Resumen de Sistemas Operativos

Sebastián Ogás

6 de noviembre de 2022

Índice

Capítulo 1

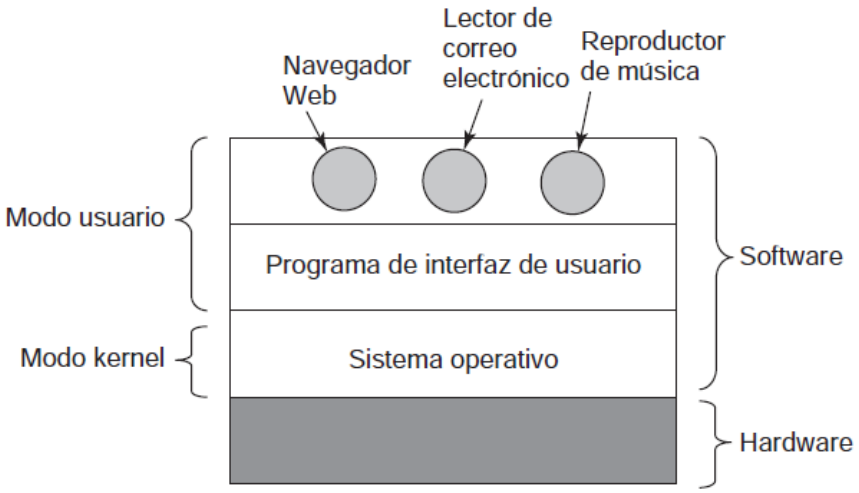
Introducción

1.1 Programas de usuario (Tanenbaum, 2009, p. 1)

- shell: línea de comandos
- GUI: Graphical User Interface, Interfaz Gráfica de Usuario

1.2 Modos (Tanenbaum, 2009, p. 1-2)

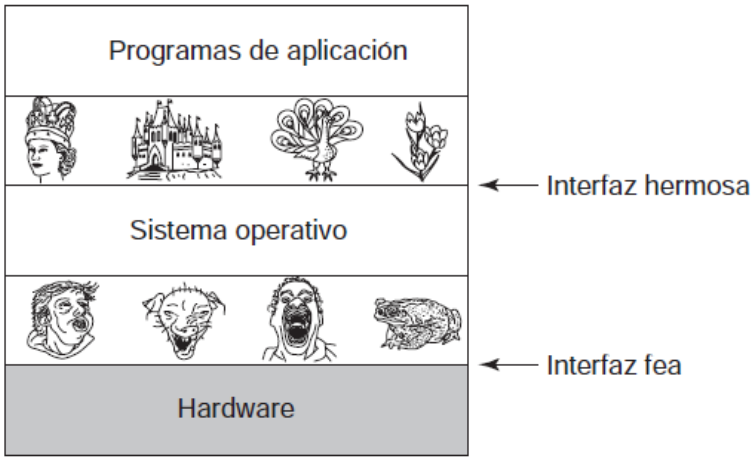
- Kernel/núcleo/supervisor: sin restricciones, tiene acceso a todo lo que el nivel ISA (Instruction Set Architecture, Set de Instrucciones de la Arquitectura) permita. El sistema operativo se ejecuta en este modo.
- Usuario: no puede ejecutar cualquier instrucción por motivos de seguridad, en particular aquellas relacionadas al control de la máquina o las E/S (entradas y salidas).



1.3 Funciones de un sistema operativo (Tanenbaum, 2009, p. 4-7) (Stallings, 2005, p. 54-57)

1.3.1. Máquina extendida

El sistema operativo toma la interfaz compleja y técnica provista por una arquitectura y la **abstrae**, ofreciendo una interfaz más sencilla a los programas y sus desarrolladores.



Se suele proporcionar servicios para:

- Desarrollo de programas
- Ejecución de programas
- Acceso a dispositivos de E/S
- Acceso controlado a los archivos
- Acceso al sistema
- Detección y respuesta a errores
- Contabilidad (estadísticas de uso y parámetros de rendimiento)

1.3.2. Administrador de recursos

El sistema operativo decide cómo distribuir (multiplexar) los recursos (CPU, Memoria, E/S y buses) cuando varias rutinas los requieren, en el espacio (dividiéndolos) y en el tiempo (turnándose).

1.4 Partes principales de una computadora (Tanenbaum, 2009, p. 19-32) (Stallings, 2005, p. 10-37)

1.4.1. Procesadores (CPU)

Ciclo básico: obtener una instrucción de memoria (ciclo de búsqueda), decodificarla y ejecutarla (ciclo de ejecución). La acción que realiza una instrucción suele entrar en una de las siguientes categorías:

- Procesador-memoria
- Procesador-E/S
- Procesamiento de datos
- Control

Métodos para mejorar el rendimiento:

- Canalización (pipeline): hay una unidad dedicada a cada proceso del ciclo básico, permitiendo la obtención, decodificación y ejecución simultánea de instrucciones consecutivas.
- CPU superescalar: hay múltiples unidades dedicadas a cada proceso del ciclo básico, siendo cada unidad de ejecución específica a un tipo de instrucción. Luego de ser decodificadas, las instrucciones son colocadas en un búfer de contención, del cuál las unidades de ejecución las obtienen.

Registros:

- Generales: para variables y resultados temporales
- PC: Contador de programa
- IR: Registro de instrucciones
- SP: Apuntador de Pila (Stack Pointer, apunta a su cima)
- PSW: Palabra de Estado del Programa (Program Status Word, almacena bits de control como resultados de comparación, prioridad de la CPU o modo kernel o usuario)
- RDIM, RDAM: Registros de Direcciones/Datos de Memoria.
- RDIE/S, RDAE/S: Registros de Direcciones/Datos de E/S.
- Registros índice
- Registros puntero de segmento

Llamadas al sistema: llamada a procedimiento en modo kernel por un programa en modo usuario. Se realizan mediante la instrucción `TRAP / INT`.

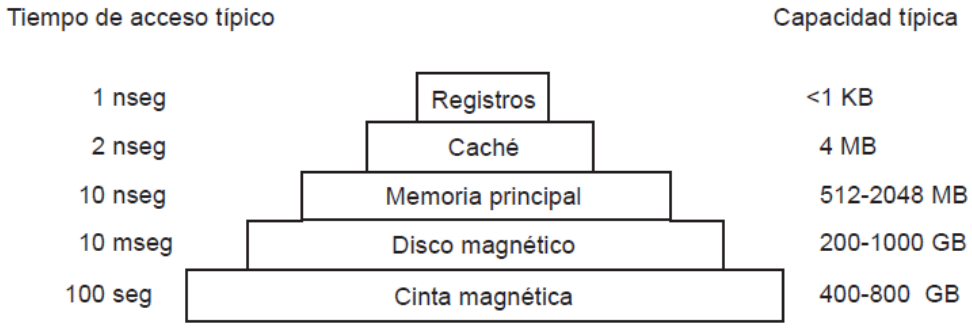
Ley de Moore: el número de transistores en un chip se duplica cada 18 meses.

Multihilamiento (multithreading): una CPU actúa como múltiples, con hilos de ejecución/threads que pueden ser alternados. No es paralelismo, pero funciona bien por la ley de Moore.

Multinúcleo (multicore): múltiples procesadores conectados por cachés dentro de una CPU. Es paralelismo verdadero.

1.4.2. Memoria

Hay tres parámetros principales que definen a una memoria y su uso indicado: velocidad, tamaño y costo. En general, a mayor velocidad, menor tamaño y mayor costo. Así, se define una jerarquía de memoria, de mayor a menor velocidad:



- Registros: igual de veloces que la CPU, tamaño de 32 x 32 bits o 64 x 64 bits dependiendo de la CPU.
- Caché: usada para acceso más veloz a datos almacenados en la memoria principal que son utilizados comúnmente. La memoria principal se divide en líneas de caché, que hacen corresponder a 64 bytes de memoria principal a una línea de caché. Cuando un procedimiento quiere obtener un dato de memoria, primero busca en las cachés. Tipos de caché:
  - L1: de 16kB, sin retraso
  - L2: de varios mB, con retraso de uno o dos ciclos de reloj

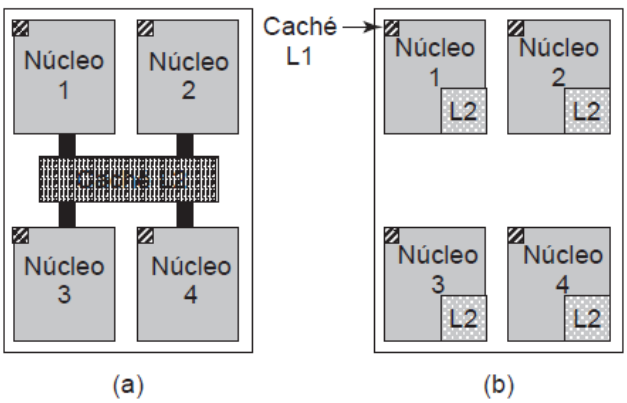


Figura 1-8. (a) Un chip de cuatro núcleos (*quad-core*) con una caché L2 compartida. (b) Un chip de cuatro núcleos con cachés L2 separadas.

- Memoria principal: es una RAM, se borra al apagarse la computadora
- Discos magnéticos (disco duro): es el doble de económico y suele tener el doble de tamaño que una RAM, pero es unas tres veces más lento que esta. Esto se debe a que es mecánico. Tiene cilindros compuestos por pistas, las cuales son leídas por la cabeza de un brazo.

- Cintas magnéticas: Es mecánica, suele utilizarse para respaldar datos en discos.
- Otros:
    - ROM (Read-Only Memory): es escrita en la fábrica y no puede ser borrada. Suele ser usada para el cargador de arranque (bootstrap loader), y en algunas tarjetas de E/S.
    - EEPROM (Electrically Erasable PROM) y flash: pueden ser borradas y reescritas, pero esto es muy lento.
    - CMOS: volátil, pero consume muy poca energía. Suele utilizarse para guardar la fecha y hora actuales.

1.4.3. Dispositivos de E/S

Consiste en el dispositivo controlador y el dispositivo en sí.

El dispositivo en sí tiene una interfaz simple estándar (para los discos, a esto se lo llama IDE (Integrated Drive Electronics, Electrónica de Unidad Integrada)). El controlador se conecta con el dispositivo en sí, y ofrece una interfaz distinta al sistema operativo.

El driver es un software que se comunica con el controlador, y se requiere uno por cada sistema operativo que soporte a un tipo de controlador específico. Requiere ejecutarse en modo kernel, por lo que debe pasar a formar parte de este. Es común necesitar reiniciar la computadora para instalar drivers (como en Windows o antiguos sistemas UNIX), aunque también existen drivers que se cargan en forma dinámica.

Dependiendo de la arquitectura de la computadora, se pueden requerir instrucciones especiales `IN` y `OUT` para E/S (si los registros de dispositivos se colocan en puertos de E/S especiales). Sin embargo, estas instrucciones no son necesarias si los registros de dispositivos se corresponden con el espacio de direcciones de sistemas operativos.

Métodos de E/S:

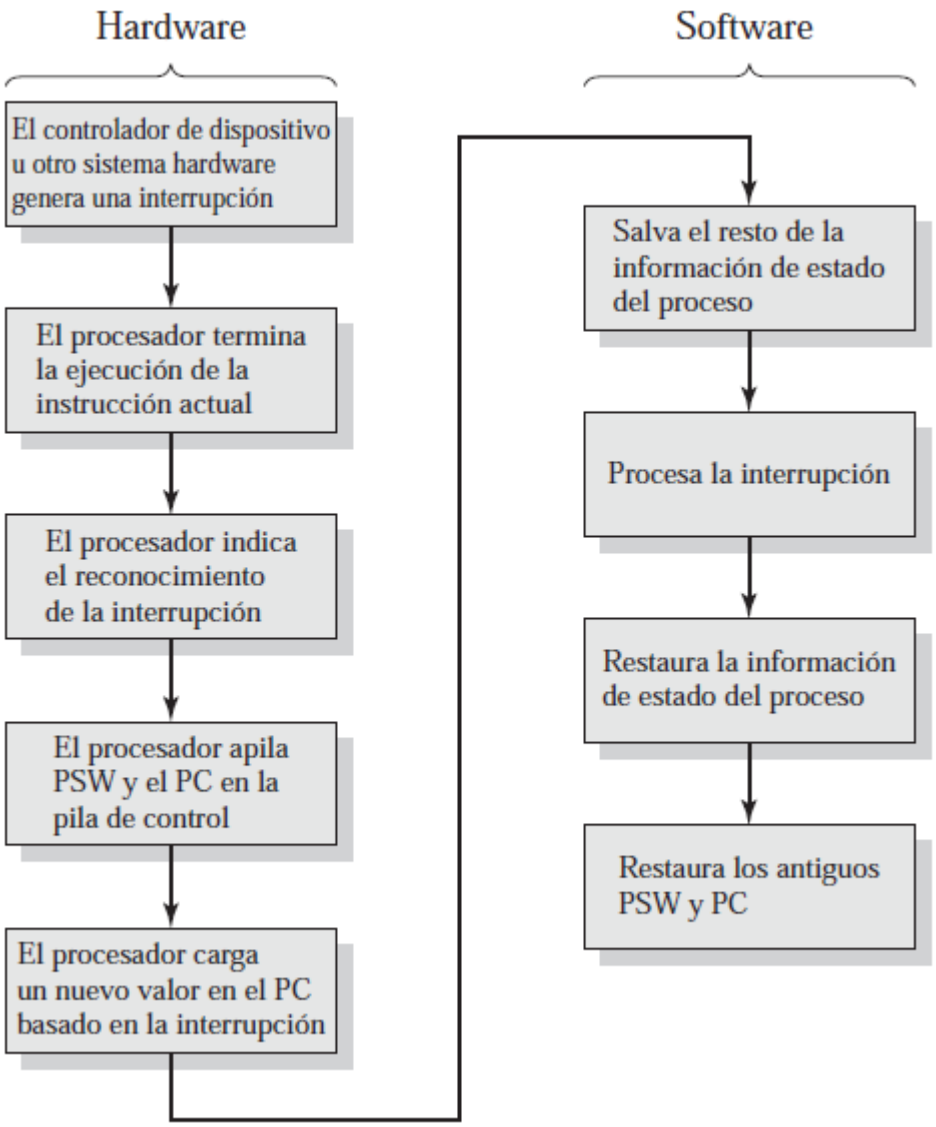
- Espera ocupada: el driver inicia la operación y espera a que termine el dispositivo. No se puede hacer nada mientras tanto.
- Interrupciones: el driver inicia la operación. El controlador genera una interrupción una vez que el dispositivo en sí termina. Esta interrupción ejecuta un manejador(handler) que pertenece al driver, y se ubica en una parte de la memoria apuntada por un elemento en un vector de interrupción.
- DMA (Direct Memory Access): utiliza un chip especial llamado DMA que maneja la comunicación entre el controlador y la memoria. La CPU lo configura al inicio, y lo deja trabajar.

La interrupciones pueden ser:

- De programa
- Por temporizador (de reloj)
- De E/S
- Por fallo del hardware

Cuando ocurre una interrupción, primero debe terminar de ejecutarse la instrucción actual (ciclos de búsqueda y ejecución) para luego ejecutarse un ciclo de interrupción.

Ejemplo de ciclo de interrupción:

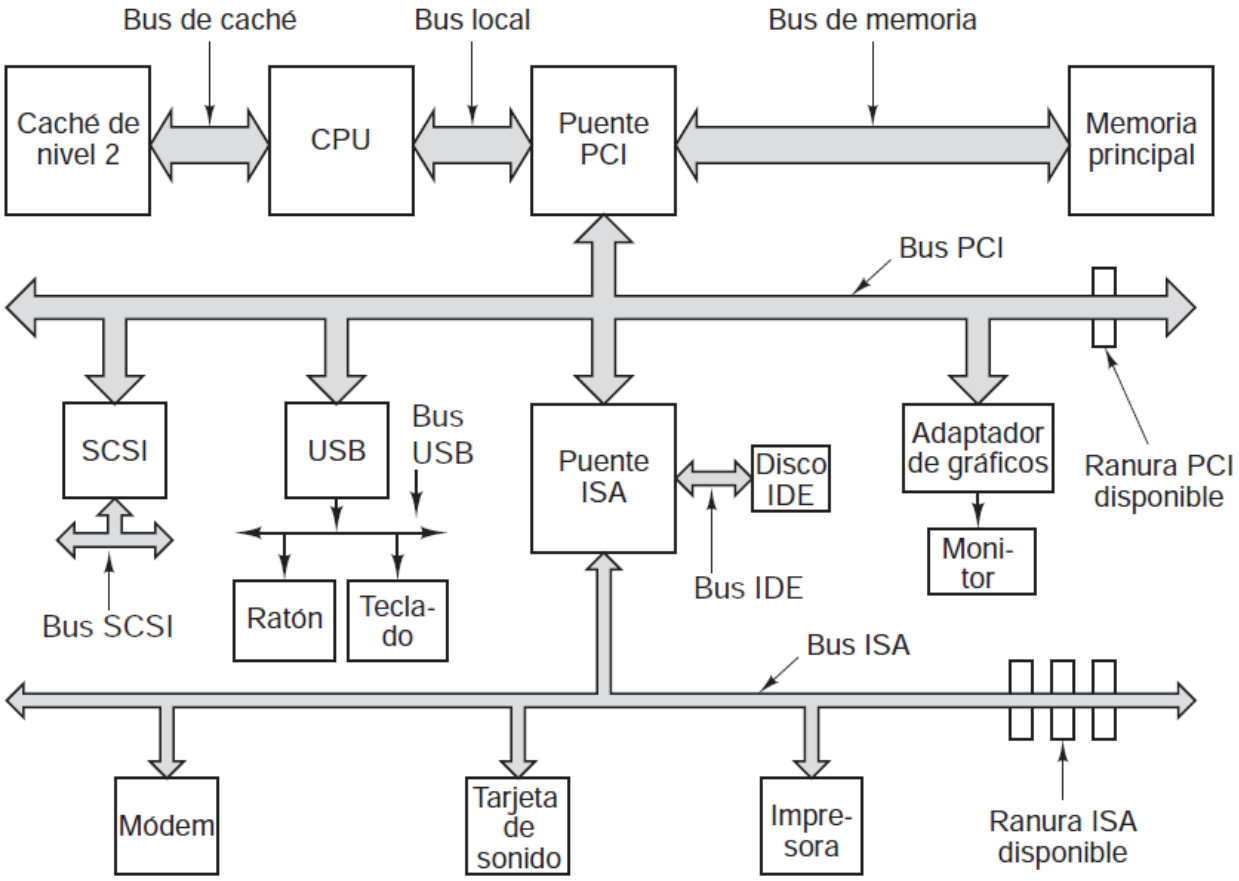


Las interrupciones pueden suceder en momentos inconvenientes; allí pueden manejarse de dos formas:

- SI: Biestable Sistema de Interrupciones. Permite bloquear las interrupciones, lo cual ocurre si ya se está manejando una.
- Priorización: se definen prioridades a la hora de atender interrupciones. Si ya se está atendiendo una y otra aparece, esta solo será atendida si tiene mayor prioridad (interrumpiendo la anterior).

1.4.4. Buses

Originalmente, solía usarse un único bus, pero con el aumento de la velocidad de los dispositivos se hizo necesaria la implementación de otros, normalmente:



- ISA (Industry Standard Architecture (Bus)): usado para mantener compatibilidad con tarjetas de E/S antiguas; obsoleto
- PCI (Peripheral Component Interconnect) y PCI Express: sucesores de ISA, más veloces
- USB (Universal Serial Bus, Bus Serial Universal): conecta con dispositivos de E/S lentos, suministrándolos de energía a la vez. Tiene un solo controlador, por lo que no es necesario reiniciar al agregar dispositivos USB
- SCSI (Small Computer System Interface): bus de alto rendimiento para dispositivos veloces que requieren un gran ancho de banda
- IDE (Integrated Drive Electronics, Electrónica de Unidad Integrada): conecta periféricos como discos duros y CD-ROM.
- IEEE 1394: bus de bits en serie más veloz que USB, pero sin un solo controlador central.
- De Memoria
- Local
- De Caché: conecta la CPU y el caché L2

Plug and play: sistema que permite asignar la información de los dispositivos de E/S, los niveles de interrupción y las direcciones de E/S de forma centralizada. Esto es para evitar que distintos periféricos interfieran entre sí porque de no estar, por ejemplo, podrían sobrescribir su información.

1.5 Arranque de la computadora (Tanenbaum, 2009, p. 33)

En las tarjetas madre, en una memoria flash, se encuentra un programa llamado BIOS (Basic Input/Output System, Sistema Básico de E/S).

Al arrancar la computadora:

1. Se ejecuta el BIOS
   - *a*) Comprueba la RAM instalada
   - *b*) Explora los buses ISA y PCI, detectando los dispositivos conectados (heredados (antes de plug and play) y no heredados)
   - *c*) Configura los dispositivos que no estaban la última vez
   - *d*) Comprueba una lista de dispositivos almacenada en la memoria CMOS, determinando el dispositivo de arranque según la siguiente prioridad:
     1) Disco flexible
     2) CD-ROM
     3) Disco duro
2. Se ejecuta un programa que se encuentra al inicio del dispositivo de arranque
   - *a*) Lee una tabla de particiones al final del sector de arranque y determina la partición activa
   - *b*) Lee un cargador de arranque secundario de la partición activa
3. Se ejecuta el sistema operativo de la partición activa
   - *a*) Pide al BIOS la información de configuración
   - *b*) Comprueba que tenga los drivers de todos los dispositivos. Solicita al usuario que instale los faltantes
   - *c*) Carga los drivers en el kernel
   - *d*) Inicialia sus tablas, crea los procesos de segundo plano requeridos y arranca un programa de inicio de sesión

1.6 Tipos de Sistemas Operativos (Tanenbaum, 2009, p. 33-37) (Stallings, 2005, p. 79-82, 172-174)

1.6.1. Mainframe

Son para computadoras que ocupan una habitación entera, con mucha memoria y capacidad para E/S. Ofrecen tres tipos de servicios:

- Procesamiento por lotes: para trabajos de rutina, sin un usuario interactivo presente
- Procesamiento de transacciones: maneja grandes cantidades de pequeñas peticiones
- Tiempo compartido: permite que múltiples usuarios remotos ejecuten trabajos en la computadora simultáneamente

1.6.2. De servidores

Dan servicio a varios usuarios por una red y les permite compartir los recursos.

1.6.3. De multiprocesadores

Conectan varias CPU a un mismo sistema. Según la forma en que un sistema operativo asigna procesos a los procesadores, puede ser:

- **SMP** significa Symmetric Multiprocessing, y hace referencia a un sistema de computación multiprocesador en el que todos tienen acceso a los mismos recursos y pueden realizar las mismas funciones. Cada procesador realiza su propia planificación de procesos.
- **Maestro-esclavo** consiste en un procesador central, en el cuál se ejecuta el núcleo del sistema operativo y se planifican los procesos de todos los otros procesadores.
- **Cluster**; a diferencia de los anteriores (multiprocesadores de memoria compartida), en este esquema cada procesador tiene su propia memoria.

1.6.4. De computadoras personales

Proporcionan buen soporte para un solo usuario.

1.6.5. De computadoras de bolsillo / PDA (Personal Digital Assistant)

Para computadoras de pequeño tamaño que realizan una variedad de funciones

1.6.6. Integrados / Incrustados / Embedded

Para computadoras que no aceptan software instalado por el usuario, pues se ubica en una ROM. Usado en autos, microondas y reproductores de MP3, etc.

1.6.7. De nodos sensores

Para redes de pequeñas computadoras que se comunican inalámbricamente. Suelen trabajar en ambientes hostiles, desatendidas por largos periodos y con energía limitada, por lo que debe soportar eventuales fallas en sus nodos.

1.6.8. En tiempo real

El tiempo es un parámetro clave. Si puede tolerar alguna falla, se dice que es suave; caso contrario, es duro. Se suelen utilizar en la industria (duros) o en sistemas de audio digital o teléfonos digitales (suaves).

1.6.9. De tarjetas inteligentes

Para dispositivos del tamaño de una tarjeta de crédito, por lo que son muy pequeños y tienen pocas funciones, frecuentemente una sola.

1.7 Conceptos de los Sistemas Operativos (Tanenbaum, 2009, p. 37-49)

1.7.1. Proceso

Programa en ejecución.

Pueden ejecutarse muchos a la vez, y ser suspendidos de forma temporal.

El sistema operativo guarda una tabla de procesos, que guarda estructuras con información acerca de cada proceso.

Cada proceso consiste en su **imagen de núcleo** (espacio de direcciones) y su entrada en la tabla de procesos.

Un proceso puede crear procesos hijos, creando así un árbol de procesos.

Una señal (entre procesos) es al software lo que una interrupción es al hardware.

Los procesos pueden comunicarse entre sí. Estos, pueden estar o no esperando la comunicación. Si no la esperan y transcurre un cierto tiempo, se la considera perdida y se envía una señal de alarma al proceso emisor para que reenvíe los datos, suspendiéndolo temporalmente.

Cada persona autorizada a usar un sistema tiene una UID (User Identificacion), y cada proceso guarda la UID de quien lo inició. Los usuarios pueden pertenecer a un grupo, los cuales tienen una GID (Group Identification).

Existe un superusuario con poder especial, al que solo pueden acceder los administradores con una contraseña.

1.7.2. Espacios de direcciones

Cada proceso guarda un conjunto de direcciones que puede utilizar, desde 0 hasta un valor máximo. Si este valor máximo excede el espacio disponible en la memoria principal, se puede almacenar parte del programa en disco e irlo cargando en memoria a medida que es requerido. Esto se llama **memoria virtual**.

También se pueden almacenar varios procesos en memoria simultáneamente, y para evitar que interfieran entre sí y con el sistema operativo se implementa un mecanismo de protección por hardware, el cual es controlado por el software.

1.7.3. Archivos

Un sistema de archivos es un árbol de archivos organizados en directorios, que buscan ocultar las peculiaridades de los discos y otros dispositivos de E/S.

Un nombre de ruta es una cadena que permite identificar un archivo en el sistema de archivos. Es absoluto si se comienza a buscar desde el directorio raíz (comienza con / en UNIX); caso contrario se comienza a buscar desde el directorio de trabajo actual de un proceso.

Al intentar abrir un archivo, se comprueban los permisos y devuelve un entero llamado descriptor para ser utilizado para leer y escribir en él.

Un sistema de archivos montado permite unir dos sistemas distintos en uno solo, al hacer que un directorio vacío del primero apunte al directorio raíz del otro.

Los dispositivos de E/S pueden ser vistos como archivos, para realizar operaciones de E/S con las mismas llamadas que para leer y escribir en archivos. Esto se logra con los archivos especiales. Hay dos tipos:

- De bloque: modelan dispositivos como discos, que contienen un conjunto de bloques de acceso aleatorio y permite acceder a ellos de forma directa, sin importar la estructura del sistema de archivos que contenga.
- De carácter: modelan dispositivos como impresoras o módems, que emiten o reciben un flujo de caracteres.

Un canal (pipe) es un pseudoarchivo que permite la comunicación entre procesos, utilizando las mismas operaciones de lectura y escritura.

1.7.4. Entrada/salida

Existe un subsistema de E/S en todo sistema operativo, con partes tanto dependientes como independientes del dispositivo.

1.7.5. Protección

Se refiere a la necesidad de los usuarios de ocultar información. Hay sistemas para limitar el acceso a otros usuarios, y sistemas para proteger de intrusos no deseados (atacantes o virus).

En UNIX, respecto a la protección de archivos y directorios, cada uno tiene un campo de 9 bits dividido en 3 subcampos de 3 bits. Los subcampos se refieren al usuario, al grupo del usuario y a otros usuarios fuera del grupo. Los bits de cada subcampo se refieren a los permisos para leer, escribir y ejecutar (o buscar, si es un directorio) (rwx).

1.7.6. Shell

Se refiere a un intérprete de comandos. Hay muchos shells para distintos sistemas operativos.

Tienen un caracter indicador de comandos (como $).

Para invocar un programaa se escribe su nombre.

Para indicar que la entrada de un programa NO es la estándar (shell), se usa el operador $<$; y para indicar que la salida no es la estándar, se usa el operador $>$.

Para crear un canal (pipe), se usa el operador $|$, el cual indica al programa a la derecha que su entrada estándar es la salida del de la izquierda.

El signo & al final de un comando indica que se debe ejecutar en segundo plano.

1.7.7. Evolución y obsolecencia

El avance de la tecnología hace que algunas se vuelvan obsoletas, pero a su vez puede provocar que otras consideradas obsoletas resurjan.

- **Memorias extensas:** las memorias pequeñas exigen un muy buen manejo del espacio disponible. Por esto, suelen ser programadas en ensamblador. Con los aumentos en capacidad, se vuelve viable programar con lenguajes de mayor nivel; pero al desarrollar nuevas computadoras con memorias más pequeñas, suele volver la necesidad de usar ensamblador.
- **Hardware de protección:** la multiprogramación exige hardware de protección que evite que un proceso utilice memoria fuera de su imagen de núcleo, pero también aumenta la complejidad requerida por el hardware y el sistema operativo. Por esto, al desarrollar nuevos procesadores

muchas veces no han tenido hardware de protección ni multiprogramación, pero con el tiempo suelen agregar soporte.

- **Discos:** los dispositivos de almacenamiento de pequeña capacidad no requieren un sistema de archivos complejo; con un solo directorio alcanza. Al desarrollarse avances en los discos, a veces han vuelto a ser usados estos sistemas de un directorio.
- **Memoria virtual:** permite ejecutar programas más grandes que la memoria principal, y ligar dinámicamente bibliotecas en tiempo de ejecución.

1.8 Llamadas al sistema (Tanenbaum, 2009, p. 49-61)

Son similares a un procedimiento, con las diferencias de que: se ejecutan en modo kernel y son activadas por la instrucción `TRAP`.

`TRAP` suele tomar un campo de 8 bits que corresponden al índice de una tabla que contiene apuntadores a manejadores de llamadas al sistema.

Un procedimiento de biblioteca provee una interfaz para los lenguajes de programación para realizar las llamadas al sistema, colocando los parámetros en los registros correspondientes y devolviendo el valor requerido.

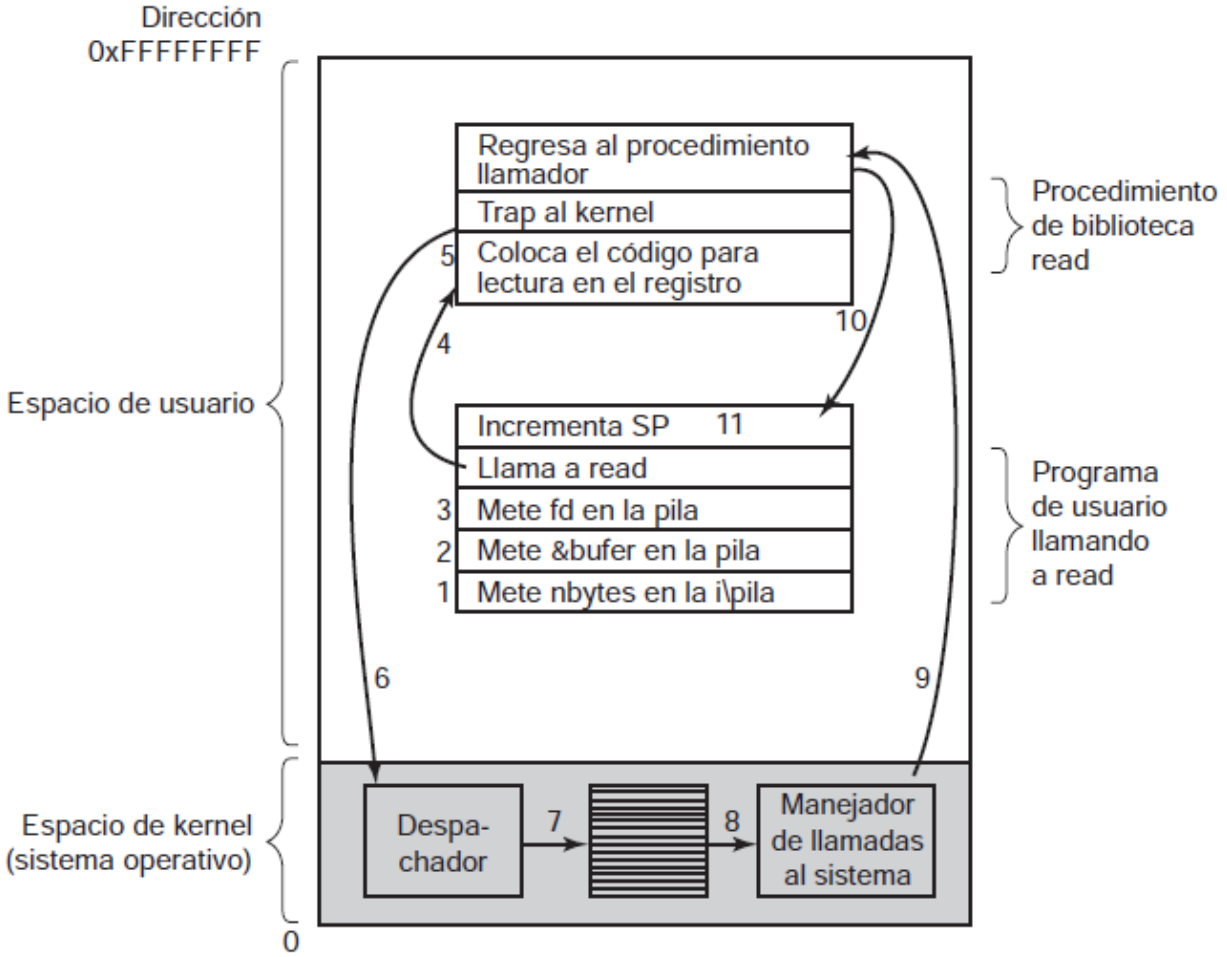


Figura 1-17. Los 11 pasos para realizar la llamada al sistema `read(fd, bufer, nbytes)`.

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

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

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

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

1.8.1. Llamadas al sistema para la administración de recursos

- `fork` crea una copia del proceso; su valor de retorno es 0 en el hijo y `pid` (process identifier) en el padre.
- `waitpid` espera a que finalice un proceso hijo específico o cualquiera (si el `pid` pasado es -1), y se coloca el valor de retorno del hijo en `statloc`.

- `exit` finaliza un proceso y devuelve el `status`.
- `execve` sustituye toda la imagen de núcleo del proceso por el archivo especificado en el primer parámetro.

1.8.2. Llamadas al sistema para la administración de archivos

- `open` abre un archivo para lectura, escritura, ambas o lo crea. Devuelve un `fd` (file descriptor).
- `close` para cerrar un archivo y liberar el `fd`.
- `read, write` leen o escriben a un archivo.
- `lseek` mueve el apuntador de posición de un archivo respecto del inicio, fin o posición actual.
- `stat, fstat` permiten accesso a información de un archivo (modo (regular, especial, directorio, etc.), tamaño, última modificación, etc.).

1.8.3. Llamadas al sistema para la administración de directorios

En UNIX, todo archivo tiene un número único (inumber) que lo identifica. Este es un índice en una tabla de inodes, uno por archivo, que indican su propietario, sus bloques en disco, cantidad de enlaces, etc.

Un directorio es un archivo que contiene un conjunto de pares: un inumber y un nombre.

- `mkdir, rmdir` crean o remueven un directorio vacío.
- `link` crea un enlace a un archivo, permitiendo acceder a este bajo distintas rutas.
- `unlink` elimina un enlace. Si este era el último a un archivo, se elimina el archivo.
- `mount, umount` monta o desmonta un sistema de archivos, permitiendo integrar una jerarquía de archivos.

1.8.4. Llamadas varias

- `chdir` cambia el directorio de trabajo
- `chmod` cambia los bits de protección de un archivo (rwx-rwx-rwx)
- `kill` envía una señal a un proceso. Si el proceso está preparado para recibirla, se ejecuta un manejador de señales; caso contrario lo mata.
- `time` devuelve la cantidad de segundos desde el 1 de enero de 1970 a medianoche.

1.8.5. Windows

Microsoft provee la API Win32, una biblioteca con procedimientos que forman la interfaz de los programadores con el sistema operativo desde Windows 95.

No todos los procedimientos que provee involucran llamadas al sistema, y esto puede incluso variar entre versiones de Windows (la GUI es manejada por el kernel en algunas, pero no en todas).

UNIX	Win32	Descripción
fork	CreateProcess	Crea un nuevo proceso
waitpid	WaitForSingleObject	Puede esperar a que un proceso termine
execve	(ninguno)	CreateProces = fork + execve
exit	ExitProcess	Termina la ejecución
open	CreateFile	Crea un archivo o abre uno existente
close	CloseHandle	Cierra un archivo
read	ReadFile	Lee datos de un archivo
write	WriteFile	Escribe datos en un archivo
lseek	SetFilePointer	Desplaza el apuntador del archivo
stat	GetFileAttributesEx	Obtiene varios atributos de un archivo
mkdir	CreateDirectory	Crea un nuevo directorio
rmdir	RemoveDirectory	Elimina un directorio vacío
link	(ninguno)	Win32 no soporta los enlaces
unlink	DeleteFile	Destruye un archivo existente
mount	(ninguno)	Win32 no soporta el montaje
umount	(ninguno)	Win32 no soporta el montaje
chdir	SetCurrentDirectory	Cambia el directorio de trabajo actual
chmod	(ninguno)	Win32 no soporta la seguridad (aunque NT sí)
kill	(ninguno)	Win32 no soporta las señales
time	GetLocalTime	Obtiene la hora actual

1.9 Estructuras de sistemas operativos (Tanenbaum, 2009, p. 62-72) (Stallings, 2005, p. 76-79)

- Sistemas monolíticos: el sistema operativo es una colección de procedimientos, y se ejecuta como un solo programa en modo kernel. No hay ocultamiento de información, pero se logra cierta estructura al clasificar los procedimientos en programa principal, de servicio (activan las `TRAP`) y utilitarios (ayudan a los de servicio).

- Sistema de capas: el sistema operativo divide los procesos en capas o anillos, cada uno dependiendo de la capa anterior y con menos privilegios mientras mayor sea el nivel. Ejemplos: THE y MULTICS. Una propuesta para sistema operativo de este tipo es hecha por Brown, R. y Denning, P., con 13 capas agrupadas en tres grupos: hardware (primeras 4 capas, no son el sistema operativo en sí), procesador únicamente (capas 5 a 7) y dispositivos externos (capas 8 a 13).
- Microkernel: sistema operativo en capas que busca reducir al máximo el tamaño del kernel, para así volverlo menos propenso a errores.
- Modelo cliente-servidor: se clasifican los procesos en servidores y clientes, tal que los clientes envían un mensaje a los servidores, estos los procesan y responden. Este modelo puede ser usado tanto en redes como en una misma computadora.
- Máquinas virtuales: se refieren a un sistema operativo que se ejecuta en simultáneo con otros. Puede implementarse como hipervisores de tipo 1 (se ejecuta directamente sobre el hardware) o de tipo 2 (se ejecuta sobre otro sistema operativo).
- Exokernel: otro método para implementar máquinas virtuales, consiste en crear una capa debajo de los kernels llamada exokernel, programa encargado de asignar recursos a las máquinas virtuales y comprobar que ninguna intente usar los de las otras.

Capítulo 2

Procesos

2.1 Modelo del proceso (Tanenbaum, 2009, p. 83-86)

Consiste en la idea de que todo el software ejecutable se organiza en varios procesos secuenciales.
Un proceso es una actividad, que tiene un programa, una entrada, una salida y un estado.
En cada CPU solo se puede ejecutar un proceso a la vez, por lo que para lograr un pseudoparalelismo cada proceso debe tener su propio contador de programa.

2.2 Creación de procesos (Tanenbaum, 2009, p. 86-88)

Pueden ser creados por:

- Arranque del sistema: tanto procesos en primer plano (que interactúan con el usuario) como en segundo plano (demonios / daemons).
- Ejecución de una llamada al sistema desde otro proceso.
- Petición de usuario (por comando o con doble clic en un ícono, por ejemplo).
- Inicio de un trabajo por lotes (al enviar trabajos de procesamiento por lotes a mainframes, que los coloca en una cola de entrada y los ejecuta creando un proceso cuando puede).

2.3 Terminación de procesos (Tanenbaum, 2009, p. 88-89)

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

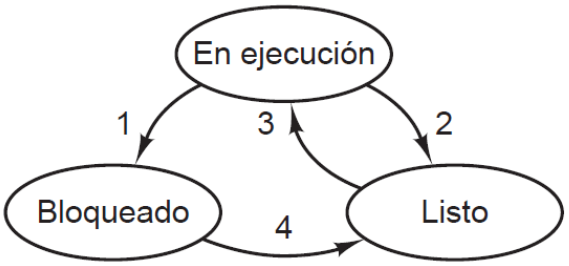
2.4 Jerarquía de procesos (Tanenbaum, 2009, p. 89)

Se refiere a la estructura que surge al un proceso padre crear procesos hijos.

En Windows no existe, pues si bien el padre recibe un token (manejador) al crear al hijo que le permite controlarlo, no hay restricciones de pasar el token a otros procesos, invalidando la jerarquía.

En UNIX sí existe. Un proceso y todos sus descendientes forman un grupo de procesos. Además, todos los procesos pertenecen a un mismo grupo llamado de raíz `init`, que es el proceso llamado al ejecutar el sistema operativo.

2.5 Estados de un proceso (Tanenbaum, 2009, p. 90-91) (Stallings, 2005, p. 117-120)



1. El proceso se bloquea para recibir entrada
2. El planificador selecciona otro proceso
3. El planificador selecciona este proceso
4. La entrada ya está disponible

El planificador de procesos es una parte del sistema operativo que asigna la CPU a los distintos procesos para lograr el pseudoparalelismo. Se encarga de alternar entre los estados “En ejecución” y “Listo”.

Cuando un proceso requiere una entrada que no está lista, su estado pasa a “Bloqueado” y, cuando finalmente lo esté, pasa a “Listo”.

También se puede considerar la existencia de otros dos estados adicionales: “Nuevo” y “Saliente”.

2.6 Tabla de procesos (Tanenbaum, 2009, p. 91-93)

Los sistemas operativos mantienen una tabla de procesos, con una entrada (llamada bloque de control de procesos (BCP)) por procesos. Cada entrada contiene, por lo general (depende del sistema operativo):

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		

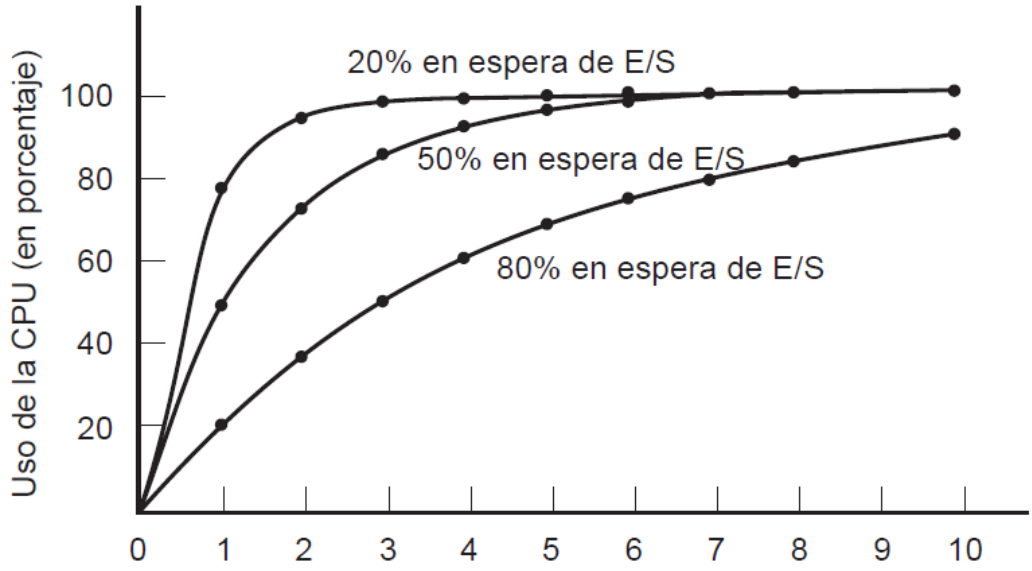
Gracias a esta tabla es que funcionan el planificador de procesos y las interrupciones.
Las interrupciones son manejadas según el algoritmo:

1. El hardware mete a la pila los registros del proceso.
2. El hardware coloca en el contador de programa la dirección en el vector de interrupciones asociada al servicio de interrupción correspondiente.
3. Se guardan los registros en la entrada de la tabla de procesos y se los quita de la pila.
4. Se establece una nueva pila para el manejador de la interrupción.
5. Se ejecutra el servicio de interrupciones.
6. El planificador decide el siguiente proceso a ejecutar.
7. Se cargan los registros y el mapa de memoria del siguiente proceso.
8. Se ejecuta el siguiente proceso.

2.7 Modelado de la multiprogramación (Tanenbaum, 2009, p. 93-95)

$$Uso_{CPU} = 1 - p^n$$

p: porcentaje del tiempo de un proceso en espera de completar una operación de E/S.
n: grado de multiprogramación: cantidad de procesos en memoria a la vez.



2.8 Hilos de control (Tanenbaum, 2009, p. 95-97)

"Miniprocesos" que forman parte de un proceso y se ejecutan en cuasi-paralelo. Múltiples hilos creados por un proceso comparten su mismo espacio de direcciones.

Son más rápidos de crear, intercambiar y eliminar que los procesos, permiten ejecutar actividades de forma que no interfieran con otras y pueden aumentar el rendimiento en procesadores multinúcleo.

2.9 Formas de construir un servidor (Tanenbaum, 2009, p. 97-100)

Modelo	Características
Hilos	Paralelismo, llamadas al sistema con bloqueo
Proceso con un solo hilo	Sin paralelismo, llamadas al sistema con bloqueo
Máquina de estados finitos	Paralelismo, llamadas al sistema sin bloqueo, interrupciones

2.10 Hilos clásicos (Tanenbaum, 2009, p. 100-104)

Multihilamiento es la presencia de múltiples hilos en un mismo proceso. Para funcionar en una sola CPU, se van turnando (como los procesos), y para ello se requieren 4 estados: en ejecución, bloqueado, listo o terminado.

Los procesos poseen algunos datos a los que, por compartir espacio de memoria, sus hilos pueden acceder; pero los hilos también tienen sus datos propios necesarios para su funcionamiento. Si bien no hay protección para que un hilo modifique los elementos de otro, esto no debería ser necesario, pues los hilos deben cooperar entre sí (a diferencia de los procesos).

Elementos por proceso	Elementos por hilo
Espacio de direcciones	Contador de programa
Variables globales	Registros
Archivos abiertos	Pila
Procesos hijos	Estado
Alarmas pendientes	
Señales y manejadores de señales	
Información contable	

Existen procedimientos de biblioteca para manipular hilos similares a aquellos para los procesos: `thread_create, thread_exit, thread_join, thread_yield` (crear hilo, salir del hilo, bloquear hilo llamador hasta que otro retorne, entregar la CPU para otros hilos)

2.11 Hilos en POSIX: Pthreads (Tanenbaum, 2009, p. 104-106)

El IEEE ha definido un estándar llamado 1003.1c, con un paquete Pthreads, usado por la mayoría de los sistemas UNIX.

Cada hilo Pthreads tiene un identificador, un conjunto de registros y un conjunto de atributos.

Pthreads ofrece más de 60 llamadas para manipular hilos, algunas son:

Llamada de hilo	Descripción
Pthread_create	Crea un nuevo hilo
Pthread_exit	Termina el hilo llamador
Pthread_join	Espera a que un hilo específico termine
Pthread_yield	Libera la CPU para dejar que otro hilo se ejecute
Pthread_attr_init	Crea e inicializa la estructura de atributos de un hilo
Pthread_attr_destroy	Elimina la estructura de atributos de un hilo

2.12 Implementación de hilos según el espacio (Tanenbaum, 2009, p. 106-112) (Stallings, 2005, p. 165-169)

2.12.1. En el espacio de usuario / ULT (User-Level Threads)

Son muy veloces. Cada proceso tiene una tabla de hilos y se encarga de administrarla, lo que permite algoritmos de planificación personalizados de ser necesarios. Pueden implementarse independientemente del soporte del sistema operativo.

Las llamadas al sistema pueden producir bloqueos, y estos son en el proceso en sí, por lo que el sistema operativo no puede saber que es solo un hilo el que no puede avanzar, y forzará un cambio de contexto entre procesos (que es más costoso que entre hilos). Esto puede evitarse comprobando si la llamada al sistema produciría un bloqueo, tal técnica se llama jacketing. Los fallos de página (salto a

una instrucción fuera de la memoria principal) también bloquean el proceso hasta que se haya cargado la memoria necesaria. Es requerido que un hilo renuncie a la CPU para que otros puedan ejecutarse.

2.12.2. En el kernel / KLT (Kernel-Level Threads)

La tabla de hilos se encuentra en el kernel. Al realizar llamadas al sistema, se bloquea el hilo y no el proceso, por lo que el planificador puede decidir si ejecutar un hilo del mismo u otro proceso. Lo mismo ocurre con los fallos de página.

Son lentos, debido al costo de procesamiento de las llamadas al sistema, en especial para crear y destruir hilos.

2.12.3. Híbridas

Tiene hilos en el kernel, y a cada uno de ellos le corresponde un conjunto de hilos en el espacio de usuario. Esto es así para obtener mayor velocidad (al no crear y destruir tantos hilos del kernel) y flexibilidad.

2.12.4. Problemas

Al crear un procedimiento con `fork`, ¿se deben replicar todos los hilos o solo uno?

Las señales se envían a los procesos, no hilos. ¿Qué hilo debería encargarse de recibirlas?

2.12.5. Activaciones del planificador

Busca la eficiencia de la implementación en el espacio de usuario con la funcionalidad de hacerlo en el kernel. Lo logra evitando transiciones innecesarias entre el usuario y el kernel. También, enviando una llamada ascendente (upcall) del kernel al sistema en tiempo de ejecución para informar del bloqueo de un hilo y replanificar los hilos en el espacio de usuario, y enviando otra llamada ascendente cuando un proceso bloqueado está listo.

Por último, si se produce una interrupción que no es de interés para el proceso interrumpido, al terminar el manejador de interrupciones se coloca el hilo interrumpido de vuelta en el estado en que estaba antes de ser interrumpido. Caso contrario, el hilo interrumpido se suspende y el sistema en tiempo de ejecución se inicia en la CPU virtual, para decidir cuál hilo planificar.

2.13 Hilos emergentes (Tanenbaum, 2009, p. 112-113)

Hay dos formas de manejar un mensaje entrante (señal): con un hilo bloqueado en una llamada al sistema `receive` o creando un **hilo emergente** (pop-up thread) cada vez que se reciba un mensaje.

Los hilos emergentes en el kernel son más veloces que si estuviera en el espacio de usuario pero, de haber errores, estos serían más peligrosos.

2.14 Comunicación entre procesos (Tanenbaum, 2009, p. 117-145) (Stallings, 2005, p. 180-181, 202-241)

Se debe resolver tres cuestiones respecto a la comunicación entre procesos (e hilos, pero solo las dos últimas): pasar información, no entrar en conflicto y usar como entrada la salida de otro.

2.14.1. Condiciones de carrera y regiones críticas

Las condiciones de carrera (race conditions) se producen cuando dos o más procesos leen o escriben simultáneamente la misma información compartida, ocasionando que la salida de estos procesos dependa del orden en que su ejecución es organizada por el planificador.

Las condiciones de carrera pueden evitarse con exclusión mutua: evitando que más de un proceso escriba o lea la información compartida a la vez. Para esto, se define como **regiones críticas** a las zonas de un proceso en que se lee o escribe, y se definen ciertas necesidades:

1. Solo puede haber un proceso a la vez en una región crítica.
2. No se debe hacer suposiciones de la velocidad o números de CPUs.
3. Ningún proceso fuera de su región crítica puede bloquear otros procesos.
4. Ningún proceso debe esperar para siempre para entrar a su región crítica.

2.14.2. Exclusión mutua con espera ocupada

Deshabilitación de interrupciones

Consiste en desactivar las interrupciones al entrar a una región crítica y reactivarlas al salir. Esto no funciona en CPUs multinúcleo porque solo de desactivan en un núcleo, mientras que los demás podrían seguir accediendo a la memoria compartida.

Además, para procesos de usuario, no es deseable dar permiso para apagar o prender interrupciones, pues podrían quedar desactivadas por error y causar fallos críticos.

Variable de candado

Consiste en crear una variable en la memoria compartida cuyo valor sea 0 si no hay procesos en su región crítica, y 1 si sí los hay. Este método podría fallar, pues cambia unas condiciones de carrera por otra.

Alternancia estricta

Consiste en crear una variable en la memoria compartida. Esta variable guardará un valor que corresponda a uno de los procesos que pueden acceder a esa memoria.

Cuando un proceso quiere entrar en su región crítica, solo lo hace si la variable corresponde a él. Si no, entra en un bucle a esperar a que sí lo haga. Al salir de su región crítica, cambia el valor de la variable para permitir que otro proceso se ejecute en su región crítica.

Esta solución no es deseable, pues no satisface la necesidad 3: un proceso puede estar esperando a ejecutar su región crítica, mientras que el proceso que corresponde al valor actual de la variable no está en su región crítica. Además, como esta espera a que el valor de la variable cambie se produce en un bucle, consume mucha CPU.

Solución de Peterson

Consiste en dos procedimientos, uno para entrar y otro para salir de la región crítica. `entrar` indica que el proceso está interesado en entrar en su región crítica, y que el siguiente turno debería ser suyo. Luego, espera a que el otro proceso indique que ya no está interesado (lo cual hace llamando a `salir`).

Instrucción `TSL` / `XCHG`

Instrucciones existentes en computadoras multiprocesadores. `TSL REGISTRO, CANDADO` guarda `CANDADO` en `REGISTRO` y coloca 1 en `CANDADO`. `XCHG REGISTRO, CANDADO` intercambia los valores de `REGISTRO` y `CANDADO`, y debe ser precedida por una instrucción `MOVE REGISTRO, 1` para tener el mismo efecto que `TSL`.

Funciona de una forma similar a la alternancia estricta. `entrar` consiste en un bucle que coloca 1 en `CANDADO`, y solo sale cuando el valor anterior de este registro fuera 0. `salir` solo coloca un 0 en `CANDADO`.

```
entrar_region:
    TSL REGISTRO,CANDADO        |copia candado al registro y fija candado a 1
    CMP REGISTRO,#0             |¿era candado cero?
    JNE entrar_region           |si era distinto de cero, el candado está cerrado, y se repite
    RET                         |regresa al llamador; entra a región crítica


salir_region:
    MOVE CANDADO,#0             |almacena 0 en candado
    RET                         |regresa al llamador
```

2.14.3. Exclusión mutua con bloqueo

El problema del productor-consumidor / búfer limitado

Dos procesos comparten un búfer común de tamaño N. El proceso productor coloca datos en el búfer, y el consumidor los saca. El productor solo debería poder colocar datos si el búfer no está lleno, y el consumidor solo debería sacarlos si no está vacío.

Este es una simplificación que busca mostrar por qué la espera ocupada no es conveniente para manejar la comunicación entre procesos, y se debe preferir los bloqueos. La diferencia es que los segundos no desperdician la CPU.

Dormir y despertar

Llamadas al sistema que provocan el bloqueo del proceso llamador y el despertar de un proceso indicado por un parámetro, respectivamente.

Aplicado al problema del productor-consumidor:

- Cuando el búfer se llene, productor se duerme.
- Cuando el consumidor quita un elemento, despierta al productor.
- Cuando el búfer se vacía, consumidor se duerme.

- Cuando el productor agrega un elemento, despierta al consumidor.

Puede ocurrir que uno de los procesos sea interrumpido antes de dormirse por el otro, y el otro intente despertarlo (perdiendo la señal). Luego, el primero se dormiría para nunca despertar (y, eventualmente, el segundo también). Esto puede solucionarse con un bit de espera de despertar que evite perder la señal, pero en un ejemplo más complejo puede no ser suficiente.

Semáforos

Registro con un número entero, por defecto 0. Una llamada a `down` lo decrementa si es mayor a 0, caso contrario pone a dormir al proceso sin concretar la llamada. Una llamada a `up` incrementa al semáforo y, si uno o más procesos estaban durmiendo en el semáforo, despierta a uno arbitrariamente, el cual termina de ejecutar la instrucción `down` (devolviendo al semáforo a 0).

Es importante que `up` y `down` deben estar implementadas de forma atómica como llamadas al sistema, para evitar condiciones de carrera y lograr la sincronización.

Los semáforos binarios o *mutexes* son aquellos utilizados para lograr la exclusión mutua. Son inicializados a 1, y cada proceso debe ejecutar `down` para entrar a su región crítica y `up` para salir. Así, solo un proceso (el primero en realizar `down`) se ejecutará. El resto que realice `down` mientras el semáforo está en 0, se dormirá, y cuando el primero regrese (con `up`), alguno será despertado. `down` sirve para entrar a la región crítica, y `up` para salir.

Aplicado al problema del productor-consumidor:

Se crean tres semáforos: llenos (valor inicial = 0, consumidor lo disminuye y productor lo aumenta), vacíos (valor inicial = N, productor lo disminuye y consumidor lo aumenta) y mutex (semáforo binario, para lograr exclusión mutua).

Mutexes y variables de condición

Un mutex es una variable que puede estar cerrada o abierta. Si un hilo quiere entrar en su región crítica, lo hace solo si el mutex está abierto (y lo cierra luego); caso contrario, el hilo se bloquea.

Las variables de condición permiten bloquear un hilo (`wait`) hasta que otro lo desbloquee (`signal`). Además, `wait` toma un mutex bloqueado como argumento, al cual desbloquea en un principio para que se ejecuten otros hilos, y lo bloquea al recibir la señal.

Se usan en conjunto con los mutex para esperar cuando un hilo no puede obtener algo que desea, e indicar cuando ya puede obtenerlo.

```
mutex_lock(&mutex);                          //Bloquear mutex
while(cond) {
    var_cond_wait(&var_cond1, &mutex);       //Esperar señal 1
}
//Acciones
var_cond_signal(&var_cond2);                 //Enviar señal 2
mutex_unlock(&mutex);                        //Desbloquear mutex
```

Monitores

Los mutexes y las variables de condición, si bien funcionan, son complejos y propensos a errores. Una alternativa a ellos son los monitores: conjuntos de procedimientos, variables y estructuras de

datos. Los dos últimos solo son accesibles desde los procedimientos, y el monitor garantiza que solo pueda haber uno de estos activo. También requieren variables de condición.

La mayor ventaja que tienen es que deben ser implementados por el compilador, por lo que evita los posibles errores introducidos por el programador con los mutexes. Solo una selección de lenguajes soportan monitores, aunque algunos (como Java) permiten implementarlos con sencillez.

2.14.4. Otros métodos

Pasaje / Transmisión de mensajes

Involucra dos llamadas al sistema: `send` y `receive`. `send` envía un mensaje(datos) desde un proceso, y `receive` lo recibe.

Un mensaje puede perderse. Para prevenir la pérdida de comunicación, al recibir un mensaje, todo proceso envía otro como respuesta llamado *acknowledgement*. Si el proceso emisor no lo recibe, vuelve a enviar el mismo mensaje. Además, cada mensaje tiene un número de secuencia que lo identifica, por lo que si se perdiera el mensaje de acknowledgement y se enviara dos veces el mismo mensaje, el receptor lo sabría e ignoraría el segundo.

Los mensajes guardan también una identificación del proceso emisor, para que así el receptor pueda autenticarlo.

Direccionamiento se refiere a dónde apuntan los mensajes, dónde serán recibidos. Esto puede ser un búfer, llamado **buzón** o **puerto**, que luego será manejado por el proceso. La otra alternativa es un proceso en sí, lo que implica una estrategia llamada **encuentro**, consistente en bloquear al receptor hasta que el emisor emita un mensaje; y bloquear al emisor hasta que el receptor esté listo para recibir un mensaje.

Barreras

Busca sincronizar distintos procesos. Consiste en crear una o más barreras que bloquean a los procesos hasta que todos hayan llegado a ella, luego los libera.

2.15 Planificación (Tanenbaum, 2009, p. 145-161)

Se refiere al algoritmo que selecciona el proceso (o hilo) a ejecutar cuando hay varios en estado listo. Es realizado por el Planificador de procesos, una parte del kernel.

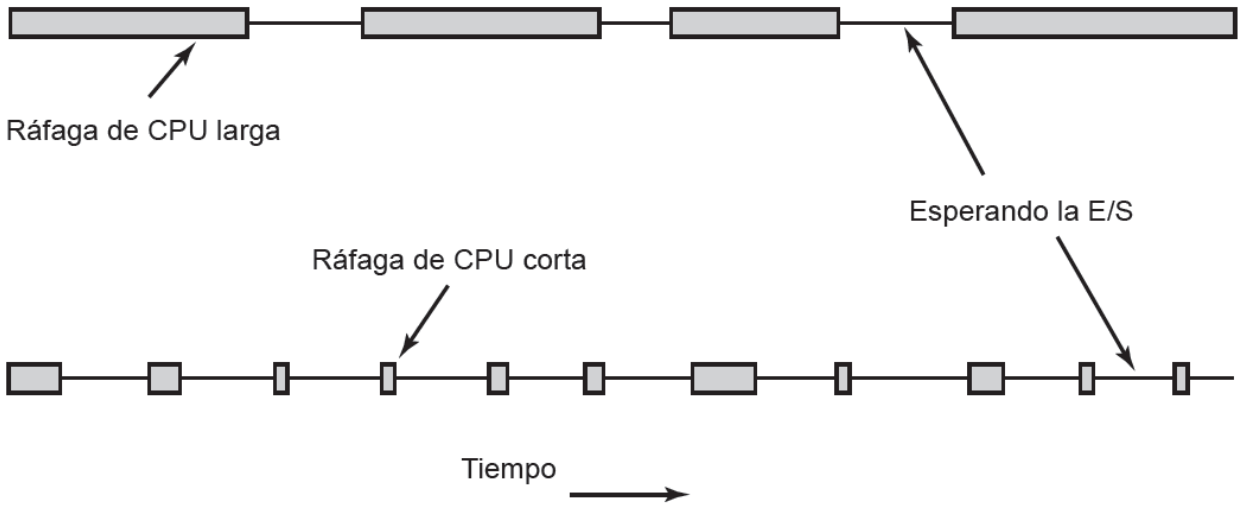
El algoritmo de planificación debe tener en cuenta una prioridad en los procesos para satisfacer mejor al usuario, y también que el alternar entre procesos es costoso para la CPU (involucra guardar el estado actual del proceso y cargar otro, además de que suele invalidar la caché).

Todo algoritmo de planificación debe perseguir algunas metas:

- Equidad: otorgar a cada proceso una parte justa de la CPU
- Aplicación de políticas: verificar que se lleven a cabo las políticas establecidas
- Balance: mantener ocupadas todas las partes del sistema

2.15.1. Comportamiento de un proceso

Un proceso puede ser limitado a cálculos (si invierte la mayor parte de su tiempo con la CPU) o limitado a E/S (I/O-bound) (si ocupa la mayor parte de su tiempo esperando la E/S).



La primera línea de tiempo corresponde a un proceso limitado a cálculos, y la segunda a uno limitado a E/S.

2.15.2. Llamadas al planificador

Ocurren automáticamente cuando:

- Al crearse un nuevo proceso, se elije si se ejecutará el padre o el hijo.
- Al terminar un proceso. Si no hubiera ningún otro proceso listo, se ejecuta uno inactivo provisto por el sistema.
- Al bloquearse un proceso. A veces, podría ser útil en estas situaciones saber el contexto para definir una prioridad particular, pero es común que el planificador no tenga acceso a este.
- Al ocurrir una interrupción de E/S.
- Al ocurrir una interrupción de reloj. Dado un sistema operativo que puede llamarlo en estas situaciones, se dice que su algoritmo de planificación es **apropiativo**. Esto implica que los procesos tienen un tiempo máximo de ejecución ininterrumpida, a diferencia de si fuera **no apropiativo**.

2.15.3. Planificación en sistemas de procesamiento por lotes

No es necesario limitar el tiempo de CPU por proceso, por lo que no hay problema con algoritmos de planificación no apropiativos (o apropiativos con largos periodos).

Metas adicionales:

- Rendimiento: maximizar el número de trabajos completos por hora
- Tiempo de retorno: minimizar el tiempo entre la entrega y la terminación
- Utilización de la CPU: mantenerla ocupada todo el tiempo

Primero en entrar, primero en ser atendido (First-Come, First-Served (FCFS))

Utiliza una cola, y es equitativo. Su mayor desventaja es que los procesos limitados a cálculos pueden ralentizar significativamente a los limitados a E/S.

El trabajo más corto primero

Prioriza los trabajos más cortos, algo que puede ser determinado estadísticamente. Así, optimiza el tiempo de retorno promedio. Es óptimo cuando todos los trabajos están disponibles a la vez.

El menor tiempo restante a continuación (Shortest Remaining Time Next (SRTN))

Similar a "El trabajo más corto primero", siempre elije ejecutar el trabajo que requiera menos tiempo para ser completado, bloqueando a uno en ejecución de ser necesario.

2.15.4. Planificación en sistemas interactivos

En sistemas interactivos (o servidores), sí es necesario contar con algoritmo apropiativo, pues no suele ser deseable que se ejecute un solo proceso por largos periodos.

Metas adicionales:

- Tiempo de respuesta: responder a las peticiones con rapidez
- Proporcionalidad: cumplir las expectativas de los usuarios

Planificación por turno circular (round-robin)

A cada proceso se le asigna un intervalo de tiempo máximo (llamado quántum) durante el cuál se puede ejecutar. Si un proceso excede su quántum, es bloqueado y se ejecuta al siguiente en una fila de procesos, colocando al proceso bloqueado al final de la fila.

La conmutación de procesos (cambio de proceso en ejecución) es muy costosa, por lo que un quántum muy chico es problemático, pero también lo es uno muy largo si se colocaran muchos procesos en la fila. Esto ocasionaría un largo tiempo de respuesta para los últimos procesos.

Planificación por prioridad

Se asigna una prioridad a los procesos, y se ejecuta el que tenga la mayor. Esto es útil cuando hay múltiples usuarios, y para manejar demonios.

Este método trae el problema de que el proceso de mayor prioridad podría ejecutarse indefinidamente. Para evitarlo, podría implementarse un quántum, o disminuir la prioridad del proceso en ejecución con cada interrupción de reloj.

Las prioridades pueden asignarse de forma estática según, por ejemplo, la importancia del usuario propietario. También pueden asignarse de forma dinámica, por ejemplo, para lograr un mejor balance (ocupar mejor todas las partes del sistema), otorgando una mayor prioridad a los procesos limitados a E/S. Para la asignación dinámica puede usarse un algoritmo como, por ejemplo, establecer la prioridad a $1/f$, donde f es la fracción del último quántum que usó un proceso.

Este método puede ser combinado con la planificación por turno circular, creando clases de prioridad que ejecuten un proceso al azar de los que pertenezcan a la clase de mayor prioridad. Requiere ajustar prioridades ocasionalmente para que todos los procesos se ejecuten.

Múltiples colas

Tiene distintas clases de prioridad, y asigna quántums más largos mientras de menor prioridad sea la clase. Así, hay distintos criterios para crear las clases y asignar prioridades.

Uno de estos criterios es el de CTSS, que asigna 1 quántum a la clase de mayor prioridad, 2 a la siguiente, 4 a la otra y así sucesivamente.

Otro es el de XDS 940, con cuatro clases: terminal, E/S, quántum corto y quántum largo (de mayor a menor prioridad). Este criterio busca favorecer a los procesos interactivos.

Además, se suele implementar un algoritmo que mueva procesos a clases de menor prioridad, para asegurarse de alternar el uso de la CPU; y también se puede implementar un algoritmo que los mueva a clases de mayor prioridad.

El proceso más corto a continuación

Similar a "El trabajo más corto primero" (de los sistemas de procesamiento por lotes). Sin embargo, esto requiere estimar el tiempo de ejecución requerido. Esto puede realizarse mediante un algoritmo como *envejecimiento*, donde se estima el tiempo de ejecución en base al tiempo que ocupó en ejecuciones anteriores, otorgando mayor importancia a las más recientes.

Planificación garantizada

Se establecen algunas reglas referidas a la proporción en que el poder de la CPU es distribuido, las cuales se intenta cumplir. Estas podrían ser que a cada usuario le corresponda una fracción del poder de la CPU, o que a cada proceso le corresponda una fracción de los ciclos de la CPU.

Planificación por sorteo

Se otorga a los procesos "boletos de lotería", y cada cierta cantidad de tiempo se realiza un sorteo. En estos, se selecciona algún boleto, y el proceso que lo tenga obtiene como premio el acceso a un recurso que requiere (como tiempo de la CPU).

Este método, de una forma sencilla, permite crear prioridades (otorgando más boletos) y una mejor cooperación entre procesos (intercambiando boletos).

Planificación por partes equitativa

Relacionada con la "planificación garantizada", pues toma en cuenta al propietario de un proceso para otorgar prioridades.

2.15.5. Planificación en sistemas de tiempo real

En sistemas de tiempo real, como se suele saber qué clase de proceso se ejecutará, puede no ser necesario un algoritmo apropiativo.

Metas adicionales:

- Cumplir con los plazos: para evitar perder datos
- Predictibilidad: evitar degradación en los sistemas multimedia

Suelen implementarse mediante muchos procesos cortos, y el planificador debe intentar organizar los procesos para cumplir con el tiempo límite siempre que detecte eventos externos.

Un sistema en tiempo real puede responder a eventos periódicos (que ocurren en intervalos regulares) o aperiódicos. Se dice que el sistema es planificable si cumple:

$$\sum_{i=0}^{m} \frac{C_i}{P_i} \leq 1$$

Donde:

- m: cantidad de eventos periódicos
- P_i: periodo del evento i
- C_i: tiempo de CPU requerido del evento i

Los algoritmos de planificación en tiempo real pueden ser estáticos (si hay información perfecta de antemano del trabajo a ejecutar, pues toman las decisiones antes de que el sistema se ejecute) o dinámicos (toman sus decisiones en tiempo de ejecución del sistema).

2.16 Política contra mecanismo (Tanenbaum, 2009, p. 161-162)

Un algoritmo de planificación puede separarse en política y mecanismo de planificación. El mecanismo siempre se encuentra en el kernel, y son los mencionados anteriormente. Sin embargo, el kernel por sí solo no tiene forma de saber si algún proceso debería tener más prioridad que otro, por lo que no es muy eficiente.

Esto puede mejorarse proveyendo llamadas al sistema que permitan que un proceso de usuario determine la política de planificación, es decir, la importancia de sus procesos hijos.

2.17 Planificación de Hilos (Tanenbaum, 2009, p. 162-163)

Depende del espacio en el que se encuentran.

2.17.1. Planificación de hilos en el espacio de usuario

El planificador de hilos se encuentra en el espacio de usuario, lo que permite un algoritmo personalizado potencialmente más eficiente para un problema en particular que un planificador en el kernel. Se pueden utilizar los métodos ya mencionados en general, aunque no hay quántums: un hilo dentro de un proceso podría ejecutarse de forma indefinida.

Los quátums son externos al proceso, es decir, el kernel puede decidir el tiempo máximo de un proceso, pero no de sus hilos.

2.17.2. Planificación de hilos en el kernel

Al involucrar llamadas al sistema, es considerablemente más lento, aunque puede ser optimizado al otorgar prioridad a hilos de un mismo proceso, para evitar cambiar el mapa de memoria e invalidar la caché. Sin embargo, permite mayor flexibilidad, pues no es necesario ejecutar una “ráfaga” de hilos pertenecientes a un mismo proceso.

La principal ventaja de este método es que puede bloquearse un hilo sin bloquear a todo su proceso.

2.18 Problemas de sincronización (Tanenbaum, 2009, p. 433-461) (Stallings, 2005, p. 203, 258-277)

2.18.1. Interbloqueo (deadlock)

Ocurre cuando, dado un conjunto de procesos, cada proceso está esperando un evento que sólo puede ser ocasionado por otro proceso del conjunto.

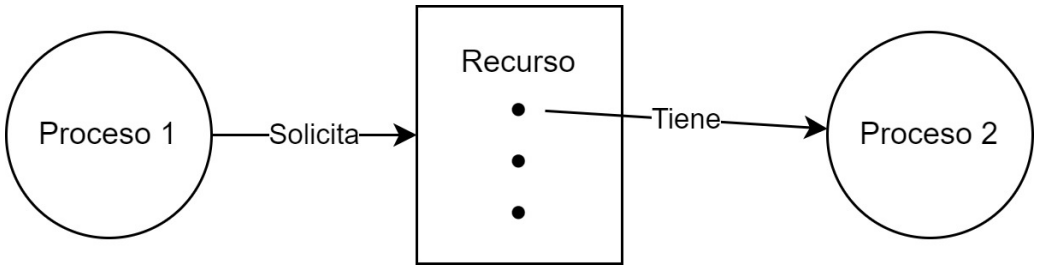
Si los eventos que esperan son liberaciones de recursos (dispositivos de hardware o piezas de información), específicamente del tipo **no apropiativo** (aquellos que no pueden ser quitados al proceso sin causar problemas), se conoce a los interbloqueos como **"de recursos"**.

Los interbloqueos de este tipo pueden darse tanto con recursos reutilizables (no se destruyen al terminar de ser usados) como consumibles (sí se destruyen, pero hay un proceso productor que los crea al ser necesitados).

Existe una convención para modelar las relaciones entre procesos y recursos mediante **grafos de asignación de recursos**, donde:

- los procesos se modelan con nodos circulares,
- los recursos se modelan con nodos rectangulares, y
- los ejemplares de recursos se modelan con puntos dentro de los nodos de recursos.

Así, una flecha de un proceso a un recurso significa que este lo solicita; y una flecha de un ejemplar de recurso a un proceso implica que el segundo lo posee.



De esta forma, podemos identificar a un interbloqueo al hallar un bucle.

Condiciones para que se de un interbloqueo:

- Exclusión mutua: un recurso solo debe poder asignarse a un proceso simultáneamente
- Contención y espera: capacidad de solicitar recursos cuando ya tiene otros previamente asignados
- Condición no apropiativa: no se puede quitar los recursos por la fuerza, el proceso debe liberarlos
- Espera circular: lista circular de dos o más procesos en que cada uno tiene uno o más recursos necesitados por el siguiente

Estrategias para la solución de interbloqueos:

- Ignorándolo (algoritmo de la avestruz)
- Detectándolo y recuperándolo:

  Detección:

  - Para interbloqueos con un recurso de cada tipo: por cada nodo, recorriendo el grafo formado, y terminando al llegar a un nodo ya analizado o recorrer todos los grafos. Los nodos son procesos que poseen y/o quieren recursos.
  - Para interbloqueos con varios recursos de cada tipo: basado en el marcaje de procesos que pueden ser finalizados con recursos potencialmente disponibles. Se busca un proceso que pueda ser finalizado con los recursos actuales:
    - Si se lo haya, se considera que este puede finalizar y que sus recursos pueden estar eventualmente disponibles. Se marca al proceso.
    - Si no se lo haya, el algoritmo termina.

    Al finalizar, los procesos sin marcar se encuentran en un interbloqueo.

  Recuperación:

  - Por apropiación
  - Por retroceso (si se realizan puntos de comprobación periódicamente, restaurando un proceso hasta un estado anterior en que no hubiera interbloqueo)
  - Por eliminación de procesos (ya sea dentro del ciclo o fuera, para liberar recursos)

- Prediciéndolo (evitándolo): distinguiendo estados seguros de inseguros (según si puede garantizarse que nunca llevará a un interbloqueo o no), y evitando asignar los recursos que un proceso requiere si esto llevaría a un estado inseguro.

  Esto se hace mediante el algoritmo del banquero, que es muy similar al algoritmo de detección para interbloqueos con varios recursos de cada tipo. La diferencia es que se activa antes, para no asignar recursos si esta asignación llevaría a un estado inseguro. Este algoritmo solo puede ser aplicado si se sabe cuántos recursos podría requerir un proceso como máximo, si no variara la cantidad de procesos, si no pudiera disminuir el número de recursos disponibles por motivos externos y si los procesos fueran independientes entre sí (su orden de ejecución no depende de otros).

  Una forma alternativa no óptima de implementar este método es impidiendo que un proceso se inicie si sus necesidades de recursos sumadas a las de los otros procesos activos superan a los recursos disponibles.

- Previniéndolo: removiendo alguna de las condiciones:

  - Exclusión mutua: inviable, pues requiere ser muy cuidadoso al asignar recursos para evitar un caos.
  - Contención y espera: puede realizarse solicitando que se asignen todos los recursos antes de comenzar la ejecución (pocas veces se sabe cuáles y cuántos se necesitará); o requiriendo que para solicitar un nuevo recurso se libere al resto, para luego solicitarlos nuevamente a la vez.

    - No apropiativa: quitar recursos es peligroso, pero si se los virtualiza (de modo que solo un demonio pueda acceder directamente a estos) se puede evitar interbloqueos relacionados a este recurso (aunque virtualizar consume espacio en disco, haciendo posible un interbloqueo aquí). No todos los recursos pueden ser virtualizados. Otra forma de solucionarlos es forzando a un proceso a liberar un recurso cuando otro proceso lo solicita; esto solo sirve para recursos cuyo estado puede ser salvado y restaurado.
    - Espera circular: forzando a que todo proceso pueda tener acceso a un solo recurso simultáneamente, o enumerando los recursos de forma que ningún proceso pueda adquirir recursos menores de los que ya contiene.

- Estrategia integrada: requiere especificar ciertas clases de recursos, utilizando la estrategia de prevención de espera circular por enumeración para evitar interbloqueos entre recursos de clases distintas. Por otro lado, para evitar interbloqueos entre recursos de una misma clase se puede elegir una estrategia que parezca apropiada.

Si bien no hay ningún algoritmo muy bueno en general, es cierto que existen algunos muy buenos en situaciones particulares. Un ejemplo es del de bloqueo de dos fases, usado en sistemas de bases de datos. Este algoritmo primero intenta bloquear todos los registros (recursos) que necesita, desbloqueándolos y comenzando de nuevo si alguno ya se encuentra bloqueado (primera fase), y actualizándolos y liberándolos al finalizar (segunda fase).

Un tipo distinto de interbloqueos son los **"de comunicación"**. En estos, el evento no es la liberación de recursos, sino la recepción de un mensaje. Un interbloqueo de comunicación puede ocurrir sencillamente por la pérdida de un mensaje. Suele poder solucionarse mediante *tiempos de espera*, es decir, reenviando un mensaje si no se ha recibido respuesta (acknowledgement) en un cierto lapso de tiempo. Esto también requiere un *protocolo*, para validar mensajes por si, por ejemplo, un mensaje se hubiera retrasado y no perdido (lo que podría causar la recepción doble del mismo mensaje).

2.18.2. Círculo vicioso / Bloqueo activo (livelock)

Ocurre cuando múltiples procesos cambian sus estados en respuestas a cambios en los estados de otros procesos, sin que ninguno avance realmente. Los procesos no dejan de ejecutarse (hasta que el planificador lo decida).

2.18.3. Inanición (starvation)

Ocurre cuando un proceso es capaz de avanzar, pero no lo hace por mal funcionamiento del algoritmo de planificación. En otras palabras, el planificador lo pospone por tiempo indefinido.

2.19 IPC: problemas clásicos de la comunicación entre procesos

(Tanenbaum, 2009, p. 163-168) (Stallings, 2005, p.241-245, 277-280, 758-763)

2.19.1. Problema de los filósofos comelones

Un filósofo ocupa su tiempo comiendo y pensando. 5 filósofos se sientan alrededor de una mesa redonda, con 5 platos de espagueti y 5 tenedores (cada plato tiene dos tenedores a sus costados). Los

filósofos no son italianos, así que necesitan dos tenedores para poder comer. Siempre que consiguen los dos, comen un poco y vuelven a pensar, liberando los tenedores. Se debe lograr un algoritmo que se ejecute bien y nunca se trabe.

Una solución es que cada filósofo, al querer comer, intente agarrar los tenedores y no los suelte hasta terminar de comer. Podría producirse un interbloqueo si, por ejemplo, todos los filósofos agarraran el tenedor izquierdo y no lo soltaran, pues ninguno comería.

Una modificación al algoritmo anterior podría ser que un filósofo, si agarra un tenedor pero no puede agarrar el otro, suelta el primero y espera un cierto tiempo para volver a intentarlo. Podría producirse una inanición si todos los filósofos empiezan a la vez, intentando comer a la vez y luego esperando mientras nadie come.

Podría arreglarse esto introduciendo aleatoriedad, volviendo estadísticamente improbable la inanición, pero esto puede no ser suficiente en un sistema crítico, que no puede fallar.

Podría utilizarse un semáforo binario, como se usó en el problema del productor-consumidor para coordinar los procesos. Esto sería confiable pero poco eficiente, porque solo un filósofo comería a la vez (cuando hasta dos podrían hacerlo con un mejor algoritmo).

Una mejora al anterior sería implementar un semáforo por filósofo y guardar un estado: comiendo, pensando o hambriento. Un filósofo solo podría comer si sus vecinos no lo están haciendo.

Este problema también puede ser solucionado con un monitor que posea dos procedimientos: uno para obtener y otro para liberar tenedores. Además, se requiere un arreglo de 5 valores booleanos que almacene la disponibilidad de tenedores, y 5 variables de condición (una por cada tenedor).

2.19.2. Problema de los lectores y escritores

Modela el acceso a una base de datos. Pueden haber muchos lectores y escritores. Múltiples lectores pueden acceder a ella a la vez, pero si un escritor está accediendo, nadie más (lector o escritor) puede hacerlo.

Una solución es implementar un semáforo binario. Al entrar un lector, se realiza una operación `down` para que bloquee a los escritores, pero se permite pasar a más lectores. Una vez que salga el último lector, realiza una operación `up`. El problema de esto es que, si hay muchos lectores, los escritores podrían nunca entrar.

La otra alternativa es que, si hay un escritor en espera, se suspenda a los lectores que lleguen tras él. Esto permite que los escritores esperen solo a los lectores que llegaron antes que ellos.

2.19.3. Problema de la barbería

Se requiere asegurar el correcto funcionamiento de una barbería con 3 barberos, 3 sillas (en las que los clientes se ubican al ser atendidos), una caja y un sillón con capacidad para 4 personas. La caja es administrada por un barbero.

Una solución puede implementarse con múltiples semáforos:

Semáforo	`wait`	`signal`
max_capacidad	Cliente espera a que haya espacio para entrar a la barbería	Cliente avisa cuando sale de la barbería
sillón	Cliente espera a que haya lugar en el sillón	Cliente avisa cuando lo van a atender (tras levantarse del sofá)
silla_barbero	Cliente espera a que se vacíe una silla	Barbero avisa cuando se vacía su silla
cliente_listo	Barbero espera a que un cliente se siente en su silla	Cliente avisa que se sentó
terminado[n]	Cliente espera a que su corte de pelo sea completado	Barbero avisa que terminó el corte
dejar_silla	Barbero espera a que el cliente se vaya de la silla	Cliente avisa que se ha levantado de la silla
pago	Cajero espera a que el cliente pague	Cliente avisa que pagó
recibo	Cliente espera para obtener un recibo por el pago	Cajero avisa que aceptó el pago
coord	Espera a que haya un barbero o cajero libre para realizar una de estas funciones	Avisa que un barbero o cajero se liberó
conseguir_numero	Cliente espera a que ningún otro cliente esté consiguiendo su número de cliente	Cliente avisa que ya obtuvo su número de cliente

Cabe mencionar que se añade el semáforo conseguir_número para lograr sincronización al terminar un barbero un corte, pues si terminado no fuera un arreglo de semáforos, al realizar up un barbero sobre este podría suceder que se eche a un cliente de su silla antes de terminar su corte, y se retenga a otro más tiempo del necesario. Al asignar un número a cada cliente, el barbero puede señalizar al semáforo terminado correspondiente.

Capítulo 3

Administración de memoria

3.1 Administrador de memoria (Tanenbaum, 2009, p. 175-176)

Parte del sistema operativo que asigna y libera memoria a los procesos, manteniendo un registro de las partes en uso (particularmente de la memoria principal). Debe **abstraer** la memoria.

3.2 Sin abstracción (Tanenbaum, 2009, p. 176-179)

Direccionamiento absoluto, es decir, los programas ven la memoria física. Hay varias formas de plantear esto, como:

- Sin multiprogramación. Si bien no causará problemas entre procesos de usuario, podría haber errores críticos si el sistema operativo (o parte de este) está en la RAM y un proceso lo modifica.

  Hay varias formas de ubicar el sistema operativo en memoria, algunas son:

  - Todo en RAM (posiciones bajas)
  - Todo en ROM (posiciones altas)
  - Controladores de dispositivos (BIOS) en ROM, resto en RAM

- Con hilos. Se supone que los hilos deben tener acceso a la misma memoria, pero es frecuente que el usuario quiera ejecutar muchos programas independientes en simultáneo.

- Con llaves. Utilizado por la IBM 360, consiste en dividir la memoria en bloques de tamaño fijo y asignarles una llave de 4 bits, la cual debe corresponder con una llave que tiene cada proceso en la PSW para que este puede utilizar el bloque.

  Sin embargo, al haber direccionamiento absoluto, un proceso no sabe cual es su ubicación en la memoria, y puede intentar (por ejemplo) saltar a un sector que no es suyo.

- Con llaves con reubicación estática (static relocation). Variación del anterior, en el cual a cada dirección de memoria del programa se le suma su dirección inicial durante la carga, para evitar referencias a sectores no propios. Funciona, pero es lento.

3.3 Abstracción con espacios de direcciones (Tanenbaum, 2009, p. 179-180)

Un espacio de direcciones es un conjunto de direcciones que usa un proceso para direccionar la memoria.

Para asignar espacios de direcciones a los procesos, se pueden usar distintos métodos: registros base y límite, intercambio y memoria virtual.

3.4 Registros base y límite (Tanenbaum, 2009, p. 180-181)

Almacenan la dirección inicial y la longitud de un programa, respectivamente. Utiliza reubicación dinámica, sumando el valor del registro base a toda referencia a memoria al momento de ejecutar la instrucción que la realiza. Es relativamente costoso, pues requiere una suma y comparación para evitar sobrepasar el límite.

3.5 Intercambio (Tanenbaum, 2009, p. 181-187)

Consiste en mover los procesos completos al disco cuando ya no se ejecutan, de modo que ya no ocupen memoria. Puede hacerse uso de reubicación estática o dinámica, mediante registros base y fuente.

Al intercambiar procesos, se va creando huecos en la memoria en los cuales no entran procesos, por lo que se los puede combinar en un solo gran hueco mediante **compactación de memoria**, moviendo a todos los procesos hacia abajo. Es muy costosa.

El espacio de direcciones de un proceso suele requerir crecer, por lo que se puede, por ejemplo, dejar huecos entre los procesos para ser ocupados de ser necesario. Si un proceso no tiene hueco para crecer, debe ser movido a un hueco más grande (o los procesos adyacentes, para crear un hueco). Si el proceso no puede crecer y el área de intercambio en el disco está llena, el proceso debe suspenderse hasta que se libere espacio o eliminarse.

Para que funcione este método, se debe poder distinguir entre memoria libre y ocupada. Esto puede lograrse con:

3.5.1. Intercambio con mapas de bits

Se crean n unidades de asignación (partes de la memoria) de m direcciones de longitud, y un mapa de bits con un bit por unidad de asignación (n bits). Cada bit representa el estado de su unidad de asignación.

Unidades de asignación pequeñas dejan huecos pequeños, pero requieren un gran mapa; y viceversa.

Para llevar un proceso a memoria, se debe buscar una secuencia de cierta cantidad de bits en el mapa que indiquen una unidad de asignación vacía. Esto es costoso.

3.5.2. Intercambio con listas enlazadas

Consiste en mantener lista(s) relacionad(as) mediante punteros con la información necesaria de los segmentos de memoria, tanto huecos como procesos. Requiere realizar fusión de huecos siem-

pre que termina un proceso para evitar fragmentación de memoria (tener muchos huecos pequeños adyacentes).

Tipos:

- Lista única: utiliza una lista de segmentos cuyos elementos especifican la dirección de inicio, la longitud y si se trata de un proceso o un hueco, además de un puntero.
- Doble lista: una para procesos y otra para huecos. Siempre es más veloz que una lista única.
- Doble lista optimizada: una para procesos y otra para huecos; esta segunda solo guarda su longitud y el puntero a la dirección del siguiente hueco, pues cada elemento de la lista se encuentra al inicio de su hueco correspondiente.
- Doble lista ordenada: una para procesos y otra para huecos, pero la segunda siempre es ordenada del hueco más pequeño al más grande.
- Múltiples listas: una lista para procesos, y una lista por cada tamaño frecuente de hueco requerido para ubicar procesos. Requiere una tabla de punteros a los primeros elementos de cada lista de huecos. Usada únicamente por el algoritmo de ajuste rápido.

Algoritmos para ubicar procesos:

- Primer ajuste: busca en la lista de segmentos (o huecos, si están separadas) desde el inicio hasta el primer hueco que acomode al proceso.
- Siguiente ajuste: primer ajuste, pero busca desde el último hueco en que se acomodó a un proceso. Es levemente menos rendidor que el algoritmo de primer ajuste.
- Mejor ajuste: busca el hueco más pequeño posible en el que entre el proceso, para evitar dividir huecos grandes. Es más lento que el algoritmo de primer ajuste (salvo en dobles listas ordenadas, donde son equivalentes) y suele causar más huecos pequeños inutilizables que los dos algoritmos anteriores.
- Peor ajuste: busca el hueco más grande, para que el hueco que queda siempre sea lo suficientemente grande para ser útil.
- Ajuste rápido: utiliza múltiples listas, busca en la lista de huecos del tamaño indicado, por lo que es la más veloz. Sin embargo, la fusión de huecos es también significativamente más costosa.

3.6 Memoria virtual (Tanenbaum, 2009, p. 188-234)

Virtualizar la memoria consiste en dividir la memoria que ocupa un proceso, con el objetivo de poder cargar en memoria un proceso (o varios) más grande que esta. Tener en cuenta que este método es para acceder a la memoria principal (física), no a memorias inferiores o superiores en la jerarquía.

A cada parte en que un proceso se divide se le llama “página”. También se debe dividir la memoria física en partes iguales capaces de contener páginas; por esto se les llama “marco de página”.

La memoria virtual es implementada por hardware mediante una **MMU** (*Memory Management Unit*), la cual mantiene una tabla de páginas (con una entrada por página).

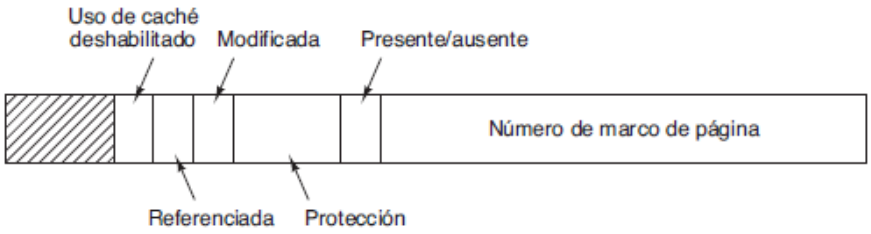
Lo que la memoria virtual logra es que los procesos se manejen con direcciones virtuales, pero que al ser procesadas y antes de mandarse a la memoria física, son traducidas en direcciones físicas por la MMU.

3.6.1. Memoria virtual a memoria física

Una dirección virtual consta de dos partes: número de página y desplazamiento. El número de página corresponde a un índice en la tabla de páginas; para obtener la dirección física se debe obtener el número de marco de página (que se halla en la entrada) y colocarlo delante del desplazamiento.

Ahora bien, es posible que la página cuyo número fue pasado en la dirección virtual no se encuentre actualmente en la memoria física (esto se sabe debido a un bit de presente/ausente en la entrada de la tabla de páginas). Cuando esto pasa, se ejecuta un `TRAP` correspondiente a un **fallo de página** y se otorga el control al sistema operativo para que este traiga la página necesaria desde la memoria secundaria a la principal.

3.6.2. Entradas en la tabla de procesos



- Número de marco de página: secuencia de bits utilizados para obtener la dirección física
- Presente/ausente: bit que indica si la página se encuentra o no en la memoria física
- Protección: bit o secuencia de bits que indica si la página puede ser, por ejemplo, leída y escrita o solo leída, o con formato rwx.
- Modificada/bit sucio (M): bit que indica si la página ha sido modificada; permite aumentar la eficiencia al traer otra página al marco que esta ocupa, pues al salir de la memoria principal sigue siendo una copia fiel de la memoria secundaria correspondiente, y no es necesario actualizar esta.
- Referenciada (R): bit que se establece siempre que una página es usada para lectura o escritura, utilizada para ayudar al sistema operativo a elegir una página para desalojar cuando es necesario.
- Uso de caché inhabilitado: bit que permite deshabilitar el uso de caché para la página, para situaciones en que la página está asociada a un registro de un dispositivo de E/S en lugar de a la memoria principal, pues se querrá que el hardware tenga acceso siempre a la palabra del dispositivo actualizada y no a una copia en la caché.

3.6.3. Aceleración de la paginación: búfers de traducción adelantada (TLB)

La tabla de páginas puede ser implementada de dos formas:

1. Con registros de hardware en la MMU: veloz pero pequeño (por ser costoso). Además, cada conmutación de contexto de los procesos se vuelve más lenta, al tener que cambiar también los valores de este arreglo de registros.

2. En la memoria principal: lento (por usar el doble de accesos a memoria) pero grande, requiere un solo registro en la MMU que apunte al inicio de la tabla de páginas, entonces las conmutaciones de contexto son más veloces.

Así como las cachés se usan para reducir el acceso a la memoria principal y aumentar la velocidad, se puede usar un dispositivo de hardware llamado TLB (*Translation Lookaside Buffer* / Búfer de Traducción Adelantada). Este es una memoria que guarda unas pocas entradas de la tabla de páginas (con el número de página que corresponda), idealmente aquellas de acceso frecuente, con el objetivo de evitar el acceso a memoria cuando la entrada esté en el TLB.

Al dar a la MMU una dirección virtual a traducir, el algoritmo es el siguiente:

1. Comparar la página virtual de la instrucción con aquellas presentes en el TLB
2. Si se encuentra allí, leer la entrada
3. Si no se encuentra allí, buscar la entrada en la tabla de páginas en la memoria principal y leerla
4. Si la página no se encuentra en memoria principal, traerla
5. Traducir la dirección

El TLB puede ser administrado por hardware o software (esto quiere decir dónde se implementará el algoritmo anterior). Cuando es administrado por software, el sistema operativo está a cargo de buscar páginas en memoria principal y modificar el TLB. El motivo de esto es simplificar la MMU, reduciendo su tamaño y permitiendo mejorar el rendimiento de la máquina al aprovechar este espacio.

La MMU es la encargada de buscar las entradas de la tabla de páginas en el TLB. Al ser administrado por software, la forma en que la MMU pasa control al sistema operativo al no encontrar una entrada es mediante un fallo de TLB.

Se puede mejorar el rendimiento de los procesadores que administran el TLB mediante software, ya sea reduciendo los fallos del TLB (al intentar adelantarse a múltiples operaciones de memoria en las mismas páginas) o reduciendo el costo de un fallo del TLB.

Se debe mantener una "caché" de entradas del TLB en software en una posición fija, cuya página siempre se encuentre en el TLB. De no ser así, podría darse la situación de que una entrada de una dirección virtual cualquiera no se encuentre en el TLB y, al intentar buscar la tabla de páginas, ocurra que esta tampoco esté en el TLB.

Un fallo suave ocurre cuando la página no está en el TLB, sino en memoria; y un fallo duro es aquel en que tampoco está en memoria, sino que se debe buscar en el disco.

3.6.4. Memorias extensas: tablas de páginas multinivel

Consiste en al menos una tabla de páginas cuyas entradas corresponden al número de marco de página de otras tablas de páginas. Esto permite tener unas pocas tablas en memoria, solo las necesarias. Requiere hacer una doble traducción de memoria virtual (si tiene dos niveles).

3.6.5. Memorias extensas: tablas de páginas invertidas

Consiste en una tabla de marcos de páginas, cuya cantidad es mucho menor que las páginas. Por esto ahorran mucho espacio, pero también son muy lentas, pues para encontrar una página en específico no se puede usar el número de página como índice. En su lugar, se debe buscar el marco de

página que haga referencia a la página buscada. Además, a menos que se use el TLB, esta búsqueda debe ocurrir siempre, no solo en los fallos de TLB.

Una forma de acelerar la búsqueda de una página es mediante una tabla que, a cada página en memoria, le haga corresponder el marco de página en el que se encuentra. A este tipo de tabla se le llama tabla de hash.

3.6.6. Algoritmos de reemplazo de páginas

Óptimo

Se etiqueta cada página con la cantidad de instrucciones que se ejecutarán antes de que se referencie a dicha página, y al realizar un cambio se selecciona aquella cuyo valor de la etiqueta sea mayor.

Es imposible de implementar: no se puede saber cuántas instrucciones se tardará en hacer la próxima referencia. Lo que sí es posible es realizar una primera corrida para estimar estos valores (pero solo sirve para un programa con ciertos datos de entrada).

No usadas recientemente (NRU)

Utiliza los bits de Referenciada y Modificada de las entradas en la tabla de procesos. Establece una jerarquía en base a estos para decidir qué página sacar de la memoria física.

La primera clase en la que se buscará una página al azar para sacar será la 0; en caso de no haber ninguna, la 1, y así sucesivamente.

Clase	Referenciado	Modificado
0	0	0
1	0	1
2	1	0
3	1	1

Primera en entrar, primera en salir (FIFO)

Mantiene una lista enlazada con las entradas de la tabla de páginas, de forma que al producirse un fallo de página se elimina la última y se agrega una al inicio. El problema es que podría eliminarse una entrada importante a lo largo del tiempo, que requerirá pronto ser buscada nuevamente.

Segunda oportunidad

Variación de FIFO. Si ocurre un fallo de página, se comprueba el último elemento de la cola. Si este no ha sido referenciado desde su carga (R = 0), se lo elimina; caso contrario (R = 1), se mueve el elemento al principio de la cola y se coloca R en 0 (renovándolo).

Reloj

Implementación más eficiente de Segunda oportunidad, utiliza una lista circular de los marcos de página y un puntero (manecilla) al que tiene la página más antigua. El funcionamiento con R es igual al del algoritmo anterior.

Menos usadas recientemente (LRU)

Consiste en eliminar la página que no se haya usado durante la mayor cantidad de tiempo. Hay varias variantes y algoritmos similares:

- **LRU mediante una lista enlazada:** con cada página, las más recientemente referenciadas al frente. Debe ser actualizada con cada referencia, buscando y llevando a la página correspondiente al frente, algo muy costoso.
- **LRU con un contador en hardware:** hay un contador C que se incrementa con cada instrucción automáticamente. Su valor debe guardarse en un campo de la entrada correspondiente a una página de la tabla cada vez que esta sea referenciada. Al requerir quitar una página de la memoria física, se eliminará aquella de menor valor en el campo mencionado.
- **LRU mediante una matriz por hardware:** si hay n páginas, la matriz debe ser de n x n bits. En las filas, se guarda un valor binario por cada página, entre los cuales el menor indica aquella que lleva más tiempo sin ser usada.

  Siempre que se referencia una página k:

  - se aumenta el valor de su fila a su máximo posible (colocando todos los bits salvo el k-ésimo en 1)
  - Se disminuye el valor de las otras filas, colocando su k-ésimo bit (columna k) en 0.
- **Not Frequently Used (NFU):** en cada interrupción de reloj, se le suma a un contador en cada entrada de la tabla de páginas el bit R. El problema es que los contadores no olvidan: si al principio se referenció mucho a una página, más adelante (cuando ya no esté en uso) su valor podría seguir siendo más alto que el de otras páginas útiles, provocando una preferencia para quitar estas páginas en uso por sobre la otra.
- **Envejecimiento (aging):** variante de NFU; en lugar de sumar R, desplaza el "contador" un bit a la derecha y coloca a R a la izquierda. Así, mientras más recientemente se haya hecho la referencia, se garantiza que mayor será el valor del contador, pero si el contador tiene n bits, luego de n pulsos de reloj las referencias pasadas serán eliminadas (evitando el problema de NFU).

Conjunto de trabajo:

Se basa en la agrupación de las páginas de uso frecuente en un momento dado, pues un proceso suele usar solo una pequeña parte de sus páginas durante cierto periodo (localidad de referencia). A este grupo se le llama **conjunto de trabajo**.

Sobrepaginación (*thrashing*) es la situación en que ocurren muchos fallos de página consecutivos en un corto periodo de tiempo. Es indeseable por su lentitud y, si no se tiene en cuenta, ocurre cada vez que se carga un proceso desde el disco. Esto puede solucionarse identificando el conjunto de trabajo y cargándolo automáticamente al traer un proceso. Esto es la *prepaginación*.

Para identificar las páginas pertenecientes al conjunto de trabajo, se debe tener en cuenta:

- un número k de referencias a memoria (o páginas) más recientes (inviable, requiere un enorme e ineficiente registro de desplazamiento por cada referencia), o

- un número τ de milisegundos de tiempo virtual durante el cual una página no debe haber sido usada

La segunda opción es posible de implementar. Requiere un campo adicional en las entradas de la tabla de páginas que almacene el tiempo en el que la página fue referenciada por última vez, además del bit R (colocado en 0 en cada interrupción de reloj).

Al ocurrir un fallo de página, se recorre la tabla. Por cada entrada:

1. Si R es 1, colocar el tiempo de última referencia como el *tiempo virtual actual* (almacenado en la tabla de procesos).

2. Si R es 0, se calcula la edad (tiempo virtual actual - tiempo de última referencia) y:

   a) Si su edad es menor a τ, se la considera parte del grupo de trabajo y se la reserva temporalmente. Hay un puntero a la entrada de la página en este grupo de mayor edad, el cual es actualizado de ser necesario.

   b) Si su edad es mayor a τ, ya no está en el grupo de trabajo y es reemplazada.

3. Si no se encontrara ninguna página de edad mayor a τ, se elimina a aquella de mayor edad (mediante el puntero).

WSClock

Es una combinación del algoritmo del reloj con el conjunto de trabajo. Al ocurrir un fallo de página, se comprueba R, τ y M.

Referenciada (R)	0				1	
Edad	$< \tau$		$> \tau$		$< \tau$	
Modificada (M)	0	1	0	1	0	1
Asignar 0 a R	No	No	No	No	Sí	Sí
Cargar página	No	No	Sí	No	No	No
Planificar escritura	No	No	No	Sí	No	No
Siguiente página	Sí	Sí	No	Sí	Sí	Sí

En un primer recorrido, se intenta buscar una página no referenciada, sin modificar y de edad mayor a τ, por ser estas las páginas ideales a sacar de la memoria física. Además, se planifica la escritura al disco de aquellas similares a las anteriores pero modificadas, para no realizar una conmutación de procesos y seguir buscando una mejor alternativa.

Luego del primer recorrido, pueden darse dos situaciones:

- Se ha planificado al menos una escritura: buscar una página "ideal" para quitar de la memoria física. Tarde o temprano, una de las planificadas será escrita al disco y su M valdrá 0.

- No se han planificado escrituras: se debe seleccionar una página al azar (preferentemente con M = 0) para sobrescribir, pues todas las páginas están en el conjunto de trabajo.

3.6.7. Políticas de asignación

Las hay locales y globales, según donde se busquen marcos de página para sobrescribir al ocurrir un fallo de página.

Las políticas globales suelen ser más convenientes, pues es común que las locales provoquen sobrepaginación aún cuando haya múltiples marcos disponibles. Además, si el conjunto de trabajo se reduce, se desperdicia memoria.

Una forma de solucionar estos problemas es repartiendo marcos de página entre los procesos. Podría hacerse equitativamente, o de forma proporcional al tamaño total de los procesos.

Con políticas globales, también se requiere distribuir los marcos de página. Una forma conveniente de decidir cuándo asignar o quitar marcos es el algoritmo *Page Fault Frequency* (PFF), que mide la proporción de fallos de los procesos en función del tiempo e identifica a aquellos que no se encuentren en un cierto rango. El algoritmo no decide qué página sustituir en un fallo.

No todos los algoritmos pueden funcionar con cualquier política de asignación. FIFO, LRU y sus variantes funcionan tanto local como globalmente, pero el resto (en especial la de conjuntos de trabajo y WSClock) solo sirven de forma local.

3.6.8. Control de carga

Incluso con políticas de asignación globales, puede ocurrir sobrepaginación. Para solucionar esto, se debe intercambiar los procesos, llevándolos al disco.

Para decidir cuál intercambiar, se debe tener en cuenta el tamaño del proceso, la proporción de páginas que tiene y el grado de multiprogramación (este último para evitar tiempos ociosos de la CPU).

3.6.9. Tamaño de página

Hay varios factores para determinar el tamaño más conveniente:

- Fragmentación interna: desperdicio de espacio de la última página (en promedio, la mitad de esta). Hay menor fragmentación interna con tamaños de página menores.
- Partes no utilizadas en memoria: si un proceso requiere una pequeña porción suya para ejecutarse, un tamaño de página grande obliga a cargar una gran parte de este sin necesidad.
- Tamaño de la tabla de páginas: a menor tamaño de página, mayor tamaño de la tabla. En algunas máquinas, al realizar la conmutación de procesos se debe cargar la tabla en registros del hardware, por lo que si esta es muy grande, la conmutación tardará más.
- Transferencias con el disco: el retraso al llevar o traer una página del disco no varía mucho según el tamaño de página, por lo que llevar la misma cantidad de memoria en múltiples páginas será considerablemente más lento.

3.6.10. Espacios separados de instrucciones y datos

Algunas arquitecturas definen espacios de direcciones “I” y “D” independientes para instrucciones y datos, cada una paginada con su propia tabla de páginas.

3.6.11. Compartición

Compartir las páginas con instrucciones de un programa no suele suponer complicaciones, pues al ser de solo lectura no se necesita aplicar exclusión mutua.

Por otro lado, compartir las páginas con los datos solo debería hacerse mientras estos sean, como las instrucciones, de solo lectura. En el momento en que se los intente modificar, se emite un `TRAP` que ordena al sistema operativo copiar la página correspondiente y asignarle tanto a la original como a la copia permiso de escritura (bits de protección en las entradas de la tabla de páginas). A esto se le llama "copiar en escritura".

Ambas posibles comparticiones se simplifican mucho si hay espacios de instrucciones y datos, aunque esto requiere dos punteros por BCP en la tabla de procesos, una por cada tabla de páginas.

Similar a las instrucciones de un programa, existen las **bibliotecas compartidas** (Dynamically Linked Library, Biblioteca de Enlaces Dinámicoss (DLLs)). Sus ventajas son:

- Menor tamaño de los ejecutables
- Se cargan una vez, aunque muchos programas las estén usando
- Actualización automática, no es necesario recompilar programas cuando las bibliotecas son actualizadas

Cada programa ve las bibliotecas compartidas como si se encontraran en su espacio de memoria, por lo que no se puede hacer direccionamiento absoluto (¿cómo sabría la biblioteca a la dirección de qué programa se refiere la instrucción?), sino que se debe utilizar código independiente de la posición, es decir, con direccionamiento relativo (`JUMP 4` lugares hacia adelante).

Las bibliotecas compartidas se implementan mediante **archivos asociados a memoria**: archivos que son accedidos como si fueran parte de la memoria virtual de un proceso (solicitando esto al sistema operativo con una llamada al sistema). Además, los archivos asociados permiten obtener un canal de comunicación de gran ancho de banda entre procesos.

3.6.12. Política de limpieza

Es común la existencia de un demonio de paginación que periódicamente copie las páginas con M = 1 al disco y colocando M en 0, para que cuando haya un fallo de página no se deba realizar el movimiento de urgencia.

Además, puede incluso desalojar páginas para acelerar la búsqueda de marcos de página libres en los fallos de página.

3.6.13. Interfaz de memoria virtual

Básicamente, lo que los procesos y programadores ven es un espacio de direcciones virtuales grande en una computadora de memoria física más pequeña. Sin embargo, algunos sistemas proporcionan mayor control del mapa de memoria, lo que posibilita los archivos asociados, sistemas de mensajes veloces que no requieren ser copiados y la memoria compartida distribuida (entre distintas máquinas, enviándose páginas).

3.6.14. Participación del sistema operativo en la paginación

Al crear un proceso

Se debe crear e inicializar la tabla de páginas, asignándole memoria. También se debe asignar espacio en el área de intercambio del disco, donde se guardan las páginas no cargadas. Se debe guardar la información de la tabla de páginas y el área de intercambio en la tabla de procesos.

Al ejecutar un proceso

Se debe establecer la MMU y vaciar el TLB para deshacerse de las páginas del proceso anterior, y también actualizarse la tabla de páginas. Puede o no hacerse prepaginación.

Al ocurrir un fallo de página

1. Determinar dirección virtual que produjo el fallo (leyendo registros de hardware)
2. Determinar página necesaria y localizarla en el disco
3. Buscar un marco de página y liberarlo de ser necesario, copiando su contenido de su página al disco si esta fue modificada
4. Copiar la página necesaria al marco de página
5. Restaurar registro contador de programa (PC)

Al terminar un proceso

Se debe liberar su tabla de páginas y el espacio que las páginas ocupan cuando están en disco, siempre y cuando no estén compartidas.

3.6.15. Manejo de fallos de página

1. El hardware guarda PC en la pila y, comúnmente, cierta información del estado de la instrucción actual en registros especiales de la CPU. `TRAP` al kernel.
2. Una rutina guarda los registros generales e información volátil. Llama al sistema operativo.
3. El sistema operativo determina la página necesaria, ya sea por un registro especial o analizando por software la instrucción que produjo el fallo.
4. El sistema operativo comprueba si la dirección es válida y no viola la protección, enviándole una señal al proceso o eliminándolo de no ser así. De otro modo, el sistema selecciona un marco de página, primero buscando alguno disponible y si no lo hay, ejecutando un algoritmo de reemplazo de páginas.
5. Si el marco está sucio, se planifica la transferencia de la página que contiene al disco y se realiza una conmutación de contexto, dejando que otros procesos se ejecuten. Se marca como "ocupado" al marco de página.

6. Cuando el marco de página esté limpio, el sistema operativo busca la página necesaria en el disco y planifica la transferencia de esta al marco de página en memoria física. Se deja a otros procesos ejecutarse mientras esto ocurre.

7. El disco emite una interrupción al terminarse la transferencia, y el sistema operativo entonces actualiza las tablas de página y se marca como "normal" al marco de página.

8. Se respalda el estado de la instrucción fallida y se restablece PC.

9. Se planifica el proceso fallido y el sistema operativo regresa el control a la rutina que lo llamó.

10. La rutina recarga los registros y demás información de estado, regresando al espacio de usuario para seguir con la ejecución.

3.6.16. Respaldo de instrucción (fallida)

Instrucciones con múltiples referencias a memoria

Se considera así a las instrucciones que ocupan múltiples palabras. Estas, deben ir aumentando el contador de programa a medida que se ejecutan, por lo que al ocurrir un fallo de página no basta con guardar el PC actual, sino que también se debe guardar el PC en que inician las instrucciones. Esto pueden solucionarlo los diseñadores de CPUs agregando un registro interno oculto.

Instrucciones con autoincremento/autodecremento

Según las microinstrucciones que usa una instrucción, puede incrementar o decrementar uno o más registros antes o después de un potencial fallo de página. Si ocurren luego no hay ningún problema; pero si son previos al fallo, se deben deshacer los cambios que hayan hecho al continuar la ejecución luego de traer la página a memoria. También puede solucionarse mediante registros que indiquen cuáles registros han sido incrementados y en cuánto.

3.6.17. E/S y reemplazo de páginas

Si un proceso inicia una transferencia de E/S mediante DMA, luego se conmuta a otro proceso y este tiene un fallo de página en que se decida mover la página encargada de recibir los datos de la transferencia al disco, parte de estos datos serán escritos a la página correspondiente pero el resto quedarán en la nueva página que ocupa el marco.

Esto puede solucionarse mediante bloqueo de páginas en memoria o **fijación**, para evitar que esta sea eliminada. Otra opción es forzar que todas las transferencias de E/S se realicen a búfers del núcleo, y que este luego copie los datos a la página correspondiente.

3.6.18. Almacén de respaldo

Se refiere a la forma en que las páginas se almacenan en el disco: en una partición *de intercambio* especial en el disco o en un disco separado del sistema operativo. Esta partición (o disco separado) se administra como una lista de trozos libres.

Al comenzar a ejecutarse un proceso, se debe inicializar el área de intercambio, ya sea:

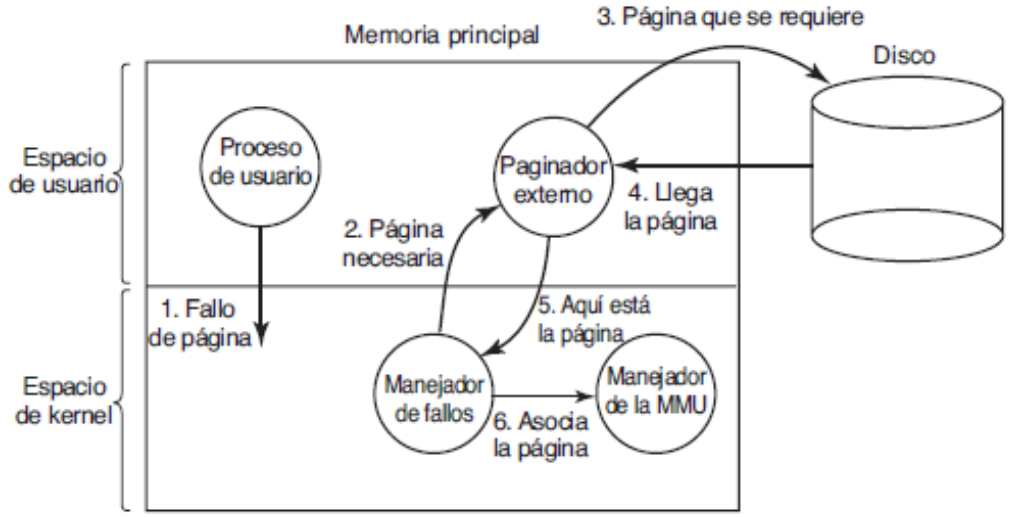
- Copiando la imagen del proceso al área de intercambio para que pueda llevarse a memoria cuando sea necesario (aquí puede ser útil designar áreas separadas para el texto, datos y pila, dado que el tamaño de los dos últimos cambia, y puede también realizarse una optimización si se designa como área de texto al archivo ejecutable, pues este no cambia). Las direcciones de inicio de las áreas de intercambio se guardan en el BCP de cada proceso.

- Cargando el proceso en memoria, y dejando que se pagine *hacia afuera*, escribiéndose al disco al realizar intercambios. Además, esto da la posibilidad de ahorrar espacio del área de intercambio si también se liberan las páginas del disco al escribirlas a la memoria principal. Las desventajas son que no se aprovecha el bit M y que hace falta una tabla por proceso que almacene la ubicación de cada página en el disco.

3.6.19. Política contra mecanismo

Para conseguir una organización más modular del código, se lo puede dividir entre los espacios de núcleo y de usuario. El mecanismo es aquel implementado en el núcleo (manejador de fallos), mientras que la política es un proceso de usuario (paginador externo).

Entonces, los fallos de página pueden ser administrados de la forma indicada en el gráfico:



Una complicación es decidir dónde se ejecuta el algoritmo de reemplazo de páginas:

- En el paginador externo no se tiene acceso a los bits R y M, por lo que probablemente haga falta pasarlos de alguna forma si el algoritmo estuviera aquí.

- En el manejador de fallos, se debe indicar al paginador externo la página a desalojar (pues este es el que interactúa con el disco).

Al separar política y mecanismo se obtiene un código modular más flexible, pero hace menos eficiente al sistema al requerir más cambios de modo.

3.7 Segmentación (Tanenbaum, 2009, 234-247)

Consiste en dividir la memoria (física o virtual) en partes llamadas “segmentos”, espacios de direcciones independientes, con las ventajas de:

- Facilitar la expansión y compresión de las partes de un proceso sin afectar a las otras.
- Optimiza la compilación, pues al cada segmento tener su propio espacio de direcciones, una actualización en uno de ellos no requiere recompilar el resto. Esto es útil para las bibliotecas compartidas.
- Permitir el establecimiento de protecciones para distintos segmentos, evitando por ejemplo que se escriba a un segmento de texto, o que se ejecute un segmento con un arreglo o una pila.

Todo esto es posible gracias a que, a diferencia de la paginación, la segmentación es manipulada por los programadores: los segmentos son unidades lógicas de las cuales el programador está consciente.

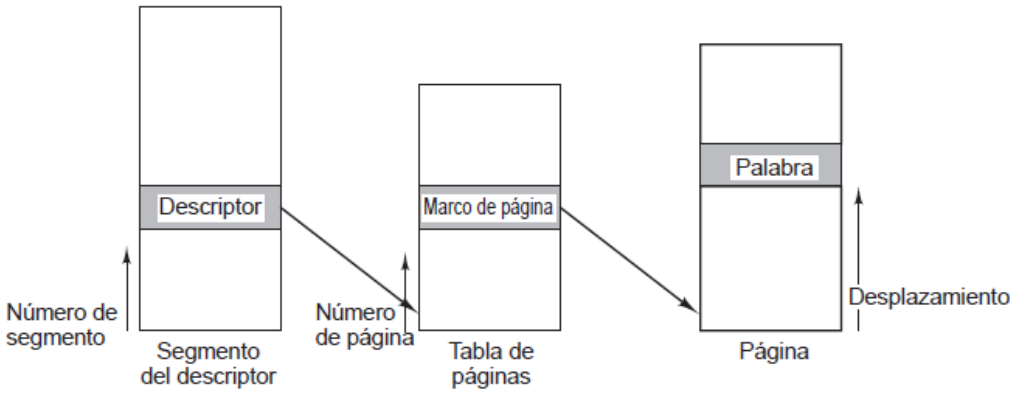
3.7.1. Segmentación pura

La memoria se divide en segmentos de distintos tamaños, que van siendo reemplazados por otros conforme pasa el tiempo. Para evitar fragmentación externa (división de la memoria en muchos trozos, algunos ocupados y otros no) se debe compactar periódicamente a la memoria (como en el intercambio de procesos).

3.7.2. Segmentación con paginación

Consiste en dividir a la memoria en segmentos, y a los segmentos en páginas. Por esto, las direcciones (o selectores) deben consistir de tres partes: un número de segmento, un número de página y un desplazamiento en esta página. En consecuencia, cada proceso tiene una tabla de segmentos, cuyas entradas se denominan *descriptores de segmento*, y cada descriptor de segmento tiene un puntero a su tabla de páginas.

La imagen que sigue describe la conversión de una dirección del sistema operativo MULTICS en una dirección de la memoria principal:



Esta organización requiere una mayor complejidad en el TLB, pues además de la página, debe almacenar el segmento. En el sistema operativo MULTICS el TLB es aún más complejo, pues soporta más de un tamaño de página.

En algunos sistemas (como Intel Pentium) hay, además de una tabla de segmentos por proceso (LDT, *Local Descriptor Table*), una tabla de segmentos global (GDT, *Global Descriptor table*) que almacena los segmentos del sistema. El Intel Pentium también soporta desactivar la segmentación y la paginación.

La protección suele implementarse por segmento. En MULTICS e Intel Pentium hay ciertos niveles, y un programa en cierto segmento en un nivel no puede acceder a segmentos en niveles menores. Hay una excepción a esto: una instrucción `CALL`, que utiliza un selector para identificar un descriptor llamado **compuerta de llamada** que contiene la dirección de un procedimiento de un nivel inferior. Esto es para evitar saltos a direcciones arbitrarias. Los `TRAP`s e interrupciones funcionan análogamente.

Capítulo 4

Archivos

4.1 Almacenamiento de información a largo plazo (Tanenbaum, 2009, p. 255-256)

Hay tres requerimientos para considerar que se tiene un almacenamiento de información a largo plazo:

- Gran capacidad para información
- Persistencia de información tras terminación de procesos de la utilicen
- Posible acceso concurrente por múltiples procesos

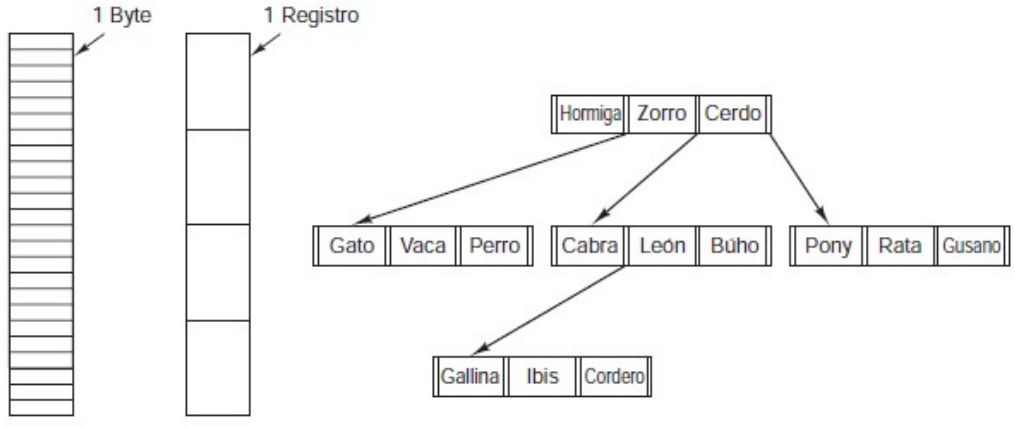
Este almacenamiento se refiere a la memoria secundaria, y suele involucrar discos (organizados linealmente en bloques de datos de tamaño fijo, con operaciones para leer o escribir un bloque de cierto índice).

El sistema operativo crea una interfaz más conveniente para que los procesos de usuario puedan manejar el almacenamiento a largo plazo: el **sistema de archivos**.

Un archivo es una unidad lógica de información creada por un proceso.

Los archivos tienen nombres. Los sistemas operativos determinan qué caracteres pueden contener los nombres. Los nombres suelen separarse en dos o más partes que, dependiendo del sistema operativo, pueden tener un significado para este o solo para el usuario. Una división común es "nombre.extensión".

4.2 Estructuras de archivos (Tanenbaum, 2009, p. 259-260)



Hay tres tipos de estructura interna de un archivo:

4.2.1. Secuencia de bytes

No tiene estructura alguna, es el tipo más flexible.

4.2.2. Secuencia de registros

Múltiples registros de tamaño fijo que pueden ser accedidos mediante un índice.

4.2.3. Árbol de registros

Cada registro (de tamaño variable) tiene una llave que debe ser proporcionada para acceder al mismo, aunque también puede accederse a los registros secuencialmente.

4.3 Tipos de archivos (Tanenbaum, 2009, p. 260-262)

4.3.1. Regulares

Archivos regulares

Contienen información del usuario.
Los hay:

- **ASCII (de texto)**, que pueden ser mostrados como están, editados e impresos con sencillez. Además, facilitan la canalización de programas (un programa usa como entrada la salida de otro). Hay distintas convenciones respecto a cómo debería terminar una línea: con un avance de línea/line feed (LF), con un retorno de carro/carriage return (CR) o con ambos (CRLF).

- **Binario**, aquellos que no son de texto. Suelen tener una estructura interna y ser usados por programas específicos, pero su contenido no podrá mostrarse directamente (como los archivos ASCII). Los sistemas operativos suelen reconocer al menos un tipo: los **ejecutables**.

Directorios regulares

Sistemas de archivos que permiten mantener la estructura.

4.3.2. Especiales

Tienen un propósito particular transparente al usuario. Algunos ejemplos podrían ser, en UNIX:

- De caracteres: para modelar dispositivos de E/S en serie
- De bloques: para modelar discos

4.4 Acceso a archivos (Tanenbaum, 2009, p. 262)

Se refiere a la lectura de datos de estos. Puede ser secuencial o de acceso aleatorio (indicando la posición). Comúnmente se usa una combinación de ambas, con las operaciones `seek` (colocar la posición a partir de la que se comenzará a leer) y `read` (leer).

4.5 Atributos de archivos (Tanenbaum, 2009, p. 263-264)

Los atributos o metadatos son campos con información adicional acerca de cada archivo que guardan los sistemas operativos.

Algunos aspectos a los que suelen relacionarse estos atributos son la protección, creador y propietario, tamaño de registro, tamaño de llave y posición de las llaves, horas de creación, últimos acceso y modificación, tamaños actual y máximo.

4.6 Operaciones de archivos (Tanenbaum, 2009, p. 264-265)

Típicamente, son:

- `Create`: sin datos, avisa de la creación y establece algunos atributos
- `Delete`: para liberar el espacio en disco
- `Open`: trae el archivo del disco a la memoria principal
- `Close`: escribe el último bloque del archivo de la memoria principal al disco (los bloques anteriores ya habían sido escritos)
- `Read`: lee datos del archivo a un búfer desde la posición actual, se debe indicar la cantidad de datos leída
- `Write`: escribe datos al archivo en la posición actual; si esta es en medio del archivo, sobrescribe el contenido, pero si es al final, aumenta el tamaño del archivo
- `Append`: `Write` que siempre escribe al final
- `Seek`: coloca el apuntador del archivo en cierta posición especificada

- `Get attributes`: devuelve el valor de los atributos
- `Set attributes`: establece el valor de ciertos atributos
- `Rename`: cambia el nombre del archivo

4.7 Sistemas de directorios (Tanenbaum, 2009, p. 268-272)

4.7.1. Sistemas de directorios de un nivel

Todos los archivos se encuentran en un único directorio. Es útil únicamente para sistemas simples.

4.7.2. Sistemas de directorios jerárquicos

Mediante un árbol de directorios. Si los directorios son considerados como archivos por un sistema, un directorio puede contener tanto directorios como otros archivos.

Estos sistemas requieren *nombres de ruta* para ubicar archivos y directorios. Los nombres de ruta consisten en una secuencia de nombres de directorios separadas por un caracter especial que el sistema operativo establece ("\" en Windows, "/" en UNIX, ">" en MULTICS). El último nombre puede ser un archivo regular.

Hay dos tipos de nombres de ruta:

- Absolutas: comienzan con el caracter especial, y se debe comenzar la búsqueda por el directorio raíz.
- Relativas: no comienzan con el caracter especial, buscan a partir del directorio de trabajo

Además, muchos sistemas operativos proveen unas entradas especiales para ser usadas en lugar de los nombres de directorios o archivos:

- ".": hace referencia al directorio de trabajo
- "..": hace referencia al directorio padre del directorio de trabajo (a excepción de la raíz, que hace referencia a sí misma).

4.8 Operaciones de directorios (Tanenbaum, 2009, p. 272-273, 283-285)

Algunas operaciones comunes son:

- `Create`: crea un directorio vacío, salvo por "." y ".."
- `Delete`: elimina un directorio siempre y cuando esté vacío (que solo contiene a "." y "..")
- `Opendir`: abre un directorio (para poder leer su contenido)
- `Closedir`: cierra un directorio para liberar espacio
- `Readdir`: devuelve la siguiente entrada en un directorio abierto

- `Rename`: cambia el nombre de un directorio (a veces es la misma operación que con los archivos)

- `Link`: vincula (vínculo o liga dura) un archivo a un directorio. La vinculación permite que un mismo archivo aparezca en múltiples directorios, con múltiples rutas. Es usado en UNIX con los Nodos-I

- `Unlink`: desvincula un archivo de un directorio, eliminando su entrada. Si esta era la única referencia al archivo, también se lo elimina definitivamente

Existen también los vínculos o ligas simbólicos: un archivo que apunta a otro. Para modificar el archivo apuntado, primero se tiene que buscar la en la ruta del primero, leerlo y buscar en la ruta que este contenía al segundo. Es menos eficiente que el vínculo duro, pero a diferencia de este puede nombrar archivos en computadoras remotas.

4.9 Distribución del sistema de archivos (Tanenbaum, 2009, p. 273-274)

Los discos se dividen en particiones, cada partición correspondiendo al sistema de archivos de un sistema operativo.

En el sector 0 de un disco, se encuentra el Registro Maestro de Arranque (MBR, Master Boot Record). Al final de este se halla una tabla de particiones, que proporciona información respecto al inicio y fin de cada una, y se marca a una partición como activa.

Cada partición, a su vez, se divide en ciertos bloques. Esta distribución varía de un sistema de archivos a otro, pero suelen tener algunos bloques en común (sin ningún orden en particular, a excepción del bloque de arranque):

- Bloque de arranque: contiene un programa que carga el sistema operativo contenido en la partición. Al arrancar la máquina, se carga y ejecuta el bloque de arranque de la partición activa. Siempre está primero.

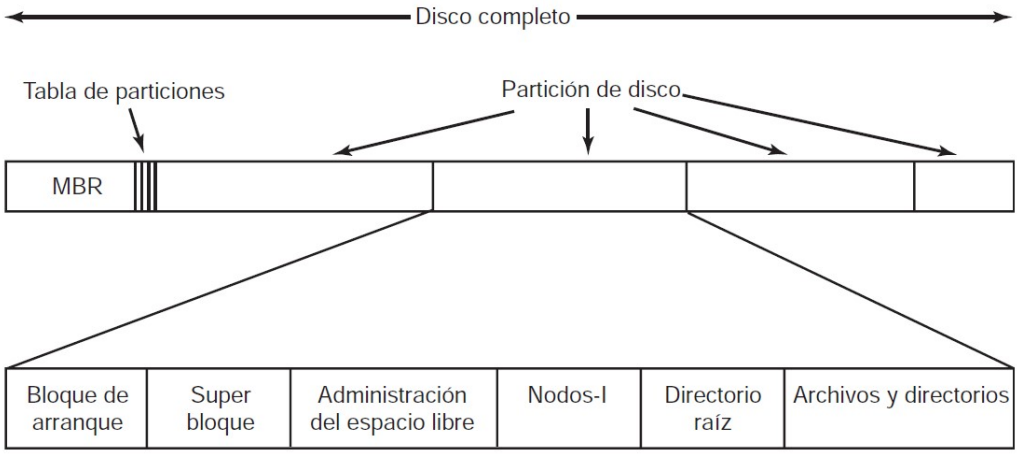
- Superbloque: contiene parámetros clave sobre el sistema de archivos, y se lee en la memoria en el arranque del sistema operativo o con la primera interacción con el sistema de archivos.

- Administración del espacio libre: mapa de bits o lista enlazada para manejar los bloques libres.

- Nodos-I: arreglo de estructuras de datos con información de los archivos (una por cada uno).

- Directorio raíz

- Otros archivos y directorios



4.10 Implementación de archivos (Tanenbaum, 2009, p. 274-280, 321-323)

Se refiere a la forma en que se organizan los archivos en el espacio destinado a ellos en el disco.

4.10.1. Asignación contigua

Es dividir el disco en bloques del mismo tamaño y colocar un archivo tras otro, ocupando una cierta cantidad de bloques. Entonces, a cada archivo se le debe asociar dos números: el bloque de inicio y la cantidad de bloques que ocupa.

El asignar bloques indivisibles implica que siempre una parte del último quedará sin utilizar; esto es la fragmentación interna.

Este método es muy sencillo de implementar y es eficiente (pues todos los bloques del archivo se encuentran juntos, solo hace falta una operación de búsqueda). Sin embargo, acarrea un problema: la fragmentación externa.

La fragmentación externa ocurre cuando se eliminan archivos, pues quedan bloques libres en medio de otros ya ocupados. Esto implica elegir una forma de agregar archivos: siempre al final, o buscando huecos libres (la segunda requiere siempre saber el tamaño máximo del archivo).

Para solucionar la fragmentación externa se debe recurrir a la compactación del disco, un proceso muy caro.

La asignación contigua puede ser ideal para almacenamiento permanente ROM, pues al nunca eliminarse archivos, no se produce fragmentación externa.

4.10.2. Asignación de lista enlazada

Un archivo consiste en un conjunto de bloques; cada bloque contiene una palabra reservada al comienzo que almacena un puntero al siguiente bloque.

Si bien soluciona la fragmentación externa, la búsqueda es muy lenta, puesto que llegar a cierto bloque requiere haber leído todos los anteriores. Otra pequeña desventaja es que cada bloque tendrá un tamaño utilizable distinto a una potencia de dos, algo que puede ralentizar las operaciones de lectura de un programa si cree que puede ocupar el tamaño real y no el disponible de un bloque (implicaría la escritura a dos bloques).

4.10.3. Asignación de lista enlazada con FAT

Utiliza el mismo principio de la lista enlazada normal, pero almacenando los punteros en una tabla FAT (File Allocation Table, Tabla de Asignación de Memoria) en memoria principal.

Este método acelera la búsqueda pues, si bien se debe recorrer toda la lista para llegar a un bloque, los punteros se encuentran en la memoria principal. Además, lógicamente, el tamaño disponible de un bloque es igual a su tamaño real, pues no se necesita guardar en él el puntero.

Sin embargo, acarrea una desventaja significativa: una gran cantidad de bloques requiere una gran tabla que debe mantenerse cargada en memoria principal todo el tiempo.

4.10.4. Nodos-I

Consiste en una estructura de datos relacionada con un archivo en particular. Estos Nodos-I (nodos índice) se guardan en el disco, y solo se cargan en memoria cuando se abre el archivo.

Como suele haber un número máximo de archivos abiertos simultáneamente, este es el mismo número máximo de Nodos-I en memoria. Entonces, el espacio que deberá reservarse en memoria es independiente del tamaño del disco y la cantidad de bloques (a diferencia de la lista enlazada con FAT), y suele ser bastante más pequeño.

Un Nodo-I contiene los atributos del archivo, algunos apuntadores a bloques con el contenido y puede tener un apuntador a un bloque con más apuntadores a contenido, por si el archivo requiriera más espacio. También lleva la cuenta de cuántas referencias hay a este archivo, para saber si puede ser eliminado (al eliminar la última referencia).

Otra forma que suelen tener los Nodos-I suele ser:

- Atributos
- 10 apuntadores directos a bloques del archivo
- 1 apuntador indirecto simple (a un bloque de apuntadores a bloques del archivo)
- 1 apuntador indirecto doble (a un bloque de apuntadores a bloques de apuntadores a bloques del archivo)
- 1 apuntador indirecto triple (similar a los anteriores, pero con 3 capas)

4.11 Implementación de directorios (Tanenbaum, 2009, p. 280-282)

4.11.1. Acceso a archivos y atributos

Un directorio debe almacenar los nombres de los archivos que contiene, y con este posibilitar acceder al archivo y sus atributos.

Implementación de Archivos	Acceso a archivos	Ubicación de atributos
Continua	Dirección de disco del archivo	En la entrada del directorio
Lista enlazada	Número del bloque inicial	En la entrada del directorio
FAT	Número del bloque inicial	En la entrada del directorio
Nodos-I	Número del Nodo-I	En el Nodo-I

4.11.2. Tamaño de nombres

Pueden ser de longitud:

- Fija: obliga a que todos los nombres ocupen el mismo espacio, lo que conlleva un desperdicio considerable si se permiten nombres largos.
- Variable:

  Al final de la entrada: requiere almacenar la longitud de cada entrada al inicio, seguida por los campos necesarios para el acceso al contenido del archivo y los atributos y, al final, el nombre del archivo en sí. Implica entradas de tamaño variable, produciendo fragmentación externa poco costosa de compactar (pues se encuentra en la memoria principal).

  En un heap: Se almacena un puntero a una dirección en un heap que tiene cada directorio para el propósito específico de guardar los nombres.

4.11.3. Aceleración de la búsqueda

Las formas de organizar las entradas de acuerdo a los nombres ya mencionadas requieren realizar una búsqueda lineal, leyendo entrada por entrada hasta encontrar aquella cuyo nombre coincida con el buscado. Esto puede ser lento si hay directorios con muchos archivos.

Con el fin de acelerar la búsqueda, puede crearse una tabla de hash que procesa el nombre del archivo y devuelve un código de hash. Todas las entradas con el mismo código se colocan en una lista enlazada, reduciendo la búsqueda a la lectura de una fracción de las entradas.

Una alternativa es la implementación de una caché, únicamente útil si los archivos a los que se accede suelen ser los mismos.

4.12 Sistemas de archivos alternativos (Tanenbaum, 2009, p. 285-291)

4.12.1. Estructurados por registro (LFS)

Dada la lenta mejora de la velocidad de los discos comparada con la evolución de otros medios de almacenamiento y hardware en general, se ha diseñado otro sistema de archivos que busca ser más eficiente.

Un LFS (Log-structured File System, Sistema de archivos estructurado por registro) se basa en que, si bien el disco no aumenta su velocidad a un ritmo suficientemente rápido, las cachés de disco

sí lo hacen. Por esto, la mayoría de las operaciones de lectura no requieren acceso al disco y no representan un problema.

El verdadero inconveniente es la gran cantidad de escrituras pequeñas que se necesitan. Un LFS utiliza un búfer en memoria en el cual se realizan todas las escrituras y, cada cierto tiempo, un hilo "escritor" realiza una escritura al disco en un solo segmento al final. Se considera al disco como un registro.

Si se utilizan Nodos-I, una gran parte de las escrituras ocurrirán sobre estos, por lo que tenerlos en una parte fija de la memoria reduciría en gran medida la efectividad de este sistema de archivos. En su lugar, se mantiene una tabla en caché que relaciona cada entrada con un nodo, apuntando a este donde sea que esté en el disco.

Además, un LFS requiere un hilo "limpiador", que recorra al registro para compactarlo. Esto implica:

- Verificar si los Nodos-I que contiene ya han sido sobrescritos (si la tabla con apuntadores no les apunta)
- Verificar si los bloques que contiene aún están en uso (si los Nodos-I no les apuntan)
- Escribir los nodos y bloques que aún estén en uso en el búfer de memoria para ser escritos en el siguiente segmento
- Liberar el segmento

En síntesis, el disco se estructura como un búfer circular, con un disco escritor que lo llena y un hilo limpiador que lo vacía.

4.12.2. Por bitácora (JFS)

Los JFS (Journaling File System, Sistema de archivos por bitácora) buscan una forma robusta de salvar las posibles fallas que puede haber durante un proceso que requiere múltiples escrituras.

Esto lo logra creando una entrada de registro en el disco con las operaciones que se debe realizar antes de ejecutarlas, y borrando la entrada al finalizar. De esta forma, si ocurre un fallo y la máquina se reinicia, lo primero que hará el sistema operativo es buscar estas entradas y realizar las acciones listadas.

Algo importante a tener en cuenta es que las acciones deben ser **idempotentes**: que puedan realizarse múltiples veces sin ningún peligro. Las acciones no idempotentes suelen poder convertirse en este tipo con algunos cambios.

Además, es conveniente que las acciones se realicen como una **transacción atómica**, para aumentar la confiabilidad al garantizar que ninguna acción quedará a medio hacer.

Algunos ejemplos de JFS son NTFS (Windows), ReiserFS y ext3 (Linux).

4.12.3. Virtuales (VFS)

Un sistema operativo suele ser capaz de manejar más de un sistema de archivos simultáneamente. Windows lo logra asignándole a cada sistema una letra de unidad (C:, D:). Los sistemas UNIX los unifican mediante la montura, con un VFS (Virtual File System, Sistema de archivos virtual).

En un VFS intervienen dos interfaces: una para los procesos de usuario (POSIX) y otra para los sistemas de archivos. Ambas consisten en ciertas llamadas: las llamadas POSIX y ciertas funciones definidas en el VFS e implementadas por cada sistema de archivos.

Un VFS define tres objetos:

- Superbloque: sistema de archivos
- Directorio: directorio del sistema de archivos
- Nodo-V: archivo

Cada objeto tiene sus propias llamadas. Al montarse un sistema de archivos (ya sea en el arranque o luego), este se debe registrar con el VFS proveyéndole todas las llamadas que necesita, ya sea en un único vector o con uno por objeto.

La operación de lectura requiere identificar cuando se accede a un sistema montado, para así:

1. Identificar el superbloque correspondiente al sistema de archivos
2. Identificar el directorio raíz en el superbloque y seguir la ruta especificada
3. Crear un Nodo-V y hacer una llamada al sistema de archivos que copie en este la información del Nodo-I correspondiente (y otra información, como el puntero al vector de funciones)
4. Crear una entrada en la tabla de descriptores de archivos (file descriptors) que apunte al Nodo-V

Así, utilizando el descriptor de archivo se puede acceder al archivo correspondiente sin importar en qué dispositivo se encuentre.

4.13 Administración del espacio en disco (Tanenbaum, 2009, p. 292-298)

4.13.1. Modos

Un archivo puede ocupar n bytes consecutivos o bloques no necesariamente contiguos. El primero es más veloz para la búsqueda pero inaceptablemente lento para ser movido a otro sector del disco, y provoca fragmentación externa.

Es por esto que se suele usar bloques de tamaño fijo.

4.13.2. Tamaño de bloque

Un tamaño de bloque grande implica un desperdicio de espacio por la fragmentación interna; mientras que uno pequeño implica un mal rendimiento, pues para archivos más grandes que el tamaño de bloque se requieren más búsquedas.

Dado que el espacio en disco aumenta más rápido que su velocidad, suele favorecerse a un tamaño de bloque más grande.

4.13.3. Bloques libres

Es necesario que el sistema operativo sea capaz de identificar aquellos bloques disponibles para ser ocupados al crearse nuevos archivos o agrandarse otros ya existentes. Esto puede hacerse de varias formas:

Lista enlazada

Se utilizan algunos bloques para guardar punteros a bloques libres. Los primeros reservan una palabra para guardar un puntero al siguiente bloque con punteros a bloques libres.

Este método ocupa mucha memoria, pues por cada bloque libre se requerirá n bits que le apunten para saber que lo está (si el disco es de 2^n). Sin embargo, el tamaño de la lista cambia dinámicamente, por lo que si disminuyera la cantidad de bloques libres, también disminuiría la cantidad de bloques ocupados por la lista (liberándolos).

Para que el sistema operativo tenga acceso a la información de aquellos bloques libres con este método, debe mantener en todo momento un bloque de punteros en la memoria principal. De este modo:

- Al agrandar (o crear) un archivo, podrá saberse qué bloques pueden ser ocupados.
- Al achicar (o eliminar) un archivo, podrá marcarse los bloques correspondientes como “libres”.

Si se requiriera más bloques libres de los apuntados por el bloque de punteros en memoria, se debe traer otro del disco; lo opuesto pasa si se han liberado más bloques de los que pueden ser apuntados por el bloque actualmente cargado en memoria.

Siempre que se intercambia un bloque de punteros con el disco es conveniente solo mover una parte del bloque, pues así se evita la posibilidad de que se produzcan muchas operaciones de E/S por intercambio de bloques. Caso contrario, de moverse bloques completos:

- Teniendo el bloque de punteros llenos, al liberarse más bloques se debe llevar el bloque actual al disco y traer uno nuevo vacío.
- Teniendo el bloque de punteros vacío, al ocuparse bloques se debe traer un bloque lleno del disco.

Está la posibilidad de que, estando en cualquiera de estas situaciones, comience a liberarse y ocuparse el disco intercaladamente, ralentizando el sistema.

Lista enlazadas para bloques consecutivos

Es una variación de las listas enlazadas que, en lugar de guardar un puntero a cada bloque libre, guarda un puntero y un número. Estos, permiten identificar una secuencia de bloques libres:

- el puntero, apuntando al primero bloque, y
- el número, guardando la cantidad de bloques libres.

Mapa de bits

Hace corresponder, de forma secuencial, un bit a cada bloque del disco, con un 0 si está ocupado y un 1 si está libre (o viceversa).

Ocupa menos espacio que la lista enlazada en general, a menos que haya mucho espacio ocupado en el disco.

Tiene un tamaño fijo, por lo que puede colocarse en la memoria virtual y ser paginado cuando sea necesario. Además, dado que está ordenado, el encontrar bloques libres requiere búsquedas de bloques cercanos en el disco, aumentando su rendimiento.

4.13.4. Cuotas de disco

Los sistemas operativos multiusuario suelen limitar el espacio en disco que puede ocupar cada uno de sus usuarios.

Esto lo logra asignando una cuota a cada usuario, las cuales se guardan en una tabla de cuotas.

Cada archivo que esté abierto tiene una entrada en la tabla de archivos abiertos, en la cual se almacena el UID del propietario del archivo y un puntero a su entrada en la tabla de cuotas. De esta forma, cualquier aumento o disminución en la cantidad de bloques (o archivos) que correspondan al usuario quedan registrados.

En la tabla de cuotas se almacena también, tanto para los bloques como para los archivos:

- Un límite suave, que puede ser sobrepasado
- Un número de advertencias pendientes, que disminuye siempre que se inicia sesión cuando se ha sobrepasado el límite suave, e impide el ingreso al usuario si agota el número de advertencias (por ignorarlo)
- Un límite duro, que no puede ser sobrepasado

4.14 Respaldo del sistema de archivos (Tanenbaum, 2009, p. 298-304)

"Vaciar" se refiere a hacer una copia de seguridad al disco. Puede ser:

- Completo (todos los archivos)
- Incremental (solo aquellos archivos que han sufrido modificaciones desde el último respaldo)

También hay vaciados:

- Físicos: mediante un programa simple que respalda todo el disco (es completo), en orden. Es rápido, pero se desaprovecha el espacio.
- Lógicos: comienzan en un directorio y respaldan sus contenidos en forma recursiva. Puede ser incremental.

En el caso de un vaciado lógico incremental, se deben respaldar primero todos los directorios que contengan archivos o directorios modificados (o hayan sido modificados ellos mismos). Luego, se deben respaldar solo los archivos modificados. Esto es así para que, al realizar la recuperación, primero se arme la estructura de directorios del sistema de archivos, antes de comenzar a colocar los contenidos.

Hay algunas otras cuestiones relacionadas con los vaciados lógicos incrementales, como los segmentos huecos dentro de los programas, los vínculos y los archivos especiales (canales).

4.15 Consistencia del sistema de archivos (Tanenbaum, 2009, p. 304-307)

Los sistemas operativos suelen proveer programas que verifican la consistencia del sistema de archivos.

Hay múltiples aspectos que pueden ser verificados, algunos de los cuales pueden ser corregidos automáticamente.

Dos verificaciones comunes son:

- Archivos: mediante dos tablas de contadores, uno para bloques ocupados por archivos y la otra para bloques libres, se analiza que todo bloque esté ocupado por un solo archivo o solo aparezca una vez en la lista enlazada o mapa de bits de bloques libres. De no ser así, pueden darse tres situaciones:

  - Si un bloque no aparece ni ocupado ni libre, se lo coloca como libre.
  - Si un bloque aparece como libre dos veces (en lista enlazada únicamente), se elimina uno de los punteros en la lista.
  - Si un bloque aparece como ocupado dos veces, se debe copiar los contenidos a un bloque libre y asignar este bloque a uno de los archivos, para que dejen de compartirlo.

- Directorios: construye una tabla de contadores de archivos. Comienza analizando desde el directorio raíz y, al terminar, verifica que la cantidad de veces que aparece un archivo coincida con aquella indicada en el Nodo-I. De no ser así, debe modificarse el valor almacenado en el Nodo-I.

4.16 Rendimiento del sistema de archivos (Tanenbaum, 2009, p. 307-312)

4.16.1. Cachés

Una caché es una colección de bloques que pertenece al disco pero se mantienen en memoria para mejorar el rendimiento. Se puede administrar con los algoritmos de reemplazo de página, pero se deben tener en cuenta dos problemas:

- ¿Es probable que el bloque se utilice de vuelta pronto?
- ¿Es el bloque esencial para la consistencia del sistema de archivos?

Así, si se organizan los bloques en una lista enlazada, se puede decidir cuáles bloques deberían ser escritos inmediatamente al disco, y cuáles pueden esperar.

Sin embargo, que se esté escribiendo en un bloque con frecuencia requiere escrituras al disco para no perder trabajo, más allá de lo que diga la lista enlazada. Esto puede ser solucionado mediante una llamada al sistema *sync* ejecutada por un programa llamado *update* cada cierto tiempo (30 segundos en UNIX).

Una forma más confiable pero no tan veloz de realizar las escrituras al disco son las cachés de escritura inmediata. Actualmente se han dejado de usar.

4.16.2. Lectura adelantada de bloque

Consiste en llevar el siguiente bloque de un archivo a la caché cuando se ha solicitado uno. Solo es beneficioso en archivos de acceso secuencial, por lo que un sistema operativo podría llevar un registro de la forma en que se ha accedido a un archivo recientemente: secuencial o aleatoria.

4.16.3. Reducción del movimiento del brazo del disco

Se puede lograr de dos maneras: colocando los bloques pertenecientes a un mismo archivo en sectores cercanos (algo sencillo si se administran los bloques libres con un mapa de bits). La otra forma es disminuyendo la distancia promedio entre los Nodos-I y los archivos: si se colocan los Nodos-I en el medio del disco, el tiempo de búsqueda promedio será la mitad que si estuvieran al inicio.

4.16.4. Desfragmentación del disco

Es un algoritmo realizado por un programa llamado *desfragmentador*; consiste en mover los bloques dentro de una partición para que todos aquellos pertenecientes a un archivo queden contiguos, y así mejorar el rendimiento.

Capítulo 5

Entradas y Salidas

5.1 Tipos de dispositivos de Entrada/Salida (Tanenbaum, 2009, p. 330)

- De bloque: se organizan por bloques de memoria, los cuales pueden ser buscados.
- De caracter: se maneja mediante un flujo de caracteres.
- Otros

5.2 Comunicación con los dispositivos de Entrada/Salida (Tanenbaum, 2009, p. 331-332)

Una unidad de E/S consiste en el dispositivo y un controlador de dispositivo o adaptador. El segundo se encarga de proveer la interfaz al sistema operativo para manejar los dispositivos correspondientes.

Por su parte, la interfaz entre la controladora y el dispositivo de E/S suele acomodarse a algún estándar. Suele ser de muy bajo nivel, cada comunicación realizándose como un flujo serial de bits precedido por un preámbulo y sucedido por un ECC (Error Correction Code, Código de Correción de Errores) o suma de comprobación.

Al recibir el flujo de bits, la controladora lo ensambla en un registro interno y verifica que no haya errores (usando el ECC).

5.3 Entradas y Salidas mediante puertos y por asignación de memoria (Tanenbaum, 2009, p. 332-336)

Los controladores tienen registros de control y búfers de datos que permiten al sistema operativo dar órdenes a los dispositivos, conocer información acerca de ellos y transmitirles o recibir información.

Estos registros de control y búfers pueden implementarse de dos maneras:

- Con puertos de E/S: asignándole a cada registro de control un número de puerto y proveyendo instrucciones especiales `IN` y `OUT` para manipularlos.

- Con asignación de memoria: escribiendo el contenido de los registros de control y búfers de datos a la memoria.

La implementación mediante asignación de memoria tiene las ventajas de no requerir instrucciones especiales, facilitando la programación a mayor nivel; permite usar las mismas instrucciones de manipulación de memoria con los registros de control sin tener que moverlos antes; y puede implementarse la protección simplemente no colocando la memoria que contiene a estos registros y búfers en el espacio de direcciones virtuales de un usuario.

Sin embargo, este método también requiere solucionar dos problemas:

- Problema de la caché: si los registros de control se colocaran en caché y una controladora los actualizara en la memoria principal, este cambio no se vería reflejado en la caché. La solución es la implementación de una caché selectiva por hardware y manejada por el sistema operativo.

- Problema del análisis de direcciones: cuando se actualiza un registro de control, la controladora correspondiente debe hacer algo en consecuencia. Si estos registros están en memoria, la única forma en que las controladoras sepan que deben hacer algo es analizando cada referencia a memoria que se haga. Hay varias soluciones posibles:

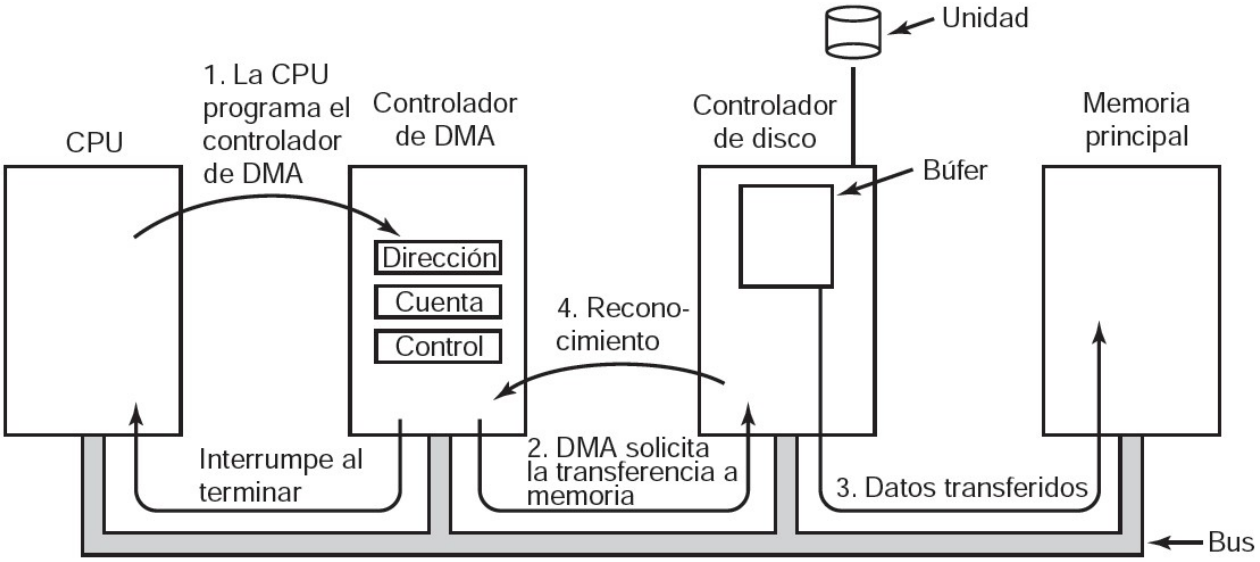
    - Enviar todas las referencias de memoria a memoria y, si esta no responde, mandarla por los otros buses.

    - Colocar un dispositivo husmeador que pase todas las direcciones del bus de memoria a los dispositivos de E/S (lo cual tiene el inconveniente de que estos pueden llegar a ser más lentos).

    - Filtrar las direcciones en el chip del puente PCI, cargando un rango de memoria al inicio del sistema en el cual se hallan los registros de control y búfers de datos. Todas las direcciones que se encuentren en este rango son enviadas al bus PCI en lugar de a la memoria.

5.4 Acceso Directo a Memoria (DMA) (Tanenbaum, 2009, p. 336-339)

El chip DMA (Direct Memory Access) permite realizar las operaciones de E/S sin depender tanto de la CPU, pues como su nombre indica, accede directamente a la memoria.



Inicialmente, la CPU programa al DMA para que sepa qué debe transferir y a dónde. Entonces, este chip maneja al controlador de dispositivo correspondiente hasta concretar la transferencia, y finaliza provocando una interrupción.

La CPU y el DMA no pueden usar el bus al mismo tiempo; de aquí surgen dos mecanismos:

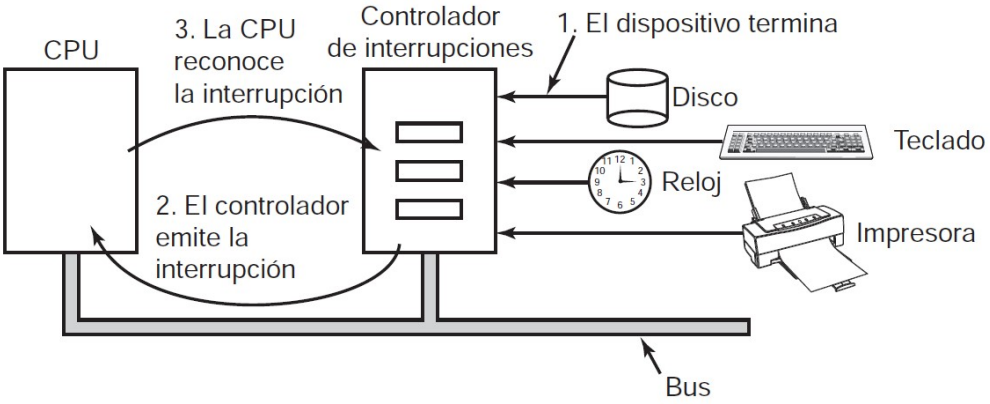
- Robo de ciclo: el controlador de DMA solicita la transferencia de una palabra y la obtiene, forzando a la CPU a esperar si también quisiera usar el bus.
- Modo de ráfaga: se realiza la transferencia de múltiples palabras de modo consecutivo.

Otros mecanismos que puede utilizar el DMA según el recorrido de los datos a transferir son:

- Modo “fly-by”: el controlador de dispositivo transfiere directamente a la memoria principal.
- Mediante el chip DMA: el controlador de dispositivo transifere al DMA, y este a la memoria. Este mecanismo es más flexible pero más costoso.

5.5 Interrupciones (Tanenbaum, 2009, p. 139-343)

Las interrupciones son emitidas por un controlador de interrupciones al recibir señales emitidas por dispositivos de E/S (indicando que finalizaron). Luego, la CPU reconoce la interrupción cuando está lista para procesar otra.



Al provocar una interrupción, el controlador de interrupciones coloca un número en las líneas de dirección (IRQ, Interrupt Request) que es usado para hallar la dirección de memoria del servicio de atención de interrupción en la tabla de vectores de interrupción.

Las interrupciones pueden clasificarse en precisas e imprecisas, entrando en la primera categoría sí y solo sí cumplen:

- El PC se guarda en un lugar conocido
- Las instrucciones anteriores a la apuntada por el PC se han ejecutado por completo
- Las instrucciones posteriores a la apuntada por el PC no se han ejecutado
- Se conoce el estado de ejecución de la instrucción apuntada por el PC

Las interrupciones imprecisas son comunes en máquinas superescalares con pipelines (tuberías). Se les dice así porque cada interrupción requeriría guardar mucha información para poder luego restaurar el contexto, por lo que a veces se opta por no guardarla. Este sería el caso de una `TRAP` (interrupción interna) producida por un error fatal como una división por 0, por ejemplo, pues nunca tendremos que restaurar el contexto.

5.6 Objetivos del software de E/S (Tanenbaum, 2009, p. 343-344)

Independencia de dispositivos: se debe poder crear programas que accedan a cualquier dispositivo de E/S sin tener que especificarlo por adelantado.

Denominación uniforme: el nombre de un archivo o dispositivo debe ser una cadena o número sin depender del dispositivo.

Manejo de errores: debe realizarse siempre que sea posible a bajo nivel, por el controlador.

Sincronía de transferencias: las transferencias suelen ser asíncronas (controladas por interrupciones), pero dado que es mucho más fácil programar considerándolas síncronas, el sistema operativo debe encargarse de que las transferencias controladas por interrupciones parezcan síncronas (mediante el bloqueo de procesos).

Uso de búfer: los datos entrantes suelen ser almacenados en un búfer del sistema operativo antes de ser copiados a su destino final.

Dispositivos compartidos y dedicados: los dispositivos pueden clasificarse en compartidos y dedicados según si es posible que más de un usuario los use a la vez. El sistema operativo debe ser capaz de manejar ambos correctamente, a fin de evitar interbloqueos.

5.7 Maneras de realizar la E/S (Tanenbaum, 2009, p. 344-347)

5.7.1. E/S programada

Utiliza *sondeo* u *ocupado en espera*: se carga un búfer con toda la información a transferir y se entra en bucle hasta que toda haya sido transferida al dispositivo, o se entra en bucle hasta recibir toda la información del dispositivo.

Su gran desventaja es que ocupa mucho la CPU, desperdiciando su tiempo.

5.7.2. E/S controlada por interrupciones

Se inicia una transferencia parcial y se bloquea al proceso que la realiza, hasta que el controlador de dispositivo emita una interrupción indicando que terminó la transferencia, permitiendo continuar con la siguiente y repitiendo lo anterior.

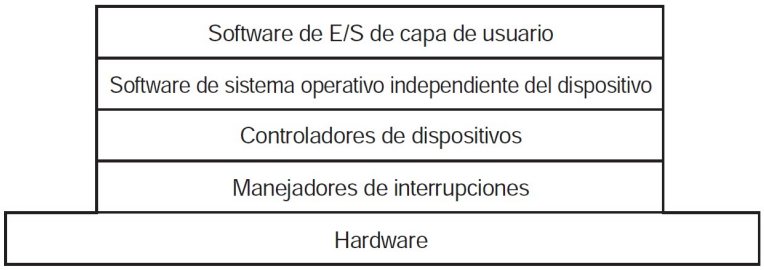
La diferencia con la E/S programada es que en este se utilizan bloqueos en lugar de espera ocupada, permitiendo a otros procesos ejecutarse mientras el dispositivo hace su trabajo.

5.7.3. E/S mediante DMA

En lugar de realizar una interrupción por parte transferida (como la E/S controlada por interrupciones), el chip DMA realiza todo el trabajo y emite una única interrupción, ahorrando tiempo de la CPU.

Las desventajas son leves: requiere el hardware especial (que es típicamente más lento que la CPU, y puede llegar a ser más lento que algún dispositivo de E/S).

5.8 Capas del software de E/S (Tanenbaum, 2009, p. 348-360)



5.8.1. Manejadores de interrupciones

Atender una interrupción requiere realizar varios pasos previos para guardar el contexto del proceso que estaba ejecutándose previamente y realizar el cambio de contexto. También involucra una llamada al planificador de procesos posterior a la atención, con su cambio de contexto correspondiente. Si la transferencia de E/S pasó un proceso de estado bloqueado a listo, este es elegible.

Este proceso ocupa una cantidad considerable de instrucciones y, dependiendo de la máquina, puede involucrar a la MMU, el TLB y la tabla de páginas.

5.8.2. Controladores de dispositivos

Los controladores de dispositivos o *drivers* son software provisto, por lo general, por el fabricante de un dispositivo de E/S. A veces se ejecutan en el kernel, y otras en el modo usuario.

Un driver hace de interfaz entre el software independiente del dispositivo y los dispositivos de E/S. Para esto, valida toda la información que la capa superior le envía, y tras comprobar que el dispositivo correspondiente se encuentra en funcionamiento y listo, le envía una secuencia de comandos.

Una vez enviada esta secuencia, el driver se bloqueará hasta que el dispositivo finalice sus actividades (haciéndoselo saber por una interrupción). Por último, si es que no hubo errores, le devuelve esta información al software independiente del dispositivo.

Algunos sistemas operativos permiten agregar o quitar dispositivos de E/S mientras la máquina esta en funcionamiento; esto requiere software adicional que permita abortar toda transferencia pendiente y en curso sin dañar ninguna estructura de datos del kernel.

5.8.3. Software independiente del dispositivo

Se encarga de realizar todas las actividades comunes a las transferencias con los dispositivos de E/S. Suele cumplir varias funciones:

Interfaz uniforme para los drivers

La interfaz puede o no ser estándar (todas iguales). Sea cual fuere el caso, para cada tipo de interfaz el sistema operativo define un conjunto de operaciones que el driver debe definir. Así, al kernel no le importa de qué dispositivo se trata; solo sabe que si quiere hacer algo con este, debe utilizar las operaciones proporcionadas.

El nombre que se le coloca a cada dispositivo también está relacionado a la interfaz uniforme.

Uso de búfer

Se refiere a las partes de la memoria que son utilizadas para almacenar los datos de una transferencia de E/S temporalmente.

De no haber búfer, por cada fragmento de la información transmitida o a transmitir se debería iniciar el proceso de usuario, para luego bloquearlo tan solo un instante después, algo muy ineficiente.

De haber un solo búfer en espacio de usuario, podría ocurrir que la página en la que este se encuentra sea llevada al disco, ocasionando problemas. Bloquear las páginas con estos búfers sería una solución viable, pero bajaría el rendimiento si hay muchas bloqueadas.

De haber un búfer en el espacio de usuario y otro en el kernel se evita el problema de las páginas, pero aún queda otro: el búfer en el kernel podría estar transfiriendo sus datos (ya sea al búfer en el espacio de usuario o al dispositivo de E/S), mientras que debería recibir otros simultáneamente.

La solución a todo esto es el doble búfer: utiliza, aparte de los ya mencionados, otro búfer en el kernel, de modo que si uno está enviando sus contenidos, el otro sea encargado de recibir otros, y viceversa.

Otra alternativa es el búfer circular, que consiste en una región de memoria y dos punteros: uno a la siguiente palabra libre, y otro a la última palabra sin eliminar.

Reporte de errores

Si bien la controladora del dispositivo es la encargada de detectar los errores de E/S reales, es el software independiente del dispositivo el encargado de proporcionar el marco para su tratamiento.

También se consideran errores de E/S aquellos en los que, por una mala programación, un proceso pide algo imposible o proporciona parámetros inválidos.

Asignación y liberación de dispositivos dedicados

Aquellos dispositivos que solo pueden ser accedidos por un proceso a la vez requieren mecanismos especiales que garanticen esto.

Un mecanismo podría ser manejarlos como archivos especiales, de modo que `open` solicite el acceso (fallando si ya estuviera ocupado) y `close` lo libere.

Otra forma sería mediante operaciones especiales que bloqueen a un proceso cuando este solicita que se le asigne un dispositivo dedicado ya ocupado, colocándolo en cola para cuando este haya sido liberado.

Tamaño de bloque independiente de dispositivo

Proporcionar un mismo tamaño de bloque (o caracter) para los procesos de usuario, sin importar cómo funcione realmente cada dispositivo.

5.8.4. Software de E/S de usuario

Consiste en procedimientos de biblioteca que realizan las llamadas al sistema de E/S (formateando o no los datos).

También se considera dentro de esta capa a las colas (*spooling*) utilizadas para virtualizar algún dispositivo de E/S, así como los daemons que las administran.

5.9 Discos (Tanenbaum, 2009, p. 360-388)

5.9.1. Discos magnéticos

Son los discos flexibles y discos duros.

Se organizan en pistas circulares concéntricas apiladas (cilindros), cada una con cabezas para leerlas. Además, las pistas se subdividen en sectores, que tienen huecos entre ellos.

Los sectores tienen un preámbulo, con información como el número de cilindro, pista y sector; los datos y un área al final destinada para el ECC. Esta organización se consigue con un **formato de bajo nivel**.

Un disco magnético tiene un controlador propio que se encarga de las transferencias de E/S, así como de la corrección de errores. También tiene búfers y cachés para el almacén de datos a escribir y datos que posiblemente sean leídos pronto.

Para realizar la búsqueda, se debe especificar el cilindro, la cabeza y el sector. Entonces, el controlador ordenará a la cabeza correspondiente que se mueva al cilindro indicado y rotará al disco hasta llegar al sector que se busca. Es común que, aunque sólo se busque un sector para lectura, en realidad se lean los siguientes también y se escriban a la caché, pues la rotación continúa.

Como toda lectura requiere una validación mediante el ECC, el leer múltiples sectores consecutivos puede demorar más tiempo del necesario si la rotación continuó y la cabeza ya pasó el siguiente sector (asumiendo que no todos entraron en la caché). Esto se puede lograr mediante el **entrelazado**: intercalar sectores de modo que cuando la controladora solicite el siguiente sector, la cabeza se encuentre próxima a este.

El primer sector en la mayoría de las computadoras es conocido como MBR (Master Boot Record, Registro Maestro de Arranque) y tiene cierto código para el inicio de la máquina, además de la tabla de particiones.

El **formato de alto nivel** consiste en la estructuración de las particiones (divisiones del disco, cada una formada por sectores) en bloques (de arranque, superbloque, administración del espacio libre, Nodos-I, etc.).

La búsqueda en el disco puede realizarse mediante distintos algoritmos. Algunos de ellos son:

- Primero en llegar, primero en ser atendido (First Come, First Served(FCFS))
- La búsqueda más corta primero (Shortest Seek First (SSF))
- Algoritmo del elevador (en la misma dirección hasta llegar al extremo)
- Siempre en la misma dirección

Estos algoritmos buscan disminuir el tiempo de búsqueda (lo que tarda el brazo en moverse del cilindro actual al requerido), pues el retraso rotacional (tiempo que tarda el sector indicado en colocarse bajo la cabeza) y el tiempo de transferencia de datos actual son irrelevantes en comparación al primero.

Un RAID (Redundant Array of Independent Disks, Arreglo Redundante de Discos Independientes) es una organización de múltiples discos que permite aumentar el rendimiento y confiabilidad al usar múltiples discos. Hay seis alternativas:

- 0: cada bloque consecutivo en otro disco (round-robin)
- 1: cada bloque consecutivo en otro disco, con el doble de discos para que cada bloque tenga una copia
- 2: cada bit (incluidos aquellos de Hamming para la detección y corrección de errores) en un disco distinto
- 3: cada bit (incluido el de paridad para la detección de errores) en un disco distinto
- 4: cada bloque consecutivo en otro disco, más un disco con bloques de paridad para los otros bloques
- 5: cada bloque consecutivo en otro disco, seleccionando algunos bloques por round-robin para contener bloques de paridad para los otros bloques

Cuando se detectan errores, ya sea en la fábrica por el controlador o al ya estar en uso, se marca al sector como defectuoso y se utilizan unos sectores adicionales que quedan en blanco para grabar los datos en estos en lugar del defectuoso. Así, hay dos estrategias:

- Escribir el contenido directamente en el sector adicional (haciendo potencialmente más lentas las búsquedas)

- Mover todos los sectores para mantener el orden (un proceso costoso, especialmente si los sectores no están en blanco, por lo que requerirían copiar no solo el preámbulo)

De forma similar a los JFS, se busca salvar las posibles fallas en los discos asegurándose de que las transferencias se concreten o no se realicen, sin punto medio. Esto se lo logra por un método llamado **almacenamiento estable**.

El almacenamiento estable utiliza dos discos, cada uno con contenidos idénticos de ser correctos. Define tres operaciones:

- Escritura estable: escribe a un sector de un disco, lee lo escrito para comprobar que sea correcto y, de no serlo, repite la operación. Si pasan muchos intentos de escritura fallidos, se considera al sector defectuoso y se comienza a probar con el siguiente. Una vez que la escritura se concretó correctamente, se hace lo mismo con el segundo disco.

- Lectura estable: se lee el bloque del primer disco. Si el ECC validara el contenido, la lectura es correcta; si no lo hiciera, se vuelve a intentar varias veces hasta que lo sea. Si tras muchas lecturas siguiera siendo incorrecto, se lee el bloque del segundo disco. La probabilidad de que ambos se hayan vuelto defectuosos es minúscula.

- Recuperación de fallas: tras una falla, se exploran los discos (o un sector en específico si sabemos en cuál se estaba al ocurrir la falla) para determinar si hubo un problema de escritura. Si dos bloques verifican el ECC y son iguales, todo está bien; si se detecta un error en uno por el ECC, se lo sobrescribe con el otro y, si ambos son correctos pero distintos, el bloque del primer disco sobrescribe al del segundo (bajo la asunción de que el primero fue escrito pero la escritura del segundo no se concretó).

5.9.2. Discos ópticos

Por lo general son de solo lectura. Tienen menor capacidad que los discos magnéticos y son más lentos, pero también más baratos.

Todo lo que se escribe en ellos se encuentra en una espiral.

CD-ROM

Se graban con hoyos y áreas lisas, y son leídos con un láser. Como los discos magnéticos, se graban por sectores con preámbulo y ECC.

CD-R

Son grabables (en distintos momentos) puesto que usan colorantes que reaccionan ante el láser en lugar de hoyos y áreas lisas.

CD-RW

Son regrabables, pues usan una aleación de plata, indio, antimonio y telurio que reacciona ante distintas potencias de láser y cambia entre sus estados cristalino y amorfo, los cuales tienen distintas reflectividades.

DVD

Significa *Digital Versatile Disk, Disco Versátil Digital*. Son muy similares a los CD-ROM pero usan hoyos más pequeños, una espiral más estrecha y un láser menos potente (rojo). Así, tienen mayor capacidad, e incluso más si se les coloca doble capa y/o se los graba de ambos lados.

Blu-ray y HD DVD

Otra mejora respecto a los DVDs, con un láser aún menos potente (azul) que aumenta incluso más la capacidad.

5.10 Relojes (Tanenbaum, 2009, p. 388-394)

5.10.1. Hardware

Los relojes o temporizadores programables utilizan un oscilador de cristal de cuarzo que genera señales periódicas con muy alta frecuencia. Cuentan con un registro contador y un registro contenedor; en el segundo se coloca la cantidad de señales que deben transcurrir hasta que se emita una interrupción. El funcionamiento consiste en cargar el contenedor en el contador, y disminuir al último en 1 con cada señal del oscilador. El contador puede volver a ser cargado con el contador una vez que llega a 0 y se produce la interrupción, si es que se busca generar **pulsos de reloj**. A este modo se le llama “de onda cuadrada”.

Las computadoras suelen contar con otro reloj de respaldo energizado por una batería, para guardar la fecha y hora aún cuando la máquina está apagada.

5.10.2. Tareas de la controladora del reloj

Mantener la hora del día

Al iniciarse la computadora, se utiliza el reloj de respaldo para obtener la hora actual (o se pide al usuario que la ingrese, de no estar este reloj). A partir de eso, la hora puede calcularse de tres formas:

- Contando los pulsos que han ocurrido desde un cierto momento en el tiempo en un registro de 64 bits, para evitar desbordamiento.
- Contar los segundos que han ocurrido desde un cierto momento en el tiempo.
- Contar los pulsos que han ocurrido desde que se inició el sistema.

Implementar el quantum

Establece un contador en la cantidad de pulsos del quantum cada vez que un proceso pasa al estado de ejecución, y lo decrementa en cada pulso hasta que llega a 0, que es cuando la controladora llama al planificador.

Contabilizar el uso de la CPU

Puede realizarse con un temporizador secundario, o con un campo en el que se incremente con cada pulso de reloj ocurrido mientras un proceso está en ejecución.

Llamada al sistema "alarm"

Puede hacerse con una tabla que guarde todas las alarmas con los tiempos en los que deben ocurrir, y una variable que proporcione el tiempo de la próxima alarma.

Una forma más eficiente es mediante una lista enlazada en la que cada ítem (alarma) guarde el tiempo que debe transcurrir desde la anterior alarma.

Temporizadores guardianes (watchdogs)

Son como la llamada `alarm`, pero para procesos del kernel. No produce interrupciones, sino que directamente llama al procedimiento proporcionado (a diferencia de `alarm`).

Perfilamiento, supervisión y recopilación de estadísticas

Algunos sistemas operativos proporcionan un mecanismo a los procesos de usuario para solicitar que se construya un histograma de las ubicaciones en las que suele pasar el PC.

5.10.3. Temporizadores de software

Son una alternativa para el manejo de la E/S periódica.

Las **interrupciones** tienen muy baja latencia (retraso), pero provocan un cambio de contexto (lo cual es muy costoso en tiempo).

El **sondeo** consiste en que el mismo programa compruebe cada cierto tiempo si la transferencia se ha concretado, lo cual puede tener una alta latencia si este evento ocurre justo después del sondeo.

Un **temporizador de software** se comprueba cada vez que la computadora está a punto de salir del modo kernel, para evitar interrupciones y cambios de contexto adicionales. El procedimiento que se debe realizar ocurre allí mismo, y el temporizador se restablece a continuación.

5.11 Interfaces de usuario (Tanenbaum, 2009, p. 394-415)

5.11.1. Teclado

Se genera una interrupción cada vez que se presiona o se suelta una tecla.

El controlador puede adoptar dos filosofías:

- Modo crudo, no canónico u orientado a caracteres: el controlador recibe los caracteres y los pasa sin procesamiento.
- Modo cocido, canónico u orientado a líneas: el controlador espera a que se termine de ingresar una línea para procesarla y corregirla.

Es común que se requiera **eco**: lo que se presiona en el teclado debe ser mostrado por el monitor en el momento.

5.11.2. Ratón

Los hay "con bolita" y ópticos. Sea cual fuere, al producir interrupciones transmiten tres parámetros: Δx, Δy y los botones presionados.

5.11.3. Interfaces Gráficas de Usuario (GUIs)

Una GUI (Graphical User Interface, Interfaz Gráfica de Usuario) es el sistema de ventanas que tienen muchos sistemas operativos interactivos modernos. Pueden estar implementados dentro (Windows) o fuera (UNIX) del kernel.

La entrada para las GUIs suele ser el teclado y el ratón, mientras que la salida suele ir a un **adaptador de gráficos**, un hardware especial que contiene una RAM de video.

Una GUI tiene cuatro elementos principales (conocidos como WIMP): ventanas (Windows), íconos (Icons), menúes (Menus) y dispositivo señalador (Pointing device).

Las GUIs reciben eventos a través de sus ventanas que colocan en una cola de eventos.

Las ventanas contienen otros elementos visuales que permiten la interacción con el usuario; a estos elementos se les llama *widgets*.

Otro concepto a tener en cuenta es el de recursos: estructuras de datos con alguna información relevante para la GUI, como las ventanas, fuentes o mapas de bits.

Por último, las ventanas pueden mostrar gráficos de dos formas: vectoriales (mediante funciones que indican cómo deben dibujarse) o con mapas de bits.

5.12 Clientes delgados (Tanenbaum, 2009, p. 415-417)

Se refiere a las "terminales tontas", aquellas máquinas que consisten solo de un monitor, teclado y tal vez ratón, que se conectan a un mainframe que realiza toda la computación y le dice al monitor qué mostrar.

Actualmente, tal modelo es viable, pues muchos usuarios quieren realizar muchas cosas desde un navegador, ahorrándose así la administración de la computadora.

5.13 Administración de la energía (Tanenbaum, 2009, p. 417-425)

La capacidad de las baterías no ha aumentado mucho con el paso de los años; es por esto que se han ideado métodos para ahorrar energía. Uno de ellos es la asignación de estados a la CPU, la memoria y los dispositivos de E/S: encendido, inactivo, hibernando y apagado. Los primeros tardan menos en reaccionar, pero consumen más energía.

Las pantallas pueden ponerse en estado de inactividad al hacer que dejen de emitir luz, pues el volver a encenderlas solo requiere leer su RAM de video. Esto puede ocurrir cuando un usuario lo decida, o tras cierto tiempo.

El disco duro puede ponerse a hibernar cuando deja de girar, pues tarda un tiempo considerable en volver a su máxima velocidad.

En cuanto a la memoria, puede vaciarse apagarse las cachés (inactividad) o puede hacerse lo mismo con la memoria principal (hibernación), lo que además requiere el apagado de la CPU.

La comunicación inalámbrica también puede representar un problema si el receptor debe estar siempre esperando a una comunicación; una solución es que avise al emisor cuando va a apagarse

para que este no envíe nada. Una vez que vuelva a prenderse, también se le avisa al emisor para que envíe todo lo que temporalmente debió guardar en un búfer.

Otro tema a considerar sería el sobrecalentamiento; este puede evitarse ya sea mediante un ventilador (que hace ruido y consume energía) o disminuyendo el consumo de energía de los aparatos ya mencionados.

Cuando una computadora tiene baterías inteligentes y no está conectada a una alimentación, debe administrarlas correctamente, puesto que así se logra avisar al usuario y un apagado ordenado al detectarse una carga baja. Además, si tiene múltiples baterías, puede también alternarlas al detectar que una tiene poca carga.

Sea como fuere, las controladoras de dispositivos suelen ser capaces de permitir la administración de la energía del dispositivo de E/S por parte de la computadora (ordenándoles cambios de estado), y algunos dispositivos también deberían ser capaces de activar a la computadora cuando la CPU y/o memoria están inactivas o hibernando.

Una forma adicional en la que se puede ahorrar energía es mediante el software, degradando su rendimiento en pos de aumentar el tiempo que durará una batería.

Glosario

BCP Bloque de Control de Proceso, entrada de la tabla de procesos. 18, 45, 48, 75

CMOS Memoria volátil que consume muy poca energía. 5, 8

DMA Direct Memory Access, chip que maneja la comunicación entre los controladores de dispositivo de E/S y la memoria principal, liberando a la CPU. 5, 47, 66, 67, 69

EEPROM Electrically Erasable PROM, memoria que puede ser borrada y reescrita lentamente. 5

grado de multiprogramación Cantidad de procesos en memoria a la vez. 19, 44

PFF Page Fault Frequency, Frecuencia de Fallos de Página. Algoritmo que mide la proporción de fallos de página por proceso respecto del tiempo e identifica a aquellos que no se encuentren en un determinado rango. 44

rwx Bits de protección de lectura (read), escritura (write) y ejecución (execute). 11, 14, 39

shell Interfaz de línea de comandos. 1, 11

SMP Symmetric Multiprocessing, sistema de computación multiprocesador en el que todos tienen acceso a los mismos recursos y pueden realizar las mismas funciones. 9

WSClock Algoritmo de reemplazo de páginas que combina los algoritmos del reloj y del conjunto de trabajo. 43, 44

Siglas

BIOS Basic Input/Output System, Sistema Básico de E/S. 8, 36

CD-ROM Compact Disc, Read-Only Memory (ROM). 7, 8, 74

CPU Central Processing Unit, Unidad Central de Procesamiento. 2, 3, 4, 5, 7, 9, 17, 18, 20, 21, 22, 23, 24, 26, 27, 29, 30, 44, 46, 47, 67, 76, 77

DLL Dynamically Linked Library, Biblioteca de Enlaces Dinámicos. 45

E/S Entradas y Salidas. 1, 2, 3, 5, 7, 8, 10, 11, 19, 27, 28, 29, 39, 47, 53, 65, 66, 67, 68, 69, 70, 71, 75, 76, 77, 78, 79, 80

ECC Error Correction Code, Código de Correción de Errores. 65, 71, 72, 73

FAT File Allocation Table, Tabla de Asignación de Memoria. 57, 58

GID Group Identification, Identificación de Grupo. 10

GUI Graphical User Interface, Interfaz Gráfica de Usuario. 1, 15, 76

IDE Integrated Drive Electronics, Electrónica de Unidad Integrada. 5, 7

IR Instruction Register, Registro de Instrucciones. 3

ISA Instruction Set Architecture, Set de Instrucciones de la Arquitectura. 1

ISA Industry Standard Architecture (Bus). 7, 8

JFS Journaling File System, Sistema de archivos por bitácora. 59, 73

LFS Log-structured File System, Sistema de archivos estructurado por registro. 58, 59

LRU Least Recently Used, Menos Usadas Recientemente (algoritmo de reemplazo de páginas). 42, 44

MBR Master Boot Record, Registro Maestro de Arranque. 55, 72

MMU Memory Management Unit, Unidad de Administración de Memoria (hardware). 38, 39, 40, 46, 69

NFU Not Frequently Used, No Usadas Frecuentemente (algoritmo de reemplazo de páginas). 42

PC Program Counter, Contador de programa. 3, 46, 47, 68, 75

PCI Peripheral Component Interconnect. 7, 8, 66

PSW Program Status Word, Palabra de Estado del Programa. 3, 36

RAID Redundant Array of Independent Disks, Arreglo Redundante de Discos Independientes. 72

RAM Random Access Memory, Memoria de Acceso Aleatorio. 4, 8, 36, 76

RDAE/S Registro de Datos de E/S. 3

RDAM Registro de Datos de Memoria. 3

RDIE/S Registro de Direcciones de E/S. 3

RDIM Registro de Direcciones de Memoria. 3

ROM Read-Only Memory, Memoria de Solo Lectura. 5, 9, 36, 79

SCSI Small Computer System Interface. 7

SP Stack Pointer, apuntador de pila. 3

TLB Translation Lookaside Buffer, Búfer de Traducción Adelantada (hardware). 40, 41, 46, 50, 69

UID User Identification, Identificación de Usuario. 10, 62

USB Universal Serial Bus, Bus Serial Universal. 7

VFS Virtual File System, Sistema de archivos virtual. 59, 60

Bibliografía

Stallings, W. (2005). *Sistemas Operativos: Aspectos internos y principios de diseño*. Pearson Prentice Hall, 5 edition.

Tanenbaum, A. S. (2009). *Sistemas Operativos Modernos*. Pearson Education, 3 edition.