Programación Avanzada(TC2025)

Tema 5. Programación concurrente

Instituto Tecnológico y de Estudios Superiores de Monterrey. Campus Santa Fe
Departamento de Tecnologías de Información y Electrónica
Dr. Vicente Cubells (vcubells@itesm.mx)

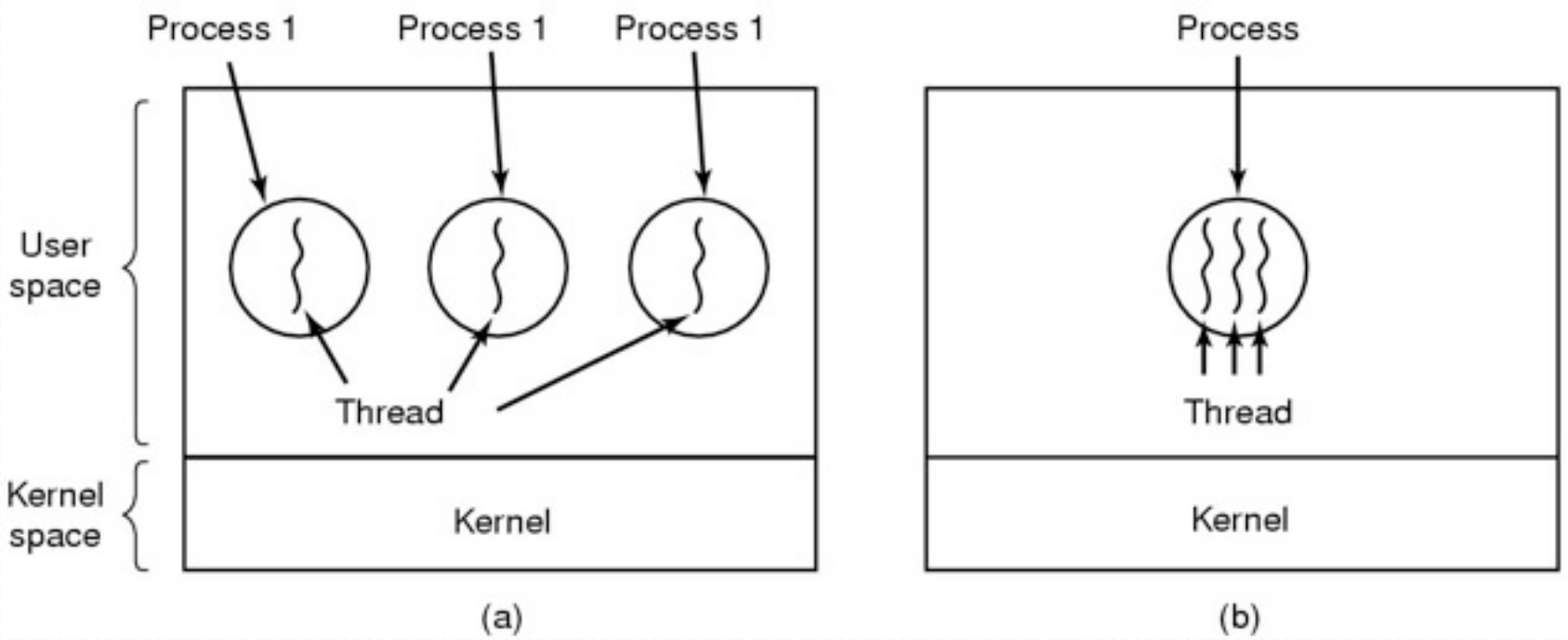
Temario

- Introducción a los hilos
- Introducción a POSIX Threads
- Compilación de programas multihilos
- Algunos ejemplos

Hilos

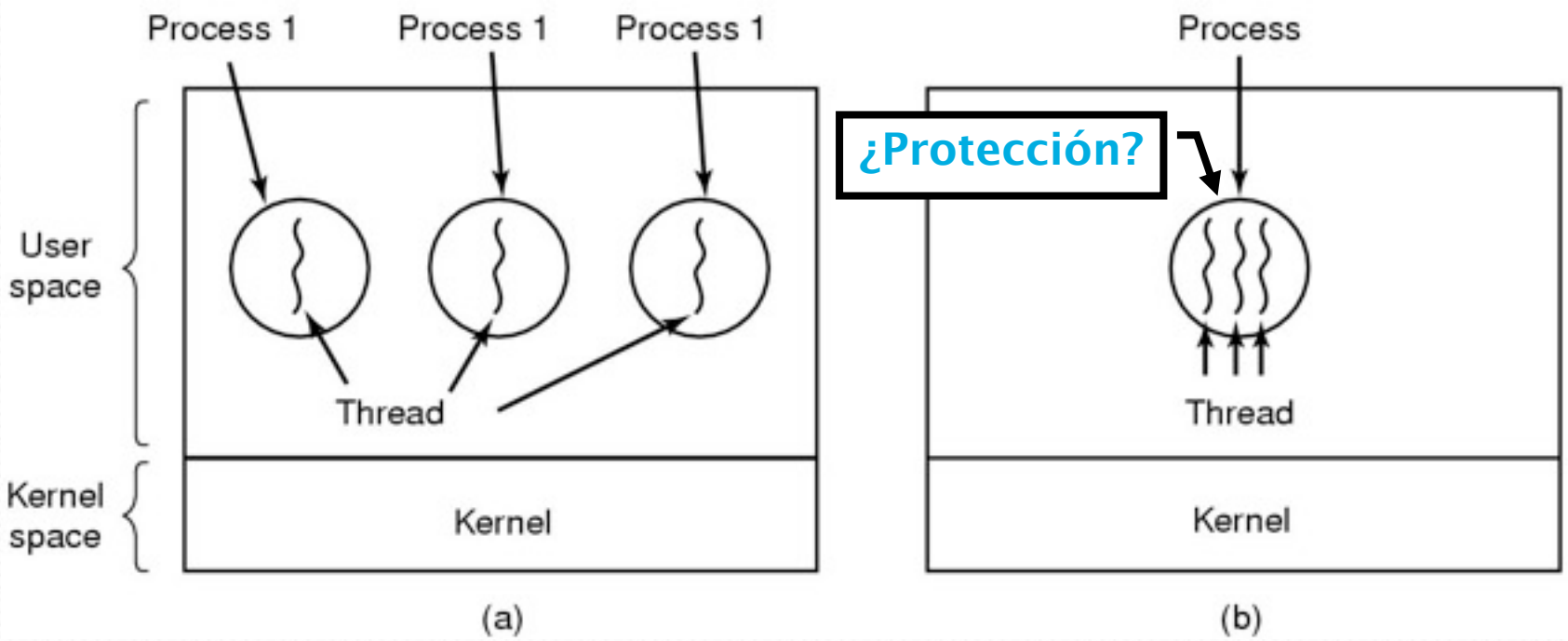
¿Es el proceso la unidad básica de procesamiento?

El modelo de hilos…



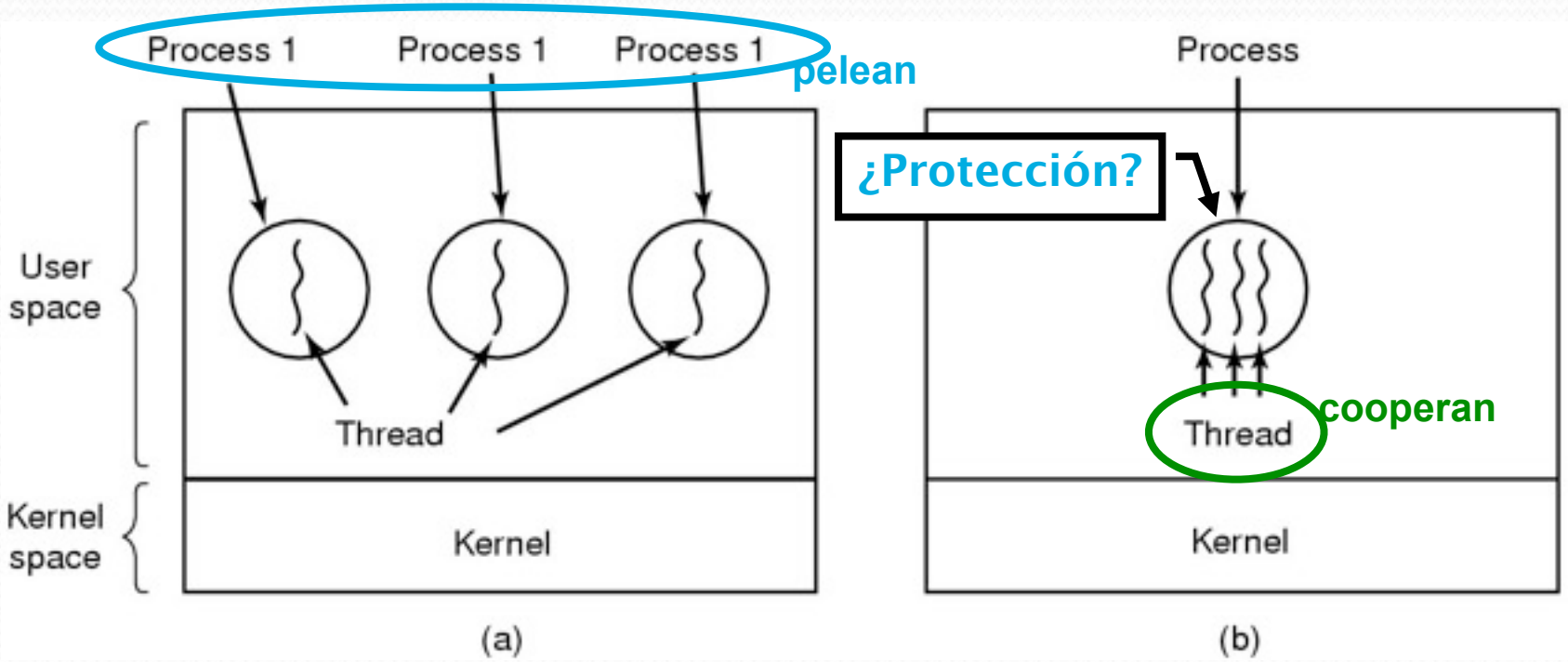
(a) Tres procesos cada uno con un hilo
(b) Un proceso con tres hilos

El modelo de hilos…



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

El modelo de hilos…



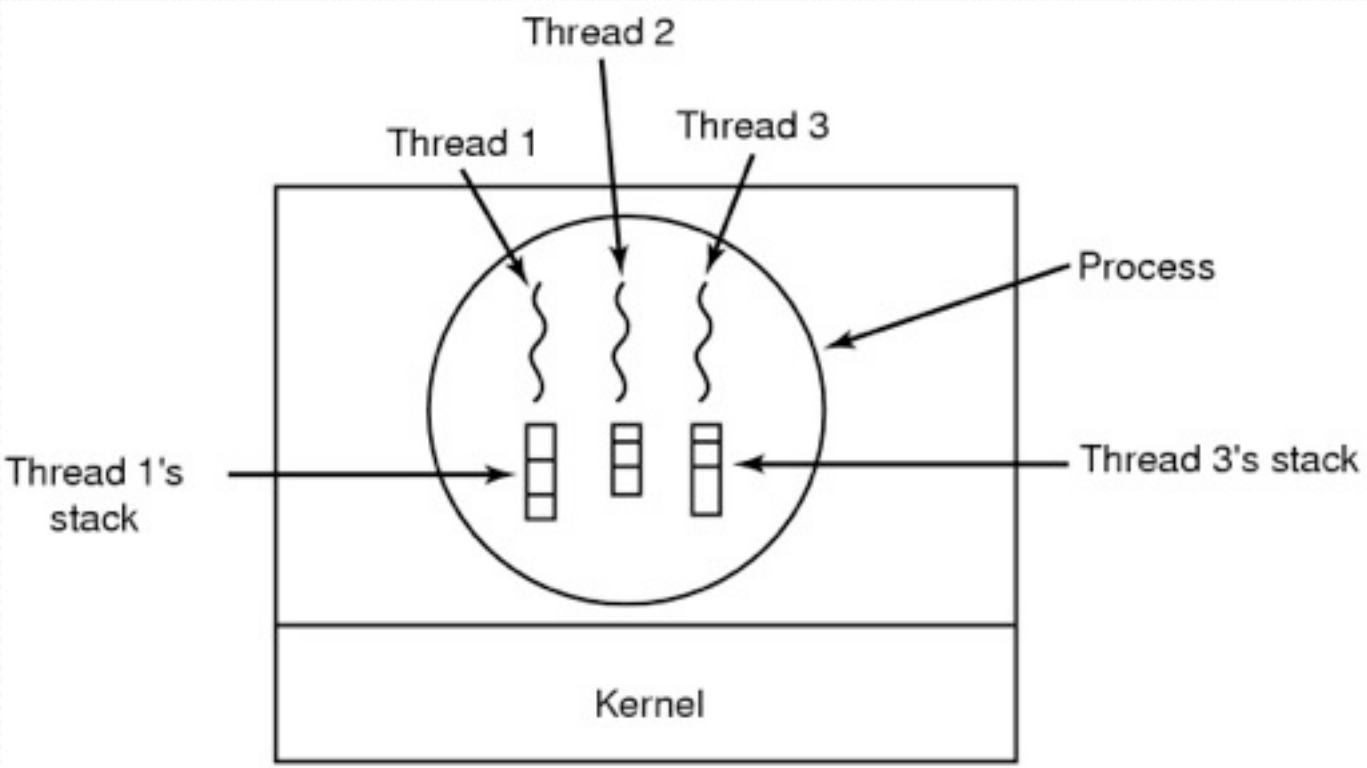
(a) Tres procesos cada uno con un hilo
(b) Un proceso con tres hilos

El modelo de hilos…

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	

- Elementos compartidos por todos los hilos de un proceso
- Elementos privados a cada hilo

El modelo de hilos



Cada hilo tiene su propia pila

Problemas con los hilos

Problemas con los hilos

- Llamada fork()
  - Proceso hijo = proceso padre
    - ¿Los mismo hilos?
  - Ejemplo: teclado
    - Dos hilos bloqueados
    - ¿Quién lee, el padre, el hijo o ambos?

Problemas con los hilos

- Llamada fork()
  - Proceso hijo = proceso padre
    - ¿Los mismo hilos?
  - Ejemplo: teclado
    - Dos hilos bloqueados
    - ¿Quién lee, el padre, el hijo o ambos?
- Estructuras de datos compartidas
  - Ejemplo: archivos

Problemas con los hilos

- Llamada fork()
  - Proceso hijo = proceso padre
    - ¿Los mismo hilos?
  - Ejemplo: teclado
    - Dos hilos bloqueados
    - ¿Quién lee, el padre, el hijo o ambos?
- Estructuras de datos compartidas
  - Ejemplo: archivos
- Solicitud de memoria
  - Reservación duplicada

Uso de los hilos…

- Entidades paralelas
  - Mismo espacio de direcciones y datos
- No tienen recursos asociados
  - Más fáciles de crear y destruir
  - Se crean 100 veces más rápido que un proceso

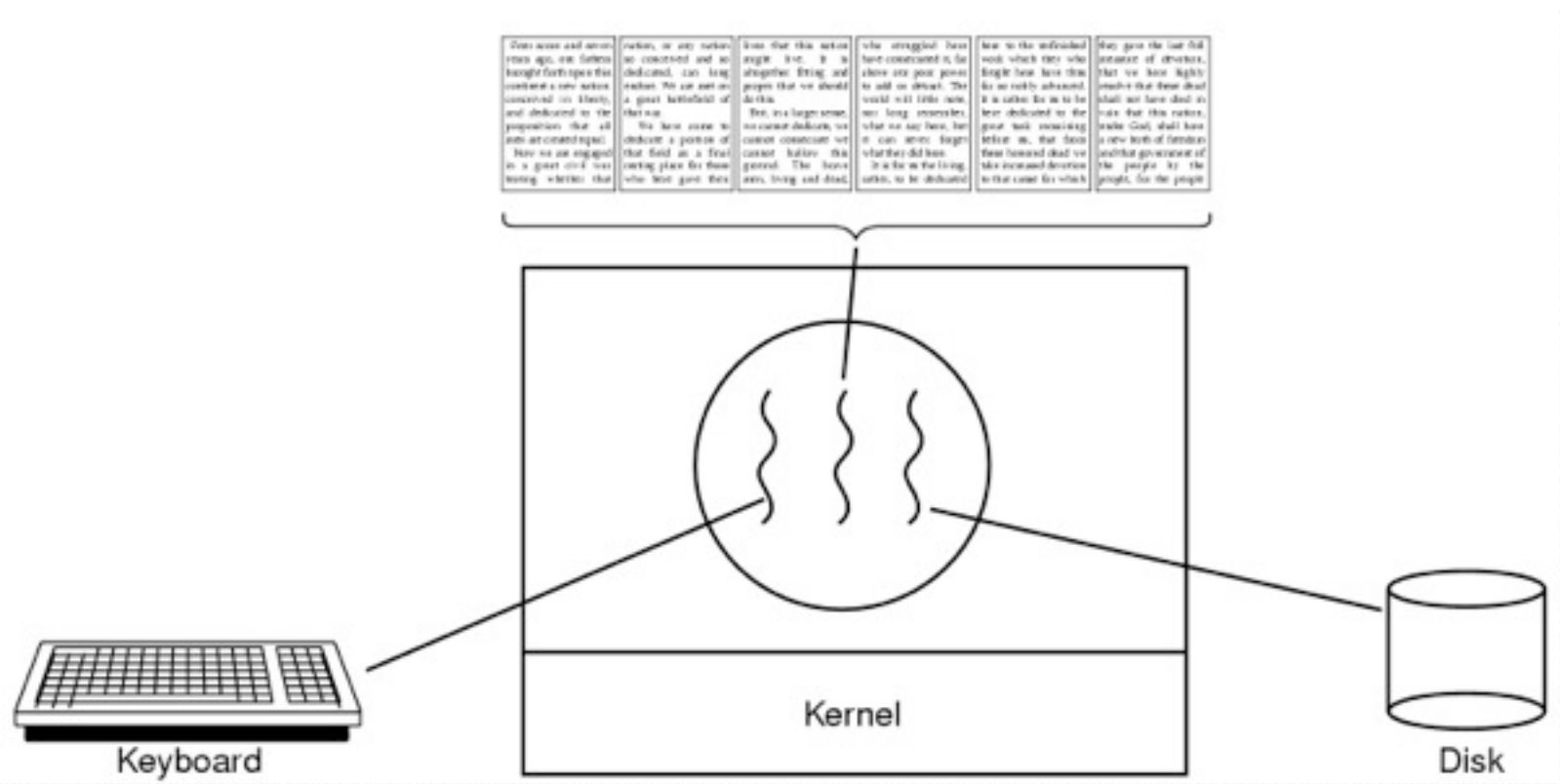
Uso de los hilos…

- Entidades paralelas
  - Mismo espacio de direcciones y datos
- No tienen recursos asociados
  - Más fáciles de crear y destruir
  - Se crean 100 veces más rápido que un proceso
- Rendimiento

Uso de los hilos…

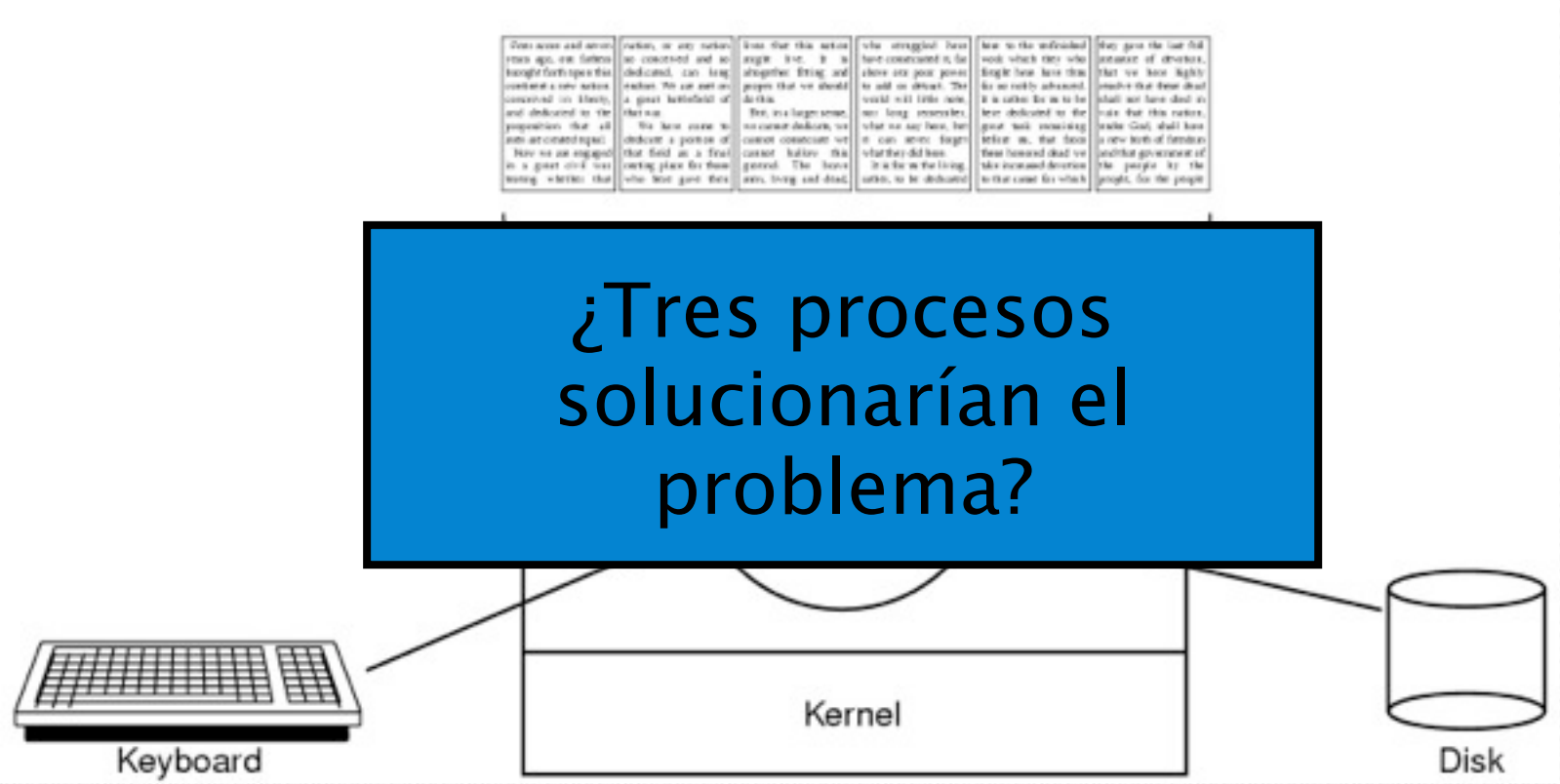
- Entidades paralelas
  - Mismo espacio de direcciones y datos
- No tienen recursos asociados
  - Más fáciles de crear y destruir
  - Se crean 100 veces más rápido que un proceso
- Rendimiento
- Sistemas multiprocesadores

Uso de los hilos…



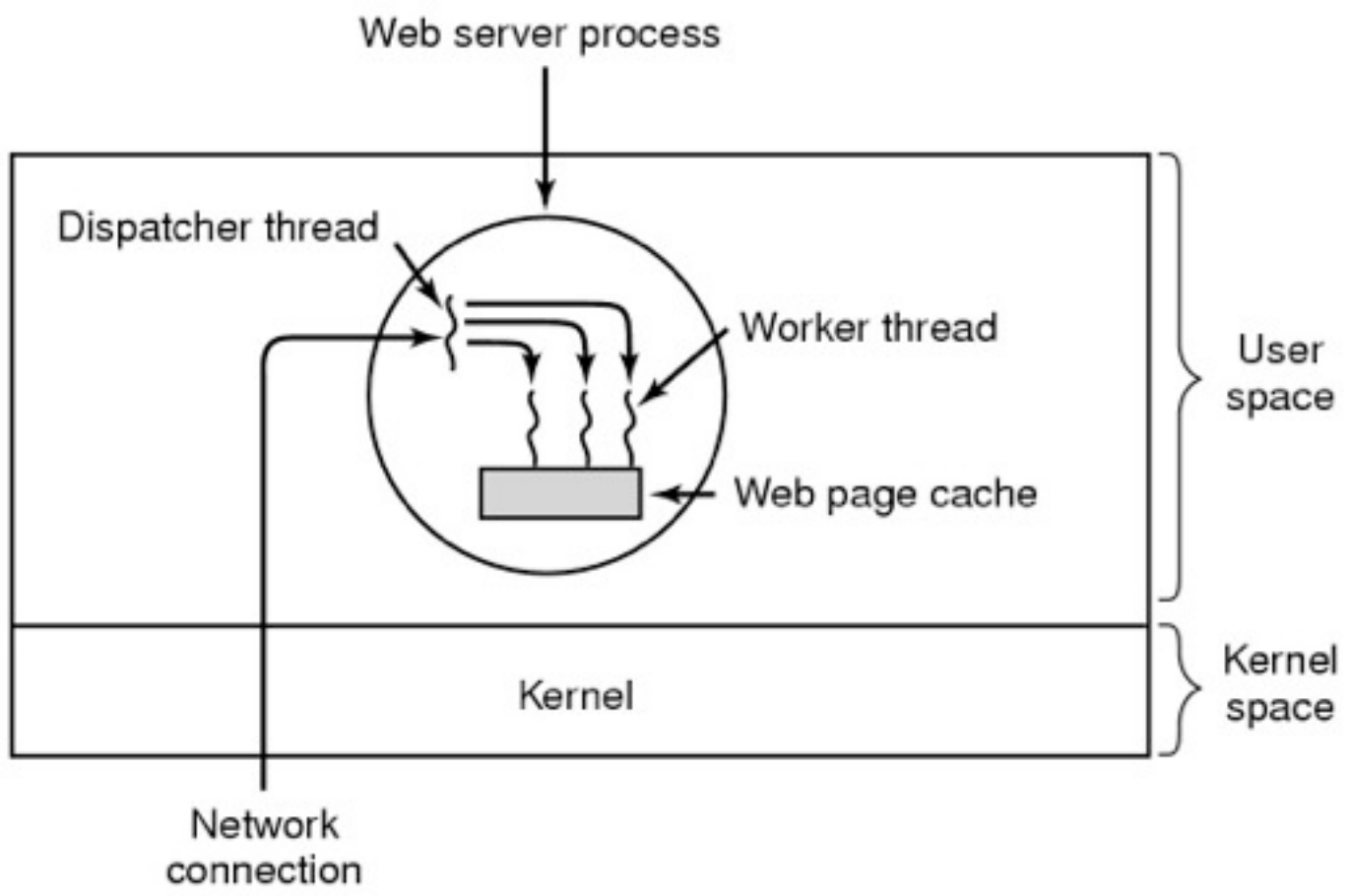
Un procesador de textos con tres hilos

Uso de los hilos…



Un procesador de textos con tres hilos

Uso de los hilos...



Un servidor Web multihilo

Uso de los hilos...

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

Ejemplo de código correspondiente a la figura anterior

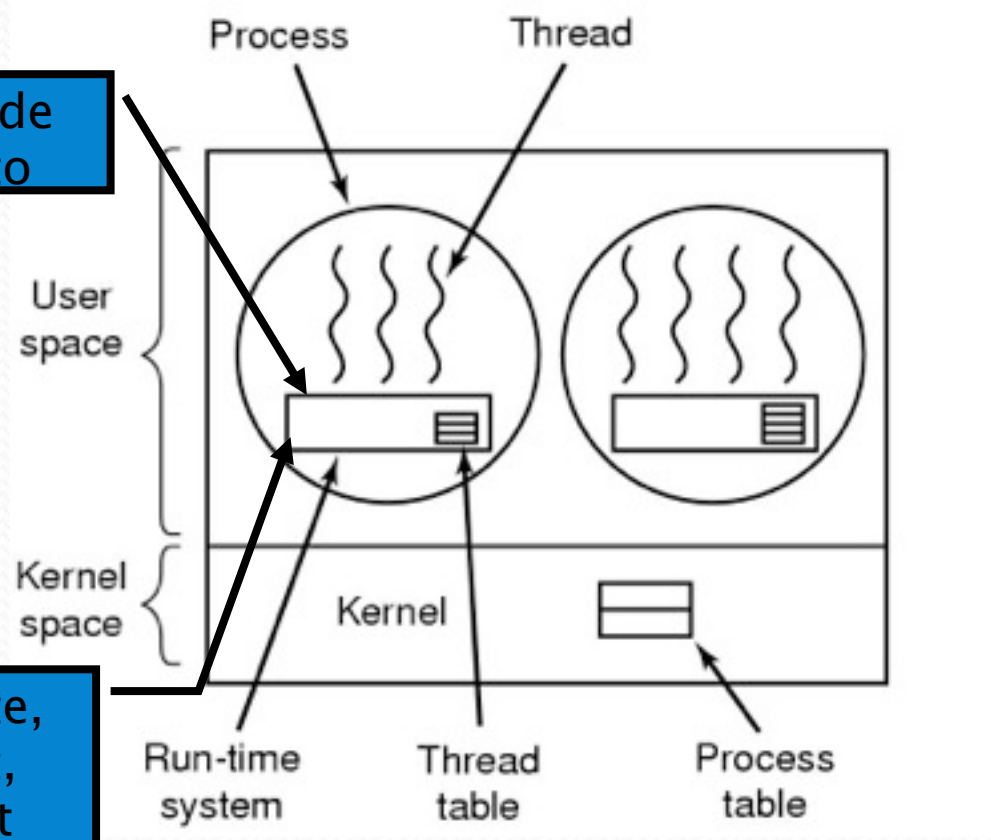
(a) Hilo distribuidor
(b) Hilo obrero

Uso de los hilos…

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

Tres formas de construir un servidor

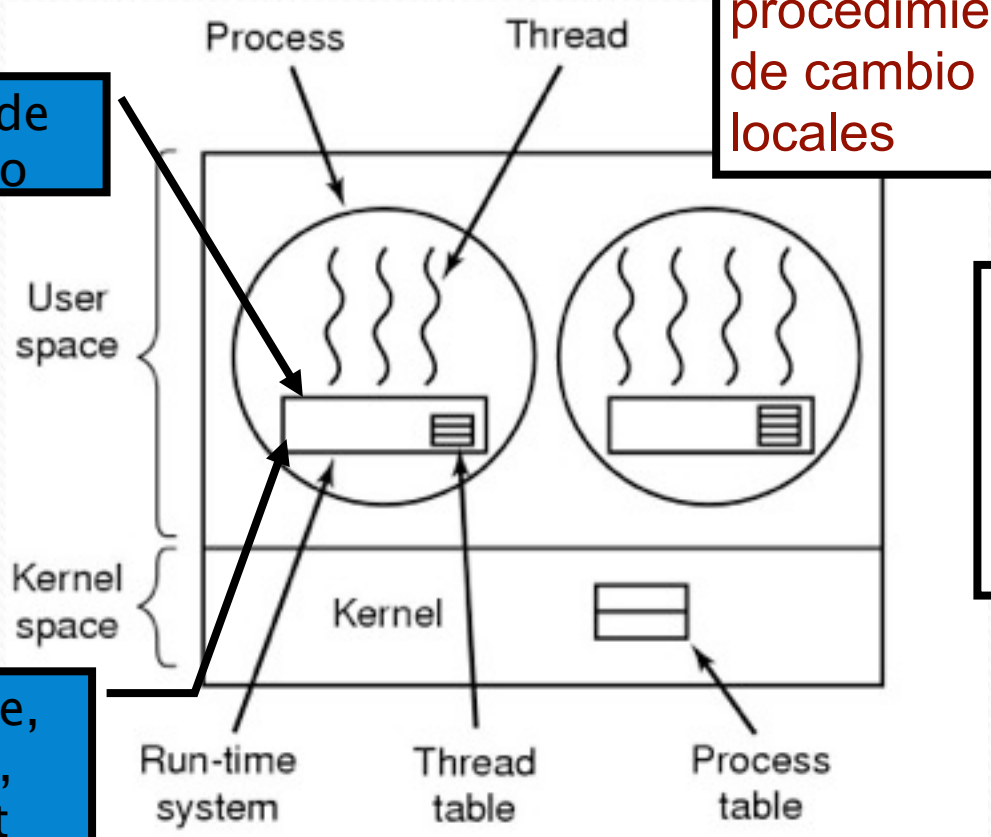
Implementando hilos en espacio de usuario



Conjunto de hilos en el nivel de usuario

Implementando hilos en [illegible] usuario

Cambio de contexto



thread_create, thread_exit, thread_wait

Planificador de hilos y procedimiento de cambio son locales

MAS EFICIENTE

No cambio de contexto
No vacío de la cache
No trap

MAS RAPIDO

Cada proceso su algoritmo de planificación

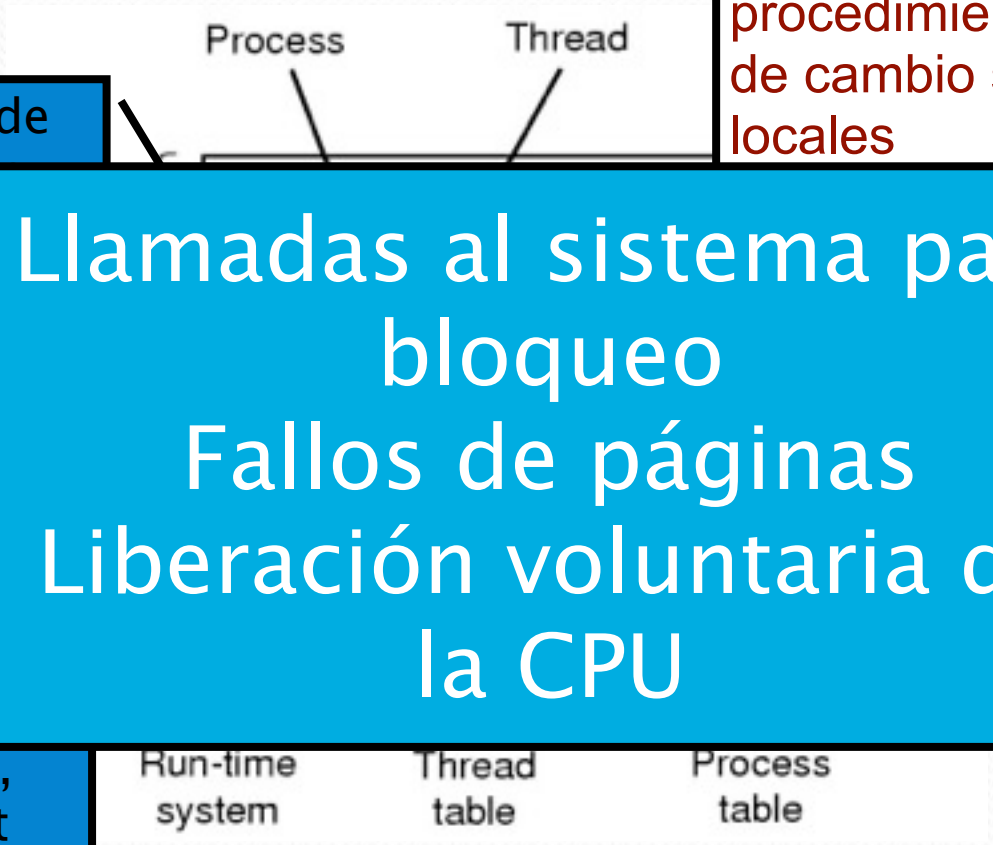
Conjunto de hilos en el nivel de usuario

Implementando hilos en [illegible] usuario

Planificador de hilos y procedimiento de cambio son locales

MAS EFICIENTE

Cambio de context[illegible]



Llamadas al sistema para bloqueo
Fallos de páginas
Liberación voluntaria de la CPU

[illegible]ambio de [illegible]exto [illegible]acío de [illegible]che [illegible]ap

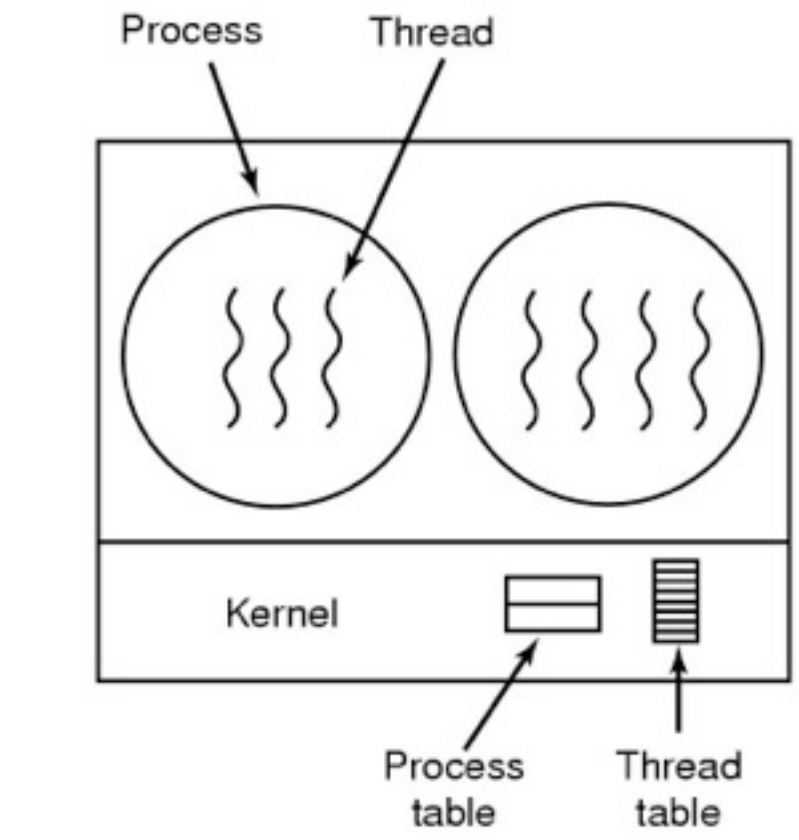
MAS RAPIDO

thread_creat[illegible]
thread_exit,
thread_wait

[illegible]ada proceso su algoritmo de planificación

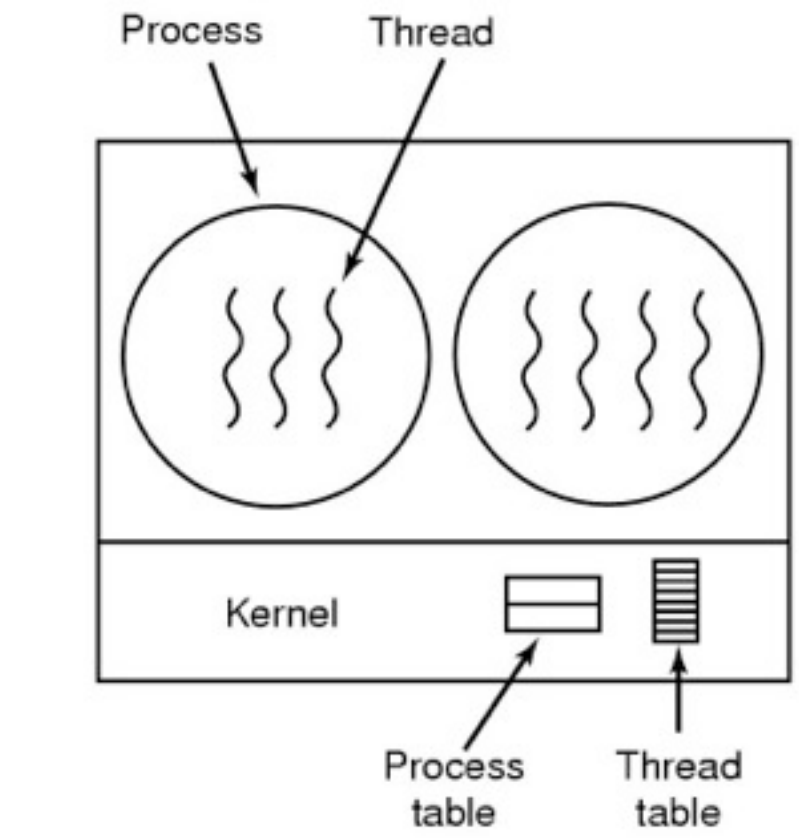
Conjunto de hilos en el nivel de usuario

Implementando hilos en el kernel



Conjunto de hilos administrado por el kernel

Implementando hilos en el kernel



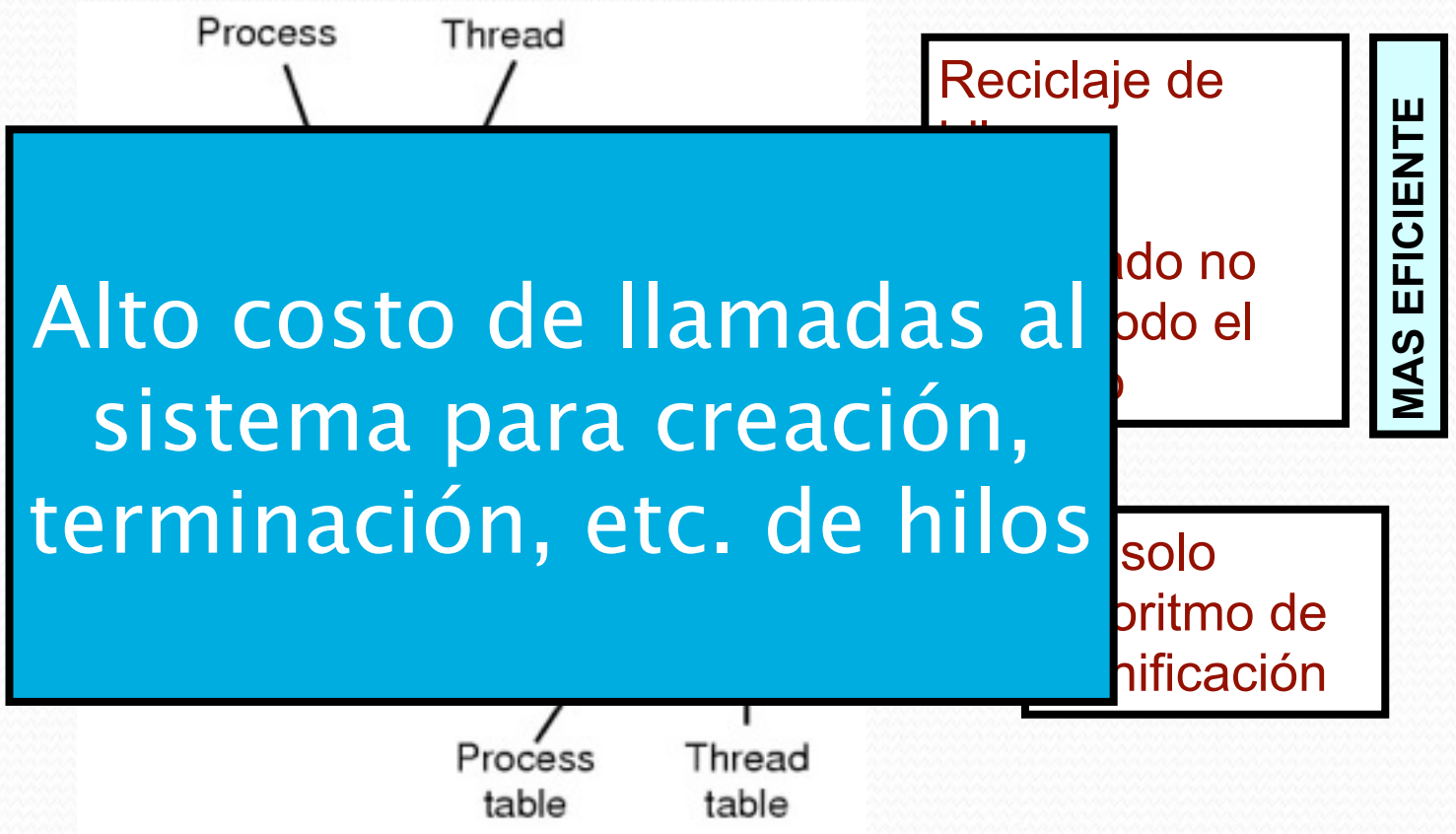
Reciclaje de hilos
Un hilo bloqueado no afecta todo el proceso

MAS EFICIENTE

Un solo algoritmo de planificación

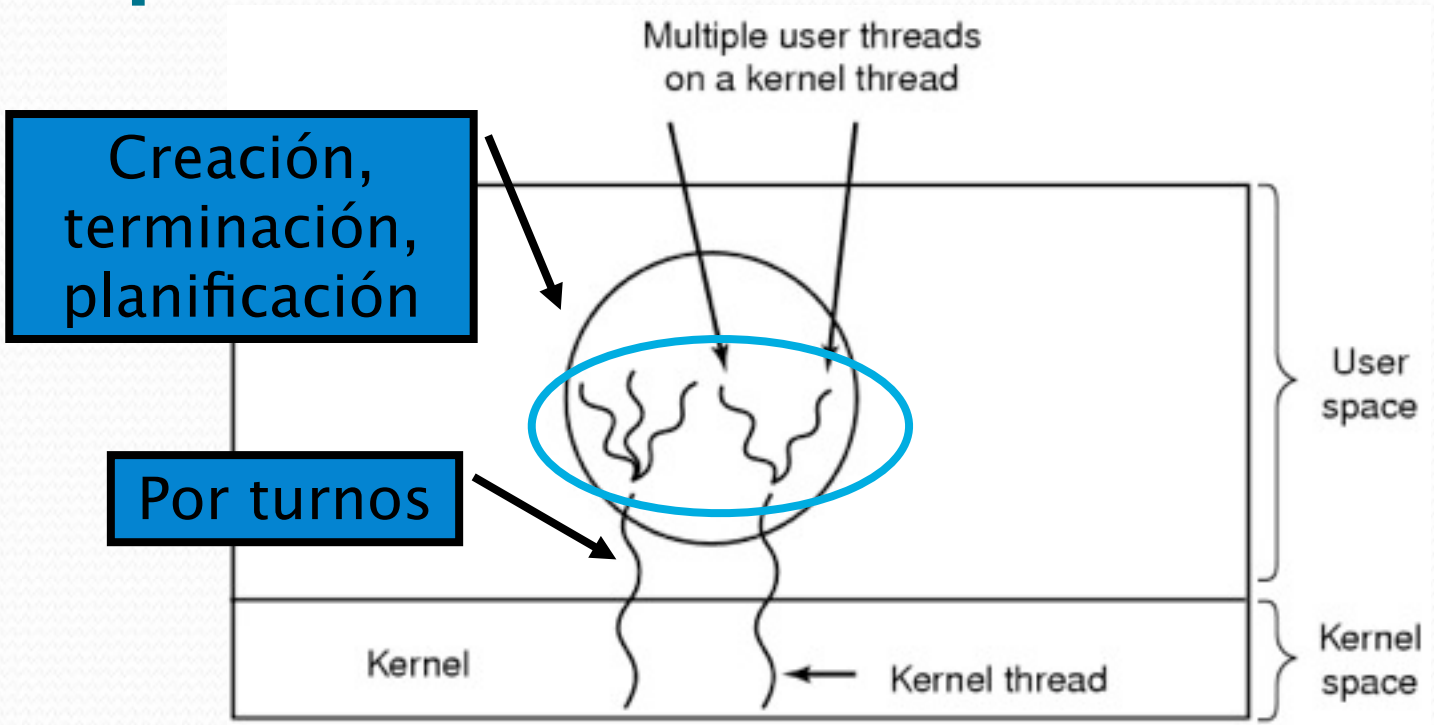
Conjunto de hilos administrado por el kernel

Implementando hilos en el kernel



Conjunto de hilos administrado por el kernel

Implementaciones híbridas



Multiplexando hilos de nivel usuario en hilos de nivel kernel

Activaciones del planificador

- Combinación de ambos esquemas

Activaciones del planificador

- Combinación de ambos esquemas
- Objetivo: funcionalidad de los hilos de nivel kernel
  - Rendimiento y flexibilidad de hilos a nivel de usuario

Activaciones del planificador

- Combinación de ambos esquemas
- Objetivo: funcionalidad de los hilos de nivel kernel
  - Rendimiento y flexibilidad de hilos a nivel de usuario
- Evita transiciones innecesarias entre los espacios de usuario y kernel

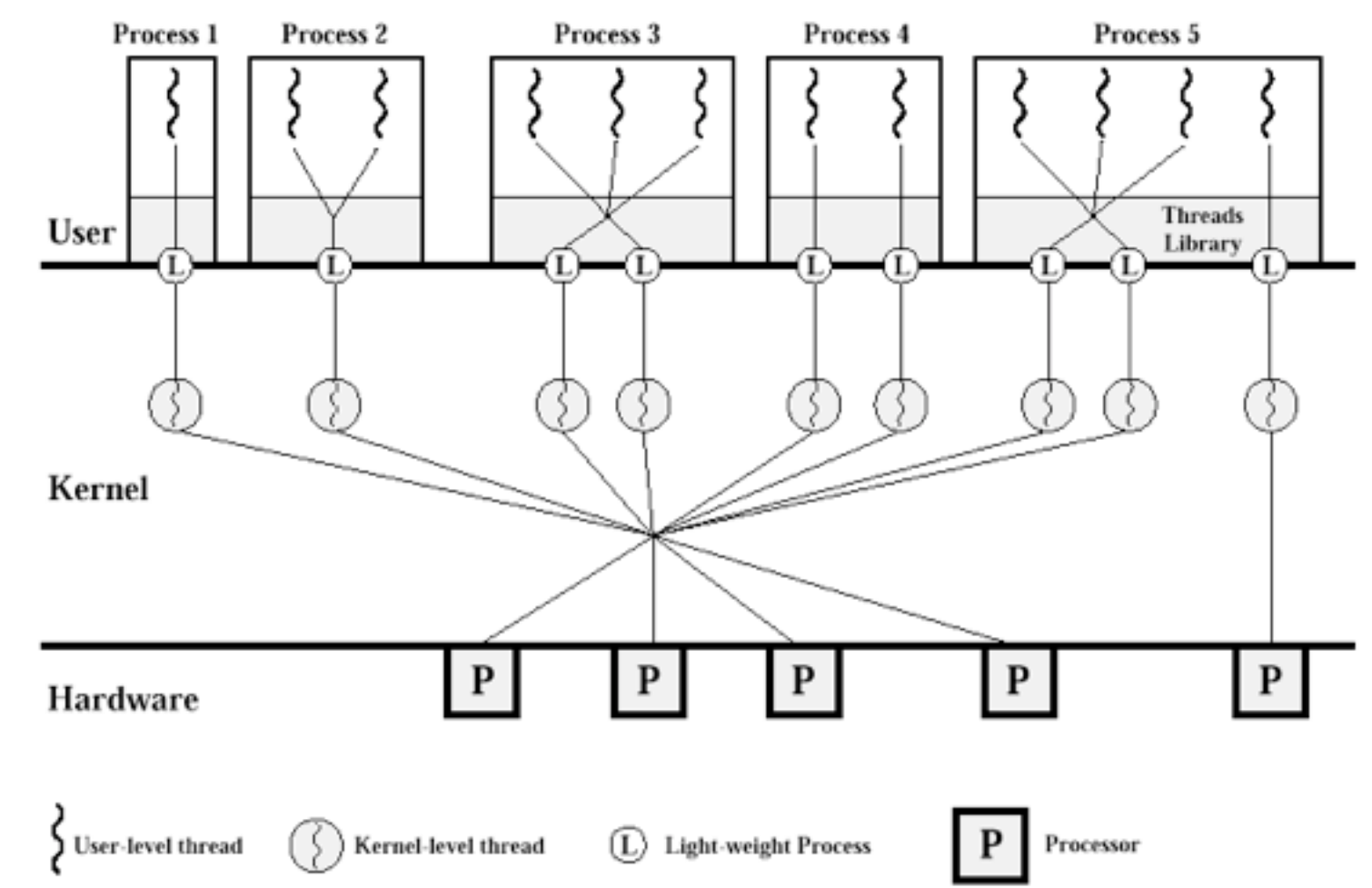
Activaciones del planificador

- Combinación de ambos esquemas
- Objetivo: funcionalidad de los hilos de nivel kernel
  - Rendimiento y flexibilidad de hilos a nivel de usuario
- Evita transiciones innecesarias entre los espacios de usuario y kernel
- Kernel asigna CPUs virtuales a cada proceso
  - El sistema de tiempo de ejecución asigne hilos a los procesadores

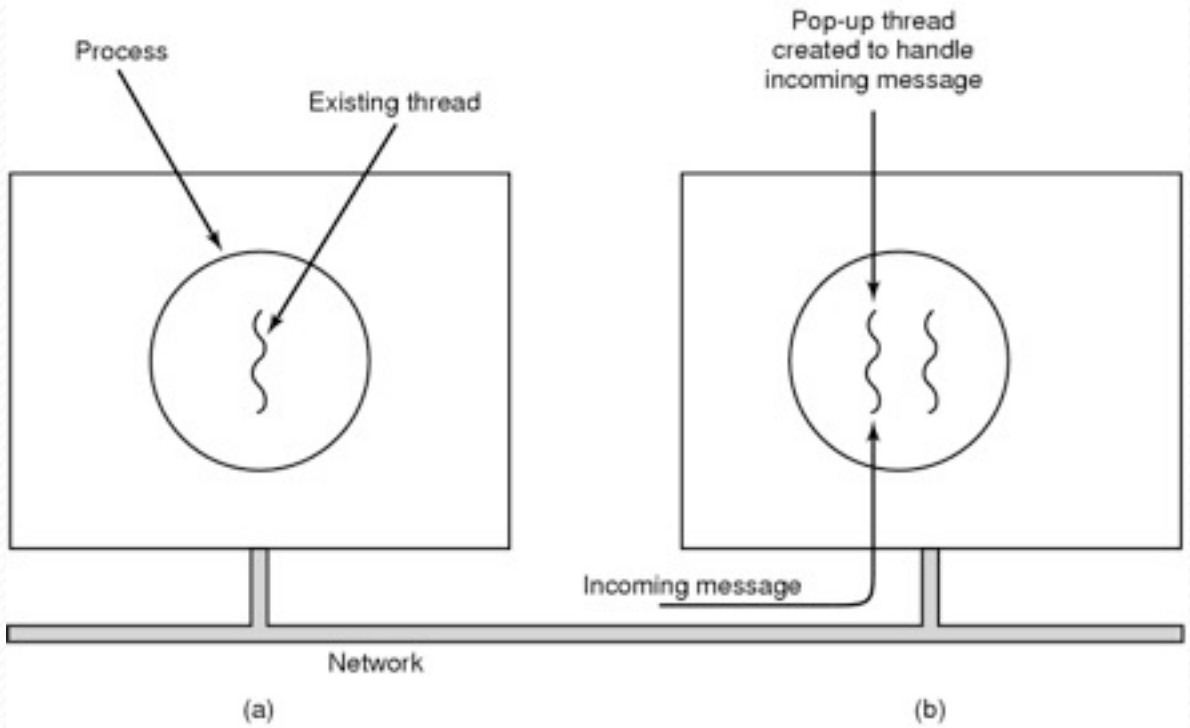
Activaciones del planificador

- Combinación de ambos esquemas
- Objetivo: funcionalidad de los hilos de nivel kernel
  - Rendimiento y flexibilidad de hilos a nivel de usuario
- Evita transiciones innecesarias entre los espacios de usuario y kernel
- Kernel asigna CPUs virtuales a cada proceso
  - El sistema de tiempo de ejecución asigne hilos a los procesadores
- Viola la estructura de un sistema por niveles

Implementación híbrida en Solaris



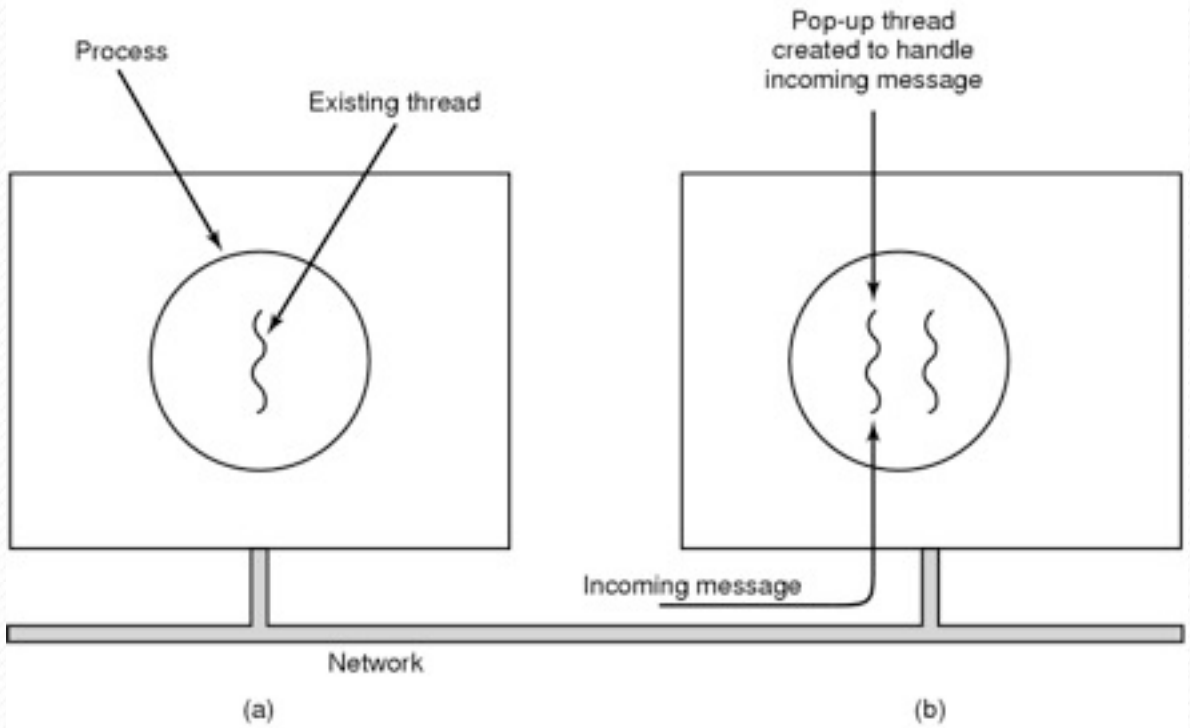
Hilos automáticos



Creación de un nuevo hilo cuando llega un mensaje

(a) Antes de llegar el mensaje
(b) Después de llegar el mensaje

Hilos automáticos



No se necesita restaurar valores

MAS RAPIDO

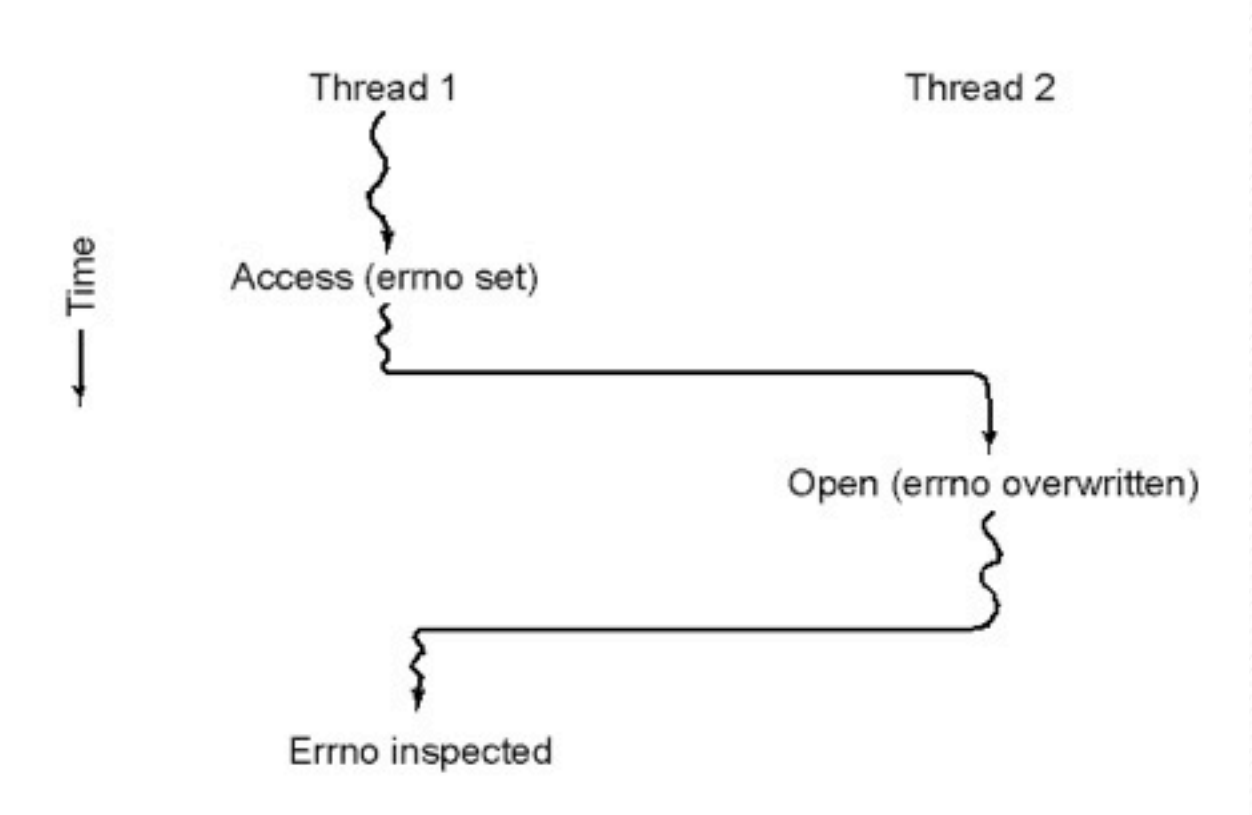
¿Dónde ejecutarlo, usuario o kernel?

Creación de un nuevo hilo cuando llega un mensaje

(a) Antes de llegar el mensaje

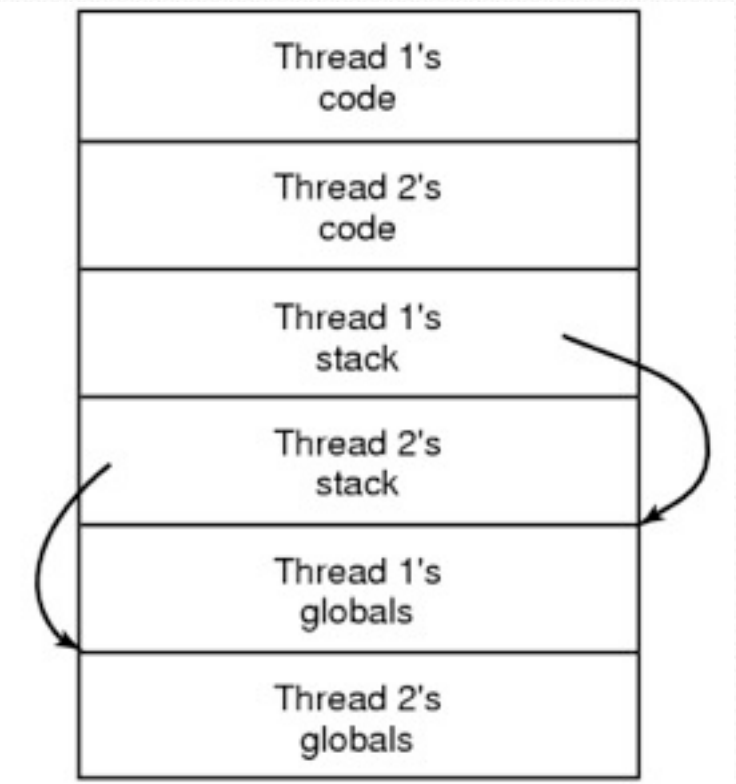
(b) Después de llegar el mensaje

Convirtiendo código de simple hilo a multihilo…



Conflictos entre hilos sobre el uso de variables globales

¿Cómo solucionarlo?



- Los hilos pueden tener variables globales privadas

Ventajas de los threads

- Comparten espacio de direcciones y datos
- No tienen recursos asignados
- Más fáciles de crear y destruir
  - Se crean hasta 100 veces más rápido que un proceso
- Velocidad de comunicación entre hilos
- Performance
  - E/S no bloquea todo el proceso
- Portabilidad

Bibliotecas para el trabajo con hilos

- POSIX: libpthread <pthreads.h>
  - Se compila con **`cc`** o **`gcc –lpthread`**

    `cc` o **`gcc –pthread`**

- Solaris: libthread
  - Se compila con **`cc`** o **`gcc -lthread`**

Biblioteca de POSIX <pthread.h>

- Creando hilos

```
int pthread_create(
    pthread_t *tid,
    const pthread_attr_t *tattr,
    void*(*start_routine)(void *),
    void *arg);
```

- Cuando los atributos no son especificados (tattr es NULL), se crea un hilo con los siguientes atributos:
  - No se puede desenlazar del proceso
  - Tiene una pila y un tamaño de pila predeterminado
  - Hereda la prioridad del padre

Biblioteca de POSIX <pthread.h>

- Un ejemplo simple

```
#include <pthread.h>

pthread_attr_t tattr;
pthread_t tid;
extern void *start_routine(void *arg);
void *arg;
int ret;
/* Comportamiento predeterminado */
ret = pthread_create(&tid, NULL, start_routine, arg);

/* inicializado con atributos */
ret = pthread_attr_init(&tattr);
ret = pthread_create(&tid, &tattr, start_routine, arg);
```

= 0 cuando no hay errores

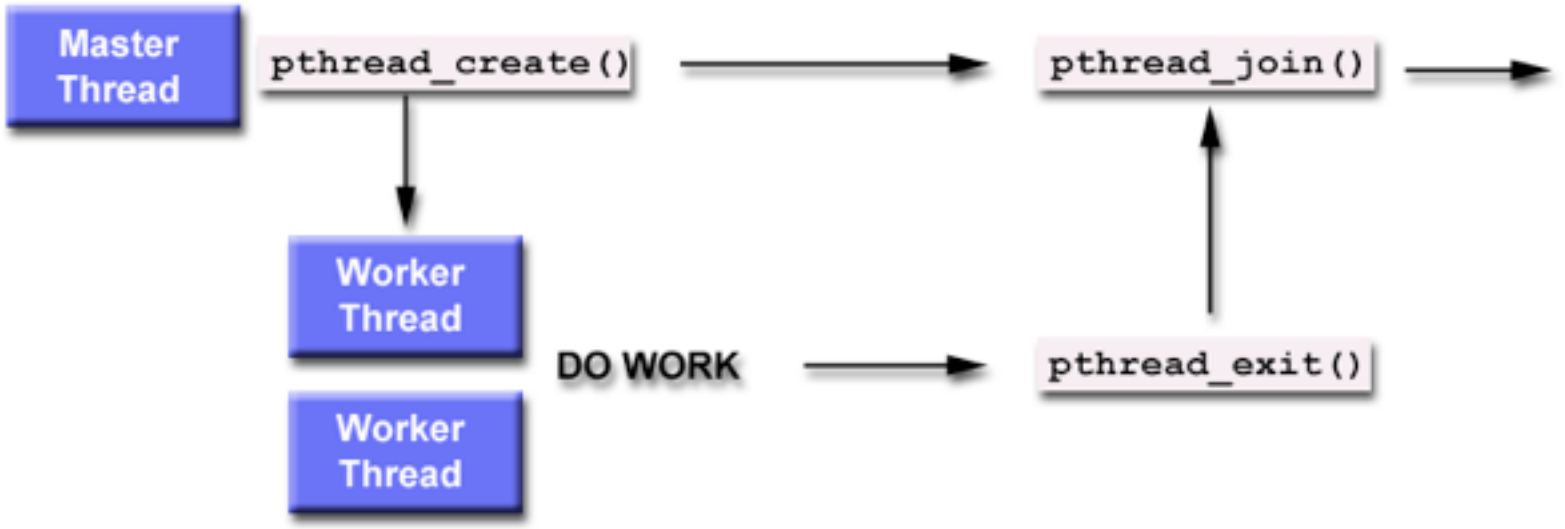
ID del nuevo hilo

Ver ejemplo: t5c2e1

Ver ejemplo: t5c2e2

Biblioteca de POSIX <pthread.h>

- Unir y separar hilos



Biblioteca de POSIX <pthread.h>

- Esperando por la terminación de un hilo

```
int pthread_join(
    thread_t tid,
    void *status);
```

- Bloquea al hilo invocador hasta que el hilo invocado termina
- Múltiples hilos no pueden esperar por la terminación de un mismo hilo
- Si status != NULL, almacena el valor de retorno conque el hilo invocado termina
- Para hilos que no se pueden separar (sincronización)

Biblioteca de POSIX <pthread.h>

- Un ejemplo simple

```
#include <pthread.h>

pthread_t tid;
int ret;
int status;

/* espera por la terminación del hilo "tid" con status */
ret = pthread_join(tid, &status);

/* espera por la terminación del hilo "tid" sin status */
ret = pthread_join(tid, NULL);
```

Ver ejemplo: t5c2e3

Biblioteca de POSIX <pthread.h>

- La función `pthread_detach()`
  - Una variante a `pthread_join()`
  - Para hilos que se pueden separar
  - Reclama la memoria ocupada por el hilo

```
#include <pthread.h>

pthread_t tid;
int ret;

/* detach thread tid */
ret = pthread_detach(tid);
```

Problemas con las variables globales

```
int pthread_mutex_lock(
                    pthread_mutex_t *mutex);

int pthread_mutex_trylock(
                    pthread_mutex_t *mutex);

int pthread_mutex_unlock(
                    pthread_mutex_t *mutex);
```

Ver ejemplo: t5c2e4

Resumiendo

- Dos bibliotecas para el trabajo con hilos
- Los hilos se pueden crear con atributos predeterminados o atributos establecidos
- Un hilo puede esperar por otro hilo que termine para unirlo al hilo principal (sincronización) o para obtener su memoria
- Un hilo puede terminar de tres maneras posibles
- Las variables globales se pueden proteger con mutexes