PROCESOS E HILOS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION-FACYT-UC
ASIGNATURA: SISTEMAS OPERATIVOS
PROFESORA PhD MIRELLA HERRERA

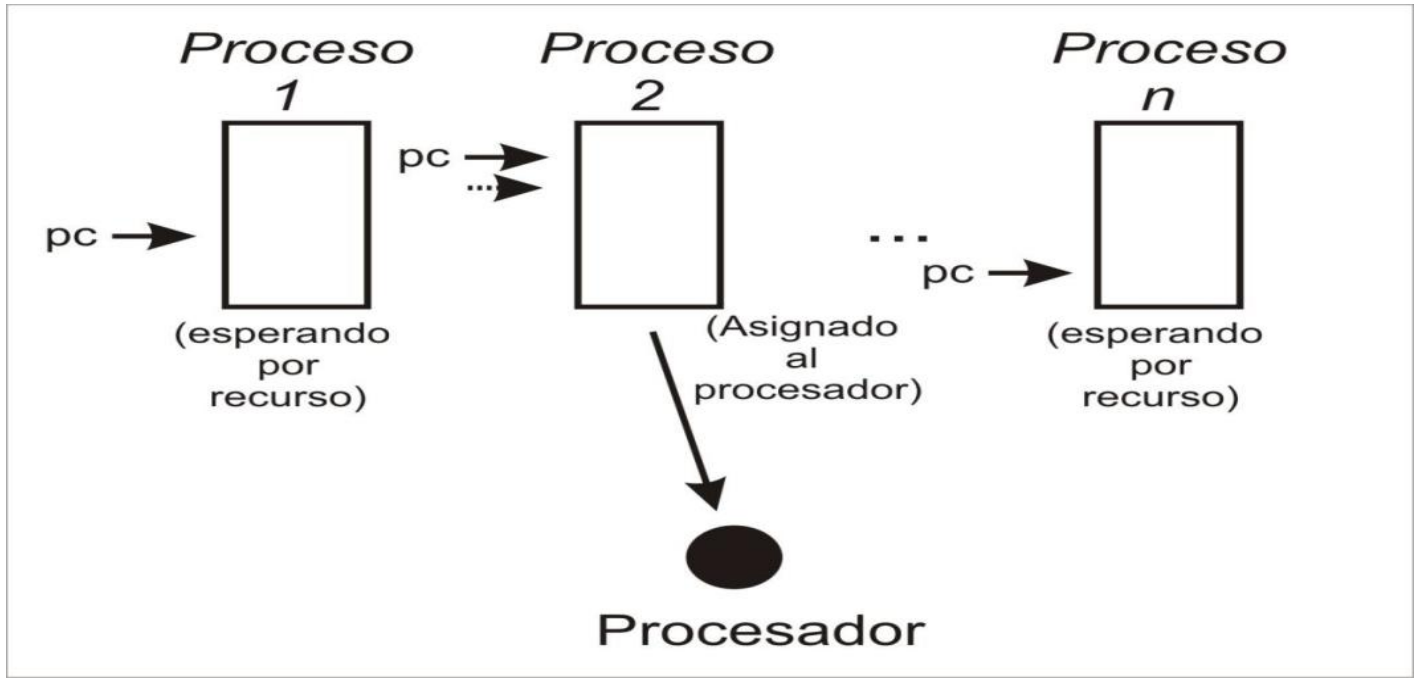
PROCESOS

DEFINICIÓN DE PROCESOS

- Instancia de un programa en ejecución
- Desde el punto de vista del Sistema Operativo es la entidad más pequeña, individualmente planificable, formada por código (instrucciones de máquina y llamadas al sistema) y datos; caracterizada por atributos (asignados por el programador del sistema o por el S.O. tales como prioridad, derechos de acceso, entre otros) y un estado dinámico (inactivo, listo, en ejecución, espera o completado)

DEFINICIÓN DE PROCESOS

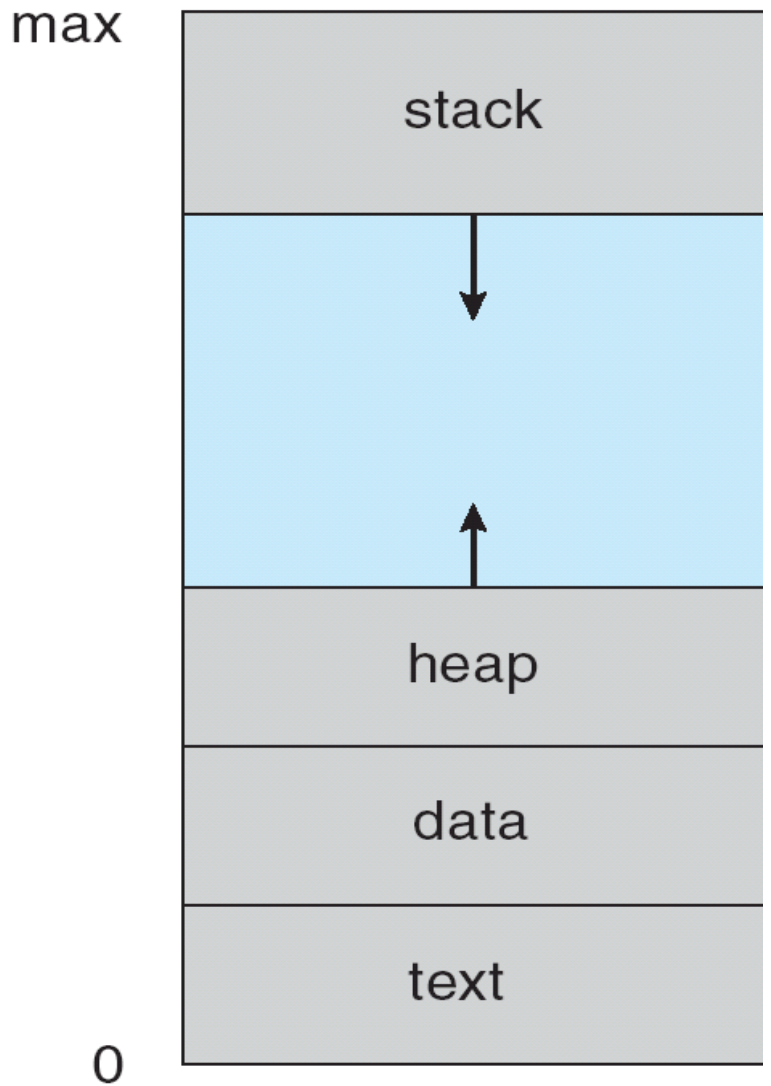
Cada proceso tiene su *program counter*, y avanza cuando el proceso tiene asignado el recurso procesador. A su vez, a cada proceso se le asigna un número que lo identifica entre los demás: identificador de proceso (*process id*).



DEFINICIÓN DE PROCESOS

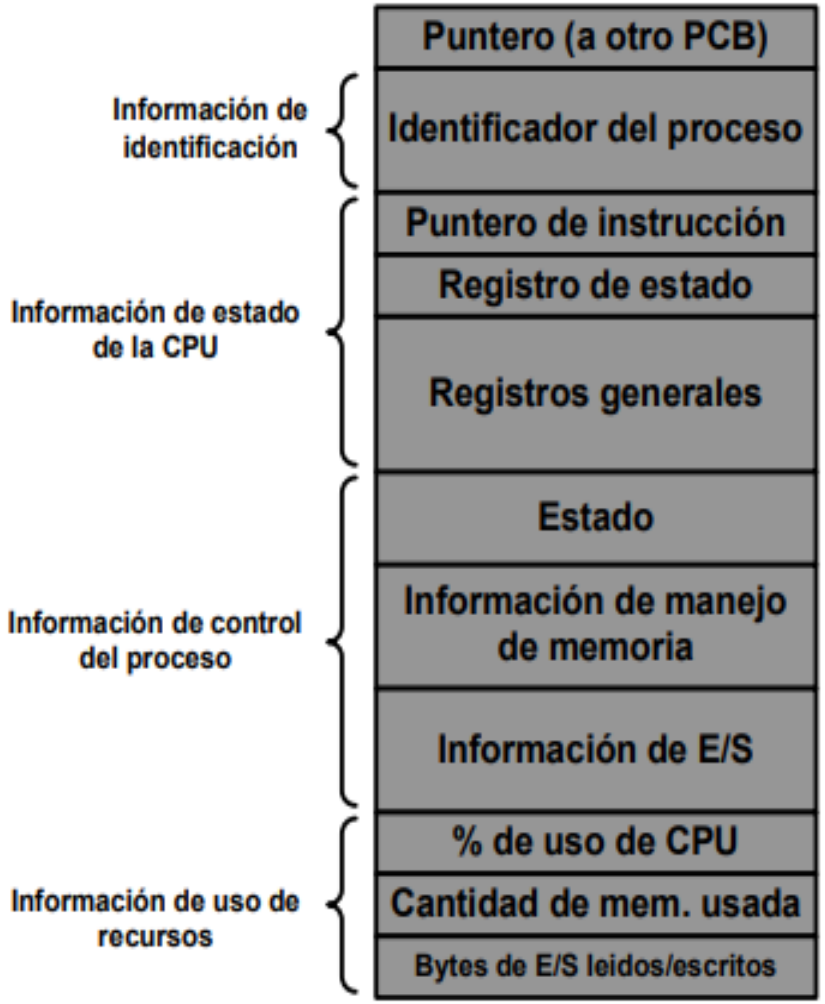
Un proceso en memoria se constituye de varias secciones:

- **Código(text):** Instrucciones del proceso
- **Datos(data):** Variables globales del proceso
- **Memoria dinámica (Heap):** Memoria dinámica que genera el proceso
- **Pila (Stack):** Utilizado para preservar el estado en la invocación anidada de procedimientos y funciones



BLOQUE DE CONTROL DE PROCESOS PCB

Es una estructura de datos en la que el S.O. agrupa toda la información que necesita conocer respecto a un proceso en particular. Cada vez que se crea un proceso el S.O. crea una PCB y cuando el proceso termina su PCB es liberada



CREACIÓN Y TEMINACIÓN DE PROCESOS

Creación de Procesos

- Envío de un trabajo por lotes (batch)
- Conexión de usuario (log-in)
- Creados para proporcionar servicios como imprimir
- Procesos que crean otros procesos

Terminación de Procesos

- Un trabajo por lotes ejecuta la instrucción *Halt*
- Un usuario se desconecta (log-out)
- Salir de una aplicación
- Errores y condiciones de fallo

TERMINACIÓN DE PROCESOS

Razones para la terminación de un Proceso:

- Finalización normal
- Se excede el tiempo límite
- No hay memoria disponible
- Violación de límites
- Error de protección
  - Ejemplo: escribir en un fichero de sólo lectura
- Error aritmético
- Tiempo de espera sobrepasado
  - Un proceso espera un evento más tiempo del máximo especificado

TERMINACIÓN DE PROCESOS

Razones para la terminación de un Proceso:

- Fallo de entrada/salida (E/S)
- Instrucción inválida
  - Cuando se intenta ejecutar datos
- Instrucciones privilegiadas
- Uso incorrecto de datos
- Intervención del Sistema Operativo
  - Cuando se detecta un interbloqueo (deadlock)
- El padre termina, así que los hijos mueren
- Por petición del proceso padre

SUSPENSIÓN DE PROCESOS

Razones para la Suspensión de un Proceso:

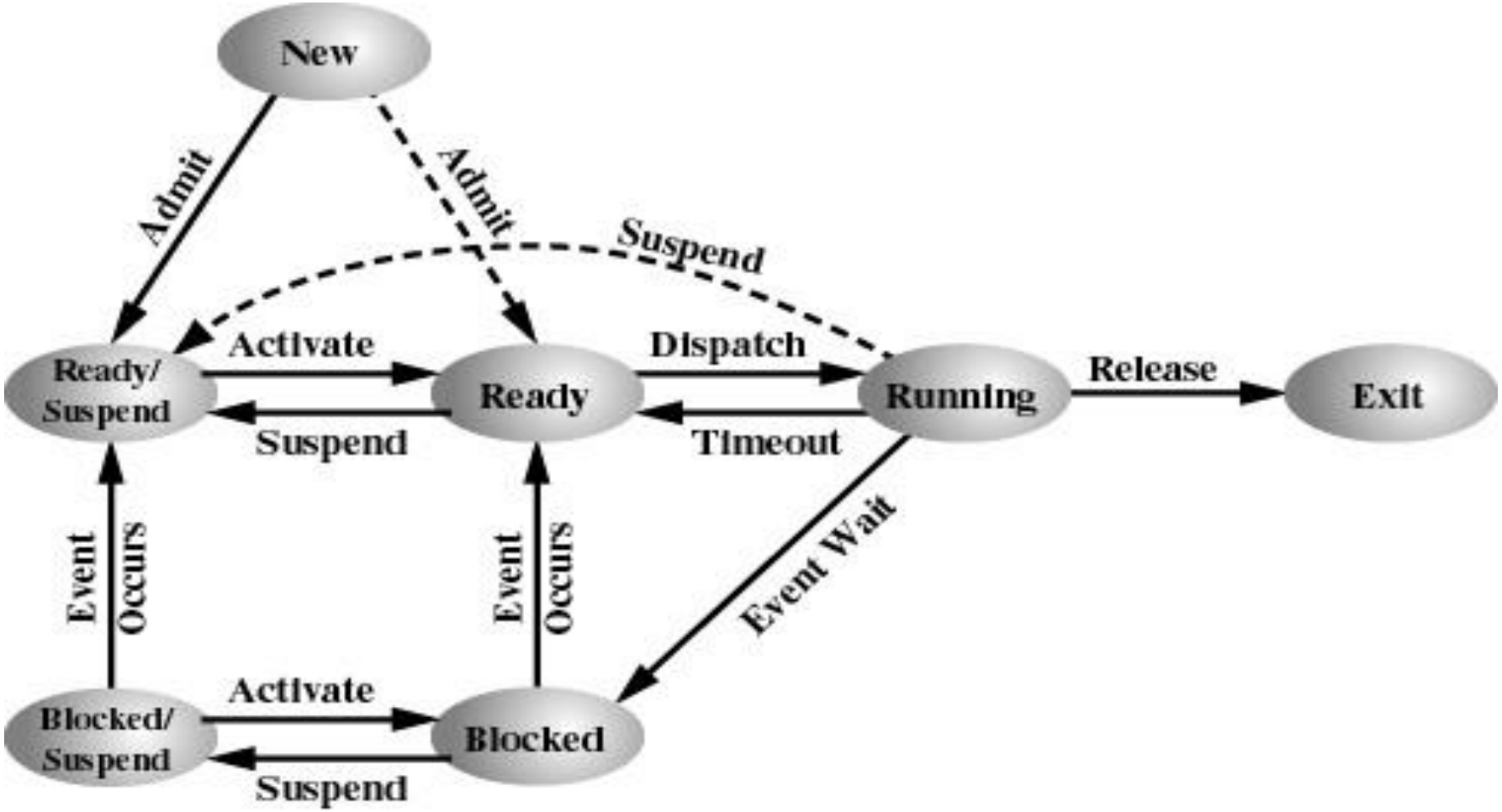
- ***Swapping***
  - El S.O. necesita liberar suficiente memoria principal para ubicar un proceso que está listo para ejecutar
- **Otra razón del sistema operativo**
  - El S.O. puede suspender un proceso del que sospecha que puede estar causando problemas
- **Petición interactiva del usuario**
  - Un usuario puede querer suspender un proceso por motivos de depuración, o relacionado con el uso de un recurso

SUSPENSIÓN DE PROCESOS

Razones para la Suspensión de un Proceso:

- **Temporización**
  - Un proceso puede ejecutarse periódicamente (monitorización o contabilidad) y suspenderse hasta el siguiente intervalo de tiempo
- **Petición del proceso padre**
  - Un proceso puede suspender al descendiente para examinarlo, modificarlo o coordinar la actividad de varios

DIAGRAMA DE ESTADO DE PROCESOS



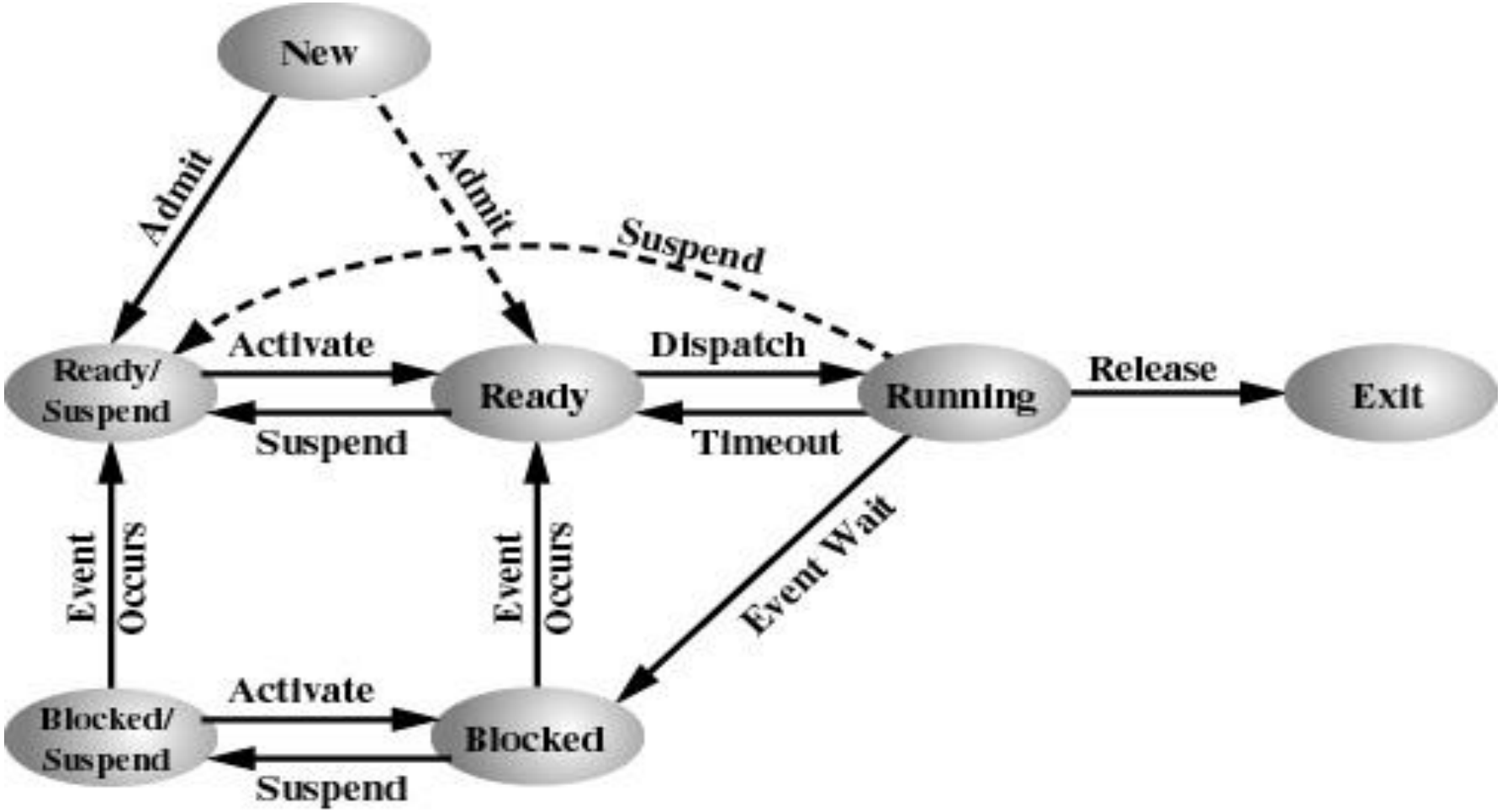
(b) With Two Suspend States

ESTADOS DE LOS PROCESOS

El estado de un proceso es definido por la actividad corriente en que se encuentra Los estados de un proceso son:

- ***Nuevo(new)***: Cuando el proceso es creado
- ***Listo (ready):*** El proceso está listo para ejecutar, solo necesita del recurso procesador
- ***Ejecutando (running):*** El proceso tiene asignado un procesador y está ejecutando sus instrucciones
- ***Bloqueado (waiting):*** El proceso está esperando por un evento (que se complete un pedido de E/S o una señal)
- ***Listo suspendido***. Está en memoria secundaria esperando un evento
- ***Bloqueado suspendido.*** Está en memoria secundaria disponible para ejecutarse
- ***Finalizado (terminated):*** El proceso finalizó su ejecución

DIAGRAMA DE ESTADO DE PROCESOS



(b) With Two Suspend States

TRANSICIÓN DE ESTADOS

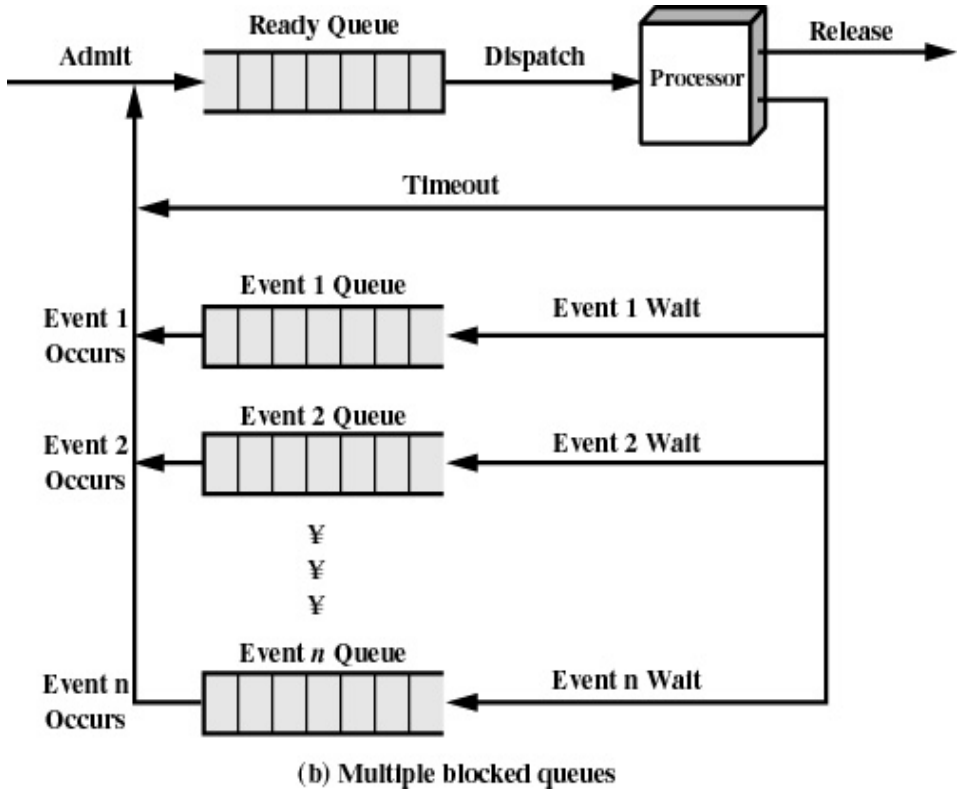
- *Nuevo => Listo suspendido/Listo.* Creamos el proceso cuando hay muchos bloqueados, o inmediatamente
- *Ejecutando => Listo/Suspendido*. Un proceso de mayor prioridad despierta de bloqueado suspendido
- *Listo suspendido => Listo*. Si no hay procesos listos, necesitamos traer uno para ejecutarlo. También puede que un proceso listo suspendido tenga la mayor prioridad.
- *Listo => Listo suspendido*. Normalmente se suspenden procesos bloqueados, pero se puede preferir suspender un proceso listo de baja prioridad ante uno bloqueado de alta prioridad. Se busca obtener más memoria libre

TRANSICIÓN DE ESTADOS

- *Bloqueado =>Bloqueado suspendido*. Si no hay procesos listos, un proceso bloqueado se pasa a disco para hacer espacio en memoria o para liberar memoria.
- *Bloqueado suspendido => Bloqueado*. Un proceso termina liberando memoria, uno de los bloqueados suspendidos tiene la mayor prioridad y el SO sospecha que el evento que espera sucederá en breve
- *Bloqueado suspendido => Listo suspendido*. Sucede lo que estaba esperando.
- *Cualquier estado => Saliente*

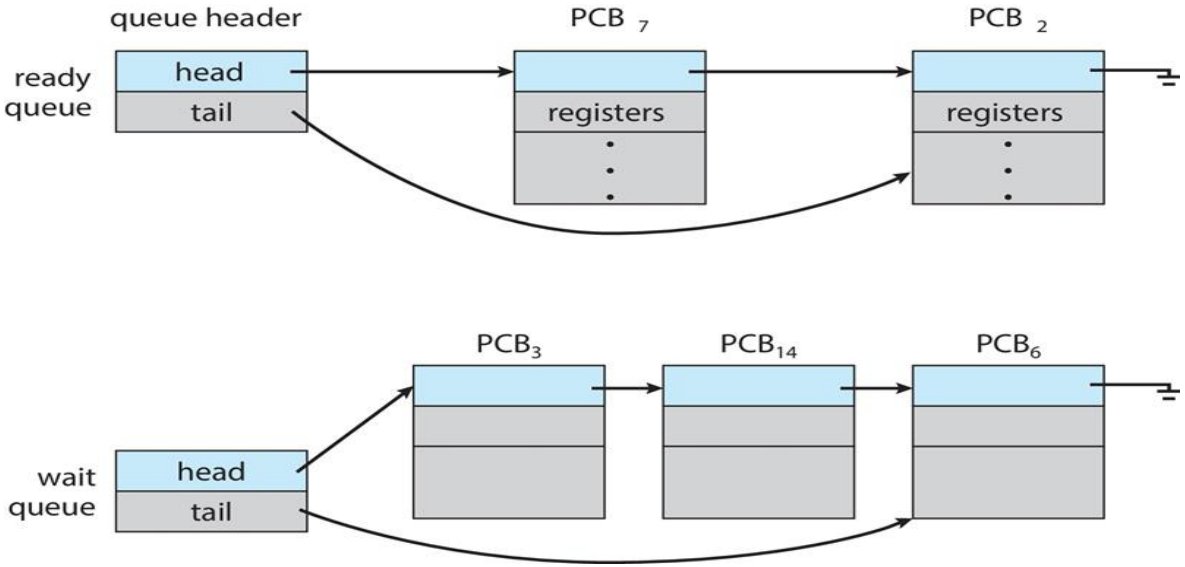
LISTAS Y COLAS DE PROCESOS

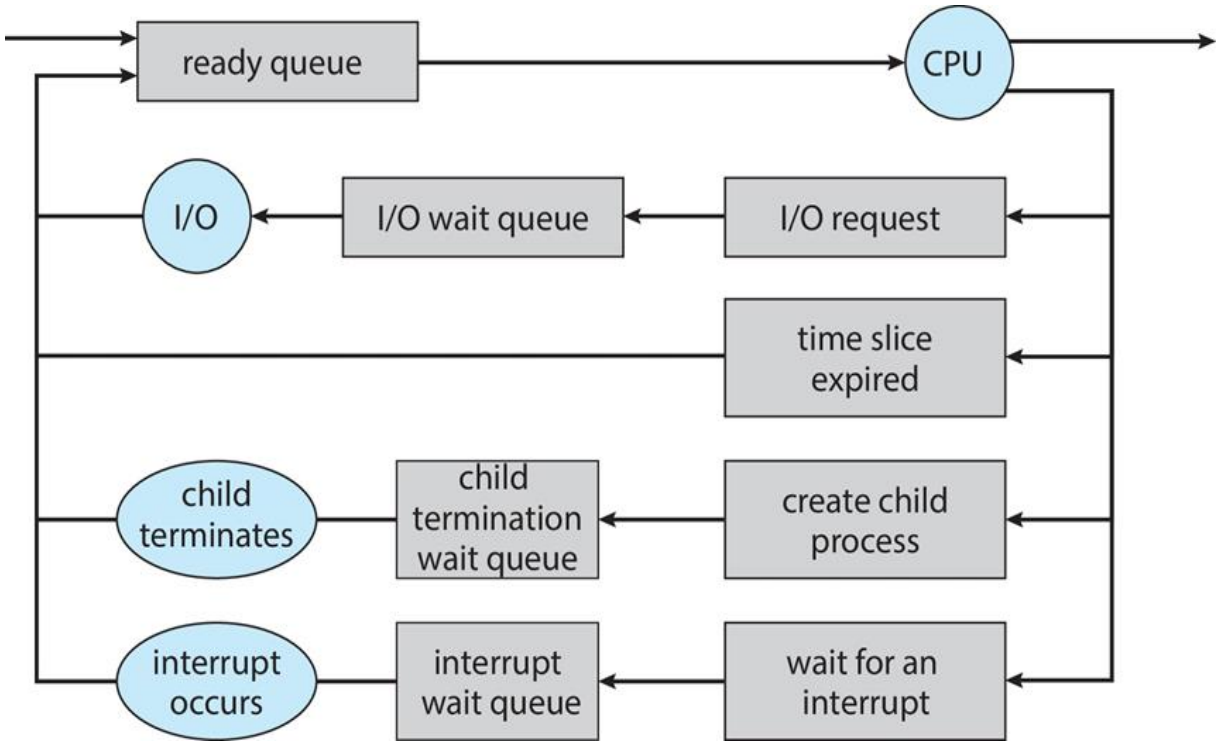
- ***Lista de procesos del sistema (job queue):*** En esta lista están todos los procesos del sistema. Al crearse un nuevo proceso se agrega el PCB a esta lista. Cuando el proceso termina su ejecución es borrado.
- ***Cola de procesos listos (ready queue):*** Esta cola se compondrá de los procesos que estén en estado listo. La estructura de esta cola dependerá de la estrategia de planificación utilizada.
- ***Cola de espera de dispositivos (device queue):*** Los procesos que esperan por un dispositivo de E/S particular, son agrupados en una lista específica al dispositivo. Cada dispositivo de E/S tendrá su cola de espera.



(b) Multiple blocked queues

LISTAS Y COLAS DE PROCESOS





CONTROL DE PROCESOS

Modos de Ejecución de Procesos

- **Modo usuario**
  - Modo con menos privilegios
  - Los programas de usuario normalmente se ejecutan en este modo
- **Modo del sistema, modo de control o modo *kernel***
  - Modo con más privilegios
  - *Kernel* del sistema operativo

El proceso está compuesto por la parte usuario (lo que el usuario implementa) y la parte de núcleo (las rutinas del núcleo que utiliza)

CONTROL DE PROCESOS

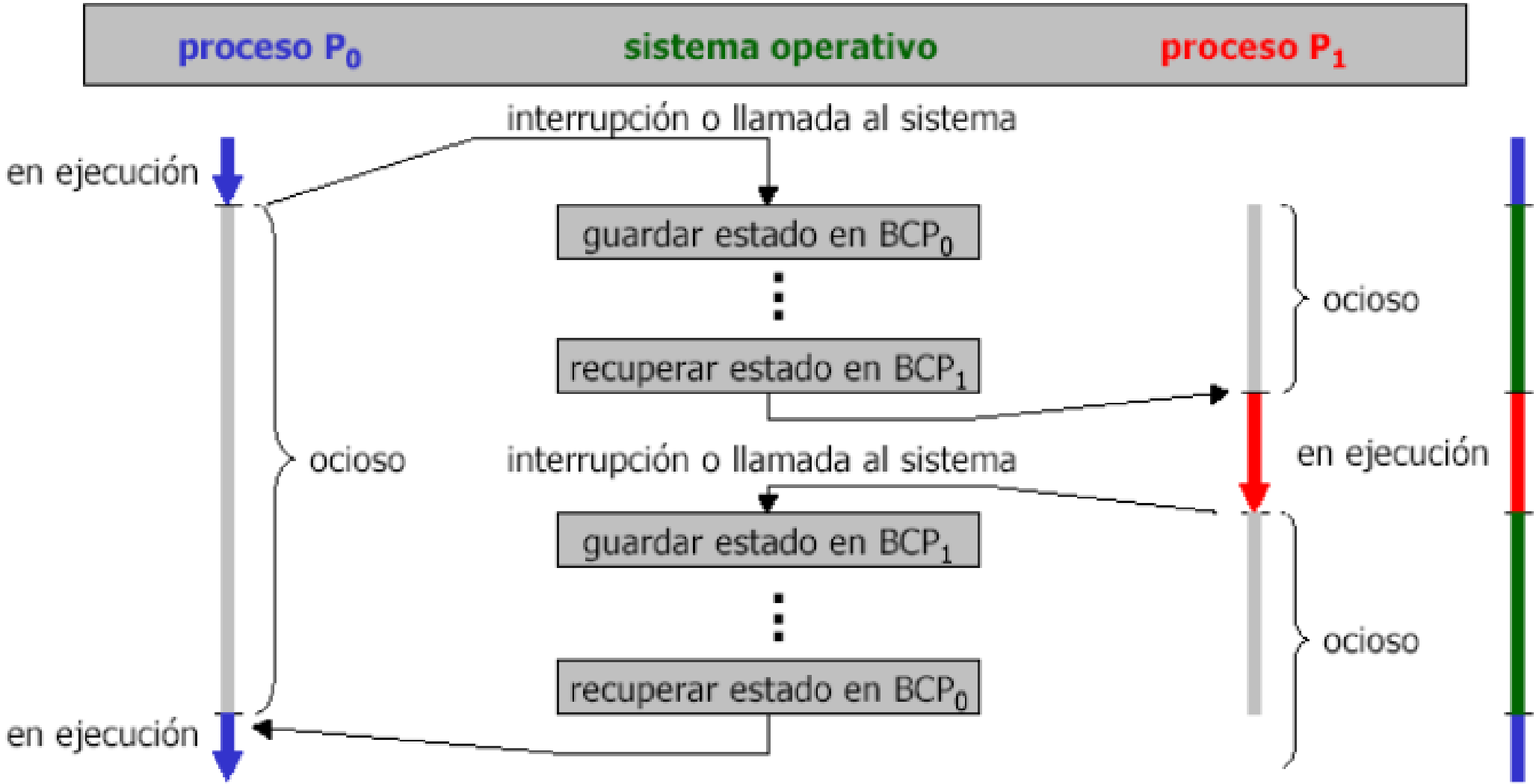
Mecanismo de cambio de modo

1. El PC pasa a apuntar a la rutina de tratamiento de la excepción
2. Cambia de modo usuario a modo núcleo
3. El S.O. guarda el contexto del proceso interrumpido en su PCB

Mecanismo de cambio de proceso

1. Salvar el estado del procesador (PC, registros...)
2. Actualizar campos PCB (estado, contabilidad, auditoría...)
3. Mover el PCB a la cola apropiada (Listo, bloqueado..)
4. Seleccionar el nuevo proceso a ejecutar
5. Actualizar el PCB del proceso elegido (estado Ejecutando...)
6. Actualizar estructuras de datos de gestión de memoria
7. Restaurar el estado del procesador cuando se interrumpió el nuevo proceso

CONTROL DE PROCESOS



CUÁNDO CONMUTAR PROCESOS

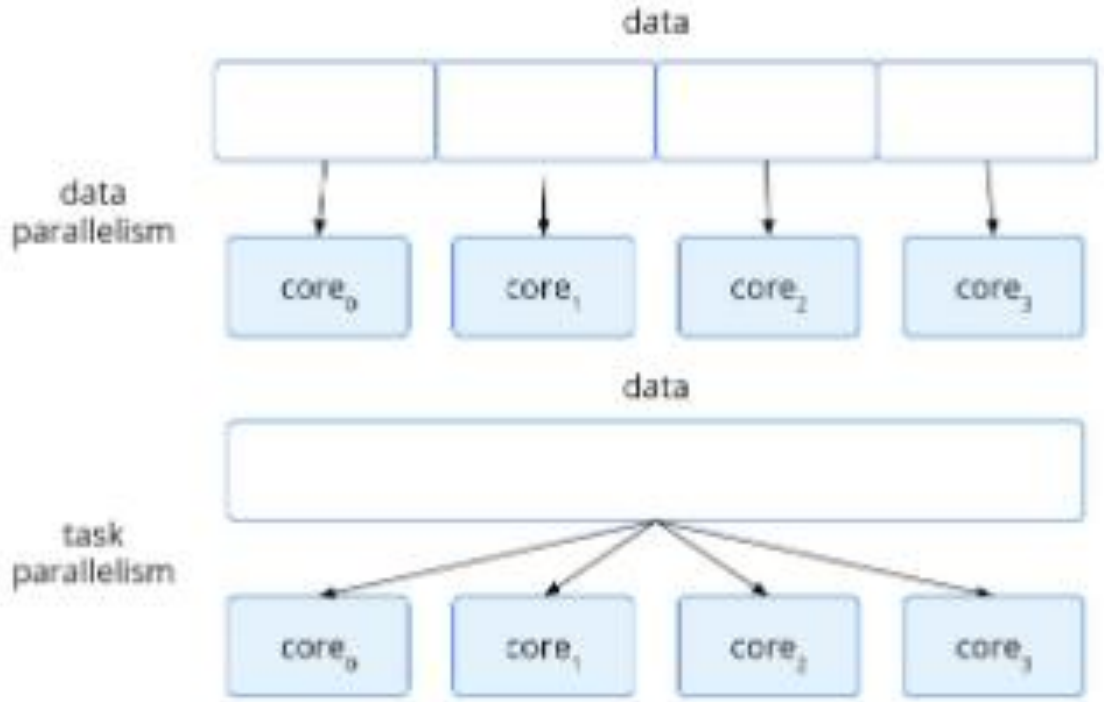
- **Interrupción del reloj (timer)**
  - El proceso se ha ejecutado durante el máximo periodo de tiempo permitido (time slice)
- **Interrupción E/S**
- **Fallo de memoria**
  - La dirección de memoria está en la memoria virtual y hay que traerla a la memoria principal
- **Trap (excepción)**
  - Ocurre un error puede causar que un proceso se mueva al estado de Exit
- **Llamada al Supervisor (al sistema)**
  - Tal como abrir un fichero

HILOS O THREADS

HILOS O THREADS

En general hay dos tipos de paralelismo

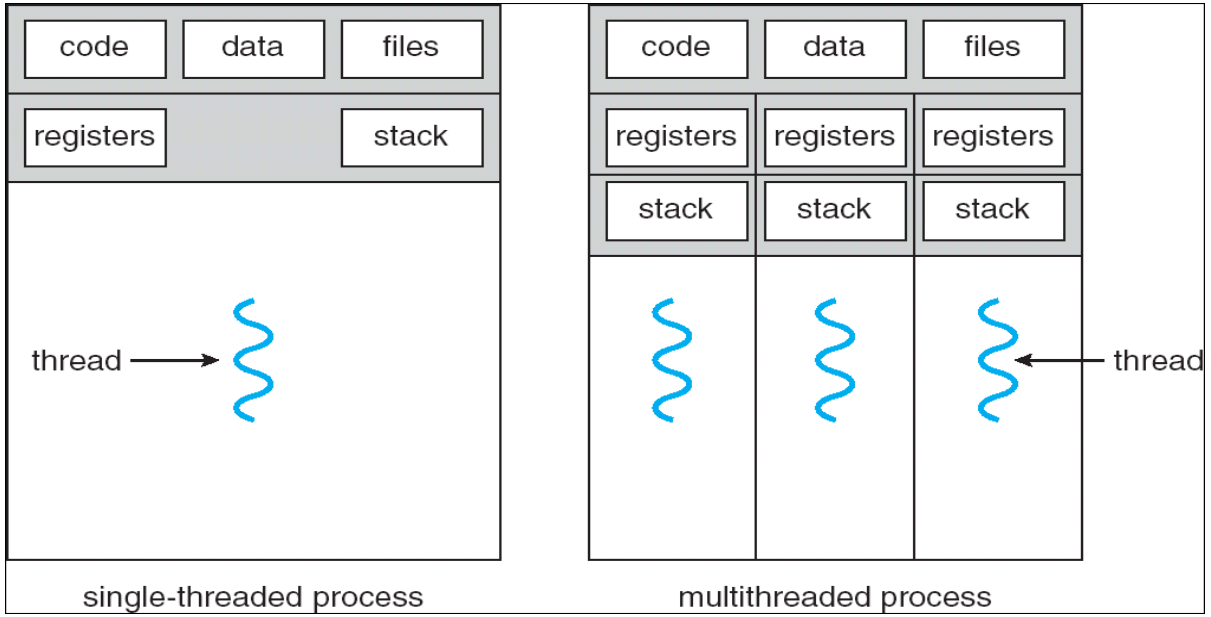
- **Paralelismo de datos:** Se centra en distribuir subconjuntos de los mismos en múltiples cores y realizar la misma operación en cada core.
- **Paralelismo de tareas:** Distribuye tareas (hilos) a través de múltiples cores. Cada hilo realiza una operación única. Diferentes hilos pueden estar operando con los mismos datos o con diferentes datos.



HILOS O THREADS

Un hilo (proceso ligero) es una unidad básica de utilización de la CPU, y consiste en un contador de programa, un juego de registros y un espacio de pila. Los hilos dentro de una misma aplicación comparten:

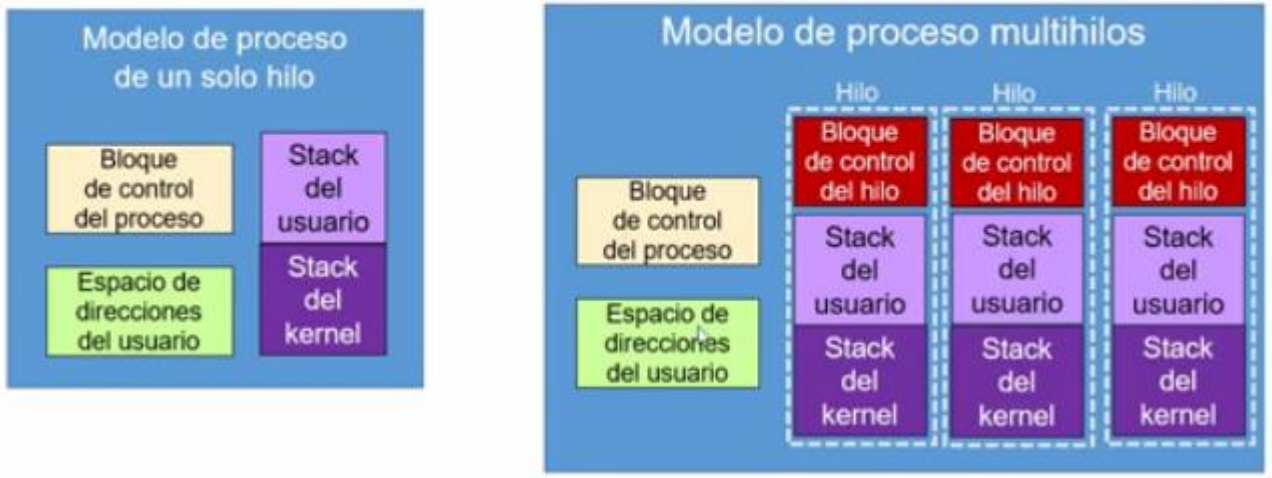
- La sección de código
- La sección de datos
- Los recursos del SO (archivos abiertos y señales)



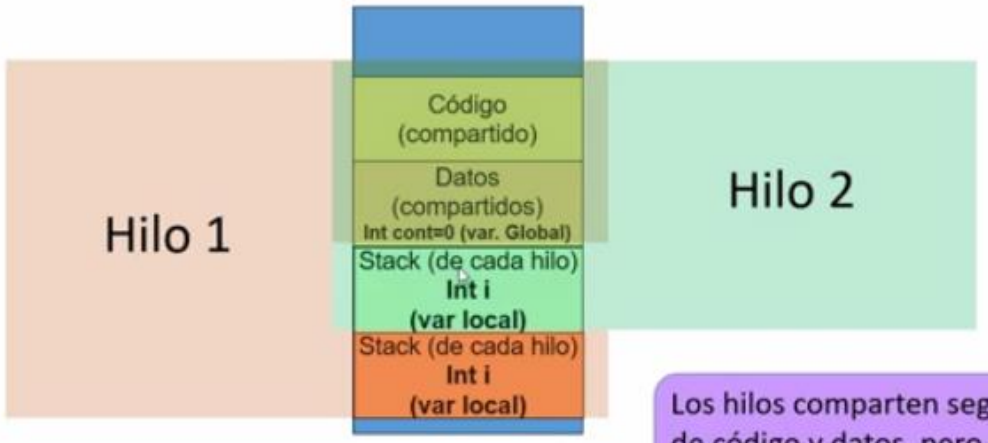
Elementos por hilo	**Elementos por proceso**
Contador de programa	Espacio de direcciones
Pila	Variables globales
Estado + contexto	Ficheros abiertos
Memoria privada	Procesos hijos
	Cronómetros
	Señales
	Semáforos
	Información contable

HILOS O THREADS

Modelos de procesos de un solo hilo y de muchos hilos



Hilos en memoria



Los hilos comparten segmento de código y datos, pero cada hilo debe tener su stack

HILOS O THREADS

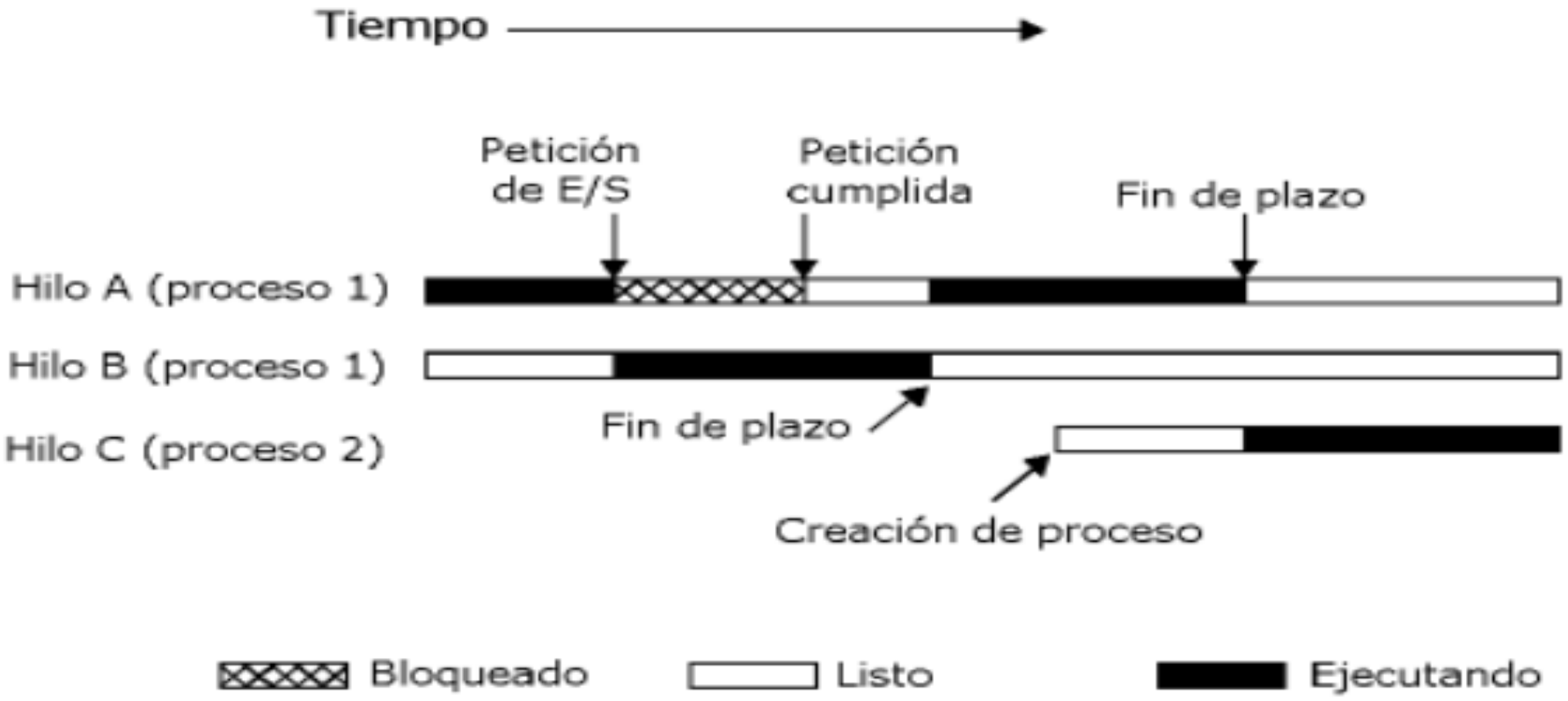
Los principales estados de un hilo son: ejecución, preparado y bloqueado y hay cuatro operaciones básicas relacionadas con el cambio de estado de los hilos:

- **Creación:** En general, cuando se crea un nuevo proceso se crea también un hilo para ese proceso. Posteriormente, ese hilo puede crear nuevos hilos dándoles un puntero de instrucción y algunos argumentos. Ese hilo se colocará en la cola de preparados.
- **Bloqueo:** Cuando un hilo debe esperar por un suceso, se le bloquea guardando sus registros. Así el procesador pasará a ejecutar otro hilo preparado.
- **Desbloqueo:** Cuando se produce el suceso por el que un hilo se bloqueó pasa a la cola de listos.
- **Terminación:** Cuando un hilo finaliza, se liberan su contexto y sus pilas.

Nota: Un punto importante es la posibilidad de que el bloqueo de un hilo lleve al bloqueo de todo el proceso.

HILOS O THREADS

En un sistema monoprocesador, la multiprogramación permite intercalar la ejecución de múltiples hilos dentro del mismo proceso:



VENTAJAS DEL USO DE THREADS

Respuesta: Desarrollar una aplicación con varios hilos de control (*threads)* permite tener un mejor tiempo de respuesta

Compartir recursos: Los *threads* de un proceso comparten la memoria y los recursos que utilizan. A diferencia de *IPC,* no es necesario acceder al *kernel* para comunicar o sincronizar los hilos de ejecución

Economía: Es más fácil un cambio de contexto entre *threads* ya que no es necesario cambiar el espacio de direccionamiento. A su vez, es más "liviano" para el sistema operativo crear un *thread* que crear un proceso nuevo

Utilización de arquitecturas con multiprocesadores: Disponer de una arquitectura con más de un procesador permite que los *threads* de un mismo proceso ejecuten en forma paralela

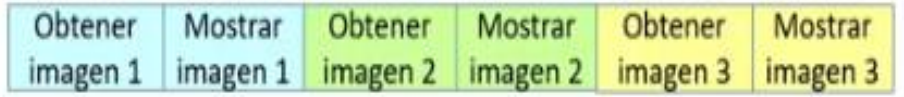
VENTAJAS DEL USO DE THREADS

Un procesador de texto puede tener un hilo para mostrar gráficos, otro hilo para responder al tecleo del usuario y un tercer hilo para realizar la revisión ortográfica y gramatical en segundo plano.

Multihilo

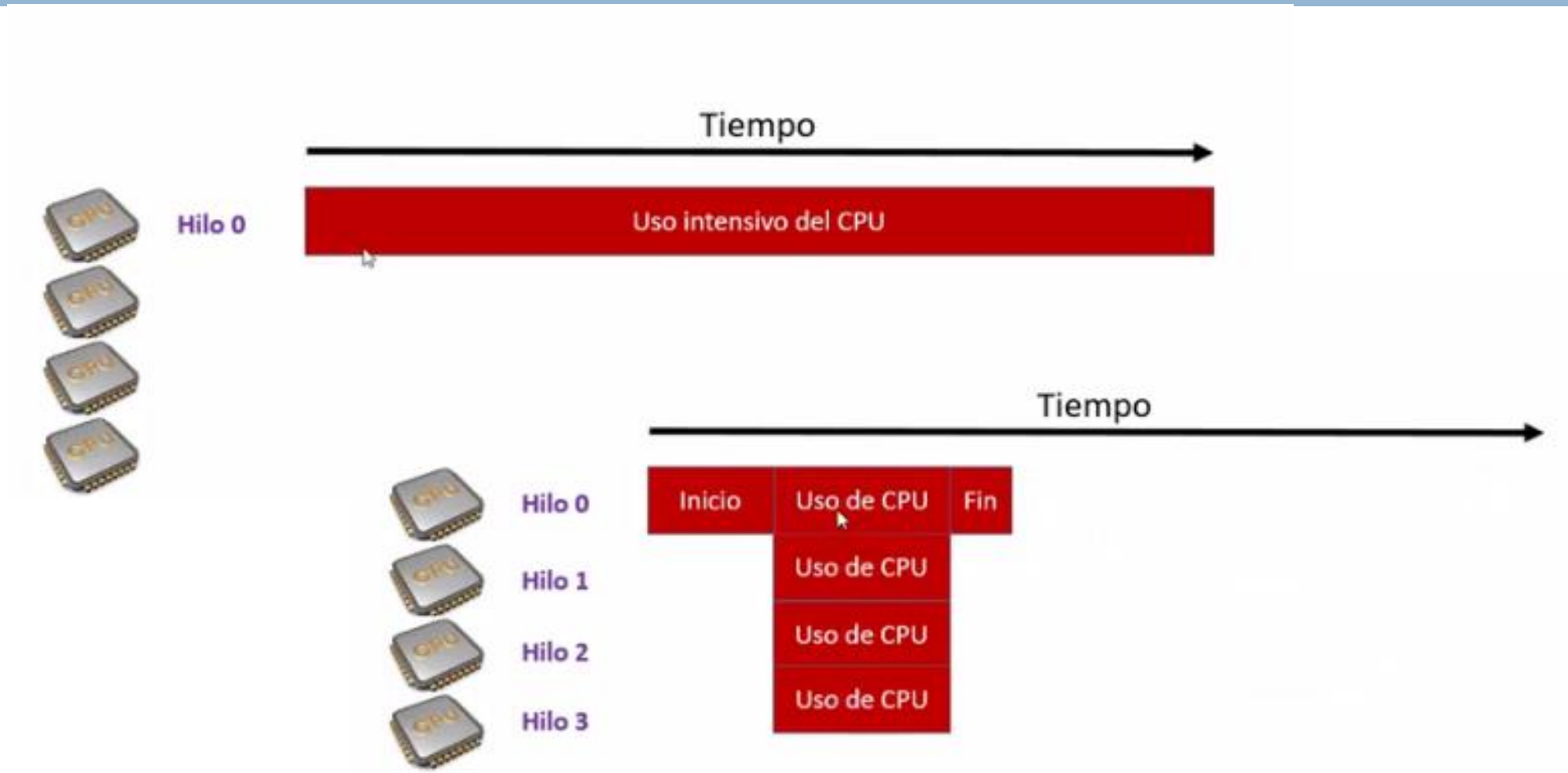
Un navegador web puede tener un hilo para leer las imágenes de la red, mientras que otro hilo las está mostrando en pantalla.

Monohilo



Multihilo

VENTAJAS DEL USO DE THREADS



IMPLEMENTACIÓN DE HILOS O THREADS

Hilos a nivel de usuario:

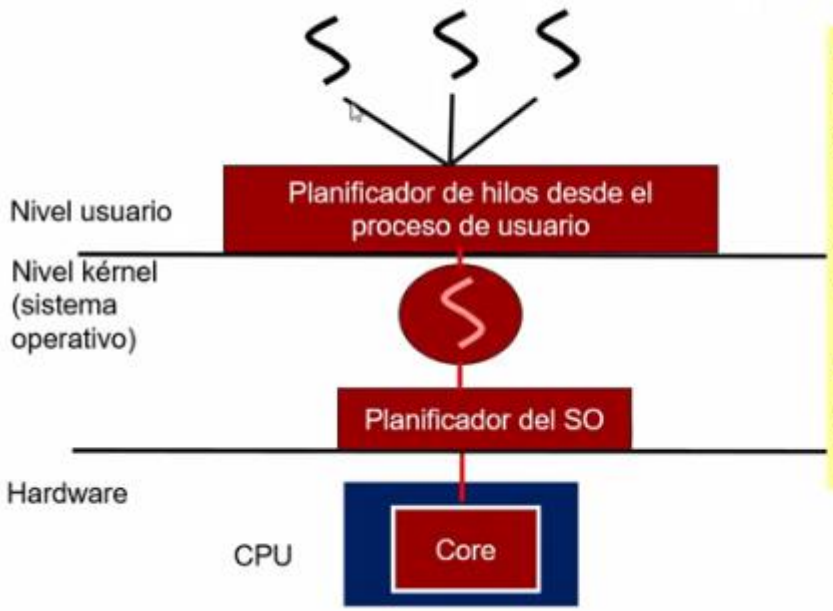
- La gestión de hilos la realiza una aplicación y el núcleo no es consciente de la existencia de hilos. Se programa una aplicación como multihilo mediante una biblioteca de funciones para gestionar ULT (crear, destruir, comunicar, planificar, etc.)

Hilos a nivel de núcleo (KLT):

- Todo el trabajo de gestión de hilos lo realiza el núcleo. En el área de la aplicación no hay código para la gestión de hilos, únicamente una Interfaz de Programas de Aplicación (API) para las funciones de gestión de hilos en el núcleo.
- Este tipo de soluciones se han implementado en Windows 2000 y Linux. Las aplicaciones se programan como multihilo y todos los hilos de esa aplicación pertenecen al mismo proceso. El núcleo del SO mantiene la información de contexto del proceso como un todo y la de cada hilo dentro del proceso.
- La principal desventaja de la solución KLT, respecto de la ULT, es que el paso de control de un hilo a otro, dentro del mismo proceso, requiere cambios de modo.

IMPLEMENTACIÓN DE HILOS O THREADS

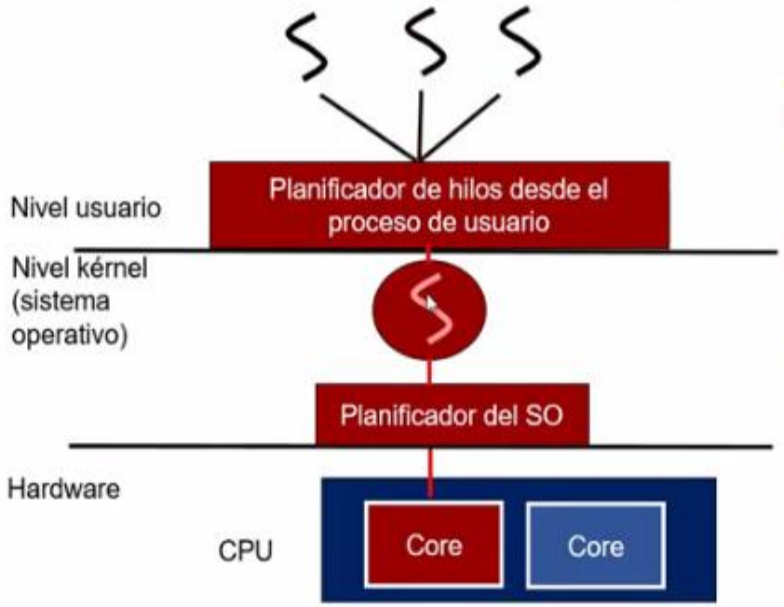
Hilos a nivel usuario



✓Los hilos podían implementarse en las aplicaciones que se ejecutaban en sistemas operativos que no son capaces de planificar hilos

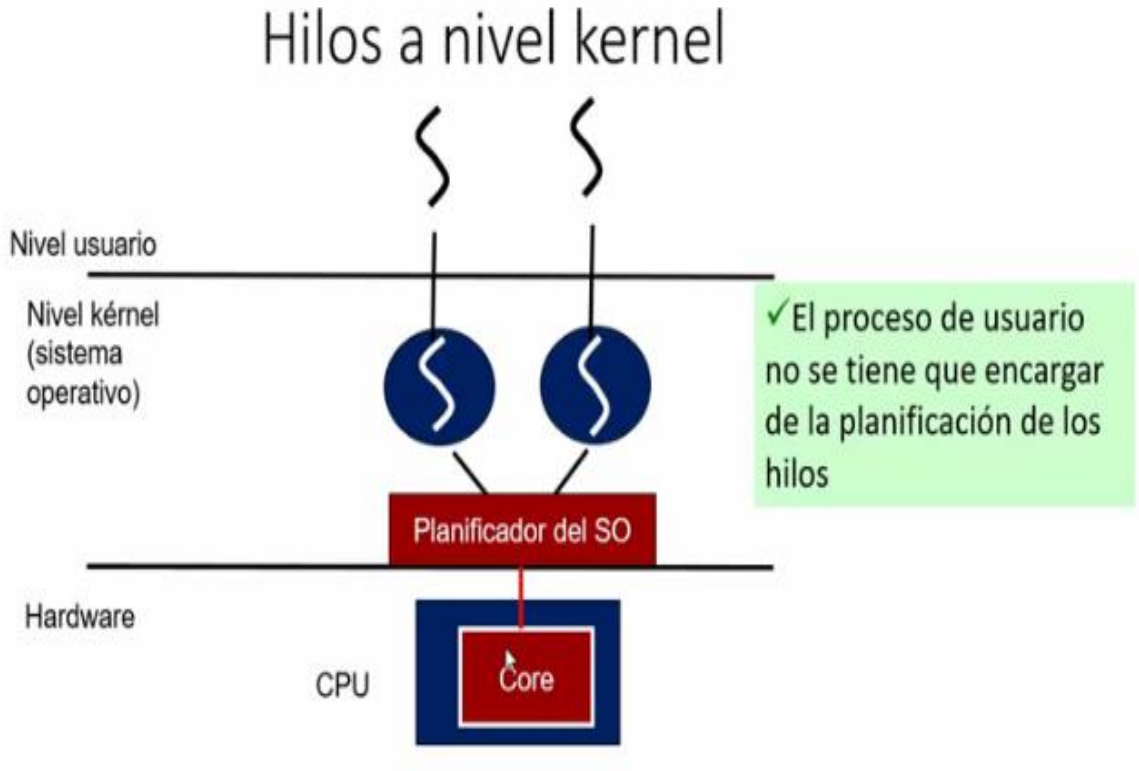
Ejemplo: Primeras implementaciones de UNIX

Hilos a nivel usuario (un CPU con dos núcleos)

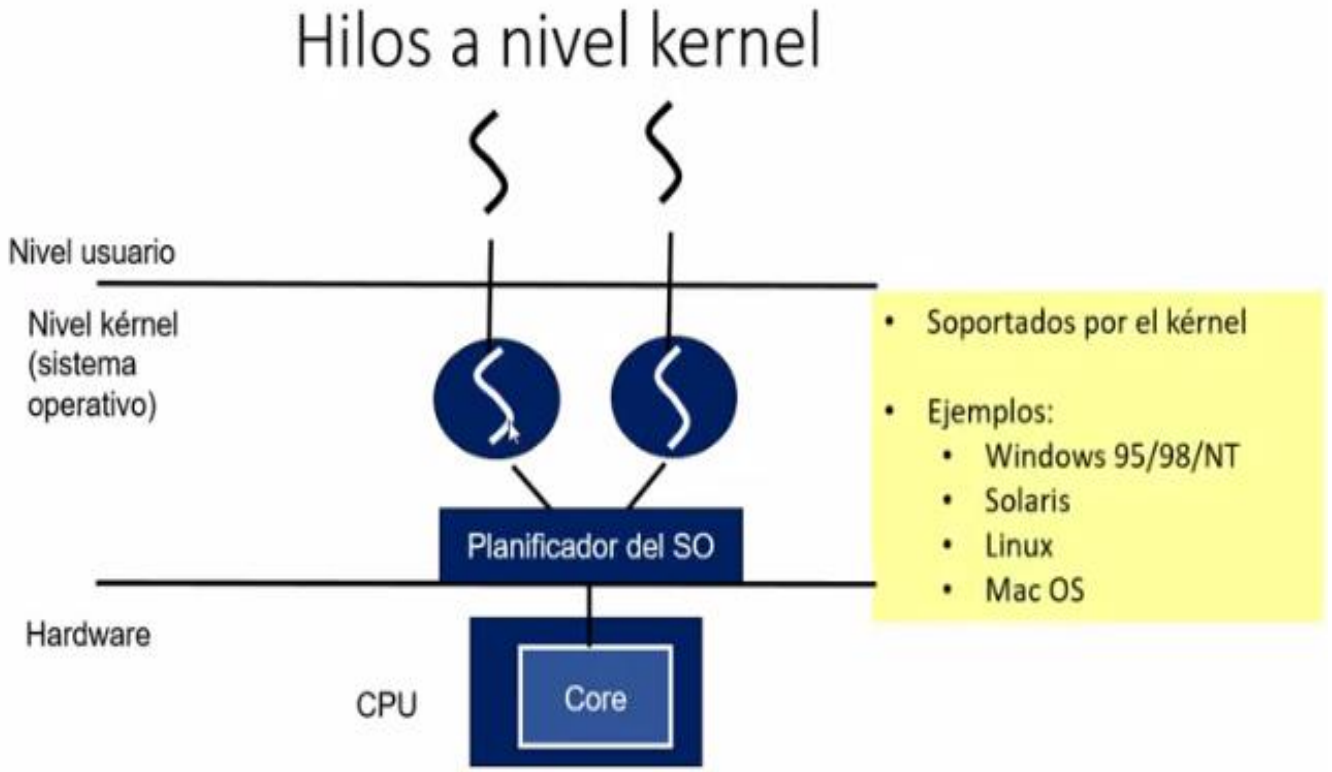


×El planificador del sistema operativo solo ve un hilo por proceso
- ×No ve los hilos a nivel usuario
- ×Usa un solo núcleo del procesador

IMPLEMENTACIÓN DE HILOS O THREADS

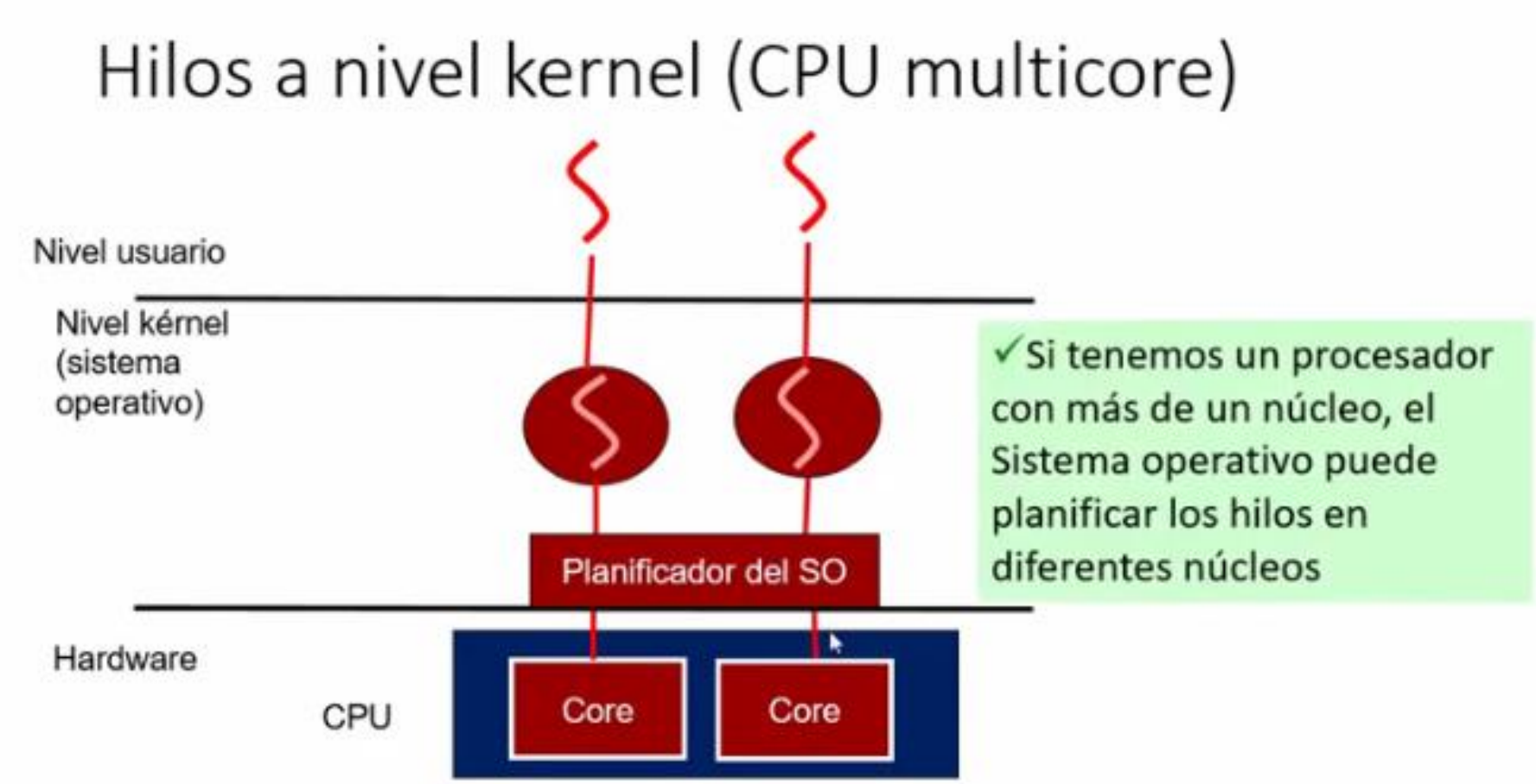


✓El proceso de usuario no se tiene que encargar de la planificación de los hilos

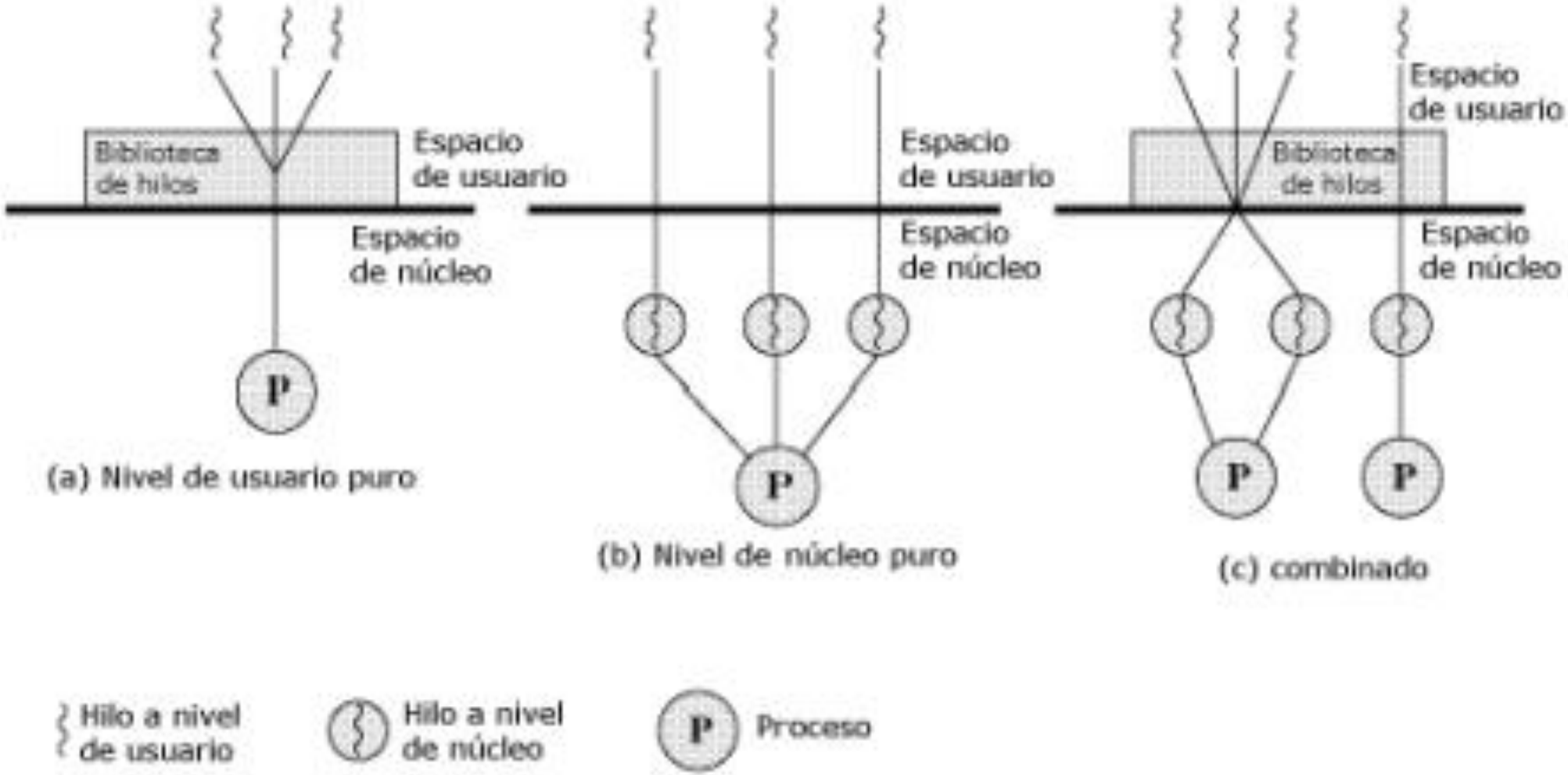


- Soportados por el kérnel
- Ejemplos:
  - Windows 95/98/NT
  - Solaris
  - Linux
  - Mac OS

IMPLEMENTACIÓN DE HILOS O THREADS



IMPLEMENTACIÓN DE HILOS O THREADS



IMPLEMENTACIÓN DE HILOS O THREADS

Consideraciones de concurrencia:

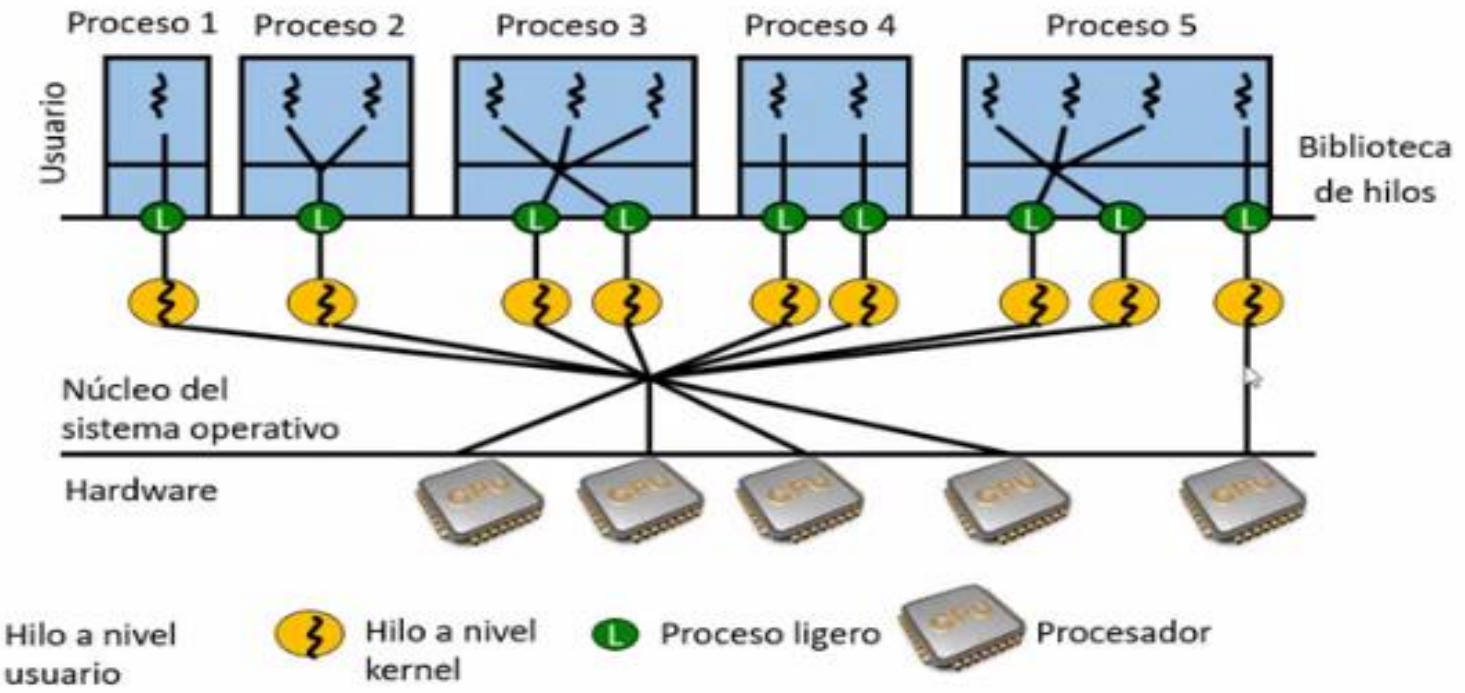
1. Mientras que el modelo muchos a uno le permite al desarrollador crear tantos hilos de usuario como desee, no produce paralelismo, porque el kernel puede planificar solo un hilo del kernel a la vez.
2. El modelo uno a uno permite una mayor concurrencia, pero el desarrollador debe tener cuidado de no crear demasiados hilos dentro de una aplicación (de hecho, en algunos sistemas, puede estar limitada la cantidad de hilos que se pueden crear).
3. El modelo de muchos a muchos no tiene ninguno de estos inconvenientes:
   - Los desarrolladores pueden crear tantos hilos de usuario como sea necesario y los correspondientes hilos del kernel pueden ejecutarse en paralelo en un multiprocesador.
   - Cuando un hilo realiza una llamada al sistema bloqueante, el kernel puede planificar otro hilo para su ejecución

IMPLEMENTACIÓN DE HILOS O THREADS

Ejemplo de la arquitectura multihilo de Solaris

Los procesos tienen un hilo principal más hilos a nivel usuario

Proceso ligero es un hilo de ejecución en un proceso. Un proceso ligero corresponde a un hilo a nivel kernel



IMPLEMENTACIÓN DE HILOS O THREADS

Ventajas de usar ULT en vez de KLT:

1. El intercambio de hilos no requiere los privilegios del modo núcleo. Así, el proceso no debe cambiar al modo núcleo y volver al modo usuario para gestionar los hilos. Esto evita una sobrecarga de procesamiento.
2. Se puede manejar una planificación específica para los hilos. Para algunas aplicaciones puede ser mejor una planificación basada en prioridades mientras que para otras puede ser mejor un turno rotatorio.
3. Los ULT se pueden ejecutar en cualquier SO. Para dar soporte a los ULT no es necesario realizar cambios en el núcleo del SO ya que se utilizan herramientas de gestión de hilos que proporciona una biblioteca para todas las aplicaciones.

IMPLEMENTACIÓN DE HILOS O THREADS

Desventajas del uso de ULT en vez de KLT:

1. En un SO la mayoría de las llamadas a sistema son "bloqueantes". Así, cuando un ULT ejecuta una llamada al sistema no solo se bloquea ese hilo, sino todos los hilos del proceso
2. Una aplicación multihilo basada en ULT no puede sacar partido de los sistemas multiprocesadores. El núcleo del SO asigna un proceso a un solo procesador y sólo se puede ejecutar en él un hilo del proceso a la vez

Para evitar estas desventajas:

1. Podemos modularizar la aplicación por procesos en vez de hilos. Pero esto elimina la principal ventaja de los hilos: cada cambio de proceso es más costoso que un cambio de hilo
2. Usar una técnica de recubrimiento (convierte una llamada bloqueante en una no bloqueante), para evitar el bloqueo de los hilos

IMPLEMENTACIÓN DE HILOS O THREADS

Una biblioteca de hilos proporciona al programador una API para crear y administrar hilos

Hay dos formas principales de implementarla:

1. **Primer enfoque:** proporcionar una biblioteca ubicada en el espacio de usuario sin soporte de kernel.
   - Códigos y estructuras de datos para la biblioteca en espacio de usuario.
   - Invocar una función en la biblioteca da como resultado una llamada de función local en el espacio del usuario y no una llamada de sistema.
2. **Segundo enfoque:** implementar una biblioteca a nivel de kernel soportada directamente por el SO.
   - Códigos y estructuras de datos para la biblioteca en espacio de kernel.
   - Invocar una función en la API para la biblioteca generalmente resulta en una llamada de sistema al kernel.

Hoy en día las principales bibliotecas de hilos son: POSIX Pthreads, Windows Threads, y JavaThreads.