# Sistemas Operativos 72.11

## Threads

ITBA
Instituto Tecnológico de Buenos Aires

# Threads

- Tradicionalmente un proceso tiene su espacio de direcciones y un único hilo (thread) de ejecución.

- Existen casos en los que es muy útil tener múltiples hilos en un mismo espacio de direcciones corriendo de forma pseudo paralela como si fueran procesos separados, salvo por el espacio de direcciones

# Threads
## Justificación

¿Para qué queremos un proceso dentro de un proceso?

- Muchas actividades simultáneas, algunas bloqueantes
  - Desglosar la solución en hilos secuenciales
    - Aumenta el uso del CPU
    - Simplifica la programación
    - Aprovecha arquitecturas con múltiples CPUs

- La misma noción de modelo de procesos
  - Abstraer detalles y pensar en procesos secuenciales
  - Con el agregado de que comparten un espacio de direcciones

- Son más “baratos”
  - Creación y destrucción
  - Hasta 10-100 veces más rápido que un proceso

# Threads

## Ejemplo: Procesador de texto

- Corrector ortográfico
- Auto guardado
- Procesar input
- Visualizar contenido

¿Cómo sería con procesos separados?

# Threads

## Ejemplo: Web server

- Atender conexiones
- Enviar páginas en cache
- Buscar páginas en disco (bloqueante)

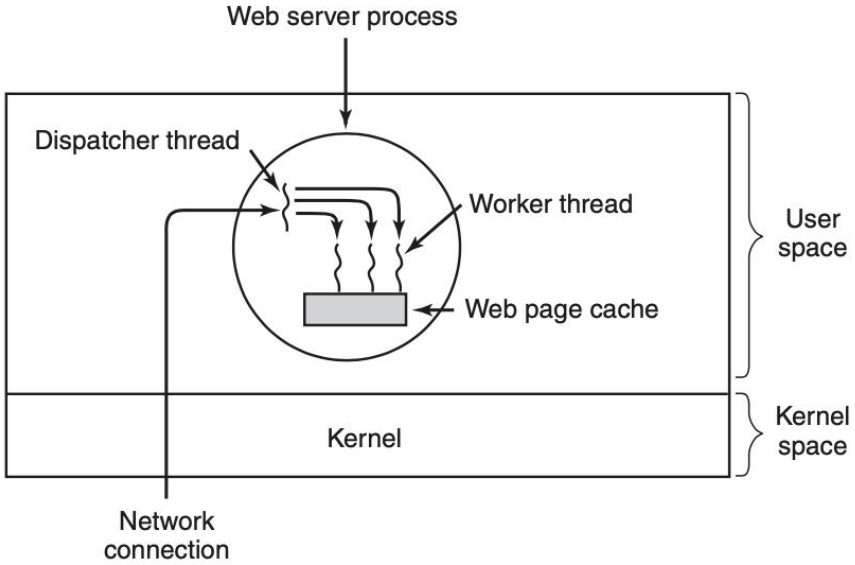



**Figure 2-8.** A multithreaded Web server.

# Threads

## Ejemplo: Web server

```
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)

**Figure 2-9.** A rough outline of the code for Fig. 2-8. (a) Dispatcher thread. (b) Worker thread.

# Threads

## Ejemplo: Web server

¿Cómo sería el servidor sin threads?

¿Y si no tengo threads y lo quiero más eficiente?

- Syscalls no bloqueantes
- Almacenar el estado de cada pedido para retomarlo cuando llegue la página del disco
- Se pierde la noción de computación secuencial
- Estamos simulando threads “the hard way”

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

**Figure 2-10.** Three ways to construct a server.

- Las syscalls bloqueantes facilitan la programación
- El paralelismo aumenta el rendimiento

# Threads

## Modelo de threads

- El modelo de procesos está basado en 2 conceptos independientes
  - Agrupación de recursos
  - Ejecución

| **Process management** | **Memory management** | **File management** |
|---|---|---|
| Registers | Pointer to text segment info | Root directory |
| Program counter | Pointer to data segment info | Working directory |
| Program status word | Pointer to stack segment info | 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 | | |

**Figure 2-4.** Some of the fields of a typical process-table entry.

# Threads
## Modelo de threads

- Proceso vs. thread
  - Un thread está contenido en un proceso, pero…
    - El proceso agrupa recursos
    - El thread es una entidad programada (scheduled) para ejecución en el CPU
  - Los threads permiten múltiples ejecuciones en el mismo conjunto de recursos

- Tener múltiples threads en un proceso es análogo a tener múltiples procesos en una PC
  - En un caso se comparte el espacio de direcciones, files, señales, etc
  - En el otro se comparte la memoria, dispositivos de entrada y salida, etc

- Thread -> light weight process

# Threads

## Modelo de threads

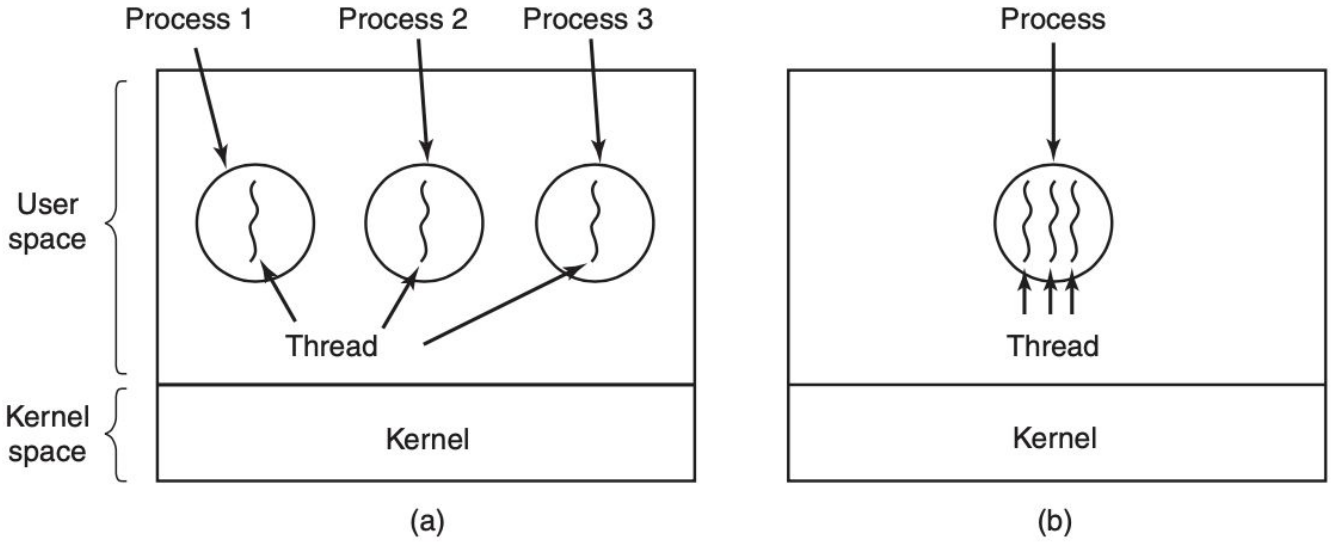



**Figure 2-11.** (a) Three processes each with one thread. (b) One process with three threads.

# Threads

## Modelo de threads

| 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 | |

**Figure 2-12.** The first column lists some items shared by all threads in a process. The second one lists some items private to each thread.

# Threads

## Modelo de threads

- Analogía misma historia narrada desde diferentes perspectivas

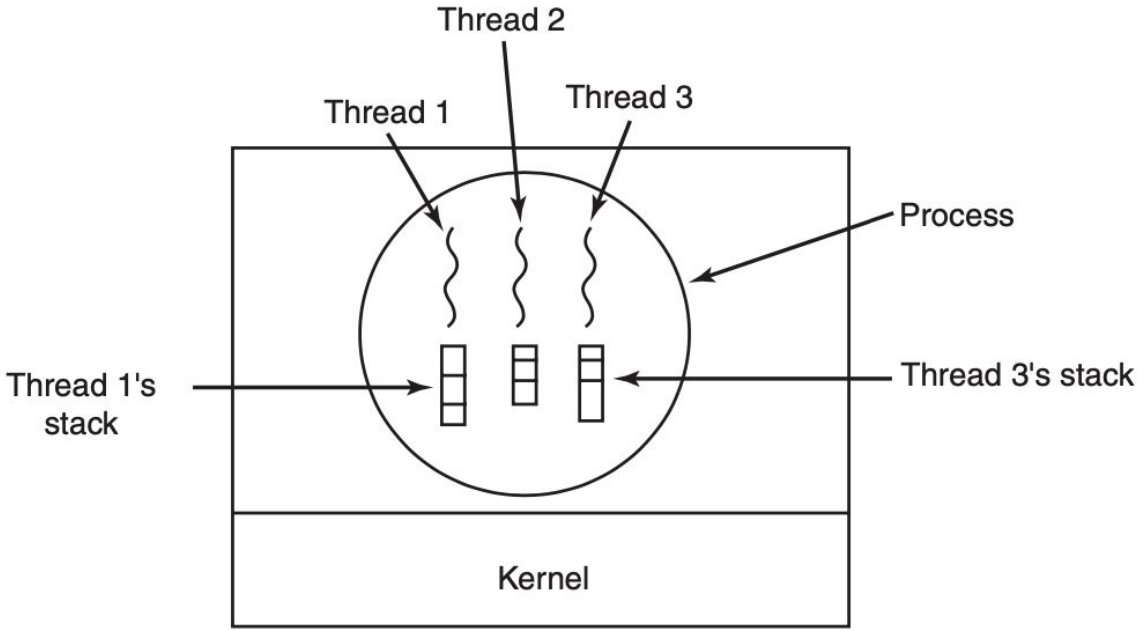



**Figure 2-13.** Each thread has its own stack.

# Threads

## Modelo de threads

- Los threads comparten el espacio de direcciones, por tanto, un thread podría modificar información mientras otro la está leyendo. Esto tiene un nombre, ¿cuál es?
- No existe protección provista por el kernel ante esta situación ¿por qué?

Julia Evans
@b0rk

# threads

drawings.jvns.ca

Threads let a process do many different things at the same time

process:

thread 1: I'm calculating ten million digits of π! so fun!

thread 2: I'm finding a REALLY BIG prime number!

threads in the same process share memory

thread 1: I'll write some digits of π to 0x129420 in memory

thread 2: uh oh that's where I was putting my prime numbers

and they share code

calculate_pi

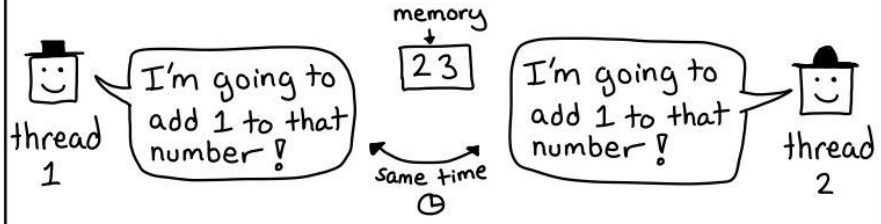
find_big_prime_number

but each thread has its own stack and they can be run by different CPUs at the same time

CPU 1 — π thread

CPU 2 — primes thread

sharing memory can cause problems (race conditions!)




RESULT: 24 ← WRONG. Should be 25!

Why use threads instead of starting a new process?

→ a thread takes less time to create

→ sharing data between threads is very easy. But it's also easier to make mistakes with threads

thread 1: you weren't supposed to CHANGE that data!

# Threads

## POSIX: API

| 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 |

**Figure 2-14.** Some of the Pthreads function calls.

# Threads

## POSIX: Ejemplo

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
        /* This function prints the thread's identifier and then exits. */
        printf("Hello World. Greetings from thread %d\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        /* The main program creates 10 threads and then exits. */
        pthread_t threads[NUMBER_OF_THREADS];
        int status, i;

        for(i=0; i < NUMBER_OF_THREADS; i++) {
                printf("Main here. Creating thread %d\n", i);
                status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

                if (status != 0) {
                        printf("Oops. pthread_create returned error code %d\n", status);
                        exit(-1);
                }
        }
        exit(NULL);
}
```

**Figure 2-15.** An example program using threads.

# Threads

## Implementación en espacio de usuario

- El kernel desconoce de su existencia
- Desde la perspectiva del kernel son procesos con un único thread
- Provee soporte para threads en caso de que el kernel no los provea
- Se implementan como una librería

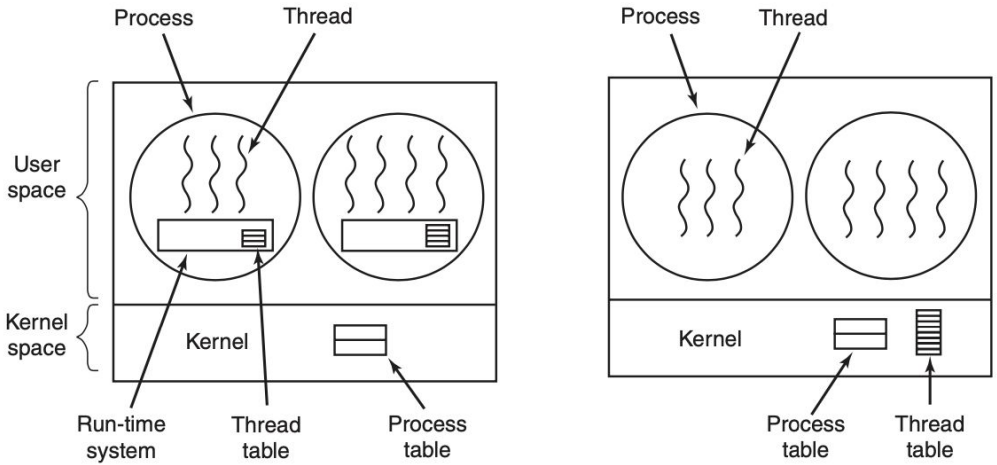



**Figure 2-16.** (a) A user-level threads package. (b) A threads package managed by the kernel.

# Threads

## Implementación en espacio de usuario

- Cada proceso necesita su tabla de threads privada -> análoga a la tabla de procesos del kernel

- Si un thread se bloquea localmente (esperando por otro thread del mismo proceso) se realiza el switch en espacio de usuario
  - Es un orden (o más) de magnitud más rápido que el switch usual con interrupciones y la intervención del kernel.
  - No es necesario flushear la cache
- Cada proceso puede tener su propio algoritmo de scheduling de threads

Todo muy lindo, pero ¿qué pasa con syscalls bloqueantes?

# Threads

## Implementación en espacio de usuario: desventajas

Syscall bloqueante

- ¿Que se bloquee todo el proceso y listo?
  - Cambiar las syscalls a no boqueantes -> modificar el kernel
  - Select(2) -> polling

Page faults

- Un thread causa un page fault -> el kernel bloquea el proceso entero hasta que llega la página

Inanición

- Un thread puede correr indefinidamente hasta que voluntariamente libere el CPU
  - Se puede solicitar una señal -> ineficiente

Analogía timer (externo) - kernel / señal - librería de threads

Uso de threads

- Separar hilos que fundamentalmente se bloquean

# Threads

## Implementación en espacio de usuario: desventajas

For applications that are essentially entirely CPU bound and rarely block, what is the point of having threads at all? No one would seriously propose computing the first *n* prime numbers or playing chess using threads because there is nothing to be gained by doing it that way.

# Threads

## Implementación en espacio de kernel

- No es necesario el run-time system ni tabla de threads (en espacio de usuario)
- Un thread se bloquea como es usual y el kernel elige otro thread (u otro proceso)
- Debido al mayor costo, se pueden reutilizar los threads

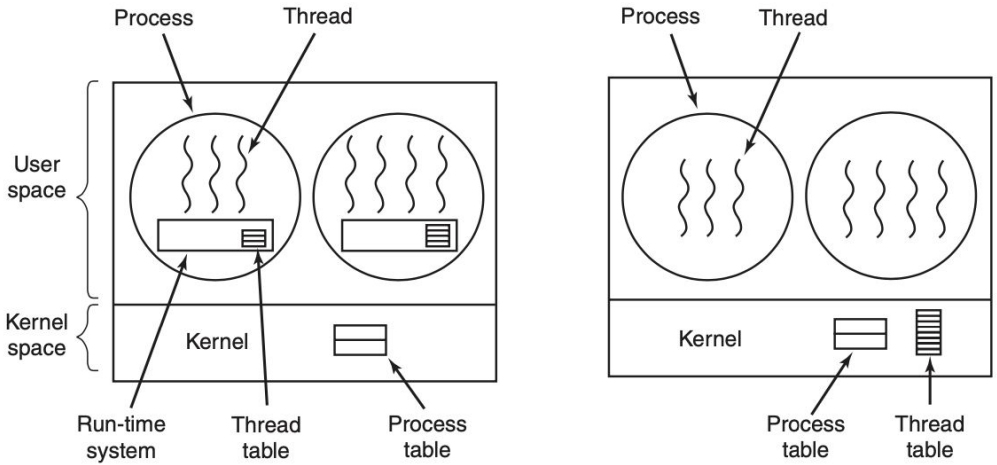



**Figure 2-16.** (a) A user-level threads package. (b) A threads package managed by the kernel.

# Threads

## Implementación en espacio de kernel

- ¿Qué pasa cuando un proceso con threads ejecuta fork?
- ¿Qué pasa con las señales? Por defecto son por proceso.

# Threads
## Implementación híbrida

- The kernel solo es consciente del kernel-thread
- Por sobre cada kernel-thread puede haber múltiples user-threads

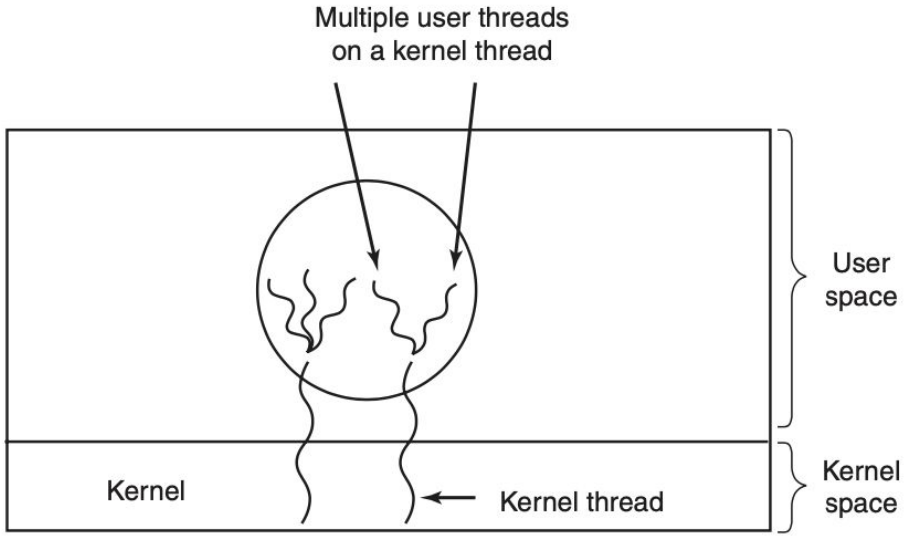
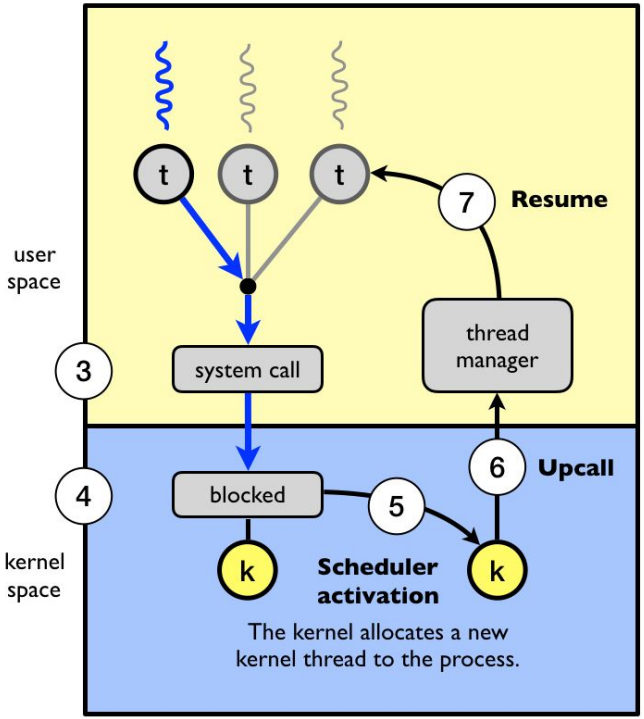



**Figure 2-17.** Multiplexing user-level threads onto kernel-level threads.

# Threads

## Scheduler activations




- Threads en espacio de usuario con la funcionalidad de aquellos en espacio de kernel.
- Al bloquearse, se crea un nuevo thread y se notifica al run-time system (thread manager) - upcall - signal
- Viola la estructura de un sistema de capas

https://www.it.uu.se/education/course/homepage/os/vt20/module-4/implementing-threads/

# Ejercicio 1

Desarrolle un programa que mediante el uso de threads **threads(7)** tenga una condición de carrera. Por ejemplo, varios threads incrementando una variable sin mecanismos de sincronización. Luego resuelva la condición de carrera mediante el uso de semáforos.

# Glosario

-