Xestión de Procesos

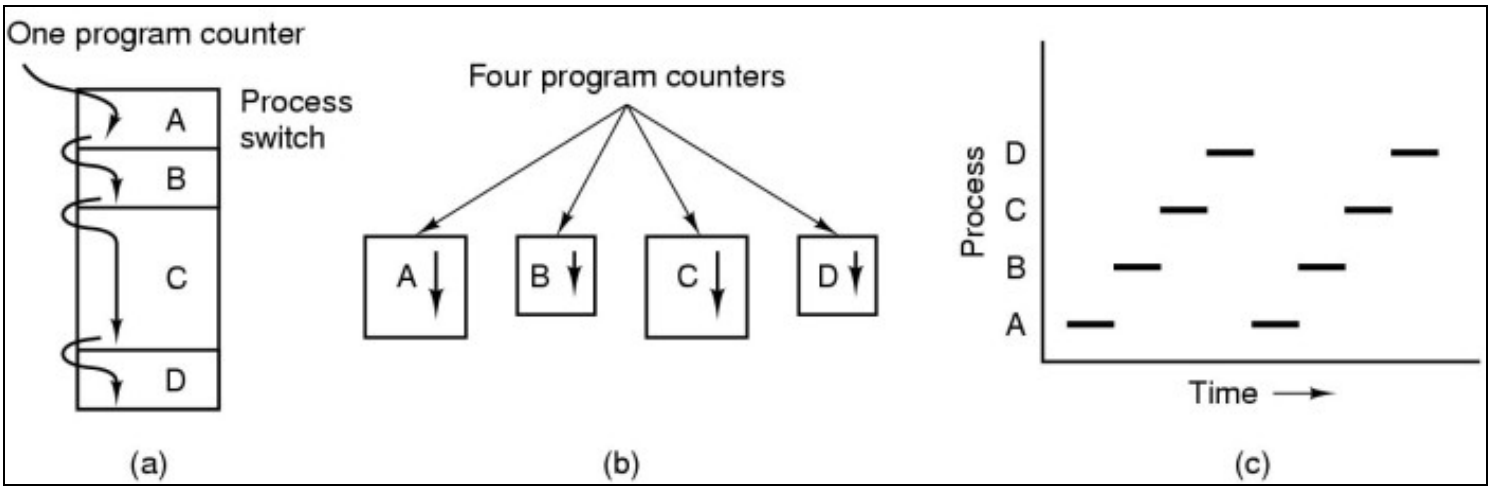
Sumario

Concepto de Proceso

- **Definición**: Un proceso se define como una instancia de un programa en ejecución. Un programa es un archivo binario en algún tipo de almacenamiento masivo (persistente), sin embargo cuando ese programa necesita ser ejecutado es necesario que el Sistema Operativo le asigne los recursos necesarios y establezca el control pertinente sobre la ejecución del programa. A la abstracción generada por los Sistemas Operativos para gestionar el recurso de procesamiento (CPU) se le denomina proceso. En resumen, un proceso es una abstracción creada por el Sistema Operativo para representar un programa en ejecución. Mediante esta abstracción el Sistema Operativo puede controlar la ejecución del Proceso y asignarle y controlar el acceso a los Recursos.

- **Multitarea**: La característica principal que permite que en la actualidad los Sistemas Operativos resulten "útiles" es la capacidad de éstos de poder ejecutar varios procesos a la vez. A esta capacidad se la denomina Multitarea o Multiprogramación. Hace muchos años los Sistemas Operativos eran Monotarea, esto quiere decir que solamente podían ejecutar una tarea o proceso en el Sistema. Además esta tarea se ejecutaba de principio a fin, sin interrupción posible, de modo que no era viable el detener un proceso a mitad de su ejecución para pasar a ejecutar otro. Si por ejemplo, el proceso tenía que realizar una lectura de datos de un dispositivo externo, como una cinta, lo cual es tremendamente lento, el proceso debía esperar en estado de ejecución a que la operación de acceso terminara, mientras la CPU estaba prácticamente ociosa a la espera de los datos leídos de la cinta. El paradigma clásico de este tipo de Sistemas era MS-DOS. En este Sistema Operativo un único proceso controlaba la ejecución de comandos o órdenes por parte del usuario. El usuario introducía una orden y esta se ejecutaba de principio a fin, cuando la orden devolvía la respuesta al usuario éste recibía de nuevo el control de la terminal y consola y podía introducir el siguiente comando.

En la actualidad los Sistemas Operativos son **Multitarea** y, por tanto, varios procesos pueden ejecutarse concurrentemente (aunque no simultáneamente porque el número de unidades de ejecución o CPUs está limitado por el hardware). Aún así es posible para el Sistema Operativo detener un proceso en medio de su ejecución y pasar a ejecutar otro. Más tarde el primero se podría volver a ejecutar en el mismo punto en el que fue interrumpido. En un sistema multitarea, el ejemplo anterior de la lectura en cinta se gestionaría deteniendo el proceso y pasando a ejecutar otro que pudiera utilizar la CPU mientras el primero espera la recepción de los datos leídos de la cinta. De este modo el uso de la CPU se vuelve mucho más eficiente y el sistema operativo consigue que los procesos avancen concurrentemente, sin que los recursos de la máquina sean desaprovechados. Al cambio del estado de ejecución de un Proceso por parte del Sistema Operativo para poder ejecutar otro distinto se denomina "Cambio de Contexto" y es el mecanismo fundamental que permite la ejecución de Sistemas Multitarea.



- **Asignación del tiempo de CPU en multitarea**

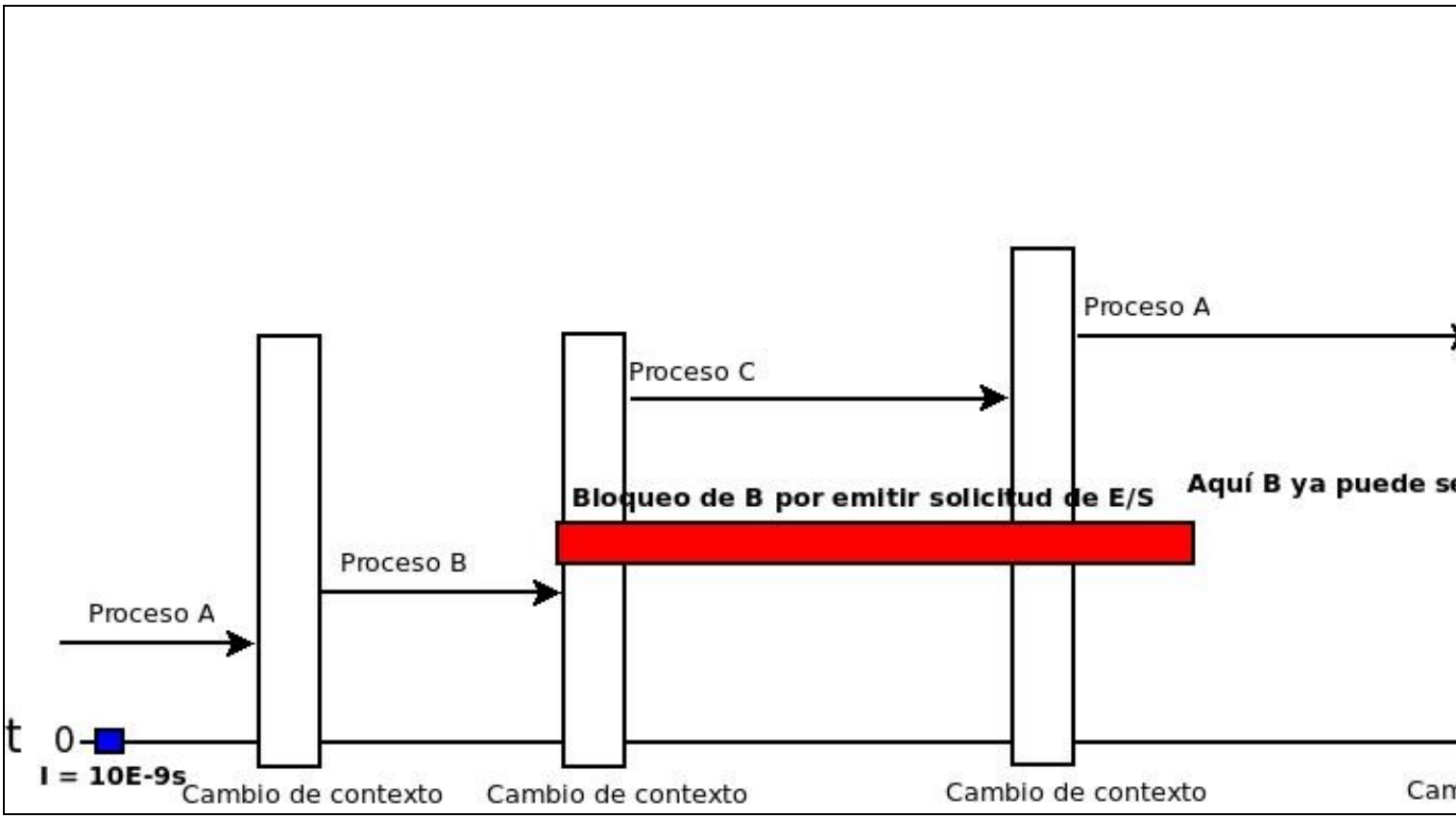
En el siguiente gráfico se ilustra el modo en que la CPU se "reparte" entre los procesos listos para ejecutarse. En este caso tenemos 3 procesos, de nombres A, B y C. Cada proceso se ejecutará durante un tiempo de CPU determinado y supervisado por el sistema operativo. El gráfico ilustra un intervalo de tiempo de ejecución de 1 segundo en una CPU arbitraria. Se menciona también la escala de tiempo significativa para la CPU mediante la identificación de un intervalo I de **1x10E-9s**, es decir, una mil millonésima de segundo, o lo que es lo mismo, una escala de tiempo correspondiente a una frecuencia de **1Ghz**. Con esto quiere hacerse notar que para una CPU, actuando en esa escala de tiempo, 1 segundo es un intervalo temporal muy significativo, durante el que puede avanzar la ejecución de varios procesos en secuencia (en el ejemplo A, B y C, pero en la práctica pueden ser muchos más). De este modo, bajo la limitación de la percepción humana, que opera en la escala del segundo, **la impresión subjetiva** es que la CPU ejecuta "simultáneamente" varios procesos a la vez, cuando ésto realmente no es así.

Durante la ejecución del mismo, el proceso:

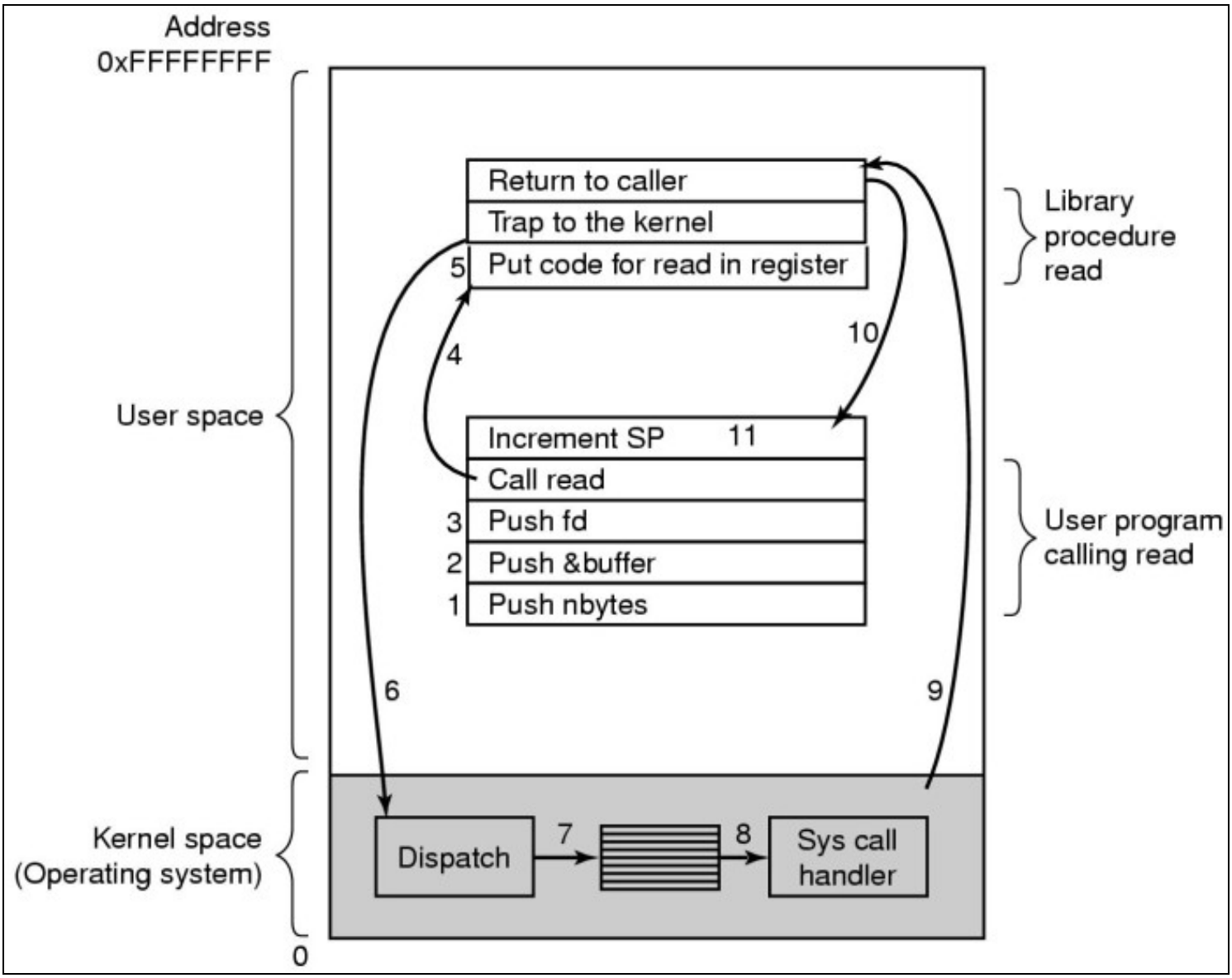
- Se bloquea en una operación de solicitud de Entrada y Salida: Como es el caso de B en el gráfico
- Es detenido por el sistema operativo transcurrido un tiempo, siendo elegido otro proceso para continuar

En cualquiera de los dos supuestos anteriores, entre la parada del proceso y la ejecución del siguiente proceso elegido, transcurre un tiempo necesario para que el sistema operativo detenga el proceso "saliente" y ponga en marcha el proceso "entrante". A este tiempo se le denomina **cambio de contexto** y durante éste el sistema operativo guarda la información de ejecución de los registros de la CPU del proceso saliente y recupera la información de ejecución de la memoria principal, cargándola en los registros de la CPU, para el proceso entrante.

Otro aspecto importante es que, en caso de que un proceso se bloquee en espera de la realización de una operación de E/S, como le ocurre a B, no se podrá retomar la ejecución de ese proceso hasta que dicha operación haya concluido. Observamos un rectángulo rojo en el gráfico que identifica precisamente este hecho. La naturaleza "lenta" de las operaciones de E/S causan el mencionado bloqueo y para la CPU ese proceso simplemente se queda "fuera de juego" hasta que los datos de la operación de E/S estén en memoria, en caso de una operación de lectura, o haya concluido la operación de escritura, en caso de que la naturaleza de la operación de E/S fuera escribir información en un dispositivo. Cuando la operación concluye se avisa al sistema operativo, mediante una **interrupción hardware**, para que cambie el estado del proceso bloqueado en esa operación a listo o "elegible" para continuar con su ejecución.



- **Modos de ejecución de la CPU**: Como es sabido la CPU tiene dos modos de ejecución, Modo Privilegiado (o Modo Kernel) y Modo No Privilegiado (o Modo Usuario). La diferencia principal es que en Modo Privilegiado el proceso en ejecución tendría acceso a todos los recursos hardware del Sistema. La política general en los Sistemas es que solamente el Sistema Operativo se pueda ejecutar en Modo Privilegiado y los programas y aplicaciones de usuario lo hagan en Modo Usuario. De este modo cuando un proceso de usuario necesita acceder a alguna característica de acceso a algún recurso del Sistema, como abrir un archivo, comunicarse con otro proceso, obtener más memoria, etc., deberá invocar al Sistema Operativo para que sea éste el que determine si se le concede o no lo solicitado. A este mecanismo, mediante el cual los procesos de usuario solicitan recursos al Sistema Operativo, se le denomina Llamada al Sistema.

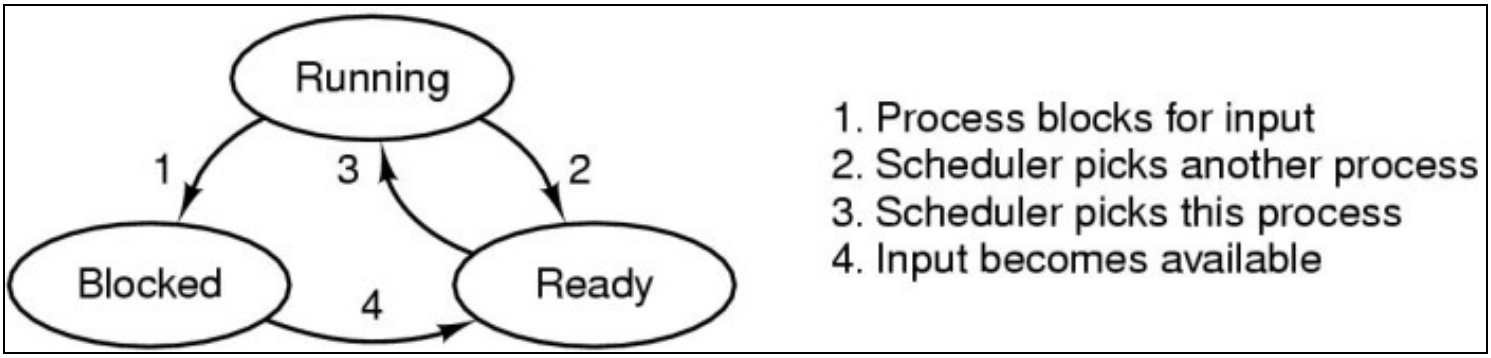

Address
0xFFFFFFFF
User space
Kernel space
(Operating system)
0
Return to caller
Trap to the kernel
5
Put code for read in register
Library
procedure
read
10
4
Increment SP
11
Call read
3
Push fd
2
Push &buffer
1
Push nbytes
User program
calling read
6
9
Dispatch
7
8
Sys call
handler

Modelo de Procesos en ejecución

- **Modelo simplificado de CPU virtual**: El Modelo de CPU virtual hace referencia a que con la Multitarea el Sistema Operativo ofrece al usuario la "ilusión" de que se están ejecutando sus procesos de modo exclusivo y dedicado en la CPU. Según este modelo, irreal pero simple (esta es una de las funciones del Sistema Operativo, ofrecer al usuario una visión de Máquina Extendida), los usuarios percibirían como sus procesos avanzan en su ejecución sin tener que preocuparse de cómo se controlan los Cambios de Contexto, es decir, la asignación de la CPU a otros procesos, típica de los Sistemas Multitarea. De este modo, se percibiría un CPU virtual por cada proceso, cuando realmente lo que ocurre es que la Unidad de Ejecución (o Unidades de Ejecución en Sistemas Multicore) se comparte, entre todos los procesos que se pueden ejecutar.

- **Estados de los procesos:** Mediante este concepto el Sistema Operativo puede organizar los Cambios de Contexto y decidir acerca de que proceso pasará a ejecutarse a continuación. Existen modelos de Estados de Procesos de hasta 7, o más, estados, sin embargo con un modelo básico de 3 estados podemos explicar fácilmente este concepto.

Los 3 estados mencionados son: **Ejecución**: Proceso que se está ejecutando en la CPU **Listo**: Proceso que podrá ejecutarse cuando el Sistema Operativo le ceda turno y le asigne la CPU **Bloqueado**: Proceso que no puede seguir ejecutándose, es decir pasar a Listo, hasta que no ocurra algún evento por el que está esperando, como por ejemplo la lectura de un archivo del Disco a la Memoria RAM.



- **Creación y Terminación de Procesos**: Las acciones de creación y terminación de procesos, como todo, es labor del Sistema Operativo. Cuando se crea un proceso se le asignarán recursos de ejecución e información de control que el Sistema Operativo utiliza para que el proceso pueda ejecutarse. Del mismo modo, al destruir un proceso el Sistema Operativo tendrá que liberar aquellos recursos, estructuras de control y datos que el proceso necesitaba para su ejecución.

El Sistema Operativo puede crear procesos cuando: Se inicie el Sistema Otro proceso o un usuario del Sistema lo solicite **El Sistema Operativo terminará un proceso cuando:** Finalice correctamente su trabajo Se produzca algún error previsto que le impida seguir su ejecución Se produzca un error excepcional no previsto que haga que el proceso no pueda continuar Se apague el Sistema

- **Jerarquía de Procesos**: La naturaleza de las interrelaciones de procesos es compleja y dependiente del contexto, no funciona igual en todos los Sistemas Operativos. Es común, como en el caso de los Sistemas basados en UNIX/Linux, que exista una jerarquía de procesos, de modo que un proceso "padre" puede crear otros procesos "hijos". El modelo padre-hijo simplifica el proceso de creación de procesos y permite estructurar de un modo sencillo los procesos en el Sistema. Un proceso padre controla a todos sus hijos, si el padre es destruido todos sus hijos desaparecen con él, al revés no. Otros Sistemas Operativos, como Windows, no utilizan directamente el concepto de Jerarquía de procesos, en el sentido padre-hijo, aunque sí reconocen relaciones de dependencia entre procesos.

Implementación de Procesos

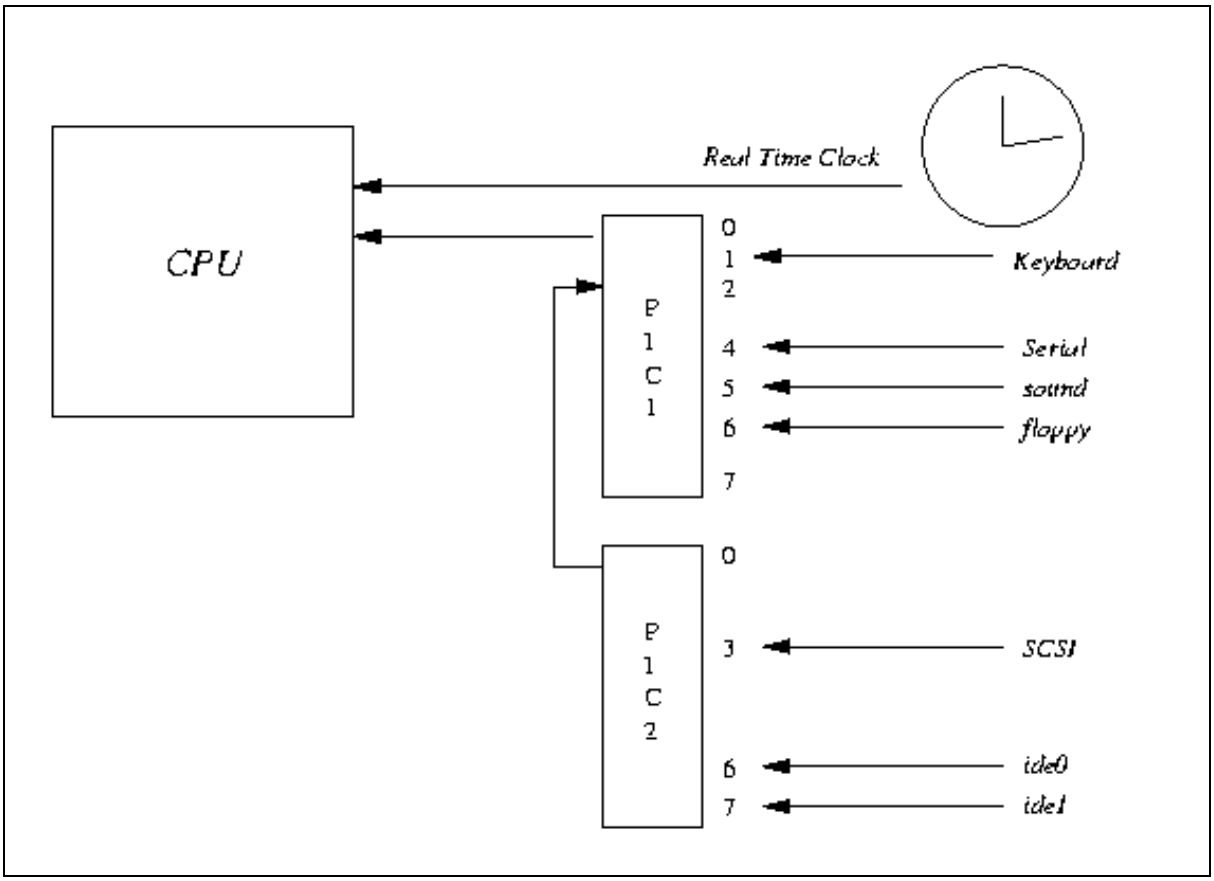
- **Tablas de Control de Procesos**: Los Cambios de Contexto dentro del Sistema Operativo, esto era la conmutación del Estado de ejecución de un proceso a otro, que permitía implementar la Multitarea, requieren que un proceso en ejecución se detenga en un instante determinado y más tarde, tras que uno o más procesos se ejecuten el tiempo que determine el Sistema Operativo, retome su estado de ejecución exactamente en el mismo punto en el que se detuvo. Recordemos que cuando un proceso se ejecuta en la CPU escribe en los registros de la Unidad de Control y la ALU la información relativa a su estado de ejecución (Contador de Programa, Puntero de Pila, Registro de Instrucción, Registro de Estado, Segmentos de Direccionamiento, etc). Toda esa información deberá almacenarse en un lugar seguro para más tarde poder ser recuperada y reincorporada a los registros de la CPU, para de este modo continuar su ejecución en el mismo punto en el que se detuvo. Este lugar "seguro" es la **Tabla de Procesos**, una estructura mantenida por el Sistema Operativo, en general en la Memoria RAM, con copia en disco. Por tanto, la Tabla de Procesos es la estructura dentro del Sistema Operativo que ayuda a organizar y controlar los Estados de los Procesos, mediante el almacenamiento en ella de toda la información de ejecución sobre el proceso que el Sistema Operativo necesita conocer. A continuación podéis ver un esquema de la información mantenida por la Tabla de Procesos del Sistema Operativo, en

ella puede observarse como se guarda información de gestión de los Procesos, pero también de la Memoria y de los Archivos

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

- **Interrupciones y Cambios de Contexto**: Entonces, según lo anterior parece que todo funciona coordinado y controlado por el Sistema Operativo, ¿no?. La respuesta es claramente afirmativa. Pero, hay que tener en cuenta lo siguiente: El Sistema Operativo es un proceso más dentro del Sistema, entonces, ¿cómo es posible que el Sistema Operativo determine cuando un proceso es detenido para darle paso a otro en el momento en que otro proceso está en ejecución?. La respuesta está en el mecanismo de Interrupciones del hardware. Una Interrupción es un evento hardware que produce una respuesta de atención automática por parte de la CPU cuando esto ocurre. Abreviadamente, la CPU comprueba en cada ciclo de instrucción si se ha habilitado alguna línea de Interrupción directa o indirectamente conectada. Si la respuesta es negativa continúa con la siguiente instrucción del proceso en ejecución; si resulta que ha ocurrido una interrupción y se ha activado una de esas líneas, entonces, tiene que interrumpir el proceso en ejecución y atender a la causa de esa Interrupción.

Por ejemplo, cuando un proceso necesita acceder al contenido de un archivo en disco, éste, para poder ser procesado deberá leerse del disco y copiarse total o parcialmente en la memoria RAM. Las lecturas de disco son tremendamente lentas en comparación con la velocidad de ejecución de la CPU y la Memoria principal, así que, lo que ocurre, es que el proceso se bloquea y espera a que el contenido del archivo esté accesible en RAM. Cuando el disco termina de copiar los datos a la RAM, debe indicar de algún modo a la CPU que el proceso ya está listo para poder continuar (es decir para que haga una transición del estado Bloqueado a Listo). El modo de conseguirlo es la emisión de una Interrupción que notifique a la CPU directamente, e indirectamente al Sistema Operativo, de que el proceso ya puede continuar. Ahora bien, ¿qué ocurriría con un proceso que está ejecutándose y no emite ningún tipo de petición bloqueante, como la lectura de un archivo del disco o el acceso a algún recurso ocupado o que lo haga bloquearse?. La respuesta, en principio, sería que ese proceso continua su ejecución hasta que termine. El caso es que esta política no es buena para Sistemas Operativos Multitarea, pues correríamos el riesgo de que un proceso monopolice el uso de la CPU y otros no puedan ejecutarse periódicamente. Para evitar esto, el hardware, el reloj concretamente (Real Time Clock, RTC), emite periódicamente una señal de Interrupción que hace que el Sistema Operativo se ejecute para controlar la situación anterior. De este modo es posible asignar tiempos de ejecución máxima de los procesos durante los cuales podrían ejecutarse sin que otros los releven, salvo que se bloqueen claro. A este tiempo de ejecución máxima se le denomina **quantum** y es un parámetro utilizado por la mayoría de Sistemas Operativos para controlar la Multitarea. En resumen, las Interrupciones son los puntos de unión del Hardware con el Software, es decir, el Sistema Operativo controla todo el Sistema de Ejecución de procesos en la medida en que es el que recibe y controla las Interrupciones.



Algoritmos de Planificación de Procesos

El requisito de multiprogramar varios procesos en un único procesador lleva a la necesidad de definir estrategias para realizar tal operación. Existen varias aproximaciones y soluciones a esta cuestión, materializadas en una serie de algoritmos de planificación o calendarización. El módulo del Sistema Operativo encargado de realizar este trabajo se denomina **planificador o calendarizador**.

Existen varios objetivos o metas que es persiguen a la hora de definir un algoritmo de planificación, algunas de ellas son:

- **Equidad**: Procesos equivalentes deben obtener un servicio similar
- **Tiempo de respuesta**: El tiempo de respuesta es el tiempo que transcurre entre la petición de un usuario interactivo y la respuesta a esa petición por parte del sistema. El objetivo, como es natural, será minimizar el tiempo de respuesta
- **Tiempo de retorno**: Tiempo transcurrido desde la llegada de un trabajo por lotes (no interactivo) al sistema y su finalización completa
- **Rendimiento**: Maximizar el número de trabajos terminados por unidad de tiempo
- **Eficiencia**: Mantener ocupada a la CPU el 100% del tiempo

Un buen algoritmo de planificación debe conseguir de un modo sencillo y flexible esos objetivos. Existen una gran cantidad de algoritmos de planificación que han sido implantados y utilizados a lo largo de la historia de la computación, veremos a continuación algunos de los más significativos.

- **Primero en llegar, Primero en ser atendido (FCFS, First Come First Served)**

Este algoritmo procesa los trabajos (o procesos) a través de una cola simple. Cuando un proceso se crea en el sistema se pone en la cola de procesos Listos para ejecutarse y espera su turno. La ventaja de este algoritmo es su sencillez pero es muy ineficiente pues el procesador no puede ser entregado a otro proceso mientras el proceso actual no termine, incluso si se bloquea. Por tanto FCFS no es adecuado para la multiprogramación.
NOTA: Aquellos algoritmos que permiten que un proceso se detenga, es decir pasar a estado bloqueado o listo, antes de que finalice su ejecución se dice que son algoritmos con **expropiación**. Los algoritmos que requieren que los procesos se ejecuten de principio a fin, sin cambios de contexto se dice que son **sin expropiación**

- **Trabajo más corto primero (SJF, Shortest Job First)**

En los tiempos de los sistemas por lotes, cuando no había usuarios que trabajaban en modo interactivo, y menos con interfaces gráficas de usuario, interesaba terminar el mayor número de trabajos por hora, aumentando el rendimiento y obteniendo un tiempo medio de retorno óptimo. Este algoritmo consiste en seleccionar, de los procesos Listos en el sistema, aquel cuyo tiempo de ejecución será menor. Esto implica conocer de antemano el tiempo

de ejecución de un proceso, información que en sistemas por lotes puede ser determinada por la experiencia previa. La desventaja principal de este algoritmo es que discrimina a los procesos con tiempo de ejecución mayor y si en el sistema existen muchos procesos con bajos tiempos de ejecución, la demora de los más largos puede ser insostenible.

- **Turno Rotatorio (RR, Round Robin)**

Este algoritmo es una versión expropiativa (en el sentido de que el Sistema Operativo expropia el procesador a los procesos) de FIFO, los procesos se mantienen en una cola de procesos Listos, esperando su turno para ejecutarse. Sin embargo, cuando un proceso se bloquea, o cuando transcurre un tiempo máximo preestablecido (denominado quantum), el Sistema Operativo otorga el procesador a otro de los procesos Listos, de este modo se dinamiza la ejecución de los procesos, optimizando la utilización del procesador y operando el sistema en modo multiprogramado. El período máximo de ejecución de un proceso, quantum, suele ser un parámetro configurable por parte del Sistema Operativo. Un quantum largo, por ejemplo 300 ms (pensemos en que los procesadores actuales trabajan en escalas de tiempo de nanosegundos, es decir la millonésima parte de un milisegundo), puede provocar que los procesos a la espera de ejecutarse sufran demoras importantes y en sistemas interactivos puede dar la impresión de lentitud en la respuesta del sistema. Por el contrario, un quantum corto, por ejemplo 20 ms, provocaría que los cambios de contexto entre procesos (operación que vimos que era costosa en términos se rendimiento) fuesen demasiado frecuentes, sobrecargando el sistema y manteniendo al procesador inactivo. Es por tanto necesario elegir un valor de quantum adecuado para el sistema. En la práctica, en sistemas interactivos, suelen utilizarse períodos de quantum en torno a los 50 ms.

- **Colas de Prioridad**

Una generalización de Turno Rotatorio surge al introducir el concepto de prioridad de un proceso. En el sistema se mantienen varias colas de procesos, una para cada nivel de prioridad de los mismos. Este algoritmo elige para ejecutar a continuación aquel proceso Listo dentro de la cola de mayor prioridad según un esquema de turno rotatorio, es decir con quantum y expropiación. Cuando los procesos Listos de una cola de máxima prioridad estén todos bloqueados o hayan terminado su ejecución se procederá a seleccionar procesos Listos de la cola de prioridad inmediatamente inferior. Este algoritmo suele perjudicar aquellos procesos de menor prioridad, por ese motivo es común que las prioridades sean dinámicas y el Sistema Operativo las vaya actualizando periódicamente a lo largo de la ejecución de un proceso.

- **Otros**

Otros algoritmos, menos utilizados: Calendarización garantizada, Calendarización por lotería, Tiempo restante de ejecución más corto, etc.

Modelo de Hilos y diferencia entre Proceso e Hilo

- **Definición y diferencia entre Hilo y Proceso**: En los Sistemas Operativos actuales se utilizan dos conceptos similares, aunque diferentes, a la hora de gestionar la ejecución de procesos. Por un lado el concepto de proceso mismo se utiliza como unidad de asignación de recursos y se usa el término hilo (thread o subproceso) para hacer referencias a las unidades de ejecución de un proceso. En general un proceso consta de varios hilos de ejecución. Cada hilo de ejecución de un proceso se ocupa de realizar alguna tarea específica en relación al proceso. Estructurar los proceso en hilos mejora el rendimiento general del Sistema y hace más flexible y cómoda la programación eficiente de ciertas tareas. Por tanto siempre que usemos un punto de vista centrado en los recursos de los procesos (espacio de direcciones, archivos que utiliza el proceso, archivos de dispositivos, señales, estructuras compartidas, etc.) estaremos manejando implícitamente la noción de proceso. Cuando pensemos en la ejecución de un proceso, es decir, los aspectos relacionados con el tiempo durante el cual el proceso que se ejecuta en la CPU estaremos manejando la noción de hilo(s). En resumen, los hilos son los "subprocesos" de un proceso que se envían a ejecución a la CPU, por tanto un proceso puede constar de varios hilos y los recursos que comparten esos hilos, es decir el contexto que integra todos los hilos es el proceso en sí. En la siguiente sección veremos un ejemplo.

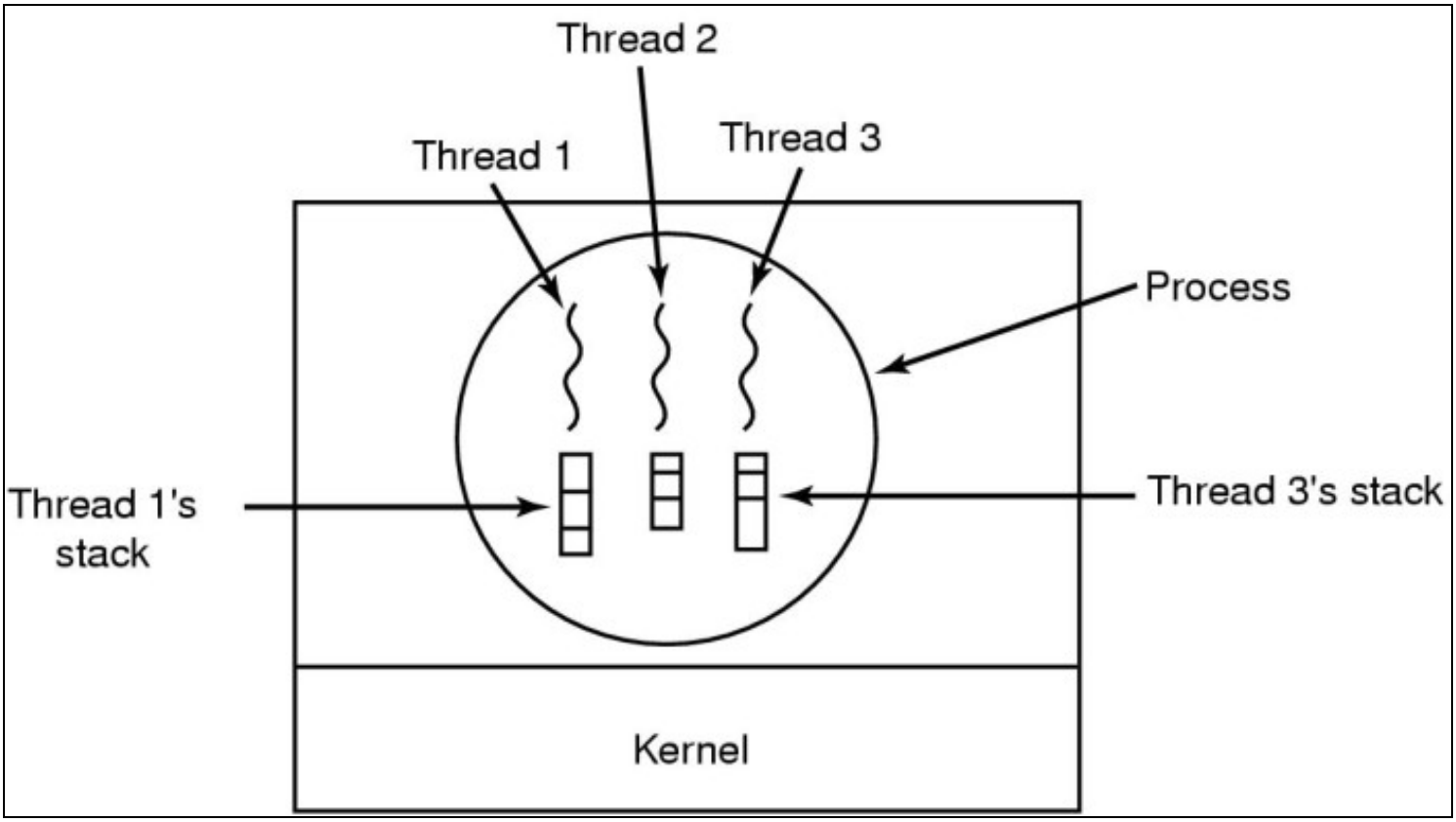
Información de control para procesos e hilos mantenidos por el Sistema Operativo:

Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- **Uso de Hilos**: Imaginemos una empresa en la que hay 20 técnicos informáticos y 1 administrativo. Esta empresa se parecería un poco a un Sistema que no distinguiera entre procesos e hilos, veamos porqué. Pensad que el administrativo es el encargado de recibir todas las peticiones de asistencia técnica de la empresa, por tanto, siempre que un cliente entre por la puerta es él el encargado de tomar nota de la incidencia y derivársela a uno de los técnicos para que la solucione. Si en un momento dado llegan muchos clientes y el administrativo está ocupado, por ejemplo atendiendo el teléfono, muy probablemente los técnicos estén gran parte de su tiempo sin hacer nada pues el administrativo no es capaz de procesar los clientes al mismo ritmo al que llegan. El problema en esta empresa es que hay un elemento "bloqueante" que supedita la prestación del servicio de asistencia técnica a su disponibilidad de tiempo libre. Esto es, más o menos lo que podría ocurrir en un Sistema Operativo con procesos de un solo hilo, en cierta medida ese único hilo sería como el administrativo del ejemplo anterior. Si pensamos que la solución al problema podría ser hacer que 4 o 5 de los técnicos hiciesen de receptores de incidencias, seguramente los técnicos estarían ocupados y no estarían ociosos esperando a recibir trabajo.

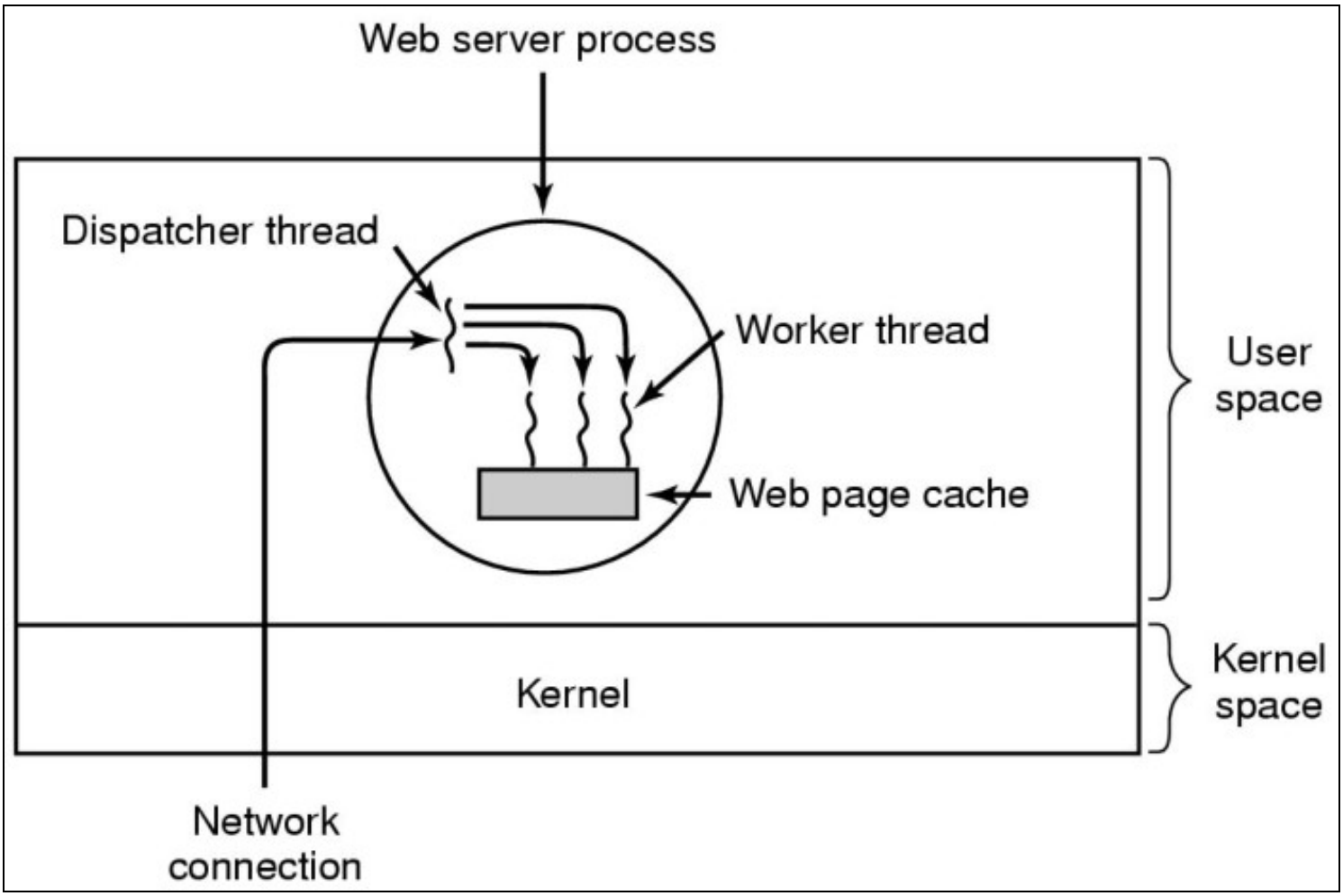
Desde un punto de vista más técnico podría decirse que cuando un proceso emite una Llamada al Sistema (petición al Sistema Operativo para hacer algún tipo de tarea que el no puede realizar directamente al no ejecutarse en modo privilegiado) éste se bloque a la espera que el Sistema Operativo le otorgue lo que necesite (aquello para lo que emitió la llamada). Ciertas Llamadas al Sistema son bloqueantes y hacen que el proceso tenga que pasar a estado Bloqueado, deteniendo su ejecución y no pudiendo avanzar hasta que se complete la operación solicitada. Si dividimos un proceso en hilos es posible hacer que aunque uno de ellos se bloquee en espera de una Llamada al Sistema, otro hilo del mismo proceso pueda ejecutarse y hacer algún trabajo productivo para el proceso. Este es el objetivo principal y la causa de que exista el concepto de hilo en los Sistemas modernos. Además, como cada vez las CPUs son más rápidas, los procesos tienden a bloquearse muy rápido y a desperdiciar tiempo de CPU, aumentando los Cambios de Contexto. Los hilos ayudan a mejorar enormemente el rendimiento en estos casos.

Gráfico de procesos e hilos dentro de un proceso:



- **Hilos en Espacio de Usuario y Kernel**: Existen dos modos de implementar la gestión de hilos en el Sistema Operativo. Bien en el espacio de usuario, como un programa, o mejor dicho una librería de soporte a los programas, o bien en espacio de kernel, es decir, como parte constitutiva del propio Sistema Operativo

Ejemplo de uso de hilos en un proceso de servidor web:



Comunicación y sincronización entre Procesos

Los procesos utilizan los recursos del sistema bajo la supervisión y control del Sistema Operativo. En ocasiones ocurren conflictos derivados de la compartición de ciertos recursos entre los procesos. Por este motivo el Sistema Operativo debe de realizar una labor de arbitraje que evite que esa compartición origine problemas de consistencia en el sistema. Un ejemplo de esta situación lo constituyen los problemas de actualización concurrente de variables compartidas de memoria, a la cual pueden acceder varios procesos. Si no se controlan los accesos a esas variables ocurren problemas de inconsistencia y modificaciones perdidas del valor de la variable

La solución pasa por delimitar claramente aquellas partes de un programa que acceden a recursos compartidos que pueden dar lugar a las denominadas **condiciones de competencia o carrera**, término que indica la inconsistencia potencial derivada de compartir esa información. A las partes de un programa dónde se accede a información y recursos que pueden dar lugar a condiciones de competencia se las denomina **secciones o regiones críticas**, siendo utilizados indistintamente ambos términos. Será labor del Sistema Operativo o del programador de aplicaciones garantizar la coherencia y consistencia del sistema derivado de las acciones de los procesos. Existen varias alternativas y aproximaciones para proporcionar una solución a este problema pero caen fuera del ámbito de este curso. En el libro Sistemas Operativos Modernos de Tanenbaum podéis encontrar más información.

Otros contextos dentro relacionados con la Comunicación entre Procesos son los mecanismos de paso y compartición de información entre ellos. Existen muchos mecanismos que posibilitan esta compartición, como son el Paso de Mensajes, las Señales y las relaciones de tipo Padre-Hijo entre procesos en aquellos Sistemas Operativos que las soporta, como Linux.

Referencias

Planificación de Procesos

http://informatica.iessanclemente.net/manuais/index.php/Algoritmos_de_Planificaci%C3%B3n_da_CPU

Comandos de monitorización útiles para los Administradores

http://www.cyberciti.biz/tips/top-linux-monitoring-tools.html

Volver

JavierFP 18:13 03 dec 2018 (CET)