06. Threads & Concurrency

CS 4352 Operating Systems

Adopted from slides of Drs. Silberschatz, Huang, & Bryant

Processes

- Recall that a process includes many things
  - An address space (defining all the code and data pages)
  - OS resources (e.g., open files) and accounting information
  - Execution state (PC, SP, regs, etc.)
- Creating a new process is costly
  - Because of all of the data structures that must be allocated and initialized
- Communicating between processes is also costly
  - Because most communication goes through the OS
    - Overhead of system calls and copying data

Concurrent Programs

- A web server (or any parallel program)...
  - Forks off copies of itself to handle multiple simultaneous requests
- To execute these programs we need to
  - Create several processes that execute in parallel
  - Cause each to map to the same address space to share data
  - They are all part of the same computation
    - Have the OS schedule these processes
- This situation is very inefficient
  - Space: PCB, page tables, etc.
  - Time: create data structures, fork and copy addr space, etc.

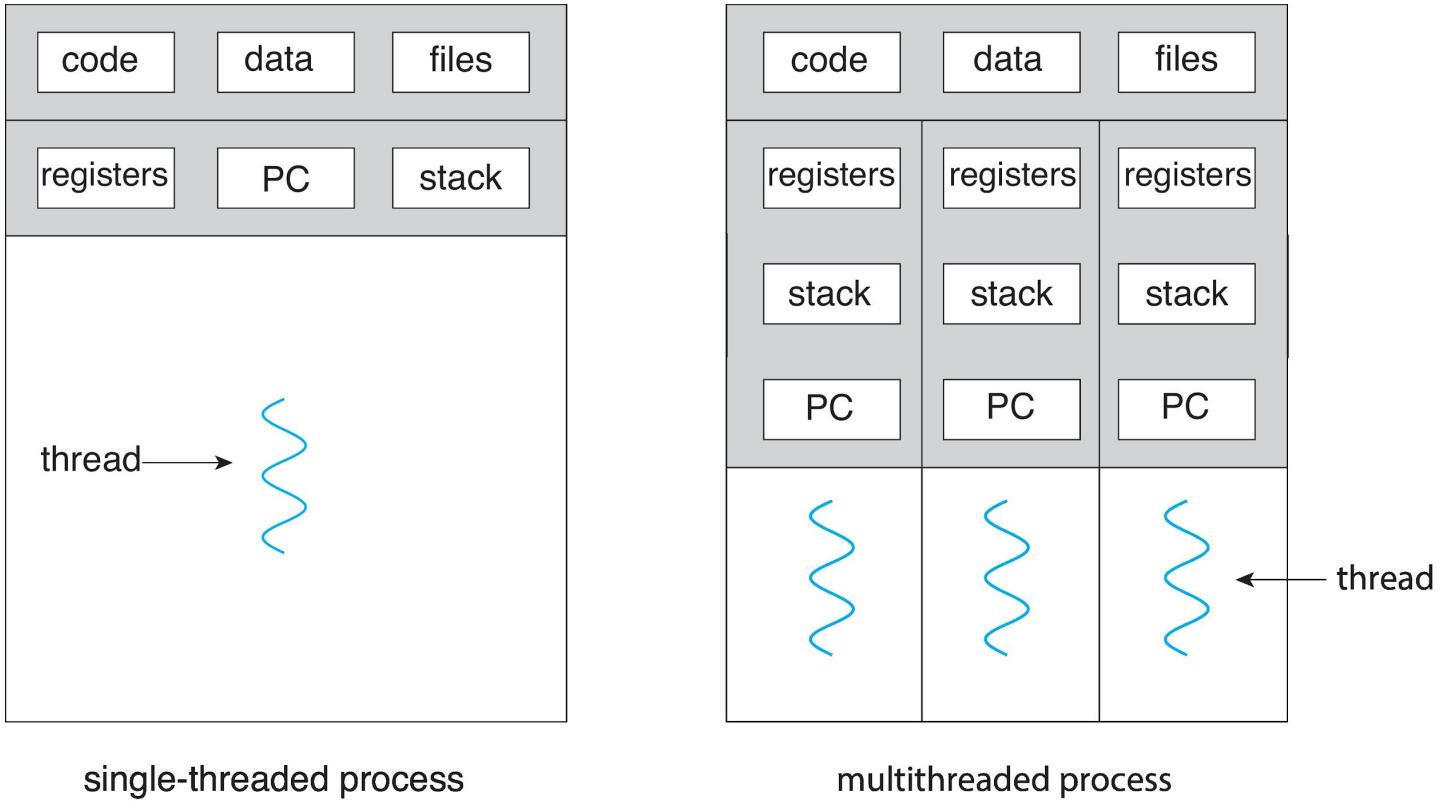
Rethinking Processes

- What is similar in these cooperating processes?
  - They all share the same code and data (address space)
  - They all share the same privileges
  - They all share the same resources (files, sockets, etc.)
- What don’t they share?
  - Each has its own execution state: PC, SP, and registers
- Key idea: Why don’t we separate the concept of a process from its execution state?
  - Process: address space, privileges, resources, etc.
  - Execution state: PC, SP, registers
- Execution state also called thread of control, or thread

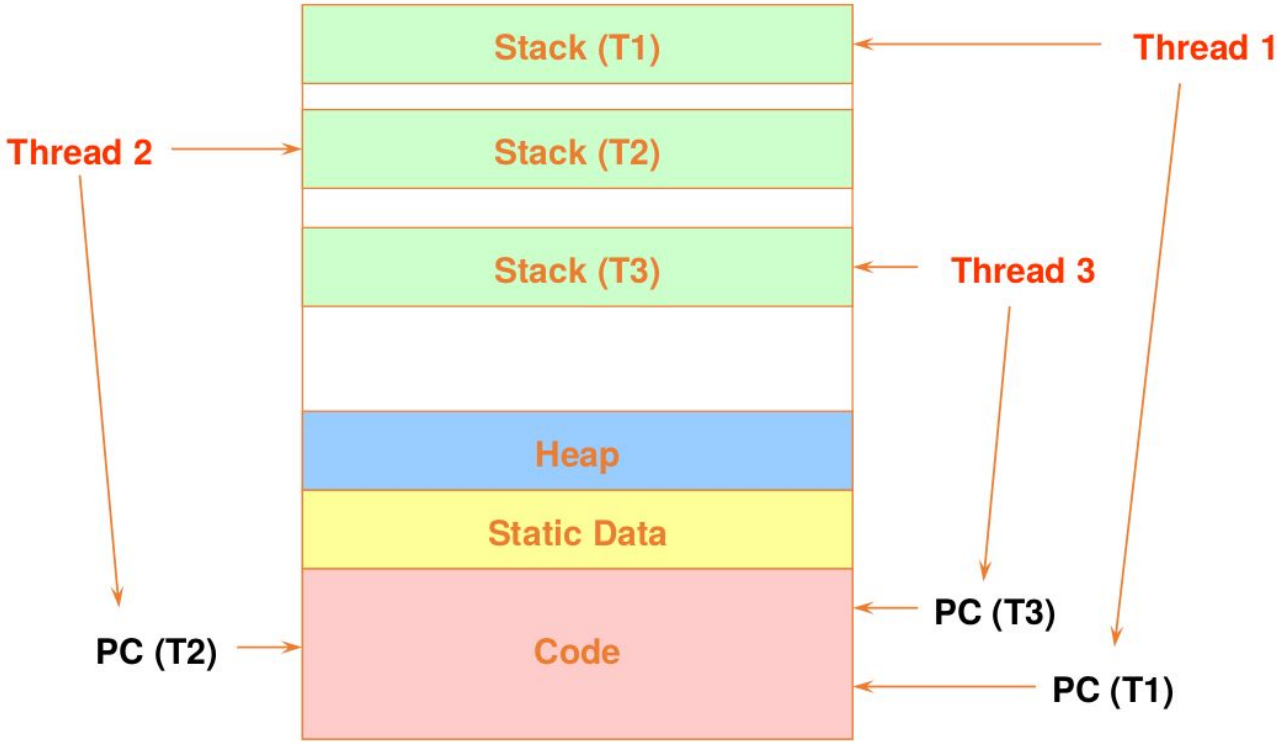
Threads

- Modern OSes separate the concepts of processes and threads
  - The thread defines a sequential execution stream within a process (PC, SP, registers)
  - The process defines the address space and general process attributes (everything but threads of execution)
- A thread is bound to a single process
  - Processes, however, can have **multiple threads**
- Most modern applications are multithreaded
  - Multiple tasks with the application can be implemented by separate threads
    - Update display
    - Fetch data
    - Spell checking
    - Answer a network request
- OS kernels are actually also multithreaded

Single and Multithreaded Processes



Threads in a Process



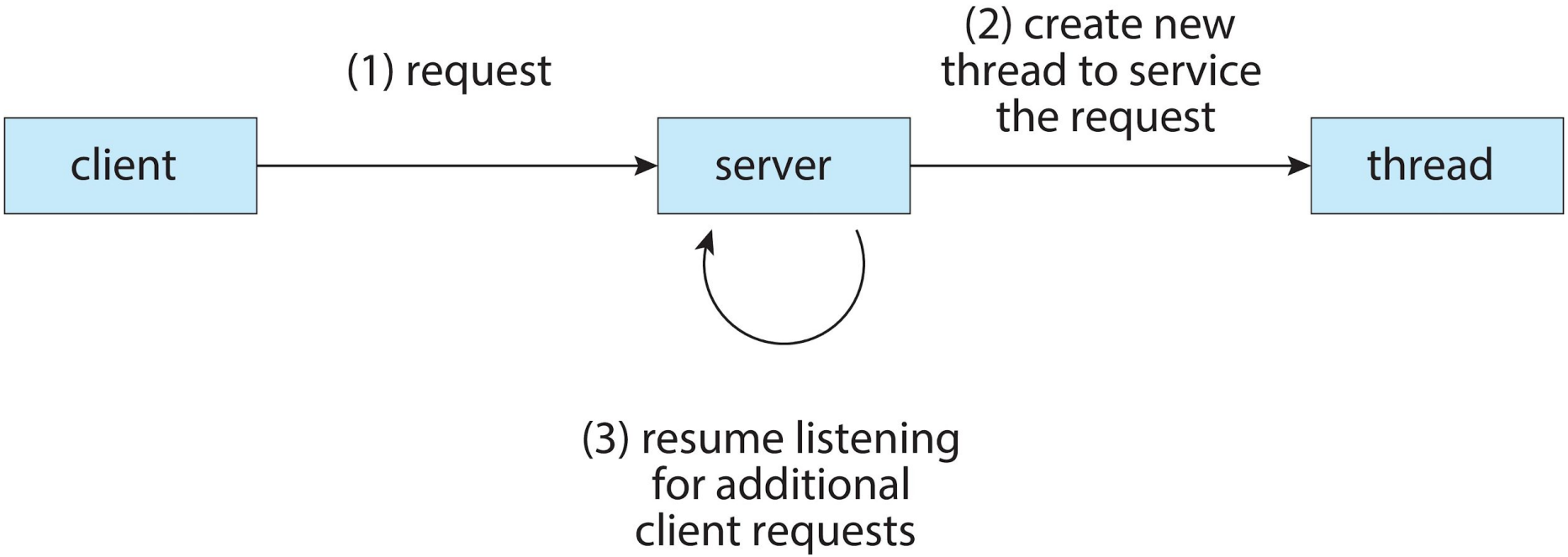
Web Server Example

- Using fork() to create new processes to handle requests is an overkill
- Instead, we can create a new thread for each request

```
web_server() {
    while (1) {
        int sock = accept();
        thread_fork(handle_request, sock);
    }
}

handle_request(int sock) {
    Process request
    close(sock);
}
```

Multithreaded Server Architecture



Benefits

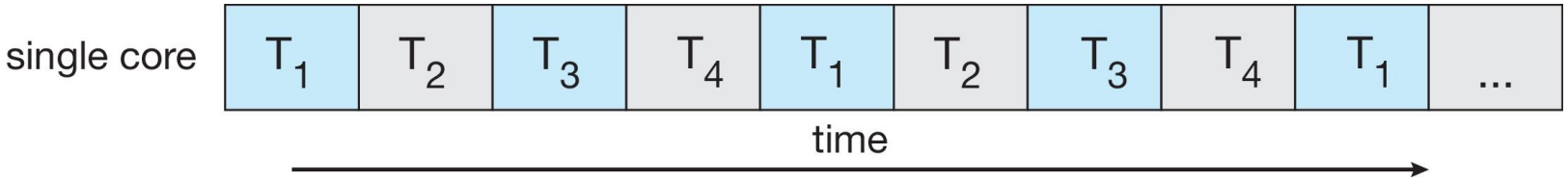
- Responsiveness – may allow continued execution if part of process is blocked
- Resource Sharing – threads share resources of process
- Economy – cheaper than process creation, thread switching lower overhead than context switching
- Scalability – process can take advantage of multicore architectures

Multicore Programming

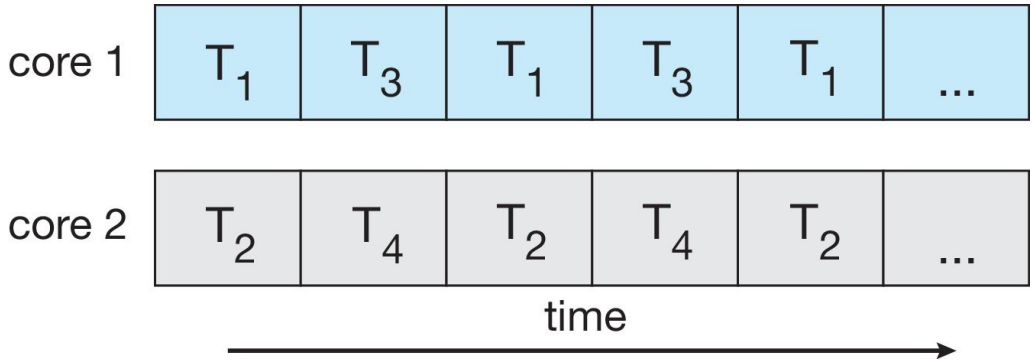
- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - Dividing activities
  - Balance
  - Data splitting
  - Data dependency
  - Testing and debugging
- **Parallelism** implies a system can perform more than one task simultaneously
- **Concurrency** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Concurrency v.s. Parallelism

- Concurrent execution on single-core system:

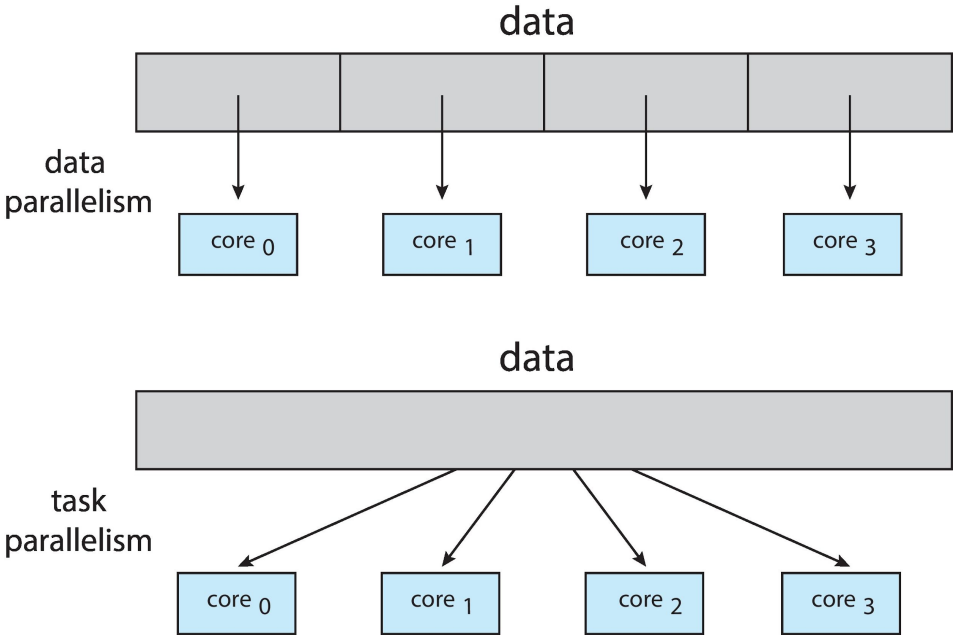


- Parallelism on a multi-core system:



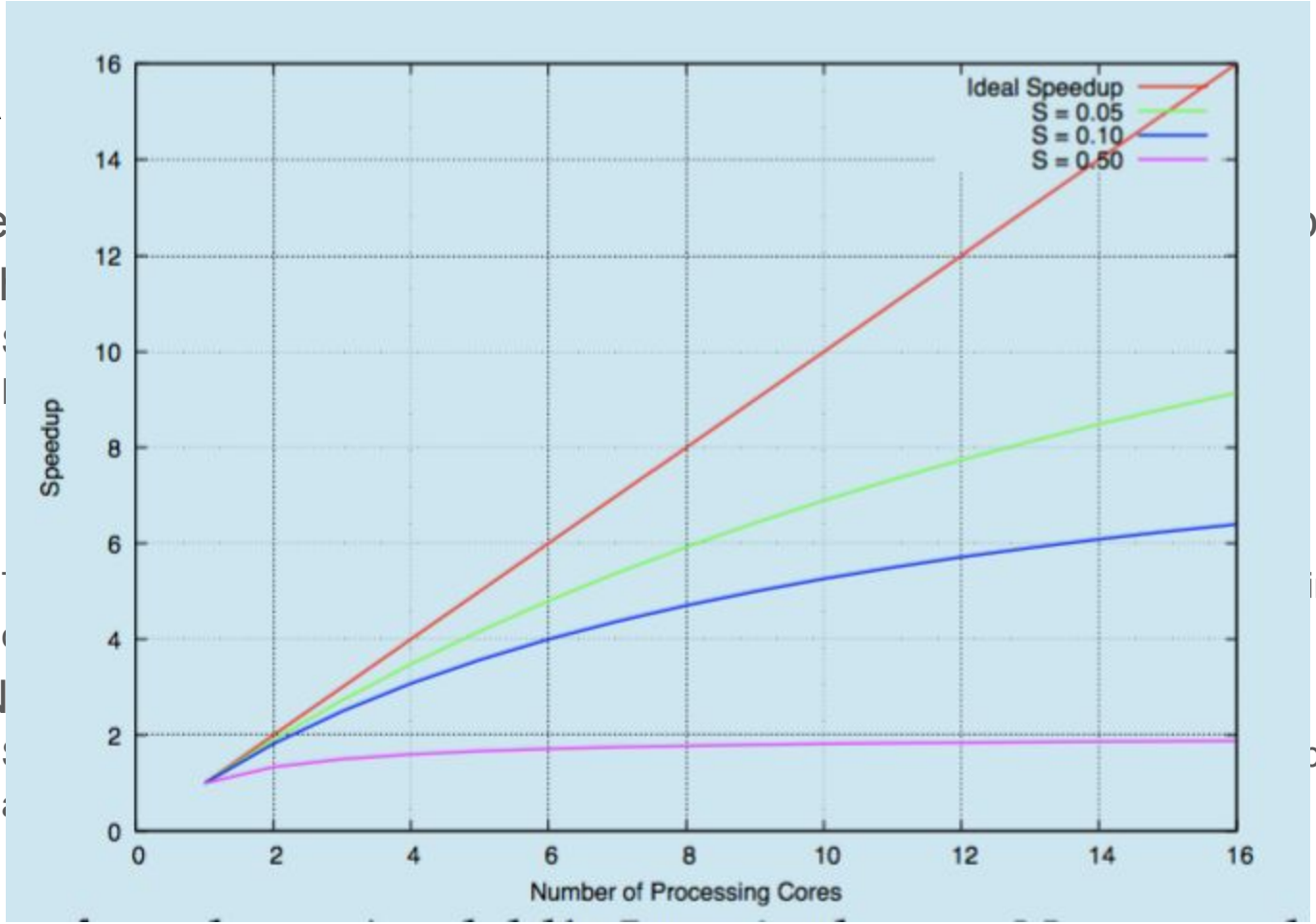
Data and Task Parallelism

- Types of parallelism
  - Data parallelism – distributes subsets of the same data across multiple cores, same operation on each
  - Task parallelism – distributes threads across cores, each thread performing unique operation



Amda

- It ide
  -
  -
- As N
  -



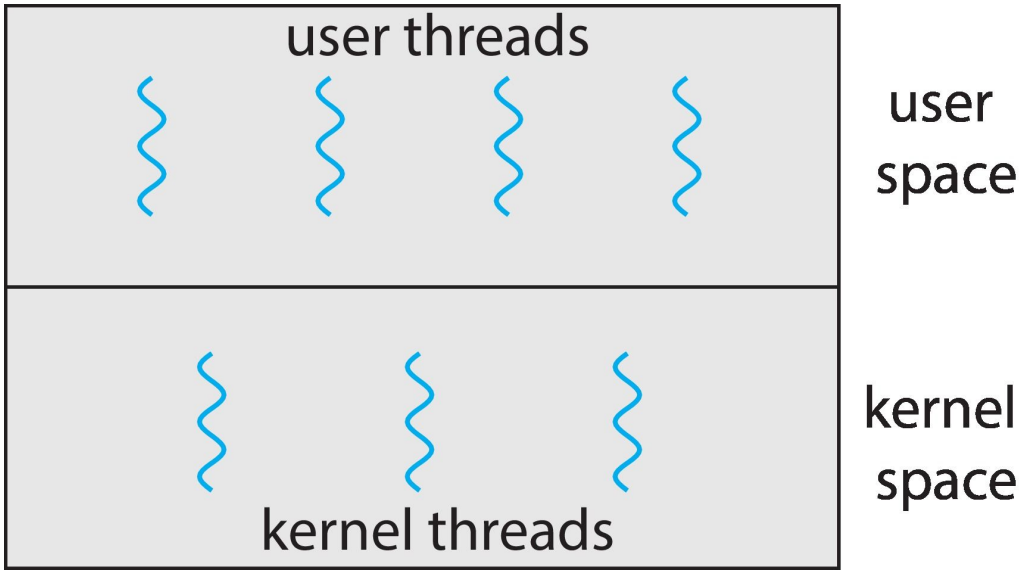
lication

in speedup

by adding

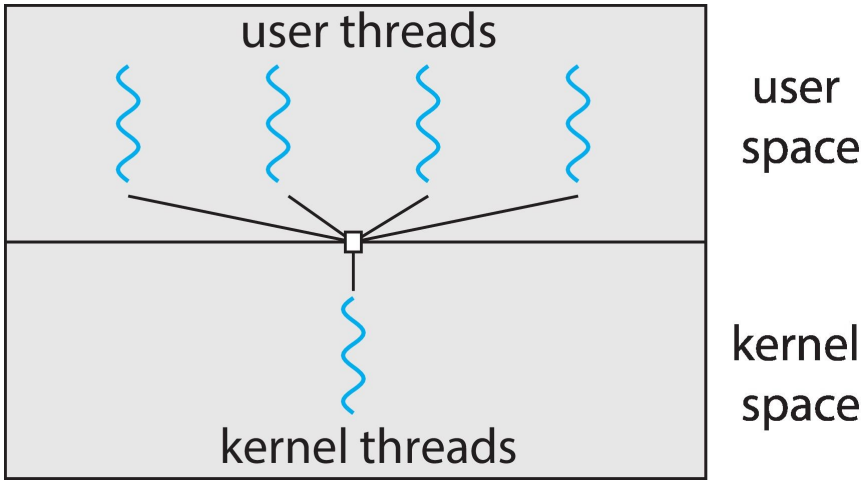
User and Kernel Threads

- User threads - management done by user-level threads library
- Kernel threads - Supported by the kernel



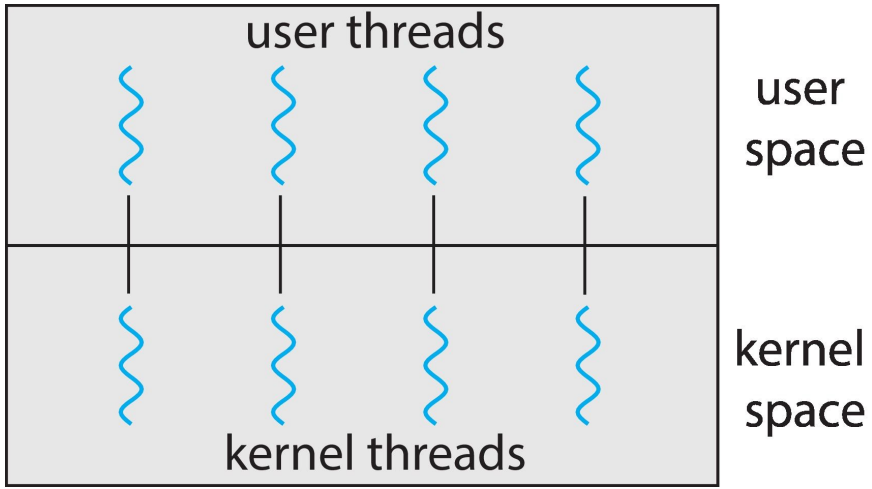
Many-to-One Multithreading Model

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time
- Few systems currently use this model



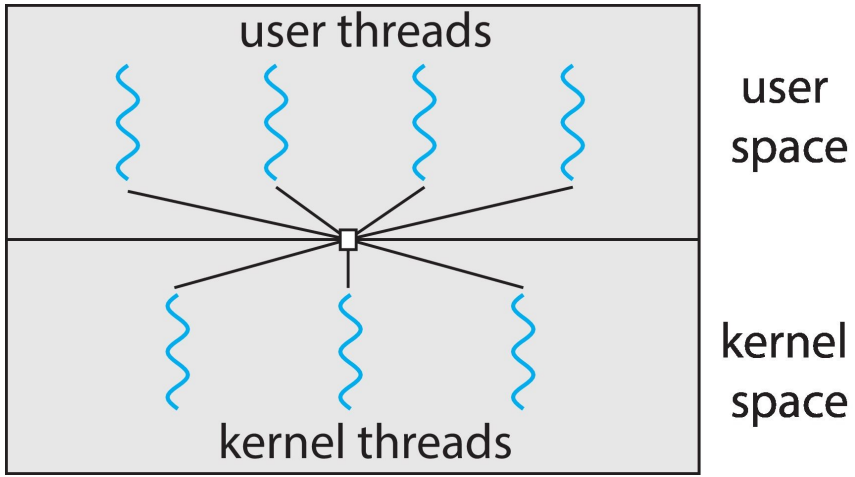
One-to-One Multithreading Model

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead



Many-to-Many Multithreading Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Windows with the ThreadFiber package
- Otherwise not very common



Thread Libraries

- Thread library provides programmer with APIs for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- **Specification**, not **implementation**
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX-like operating systems (Linux & Mac OS X)

Pthread APIs

- Basic functions:
  - pthread_create() -- create a thread
  - pthread_join() -- join (“wait for”) a terminated thread
  - pthread_exit() -- terminate the calling thread (does not cause the whole program to terminate)
- Other functions:
  - pthread_attr_init() -- specify “attributes” for the thread, e.g., stack size, is the thread detached
  - pthread_self() -- get a “handle” to a pthread
  - pthread_cancel() -- cancel a thread
  - pthread_kill() -- send a signal to a thread
  - pthread_detach() -- detach a thread: automatically free its resources when it’s done
  - pthread_equal() -- compare two threads for equality

Pthread Example

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

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   /* set the default attributes of the thread */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid, &attr, runner, argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}
```

```
/* The thread will execute in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

Pthread Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Semantics of fork() and exec()

- Does fork() duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork
- exec() usually works as normal – replace the running process including all threads

Linux Threads

- Linux refers to them as **tasks** rather than **threads**
- Thread creation is done through **clone()** system call
- clone() allows a child task to share the address space of the parent task (process)
    - Flags control behavior

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.

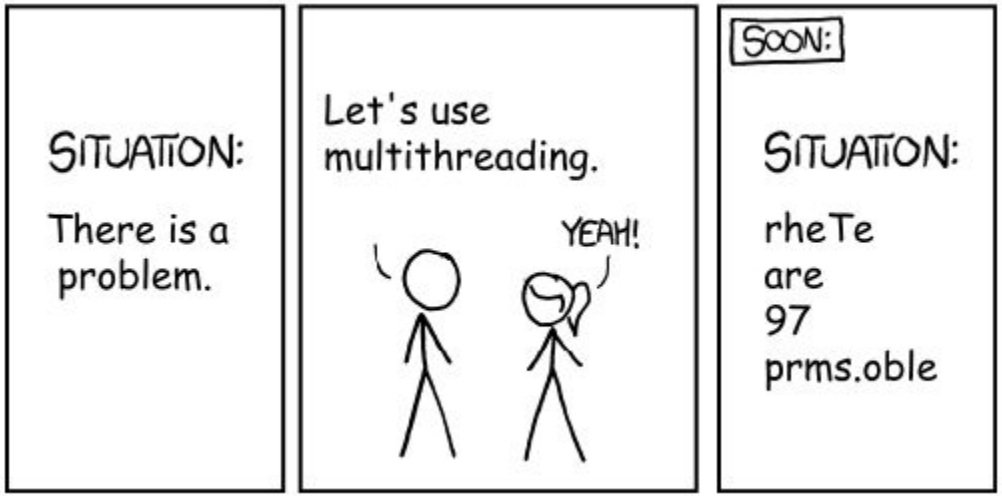
- **struct task_struct** is used for process and threads

Homework

- Read Chapters 6 & 7
- Assignment 2 is due tomorrow

Next Lecture

- We will look at synchronization



Credit: https://www.reddit.com/r/ProgrammerHumor/comments/dtiufv/multithreading_fixing_a_problem/