UADER | FCyT

Sistemas Operativos

Procesos

Clase 4 - Unidad II

Lic. Alexis Sostersich
sostersich.alexis@uader.edu.ar

Teoría – Sistemas Operativos

Procesos

Unidad II – Clase 4

- Procesos.
  - Concepto de proceso.
  - Tipos de procesos.
  - Estados de un proceso.
  - Transiciones de un proceso.

Concepto de proceso

Procesos

El concepto más importante en cualquier sistema operativo es el de proceso, una abstracción de un programa en ejecución; todo lo demás depende de este concepto, por lo cual es importante que el diseñador del sistema operativo (y el estudiante) tenga una comprensión profunda acerca de lo que es un proceso lo más pronto posible.

Los procesos son una de las abstracciones más antiguas e importantes que proporcionan los sistemas operativos: **proporcionan la capacidad de operar (pseudo) concurrentemente, incluso cuando hay sólo una CPU disponible**. Convierten una CPU en varias CPU virtuales. Sin la abstracción de los procesos, la computación moderna no podría existir.

Procesos

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

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

El modelo del proceso

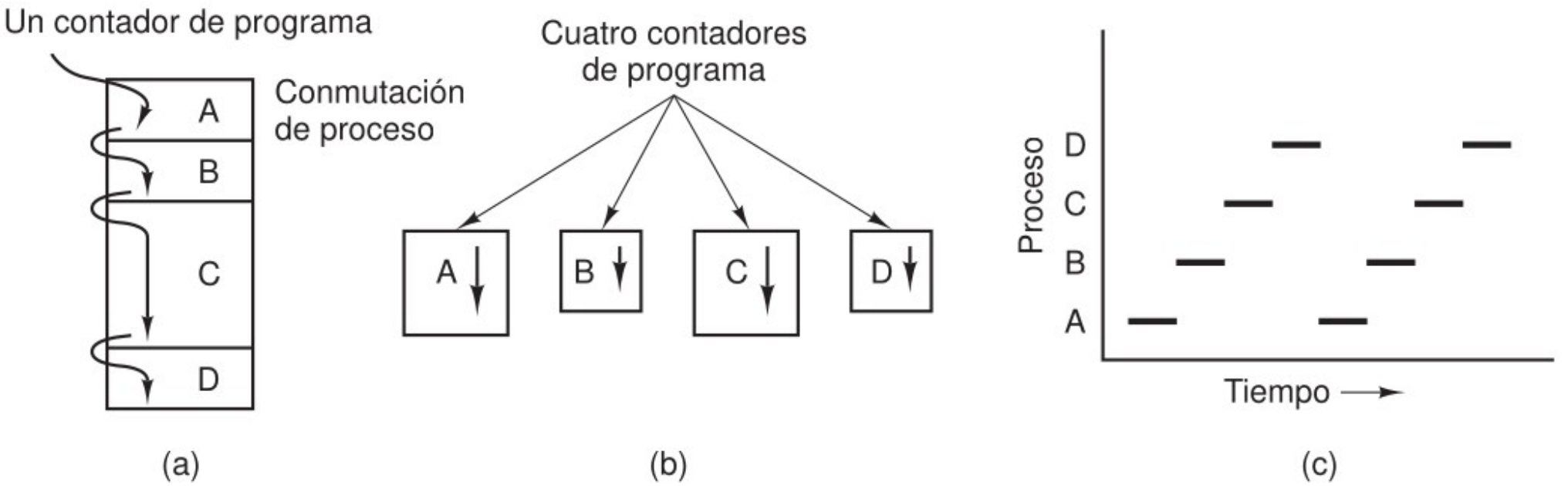
En este modelo, todo el software ejecutable en la computadora, que algunas veces incluye al sistema operativo, se organiza en varios procesos secuenciales (procesos, para abreviar).

Un proceso no es más que una instancia de un programa en ejecución, incluyendo los valores actuales del contador de programa, los registros y las variables.

En concepto, cada proceso tiene su propia CPU virtual; en la realidad, la CPU real conmuta de un proceso a otro, pero para entender el sistema es mucho más fácil pensar en una colección de procesos que se ejecutan en (pseudo) paralelo, en vez de tratar de llevar la cuenta de cómo la CPU conmuta de programa en programa.

Esta conmutación rápida de un proceso a otro se conoce como multiprogramación

Multiprogramación



Creación de un proceso

Los sistemas operativos necesitan cierta manera de crear procesos. En sistemas muy simples o sistemas diseñados para ejecutar sólo una aplicación, es posible tener presentes todos los procesos que se vayan a requerir cuando el sistema inicie. No obstante, en los sistemas de propósito general se necesita cierta forma de crear y terminar procesos según sea necesario durante la operación.

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.

Arranque del sistema

Cuando se arranca un sistema operativo se crean varios procesos.

Algunos de ellos son procesos en primer plano; es decir, procesos que interactúan con los usuarios (humanos) y realizan trabajo para ellos.

Otros son procesos en segundo plano, que no están asociados con usuarios específicos sino con una función específica.

Los procesos que permanecen en segundo plano se conocen como demonios (daemons). Los sistemas grandes tienen comúnmente docenas de ellos.

En UNIX podemos utilizar el programa ps para listar los procesos en ejecución. En Windows podemos usar el administrador de tareas.

Creación de procesos

Además de los procesos que se crean al arranque, posteriormente se pueden crear otros. A menudo, **un proceso en ejecución emitirá llamadas al sistema para crear uno o más procesos nuevos, para que le ayuden a realizar su trabajo.**

En especial, es útil crear procesos cuando el trabajo a realizar se puede formular fácilmente en términos de varios procesos interactivos relacionados entre sí, pero independientes en los demás aspectos. Por ejemplo, si se va a obtener una gran cantidad de datos a través de una red para su posterior procesamiento, puede ser conveniente crear un proceso para obtener los datos y colocarlos en un búfer compartido, mientras un segundo proceso remueve los elementos de datos y los procesa.

En un multiprocesador, al permitir que cada proceso se ejecute en una CPU distinta también se puede hacer que el trabajo se realice con mayor rapidez.

Petición de usuario

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

En los sistemas UNIX basados en comandos que ejecutan X, el nuevo proceso se hace cargo de la ventana en la que se inició.

En Microsoft Windows, cuando se inicia un proceso no tiene una ventana, pero puede crear una (o más) y la mayoría lo hace.

En ambos sistemas, los usuarios pueden tener varias ventanas abiertas a la vez, cada una ejecutando algún proceso. Mediante el ratón, el usuario puede seleccionar una ventana e interactuar con el proceso, por ejemplo para proveer datos cuando sea necesario.

Trabajo por lotes

La última situación en la que se crean los procesos se aplica sólo a los sistemas de procesamiento por lotes que se encuentran en las mainframes grandes.

Aquí los usuarios pueden enviar trabajos de procesamiento por lotes al sistema (posiblemente en forma remota).

Cuando el sistema operativo decide que tiene los recursos para ejecutar otro trabajo, crea un proceso y ejecuta el siguiente trabajo de la cola de entrada.

UNIX

En UNIX sólo hay una llamada al sistema para crear un proceso: fork. Esta llamada crea un clon exacto del proceso que hizo la llamada. Después de fork, los dos procesos (padre e hijo) tienen la misma imagen de memoria, las mismas cadenas de entorno y los mismos archivos abiertos. Eso es todo.

Por lo general, el proceso hijo ejecuta después a execve o una llamada al sistema similar para cambiar su imagen de memoria y ejecutar un nuevo programa.

Por ejemplo, cuando un usuario escribe un comando tal como sort en el shell, éste crea un proceso hijo, que a su vez ejecuta a sort. La razón de este proceso de dos pasos es para permitir al hijo manipular sus descriptores de archivo después de fork, pero antes de execve, para poder lograr la redirección de la entrada estándar, la salida estándar y el error estándar.

Windows

En Windows una sola llamada a una función de Win32 (CreateProcess) maneja la creación de procesos y carga el programa correcto en el nuevo proceso.

Esta llamada tiene 10 parámetros, que incluyen el programa a ejecutar, los parámetros de la línea de comandos para introducir datos a ese programa, varios atributos de seguridad, bits que controlan si los archivos abiertos se heredan, información de prioridad, una especificación de la ventana que se va a crear para el proceso (si se va a crear una) y un apuntador a una estructura en donde se devuelve al proceso que hizo la llamada la información acerca del proceso recién creado.

Además de CreateProcess, Win32 tiene cerca de 100 funciones más para administrar y sincronizar procesos y temas relacionados.

UNIX y Windows

Tanto en UNIX como en Windows, una vez que se crea un proceso, el padre y el hijo tienen sus propios espacios de direcciones distintos. Si cualquiera de los procesos modifica una palabra en su espacio de direcciones, esta modificación no es visible para el otro proceso.

En UNIX, el espacio de direcciones inicial del hijo es una copia del padre, pero en definitiva hay dos espacios de direcciones distintos involucrados; no se comparte memoria en la que se pueda escribir (algunas implementaciones de UNIX comparten el texto del programa entre los dos, debido a que no se puede modificar). Sin embargo, es posible para un proceso recién creado compartir algunos de los otros recursos de su creador, como los archivos abiertos.

En Windows, los espacios de direcciones del hijo y del padre son distintos desde el principio.

Terminación de procesos

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

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

Salida normal

La mayoría de los procesos terminan debido a que han concluido su trabajo. Cuando un compilador ha compilado el programa que recibe, ejecuta una llamada al sistema para indicar al sistema operativo que ha terminado. Esta llamada es exit en UNIX y ExitProcess en Windows.

Los programas orientados a pantalla también admiten la terminación voluntaria. Los procesadores de palabras, navegadores de Internet y programas similares siempre tienen un icono o elemento de menú en el que el usuario puede hacer clic para indicar al proceso que elimine todos los archivos temporales que tenga abiertos y después termine.

Salida por error

La segunda razón de terminación es que el proceso descubra un error.

Por ejemplo, si un usuario escribe el comando para compilar el programa foo.c y no existe dicho archivo, el compilador simplemente termina.

Los procesos interactivos orientados a pantalla por lo general no terminan cuando reciben parámetros incorrectos. En vez de ello, aparece un cuadro de diálogo y se le pide al usuario que intente de nuevo.

Error fatal

La tercera razón de terminación es un error fatal producido por el proceso, a menudo debido a un error en el programa.

Algunos ejemplos incluyen el ejecutar una instrucción ilegal, hacer referencia a una parte de memoria no existente o la división entre cero.

En algunos sistemas (como UNIX), un proceso puede indicar al sistema operativo que desea manejar ciertos errores por sí mismo, en cuyo caso el proceso recibe una señal (se interrumpe) en vez de terminar.

Eliminado por otro proceso

La cuarta razón por la que un proceso podría terminar es que ejecute una llamada al sistema que indique al sistema operativo que elimine otros procesos.

En UNIX esta llamada es kill. La función correspondiente en Win32 es TerminateProcess.

En ambos casos, el proceso eliminador debe tener la autorización necesaria para realizar la eliminación.

En algunos sistemas, cuando un proceso termina (ya sea en forma voluntaria o forzosa) todos los procesos que creó se eliminan de inmediato también. Sin embargo, ni Windows ni UNIX trabajan de esta forma.

Jerarquías de procesos

Jerarquías de procesos

En algunos sistemas, cuando un proceso crea otro, el proceso padre y el proceso hijo continúan asociados en ciertas formas. El proceso hijo puede crear por sí mismo más procesos, formando una jerarquía de procesos. Observe que, 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. Cuando un usuario envía una señal del teclado, ésta se envía a todos los miembros del grupo de procesos actualmente asociado con el teclado (por lo general, todos los procesos activos que se crearon en la ventana actual). De manera individual, cada proceso puede atrapar la señal, ignorarla o tomar la acción predeterminada que es ser eliminado por la señal.

Jerarquías de procesos

Como otro ejemplo dónde la jerarquía de procesos juega su papel, veamos la forma en que UNIX se inicializa a sí mismo cuando se enciende la computadora.

Hay un proceso especial (llamado init) en la imagen de inicio. Cuando empieza a ejecutarse, lee un archivo que le indica cuántas terminales hay. Después utiliza fork para crear un proceso por cada terminal.

Estos procesos esperan a que alguien inicie la sesión. Si un inicio de sesión tiene éxito, el proceso de inicio de sesión ejecuta un shell para aceptar comandos.

Éstos pueden iniciar más procesos y así sucesivamente. Por ende, todos los procesos en el sistema completo pertenecen a un solo árbol, con init en la raíz.

Jerarquías 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 un proceso

Estados de un proceso

Aunque cada proceso es una entidad independiente, con su propio contador de programa y estado interno, a menudo los procesos necesitan interactuar con otros.

Un proceso puede generar cierta salida que otro proceso utiliza como entrada. En el comando de shell

```
$ cat capitulo1 capitulo2 capitulo3 | grep arbol
```

el primer proceso, que ejecuta cat, concatena tres archivos y los envía como salida, puede ocurrir que grep esté listo para ejecutarse, pero que no haya una entrada esperándolo. Entonces debe bloquear hasta que haya una entrada disponible.

Cuando un proceso se bloquea, lo hace debido a que por lógica no puede continuar, comúnmente porque está esperando una entrada que todavía no está disponible.

Estados de un proceso (Tanenbaum)

También es posible que un proceso, que esté listo en concepto y pueda ejecutarse, se detenga debido a que el sistema operativo ha decidido asignar la CPU a otro proceso por cierto tiempo. Estas dos condiciones son completamente distintas. En el primer caso, la suspensión está inherente en el problema (no se puede procesar la línea de comandos del usuario sino hasta que éste la haya escrito mediante el teclado).

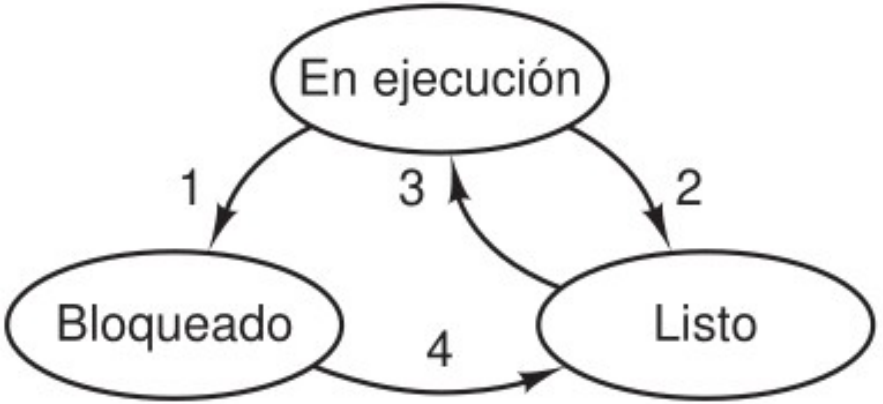
En el segundo caso, es un tecnicismo del sistema (no hay suficientes CPUs como para otorgar a cada proceso su propio procesador privado).

Los tres estados en los que se puede encontrar un proceso:

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).

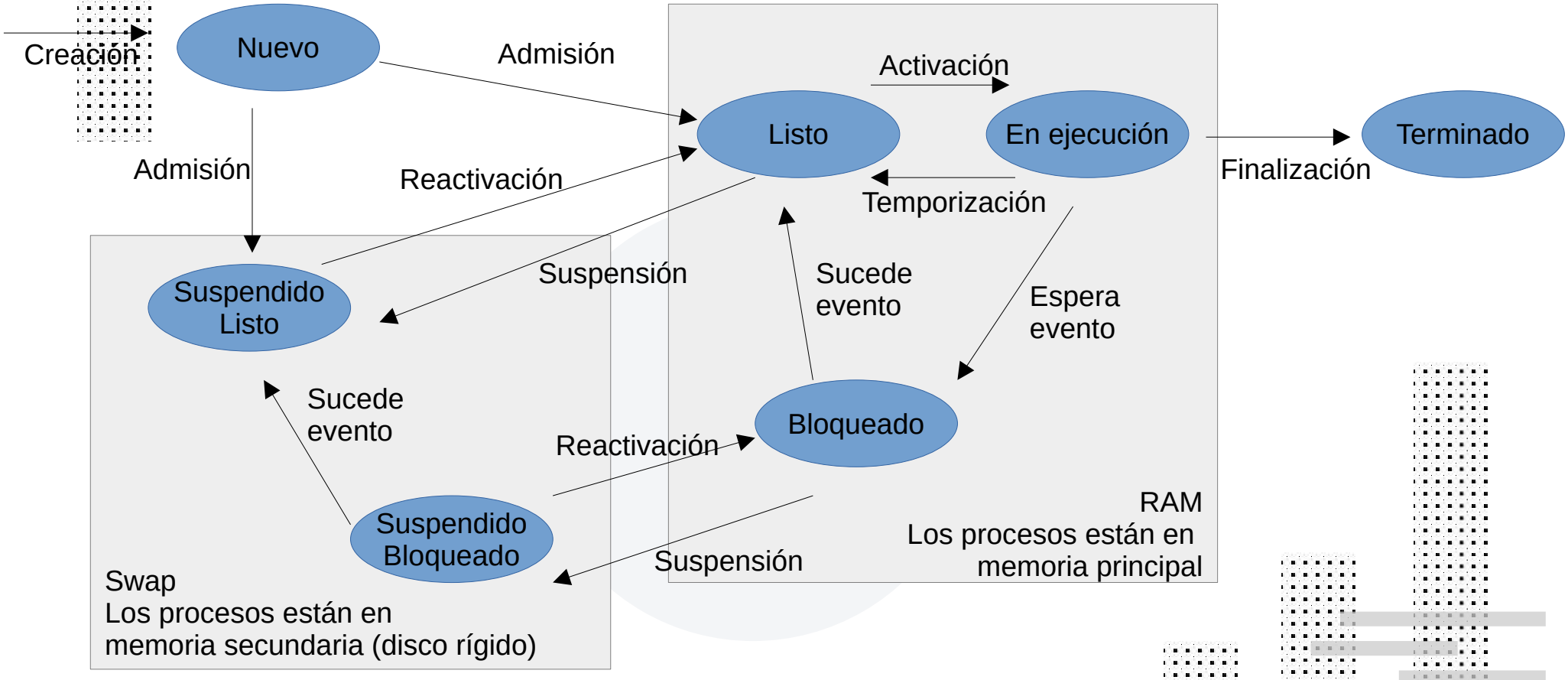
Estados de un proceso (Tanenbaum)

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



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

Estados de un proceso (Stallings)



Transiciones de un proceso

Transiciones de un proceso

Hay cuatro transiciones posibles entre estos tres estados, como se indica.

La transición 1 ocurre cuando el sistema operativo descubre que un proceso no puede continuar justo en ese momento. En algunos sistemas el proceso puede ejecutar una llamada al sistema, como pause, para entrar al estado bloqueado. En otros sistemas, incluyendo a UNIX, cuando un proceso lee datos de una canalización o de un archivo especial (como una terminal) y no hay entrada disponible, el proceso se bloquea en forma automática.

Las transiciones 2 y 3 son producidas por el planificador de procesos, una parte del sistema operativo, sin que el proceso sepa siquiera acerca de ellas.

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.

Transiciones de un proceso

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. El tema de la planificación de procesos (decidir qué proceso debe ejecutarse en qué momento y por cuánto tiempo) es importante; más adelante en este capítulo lo analizaremos. Se han ideado muchos algoritmos para tratar de balancear las contrastantes demandas de eficiencia para el sistema como un todo y de equidad para los procesos individuales; más adelante en este capítulo estudiaremos algunas.

La transición 4 ocurre cuando se produce el evento externo por el que un proceso estaba esperando (como la llegada de ciertos datos de entrada). Si no hay otro proceso en ejecución en ese instante, se activa la transición 3 y el proceso empieza a ejecutarse. En caso contrario, tal vez tenga que esperar en el estado listo por unos instantes, hasta que la CPU esté disponible y sea su turno de utilizarla.

Implementación de los procesos

Implementación de los procesos

Para implementar el modelo de procesos, **el sistema operativo mantiene una tabla de procesos**, con sólo una entrada por cada proceso que contiene información importante acerca del estado del proceso, incluyendo su contador de programa, apuntador de pila, asignación de memoria, estado de sus archivos abiertos, información de contabilidad y planificación, y todo lo de más que debe guardarse acerca del proceso cuando éste cambia del estado en ejecución a listo o bloqueado, de manera que se pueda reiniciar posteriormente como si nunca se hubiera detenido.

Hay que recalcar que los campos contenidos en la tabla de procesos varían de un sistema a otro.

Implementación de los procesos

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		

Hilos

Hilos

En los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones y un solo hilo de control. De hecho, ésa es casi la definición de un proceso. Sin embargo, con frecuencia 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).

La principal razón de tener hilos es que en muchas aplicaciones se desarrollan varias actividades a la vez. Algunas de ésas se pueden bloquear de vez en cuando. Al descomponer una aplicación en varios hilos secuenciales que se ejecutan en cuasi-paralelo, el modelo de programación se simplifica.

Hilos

Ya hemos visto este argumento antes: es precisamente la justificación de tener procesos. En vez de pensar en interrupciones, temporizadores y conmutaciones de contexto, podemos pensar en procesos paralelos.

Sólo que ahora con los hilos agregamos un nuevo elemento: la habilidad de las entidades en paralelo de compartir un espacio de direcciones y todos sus datos entre ellas.

Esta habilidad es esencial para ciertas aplicaciones, razón por la cual no funcionará el tener varios procesos (con sus espacios de direcciones separados).

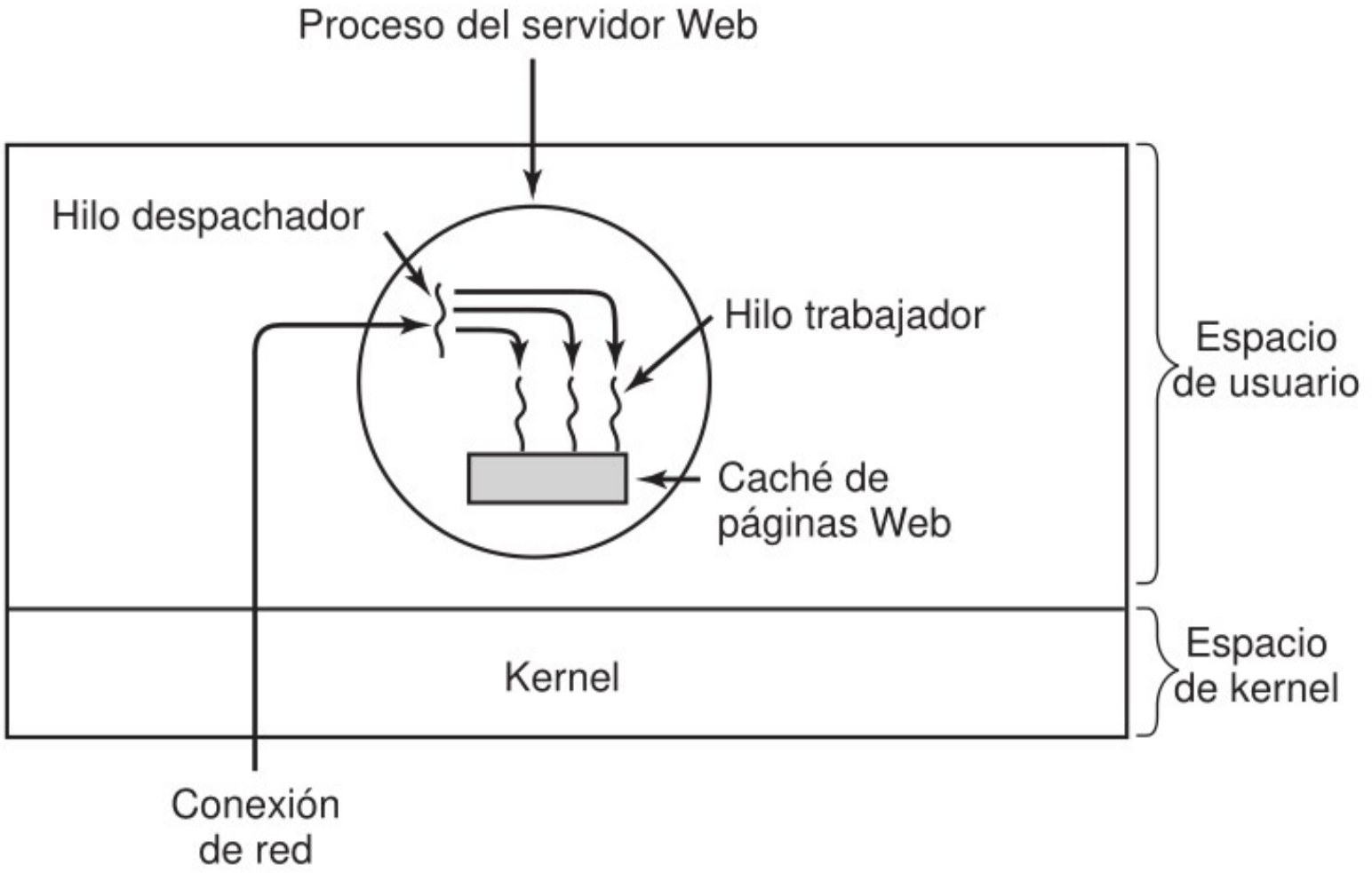
Hilos

Una tercera razón de tener hilos es también un argumento relacionado con el rendimiento.

Los hilos no producen un aumento en el rendimiento cuando todos ellos están ligados a la CPU, 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.

Por último, los hilos son útiles en los sistemas con varias CPUs, en donde es posible el verdadero paralelismo.

Hilos


Proceso del servidor Web
Hilo despachador
Hilo trabajador
Caché de
páginas Web
Espacio
de usuario
Kernel
Espacio
de kernel
Conexión
de red

Bibliografía

- **Tanenbaum, A. S. (2009)**. Sistemas operativos modernos (3a ed.) (pp. 1-18). México: Pearson Educación.
- **Stallings, W. (2005)**. Sistemas operativos (5a ed.) (pp. 54-67). Madrid: Pearson Educación.