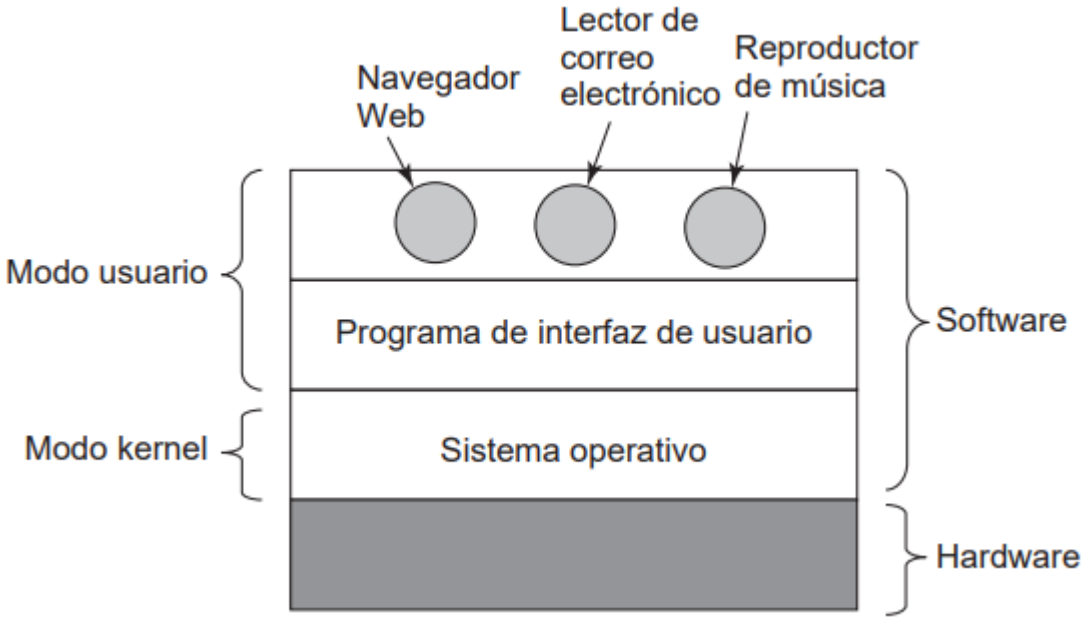
Sistemas Operativos

Podemos definir al **sistema operativo** como un software que permite la **abstracción** de los recursos de hardware a las aplicaciones y se encarga de la **administración** de los mismos. Brindando abstracciones agradables, elegantes y simples a usuarios o programadores con las que puedan trabajar.
El programa con el que los usuarios interactúan se denomina Shell cuando está en forma de texto, o GUI cuando se trata de una interfaz gráfica, aunque no forma parte del sistema operativo, lo utiliza para llevar a cabo su trabajo.
La mayoría de computadoras tienen **dos modos de ejecución**:
Modo kernel: El sistema operativo tiene acceso completo a todo el hardware y puede ejecutar cualquier instrucción que la máquina sea capaz de ejecutar
Modo usuario: El resto del software se ejecuta en modo usuario, en el cual sólo un subconjunto de las instrucciones de máquina es permitido. En particular, las instrucciones que afectan el control de la máquina o que se encargan de la E/S (entrada/salida) están prohibidas para los programas en modo usuario. Siendo el programa de interfaz de usuario, shell o GUI, el nivel más bajo del software en modo usuario y permitiendo la ejecución de otros programas.



Objetivos del sistema operativo:

1. **Facilidad de Uso**: Un sistema operativo hace que un computador sea más cómodo de utilizar. Actúa como una interfaz entre el usuario y el hardware, ocultando los detalles técnicos y proporcionando una

experiencia amigable. Así, los usuarios pueden interactuar con la computadora de manera más sencilla.

2. **Eficiencia**: El sistema operativo permite que los recursos de un sistema informático se aprovechen de manera más eficiente. Administra la memoria, el CPU, las entradas y salidas de datos, y la información en general. Su objetivo es optimizar el uso de estos recursos para un rendimiento óptimo.
3. **Capacidad de evolución**: Un buen sistema operativo debe ser flexible y permitir el desarrollo efectivo, la verificación y la introducción de nuevas funciones sin interferir en los servicios existentes. De esta manera, se adapta a las necesidades cambiantes y garantiza la continuidad del sistema.

Servicios del sistema operativo:

Los servicios del sistema operativo son las funciones básicas que proporciona el software principal o conjunto de programas que gestionan todos los procesos que surgen dentro de un aparato electrónico. Algunos servicios comunes del sistema operativo son:

1) **Desarrollo de programas**
2) **Ejecución de programas**
3) **Acceso a dispositivos de E/S**
4) **Acceso a archivos**
5) **Acceso al sistema (login)**
6) **Detección y respuestas de errores**
7) **Contabilidad**

Sistema Operativo como Máquina Extendida

La arquitectura (conjunto de instrucciones, organización de memoria, E/S y estructura de bus) de la mayoría de las computadoras a nivel de lenguaje máquina es primitiva y compleja de programar, en especial para la entrada/salida.

La abstracción es la clave para lidiar con la complejidad. Las buenas abstracciones convierten una tarea casi imposible en tareas manejables. El trabajo del sistema operativo es crear buenas abstracciones para después implementar y administrar los objetos abstractos entonces creados.

Una de las principales tareas del sistema operativo es ocultar el hardware y presentar a los programas (y a sus programadores) abstracciones agradables, elegantes, simples y consistentes con las que puedan trabajar.

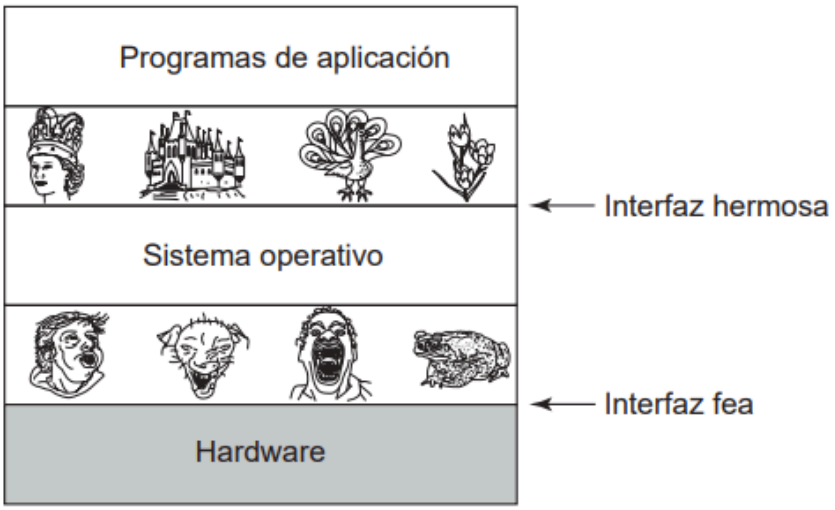


Figura 1-2. Los sistemas operativos ocultan el hardware feo con abstracciones hermosas.

Sistema Operativo como Administrador de Recursos:

El sistema operativo está presente para administrar todas las piezas de un sistema complejo.
El trabajo del sistema operativo es **proporcionar una asignación ordenada y controlada de los** procesadores, memorias y dispositivos de E/S, entre los diversos programas que compiten por estos **recursos**.

La administración de recursos incluye el multiplexaje (compartir) de recursos en dos formas distintas: en el tiempo y en el espacio.

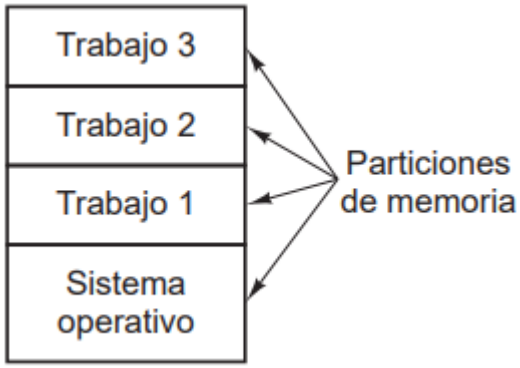
- En tiempo: Los distintos programas o usuarios toman turnos para utilizarlo, uno de ellos obtiene acceso al recurso, después otro, y así en lo sucesivo. La tarea de determinar cómo se multiplexa el recurso en el tiempo es responsabilidad del sistema operativo.
- En espacio: En vez de que los clientes tomen turnos, cada uno obtiene una parte del recurso. Suponiendo que hay suficiente memoria como para contener varios programas, es más eficiente contener varios programas en memoria a la vez, en vez de proporcionar a un solo programa toda la memoria, en especial si sólo necesita una pequeña

Top down
Proporciona abstracción a los programas y aplicaciones.
Bottom up view
Administra las piezas de un sistema complejo.
Alternative view

Proporciona una asignación ordenada de los recursos entre los diferentes programas.

Multiprogramación:

La multiprogramación en sistemas operativos es una técnica que permite cargar varios programas en la memoria de manera simultánea. El CPU cambia rápidamente entre estos programas, lo que posibilita su ejecución simultánea.



(Tres trabajos en memoria)

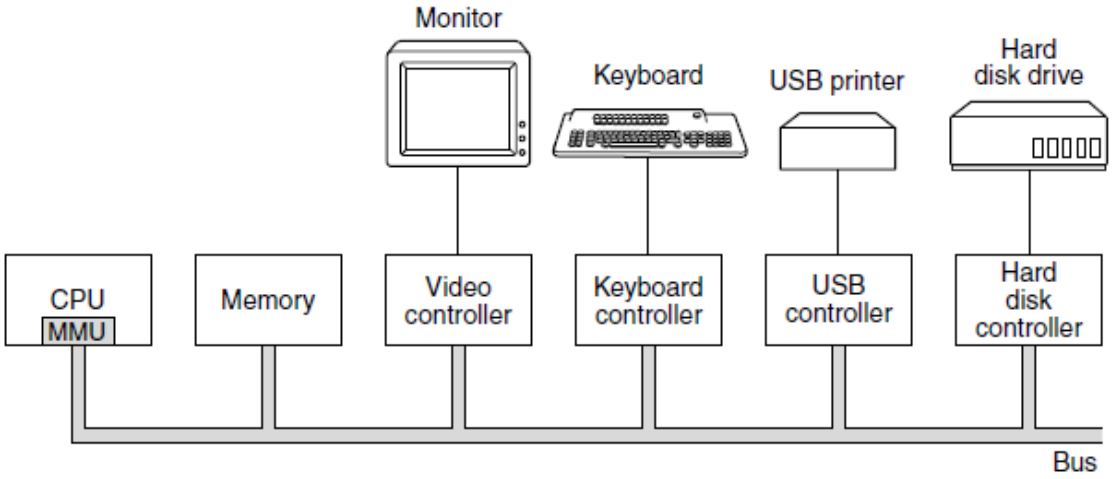
Principales Logros de los Sistemas Operativos:

1. **Procesos**: El concepto de proceso es fundamental en la estructura de los sistemas operativos. Se refiere a:
    a. Un programa en ejecución.
    b. El "espíritu animado" de un programa.
    c. La entidad que puede ser asignada al procesador y ejecutada por él. Los sistemas operativos modernos han desarrollado abstracciones para resolver las dificultades prácticas relacionadas con los procesos.
2. **Gestión de memoria**: Los sistemas operativos han logrado administrar la memoria de acceso aleatorio (RAM) de manera eficiente. Esto implica asignar y liberar recursos de memoria según las necesidades de los programas en ejecución.
3. **Seguridad y protección de la información**: Los sistemas operativos han implementado mecanismos para proteger los datos y garantizar la confidencialidad, integridad y disponibilidad de la información. Esto incluye control de acceso, autenticación y cifrado.
4. **Planificación y gestión de recursos**: Los sistemas operativos planifican y distribuyen los recursos del sistema de manera óptima. Esto incluye la asignación de tiempo de CPU, manejo de colas de procesos y priorización de tareas.

5. **Estructura del sistema**: Los sistemas operativos han evolucionado en su diseño y organización. La estructura interna incluye componentes como el kernel, los controladores de dispositivos y las capas de abstracción que permiten una interacción eficiente entre el hardware y el software de aplicación.

Hardware de la computadora

Conceptualmente, una computadora personal simple se puede abstraer mediante un modelo como el de la figura



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

Las CPU contienen ciertos registros en su interior para contener las variables clave y los resultados temporales:

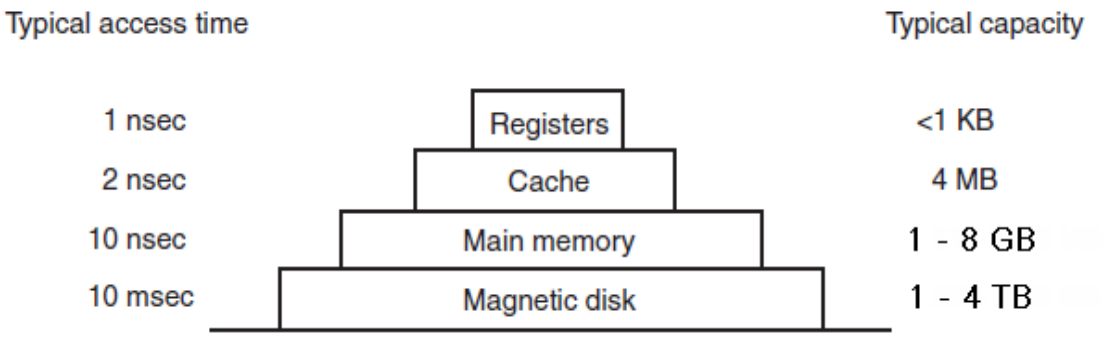
- **Registros especiales**
- **Contador de programa**: Contiene la dirección de memoria de la siguiente instrucción a obtener.
- **Apuntador de pila**: La pila contiene un conjunto de valores por cada procedimiento al que se ha entrado, pero del que todavía no se ha salida.
- **PSW**: Este registro contiene los bits de código de condición, que se asignan cada vez que se ejecutan las instrucciones de comparación, la

prioridad de la CPU, el modo (usuario o kernel) y varios otros bits de control. El S.O. debe estar al tanto de todos los registros.

Canalización: Capacidad de ejecutar más de una instrucción al mismo tiempo.
Multihilamiento: Permite que la CPU contenga el estado de varios hilos de ejecución distintos y luego alterne entre uno y otro con una escala de tiempo en nanosegundos. Este no ofrece un verdadero paralelismo, solo hay un proceso en ejecución a la vez.

Multihilamiento:Permite que la CPU contenga el estado de varios hilos de ejecución distintos y luego alterne entre uno y otro con una escala de tiempo en nanosegundos. Este no ofrece un verdadero paralelismo, solo hay un proceso en ejecución a la vez.

Memoria: Una memoria debe ser en extremo rápida (más rápida que la velocidad de ejecución de una instrucción, de manera que la memoria no detenga a la CPU), de gran tamaño y muy económica. Actualmente, no existe ninguna tecnología que cumpla con todos esos objetivos. Por ello, se adoptó un sistema de jerarquía de capas.



La capa superior consiste en los registros internos de la CPU. El siguiente nivel es la memoria caché, que el hardware controla de manera parcial.
La memoria principal se divide en líneas de caché. Las líneas de caché que se utilizan con más frecuencia se mantienen en una caché de alta velocidad, ubicada dentro o muy cerca de la CPU. Cuando el programa necesita leer una palabra de memoria, el hardware de la caché comprueba si la línea que se requiere se encuentra en la caché. Si es así (a lo cual se le conoce como acierto de caché), la petición de la caché se cumple y no se envía una petición de memoria a través del bus hacia la memoria principal.

Discos:
El siguiente lugar en la jerarquía corresponde al disco magnético (disco duro). Muchas computadoras presentan un esquema conocido como memoria virtual. Este esquema hace posible la ejecución de programas más grandes que la memoria física al colocarlos en el disco y utilizar la memoria principal como un tipo de caché para las partes que se ejecutan con más frecuencia.
Estos discos se pueden particionar, siendo estas particiones espacios de almacenamiento independientes contenidos dentro de un mismo disco duro. Esto genera una tabla de particiones, la cual brinda información de las particiones, indicando desde y hasta donde va la partición (1,x,y).
Tipos de particiones:

- **Primaria**: Están declaradas en la tabla de particiones y están diseñadas para contener datos
- **Lógicas**: Están declaradas en la tabla de particiones y están diseñadas para contener más particiones
- **Extendidas**: No están declaradas en la ta tabla de particiones y contienen más particiones después de la lógica y permiten guardar datos

Cintas: La última capa de la jerarquía en la memoria es la cinta magnética. Este medio se utiliza con frecuencia como respaldo para el almacenamiento en disco y para contener conjuntos de datos muy extensos. Se manipulan de manera secuencial y tienen un bajo costo.

Dispositivos de E/S: Los dispositivos de E/S generalmente constan de dos partes:
Un dispositivo controlador: Es un chip o conjunto de chips que controla físicamente el dispositivo. El software que se comunica con un dispositivo controlador, que le proporciona comandos y acepta respuestas, se conoce como driver (controlador). Cada fabricante de dispositivos controladores tiene que suministrar un driver específico para cada sistema operativo en que pueda funcionar.
El dispositivo en sí: Estos tienen interfaces bastante simples, debido que no puede hacer mucho y también para estandarizarlas.

Buses: Es un mecanismo de comunicación que une los dispositivos electrónicos en una computadora. Esta conexión les permite intercambiar información entre ellos, permitiendo así al procesador realizar operaciones y almacenar y recuperar datos.
Los dos buses principales son **ISA** y **PCI**

Plug and Play
Su función es permitir que el sistema recolecte automáticamente la información acerca de los dispositivos de E/S, asigne los niveles de interrupción y las direcciones de E/S de manera central, para que después indique a cada tarjeta cuáles son sus números.

Arranque de la computadora:

Cada procesador contiene una tarjeta madre, en la cual se encuentra un programa conocido como BIOS. Este contiene software E/S de bajo nivel y está contenido en una RAM tipo flash que es no volátil.
Cuando arranca la computadora, el BIOS inicia su ejecución, comprueba el estado del hardware (cantidad de memoria RAM instalada, teclado y otros dispositivos funcionan) y explora los buses ISA y PCI para detectar todos los dispositivos conectados a ellos. Luego determina el dispositivo de arranque (prueba una lista de dispositivos almacenada en la memoria CMOS), con este se arranca el sistema operativo.
Luego el sistema operativo consulta al BIOS para obtener la información específica, para ser cargados en el kernel, y finalmente, inicializa sus tablas, crea los procesos de segundo plano que se requieran, y arranca un programa de inicio de sesión o GUI.
Podríamos ordenarlo secuencialmente de la siguiente manera:

1) BIOS comprueba estado del hardware
2) BIOS explora buses ISA y PCI
3) BIOS determina dispositivo de arranque del S.O
4) S.O consulta al BIOS información específica para cargar en el kernel
5) Arranca un programa de inicio de sesión o GUI

Tipos de Sistemas Operativos:

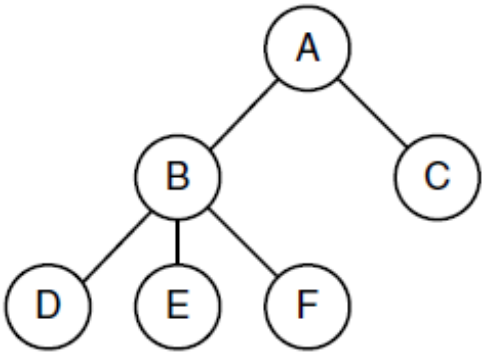
1. Mainframe
2. Servidores
3. Multiprocesadores
4. De computadoras personales
5. De computadoras de bolsillo
6. Integrados
7. De Sensores
8. De tiempo real
9. De tarjetas inteligentes

Conceptos varios en los SO:

Procesos

Un proceso es en esencia un programa en ejecución, es un recipiente que guarda toda la información necesaria para ejecutar un programa.
Cada proceso tiene asociado un espacio de direcciones, una lista de ubicaciones de memoria que va desde algún mínimo (generalmente 0) hasta cierto valor máximo, donde el proceso puede leer y escribir información. También hay asociado a cada proceso un conjunto de recursos (registros, lista de archivos abiertos, alarmas pendientes, lista de procesos relacionados, etc). Posee toda la información que necesita un programa para correr

En muchos sistemas operativos, toda la información acerca de cada proceso (además del contenido de su propio espacio de direcciones) se almacena en una tabla del sistema operativo, conocida como la tabla de procesos, la cual es un arreglo de estructuras, una para cada proceso que se encuentre actualmente en existencia.
Así, un proceso (suspendido) consiste en su espacio de direcciones, que se conoce comúnmente como imagen de núcleo y su entrada en la tabla de procesos, que guarda el contenido de sus registros y muchos otros elementos necesarios para reiniciar el proceso más adelante.
Los procesos relacionados que cooperan para realizar un cierto trabajo a menudo necesitan comunicarse entre sí y sincronizar sus actividades. A esta comunicación se le conoce como comunicación entre procesos.



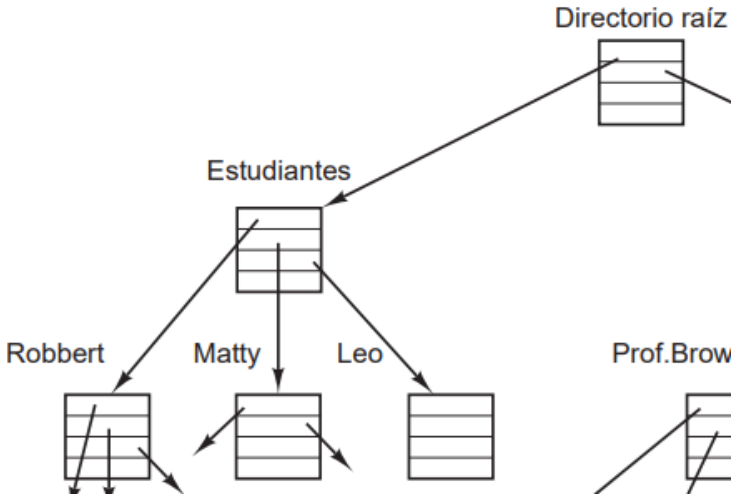
Cada persona autorizada para utilizar un sistema recibe una UID (User Identification, Identificación de usuario) que el administrador del sistema le asigna. Cada proceso iniciado tiene el UID de la persona que lo inició. Un proceso hijo tiene el mismo UID que su padre. Los usuarios pueden ser miembros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

Identificación de grupo). Una UID conocida como superusuario (superuser en UNIX) tiene poder especial y puede violar muchas de las reglas de protección.

Espacio de Direcciones
Cada proceso tiene cierto conjunto de direcciones que puede utilizar, que generalmente van desde 0 hasta cierto valor máximo. Si un proceso tiene más espacio de direcciones que la memoria principal entra en juego lo que conocemos como memoria virtual, en la cual el S.O. mantiene una parte del espacio de direcciones en memoria principal y otra parte en el disco, moviendo pedazos de un lugar a otro según sea necesario.

Archivos
Para proveer un lugar en donde se puedan mantener los archivos, la mayoría de los sistemas operativos tienen el concepto de un directorio como una manera de agrupar archivos.
Un directorio es simplemente un archivo que contiene un conjunto de pares, sus entradas pueden ser archivos u otros directorios.

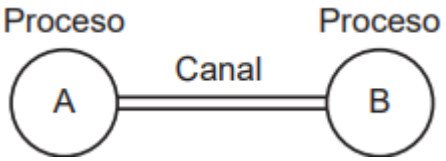
Este modelo también da surgimiento a una jerarquía (el sistema de archivos).



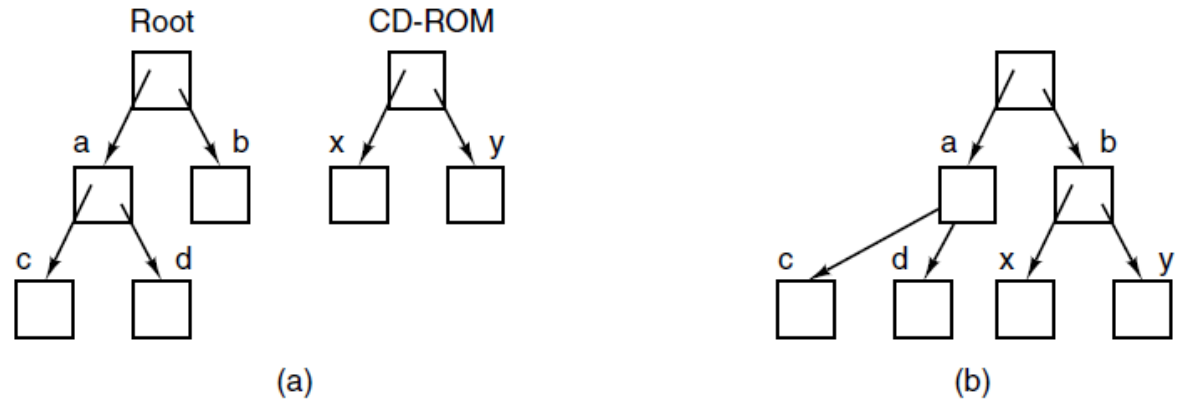
Para especificar cada archivo dentro de la jerarquía de directorio, se proporciona su nombre de ruta de la parte superior de la jerarquía de directorios, el directorio raíz.
Existen 2 tipos de rutas:

1) Rutas absolutos: Lista de directorios que deben recorrerse desde el directorio raíz para llegar al archivo, utilizando "/" como separador.
2) Rutas Relativas: no parten del directorio raíz y, además no inician con (/).

El descriptor de archivos es quien comprueba los permisos a la hora de querer abrir un archivo. Este es un número.
Los archivos especiales se proporcionan para poder hacer que los dispositivos de E/S se vean como archivos.
Existen dos **tipos de archivos especiales**: archivos especiales de bloque y archivos especiales de carácter. Los archivos especiales de bloque se utilizan para modelar dispositivos que consisten en una colección de bloques direccionables al azar, tales como discos.
Los archivos especiales de carácter se utilizan para modelar impresoras, módems y otros dispositivos que aceptan o producen como salida un flujo de caracteres.
Un canal (pipe) es un tipo de pseudoarchivo que puede utilizarse para conectar dos procesos



Montar/Desmontar
Para montar un sistema de archivos en algún directorio debemos usar la llamada al sistema 'mount', casi siempre los sistemas de archivos se montan en directorios. Esta permite combinar dos sistemas de archivos en uno. Lo que hace es asociar a un directorio un dispositivo. En contraste, para desmontar debemos desasociar.



Entrada/Salida:
Existe una parte del sistema operativo que se encarga de la administración de los dispositivos de E/S. Independiente del tipo de dispositivo.

La funcionalidad del driver depende del tipo de dispositivo de E/S, no así, el subsistema de E/S del Sistema Operativo

Protección
Las computadoras contienen grandes cantidades de información que los usuarios comúnmente desean proteger y mantener de manera confidencial. Es responsabilidad del sistema operativo administrar la seguridad del sistema

Shell
Este es un intérprete de comando, es la interfaz principal entre un usuario sentado en su terminal y el sistema operativo. El shell tiene la terminal como entrada estándar y salida estándar

LLamadas al Sistema:
Las llamadas al sistema son las solicitudes que realizan las aplicaciones al hardware, estas entran al kernel.

- Para la administración de procesos

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

Fork devuelve un valor, que es cero en el hijo e igual al identificador del proceso (PID) hijo en el padre. Mediante el uso del PID devuelto, ambos procesos pueden ver cuál es el padre y cuál el hijo. Después de está llamada el hijo tendrá que ejecutar código distinto al del padre. En UNIX los procesos tienen su memoria dividida en tres segmentos: el segmento de texto (es decir, el código del programa), el segmento de datos (es decir, las variables) y el segmento de pila.

- Para la administración de archivos

Administración de archivos

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

Con cada archivo hay un apuntador asociado, el cual indica la posición actual en el archivo.

- Para la administración del sistema de directorios y archivos

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

La llamada al sistema 'chmod' hace posible modificar el modo de un archivo. La llamada al sistema 'kill' es la forma en que los usuarios y los procesos de usuario envían señales.

- La API Win32 de Windows.
  Windows y UNIX difieren de una manera fundamental en sus respectivos modelos de programación. Un programa de UNIX consiste en código que realiza una cosa u otra, haciendo llamadas al sistema para realizar ciertos servicios. En contraste, un programa de Windows es por lo general manejado por eventos. El programa principal espera a que ocurra cierto evento y después llama a un procedimiento para manejarlo

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

ESTRUCTURA DE UN SISTEMA OPERATIVO

Sistemas monolíticos

En este todo el sistema operativo se ejecuta como un solo programa en modo kernel. El S.O. se escribe como una colección de procedimientos, enlazados entre sí en un solo programa binario ejecutable extenso, donde cada procedimiento en el sistema tiene la libertad de llamar a cualquier otro.
Estructura básica:
1. Un programa principal que invoca el procedimiento de servicio solicitado.
2. Un conjunto de procedimientos de servicio que llevan a cabo las llamadas al sistema.
3. Un conjunto de procedimientos utilitarios que ayudan a los procedimientos de servicio.
En este modelo, para cada llamada al sistema hay un procedimiento de servicio que se encarga de la llamada y la ejecuta

Sistemas de capas

Consiste en organizar el sistema operativo como una jerarquía de capas, cada una construida encima de la que tiene abajo.

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

Es una serie de anillos concéntricos, en donde los interiores tenían más privilegios que los exteriores (que en efecto viene siendo lo mismo).
Cuando un procedimiento en un anillo exterior quería llamar a un procedimiento en un anillo interior, tenía que hacer el equivalente de una llamada al sistema; es decir, una instrucción TRAP c.

Microkernels

Su idea básica es lograr una alta confiabilidad al dividir el sistema operativo en módulos pequeños y bien definidos, sólo uno de los cuales (el microkernel) se ejecuta en modo kernel y el resto se ejecuta como procesos de usuario ordinarios, sin poder relativamente. En especial, al ejecutar cada driver de dispositivo y sistema de archivos como un proceso de usuario separado. Al tener un kernel más reducido se reducen las fallas críticas del Sistema Operativo (ya que estas ocurren ahí).
Un servidor interesante es el **servidor de reencarnación**, cuyo trabajo es comprobar si otros servidores y drivers están funcionando en forma correcta. Si se detecta uno defectuoso, se reemplaza automáticamente sin intervención del usuario. Consiguiendo que el sistema sea autocorregible y pueda lograr una alta confiabilidad.

Modelo cliente-servidor

Esta es una variación de la idea del microkernel que diferencia dos procesos: los servidores (quienes proporcionan ciertos servicios), y los clientes (que utilizan los servicios).
La comunicación entre estos se lleva a cabo por mediante el paso de mensajes.
Se dice que este es una abstracción que se puede utilizar para un solo equipo o para una red de equipos.

Máquinas virtuales

Son copias exactas del hardware, incluyendo el modo kernel/ usuario, la E/S, las interrupciones y todo lo demás que tiene la máquina real, con el detalle de que cada una puede ejecutar cualquier sistema operativo que se ejecute directamente sólo en el hardware. Distintas máquinas virtuales pueden ejecutar distintos sistemas operativos. Los sistemas interactivos de un solo usuario llamados CMS (Sistema monitor conversacional) se encargan de atrapar las llamadas al sistema del SO virtual y emularlas. Exokernels Consiste en particionar la máquina actual, donde a cada usuario se le proporciona un subconjunto de los recursos. Su trabajo es asignar recursos a las máquinas virtuales y después comprobar los intentos de utilizarlos, para asegurar que ninguna máquina trate de usar los recursos de otra.

Procesos

Un **proceso** no es más que una instancia de un programa en ejecución, incluyendo los valores actuales del contador del programa, los registros y las variables.
En concepto, cada proceso tiene su propia CPU virtual; en la realidad, la CPU real conmuta de un proceso a otro, esta conmutación rápida de un proceso a otro se lo conoce como **multiprogramación**, aunque en cualquier instante, la CPU solo está ejecutando un solo proceso.

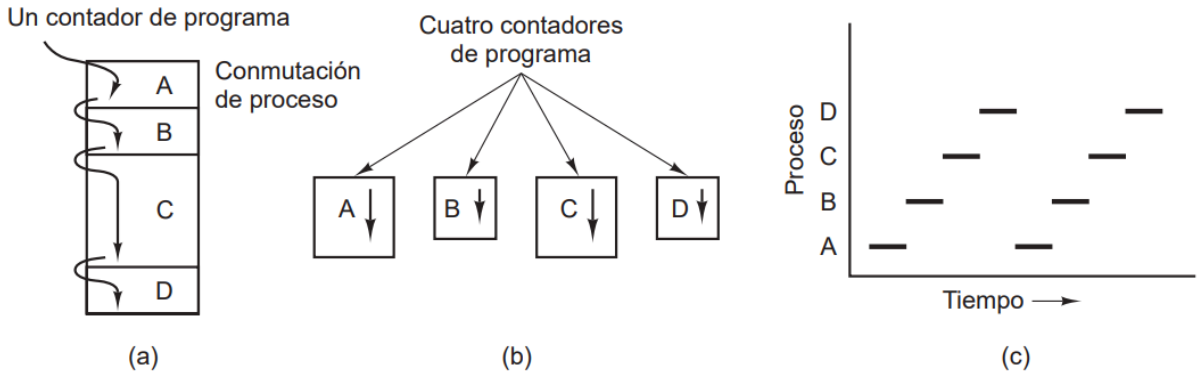


Figura 2-1. (a) Multiprogramación de cuatro programas. (b) Modelo conceptual de cuatro procesos secuenciales independientes. (c) Sólo hay un programa activo a la vez.

Hay **cuatro eventos principales** que provocan la **creación de procesos**:

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

Existen dos tipos de procesos:

1) **En primer plano**, que son los que interactúan con los usuarios y realizan trabajo para ellos.
2) **En segundo plano** (demonios), que no están asociados con usuarios específicos sino con una función específica.

En todos estos casos, se ejecutan llamadas al sistema (ya sea por el usuario o por otro proceso) para crear el proceso.

En UNIX, sólo existe una llamada al sistema para crear un proceso: **fork**. Esta llamada al sistema, crea un clon exacto del proceso que hizo la llamada. Después de fork, los dos procesos (padre e hijo) tienen la misma imagen de memoria, las mismas cadenas de entorno y los mismos archivos abiertos.
Por el contrario, en Windows una sola llamada a una función de Win32 (**CreateProcess**) maneja la creación de procesos y carga el programa correcto en el nuevo proceso.
Tanto en UNIX como en Windows, una vez que se crea un proceso, el padre y el hijo tienen sus propios espacios de direcciones distintos. Si cualquiera de los procesos modifica una palabra en su espacio de direcciones, esta modificación no es visible para el otro proceso.

Pasos en la creación de un proceso

1) Asignar Pid
2) Reservar Espacio
3) Inicializar BCP
4) Establecer enlaces
5) Creación o expansión de estructuras de datos

Llegará un momento, donde estos **procesos terminan**, por lo general, esto ocurre debido a las siguientes **condiciones**:

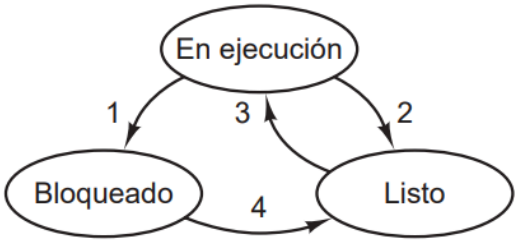
1. **Salida normal** (voluntaria).
   Simplemente han concluido con su trabajo.
2. **Salida por error** (voluntaria).
   El proceso ha descubierto un error y decide salir.
3. **Error fatal** (involuntaria).
   Esta es producida por el proceso, a menudo debido a un error en el programa.
4. **Eliminado por otro proceso** (involuntaria).
   Se puede dar si se ejecute una llamada al sistema que indique que elimine otros procesos (kill/TerminateProcess).

En algunos sistemas, cuando un proceso crea otro, el proceso padre y el proceso hijo continúan asociados en ciertas formas. El proceso hijo puede crear por sí mismo más procesos, formando una **jerarquía de procesos**. Un proceso sólo tiene un padre (pero cero, uno, dos o más hijos).
En UNIX, un proceso y todos sus hijos, junto con sus posteriores descendientes, forman un grupo de procesos.

En contraste, **Windows no tiene un concepto de una jerarquía de procesos**. Todos los procesos son iguales. La única sugerencia de una jerarquía de procesos es que, cuando se crea un proceso, el padre recibe un indicador especial un token (llamado **manejador**) que puede utilizar para controlar al hijo. Sin embargo, tiene la libertad de pasar este indicador a otros procesos, con lo cual invalida la jerarquía. **Los procesos en UNIX no pueden desheredar a sus hijos.**

Estados de los Procesos:

1. **En ejecución** (en realidad está usando la CPU en ese instante).
2. **Listo** (ejecutable; se detuvo temporalmente para dejar que se ejecute otro proceso).
3. **Bloqueado** (no puede ejecutarse sino hasta que ocurra cierto evento externo).



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

Figura 2-2. Un proceso puede encontrarse en estado "en ejecución", "bloqueado" o "listo". Las transiciones entre estos estados son como se muestran.

- La transición 1 ocurre cuando el sistema operativo descubre que un proceso no puede continuar justo en ese momento.
- La transición 2 ocurre cuando el planificador decide que el proceso en ejecución se ha ejecutado el tiempo suficiente y es momento de dejar que otro proceso tenga una parte del tiempo de la CPU.
- La transición 3 ocurre cuando todos los demás procesos han tenido su parte del tiempo de la CPU y es momento de que el primer proceso obtenga la CPU para ejecutarse de nuevo.
- La transición 4 ocurre cuando se produce el evento externo por el que un proceso estaba esperando.

Estados en el otro libro:

• **Listo**: El proceso está en memoria principal y listo para la ejecución.
• **Bloqueado**: El proceso está en memoria principal esperando un suceso.
• **Bloqueado y suspendido**: El proceso está en memoria secundaria esperando un suceso.
• **Listo y suspendido**: El proceso está en memoria secundaria, pero está disponible para su ejecución tan pronto como se cargue en la memoria principal.

Implementación de los procesos

Para implementar el modelo de procesos, el sistema operativo mantiene una tabla (un arreglo de estructuras) llamada **tabla de procesos**, con sólo una entrada por cada proceso (algunos autores llaman a estas entradas **bloques de control de procesos**). Esta entrada contiene **información** importante acerca del **estado del proceso** **(dat**, es decir, todo lo que debe guardarse acerca del proceso para que cuando éste cambia de estado, se pueda reiniciar posteriormente como si nunca se hubiera detenido.

Esta colección de programa, datos, pila y atributos puede llamarse **imagen del proceso**:

Datos de Usuario: La parte modificable del espacio de usuario. Puede guardar datos del programa, una zona para una pila del usuario y programas que pueden modificarse.

Programa de Usuario: El programa a ejecutar.

Pila del Sistema: Cada proceso tiene una o más pilas (el último que entra es el primero en salir) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno.

Bloque de Control de Proceso: Información necesaria para que el sistema operativo controle al proceso

- Identificación del proceso (Pid, Ppid, UID,)
- Información del estado del procesador (Puntero Pila, registros de procesador)
- Información de control del proceso (Estado y planificación, prioridad, etc.)

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		

Figura 2-4. Algunos de los campos de una entrada típica en la tabla de procesos.

Con cada clase de E/S hay una ubicación asociada (por lo general, en una ubicación cerca de la parte final de la memoria), a la cual se le llama vector de interrupción. Esta ubicación contiene la dirección del procedimiento del servicio de interrupciones.

Modos de Ejecución - Funciones Modo Kernel

TABLA 3.10 Funciones Básicas del Núcleo de un Sistema Operativo

Gestión de Procesos

- Creación y terminación de los procesos
- Planificación y expedición de los procesos
- Cambio de procesos
- Sincronización de procesos y soporte para la comunicación entre procesos
- Gestión de los bloques de control de procesos

Gestión de memoria

- Asignación de espacios de direcciones a los procesos
- Intercambio
- Gestión de páginas y segmentos

Gestión de E/S

- Gestión de buffers
- Asignación de canales de E/S y dispositivos a los procesos

Funciones de Soporte

- Tratamiento de interrupciones
- Contabilidad
- Supervisión

Modelación de la Multiprogramación (salteado)

Hilos

En los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones y un solo hilo de control. Sin embargo, hay situaciones en las que es conveniente tener varios hilos de control en el mismo espacio de direcciones que se ejecuta en cuasi-paralelo, como si fueran procesos (casi) separados (excepto por el espacio de direcciones compartido).
Existen varias razones de tener estos **miniprocesos**, conocidos como hilos.

Uso de hilos

Razones para tener hilos:
1. La habilidad de las entidades en paralelo de compartir un espacio de direcciones y todos sus datos entre ellas.
2. Como son más ligeros que los procesos, son más fáciles de crear y destruir.
3. Los hilos no producen un aumento en el rendimiento, pero cuando hay una cantidad considerable de cálculos y operaciones de E/S, al tener hilos estas actividades se pueden traslapar, con lo cual se agiliza la velocidad de la aplicación
4. Estos son útiles en los sistemas con varias CPU, en donde es posible el verdadero paralelismo.

Si tenemos varios hilos en un proceso, estos comparten una memoria común y todos tienen acceso a dicha memoria, las modificaciones de uno se verán reflejadas en los otros.

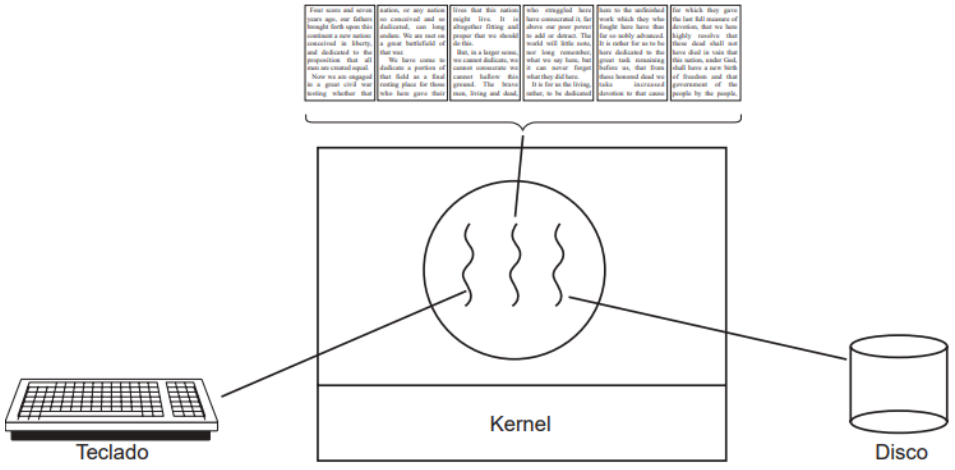


Figura 2-7. Un procesador de palabras con tres hilos.

El primer hilo interactúa con el usuario y el segundo se encarga de volver a dar formato en segundo plano, el tercer hilo se encarga de los respaldos en disco sin interferir con los otros dos.
Debemos aclarar que aquí no funcionaría tener tres procesos separados, ya que los tres hilos necesitan operar en el documento. Al tener tres hilos en vez de tres procesos, comparten una memoria común y por ende todos tienen acceso al documento que se está editando.

El modelo clásico de hilo

El modelo de procesos se basa en dos conceptos independientes: agrupamiento de recursos y ejecución. Algunas veces es útil separarlos; aquí es donde entran los hilos.

Una manera de ver a un proceso es como si fuera una forma de agrupar recursos relacionados.

El modelo de procesos se basa en dos conceptos independientes:

Agrupaciones de recursos: Un proceso tiene un espacio de direcciones que contiene texto y datos del programa, así como otros recursos.

Hilo de ejecución: En general solo se le llama hilo. El hilo tiene un contador de programa que lleva el registro de cuál instrucción se va a ejecutar a continuación. Tiene registros que contienen sus variables de trabajo actuales. Tiene una pila, que contiene el historial de ejecución, con un conjunto de valores para cada procedimiento al que se haya llamado, pero del cual no se haya devuelto todavía

Los procesos se utilizan para agrupar los recursos; son las entidades planificadas para su ejecución en la CPU.

Lo que agregan los hilos al modelo de procesos es permitir que se lleven a cabo varias ejecuciones en el mismo entorno del proceso, que son en gran parte independientes unas de las otras.

Como los hilos tienen algunas de las propiedades de los procesos, algunas veces se les llama procesos ligeros. El término multihilamiento también se utiliza para describir la situación de permitir varios hilos en el mismo proceso.

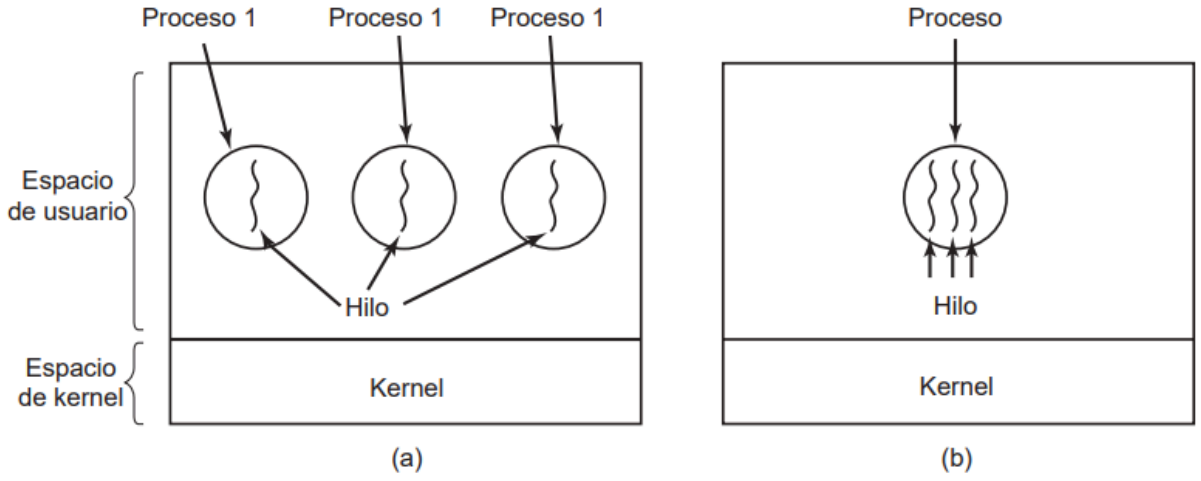


Figura 2-11. (a) Tres procesos, cada uno con un hilo. (b) Un proceso con tres hilos.

- Cuando se ejecuta un proceso con multihilamiento en un sistema con una CPU, los hilos toman turnos para ejecutarse, de igual manera que la

multiprogramación, es decir, la CPU conmuta rápidamente entre un hilo y otro, dando la ilusión de que los hilos se ejecutan en paralelo.

Los distintos hilos en un proceso no son tan independientes como los procesos. No hay protección entre los hilos debido a que (1) es imposible y (2) no debe ser necesario.

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	

Figura 2-12. La primera columna lista algunos elementos compartidos por todos los hilos en un proceso; la segunda, algunos elementos que son privados para cada hilo.

Un hilo puede estar en uno de varios estados:

- **En ejecución**: Tiene la CPU en un momento dado y está activo
- **Bloqueado**: Está esperando a que cierto evento lo desbloquee
- **Listo**: Se programa para ejecutarse y lo hará tan pronto como sea su turno
- **Terminado**: Terminó su trabajo y salio mediante una llamada

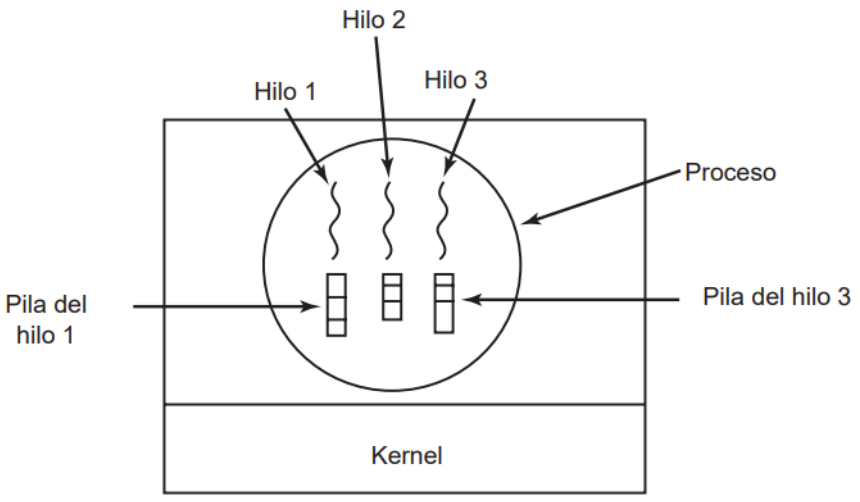


Figura 2-13. Cada hilo tiene su propia pila.

Por lo general los procesos empiezan con un solo hilo presente. Este hilo tiene la habilidad de crear hilos mediante la llamada a un procedimiento de biblioteca,

como thread_create. Algunas veces los hilos son jerárquicos, con una relación padre-hijo, pero a menudo no existe dicha relación y todos los hilos son iguales. Aún así el hilo creador generalmente recibe un identificador de hilo que da nombre al nuevo hilo.

Comunicación entre Procesos

Los procesos necesitan comunicarse con otros procesos.
Hay tres cuestiones aquí. La primera alude a cómo un proceso puede pasar información a otro. La segunda está relacionada con hacer que dos o más procesos no se interpongan entre sí, la tercera trata acerca de obtener la secuencia apropiada cuando hay dependencias presentes.

Condiciones de Carrera

En algunos sistemas operativos, los procesos que trabajan en conjunto pueden compartir cierto espacio de almacenamiento en el que pueden leer y escribir datos. El almacenamiento compartido puede estar en la memoria principal (posiblemente en una estructura de datos del kernel) o puede ser un archivo compartido.

Cuando dos o más procesos están leyendo o escribiendo algunos datos compartidos y el resultado final depende de quién se ejecuta y exactamente cuándo lo hace, se conocen como **Condiciones de Carrera** (Los procesos compiten por los recursos físicos y lógicos).

La forma de evitar dicho problema se conoce como **exclusión mutua**, y consiste en prohibir que más de un proceso lea y escriba datos al mismo tiempo.

Esa parte del programa en la que se accede a la memoria compartida se conoce como **región crítica** o **sección crítica**.

Si pudiéramos ordenar las cosas de manera que dos procesos nunca estuvieran en sus regiones críticas al mismo tiempo, podríamos evitar las carreras. Aunque este requerimiento evita las condiciones de carrera, no es suficiente para que los procesos en paralelo cooperen de manera correcta y eficiente al utilizar datos compartidos.

Necesitamos cumplir con cuatro condiciones para tener una buena solución:

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

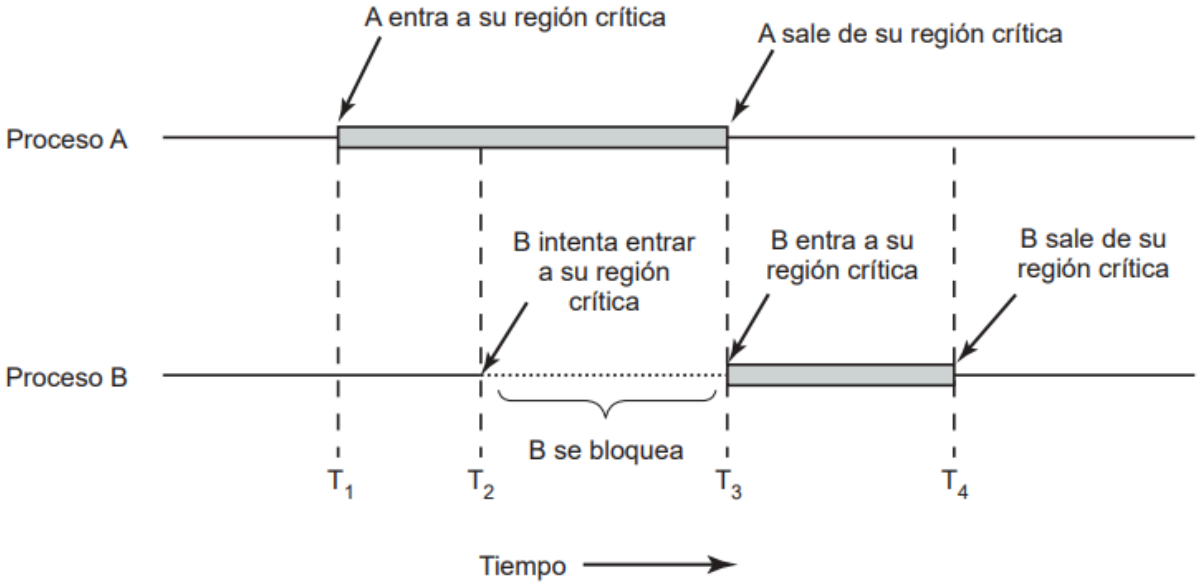


Figura 2-22. Exclusión mutua mediante el uso de regiones críticas.

Exclusión Mutua con espera ocupada

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

Deshabilitando interrupciones: En un sistema con un solo procesador, la solución más simple es hacer que cada proceso deshabilite todas las interrupciones justo después de entrar a su región crítica y las rehabilite justo después de salir. Con las interrupciones deshabilitadas, no pueden ocurrir interrupciones de reloj.

Por lo general este método es poco atractivo, ya que no es conveniente dar a los procesos de usuario el poder para desactivar las interrupciones.

En un multinúcleo (es decir, sistema con multiprocesadores) al deshabilitar las interrupciones de una CPU no se evita que las demás CPUs interfieran con las

operaciones que la primera CPU está realizando. En consecuencia, se requieren esquemas más sofisticados.

Variables de candado: Considere tener una sola variable compartida (de candado), que al principio es 0. Cuando un proceso desea entrar a su región crítica primero evalúa el candado. Si este candado es 0, el proceso lo fija en 1 y entra a la región crítica. Si el candado ya es 1 sólo espera hasta que el candado se haga 0. Por ende, un 0 significa que ningún proceso está en su región crítica y un 1 significa que algún proceso está en su región crítica. Por desgracia, tiene el error de que en un momento dado dos procesos pueden estar al mismo tiempo (si ambos llegan y entran en 0).

Alternancia estricta: La variable entera turno (que al principio es 0) lleva la cuenta acerca de a qué proceso le toca entrar a su región crítica y examinar o actualizar la memoria compartida. Al principio, el proceso 0 inspecciona turno, descubre que es 0 y entra a su región crítica. El proceso 1 también descubre que es 0 y por lo tanto se queda en un ciclo estrecho, evaluando turno en forma continua para ver cuándo se convierte en 1 (usando la espera ocupada). Tomar turnos no es una buena idea cuando uno de los procesos es mucho más lento que el otro.
A la acción de evaluar en forma continua una variable hasta que aparezca cierto valor se le conoce como **espera ocupada**. Por lo general se debe evitar, ya que desperdicia tiempo de la CPU. La espera ocupada sólo se utiliza cuando hay una expectativa razonable de que la espera será corta. A un candado que utiliza la espera ocupada se le conoce como **candado de giro**.
Esta situación viola la condición 3 antes establecida: el proceso 0 está siendo bloqueado por un proceso que no está en su región crítica. Volviendo al directorio de spooler antes mencionado, si ahora asociamos la región crítica con la lectura y escritura del directorio de spooler, el proceso 0 no podría imprimir otro archivo debido a que el proceso 1 está haciendo otra cosa

Solución Peterson: Al combinar la idea de tomar turnos con la idea de las variables de candado y las variables de advertencia, se idea una solución de software para el problema de la exclusión mutua que no requiere de una alternancia estricta.
Al principio ningún proceso se encuentra en su región crítica. Ahora el proceso 0 llama a entrar_region. Indica su interés estableciendo su elemento del arreglo y fija turno a 0. Como el proceso 1 no está interesado, entrar_region regresa de inmediato. Si ahora el proceso 1 hace una llamada a entrar_region, se quedará

ahí hasta que interesado[0] sea FALSE, un evento que sólo ocurre cuando el proceso 0 llama a salir_region para salir de la región crítica.
Ahora considere el caso en el que ambos procesos llaman a entrar_region casi en forma simultánea. Ambos almacenarán su número de proceso en turno. Cualquier almacenamiento que se haya realizado al último es el que cuenta; el primero se sobrescribe y se pierde.

Instrucción TSL: Para usar la instrucción TSL necesitamos una variable compartida (candado) que coordine el acceso a la memoria compartida. Cuando candado es 0, cualquier proceso lo puede fijar en 1 mediante el uso de la instrucción TSL y después una lectura o escritura en la memoria compartida. Cuando termina, el proceso establece candado de vuelta a 0 mediante una instrucción move ordinaria. (usando la espera ocupada) Una instrucción alternativa para TSL es XCHG, que intercambia el contenido de dos ubicaciones en forma atómica. (usando la espera ocupada)

Dormir y Despertar
La inversión de prioridades es un caso problemático en la planificación de tareas. Sucede cuando dos tareas de distinta prioridad comparten un recurso, y la tarea de menor prioridad bloquea el recurso antes que la de mayor prioridad. Como resultado, la tarea de mayor prioridad queda bloqueada en el momento en que necesita utilizar el recurso compartido.

Sleep es una llamada al sistema que hace que el proceso que llama se bloquee o desactive, es decir, que se suspenda hasta que otro proceso lo despierte.
La llamada **wakeup** tiene un parámetro, el proceso que se va a despertar o activar.
De manera alternativa, tanto sleep como wakeup tienen un parámetro, una dirección de memoria que se utiliza para asociar las llamadas a sleep con las llamadas a wakeup.

- **Búfer:** Es un espacio temporal de memoria física que se utiliza para almacenar información mientras se envía de un lado a otro.

Semáforos
Un semáforo variable entera para contar el número de señales de despertar, guardadas para un uso futuro.
Un semáforo podría tener el valor 0, indicando que no se guardaron señales de despertar o algún valor positivo si estuvieran pendientes una o más señales de despertar.

Éste posee dos operaciones, **down** y **up**.
La operación **down** en un semáforo comprueba si el valor es mayor que 0. De ser así, disminuye el valor y sólo continúa. Si el valor es 0, el proceso se pone a dormir sin completar la operación down por el momento. Las acciones de comprobar el valor, modificarlo y posiblemente pasar a dormir, se realizan en conjunto como una sola **acción atómica indivisible**. Se garantiza que, una vez que empieza una operación de semáforo, ningún otro proceso podrá acceder al semáforo sino hasta que la operación se haya completado o bloqueado.
La operación **up** incrementa el valor del semáforo direccionado. Si uno o más procesos estaban inactivos en ese semáforo, sin poder completar una operación down anterior, el sistema selecciona uno de ellos (al azar) y permite que complete su operación down. Así, después de una operación up en un semáforo que contenga procesos dormidos, el semáforo seguirá en 0 pero habrá un proceso menos dormido en él. La operación de incrementar el semáforo y despertar a un proceso también es indivisible.

Los semáforos que se inicializan a 1 y son utilizados por dos o más procesos para asegurar que sólo uno de ellos pueda entrar a su región crítica en un momento dado se llaman **semáforos binarios**.
Si cada proceso realiza una operación down justo antes de entrar a su región crítica y una operación up justo después de salir de ella, se garantiza la exclusión mutua.
El **semáforo mutex** se utiliza para la exclusión mutua. El otro uso de los semáforos es para la **sincronización**.

Mutexes
Cuando no se necesita la habilidad del semáforo de contar, se utiliza una versión simplificada, llamada mutex. Los mutexes son buenos sólo para administrar la exclusión mutua para cierto recurso compartido o pieza de código.
Un **mutex** es una variable que puede estar en uno de dos estados: abierto (desbloqueado) o cerrado (bloqueado). En consecuencia, se requiere sólo 1 bit para representarla.

Se utilizan **dos procedimientos** con los mutexes. Cuando un hilo (o proceso) necesita acceso a una región crítica, llama a mutex_lock. Si el mutex está actualmente abierto (lo que significa que la región crítica está disponible), la llamada tiene éxito y entonces el hilo llamador puede entrar a la región crítica. Por otro lado, si el mutex ya se encuentra cerrado, el hilo que hizo la llamada se bloquea hasta que el hilo que está en la región crítica termine y llame a mutex_unlock.
Si se bloquean varios hilos por el mutex, se selecciona uno de ellos al azar y se permite que adquiera el mutex.
Variables de condición: permiten que los hilos se bloqueen debido a que cierta condición no se está cumpliendo. Hay llamadas para crear y destruir variables de condición.
Las variables de condición y los mutexes siempre se utilizan en conjunto. El patrón es que un hilo cierre un mutex y después espere en una variable condicional cuando no pueda obtener lo que desea

Monitores
Un **monitor** es una colección de procedimientos, variables y estructuras de datos que se agrupan en un tipo especial de módulo o paquete. Las **características** básicas de un monitor son las siguientes:

1) Las variables de datos locales están solo accesibles para los procedimientos del monitor y no para procedimientos externos.
2) Un proceso entra en el monitor invocando a uno de sus procedimientos
3) Solo un proceso puede estar ejecutando en el monitor en un instante dado, cualquier otro proceso que haya invocado al monitor quedará suspendido mientras espera a que el monitor esté disponible.

Si se cumple la norma de un proceso cada vez, el monitor puede ofrecer un servicio de **exclusión mutua**.
Al convertir todas las regiones críticas en procedimientos de monitor, nunca habrá dos procesos que ejecuten sus regiones críticas al mismo tiempo.
Para que resulten útiles en el proceso concurrente, los monitores deben incluir herramientas de sincronización.
Un monitor proporciona **sincronización** por medio de las variables de condición que se incluyen dentro del monitor y que son accesibles sólo desde dentro. Hay dos funciones para operar con las variables de condición:

- wait(c): Suspende la ejecución del proceso llamado bajo la condición c. El monitor está ahora disponible para ser usado por otro proceso.

- signal(c): Reanuda la ejecución de algún proceso suspendido después de un wait bajo la misma condición. Si hay varios procesos, elige uno de ellos; si no hay ninguno, no hace nada.

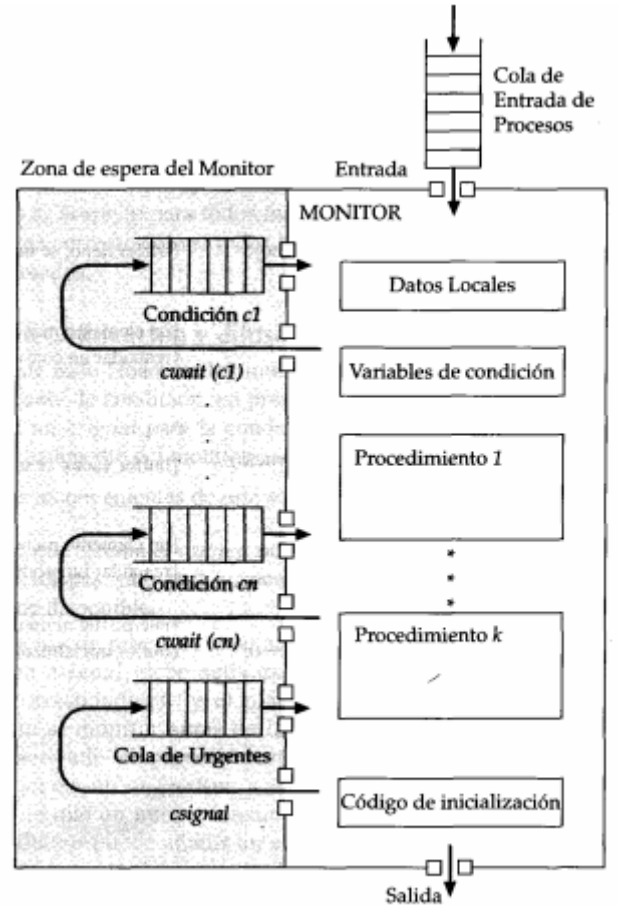


FIGURA 4.23 Estructura de un monitor

La **desventaja** de este radica en que los monitores son un concepto de lenguaje de programación, el cual pocos programas lo utilizan

Pasaje (transmisión) de mensajes

Este método de comunicación entre procesos utiliza dos primitivas (send y receive), las cuales, son llamadas al sistema en vez de construcciones de lenguaje.

La primera llamada envía un mensaje a un destino especificado y la segunda recibe un mensaje de un origen especificado (o de CUALQUIERA, si al receptor no le importa).

Un problema de estos es que se pueden perder mensajes en la red. Para evitar esto se utiliza **acknowledgement** que consiste en que el emisor y receptor acuerden que, tan pronto como haya recibido un mensaje, el receptor enviará de vuelta un mensaje especial de acuse de recibo.

Si el emisor no ha recibido el acuse dentro de cierto intervalo de tiempo, vuelve a transmitir el mensaje.

Suponiendo que el mensaje llega correctamente, pero se pierde el acuse, este problema se resuelve colocando números de secuencia consecutivos en cada mensaje original.
Un **buzón** es un lugar para colocar en el búfer cierto número de mensajes, que por lo general se especifica a la hora de crear el buzón. Cuando se utilizan buzones, los parámetros de dirección en las llamadas a send y receive son buzones, no procesos. Cuando un proceso trata de enviar un buzón que está lleno, se suspende hasta que se remueva un mensaje de ese buzón, haciendo espacio para un nuevo mensaje.

El pasaje de mensajes se utiliza con frecuencia en los sistemas de programación en paralelo.

Barreras
Mecanismo de sincronización destinado a los grupos de procesos. Algunas aplicaciones se dividen en fases y tienen la regla de que ningún proceso puede continuar a la siguiente fase sino hasta que todos los procesos estén listos para hacerlo. Para lograr este comportamiento, se coloca una barrera al final de cada fase. Cuando un proceso llega a la barrera, se bloquea hasta que todos los procesos han llegado a ella.

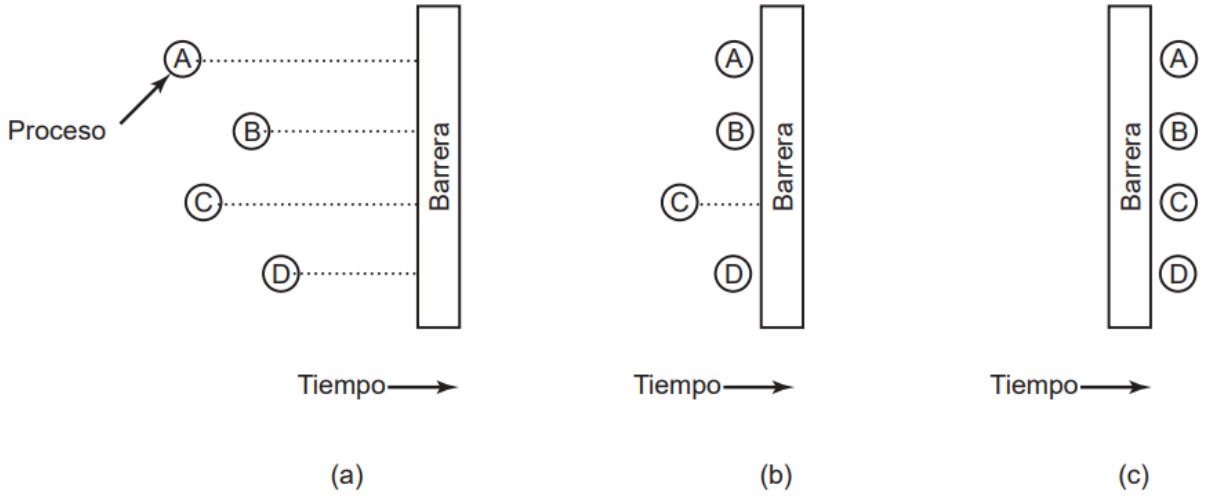


Figura 2-37. Uso de una barrera. (a) Procesos acercándose a una barrera. (b) Todos los procesos, excepto uno, bloqueados en la barrera. (c) Cuando el último proceso llega a la barrera, todos se dejan pasar.

Planificación de Procesos

El **planificador de procesos** es la parte del sistema operativo que realiza la decisión de cuál proceso se va a ejecutar a continuación. Esta decisión se debe a que dos o mas procesos se encuentran al mismo tiempo en el estado listo.
El algoritmo que utiliza esta parte del S.O se lo conoce como **algoritmo de planificación**.
Además de elegir el proceso correcto que se va a ejecutar a continuación, el planificador también tiene que preocuparse por hacer un uso eficiente de la CPU, debido a que la conmutación de procesos es cara.

Comportamiento de un Proceso

Casi todos los procesos alternan ráfagas de cálculos con peticiones de E/S (de disco). Por lo general la CPU opera durante cierto tiempo sin detenerse, después se realiza una llamada al sistema para leer datos de un archivo o escribirlos en el mismo. Cuando se completa la llamada al sistema, la CPU realiza cálculos de nuevo hasta que necesita más datos o tiene que escribir más datos y así sucesivamente.

- **Procesos limitados a cálculos**: Tienen ráfagas de CPU largas y en consecuencia, esperas infrecuentes por la E/S
- **Procesos limitados a E/S (I/O-bound)**: Tienen ráfagas de CPU cortas y por ende, esperas frecuentes por la E/S

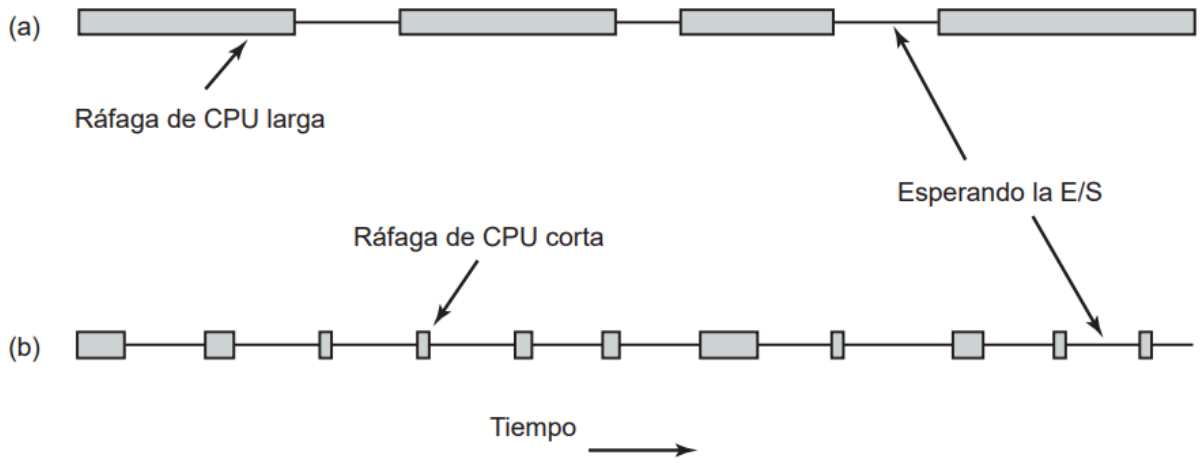


Figura 2-38. Las ráfagas de uso de la CPU se alternan con los periodos de espera por la E/S. (a) Un proceso ligado a la CPU. (b) Un proceso ligado a la E/S.

La idea básica aquí es que, si un proceso limitado a E/S desea ejecutarse, debe obtener rápidamente la oportunidad de hacerlo para que pueda emitir su petición de disco y mantener el disco ocupado

Los **algoritmos de planificación** se pueden dividir en **dos categorías** con respecto a la forma en que manejan las interrupciones del reloj:
No apropiativo: Selecciona un proceso para ejecutarlo y después sólo deja que se ejecute hasta que el mismo se bloquea o hasta que libera la CPU en forma voluntaria. En efecto, no se toman decisiones de planificación durante las interrupciones de reloj.
Apropiativo: Selecciona un proceso y deja que se ejecute por un máximo de tiempo fijo. Si sigue en ejecución al final del intervalo de tiempo, se suspende y el planificador selecciona otro proceso para ejecutarlo.

Categorías de los algoritmos de planificación

Lo que el planificador debe optimizar no es lo mismo en todos los sistemas. Tres de estos entornos que podríamos mencionar son:

1) **Procesamiento por lotes**: En los sistemas de procesamiento por lotes no hay usuarios que esperan impacientemente en sus terminales para obtener una respuesta rápida a una petición corta. En consecuencia, son aceptables los algoritmos no apropiativos (o apropiativos con largos periodos para cada proceso).
2) **Interactivo**: En un entorno con usuarios interactivos, la apropiación es esencial para evitar que un proceso acapare la CPU y niegue el servicio a los demás
3) **De tiempo real**: En los sistemas con restricciones de tiempo real, la apropiación a veces es no necesaria debido a que los procesos saben que no se pueden ejecutar durante periodos extensos, que por lo general realizan su trabajo y se bloquean con rapidez.

Metas de los algoritmos de planificación

Para poder diseñar un algoritmo de programación, es necesario tener cierta idea de lo que debe hacer un buen algoritmo. Algunos objetivos dependen del entorno, pero hay también algunos otros que son deseables en todos los casos.

Todos los sistemas

- **Equidad** - Otorgar a cada proceso una parte justa de la CPU
- **Aplicación de políticas** - Verificar que se lleven a cabo las políticas establecidas
- **Balance** - Mantener ocupadas todas las partes del sistema

Sistemas de procesamiento por lotes

- **Rendimiento** - Maximizar el número de trabajos por hora

- **Tiempo de retorno** - Minimizar el tiempo entre la entrega y la terminación
- **Utilización de la CPU** - Mantener ocupada la CPU todo el tiempo

Sistemas interactivos

- **Tiempo de respuesta** - Responder a las peticiones con rapidez
- **Proporcionalidad** - Cumplir las expectativas de los usuarios

Sistemas de tiempo real

- **Cumplir con los plazos** - Evitar perder datos
- **Predictibilidad** - Evitar la degradación de la calidad en los sistemas multimedia

Métricas de los Algoritmos de Planificación

Procesamiento por lotes

- **Rendimiento**: Es el número de trabajos por hora que completa el sistema.
- **Tiempo de retorno**: Es el tiempo estadísticamente promedio desde el momento en que se envía un trabajo por lotes, hasta el momento en que se completa.
- **Utilización de la CPU**: Es útil poder detectar cuando el uso de la CPU está llegando a 100%, para saber cuándo es momento de obtener más poder de cómputo.

Sistemas interactivos

- **Tiempo de respuesta**: El tiempo que transcurre entre emitir un comando y obtener el resultado
- **Proporcionalidad**: El tiempo que supone el usuario como correcto frente a determinadas tareas

Sistemas en tiempo real

- **Tiempo límite de proceso**

Planificación en Sistemas de Procesamiento por Lotes

Primero en entrar, primero en ser atendido (FCFS)

La CPU se asigna a los procesos en el orden en el que la solicitan. Es no apropiativo. En esencia hay una sola cola de procesos listos. Cuando el primer trabajo entra al sistema desde el exterior en la mañana, se inicia de inmediato y se le permite ejecutarse todo el tiempo que desee.

El trabajo más corto primero (SJF)

Es no apropiativo. Supone que los tiempos de ejecución se conocen de antemano. Cuando hay varios trabajos de igual importancia esperando a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero.

Figura 2-40. Un ejemplo de planificación tipo el trabajo más corto primero. (a) Ejecución de cuatro trabajos en el orden original. (b) Ejecución de cuatro trabajos en el orden del tipo "el trabajo más corto primero".

El menor tiempo restante a continuación (SRTN)

Es una versión apropiativa del algoritmo tipo el trabajo más corto primero. Se debe conocer el tiempo de ejecución de antemano. El planificador siempre selecciona el proceso cuyo tiempo restante de ejecución sea el más corto. Cuando llega un nuevo trabajo, su tiempo total se compara con el tiempo restante del proceso actual. Si el nuevo trabajo necesita menos tiempo para terminar que el proceso actual, éste se suspende y el nuevo trabajo se inicia.

Planificación en sistemas interactivos

Planificación por turno circular (round-robin)

Uno de los algoritmos más antiguos, simples, equitativos y de mayor uso.
A cada proceso se le asigna un intervalo de tiempo, conocido como quántum, durante el cual se le permite ejecutarse. Si el proceso se sigue ejecutando al final del cuanto, la CPU es apropiada para dársela a otro proceso. Si el proceso se bloquea o termina antes de que haya transcurrido el quántum, la conmutación de la CPU se realiza cuando el proceso se bloquea.
Surge un problema al establecer el quantum, ya que, si se establece el quántum demasiado corto se producen demasiadas conmutaciones de procesos y se

reduce la eficiencia de la CPU, pero si se establece demasiado largo se puede producir una mala respuesta a las peticiones interactivas cortas.

Planificación por Prioridad

A cada proceso se le asigna una prioridad y el proceso ejecutable con la prioridad más alta es el que se puede ejecutar. Para evitar que los procesos con alta prioridad se ejecuten de manera indefinida, el planificador puede reducir la prioridad del proceso actual en ejecución en cada pulso del reloj. Si esta acción hace que su prioridad se reduzca a un valor menor que la del proceso con la siguiente prioridad más alta, ocurre una conmutación de procesos.

De manera alternativa, a cada proceso se le puede asignar un quántum de tiempo máximo que tiene permitido ejecutarse. Cuando este quántum se utiliza, el siguiente proceso con la prioridad más alta recibe la oportunidad de ejecutarse.

A las prioridades se les pueden asignar procesos en forma estática o dinámica.

A menudo es conveniente agrupar los procesos en clases de prioridad y utilizar la planificación por prioridad entre las clases, pero la planificación por turno circular dentro de cada clase. Si las prioridades no se ajustan de manera ocasional, todas las clases de menor prioridad pueden morir de hambre (sin tiempo de CPU).

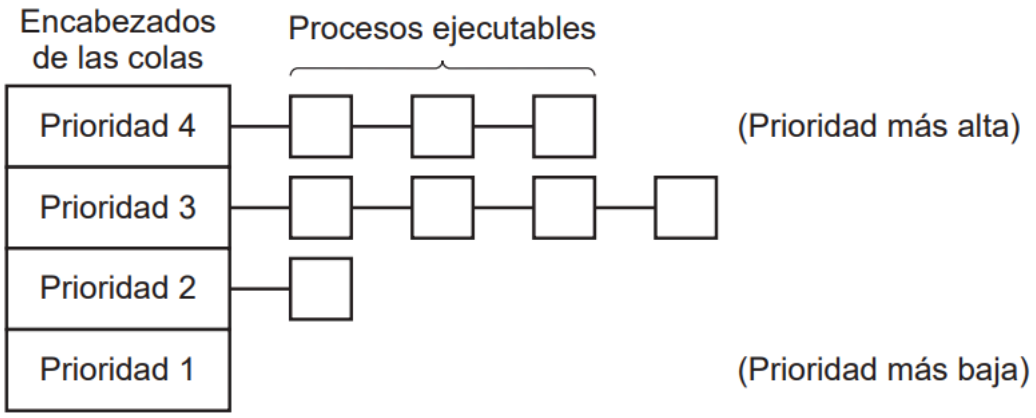


Figura 2-42. Un algoritmo de planificación con cuatro clases de prioridad.

El proceso más corto a continuación

Si consideramos la ejecución de cada comando como un "trabajo" separado, entonces podríamos minimizar el tiempo de respuesta total mediante la ejecución del más corto primero. Utiliza envejecimiento para estimar el siguiente valor en una serie mediante la obtención del promedio ponderado del valor actual medido y la estimación anterior.

Planificación garantizada

Hacer promesas reales a los usuarios acerca del rendimiento y después cumplirlas.

Planificación por sorteo

La idea básica es dar a los procesos boletos de lotería para diversos recursos del sistema, como el tiempo de la CPU. Cada vez que hay que tomar una decisión de planificación, se selecciona un boleto de lotería al azar y el proceso que tiene ese boleto obtiene el recurso.
Los procesos más importantes pueden recibir boletos adicionales, para incrementar su probabilidad de ganar.
La planificación por lotería tiene varias propiedades interesantes:

- Tiene un alto grado de respuesta.
- Los procesos cooperativos pueden intercambiar boletos si lo desean.
- Se puede utilizar para resolver problemas que son difíciles de manejar con otros métodos.

Planificación por partes equitativas

En este modelo, a cada usuario se le asigna cierta fracción de la CPU y el planificador selecciona procesos de tal forma que se cumpla con este modelo.

Planificación en sistemas de tiempo real

Los sistemas de tiempo real se categorizan como de **tiempo real duro**, lo cual significa que hay tiempos límite absolutos que se deben cumplir, y como de **tiempo real suave**, lo cual significa que no es conveniente fallar en un tiempo límite en ocasiones, pero sin embargo es tolerable. En ambos casos, el comportamiento en tiempo real se logra dividiendo el programa en varios procesos, donde el comportamiento de cada uno de éstos es predecible y se conoce de antemano.
Los eventos a los que puede llegar a responder un sistema de tiempo real se pueden categorizar como:
Periódicos, que ocurren en intervalos regulares.
Aperiódicos, que ocurren de manera impredecible
Los algoritmos de planificación en tiempo real pueden ser:
Estáticos: Toman sus decisiones de planificación antes de que el sistema deje de ejecutarse
Dinámicos: Toman sus decisiones de planificación durante la ejecución del sistema. Estos no tienen restricciones

Interbloqueos

Esto sucede cuando dos o más procesos distintos tienen recursos, y sin liberar los que ya tienen, solicitan recursos de otros procesos, provocando así que se bloqueen los procesos, esto se conoce como interbloqueo.

RECURSOS
Nos referiremos a los objetos otorgados como recursos, ya sea una pieza de hardware o una pieza de información.
Un **recurso** es cualquier cosa que se debe **adquirir**, **utilizar** y **liberar** con el transcurso del tiempo.

Recursos apropiativos y no apropiativos
Los recursos son de dos tipos: apropiativos y no apropiativos.
Recurso Apropiativo: Se puede quitar al proceso que lo posee sin efectos dañinos.
Recurso No Apropiativo: No se puede quitar a su propietario actual sin hacer que el cómputo falle.

En general, los interbloqueos **involucran** a los **recursos no apropiativos**. Los interbloqueos potenciales que pueden involucrar a los recursos apropiativos por lo general se pueden resolver mediante la reasignación de los recursos de un proceso a otro.

Para utilizar un recurso tenemos una secuencia de eventos:

1) Solicitar el recurso.
2) Utilizar el recurso.
3) Liberar el recurso.

Si el recurso no está disponible cuando se le solicita, el proceso solicitante se ve obligado a esperar.
Un proceso al que se le ha negado la petición de un recurso por lo general permanece en un ciclo estrecho solicitando el recurso, después pasa al estado inactivo y después intenta de nuevo.

Adquisición de recursos
Una manera de permitir que los usuarios administren los recursos es asociar un **semáforo** o **mutex** con cada recurso.
Algunas veces los procesos necesitan dos o más recursos. Se pueden adquirir de manera **secuencial**, si se necesitan más de dos recursos, sólo se adquieren uno después del otro.
Abrazo Mortal, donde cada uno de los procesos involucrados se bloquea hasta conseguir el recurso que tiene el otro proceso.

INTRODUCCIÓN A LOS INTERBLOQUEOS
El **interbloqueo** se puede definir formalmente de la siguiente manera:

- `Un conjunto de procesos se encuentra en un interbloqueo si cada proceso en el conjunto está esperando un evento que sólo puede ser ocasionado por otro proceso en el conjunto.`

Debido a que todos los procesos están en espera, ninguno de ellos producirá alguno de los eventos que podrían despertar a cualquiera de los otros miembros del conjunto, y todos los procesos seguirán esperando para siempre.
En otras palabras, cada miembro del conjunto de procesos en interbloqueo está esperando un recurso que posee un proceso en interbloqueo. Ninguno de los procesos se puede **ejecutar**, **liberar recursos** y ser **despertado**.
A este tipo de interbloqueo se le conoce como interbloqueo de recursos, es el tipo más común de interbloqueo.

6.2.1 Condiciones para los interbloqueos de recursos

1) **Condición de exclusión mutua.** Cada recurso se asigna en un momento dado a sólo un proceso, o está disponible.
2) **Condición de contención y espera.** Los procesos que actualmente contienen recursos que se les otorgaron antes pueden solicitar nuevos recursos.
3) **Condición no apropiativa.** Los recursos otorgados previamente no se pueden quitar a un proceso por la fuerza. Deben ser liberados de manera explícita por el proceso que los contiene.
4) **Condición de espera circular.** Debe haber una cadena circular de dos o más procesos, cada uno de los cuales espera un recurso contenido por el siguiente miembro de la cadena.

Las **cuatro condiciones deben estar presentes** para que ocurra un interbloqueo. Si una de ellas está ausente, no es posible el interbloqueo de recursos.

Modelado de interbloqueos
Se pueden modelar estas cuatro condiciones mediante el uso de gráficos dirigidos.

Se utilizan dos tipos de nodos: procesos (círculos), y recursos (cuadros).
Si la arista va de un proceso a un recurso significa que el proceso está actualmente bloqueado, en espera de ese recurso, si la arista va de un recurso a proceso significa que el recurso fue solicitado previamente por, y asignado a, y actualmente está contenido por ese proceso.

Un ciclo en el gráfico indica que hay un interbloqueo que involucra a los procesos y recursos en el ciclo (suponiendo que hay un recurso de cada tipo).

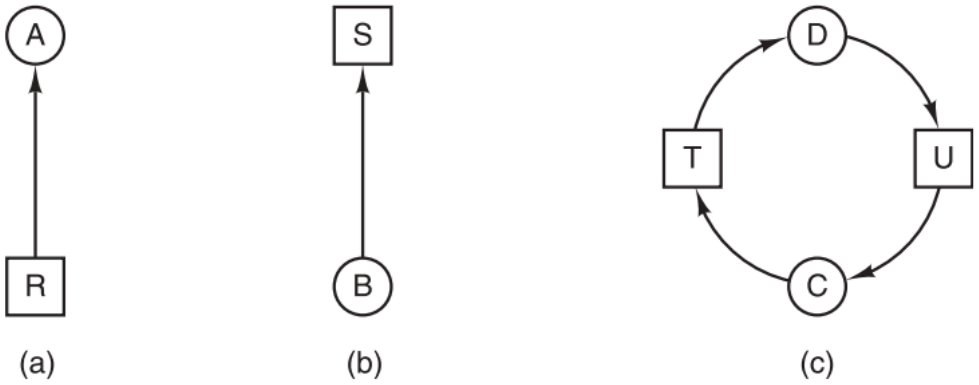


Figura 6-3. Gráficos de asignación de recursos. (a) Contención de un recurso. (b) Petición de un recurso. (c) Interbloqueo.

Cuando los procesos se ejecutan en forma secuencial, no hay posibilidad de que mientras un proceso espera la E/S, otro pueda utilizar la CPU. Por ende, ejecutar los procesos estrictamente en forma secuencial puede no ser óptimo.

Se utilizan cuatro estrategias para lidiar con los interbloqueos:

1) **Sólo ignorar el problema**. Tal vez si usted lo ignora, él lo ignorará a usted. Algoritmo de la Avestruz
2) **Detección y recuperación**. Dejar que ocurran los interbloqueos, detectarlos y tomar acción.
3) **Evitarlos en forma dinámica mediante la asignación cuidadosa de los recursos**.
4) **Prevención**, al evitar estructuralmente una de las cuatro condiciones requeridas.

Algoritmo de la avestruz

El método más simple es el algoritmo de la avestruz: meta su cabeza en la arena y pretenda que no hay ningún problema.

Esta estrategia consiste en ver con qué frecuencia falla el sistema y qué tan grave es un interbloqueo. Si ocurren interbloqueos en promedio de una vez cada cinco años, los interbloqueos no representan un problema inmediato el cual se tenga que resolver.

DETECCIÓN Y RECUPERACIÓN DE UN INTERBLOQUEO

Una segunda técnica es la detección y recuperación. Cuando se utiliza esta técnica, el sistema no trata de evitar los interbloqueos. En vez de ello, intenta detectarlos cuando ocurran y luego realiza cierta acción para recuperarse después del hecho.

Detección de interbloqueos con un recurso de cada tipo

Detección de interbloqueos con un recurso de cada tipo. Si sólo existe un recurso de cada tipo, podemos construir un gráfico de recursos. Si este gráfico contiene uno o más ciclos, existe un interbloqueo. Cualquier proceso que forme parte de un ciclo está en interbloqueo. Si no existen ciclos, el sistema no está en interbloqueo.

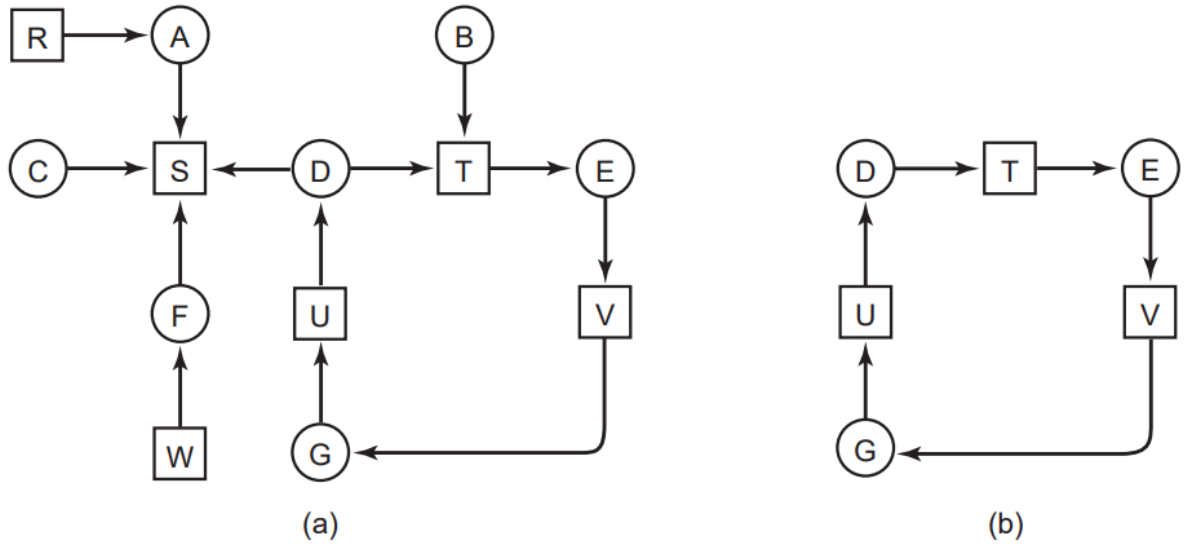


Figura 6-5. (a) Un gráfico de recursos. (b) Un ciclo extraído de (a).

Existen muchos algoritmos conocidos para detectar ciclos en gráficos dirigidos. Lo que hace este algoritmo es tomar cada nodo en turno, como la raíz de lo que espera sea un árbol, y realiza una búsqueda de un nivel de profundidad en él. Si alguna vez regresa a un nodo que ya se haya encontrado, entonces ha

encontrado un ciclo. Si agota todos los arcos desde cualquier nodo dado, regresa al nodo anterior. Si regresa hasta la raíz y no puede avanzar más, el subgráfico que se puede alcanzar desde el nodo actual no contiene ciclos. Si esta propiedad se aplica para todos los nodos, el gráfico completo está libre de ciclos, por lo que el sistema no está en interbloqueo.

Detección del interbloqueo con varios recursos de cada tipo

No está en diapositivas

Para esto necesitamos un método distinto. Ahora presentaremos un algoritmo basado en dos matrices y dos vectores.

- E es el vector de recursos existentes. Este vector proporciona el número total de instancias de cada recurso en existencia.
- A sea el vector de recursos disponibles. Este vector proporciona el número de instancias del recurso i que están disponibles en un momento dado (es decir, sin asignar).
- C, la matriz de asignaciones actuales y R, la matriz de peticiones. La i-ésima fila de C nos indica cuántas instancias de cada clase de recurso contiene Pi. Cij es el número de instancias del recurso j que están contenidas por el proceso i. De manera similar, Rij es el número de instancias del recurso j que desea Pi.

RECUPERACIÓN DE UN INTERBLOQUEO

Recuperación por medio de apropiación

Consiste en quitar temporalmente un recurso a su propietario actual y otorgarlo a otro proceso. En muchos casos se puede requerir intervención manual, en especial en los sistemas operativos de procesamiento por lotes que se ejecutan en mainframes.

La habilidad de quitar un recurso a un proceso, hacer que otro proceso lo utilice y después regresarlo sin que el proceso lo note, depende en gran parte de la naturaleza del recurso. Con frecuencia es difícil o imposible recuperarse de esta manera

Recuperación a través del retroceso

Si los diseñadores de sistemas y operadores de máquinas saben que es probable que haya interbloqueos, pueden hacer que los procesos realicen **puntos de comprobación** (checkpoint) en forma periódica.

El punto de comprobación no sólo contiene la imagen de la memoria, sino también el estado del recurso. Los nuevos puntos de comprobación no deben sobrescribir a los anteriores.
Cuando se detecta un interbloqueo, es fácil ver cuáles recursos se necesitan. Para realizar la recuperación, un proceso que posee un recurso necesario se revierte a un punto en el tiempo antes de que haya adquirido ese recurso, para lo cual se inicia uno de sus puntos de comprobación anteriores.

Recuperación a través de la eliminación de procesos

La forma más cruda y simple de romper un interbloqueo es eliminar uno o más procesos. Si esto no ayuda, se puede repetir hasta que se rompa el ciclo. De manera alternativa, se puede elegir como víctima a un proceso que no esté en el ciclo, para poder liberar sus recursos. Si bien es un algoritmo muy agresivo, es mejor eliminar un proceso que se pueda volver a ejecutar desde el principio sin efectos dañinos.

CÓMO EVITAR INTERBLOQUEOS

El sistema debe ser capaz de decidir si es seguro otorgar un recurso o si no lo es, y realizar la asignación sólo cuando sea seguro. Podemos evitar los interbloqueos, pero sólo si hay cierta información disponible de antemano.

Trayectorias de los recursos

Los principales algoritmos para evitar interbloqueos se basan en el concepto de los estados seguros.

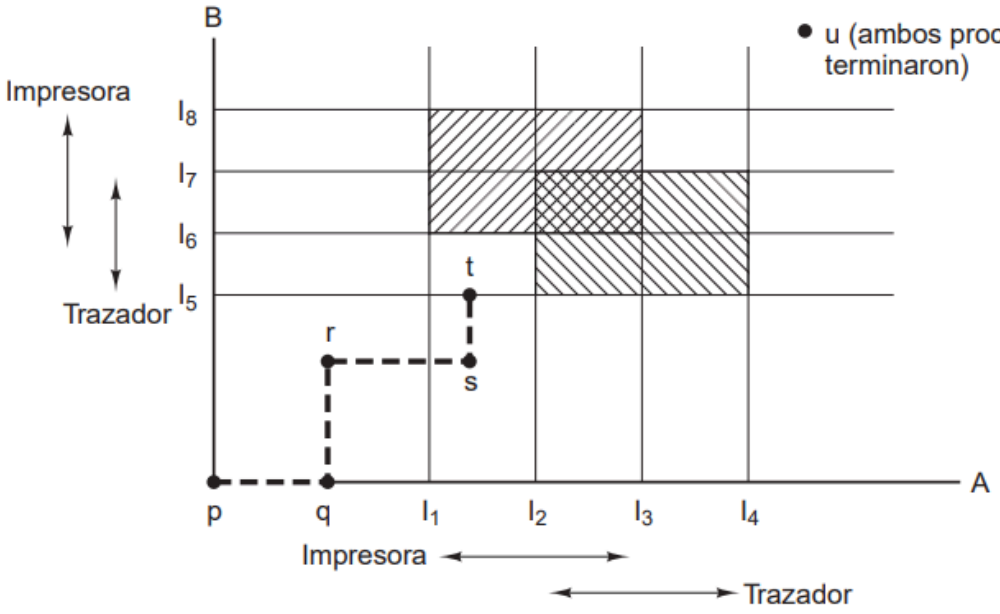


Figura 6-8. Trayectorias de recursos de dos procesos.

Con un solo procesador, todas las rutas deben ser horizontales o verticales, nunca diagonales. Además, el movimiento siempre es hacia el norte o al este. Las regiones sombreadas son en especial interesantes. Si el sistema entra alguna vez al cuadro delimitado entrará en interbloqueo en un momento dado. Todo el cuadro es inseguro y no se debe entrar en él.

Estados seguros e inseguros

Se dice que un estado es **seguro** si hay cierto orden de programación en el que se puede ejecutar cada proceso hasta completarse, incluso aunque todos ellos solicitaran de manera repentina su número máximo de recursos de inmediato.

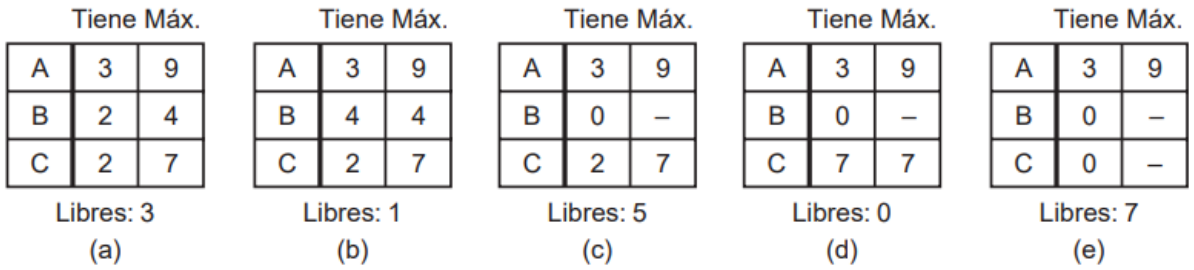


Figura 6-9. Demostración de que el estado en (a) es seguro.

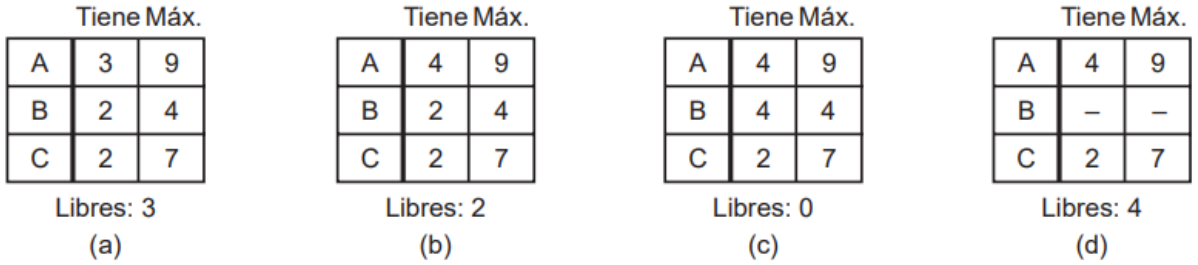


Figura 6-10. Demostración de que el estado en (b) no es seguro.

Sólo tenemos cuatro instancias del recurso libres, y cada uno de los procesos activos necesita cinco. No hay una secuencia que garantice que los procesos se completarán. Así, la decisión de asignación que cambió al sistema de la figura 6-10(a) a la figura 6-10(b) pasó de un estado seguro a uno inseguro. Vale la pena observar que un estado inseguro no es un estado en interbloqueo.

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

CÓMO PREVENIR INTERBLOQUEOS

Sí podemos asegurar que por lo menos una de estas condiciones nunca se cumpla, entonces los interbloqueos serán estructuralmente imposibles.

Condición	Método
Exclusión mutua	Poner todo en la cola de impresión
Contención y espera	Solicitar todos los recursos al principio
No apropiativa	Quitar los recursos
Espera circular	Ordenar los recursos en forma numérica

Figura 6-14. Resumen de los métodos para evitar interbloqueos.

Cómo atacar la condición de exclusión mutua

Si ningún recurso se asignara de manera exclusiva a un solo proceso, nunca tendríamos interbloqueos. Ejemplo: Al colocar la salida de la impresora en una cola de impresión, varios procesos pueden generar salida al mismo tiempo. En este modelo, el único proceso que realmente solicita la impresora física es el demonio de impresión. Como el demonio nunca solicita ningún otro recurso, podemos eliminar el interbloqueo para la impresora.

Cómo atacar la condición de contención y espera

Si podemos evitar que los procesos que contienen recursos esperen por más recursos, podemos eliminar los interbloqueos. Una forma de lograr esta meta es requerir que todos los procesos soliciten todos sus recursos antes de empezar su ejecución. Si todo está disponible, al proceso se le asignará lo que necesite y podrá ejecutarse hasta completarse. Si uno o más recursos están ocupados, no se asignará nada y el proceso sólo esperará.

Un problema inmediato con este método es que muchos procesos no saben cuántos recursos necesitarán hasta que hayan empezado a ejecutarse. Otro problema es que los recursos no se utilizarán de una manera óptima con este método.

Una manera ligeramente distinta de romper la condición de contención y espera es requerir que un proceso que solicita un recurso libere temporalmente todos los recursos que contiene en un momento dado. Después puede tratar de obtener todo lo que necesite a la vez.

Cómo atacar la condición no apropiativa

Si a un proceso se le ha asignado la impresora y está a la mitad de imprimir su salida, quitarle la impresora a la fuerza es algo engañoso como máximo. Sin embargo, ciertos recursos se pueden virtualizar para evitar esta situación.

Al colocar en una cola de impresión en el disco la salida de la impresora y permitir que sólo el demonio de impresión tenga acceso a la impresora real, se

eliminan los interbloqueos que involucran a la impresora, aunque se crea uno para el espacio en disco.

Cómo atacar la condición de espera circular

La espera circular se puede eliminar de varias formas. Una de ellas es simplemente tener una regla que diga que un proceso tiene derecho sólo a un recurso en cualquier momento. Si necesita un segundo recurso, debe liberar el primero.

Otra manera de evitar la espera circular es proporcionar una numeración global de todos los recursos: Los procesos pueden solicitar recursos cada vez que quieran, pero todas las peticiones se deben realizar en orden numérico. Un proceso puede pedir primero una impresora y después una unidad de cinta, pero tal vez no pueda pedir primero un trazador y después una impresora.

Figura 6-13. (a) Recursos ordenados en forma numérica. (b) Un gráfico de recursos.

Una variación menor de este algoritmo es retirar el requerimiento de que los recursos se adquieran en una secuencia cada vez más estricta, y simplemente insistir que ningún proceso puede solicitar un recurso menor del que ya contiene.