos requeridos y completar su ejecución. Sin embargo, con malísima suerte, los procesos involucrados quedarían permanentemente sin progresar, reintentando continuamente sin éxito obtener los dos recursos necesitados.

De alguna manera, se puede interpretar en el ejemplo que los propios procesos, en vez de una entidad externa como el sistema operativo, han implementado su propia técnica para evitar el interbloqueo, liberando recursos e intentando volver a obtenerlos en caso de conflicto: han evitado el interbloqueo, pero creando un escenario donde puede producirse un livelock.

En cualquier caso, hay que resaltar que, en el momento en el que se decide volver al inicio para realizar un reintento, el proceso ya ha realizado un cierto uso del primer recurso. Por tanto, hay que deshacer de alguna manera ese trabajo llevado a cabo, puesto que se volverá a repetir en el próximo reintento. En numerosas ocasiones, como se analizará más adelante, en un sistema operativo de propósito general, no es factible realizar esta operación de deshacer el trabajo ya hecho. Sin embargo, en ciertas situaciones, no es necesario llevarla a cabo porque la operación realizada es repetible (idempotente) y no importa si vuelve a ejecutarse, ya que no tiene ningún efecto colateral.

Para disminuir la probabilidad de reincidir en este problema, se puede usar la técnica denominada exponential backoff, en la que, ante un conflicto, los procesos involucrados esperan un tiempo aleatorio, que puede crecer exponencialmente, antes de volver al inicio para hacer un reintento, de manera similar a las estrategias utilizadas para tratar las colisiones en algunos protocolos de red. Retomando el ejemplo del puente estrecho, si cuando hay un conflicto, los conductores de ambos vehículos, por su cuenta y sin la intervención de un agente, dan marcha atrás y vuelven al inicio de su extremo del puente, podrían esperar un tiempo aleatorio antes de volver a intentar cruzarlo para disminuir la posibilidad de que se repita el problema.

Dada la relación entre ambos problemas y puesto que el interbloqueo se puede considerar un problema más genérico y más ampliamente estudiado, el resto del capítulo se centra en el mismo, aunque volverá a surgir el problema del livelock puesto que, como se acaba de analizar, algunas soluciones al problema del interbloqueo pueden degenerar en un livelock.

Definición del interbloqueo

Como primer paso en el estudio del interbloqueo, parece conveniente intentar definirlo más formalmente para poder así caracterizarlo. Una posible definición de interbloqueo sería:

- Un conjunto de procesos está en interbloqueo si cada proceso está esperando recursos que solo puede liberar otro proceso del conjunto.

Normalmente, cada proceso implicado en el interbloqueo estará bloqueado esperando recursos, pero, como se explicó previamente, esto no es estrictamente necesario, ya que el interbloqueo también existiría, aunque los procesos involucrados realizasen una espera activa. La espera activa tiene como consecuencia un uso innecesario del procesador, pero, por lo que se refiere a los interbloqueos, no implica ninguna diferencia.

Como ya se analizó anteriormente, no hay que confundir el interbloqueo con otras situaciones patológicas en la asignación de recursos como el livelock. El interbloqueo representa una situación permanente de no progreso que requiere siempre algún tipo de intervención externa, mientras que en el livelock el escenario de no progreso no tiene por qué ser definitivo, pudiendo resolverse en cualquier reintento sin ningún apoyo externo. En cualquier caso, como ya se comentó previamente, hay cierta relación entre ambos problemas y, en algunas

circunstancias, uno puede degenerar en el otro. Téngase en cuenta que estas dos patologías se encuadran dentro del problema general de la inanición, en el que se enmarcan otras como, por ejemplo, situaciones en las que un proceso espera un plazo de tiempo no acotado para usar un recurso debido a una política de asignación de recursos que no es equitativa.

A partir de la definición, es preciso caracterizar un interbloqueo. Coffman identificó las siguientes condiciones como necesarias para que se produzca un interbloqueo:

Exclusión mutua. Los recursos implicados deben usarse en exclusión mutua, o sea, debe tratarse de recursos de uso exclusivo.

Retención y espera. Cuando no se puede satisfacer la petición de un proceso, este se bloquea manteniendo los recursos que tenía previamente asignados. Se trata de una

condición que refleja una forma de gestión de recursos que se corresponde con la usada en los sistemas reales.

Sin expropiación. No se pueden expropiar los recursos que tiene asignado un proceso.

Un proceso solo libera sus recursos voluntariamente.

Espera circular. Debe existir una lista circular de procesos, tal que cada proceso en la lista esté esperando por uno o más recursos que tiene asignados el siguiente proceso

Estas cuatro condiciones no son todas de la misma índole. Las tres primeras tienen que ver con aspectos estáticos del sistema, tales como qué características han de tener los recursos implicados o cómo debe ser la política de gestión de recursos. Sin embargo, la condición de espera circular refleja cómo debe ser el comportamiento dinámico de los procesos para que se produzca el interbloqueo. El interés de las tres primeras condiciones se podrá apreciar mejor cuando se analicen las estrategias de prevención del interbloqueo. Como se ha comentado al principio del tema, los estudios teóricos sobre esta materia plantean un modelo general que contempla características que van más allá de las presentes en un sistema operativo de propósito general.

Para facilitar la lectura de este tema a aquellos que no estén interesados en los aspectos más avanzados, pero permitiendo que el resto de lectores tengan acceso a todo el material teórico, en las dos secciones siguientes se van a plantear dos modelos de sistema incrementales: un modelo básico, que se correspondería con las características presentes en cualquier sistema operativo de propósito general, y un modelo extendido, que contempla un sistema con una funcionalidad más avanzada.

El lector que no esté interesado en profundizar en los aspectos teóricos puede saltarse la sección vinculada con el modelo extendido. En cuanto a aquellos interesados en los conceptos avanzados, deberían leer ambas secciones para poder apreciar de manera incremental el tipo de estrategias usadas para afrontar este problema.

Modelo básico del sistema

Desde el punto de vista del estudio del interbloqueo, en un sistema se pueden distinguir las siguientes entidades y relaciones:

- Un conjunto de procesos.
- Un conjunto de recursos de uso exclusivo.
- Un primer conjunto de relaciones entre procesos y recursos que define qué asignaciones de recursos están vigentes en el sistema en un momento dado. Esta relación define si un proceso tiene asignado un determinado recurso.
- Un segundo conjunto de relaciones entre procesos y recursos que define qué solicitudes de recursos están pendientes de satisfacerse en el sistema en un instante dado. Esta relación define si un proceso tiene un recurso pedido y no concedido.

En un sistema real, generalmente, no habrá una única función para solicitar recursos o liberarlos. Dada la gran heterogeneidad de los recursos, existirán servicios específicos para distintos tipos de recursos. Algunos ejemplos en UNIX de operaciones para manejo de recursos serían open y close para el acceso a dispositivos, pthread_mutex_lock y pthread_mutex_unlock para manejar cerrojos de tipo mutex, y fcntl para establecer y quitar cerrojos en ficheros, como se presenta en el capítulo dedicado al sistema de ficheros.

Para realizar el estudio formal de los interbloqueos, se van a plantear dos operaciones hipotéticas, independientes del tipo de cada recurso en particular o de un sistema operativo específico, que representan de forma genérica a todas estas funciones específicas: una operación de solicitud para pedir recursos y una de liberación para devolverlos:

- Solicitud (S(Ri)): Permite que un proceso, que, evidentemente, no está bloqueado, pida un recurso (Ri). Si dicho recurso está disponible, se concederá inmediatamente la petición asignándoselo al proceso. En caso contrario, se bloqueará el proceso hasta que esté disponible (como resultado de una operación de liberación).

- Liberación (L(Ri)): Permite que un proceso, que, evidentemente, no está bloqueado, libere un recurso que tenía asignado (Ri). La liberación puede causar que se satisfaga una solicitud pendiente de otro proceso, provocando su desbloqueo y la asignación del recurso.

Es importante resaltar que un sistema real tiene un carácter dinámico. En un sistema informático se están continuamente creando y destruyendo tanto procesos como recursos. Por tanto, las estructuras de información usadas para modelar el sistema tendrán también una evolución dinámica.

Representación mediante un grafo de asignación de recursos

Existen diversas maneras de representar la información asociada al modelo planteado. Aunque todas las representaciones son lógicamente equivalentes, la representación más habitual es el grafo de asignación de recursos, que resulta muy intuitiva al tener asociada una visualización gráfica directa del sistema y, como se verá en breve, se integra de forma relativamente natural en las estructuras de datos que gestiona el sistema operativo.

Un grafo de asignación de recursos G es un grafo bipartito dirigido que consta de un conjunto de nodos (o vértices) N y un conjunto de aristas (o arcos) A: G={{N}, {A}}. El conjunto de nodos N, de tamaño n, se descompone, a su vez, en dos subconjuntos disjuntos: el conjunto de procesos P, de tamaño p, y el de recursos R, de tamaño r. El conjunto de aristas también se divide en dos subconjuntos que se corresponden con las dos relaciones antes planteadas:

- Aristas de asignación, que relacionan recursos con procesos. Una arista desde un recurso Ri a un proceso Pj indica que ese proceso tiene asignado dicho recurso.

- Aristas de solicitud, que relacionan procesos con recursos. Una arista desde un proceso Pi a un recurso Rj indica que dicho proceso está esperando la concesión de ese recurso.

Obsérvese que en este modelo básico solo podrá haber una arista, sea de asignación o de solicitud, entre un proceso y un recurso. A partir de la especificación de las funciones genéricas de solicitud y liberación, se puede analizar cómo afectaría su procesamiento al grafo que representa el estado del sistema.

- Solicitud del proceso Pi del recurso Rj. Se presentan dos situaciones dependiendo de si el recurso pedido está disponible o no.
  - Si no lo está, se bloquea el proceso añadiendo al grafo una arista desde el nodo Pi hasta Rj. Cuando el proceso posteriormente se desbloquee al quedar disponible el recurso requerido, se eliminará del grafo esa arista de solicitud y se añadirá la misma arista de asignación que en el caso de que el recurso hubiese estado disponible desde el principio.
  - Si está disponible, ya sea desde el principio1 o posteriormente, se añade al grafo una arista desde el nodo Rj hasta Pi. Liberación por parte del proceso Pi del recurso Rj. Se elimina del grafo la arista desde el nodo Rj hasta Pi.

Nótese que solo se añaden aristas en el grafo durante las solicitudes, tanto si se trata de aristas de solicitud como de asignación. En cuanto a la eliminación de aristas, en la liberación se quitan las aristas de asignación, mientras que las de solicitud se retiran en el desbloqueo de un proceso que realizó una petición que ya puede satisfacerse. Además de aristas, en el grafo se añadirán y eliminarán nodos conforme se van creando y destruyendo procesos y recursos.

A continuación, se muestran los grafos de asignación de recursos correspondientes a los dos primeros ejemplos planteados en el tema.

La secuencia de ejecución conflictiva del primer ejemplo era la siguiente:

1. P1: Solicita(C)
2. P2: Solicita(I)
3. P2: Solicita(C) $\rightarrow$ se bloquea puesto que el recurso no está disponible
4. P1: Solicita(I) $\rightarrow$ se bloquea por recurso no disponible: hay interbloqueo

En consecuencia, el grafo de asignación de recursos después de la ejecución de dicha secuencia sería:

- N = {P1, P2, C, I}
- A = {C→P1, I→P2, P2→C, P1→I}

Para poder entender de una forma intuitiva el estado de un sistema, es conveniente establecer una representación gráfica del grafo de asignación de recursos. La convención que se suele utilizar es la siguiente:

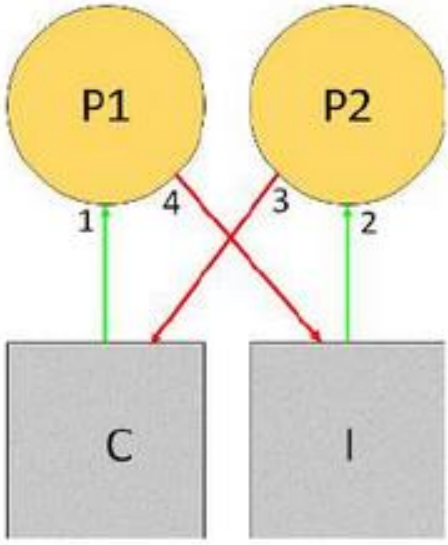
- Cada proceso se representa con un círculo.
- Cada recurso con un cuadrado.

Las aristas de solicitud se representan como arcos que van desde el proceso hasta el cuadrado que representa al recurso.

- Las aristas de asignación se dibujan como arcos que unen el cuadrado que representa el recurso con el proceso correspondiente.

Siguiendo esta convención, en la siguiente figura se muestra la representación gráfica del grafo de ese primer ejemplo:

Figura 50. *Asignación de recursos del primer ejemplo*



Fuente: (Studocu, s.f.)

En cuanto al segundo ejemplo, la traza de ejecución problemática era la siguiente:

1. P1: lock (Ma)

2. P2: lock (Mb)

3. P3: lock (Mc)

4. P3: lock (Ma) → se bloquea puesto que el recurso no está disponible

5. P1: lock (Mb) → se bloquea puesto que el recurso no está disponible

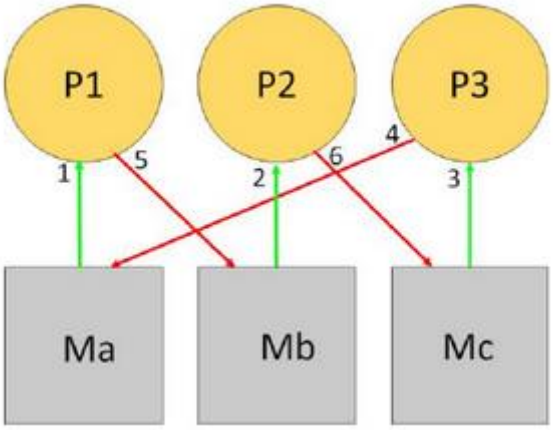
6. P2: lock (Mc) → se bloquea por recurso no disponible: hay interbloqueo

Por tanto, el grafo de asignación de recursos después de la ejecución de esa traza sería:

- N = {P1, P2, P3, Ma, Mb, Mc}
- A = {Ma→P1, Mb→P2, Mc→P3, P3→Ma, P1→Mb, P2→Mc}

Y la representación gráfica del mismo se muestra en la siguiente figura

Figura 51. *Asignación de Recursos del Segundo Ejemplo*



Fuente: (Studocu, s.f.)

Por último, se muestra un tercer ejemplo de un sistema con 3 recursos R1, R2 y R3, donde se ejecutan cuatro procesos que realizan la siguiente traza de peticiones:

1. P1: solicita(R1)

2. P2: solicita(R2)

3. P2: solicita(R1) → se bloquea, puesto que el recurso no está disponible

4. P3: solicita(R2) → se bloquea, puesto que el recurso no está disponible

5. P4: solicita(R3)

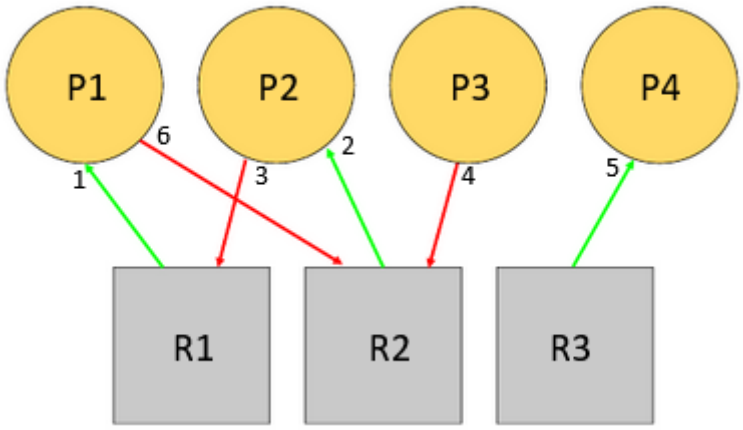
6. P1: solicita(R2) → se bloquea, puesto que el recurso no está disponible

El grafo que representa la situación del sistema después de ejecutarse esa secuencia es:

- N = {P1, P2, P3, P4, R1, R2, R3}
- A = {R1→P1, R2→P2, P2→R1, P3→R2, R3→P4, P1→R2}

Y su representación gráfica se muestra a continuación:

Figura 52. *Asignación de Recursos del Tercer Ejemplo*



Fuente: (Studocu, s.f.)

Comprobación de la existencia de un interbloqueo

Las cuatro condiciones de Coffman son necesarias y suficientes para la existencia de un interbloqueo en el caso de un sistema que tenga las propiedades especificadas para este modelo básico. Por tanto, dado que las tres primeras condiciones tienen que ver con aspectos estructurales del sistema y no con el estado del mismo en un instante particular,

sería suficiente con comprobar que se cumple la condición de espera circular para asegurar que hay un interbloqueo.

En el grafo de asignación de recursos, esa lista de espera circular se corresponde con un ciclo en el mismo. En consecuencia, para comprobar la existencia de interbloqueos en un sistema, basta con detectar si hay ciclos en el grafo que lo representa, estando afectados por el interbloqueo todos los procesos incluidos en esos ciclos.

En los tres ejemplos planteados en el apartado previo, se pueden identificar las siguientes listas de espera circular y, por tanto, los correspondientes ciclos en el grafo:

- En el primer ejemplo, P1 está esperando por un recurso que mantiene P2, que, a su vez, está esperando por un recurso asignado a P1. Esto se visualiza en el grafo como un ciclo formado por las cuatro aristas presentes en el mismo afectando a ambos procesos que, por tanto, están involucrados en el interbloqueo.

- En el segundo ejemplo, P1 está esperando por un recurso que mantiene P2, que, a su vez, está esperando por un recurso asignado a P3, que, por último, está a la espera de un recurso asignado a P1. Esta dependencia se concreta en un ciclo en el grafo que afecta a los tres procesos y está formado por las seis aristas del mismo. En consecuencia, los tres procesos están afectados por un interbloqueo.

- En el tercer ejemplo, existe la misma lista de espera circular que en el primero, que corresponde a un ciclo formado por las aristas 1, 6, 2 y 3. Por consiguiente, solo los procesos P1 y P2 sufren un interbloqueo.

Para detectar los ciclos en un grafo se puede usar cualquier algoritmo de búsqueda de ciclos en grafos dirigidos propuesto dentro de la disciplina de teoría de grafos. La complejidad de los mejores algoritmos de este tipo es proporcional al número de nodos más el número de aristas que tenga el grafo ($O(|N|+|A|)$). Teniendo en cuenta que en este modelo básico de cada proceso y de cada recurso solo puede salir una única arista (de solicitud y de asignación, respectivamente), el máximo número de aristas sería igual al número de nodos y, por tanto, la complejidad proporcional al mismo ($O(2|N|) = O(2n) = O(2(p+r))$).

Como se analizará cuando se estudien más adelante las estrategias de tratamiento del interbloqueo, en algunos casos se estará interesado en detectar qué ciclos hay en todo el grafo de asignación de recursos, requiriendo, por tanto, un algoritmo que analice todo el grafo, mientras que en otros casos solo se necesitará comprobar si un determinado proceso está involucrado en un interbloqueo, para lo que se pueden usar algoritmos que se limiten a comprobar si dicho proceso forma parte de un ciclo.

Podría parecer a priori que la inclusión dentro de un sistema operativo de la gestión de un grafo de asignación de recursos requeriría añadir nuevas estructuras de datos específicas

para tal fin. Sin embargo, normalmente, la información requerida ya la está manejando el sistema operativo para otros propósitos y no será necesario, por tanto, añadirla explícitamente. Así, tomando como referencia el segundo ejemplo, que muestra un escenario de interbloqueo en el uso de mutex, la información requerida, pero ya presente, sería la siguiente:

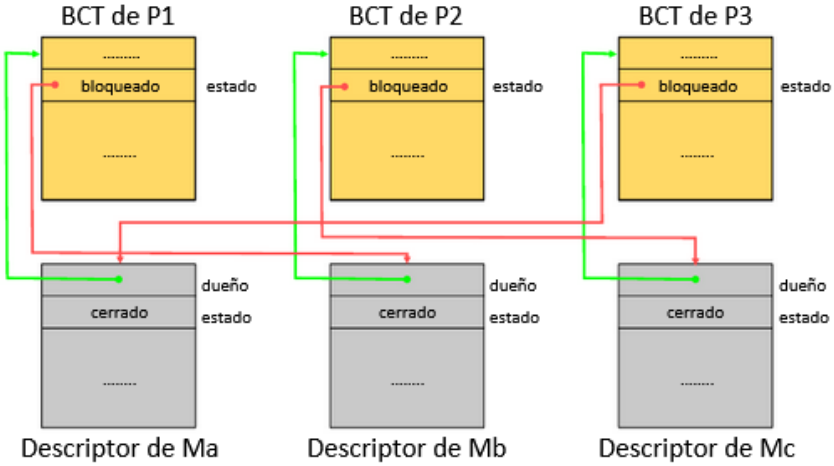
- El sistema operativo debe conocer por qué motivo está bloqueado cada thread que esté en ese estado. Por tanto, en el bloque de control del thread (BCT) guardará información de qué evento está esperando ese proceso para continuar su ejecución. Esa información se correspondería con una arista de solicitud en el grafo de asignación de recursos. En el ejemplo, dicha información permitiría saber por qué mutex está esperando el thread.

- El sistema operativo tiene que saber qué thread tiene asignado un determinado recurso. Para ello, tendrá que guardar esa información en el bloque descriptor del recurso. En este caso, esta información correspondería a una arista de asignación en el grafo de recursos. En el ejemplo, esa información no solo la necesitaría el sistema operativo para poder gestionar interbloqueos, sino que también la requeriría para implementar otros aspectos como, por ejemplo, que los mutex sean de tipo recursivo.

En la siguiente figura, que corresponde a ese segundo ejemplo, se muestra de forma simplificada qué estructuras de datos ya gestiona el sistema operativo para su modo de

operación interno, las cuales proporcionan soporte directo al grafo de asignación de recursos requerido para tratar el interbloqueo.

Figura 53. *Tercer ejercicio implementando en el sistema operativo*



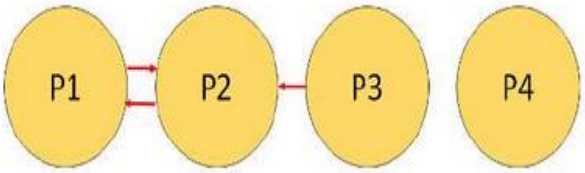
Fuente: (Studocu, s.f.)

Dado que la esencia del problema está en las dependencias entre los procesos, existe una representación simplificada alternativa, denominada grafos de espera (Wait-for Graphs), en la que se prescinde de los recursos y solo se tienen en cuenta los procesos y sus mutuas dependencias. En el caso del tercer ejemplo, el grafo de espera sería:

- N = {P1, P2, P3, P4}
- A = {P1→P2, P2→P1, P3→P2}

La representación gráfica del mismo se muestra en la siguiente figura:

Figura 54. *Espera del tercer ejemplo*



Fuente: (Studocu, s.f.)

Como se puede apreciar en la figura, aparece también un ciclo en esta representación y, por tanto, bastaría con comprobar la existencia de ciclos en este grafo simplificado para determinar la existencia de interbloqueos. En cualquier caso, como se acaba de explicar, las estructuras de datos del sistema operativo, generalmente, dan soporte directo al grafo de asignación de recursos completo, no proporcionando beneficios reseñables el uso de esta representación simplificada.

Modelo extendido del sistema

En esta sección de carácter avanzado se va a estudiar un modelo de sistema más general, que contempla algunas de las características obviadas en el modelo básico, aunque la mayoría de ellas no estén presentes en los sistemas operativos convencionales.

Para entender en qué consiste esta funcionalidad extendida, antes de plantear este nuevo modelo, se va a analizar con más detalle qué características tienen los diversos tipos de recursos que pueden verse involucrados en un interbloqueo, así como las diferentes clases de operaciones que llevan a cabo los procesos sobre esos recursos.

Una vez completado ese análisis, se plantea de manera incremental un modelo extendido que recoge todas estas características avanzadas.

Tipos de recursos

En un sistema informático existe una gran variedad de recursos. Por un lado, recursos físicos, tales como procesadores, memoria, interrupciones o dispositivos. Por otro, recursos lógicos, como ficheros, semáforos, cerrojos de tipo mutex, cerrojos de ficheros, mensajes o señales.

Dada la diversidad de recursos existentes en un sistema y su diferente comportamiento con respecto al interbloqueo, a continuación, se establecerá una clasificación de los distintos recursos, que permitirá incorporarlos al modelo general extendido teniendo en cuenta las particularidades de cada uno. Desde el punto de vista del estudio del interbloqueo, los recursos presentes en un sistema se pueden clasificar siguiendo varios criterios:

- Los procesos pueden compartir el uso de un recurso o lo deben usar en modo exclusivo o dedicado.
- Es factible expropiar el recurso al proceso que lo está utilizando.
- Hay un único ejemplar de cada recurso o existen múltiples unidades de cada uno.

- El recurso sigue existiendo después de que un proceso lo use (recurso reutilizable) o desaparece una vez utilizado (recurso consumible).

Recursos compartidos o exclusivos

Como se ha comentado previamente, para que un recurso pueda verse afectado por un interbloqueo, tiene que ser de carácter exclusivo o dedicado, es decir, que cuando un proceso lo esté usando, ningún otro lo pueda utilizar. Sin embargo, esto no es así para todos los recursos.

Algunos recursos pueden ser usados simultáneamente por varios procesos: son recursos compartidos. Como es evidente, los recursos de tipo compartido no se ven afectados por los interbloqueos, ya que los procesos que quieran usarlos podrán hacerlo inmediatamente sin posibilidad de quedarse bloqueados.

Es interesante resaltar que en un sistema pueden existir recursos que tengan ambos modos de uso (compartido y exclusivo). Cuando un proceso quiere usar un recurso de este tipo, debe especificar en su solicitud si desea utilizarlo en modo exclusivo o compartido. El sistema permitirá que varios procesos utilicen el recurso si lo hacen todos ellos en modo compartido, pero solo permitirá que un único proceso lo use en modo exclusivo. Así, una solicitud de un recurso para su uso en modo compartido se satisfará inmediatamente siempre que el recurso no esté siendo usado en modo exclusivo. En cambio, una solicitud de uso en modo exclusivo solo se concederá si el recurso no está siendo utilizado en ese momento.

Un ejemplo de este tipo de recursos son los cerrojos sobre ficheros, que se estudian en el capítulo dedicado a los sistemas de ficheros. Se pueden especificar dos tipos de cerrojos: de lectura, que correspondería a un uso compartido del recurso, o de escritura, que implicaría un uso exclusivo. Este tipo de recursos también aparece frecuentemente en las bases de datos.

Otros ejemplos serían los cerrojos de tipo lectores-escritores, ya sean bloqueantes (en POSIX, pthread_rwlock_rdlock y pthread_rwlock_wrlock), o con espera activa (dentro del núcleo de Linux, correspondería a las funciones read_lock y write_lock). A continuación, se muestra un ejemplo donde puede producirse un interbloqueo con este tipo de recursos:

Proceso P1	Proceso P2	Proceso P3
Lock RD (Ca)	lock RD (Ca)	lock WR (Cb)
Lock RD (Cb)	lock WR (Cb)	lock WR (Ca)
.....	.....	.....

Si todos los procesos consiguen satisfacer su primera petición, habrá un interbloqueo en el sistema. Como puede apreciarse, se trata del mismo problema planteado previamente, pero con la peculiaridad de que varios procesos pueden compartir un recurso simultáneamente.

Para simplificar, no se considerará en esta exposición el tratamiento de los interbloqueos para este tipo de recursos. En la sección de ejercicios, se pide al lector que plantee cómo se podrían adaptar los algoritmos que se presentan a lo largo del tema para que puedan tratar recursos de este tipo.

Recursos expropiables o no

Una de las estrategias para la prevención de los interbloqueos que se estudiará a lo largo de este capítulo implicará la expropiación de recursos, o sea, la revocación de un recurso asignado a un proceso, mientras este lo está usando, para otorgárselo a otro proceso y, posteriormente, cuando termine de usarlo, reasignárselo de forma transparente al proceso original. Para evitar la pérdida de información, esta expropiación implica salvar de alguna forma el trabajo que llevaba hecho el proceso con el recurso expropiado. La factibilidad de esta expropiación va a depender de las características específicas del recurso, pudiéndose, por tanto, clasificar los recursos de acuerdo a este criterio.

La mayoría de los recursos de uso exclusivo tienen un carácter no expropiable, ya que, o bien no es factible esta operación o, en caso de serlo, sería totalmente ineficiente. Dentro de los recursos físicos, por ejemplo, no tendría sentido quitarle a un proceso un trazador gráfico cuando lo está usando, porque con ello se perdería todo el trabajo realizado.

Con respecto a los recursos lógicos, como se comentó previamente, no se puede expropiar un cerrojo de tipo mutex a un proceso y reasignárselo posteriormente, puesto que se rompería la esencia de este recurso: asegurar la exclusión mutua mientras se ejecuta un conjunto de operaciones.

Otros recursos, sin embargo, pueden expropiarse de manera relativamente eficiente. Considérese, por ejemplo, el caso de un procesador. Cada vez que se produce un cambio de proceso, el sistema operativo está expropiando el procesador a un proceso para asignárselo a otro. La expropiación de un procesador implicaría únicamente salvar el estado del mismo en el bloque de control del proceso correspondiente, para que así este pueda seguir ejecutando normalmente cuando se le vuelva a asignar. Obsérvese que, conceptualmente, el procesador, como cualquier otro recurso de uso exclusivo, puede verse implicado en interbloqueos.

Suponga, por ejemplo, una situación en la que existe un proceso que tiene asignada una cinta y está en estado listo para ejecutar (o sea, no tiene asignado el procesador). Si el proceso que está en ejecución (es decir, que tiene asignado el procesador) solicita la cinta, se producirá un interbloqueo, ya que cada proceso necesita un recurso que posee el otro. Este interbloqueo potencial no ocurre en la práctica, puesto que, en un sistema multiprogramado, cuando el proceso en ejecución se bloquea, se reasigna automáticamente el procesador a otro proceso gracias al carácter expropiable de este recurso.

Es interesante remarcar que esta expropiación del procesador se puede llevar a cabo porque cada proceso ejecuta en un contexto independiente. Sin embargo, como se analizó previamente en el escenario de interbloqueos asociados al tratamiento de eventos asíncronos, esta expropiación no será factible cuando un flujo de ejecución se vea interrumpido asíncronamente por el tratamiento de algún tipo de evento, como, por ejemplo, una señal o, en el caso del modo de operación interno del sistema operativo, la interrupción de un dispositivo.

En los sistemas que utilizan un dispositivo de almacenamiento secundario (generalmente un disco) como respaldo de la memoria principal (como son los sistemas con memoria virtual, que se estudian en el capítulo dedicado a la gestión de memoria), el recurso memoria también es expropiable. Cuando se requiera expropiar a un proceso parte de la memoria que está usando, se copiará el contenido de la misma en el dispositivo de respaldo, dejándola libre para que pueda usarla otro proceso. Nótese que dicha operación de copia tiene un coste asociado que afectará al rendimiento del sistema.

Recursos con una única unidad o con múltiples

En el modelo básico, se ha considerado que cada recurso es una entidad única. Sin embargo, en un sistema pueden existir múltiples ejemplares o unidades de un determinado recurso. Una solicitud de ese recurso por parte de un proceso podría satisfacerse con cualquier ejemplar del mismo. Así, por ejemplo, en un sistema en el que haya cinco impresoras, cuando un proceso solicita una impresora, se le podría asignar cualquier unidad que esté disponible. La existencia de múltiples unidades de un mismo recurso también permite generalizar los servicios de solicitud, de forma que un proceso pueda pedir simultáneamente varios ejemplares de un recurso.

Evidentemente, el número de unidades solicitadas nunca debería ser mayor que el número de unidades existentes. Nótese que, a veces, podría ser discutible si dos elementos constituyen dos ejemplares de un recurso o se trata de dos recursos diferentes. Incluso distintos usuarios pueden querer tener una visión u otra de los mismos. Considérese, por ejemplo, un equipo que tiene conectadas dos impresoras láser con la misma calidad de impresión, pero tal que una de ellas es algo más rápida.

Un usuario puede querer usar indistintamente cualquiera de estas impresoras, con lo que preferiría considerarlas como dos ejemplares del mismo recurso. Sin embargo, otro usuario, que necesite con urgencia imprimir un documento, requeriría verlas como dos recursos separados, para poder especificar la impresora más rápida.

Lo razonable en este tipo de situaciones sería proporcionar a los usuarios ambas vistas de los recursos. Sin embargo, las soluciones clásicas del interbloqueo no contemplan esta posibilidad de un doble perfil: o son recursos independientes o son ejemplares del mismo recurso.

Los sistemas operativos convencionales, generalmente, consideran cada recurso de forma individual, no contemplando el concepto de ejemplares o instancias de un recurso. En cualquier caso, aunque no sea de forma evidente, se pueden buscar ejemplos en estos sistemas que, hasta cierto punto, encajen en un modelo de recursos con múltiples ejemplares de cada recurso. Considérese el caso de la memoria. Se trata de un único recurso con múltiples unidades: cada palabra que forma parte de la misma. Cuando un proceso solicita una reserva de memoria de un determinado tamaño, está solicitando el número de unidades de ese recurso que se corresponde con dicho tamaño.

Nótese que en este caso existe una restricción adicional, ya que las unidades asignadas, sean cuales sean, se deben corresponder con posiciones de memoria contiguas (no ocurriría lo mismo en caso de que se solicitase un conjunto de marcos de página cualesquiera en un sistema basado en paginación).

Otro ejemplo de una situación donde se manejan recursos con múltiples unidades podría ser la asignación de procesadores a aplicaciones paralelas en un multiprocesador, en la que se requiere un número de procesadores determinado para cada aplicación, pero da igual, en principio, cuáles se le asignen.

Recursos reutilizables o consumibles

Como se comentó previamente, un recurso reutilizable se caracteriza porque el recurso sigue existiendo después de que un proceso lo use, quedando disponible para otros procesos. Por tanto, la “vida” del recurso es independiente de su utilización: o bien existe desde el principio, en el caso de que se trate de un recurso físico (nótese que en un sistema con hot plugging se pueden conectar en cualquier momento ciertos recursos físicos, como,

por ejemplo, un dispositivo USB), o bien, una vez creado, sigue existiendo hasta que se destruya explícitamente, como sucede con los recursos lógicos. Dentro de esta categoría se engloban la mayoría de los recursos físicos y algunos recursos lógicos, tales como los cerrojos de ficheros o de tipo mutex.

Todos los ejemplos propuestos hasta este momento, así como el modelo básico planteado, se centran en este tipo de recursos. El primer ejemplo mostrado en este tema usaba dos procesos y dos recursos reutilizables físicos (la cinta y la impresora), mientras que el segundo utilizaba tres threads y tres recursos reutilizables lógicos: los cerrojos de tipo mutex.

La sección se centra, por tanto, en los recursos consumibles, analizando en qué se distinguen de los reutilizables. Un recurso consumible se caracteriza porque deja de existir una vez que un proceso lo usa. Un proceso genera o produce el recurso y el otro lo utiliza consumiéndolo. En esta categoría se encuentran recursos físicos como las interrupciones, así como los recursos lógicos relacionados con la comunicación y algunos vinculados con la sincronización de procesos.

Algunos ejemplos de recursos lógicos consumibles serían los mensajes, las señales o los semáforos. En general, cualquier situación donde un proceso esté esperando que se produzca un determinado evento puede considerarse como un escenario de uso de recursos consumibles. Por ejemplo, un flujo de ejecución, ya sea un proceso o un thread, que está esperando la finalización de otro flujo, corresponde también a una situación de uso de recursos consumibles y, por tanto, puede implicar interbloqueos.

En la clasificación planteada, se ha considerado que, a pesar de sus similitudes, los semáforos son recursos consumibles, mientras que los cerrojos de tipo mutex son reutilizables.

GLOSARIO DE TÉRMINOS

- **abstracción de proceso:** método utilizado por un Sistema Operativo para representar un proceso tanto de forma interna en memoria, como externamente para mostrar sus características al usuario.
- **acumulador:** registro interno de la CPU que recoge los resultados intermedios de las operaciones que se realizan en la ALU.
- **apuntador o puntero:** tipo de dato formado por una dirección de memoria principal. A través de este tipo de dato se puede acceder a cualquier bloque de memoria que esté referenciado, es decir que sepamos su dirección lógica.
- **argumentos:** valores de entrada con los cuales trabaja una rutina o programa. Cuando un argumento se usa para ajustar un programa a los requerimientos de un cliente se llama parámetro.
- **asíncrono:** término utilizado para especificar la ejecución de distintos procesos de forma independiente unos de los otros respecto al tiempo. En Hardware este concepto es aplicable a dispositivos físicos como la memoria y la CPU, etc.
- **batch:** término inglés que en informática se traduce por los términos 'por lotes', 'conjunto' o 'grupo'. Característica típica de ciertos procesos, que indica una serie de tareas que se ejecutan de forma sucesiva en el ordenador y se consideran como una unidad.
- **bit de presencia:** bit que sirve para identificar si se encuentra en un momento dado un bloque, página o área de información en la memoria principal.
- **bits de mayor peso:** bits más a la izquierda de un número bajo una representación en código binario. Los de menor peso son los bits que se sitúan más a la derecha.
- **bus de direcciones:** canal físico consistente en una serie de líneas de circuito impreso, que permite la transmisión de direcciones entre la Memoria principal y la CPU.
- **capas o niveles de software:** son todos aquellos niveles software que se sitúan entre el usuario y los dispositivos físicos de la máquina, de forma que el nivel de complejidad para la utilización de las capas o niveles aumenta conforme nos acercamos al hardware.

- **carga del sistema:** número de procesos (programas activos), que tenemos actualmente cargados en memoria principal. Sobre cualquiera de ellos puede actuar el planificador.
- **cargador:** programa encargado de cargar las distintas líneas de código de un programa ejecutable en la memoria principal del ordenador para su posterior ejecución.
- **cita (rendezvous):** del inglés 'rendezvous', en informática conocido por 'cita'. Tipo de sincronización doble, en el que los dos procesos se sincronizan en un determinado punto de cada uno de los procesos afectados. Uno espera al otro para poder avanzar.
- **cliente:** aquel proceso o dispositivo solicitante que realiza la petición de un servicio o trabajo a otro proceso denominado 'servidor'.
- **código portable:** aquel que puede moverse fácilmente de un tipo de máquina a otra, aunque cambien las características internas de la misma.
- **cola:** estructura de datos dinámica residente en memoria principal derivada de las listas, la cual puede cambiar de tamaño en tiempo de ejecución. Las inserciones y eliminaciones se van a realizar atendiendo al método FIFO (Primero en entrar, Primero en salir).
- **consola:** dispositivo de entrada de datos a un ordenador, conocido más genéricamente como teclado o terminal, ya sea manual o por conectores.
- **contexto o entorno:** término que engloba a las principales características de un proceso que nos proporcionan información sobre el entorno del mismo: Contador de Programa, registros del procesador, información de la pila y sus principales atributos.
- **controladores de E/S:** dispositivos físicos encargados de gestionar la transferencia de datos entre la CPU, a través del bus de datos del ordenador y un periférico conectado al puerto de E/S.
- **CPU:** Acrónimo inglés de "Central Processing Unit", en castellano, "Unidad Central de Proceso". Aquella unidad incluida íntegramente en el microprocesador (solo en PC's) de un computador que controla el resto de unidades. Formada por la unidad aritmético-lógica, la unidad de control y de pequeños registros principalmente. El control lo lleva a cabo

mediante la interpretación y la ejecución de instrucciones, según las señales que le dicta un reloj.

- **cuanto:** unidad elemental de tiempo utilizado por el Sistema Operativo para la planificación de procesos. Puede variar para distintos Sistemas Operativos y normalmente consta de una serie de ciclos de reloj (ver en tema de planificación).
- **depurador:** programa especializado en la corrección y eliminación de errores en un código fuente durante su edición, o capturando diversos datos durante su ejecución.
- **descriptor de fichero:** número que representa a un fichero lógico con el que se trabaja en lenguaje C. Se obtiene a partir de la asociación que se realiza en una operación de apertura del fichero con su correspondiente nombre físico.
- **direccionar:** hacer referencia a una dirección dentro de un mapa de memoria, ya sea real o virtual.
- **DMA:** acrónimo inglés de 'Direct Memory Access', en castellano, Acceso Directo a Memoria. Consiste en una técnica, implementada bajo un microprocesador especializado, orientada a la transferencia de información desde un dispositivo periférico hasta la memoria principal, sin la utilización para controlarla de la CPU. La CPU solo actúa al principio y fin de dicha transferencia.
- **editor de texto:** programa orientado a la edición, y manipulación de archivos de texto. No presenta características de procesamiento complejas.
- **enlazador:** programa encargado de unir o enlazar el código de varios módulos separados que mantienen llamadas o referencias cruzadas entre los mismos, de forma que se pueda realizar su posterior ejecución.
- **ensamblador:** lenguaje de programación de medio nivel, el cual es traducible directamente a lenguaje máquina. Está constituido por un juego de instrucciones característico de cada máquina con distinta arquitectura.
- **espera bloqueada:** tipo de 'espera' en primitivas para solucionar el problema de la exclusión mutua y sincronización mediante el mantenimiento de colas de procesos

bloqueados hasta que se dé una cierta condición y pasen al estado de listos. Se da en mecanismos software como los semáforos y monitores.

- **estrategias de planificación:** cada una de las distintas determinaciones que se pueden tomar para realizar un tipo u otro de planificación, basándose en una serie de criterios.
- **evento o suceso:** un evento o suceso va a consistir en cualquier actuación, ya sea externa o interna que pueda provocar un cambio de estado, a nivel individual en un dispositivo o de forma general en un sistema.
- **gasto extra:** gasto de tiempo o cualquier recurso que se deriva de mantener mayor dedicación a tareas de planificación y gestión que a la propia dedicación de tiempo de CPU a los procesos de usuario.
- **gestor de interrupciones:** programa encargado de controlar mediante una trampa y redirigir el flujo del programa hacia las rutinas pertinentes que solucionen el problema que provocó la excepción o interrupción.
- **grado de protección:** posibilita establecer un nivel de protección para cada una de las posibles particiones de memoria principal, de forma que el acceso a las mismas esté limitado según unos niveles de protección.
- **grafos dirigidos:** estructuras de datos representadas mediante una serie de nodos y relaciones entre los mismos. La red de conexión que se establece entre los nodos no tiene restricciones, y se pueden realizar de muchos a muchos.
- **hardware:** conjunto de dispositivos físicos, metálicos y de materiales plásticos que forman el ordenador u otro dispositivo conectado al mismo.
- **interactividad:** propiedad que es característica de entes comunicativos en los cuales priman los tiempos cortos de respuesta. Es característico de ciertos procesos y dispositivos.
- **interfaz o interface:** de un objeto es la parte de un objeto accesible desde su exterior, que nos permite utilizarlo y consultar su estado interno. La vamos a utilizar como medio de comunicación entre el objeto y cualquier ente que quiera comunicarse con él.

- **interrupción:** detención temporal en un punto determinado de un programa, para permitir el acceso y proceso de una rutina que tomará el control del sistema. Puede ocurrir debido a diferentes causas ya sean hardware o software.
- **lenguaje máquina:** (instrucciones) son aquellas que basadas en secuencias de ceros y unos (código binario), son entendibles directamente por el ordenador y por lo tanto ejecutables sin necesidad de traducción.
- **lenguajes de alto nivel:** lenguajes simbólicos constituidos por un léxico, una sintaxis y una semántica; los cuales son fácilmente entendibles por el usuario cuando tiene que realizar programas para computadoras.
- **lista circular:** estructura de datos dinámica residente en memoria principal, derivada de las listas, la cual puede cambiar de tamaño en tiempo de ejecución. Su acceso es secuencial y tiene como característica que el último elemento de la misma mantiene un enlace al primero, uniendo a ambos.
- **lista doblemente enlazada:** estructura de datos dinámica residente en memoria principal, derivada de las listas, la cual puede cambiar de tamaño en tiempo de ejecución. Su acceso es secuencial pero además de estar enlazados cada elemento hacia el siguiente lo está hacia el anterior.
- **lista enlazada:** estructura de datos dinámica residente en memoria principal, la cual puede cambiar de tamaño en tiempo de ejecución. Su acceso es secuencial y está formada por un grupo de elementos en el que cada uno de ellos apunta al próximo.
- **llamada por valor:** llamada a un procedimiento de forma que, con la ayuda de parámetros el objeto que se envía al mismo es una copia del original que guarda el procedimiento llamador. De esta forma cualquier cambio sobre el mismo afectará solo a la copia.
- **llamadas por referencia:** llamada a un procedimiento de forma que, mediante parámetros el objeto que se envía al mismo es el espacio de direcciones del objeto original, el cual guarda el procedimiento llamador. De esta forma cualquier cambio sobre el mismo será más difícil de implementar debido al cruce de direcciones.

- **máquina virtual:** es aquella interface que mantiene una máquina mediante la cual nos comunicamos con los dispositivos hardware del ordenador. De esta forma nosotros trabajamos a un nivel superior eliminando la complejidad de dichos dispositivos. De esta forma podemos mantener distintos Sistemas Operativos corriendo sobre una misma máquina.
- **marcos o páginas físicas:** corresponde a cada una de las zonas en las que se divide la memoria principal, para abarcar una página cargada desde memoria secundaria en un proceso de intercambio o carga de páginas.
- **memoria de semiconductores:** memoria basada en principios electrónicos basados en la semiconductividad de algunos materiales. Prestan mejor rendimiento que las antiguas válvulas de vacío en cuanto a tamaño, velocidad y precio.
- **memoria dinámica:** zona de la memoria principal que reservan los compiladores para poder utilizarla mediante una serie de peticiones al gestor de memoria en tiempo de ejecución.
- **memoria principal:** dispositivo que almacena información en forma de datos codificados en binario. Es accedida directamente por la CPU y además es de lectura y escritura, pero volátil. Está formada por una serie de posiciones o celdillas de memoria, las cuales son referenciadas mediante un sistema de direcciones lógicas.
- **memoria RAM:** acrónimo ingles de Random Access Memory, en castellano, Memoria de Acceso Aleatorio. Tipo de memoria de la que está compuesta la memoria principal, de forma que en la misma vamos a tener almacenado el sistema operativo en cada momento, y aquel proceso o procesos que hayan sido cargados en la misma. Es condición necesaria que todos los datos y programas estén almacenados dentro de la misma antes de poder ser ejecutados.
- **memoria ROM:** acrónimo ingles de Read Only Memory, en castellano, Memoria de Solo Lectura. Memoria más pequeña y de solo lectura donde están grabados los gestores software de dispositivos y configuración interna del hardware del sistema.

- **memoria secundaria:** tipo de memoria auxiliar que retiene toda la información almacenada cuando se desconecta de la fuente de alimentación. Es de acceso lento, pero de gran capacidad.
- **miniordenadores:** ordenadores de gran potencia, que fueron diseñados para gestionar empresas de tamaño medio con capacidades de información media y sistemas multiusuario.
- **modo supervisor:** modo de trabajo del sistema operativo mediante el cual solo el es capaz de controlar y acceder a los recursos del ordenador mediante instrucciones privilegiadas que no están disponibles para los programas de aplicación o de usuario.
- **modo usuario:** modo de trabajo del sistema operativo mediante el cual es el usuario el que tiene el acceso y control de todo el sistema. Se denota cuando el intérprete de órdenes está activo mediante el prompt o inductor.
- **monopolización de la CPU:** un determinado tipo de procesos adquieren los derechos de tiempo de ejecución de forma exclusiva. El resto de procesos no pueden acceder a este recurso imprescindible para avanzar en la ejecución.
- **multiprocesador:** sistema informático basado en la ejecución de programas mediante la utilización de varios procesadores trabajando de forma simultánea. Pueden disponerse físicamente de varias formas: en serie, paralelo, etc.
- **multiprogramación:** sistema de programación que permite la ejecución simultanea de varios procesos residentes en la memoria principal. Es decir, todos los procesos avanzan en su ejecución (ver figura).
- **multiusuario:** sistema informático capaz de soportar el trabajo de varios usuarios en una misma máquina o grupo reducido de ellas. (ver apéndice multiusuario)
- **naturaleza de un recurso:** características principales de un recurso que lo definen y lo determinan. La más importante es la compartibilidad.

- **nombre simbólico:** nombre de variable, constante o tipo de dato, mediante el cual accedemos lógicamente a los datos en memoria principal sin tener que referenciar sus posiciones físicas. Utilizado en lenguajes de programación.
- **octal o hexadecimal:** sistemas de numeración utilizados en la representación de datos numéricos y alfanuméricos en base "8" y en base "16" respectivamente.
- **palabra de estado (PSW):** del inglés, Processor Status Word, en castellano, Palabra de estado del microprocesador. Palabra que recoge en binario el estado del entorno de programa, después de la ejecución de cada instrucción.
- **partición:** cada una de las zonas en las que se divide la memoria principal para alojar los procesos del sistema.
- **PCB:** acrónimo inglés de 'Process Central Block', en castellano, Bloque Central de Proceso, el cual almacena información específica sobre un proceso en concreto.
- **Periféricos:** serán aquellos dispositivos que no forman parte del computador central estando este formado por la memoria principal y la CPU. Sirven para comunicarse con el mismo.
- **pila:** zona reservada de la memoria o registros hardware donde se almacena temporalmente el estado o información de un programa, rutina, etc. Mantiene una política de inserciones y eliminaciones LIFO (Primero en entrar Último en salir).
- **planificable:** todo dispositivo o entidad que es susceptible de ser planificada, en cuanto a tiempo, medios o cualquier otro recurso utilizable por la misma.
- **previsibilidad:** característica cualitativa a priori, respecto de la actuación o no de un proceso en un determinado momento. Se dice que el comportamiento de un proceso no es previsible en cada ejecución.
- **prioridad:** orden de importancia relativa de una serie de operaciones planificables. Una operación con una prioridad alta se ejecuta antes que una de baja prioridad, siendo el sentido de mayor a menor prioridad o viceversa.

- **procedimiento:** rutina o conjunto de instrucciones ejecutable, totalmente independiente, la cual realiza un proceso determinado a partir de una serie de parámetros de entrada, ofreciendo los resultados mediante parámetros de salida.
- **Prompt (indicador):** indicador de modo usuario, que permite a este realizar peticiones de servicios al intérprete de órdenes. Este se sustituye actualmente por sistemas de ventanas.
- **protocolos:** estándar de comunicación entre distintos dispositivos físicos o procesos mediante el cual se regulan los mecanismos y políticas a seguir para llevar a cabo una transmisión perfecta de datos.
- **puerto:** dispositivo físico que conecta los canales internos de información del ordenador con las líneas de comunicación de los posibles periféricos conectables.
- **puntos de verificación/reinicio:** punto donde se realiza una grabación intermedia del estado del sistema en un punto concreto donde se da cualquier tipo de interbloqueo. De esta forma se puede reanudar fácilmente el trabajo o trabajos previamente bloqueados.
- **red de máquinas heterogéneas:** conjunto de ordenadores de distinta naturaleza o composición interna debido a sus distintas arquitecturas.
- **red de ordenadores:** conjunto de ordenadores conectados físicamente entre sí mediante los cuales realizo operaciones tales como conexiones, transferencias y tratamiento de información, de forma que cada uno de ellos es consciente del lugar físico que ocupan los demás.
- **registro:** pequeña memoria interna del microprocesador, formada generalmente por biestables. Es de rápido acceso y son de muy utilizados programando en lenguaje ensamblador.
- **reloj de interrupciones:** reloj que genera interrupciones para marcar los instantes en los que se interrumpe el proceso que se está ejecutando en la CPU, para dar paso a otro elegido entre los procesos de la cola de procesos listos, mediante un algoritmo de planificación.

- **rutinas de E/S:** conjunto de instrucciones encargado de controlar la transferencia de datos entre los periféricos y el ordenador. Gestionan las capacidades de los puertos.
- **secuenciamiento:** método de ejecutar las ordenes de un programa o una serie de procesos de forma secuencial, es decir, unos después de otros.
- **semáforo:** Un semáforo es un mecanismo software mediante el cual consigo la sincronización entre procesos concurrentes. Su implementación estará basada en dos elementos: un entero y un puntero asociado a una cola (que puede ser nulo). Está controlado mediante una serie de señales conocidas como WAIT y SIGNAL.
- **serie:** método de procesamiento o transmisión de datos basado en etapas sucesivas, no simultáneas. Ejemplo de transmisión bita a bit.
- **servicio:** trabajo o tarea ofrecida por el sistema operativo ante una llamada al sistema por parte del usuario.
- **servidor:** proceso o dispositivo encargado de resolver o solucionar la petición de servicio o trabajo realizado por un proceso denominado 'cliente'. Además, debe dar una respuesta con el resultado de la consulta.
- **sheduling**: del inglés 'scheduler', en castellano, planificación, mediante la cual se decide cuál de las tareas activas en el sistema seguirá ejecutándose, tomando esta la posesión del tiempo de CPU.
- **Shells (interprete de órdenes):** término inglés traducido por intérprete de órdenes. Este es capaz de recoger las órdenes que el usuario realiza, pasándolas al núcleo del Sistema Operativo para su ejecución.
- **SIGNAL:** operación de continuación sobre un semáforo. Los semáforos serán rutinas, programadas para utilizarlas en programación concurrente.
- **síncrono:** término asociable a distintos dispositivos y procesos. En sistemas multiprocesador representa la sincronización o accionamiento simultaneo de varios componentes como la memoria principal y la CPU.

- **sistema distribuido:** conjunto de ordenadores conectados entre sí de forma que la ejecución de aplicaciones en uno de ellos no depende de donde esté situada la misma. Además, permite montar arquitecturas cliente/servidor.
- **sistemas de bases de datos:** sistema para manejar la información almacenada de forma masiva en una Base de Datos y gestionar grandes cantidades de información de una forma estructura, haciendo posible unos accesos a los datos, ágiles y rápidos.
- **software de sistemas:** aquel que está orientado a la manipulación y control directo del hardware mediante una serie de rutinas especializadas.
- **software:** son los programas, incluyendo procedimientos, utilidades, sistemas operativos, programas de aplicación y paquetes informáticos, implementados para un sistema informático.
- **tabla de procesos:** estructura de datos que, actuando como una tabla de una entrada, almacena los procesos creados en el sistema ya estén listos o bloqueados.
- **tamaño de cuánto:** tamaño en unidades de tiempo, del orden de microsegundos, que dura la unidad mínima de ejecución, el cuánto. De este parámetro depende en gran parte la eficiencia del planificador asignando la CPU a procesos.
- **tarjetas:** dispositivos estandarizados de material plástico con perforaciones que permiten el almacenamiento de programas de forma permanente.
- **terminal:** dispositivo de E/S de datos de un ordenador, el cual no tiene por qué estar localizado junto a la carcasa de la CPU, sino que puede estar en una localización remota (ver consola).
- **tiempo compartido:** sistema de reparto de la capacidad de proceso basado en la división del tiempo de CPU entre los distintos trabajos que hay en memoria principal, para que todos avancen en su ejecución. (ver apéndice)
- **tiempo de búsqueda:** tiempo que transcurre desde que se recibe la orden de transferencia hasta que la cabeza lectora se posiciona sobre el disco donde buscamos los datos.

- **tiempo de latencia:** tiempo que transcurre desde que se da la orden de posicionamiento sobre la cabeza lectora al cabezal que soporta el disco, hasta que está colocado en su lugar correcto donde están los datos se quiere transmitir.
- **tiempo de respuesta:** tiempo que se tarda desde que un proceso está listo para ejecutarse hasta que el recurso de la CPU es adquirido por el mismo. De forma general: tiempo total que se tarda en atender la CPU un proceso interactivo ante una petición de servicio.
- **tiempo de transferencia:** tiempo que se tarda en transmitir una serie de datos desde el disco a memoria principal, desde que los datos son encontrados por la cabeza lectora hasta que finaliza la transmisión.
- **tiempo real:** un sistema de este tipo es aquel que necesita de tiempos de respuesta muy cortos, incluso del orden de microsegundos, en el caso de procesos críticos.
- **traductor:** programa que convierte mediante una traducción un lenguaje fuente en un lenguaje objeto, sin que por ello varíe la semántica del código traducido, ya que tan solo cambia su representación.
- **trampa de fallo de página:** interrupción provocada por un acceso a una página en memoria principal, la cual no estaba en la misma debido a que una administración virtual de la memoria provocó su traslado a disco, mediante un intercambio de páginas.
- **trampa:** método consistente en atrapar u o capturar una interrupción mediante la comprobación de una condición en particular en un programa en ejecución, para procurar después la ejecución de la rutina correspondiente que resuelva dicha interrupción (ver gestor de interrupciones).
- **tubo o tubería:** mecanismo software para solucionar problemas de comunicación y sincronización entre distintos procesos, mediante la implementación de un canal software de comunicación asíncrono.
- **WAIT:** operación de espera sobre un semáforo. Los semáforos serán rutinas programadas utilizadas en programación concurrente.

- **hilo:** cada una de las unidades de asignación de un proceso, de esta forma cada proceso (elemento que posee recursos), puede mantener varias unidades de asignación de CPU en una misma ejecución. Este concepto se conoce también como 'thread', hebra o proceso ligero.
- **recurso:** objetos o dispositivos que son utilizados por una computadora, para poder realizar todos los trabajos y tareas que se requieren a partir de las peticiones que van realizando los procesos del sistema. Entre otros en una computadora tenemos como recursos los siguientes: memoria principal, dispositivos de memoria secundaria, y cualquier dispositivo periférico direccionable. Además, estos pueden ser o no compartibles.

REFERENCIAS BIBLIOGRÁFICAS

unefa.blogspot. (s.f.). Obtenido de http://sistemasoperativos03-unefa.blogspot.com/2012/01/gestion-de-entradas-y-salidas.html

ALEGSA, s/f. Hardware de una computadora. (En línea). Consultado, 7 de noviembre de 2015. Disponible en: http://www.alegsa.com.ar/Dic/cluster.php

Andrew S, T. (s.f.). *Sistemas operativos. 3 edición.*

Bayoc, C. 2011. Cintas magnéticas. (En línea). Consultado, 7 de noviembre de 2015. Disponible en: http://es.slideshare.net/Bayod/cintas-magnticas

Educatica. (s.f.). Obtenido de https://www.educatica.es/sistemas-operativos/principios-basicos/sistemas-de-ficheros/#Ficheros_y_directorios

Fing. (s.f.). Obtenido de https://www.fing.edu.uy/tecnoinf/maldonado/cursos/so/material/teo/so05-hilos.pdf

Graphical User Interface. (s.f.).

KIOSKEA, s/f. Sistema Operativo. (En línea). Consultado, 9 de noviembre de 2015. Disponible en: http://computadoras.about.com/od/conoce-procesadores/a/Que-Es-Un-Procesador.htm

Miscelaneos. (s.f.). Obtenido de http://sopa.dis.ulpgc.es/ii-dso/leclinux/miscelaneos/llamadas/LEC_llamadas.pdf

Paoguaman. (s.f.). Obtenido de revision-del-hardware-de-computadora-y.html: http://paoguaman.blogspot.com/2015/11/revision-del-hardware-de-computadora-y.html

Profesores. (s.f.). Obtenido de https://www.profesores.frc.utn.edu.ar/sistemas/ingsanchez/SOP/Archivos/SOPII.pdf

Scribd. (s.f.). Obtenido de https://es.scribd.com/document/249893614/Unidad-3-Administracion-de-Memoria

Sistop. (s.f.). Obtenido de http://gwolf.sistop.org/laminas/05-procesos-e-hilos.pdf

Studocu. (s.f.). Obtenido de https://www.studocu.com/ec

TemasSOuJaen. (s.f.). Obtenido de https://lsi.vc.ehu.eus/pablogn/docencia/manuales/SO/TemasSOuJaen/CONCURRENCIA/4ProblemasClasicosdeComunicacionentreProcesos.htm

Webcindario. (s.f.). Obtenido de https://sistop.webcindario.com/pagina.html

Wordpress. (s.f.). Obtenido de https://lcsistemasoperativos.wordpress.com/tag/monoliticos/

INSTITUTO SUPERIOR TECNOLÓGICO UNIVERSITARIO CORPORATIVO
EDWARDS DEMING
Saber qué hacer y cómo hacerlo

Deming
EDITORIAL TECNOLÓGICA

EDICIÓN PRIMERA
2025

ISBN: 978-9942-692-54-2