Gestión de threads (hilos)

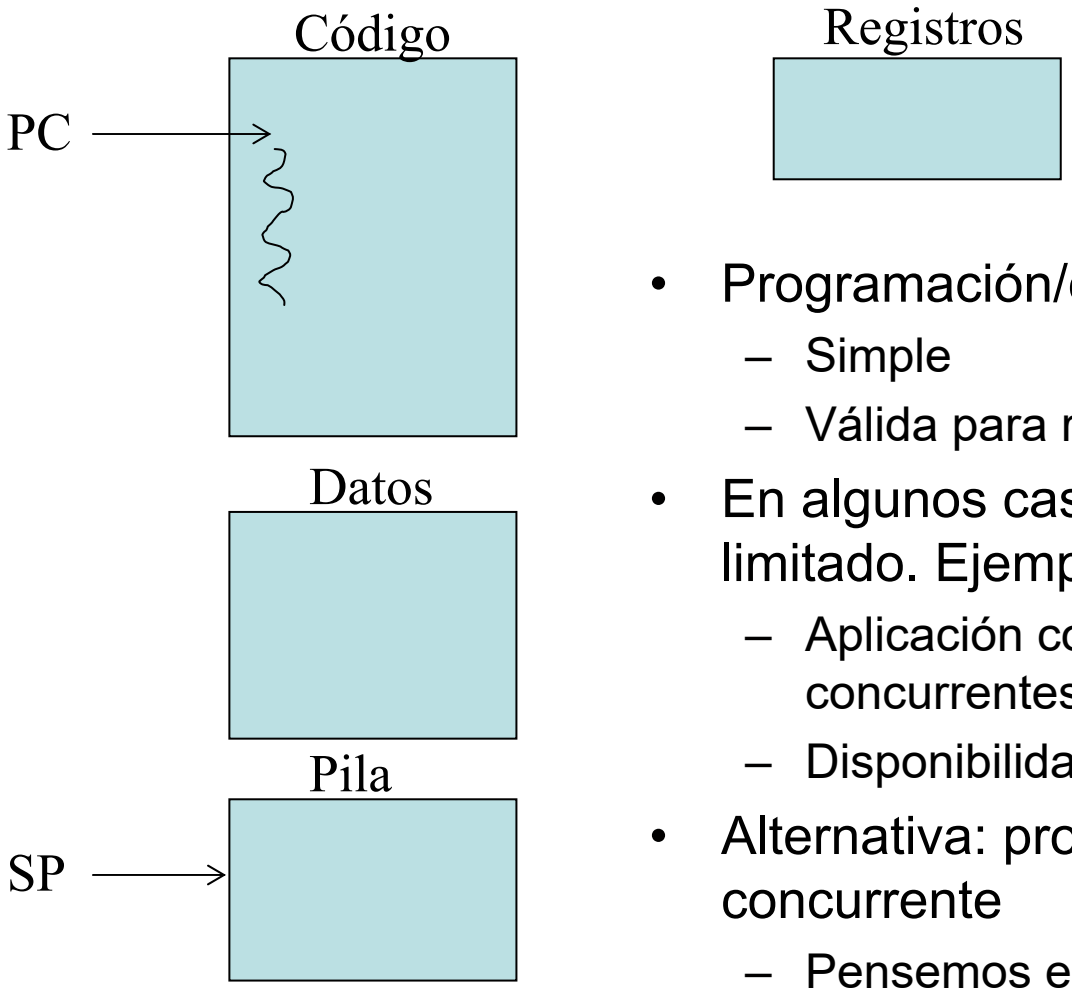
Sistemas Operativos

Pablo Ibáñez

[Sta]: capítulo 4

[SGG]: capítulo 4

Modelo de proceso



- Programación/ejecución secuencial
  - Simple
  - Válida para resolver muchos problemas
- En algunos casos es un modelo muy limitado. Ejemplos
  - Aplicación con varias actividades concurrentes de E/S y/o cálculo
  - Disponibilidad de varios procesadores
- Alternativa: programación/ejecución concurrente
  - Pensemos en varios procesos

Ejemplo 1: servidor de ficheros en red

- Aplicación con E/S concurrentes
  - Recibe peticiones (read, write, …) de procesos de otras máquinas
  - realiza las operaciones sobre disco y responde las peticiones
- Versión secuencial simple: servicio de peticiones en serie
  - Bucle que lee petición, realiza llamada al sistema bloqueante, y responde
  - Muy bajas prestaciones
- Versión secuencial optimizada: código complejo
  - Usa llamadas al sistema no bloqueantes para recibir peticiones y para realizar las operaciones en disco
  - guarda memoria de las operaciones pendientes de respuesta, que posiblemente llegarán en desorden
- Solución concurrente
  - Cada petición crea un proceso que realiza la operación sobre disco y responde

Ejemplo 2: word

- Aplicación con varias actividades concurrentes: entrada de texto, corrector ortográfico, formato en pantalla
- Cada vez que se introduce nuevo texto desde teclado hay que ejecutar el código del corrector ortográfico. Opciones:
  - Se ejecuta corrector y después se continúa: se desatiende la lectura de teclado durante un tiempo
  - Se ejecuta corrector a la vez que se atiende teclado: se complica mucho la programación de la aplicación
- Además, word tiene que realizar otras tareas como la organización del texto en pantalla
  - espaciado entre caracteres, líneas y párrafos
  - formatos de letra, …
- Solución concurrente: cada actividad un proceso
  - Cada actividad se programa de forma independiente
  - Se despierta solo cuando se requiere, distintas prioridades, ...

Ejemplo 3: cálculo masivo

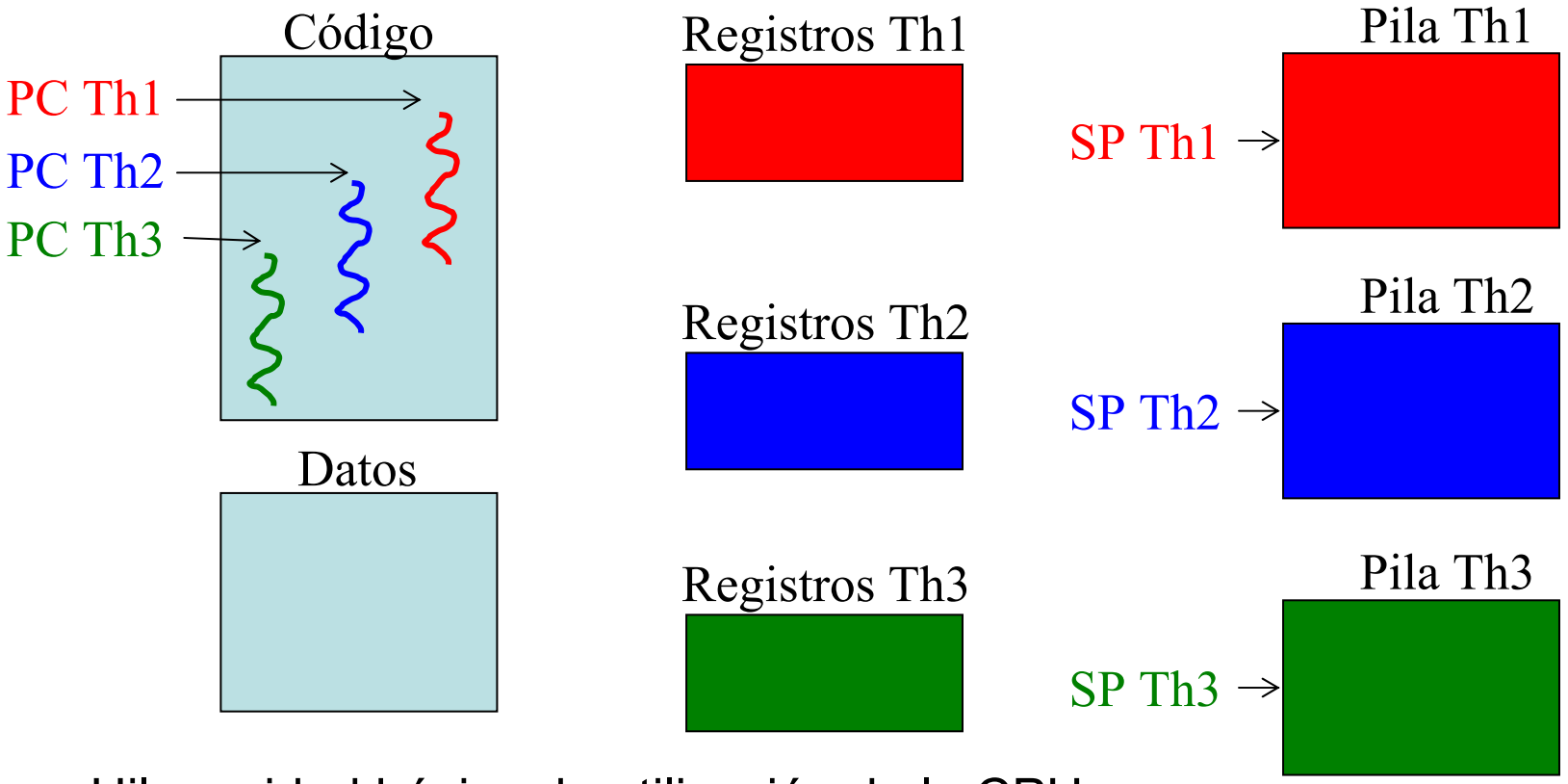
- Ejemplo: sumar los elementos de un vector muy grande
- Disponibilidad de varios procesadores
- Objetivo: aumentar rendimiento
  - Repartir el trabajo entre varios procesos
  - Cada proceso trabaja sobre un trozo del vector
  - Cada proceso ejecuta en un procesador
- Ejemplos 1 y 2
  - independientes de la plataforma, sirven para uno o varios cores
  - Objetivo principal: facilitar la programación
- Ejemplo 3
  - Solo tiene sentido en un sistema multicore,
  - Casi siempre con numero de threads ≤ numero de cores
  - Objetivo: rendimiento
  - Complica la programación

Procesos e hilos

- Los procesos (tal como los hemos visto hasta ahora):
  - **Son propietarios de recursos** – espacio de memoria para almacenar su imagen, ficheros abiertos, tratamiento de señales, …
  - **Son la unidad de ejecución/planificación** – siguen un camino de ejecución, el SO les otorga tiempo en CPU

- En el modelo proceso/hilo, estas dos características se tratan de forma independiente por el sistema operativo
  - Los procesos **son propietarios de recursos**
  - Los hilos **son la unidad de ejecución/planificación**
  - Un proceso puede tener varios hilos

Modelo de proceso con varios hilos



- Hilo: unidad básica de utilización de la CPU
  - Comprende un ID de hilo, un PC, un conjunto de registros y una pila
  - Comparte con otros hilos: código, datos y recursos de sistema asignados al programa (ficheros abiertos, señales, …)

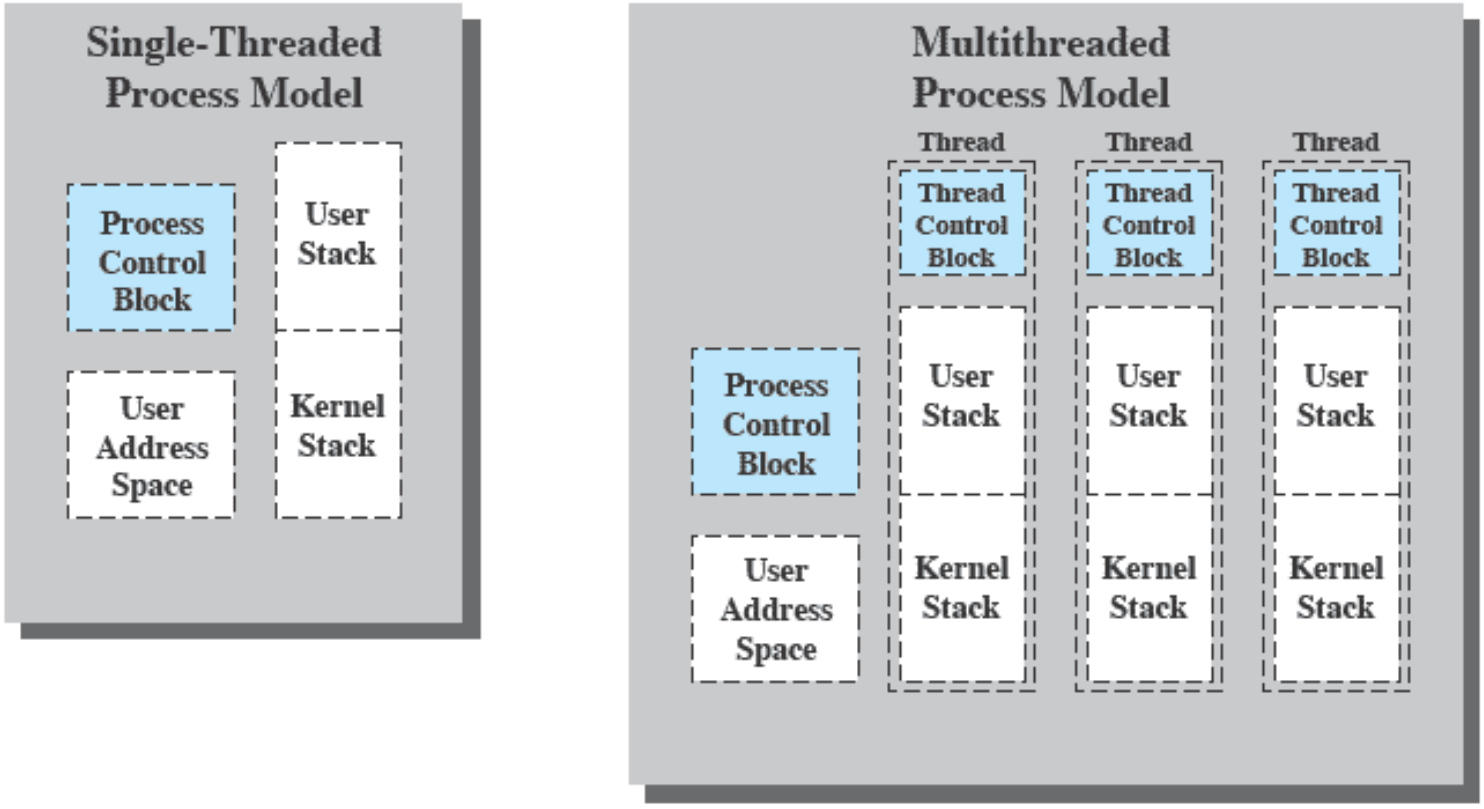
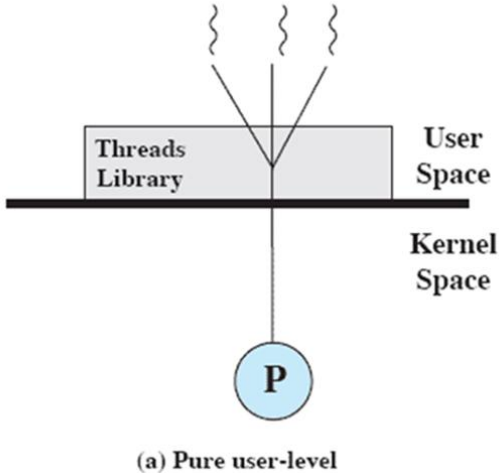


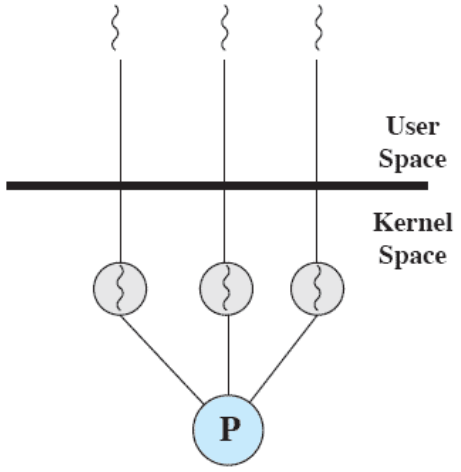
Figure 4.2 Single Threaded and Multithreaded Process Models

Formas de implementar hilos

- Hilos de usuario (User Level Thread, ULT)
  - La gestión de hilos la lleva la propia aplicación (biblioteca)
  - El SO no da ningún soporte, no conoce los hilos
- Hilos de sistema (Kernel level Thread, KLT)
  - también llamados: kernel-supported threads, lightweight processes
  - El SO conoce y gestiona hilos, planifica a nivel de hilo



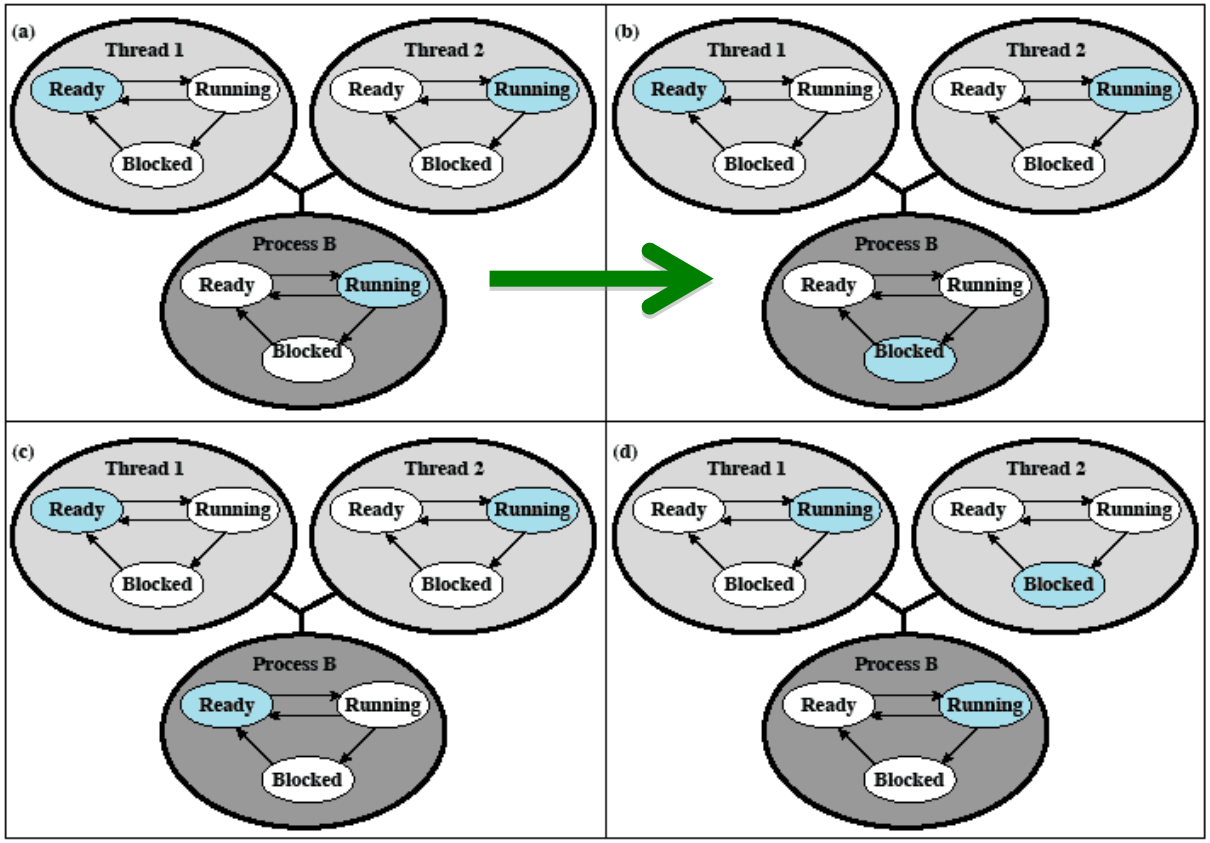
(a) Pure user-level



(b) Pure kernel-level

Hilos de usuario: planificación en dos niveles

Situación inicial



Colored state
is current state

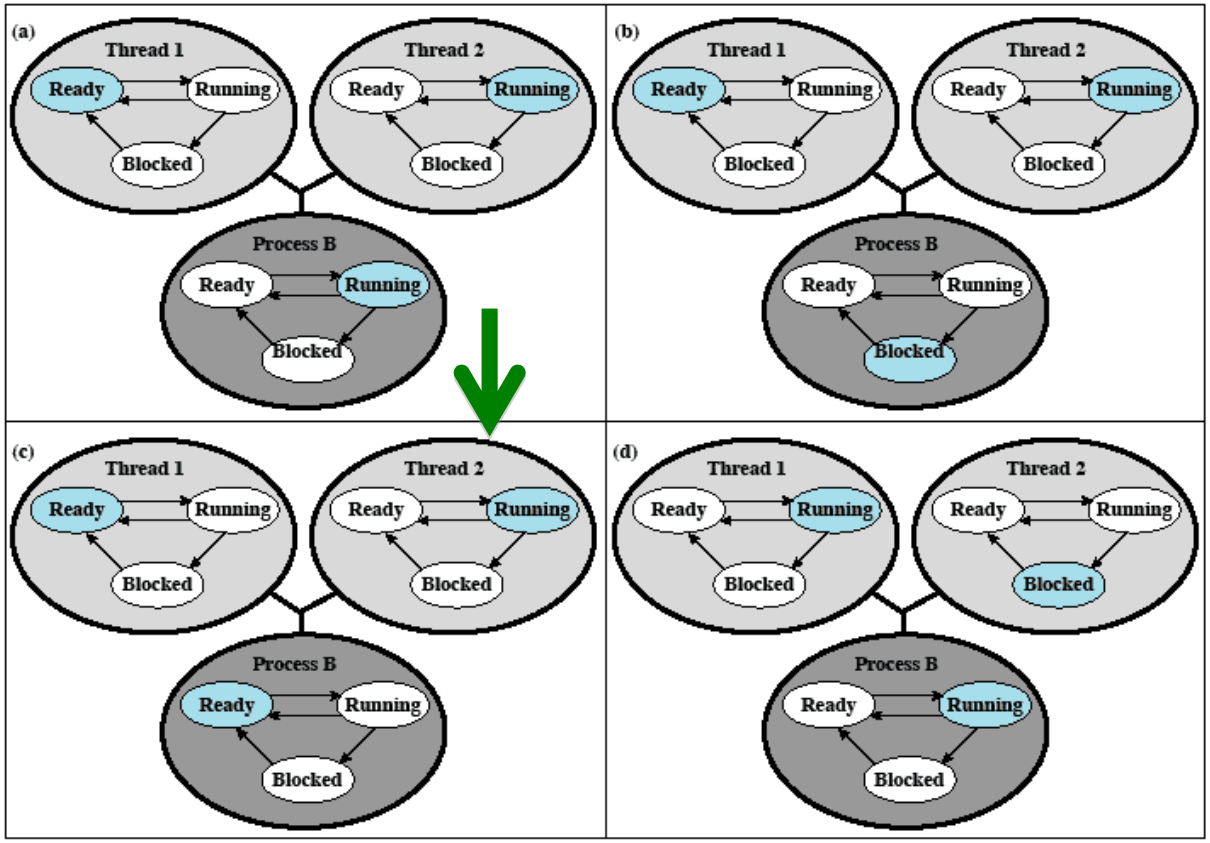
Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Th2 SC
Bloquea
Al proceso B

Hilos de usuario: planificación en dos niveles

Situación inicial

Quantum proceso B

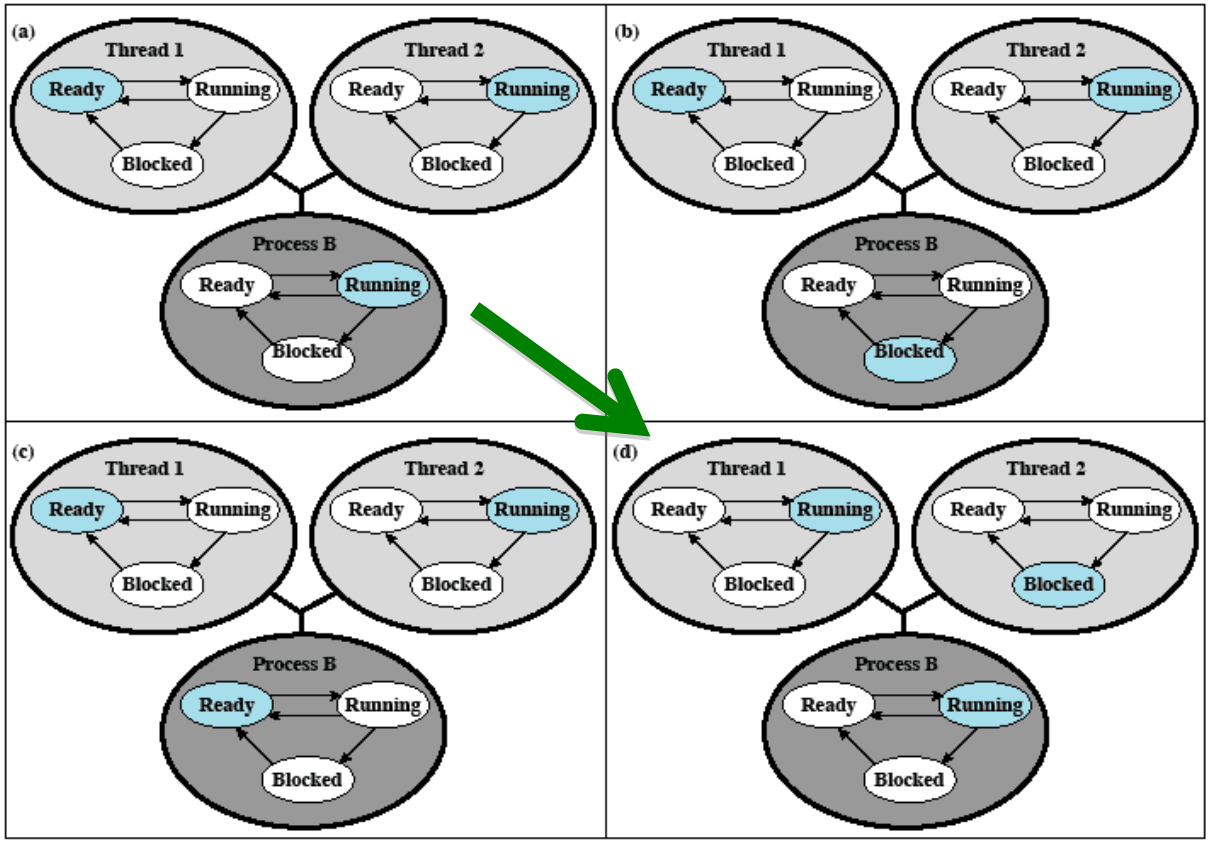


Colored state
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Hilos de usuario: planificación en dos niveles

Situación inicial



Colored state
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Th2 sincroniza con Th1 y se Bloquea

Hilos de sistema: Ventajas/Desventajas

- Si un hilo se bloquea, no bloquea a todo el proceso. El kernel puede poner en ejecución otros hilos del mismo proceso
- En un sistema multiprocesador, el kernel puede poner simultáneamente en ejecución varios hilos del mismo proceso
- Tienen mayor coste en cambio de contexto de hilo
- Tienen mayor coste para crear, señalizar/esperar, …

Latencia en uSg. VAX/UNIX	ULT	KLT	proceso
Crear	34	948	11.300
Señalizar-esperar	37	441	1.840

Bibliotecas de hilos

- Contienen la colección de funciones ofrecidas al usuario para trabajar con hilos
  - Crear, terminar, funciones de sincronización,…
- Ejemplos:
  - POSIX Pthreads (existe como ULT y KLT)
  - Windows Threads (KLT)
  - Java Threads (se implementa sobre la librería del host)
- Veremos Pthreads en el siguiente tema

Hilos en Linux

- Además de fork(), linux ofrece llamada clone()
- Clone() permite determinar el grado de compartición entre padre e hijo

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.