Sistemas Operativos I

Módulo II

PROCESOS E HILOS

1

Temas del Módulo II

- **Procesos.**
  - Concepto de proceso.
  - Creación de un proceso.
  - Terminación de un proceso.
  - Jerarquías de procesos.
  - Estados de un proceso.
  - Implementación de los procesos.
    - Estructuras de control del SO.
    - Componentes de un proceso.
  - Cambio de proceso.
- **Hilos.**
  - Uso de hilos.
  - Modelo clásico de hilos.
  - Hilos en POSIX.
  - Hilos en el espacio de usuario.
  - Hilos en el espacio de núcleo.

2

Procesos

- **Concepto de proceso.**
- Todas las computadoras modernas ofrecen la posibilidad de realizar múltiples tareas “al mismo tiempo”; esta característica está tan naturalizada que no somos completamente conscientes de este hecho.
- Un ejemplo es el caso de una computadora personal, cuando arranca el sistema se inician muchos procesos en forma oculta, que por lo general el usuario desconoce.
- Típicamente, un antivirus que actualiza las definiciones de virus, un cliente de correo electrónico que espera el correo entrante, etc. Todo esto mientras el usuario navega en Internet.
- Toda esta actividad se tiene que administrar, y en estos casos el concepto de proceso es muy útil.

3

Procesos

- **Concepto de proceso.** (*cont.*)
- Los procesos son una de las más antiguas e importantes abstracciones que proporcionan los sistemas operativos: brindan la capacidad de operar concurrentemente, incluso cuando sólo hay una CPU disponible. Esto es, convierten una CPU en varias CPU virtuales.
- En el modelo de procesos, todo el software ejecutable, incluyendo el SO, se organiza en **procesos secuenciales** (procesos, para abreviar). Un proceso no es más que una instancia de un programa en ejecución, incluyendo su contexto de ejecución (contador de programa, registros, variables).
- Conceptualmente, cada proceso tiene su propia CPU virtual; en la realidad, la CPU real conmuta rápidamente de un proceso a otro. Pero es más fácil pensar en un conjunto de procesos ejecutando en paralelo.

4

Procesos

- **Concepto de proceso.** (*cont.*)
- Esta conmutación rápida de un proceso a otro se conoce como multiprogramación. La cantidad de procesos que pueden almacenarse en memoria principal determina el grado de multiprogramación de un sistema.

5

Procesos

- **Concepto de proceso.** (*cont.*)
- La diferencia entre un proceso y un programa es sutil, pero crucial. Una analogía para entenderla, sería la siguiente: una **persona** tiene que cocinar un estofado; para ello cuenta con una **receta**, una cocina y los **ingredientes** (verduras, carne, condimentos).
- La **receta** sería el programa, es decir el algoritmo expresado en una notación adecuada, los pasos a seguir. La **persona** es el procesador (CPU) y los **ingredientes**, los datos de entrada.
- El **proceso**, entonces, es la **actividad** que consiste en que la persona vaya leyendo la receta, obteniendo los ingredientes y cocinando el estofado.

6

Procesos

- **Concepto de proceso.** (*cont.*)
- Imaginemos que el hijo de esta persona entra a la cocina corriendo y gritando que se lastimó la rodilla. Entonces, la persona anota hasta dónde llegó con la receta (**guarda el estado del proceso**), saca un spray antiséptico del botiquín y sigue las instrucciones del envase para aplicárselo al hijo.
- Aquí el procesador conmuta de un proceso (cocinar el estofado) a uno de mayor prioridad (aplicar el antiséptico), cada uno con un programa distinto (la receta y las instrucciones en el envase del antiséptico).
- Cuando se desocupa de la rodilla lastimada, retoma la cocción del estofado en el punto que había anotado.

7

Procesos

- **Concepto de proceso.** (*cont.*)
- La clave es que **un proceso es una actividad**: tiene un **programa**, **datos** de entrada y salida (¡el estofado!), y un **estado**. Varios procesos pueden compartir un procesador mediante un algoritmo de planificación para determinar cuándo detener la ejecución de un proceso para dar servicio a otro.

8

Procesos

- **Creación de un proceso.**
- Los SO necesitan cierta manera de crear procesos. En sistemas pequeños, como los embebidos, se puede saber con exactitud los procesos que se el sistema va a requerir al inicio del mismo; pero en sistemas de propósitos generales, es necesario poder crear procesos dinámicamente, a medida que son requeridos.
- Existen cuatro eventos principales que crean eventos:
  - El arranque del sistema.
  - La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.
  - Una petición de usuario para crear un proceso.
  - El inicio de un trabajo por lotes.

9

Procesos

- **Creación de un proceso.** *(cont.)*
- En el arranque del sistema: Cuando arranca un SO se crean varios procesos. Algunos son procesos en primer plano, es decir, que interactúan con los usuarios y realizan algún trabajo para ellos. Otros son en segundo plano, esto es, no están asociados con usuarios específicos sino con una tarea específica; éstos se conocen como demonios (daemons).
- Desde un proceso: Posterior al arranque se pueden crear otros procesos. A menudo, un proceso en ejecución puede emitir llamadas al sistema para crear procesos que le ayuden en su trabajo, típicamente cuando se puede plantear un esquema de varios procesos relacionados entre sí, pero independientes en otros aspectos.

10

Procesos

- **Creación de un proceso.** *(cont.)*
- Por pedido de un usuario: En sistemas interactivos, un usuario puede iniciar un programa escribiendo un comando o haciendo clic en un ícono. Cualquiera de estas acciones inicia un proceso y ejecuta el programa asociado.
- Un trabajo por lotes: El caso de la creación de un de proceso por inicio de un trabajo por lotes se aplica a sistemas de procesamiento que se encuentran en las mainframes grandes; aquí, los usuarios envían trabajos al sistema, generalmente en forma remota. Cuando el SO tiene los recursos, crea un proceso y ejecuta el siguiente trabajo en la cola de entrada.

11

Procesos

- **Terminación de un proceso.**
- Luego de su creación, un proceso empieza a ejecutar y realiza el trabajo para el que fue destinado. Pero, como en la vida misma, nada es para siempre: tarde o temprano, un proceso terminará su ejecución debido a alguna de las siguientes condiciones:
  - Salida normal (voluntaria).
  - Salida por error (voluntaria).
  - Error fatal (involuntaria).
  - Eliminado por otro proceso (involuntaria).
- Salida normal: La mayoría de los procesos terminan su ejecución porque han concluido su trabajo. Por ejemplo, cuando un compilador ha compilado un programa, ejecuta una llamada al sistema para indicar al SO que ha terminado.

12

Procesos

- **Terminación de un proceso.** *(cont.)*
- Salida por error: La segunda razón de terminación es que el proceso descubre un error en su ejecución. Por ejemplo, si un usuario ejecuta un comando para compilar un archivo fuente y éste último no existe; el compilador, al no encontrar el archivo, simplemente termina.
- Error fatal: Se produce un error fatal, a menudo, debido a un error en el programa, tales como ejecutar una instrucción ilegal, hacer referencia a un lugar de memoria no existente o restringido, o bien una división por cero.
- Eliminado por otro proceso: Un proceso puede invocar una llamada al sistema que indique al SO que elimine otros procesos. Para ello, el proceso eliminador debe contar con la autorización necesaria para realizar la eliminación.

13

Procesos

- **Jerarquías de procesos.**
- En algunos sistemas, cuando un proceso crea otro, padre e hijo continúan asociados de ciertas formas. El proceso hijo puede crear por sí mismo más procesos, formando una jerarquía de procesos.
- En POSIX, un proceso y todos sus hijos, y sus descendientes, forman un grupo de procesos. Cuando un usuario envía una señal del teclado, ésta se envía a todos los miembros del grupo actualmente asociado al teclado. Cada proceso puede atrapar la señal, ignorarla o tomar la acción predeterminada.
- Otro ejemplo del papel de la jerarquía de procesos es la forma en la que POSIX inicializa al encender la computadora: hay un proceso especial, *init*, en la imagen de inicio. Cuando ejecuta, lee un archivo (*/etc/ttytab*) que le indica cuántas terminales hay.

14

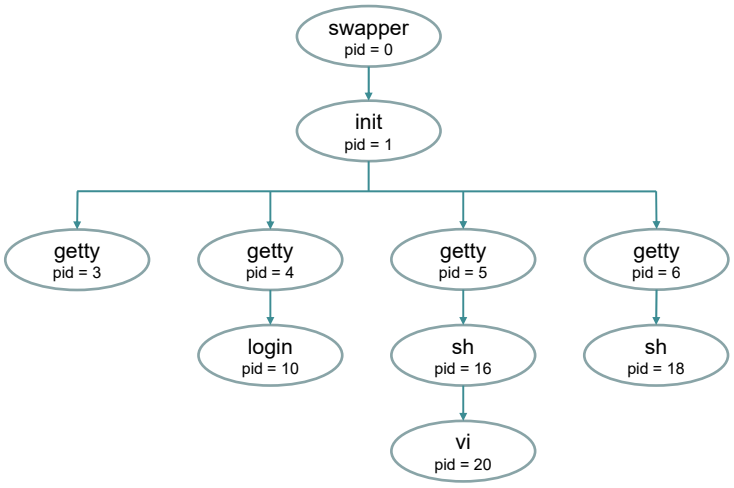
Procesos

- **Jerarquías de procesos.** (*cont.*)
- Después utiliza *fork* para crear un proceso (*getty*) por cada terminal; estos procesos esperan a que alguien inicie sesión. Cuando un usuario entra al sistema se ejecuta *login* con el nombre como argumento, y si el inicio de sesión tiene éxito, se ejecuta un *shell* (*sh*) para aceptar comandos.
- El *shell* del usuario se especifica en el archivo */etc/passwd*; éste contiene información completa acerca del usuario: nombre de la cuenta (*login*), contraseña (encriptada), UID, GID, nombre completo del usuario, directorio de trabajo y el intérprete de comando que utiliza.
- El *shell*, entonces, lanzará un *fork* y un *execve* por cada comando, iniciando así más procesos. Por ende, todos los procesos en el sistema pertenecen a un solo árbol, con *init* en la raíz.

15

Procesos

- **Jerarquías de procesos.** (*cont.*)
- El árbol de procesos en POSIX se arma de la siguiente manera:



16

Procesos

- **Jerarquías de procesos.** (*cont.*)
- Windows, en cambio, no incorpora el concepto de jerarquía de procesos; todos los procesos son iguales.
- La única sugerencia de jerarquía de procesos es que, cuando se crea un proceso, el padre recibe un indicador especial, llamado **manejador** (*handler*), 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 POSIX no pueden desheredar a sus hijos.

17

Procesos

- **Estados de un proceso.**
- Aunque cada proceso es una entidad independiente, a menudo necesitan interactuar con otros. Un proceso puede generar una salida que otro utiliza como entrada.
- Dependiendo de la velocidad relativa de los procesos, puede ocurrir que uno esté listo para ejecutar pero que aún no haya una entrada de datos disponible. En este caso, el proceso debe bloquearse hasta que haya una entrada.
- También es posible que un proceso que esté en ejecución se detenga debido a que el SO ha decidido asignar la CPU a otro proceso por un cierto tiempo.
- Estas dos situaciones son completamente distintas. En el primer caso, la suspensión es inherente al problema (no se dispone del recurso para seguir), mientras que, en el segundo, es un tecnicismo (no hay suficientes CPU).

18

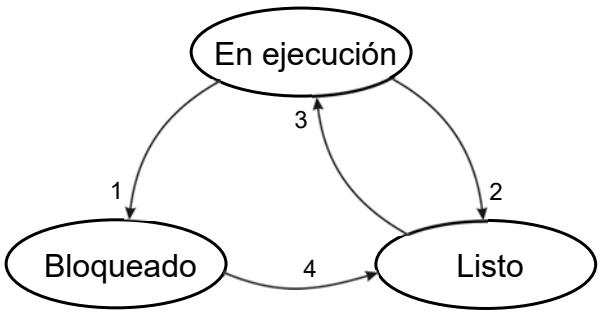
Procesos

- **Estados de un proceso.** (*cont.*)
- Entonces, un proceso puede encontrarse en tres estados posibles:
  - **En ejecución** (está listo y usando la CPU en ese instante).
  - **Listo** (ejecutable; se detuvo temporalmente para dar lugar a otro proceso).
  - **Bloqueado** (no puede continuar hasta que ocurra un evento externo).
- En sentido lógico, los dos primeros son similares: en ambos el proceso está listo para ejecutar, sólo que en el segundo no hay temporalmente una CPU disponible.
- El tercer estado es distinto de los dos primeros en cuanto a que el proceso no puede ejecutar, incluso habiendo una CPU disponible.

19

Procesos

- **Estados de un proceso.** (*cont.*)
- Hay cuatro transiciones posibles entre los tres estados:



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.

20

Procesos

- **Estados de un proceso.** (*cont.*)
- Si utilizamos el modelo de procesos, es mucho más fácil pensar que está ocurriendo dentro del sistema. Algunos procesos ejecutan programas que llevan a cabo los comandos que escribe un usuario; otros, son parte del SO y se encargan de cumplir con las peticiones de servicios como las *system calls*.
- Por ejemplo, cuando ocurre una IRQ de disco, el SO toma la decisión de dejar de ejecutar el proceso actual y ejecutar el proceso de disco que está bloqueado esperando esa IRQ.
- Entonces, en vez de pensar en IRQ's, podemos pensar en procesos de usuario, procesos de disco, procesos de terminal, etc., que se bloquean cuando están esperando a que algo ocurra.

21

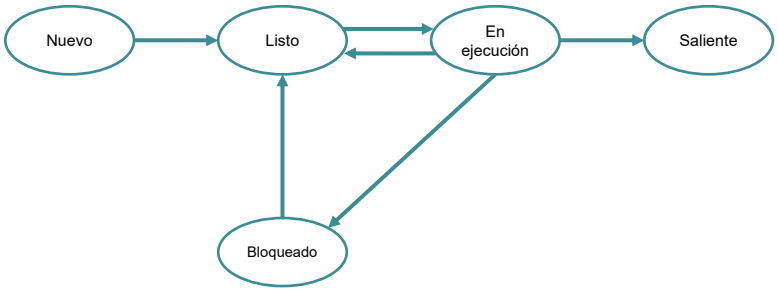
Procesos

- **Estados de un proceso.** (*cont.*)
- Podemos refinar el modelo anterior agregando dos estados muy útiles para la gestión de procesos:
  - **Nuevo:** se corresponde con un proceso que acaba de ser creado. En este estado, el SO crea todas las estructuras de datos necesarias para gestionar al nuevo proceso, pero aún no le ha sido asignado espacio en memoria principal.
  - **Saliente:** al terminar un proceso es movido a éste estado, donde deja de ser elegible para ejecutar. El SO puede mantener las estructuras de datos asociadas para extraer información, por ejemplo, de auditoría, tales como el tiempo de ejecución, recursos utilizados, etc., lo cual es útil para el análisis de rendimiento del sistema.

22

Procesos

- **Estados de un proceso.** (*cont.*)
- El diagrama de estados, incorporando éstos últimos, sería el siguiente:



Stallings, 2005

23

Procesos

- **Estados de un proceso.** (*cont.*)
- Mencionamos que los procesos se almacenan en memoria principal para poder ser ejecutados. Normalmente, los SO's tienen definido un grado de multiprogramación, que lo establece en cierta forma el rendimiento del mismo.
- En algunos casos hay procesos que se bloquean esperando un evento, típicamente de E/S, el cual puede tomar más tiempo de lo habitual, debido por ejemplo, a un gran volumen de datos a transferir.
- Esto nos lleva a tener procesos en memoria principal que no están en ejecución, ocupando espacio que podría se aprovechado por otro proceso que está listo y que por falta de espacio en memoria principal, continúa, por ejemplo, en el estado Nuevo.

24

Procesos

- **Estados de un proceso.** (*cont.*)
- Esta situación plantea la necesidad de quitar esos procesos bloqueados de memoria y llevarlos a un almacenamiento secundario (disco).
- La operación que lleva al proceso de memoria a disco se denomina *swapping* o *intercambio*.
- Tendremos, entonces, un conjunto de procesos bloqueados esperando un evento, almacenados en disco. Una vez que dicho evento se produce y/o hay disponibilidad de memoria, el proceso puede volver a ésta.
- También es posible que un proceso esté en el estado Listo y el SO decida otorgar el uso de CPU a un proceso de mayor prioridad, pero que no puede ser ejecutado por falta de espacio en memoria.

25

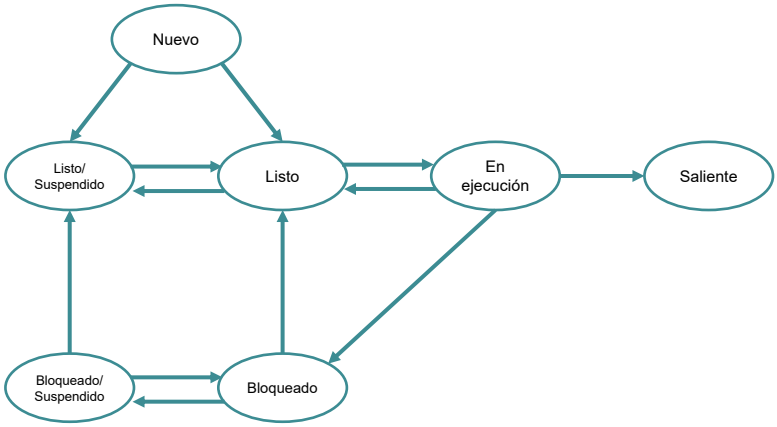
Procesos

- **Estados de un proceso.** (*cont.*)
- Entonces, se mueve al proceso que está listo a disco, y se otorga el espacio al proceso prioritario. Esto también implica una operación de *swapping*, pero desde el estado Listo.
- Por lo tanto, tendremos dos nuevos estados en el diagrama:
  - **Bloqueado/Suspendido:** el proceso está en almacenamiento secundario y esperando un evento.
  - **Listo/Suspendido:** el proceso está en almacenamiento secundario pero disponible para su ejecución en cuanto sea cargado en memoria principal.

26

Procesos

- **Estados de un proceso.** (*cont.*)
- El diagrama, incorporando los estados suspendidos, sería el siguiente:



Stallings, 2005

27

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.**
- Si el SO se encarga de la gestión de procesos y recursos, debe disponer de información sobre el estado actual de cada proceso y cada recurso. El mecanismo universal para esto es construir y mantener tablas de información sobre cada entidad que gestiona.
- Básicamente, debe administrar cuatro entidades primordiales:
  - Memoria.
  - Entrada/Salida.
  - Archivos.
  - Procesos.
- Entonces, el SO construirá tablas para administrar cuatro entidades.

28

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.** (*cont.*)
- Las tablas de memoria se usan para mantener un registro tanto de la memoria principal como de la secundaria. Parte de la memoria principal está reservada para el uso del SO; el resto está disponible para los procesos.
- Las tablas de memoria deben incluir la siguiente información:
  - Las reservas de memoria principal por parte de los procesos.
  - Las reservas de memoria secundaria por parte de los procesos.
  - Los atributos de protección que restringen el uso de memoria principal y secundaria.
  - La información necesaria para manejar la memoria virtual.

29

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.** (*cont.*)
- Las tablas de E/S se utilizan para gestionar los dispositivos de E/S (discos, placas de red, etc.). Contiene información sobre el estado de uso de los dispositivos como también de las colas de procesos que esperan para su uso.
- Un dispositivo puede estar disponible o asignado a un proceso en particular.
- Si una operación de E/S se está realizando, el SO necesita conocer el estado de la operación y la dirección de memoria principal del área usada como fuente o destino de la transferencia de E/S.

30

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.** (*cont.*)
- Las tablas de archivos proveen información sobre la existencia de archivos, su ubicación en almacenamiento secundario, su estado actual (abiertos, bloqueados, compartidos), y otros atributos (permisos, fechas y horas de creación).
- Esta información se puede gestionar a través del sistema de archivos, una funcionalidad especial del SO.
- Por último, las tablas de procesos son utilizadas para la administración de los procesos.
- Existe una tabla primaria de procesos, a la cual llamaremos simplemente **tabla de procesos**. Cada entrada de ésta tabla contiene una referencia a cada proceso en el sistema.

31

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.**
- Ahora bien, ¿cómo es la representación física de un proceso? Como mínimo, un proceso debe contar con los siguientes componentes:
  - Un programa o un conjunto de programas a ejecutar, esto es, código ejecutable.
  - Asociados con los programas, posiciones de memoria para los datos de variables locales y globales y de cualquier constante definida.
  - Una pila, esto es, un espacio adicional de memoria incluida en la ejecución del programa, utilizado para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas al sistema.
  - Atributos utilizados por el SO para controlar el proceso. El conjunto de éstos se denomina **bloque de control del proceso** (**PCB**, su sigla en inglés).
- El conjunto formado por el programa, datos, pila y atributos se denomina **imagen del proceso**.

32

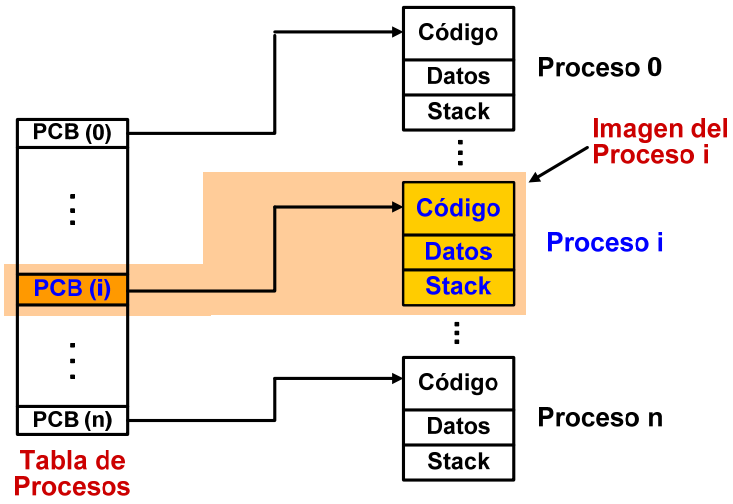
Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- La posición de la imagen del proceso dependerá de cómo el SO gestione la memoria. Usualmente, se mantiene en memoria secundaria (disco).
- Para que el SO pueda gestionar el proceso, al menos una pequeña parte de su imagen se debe mantener en memoria principal; para su ejecución, una parte mayor o completo.
- La tabla de procesos, en cambio, debe residir en memoria principal.
- Cada entrada en la tabla de procesos, contiene entonces, al menos un puntero a la imagen del proceso. Si ésta contiene múltiples bloques, puede tener referencias cruzadas entre las tablas de memoria.

33

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)



34

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Se mencionó que los **atributos del proceso** están contenidos en el bloque de control de proceso (**PCB**). Podemos agrupar la información de éste último en tres categorías generales:
  - Identificación del proceso.
  - Información de estado del procesador.
  - Información de control del proceso.

35

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Identificación del proceso: son números generados por el SO que contienen la identificación única del proceso (PID), la identificación del proceso que lo creó (proceso padre, PPID), y la identificación del usuario del proceso (UID).
- Información de estado del procesador: es el contenido de los registros del procesador:
  - Registros visibles por el usuario.
  - Registros de estado y control.
  - Puntero de pila.

36

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Información de control de proceso: necesaria para poder controlar y coordinar las actividades de los diversos procesos del sistema:
  - Información de estado y de planificación: necesaria para que el SO pueda analizar las funciones de planificación: estado del proceso, prioridad, parámetros de planificación, eventos.
  - Estructuras de datos: un proceso puede estar enlazado con otros en una cola o cualquier otra estructura, por ejemplo, en los estados de espera, o para indicar una relación padre-hijo.

37

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Información de control de proceso: (*cont.*)
  - Comunicación entre procesos: pueden asociarse banderas, señales y mensajes relativos a la comunicación entre procesos independientes.
  - Privilegios de proceso: de acuerdo a la memoria que van a acceder y los tipos de instrucciones que pueden ejecutar, como también el acceso a funcionalidades de sistema.
  - Gestión de memoria: conjuntos de punteros a tablas de segmentos del proceso que describen la asignación de memoria (mód. IV).
  - Apropiación de recursos y utilización: indica los recursos asignados y un histórico de utilización de los mismos.

38

Procesos

- **Cambio de proceso.**
- En el Módulo Introductorio vimos las técnicas de comunicación entre los componentes de una computadora, en particular las IRQ. La rutina IH se encargaba de verificar si se atendía o no la IRQ. ¿Cómo impacta esto a la hora de manejar varios procesos?
- Un cambio de proceso se produce cuando, por alguna causa, el SO retira de ejecución el proceso que está haciendo uso de la CPU, e instala otro proceso diferente.
- Las causas que pueden llevar a un cambio de proceso son las siguientes:
  - Interrupción: es producida por un evento externo al proceso. Las interrupciones pueden ser:
    - De reloj: cuando termina el tiempo de uso ininterrumpido de CPU; el proceso cambia al estado Listo y otro proceso pasará al estado En ejecución.

39

Procesos

- **Cambio de proceso.** (*cont.*)
  - Interrupción: (*cont.*)
    - De E/S: cuando un dispositivo indica al SO que ha completado una petición, por ejemplo, bloques de datos de un disco. Se mueve todos los procesos que esperaban este evento al estado Listo, y el SO decide si reanuda la ejecución del proceso interrumpido o si otorga el uso de CPU a otro proceso con mayor prioridad:
    - Por fallo de memoria: se produce cuando la CPU se encuentra con una referencia a una dirección de memoria virtual que no se encuentra en memoria principal. El proceso se bloquea hasta que el SO trae de disco el bloque que contiene la referencia y se pone en ejecución otro proceso.

40

Procesos

- **Cambio de proceso.** (*cont.*)
  - Trap: la produce un evento asociado a la ejecución de una instrucción del proceso que lleva a una condición de error. El SO detecta si esta condición es irreversible, en cuyo caso fuerza la terminación del proceso y elige otro para ejecutar.
  - Llamada al sistema: es una solicitud explícita de cambio de proceso. En este caso, un proceso del SO es quien pasa a hacer uso de la CPU.

41

Procesos

- **Cambio de proceso.** (*cont.*)
- Cuando se interrumpe la ejecución de un proceso, si el SO determina que el proceso interrumpido dejará de utilizar la CPU, se debe guardar el estado de ejecución del mismo para poder retomarlo en otro momento.
- Entonces, los pasos para un cambio de proceso son:
  - Salvar el estado del procesador.
  - Actualizar el bloque de control del proceso actual (estado).
  - Mover el proceso a la cola apropiada (Listo, Bloqueado).
  - Seleccionar un nuevo proceso a ejecutar.
  - Actualizar el bloque de control del proceso elegido (estado Ejecutando).
  - Actualizar estructuras de datos de gestión de memoria.
  - Restaurar el estado del procesador que tenía el proceso seleccionado.

42

Procesos

- **Cambio de proceso.** (*cont.*)
- Si se produce una interrupción, el SO debe determinar la causa de la misma y ejecutar la rutina de servicio de interrupción correspondiente. Para ello coloca en el contador de programa la dirección de comienzo de la rutina de manejo de interrupciones, y cambia de modo usuario a modo kernel, de manera que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.
- Dependiendo de la naturaleza de la interrupción, se puede producir o no un cambio de proceso. Al cambiar de modo de ejecución, el proceso que fue interrumpido no cambió aún de estado Ejecutando, por lo que, si la interrupción no produce un cambio de proceso, se retoma su ejecución.

43

Procesos

- **Cambio de proceso.** (*cont.*)
- Esta situación se conoce como cambio de modo, en donde la salvaguarda del estado y su posterior restauración representan sólo una ligera sobrecarga. No así un cambio de proceso completo, ya que el SO deberá realizar las tareas descriptas anteriormente, las cuales insumen una cantidad de ciclos de CPU varios órdenes de magnitud mayor.

44

Hilos

- **Uso de hilos.**
- En los SO tradicionales, cada proceso tiene un espacio de direcciones y un sólo hilo de ejecución. Sin embargo, hay situaciones en las que es conveniente tener varios hilos de control en el mismo espacio de direcciones.
- La principal razón de tener hilos es que en muchas aplicaciones se desarrollan varias actividades a la vez: un modelo de programación con ejecución cuasi-paralelo.
- Un segundo argumento es que son más ligeros que los procesos, son más rápidos para crear y destruir, típicamente de 10 a 100 veces más.
- Una tercera razón está relacionada con el rendimiento: el uso de hilos permite solapar tareas que usen sólo CPU con operaciones de E/S.
- Por último, en sistemas con múltiples procesadores es posible implementar el verdadero paralelismo.

45

Hilos

- **Uso de hilos.** (*cont.*)
- La diferencia de tener varios procesos o varios hilos es que en el caso de varios procesos tenemos varios espacios de direcciones separados, que para poder compartirlos es necesaria la intervención del SO.
- En cambio, los hilos comparten un mismo espacio de direcciones, por lo que para realizar la comunicación entre hilos no es necesario invocar llamadas al sistema, ahorrando así sobrecarga en la ejecución de la aplicación.
- Consideremos el caso de un programa **procesador de textos**. Éstos realizan varias tareas a la vez: muestran en pantalla el texto que se escribe, capturan los caracteres y los clics de mouse para hacer alguna acción en particular, realizan copias de respaldo del archivo que se está editando, etc.

46

Hilos

- **Uso de hilos.** (*cont.*)
- Si estas tareas se realizaran en un único proceso, cada vez que se inicie un respaldo en disco del texto se ignorarían los comandos de teclado y mouse hasta que termine el respaldo; o bien el respaldo ser interrumpiría al detectar entradas de teclado. El usuario considerará esto como un rendimiento pobre de la aplicación.
- Tampoco funcionaría utilizar varios procesos, ya que cada uno necesitará acceder al archivo del texto, lo que implica cambiar la asignación del recurso entre los procesos, lo cual implica una sobrecarga importante.
- Si en cambio, se utiliza un proceso con varios hilos, cada uno de ellos ejecutando una tarea de las descripta, estas pueden incluso ejecutar en un cuasi-paralelo, además de la ventaja de compartir los recursos asignados al proceso.

47

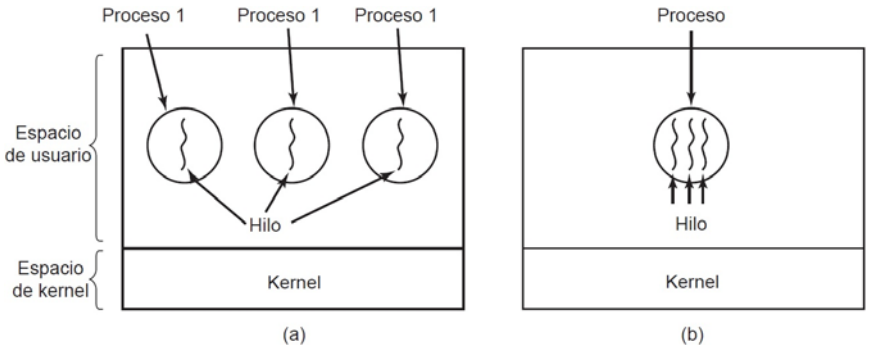
Hilos

- **Modelo clásico de hilos.**
- El modelo de proceso se basa en 2 conceptos independientes: agrupamiento de recursos y ejecución.
- Una manera de ver un proceso es como una forma de agrupar recursos relacionados. Tiene un espacio de direcciones que contiene código y datos del programa, archivos abiertos, procesos hijos, alarmas, señales, etc. El proceso es, por tanto, una unidad de posesión de recursos.
- Otro concepto que tiene el proceso es el *hilo de ejecución*, llamado simplemente **hilo**. Éste tiene un contador de programa, registros para almacenar variables, una pila, etc. El hilo es, entonces, una unidad de ejecución.
- Lo que agregan los hilos al modelo de procesos es permitir que se lleven a cabo varias ejecuciones independientes en el mimo entorno del proceso.

48

Hilos

- **Modelo clásico de hilos.** (*cont.*)
- Comparación entre múltiples procesos independientes y múltiples hilos independientes:



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

49

Hilos

- **Hilos en POSIX.**
- El IEEE ha definido un estándar para los hilos: 1003.1c. El paquete de hilos se conoce como **Pthreads**. La mayoría de los sistemas compatibles con POSIX aceptan éste paquete.

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

Algunas de las llamadas a funciones de Pthreads.

50

Hilos

- **Hilos en el espacio de usuario.**
- Existen dos categorías de implementación de hilos en SO: hilos en el espacio de usuario (user-level threads, ULT) e hilos en el espacio de núcleo (kernel-level threads, KLT).
- En el enfoque de hilos en espacio de usuario, la aplicación gestiona todo el trabajo de los hilos y el núcleo del SO no es consciente de la existencia de los mismos. Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos, que es un paquete de rutinas para la gestión de ULT.
- Por defecto, una aplicación comienza con un hilo, los que se localizan en un solo proceso gestionado por el núcleo. La creación, destrucción y planificación de los hilos se realiza llamando a funciones de la biblioteca de hilos.

51

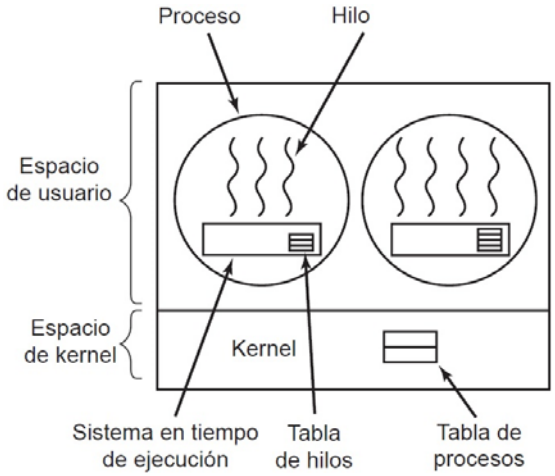
Hilos

- **Hilos en el espacio de usuario.** (*cont.*)
- Las ventajas de utilizar ULT son:
  - El cambio de hilo no requiere privilegios de modo núcleo, ya que todas las estructuras de datos se encuentran en el espacio de usuario, lo que ahorra la sobrecarga de los cambios de modo.
  - La planificación la puede realizar la aplicación, eligiendo el algoritmo más conveniente para cada aplicación.
  - Los ULT pueden ejecutar en cualquier SO; no se necesitan cambios en el núcleo para dar soporte a los ULT.
- Desventajas de utilizar ULT:
  - En un SO típico, muchas llamadas al sistema son bloqueantes, por lo que, si un hilo realiza una llamada al sistema, no sólo se bloquea el hilo sino todos los hilos del proceso.
  - No se puede sacar provecho de sistemas multiprocesadores, ya que el núcleo asigna el proceso a un solo procesador al mismo tiempo. Esto implica que, en determinado momento, sólo puede ejecutar un hilo del proceso, aunque sí se puede aplicar multiprogramación.

52

Hilos

- **Hilos en el espacio de usuario.** (*cont.*)
- Modelo de hilos en espacio de usuario:



53

Hilos

- **Hilos en el espacio de núcleo.**
- En este enfoque, la gestión de los hilos la realiza el núcleo del SO. No hay código de gestión en la aplicación, sólo una interfaz (API) para acceder a las utilidades de hilos del núcleo. Además, el núcleo mantiene información del contexto del proceso como una entidad y de los hilos individuales del proceso.
- Este enfoque resuelve los dos principales problemas del ULT:
  - Por un lado, el núcleo puede planificar simultáneamente múltiples hilos de un solo proceso en múltiples procesadores.
  - Por otro lado, si se bloquea un hilo, el núcleo puede planificar otro hilo del mismo proceso. Además, las rutinas del núcleo pueden ser en sí mismas multihilo.

54

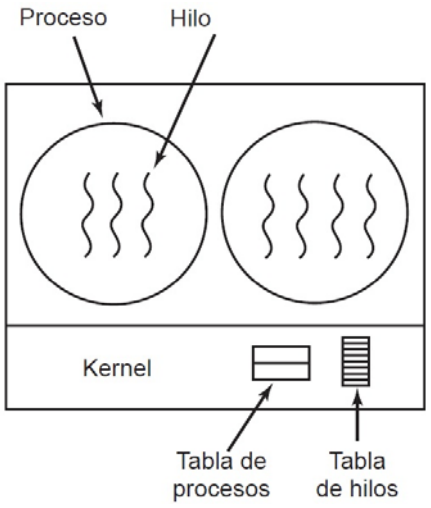
Hilos

- **Hilos en el espacio de núcleo.** (*cont.*)
- La principal desventaja del enfoque KLT es que la transferencia de control de un hilo a otro requiere de un cambio de modo al núcleo, ya que se crean y gestionan en su espacio de direcciones.
- Los SO multihilos presentan mayor performance global; sin embargo, el máximo beneficio se obtiene cuando las aplicaciones de usuario también son programadas multihilos.

55

Hilos

- **Hilos en el espacio de núcleo.** (*cont.*)
- Modelo de hilos en espacio de núcleo:



56

Fin del Módulo II

57