# Contenido 4: Threads

Lesly Reyes
lareyes2@uc.cl

Álvaro Quintana
alvaroq@uc.cl

Departamento de Ciencias de la Computación
Pontificia Universidad Católica Chile

August 21, 2024

# Contenido

Thread
- Threads y Multicores
- ¿Para qué threads?
- Modelos de Threads
- Funciones típicas de threads
- Implementación de threads
- Implementaciones Híbridas

Material Adicional

# Thread

## ¿Qué es un thread?

**Thread** Como un proceso, pero más liviano.
Unidad básica de uso de CPU

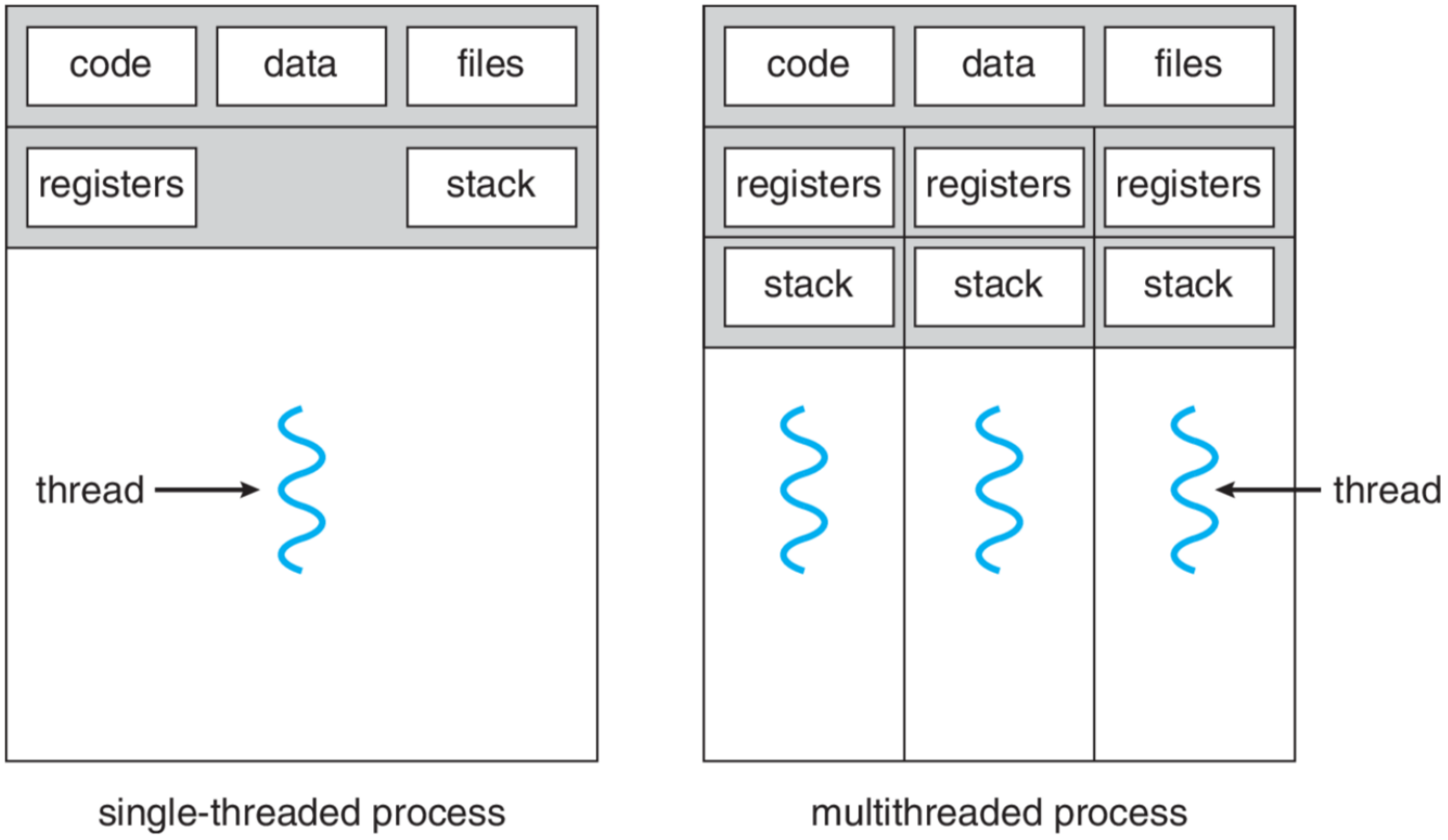
- Thread ID (tid)
- Program Counter
- Registros
- Stack

El resto está compartido con otros threads del mismo proceso.

# Thread

## Single-threading vs Multi-threading

Procesos pueden tener uno o más threads



# Thread

## ¿Para qué threads?

**Procesos complejos deben atender múltiples tareas**

- Programar con threads puede hacer el código más simple
  - Threads pueden ejecutar distintas partes del código de un proceso
  - Apropiados para programación guiada por eventos
- Programar con threads puede ayudar a paralelizar/distribuir tareas
  - Cuidado. Threads no necesariamente ejecutan al mismo tiempo.
  - Solo en presencia de múltiples CPU (cores) podría haber real concurrencia
  - Un programa con mútiples threads podría ser más lento

# Thread

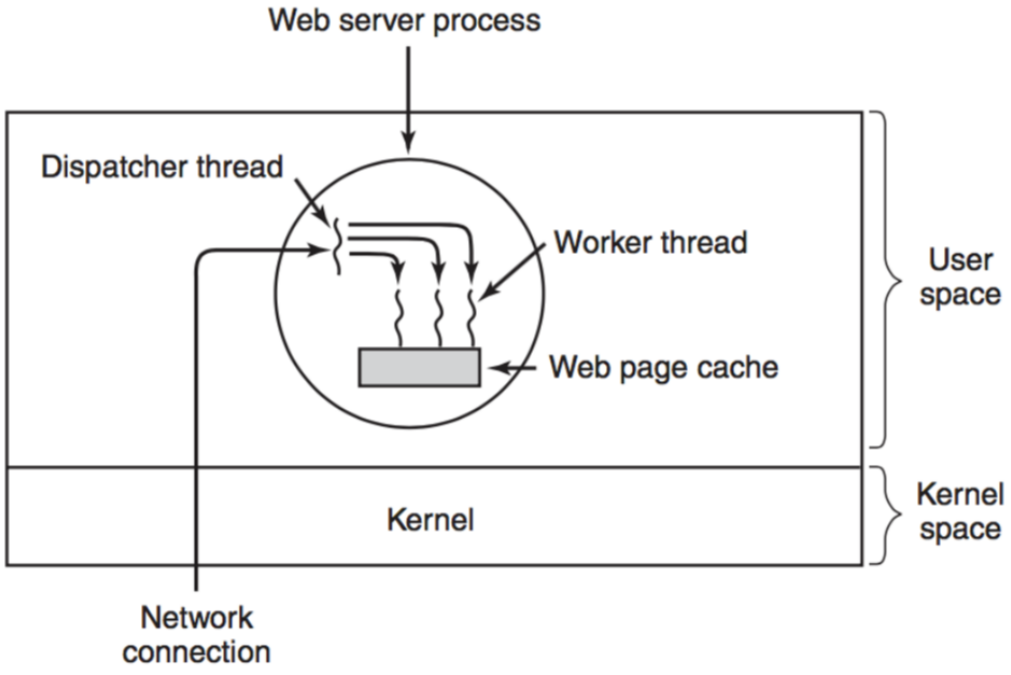
## ¿Para qué threads?

Entonces, ¿crear threads o procesos?

- Creación/destrucción de threads es sustancialmente más liviana que un proceso

# Thread

## Multihreaded webserver



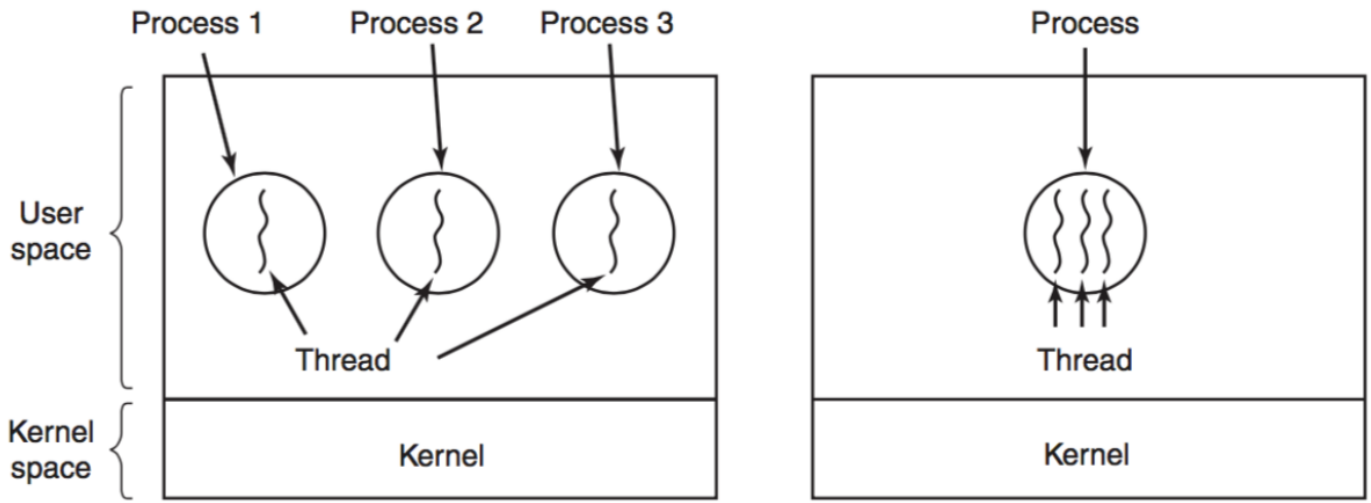

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

```
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);
}
```

# Modelos de Threads

Threads surgen de separar el concepto de recursos y ejecución

- Procesos agrupan recursos (y ejecución)
- Threads son solamente unidades de ejecución: lightweight processes



# Modelos de Threads

## Threads comparten

- Espacio de memoria
- Variables globales (sección data)
- Archivos abiertos
- Procesos Hijo
- Alarmas pendiente
- Señales y manejadores de señales
- Información de accountability

# Modelos de Threads

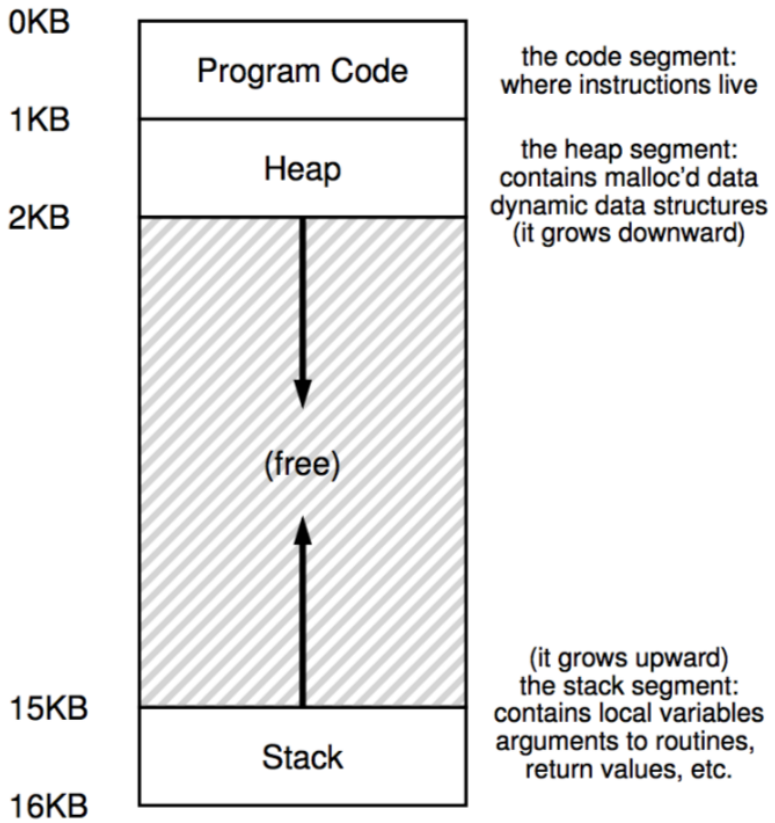
## Threads NO comparten

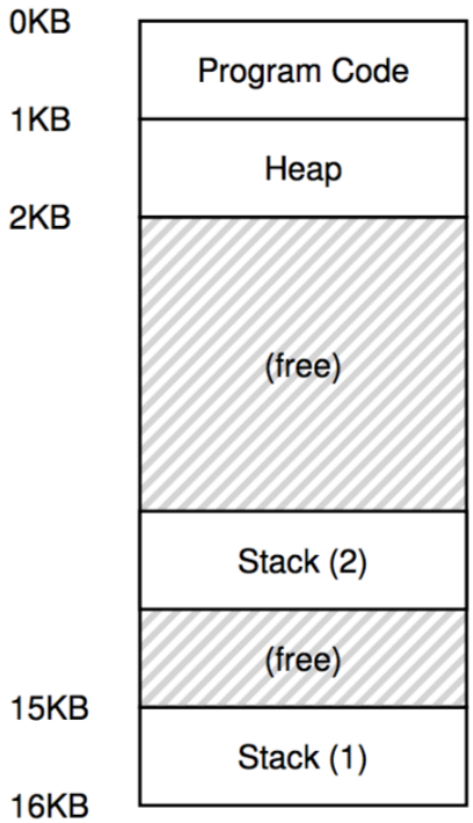
- Stack
- Registros (incluyendo PC)
- Estado

Información privada tiene que ver con ejecución. Información compartida tiene que ver con recursos

# Modelos de Threads

## ¿Cómo se mantienen múltiples stacks?



# Modelos de Threads

## Funciones típicas de threads

- thread_create()
- thread_exit()
- thread_join()
- thread_yield()

# Modelos de Threads

## Funciones típicas de threads

### POSIX THREADS

Estándar IEEE 1003.1c. Librería pthreads

| Thread call | Description |
|---|---|
| Pthread_create | Create a new thread |
| Pthread_exit | Terminate the calling thread |
| Pthread_join | Wait for a specific thread to exit |
| Pthread_yield | Release the CPU to let another thread run |
| Pthread_attr_init | Create and initialize a thread's attribute structure |
| Pthread_attr_destroy | Remove a thread's attribute structure |

- Ejemplo: POSIX Threads en acción
- Ejemplo: POSIX Threads compartiendo memoria

# Implementación de threads

Algunos aspectos no son obvios al diseñar un sistema de multithreading

- Semántica de fork(), exec
- Señales: ¿quién las recibe?
- Bloqueos: ¿quién se bloquea?

# Implementación de threads

Dos opciones

- User space
- Kernel space
- Hybrid (ok, 3 opciones)

# Implementación de threads

## User Space threads

Implementados y manejados por una biblioteca de usuario
**Kernel** no ve threads. Solo ve procesos single threaded

**Lo Bueno:**

- Portable. Implementable en cualquier S.O.
- Context switch y scheduling más rápido.
- Mejor escalabilidad (usa memoria del proceso)

**Lo Malo:**

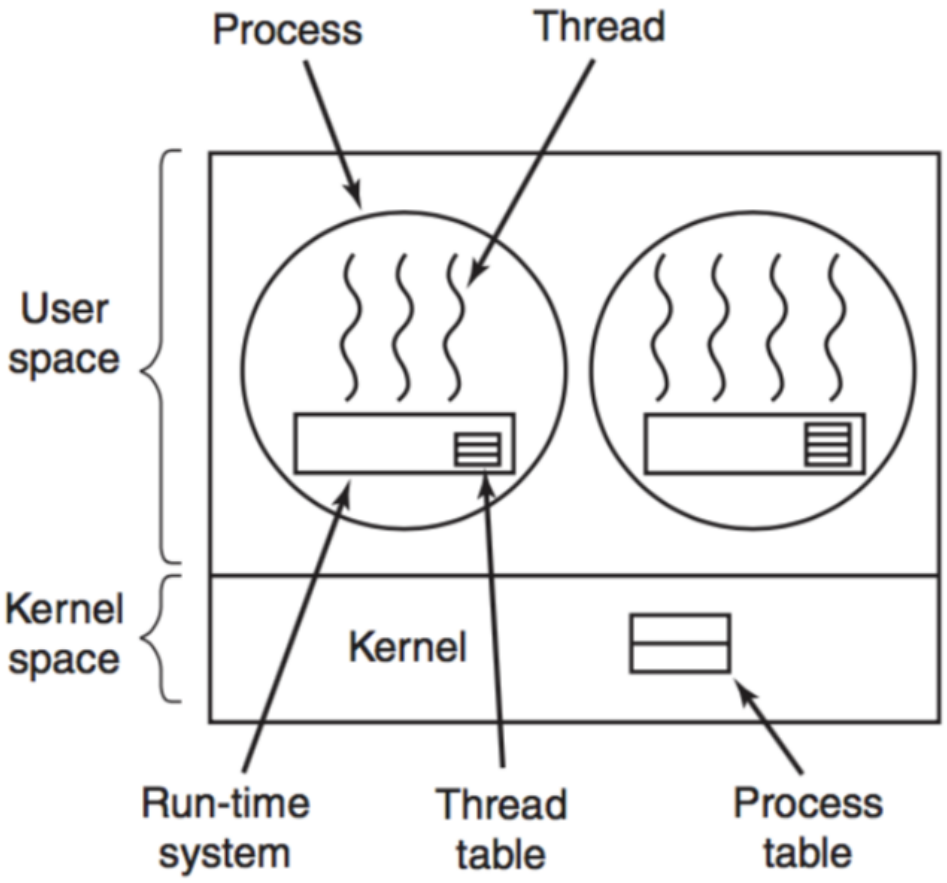
- Llamadas a syscalls pueden bloquear el proceso
- Requiere scheduling cooperativo (no hay timers)

Piensen en su uso en multithreaded web server

Implementaciones usan variante no-bloqueantes de algunas syscalls

# Implementación de threads

## User Space Threads



# Implementación de threads

## Kernel Space Threads

Implementados y manejados nativamente por estructuras del kernel

No necesita bibliotecas de usuario

Syscall manejados por el S.O.

- Lo bueno:
  - No requiere bibliotecas. Soporte nativo.
  - Syscalls no bloquean el proceso.
- Lo malo:
  - Syscalls más costosas. Requiere llamadas al kernel.
  - Escalabilidad limitada por el S.O.
  - Semántica de fork() no es clara.
  - Semántica de signals no es clara.

Implementaciones evitan crear y destruir threads. Kernel utiliza thread pools y reutiliza estructuras.

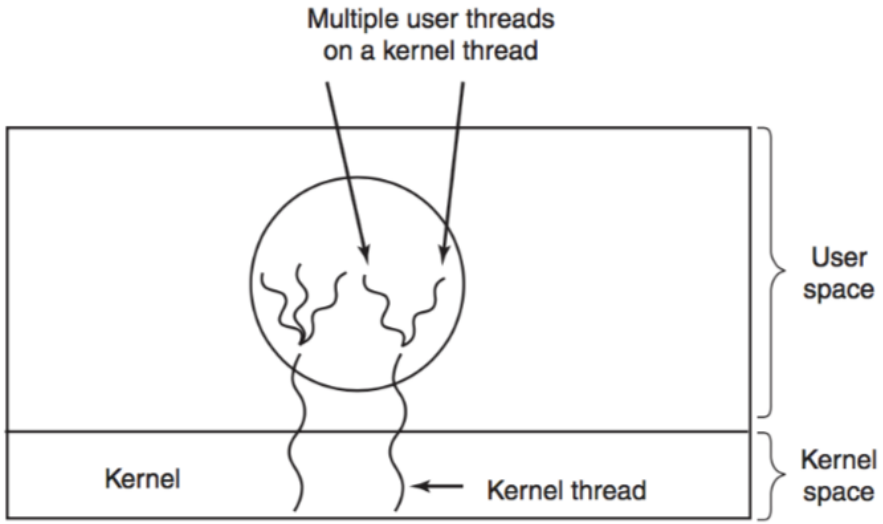
# Implementación de threads

## Implementaciones Híbridas

- Kernel threads multiplexados entre user threads
  - Kernel solo ve kernel threads
- Varios user threads asignados a un kernel thread
  - Más livianos de crear
  - Syscalls no bloquean el proceso.
  - Escalable como los user threads

# Implementación de threads

## Implementaciones Híbridas



# Material Adicional

- Video explicativo Threads Link Video
- Video explicativo Implementación Threads Link Video

**Próximo contenido a revisar será Sincronización**