CS 452 Operating Systems

Threads

Dr. Denton Bobeldyk

Threads

- What is a thread?
  - Basic unit of CPU utilization
  - Comprised of
    - Thread ID
    - Program counter
    - Register set
    - Stack

Threads

- Multiple threads can run in a process and share:
  - Code section
  - Data section
  - OS resources (e.g., open files)

Threads

- Traditional (Heavyweight) process has a single thread of control
- Multithreaded process can do more than one task at a time

Threads

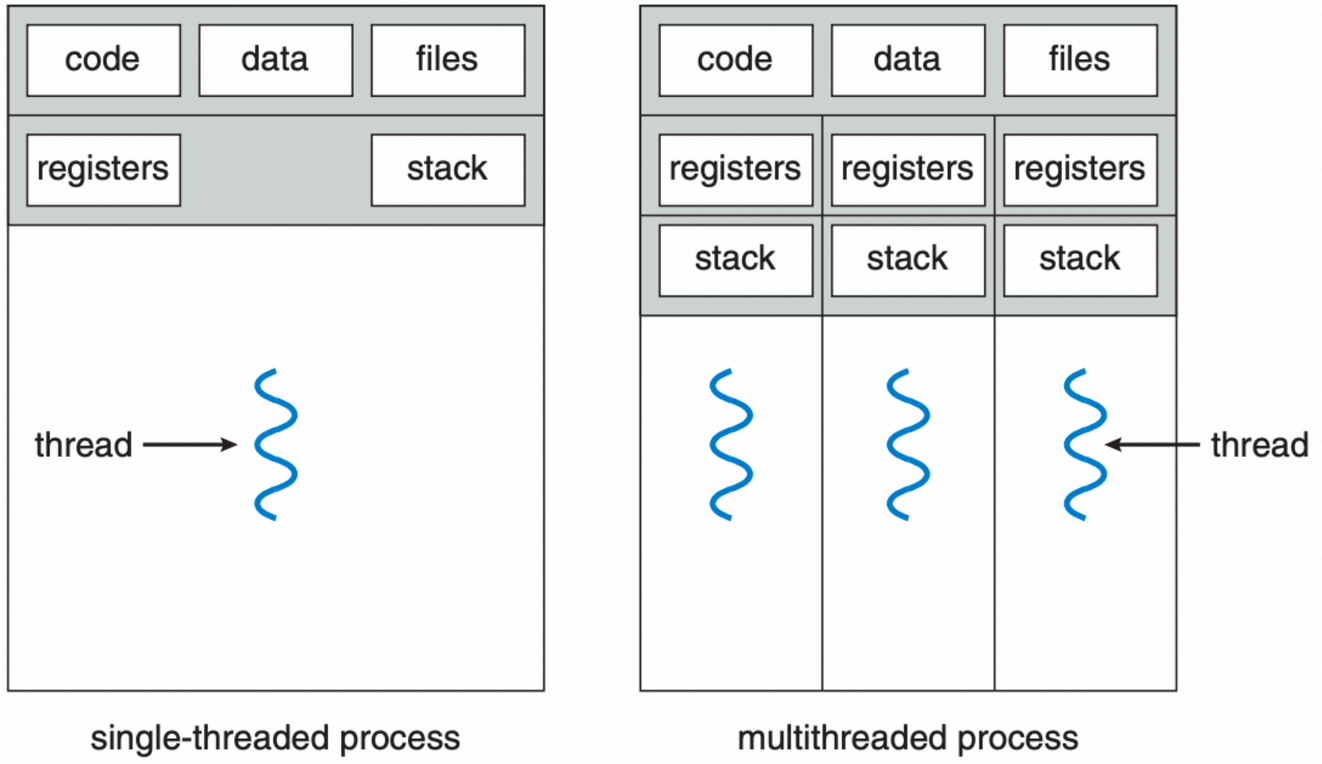


Figure 4.1 Single-threaded and multithreaded processes.

Threads - Application Example

- Web Browser with multiple threads:
  - Retrieve data from the network
  - Display images
  - Display text

Threads - Application Example

- Web server
  - Accepts client requests for images, sound, etc

Threads - Example

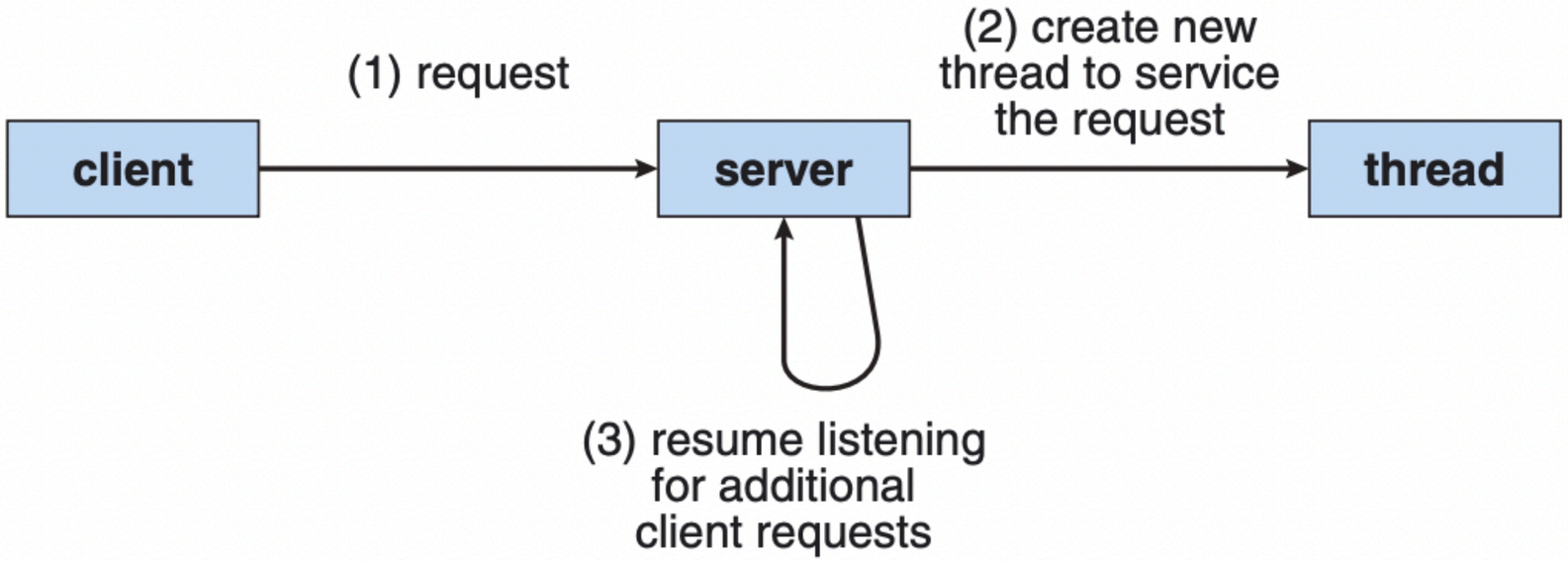


Figure 4.2 Multithreaded server architecture.

Threads vs. Processes

- Why Threads?

Threads vs. Processes

- Why Threads?

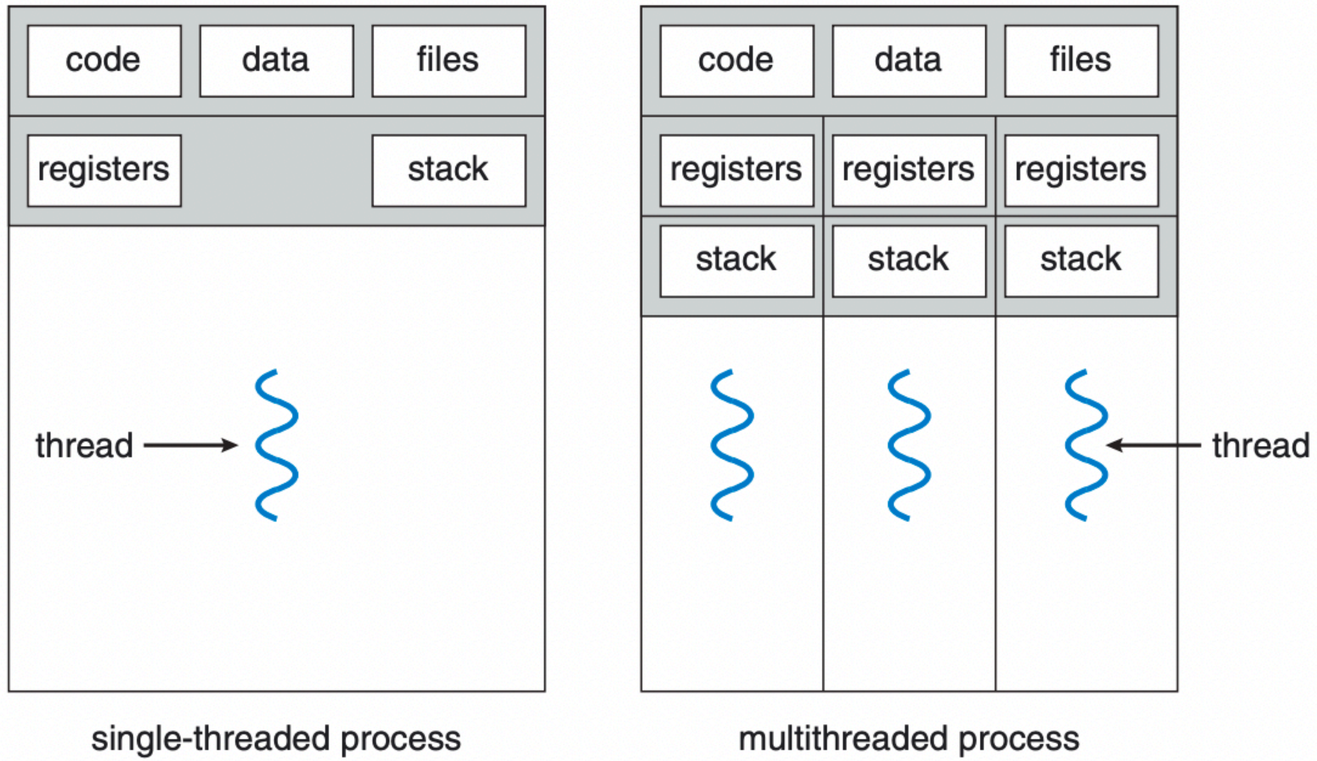


Figure 4.1 Single-threaded and multithreaded processes.

Thread Benefits

- Responsiveness
- Resource Sharing
- Economy
  - Solaris OS creating a thread is 30 times faster
  - Solaris OS context switching is 5 times faster with threads
- Scalability
  - Can run threads on multiple processors

Multicore Programming

- Parallelism - can perform more than one task simultaneously
- Concurrency - allows all tasks to make progress by rapidly switching between processes on a single CPU

Multicore Programming

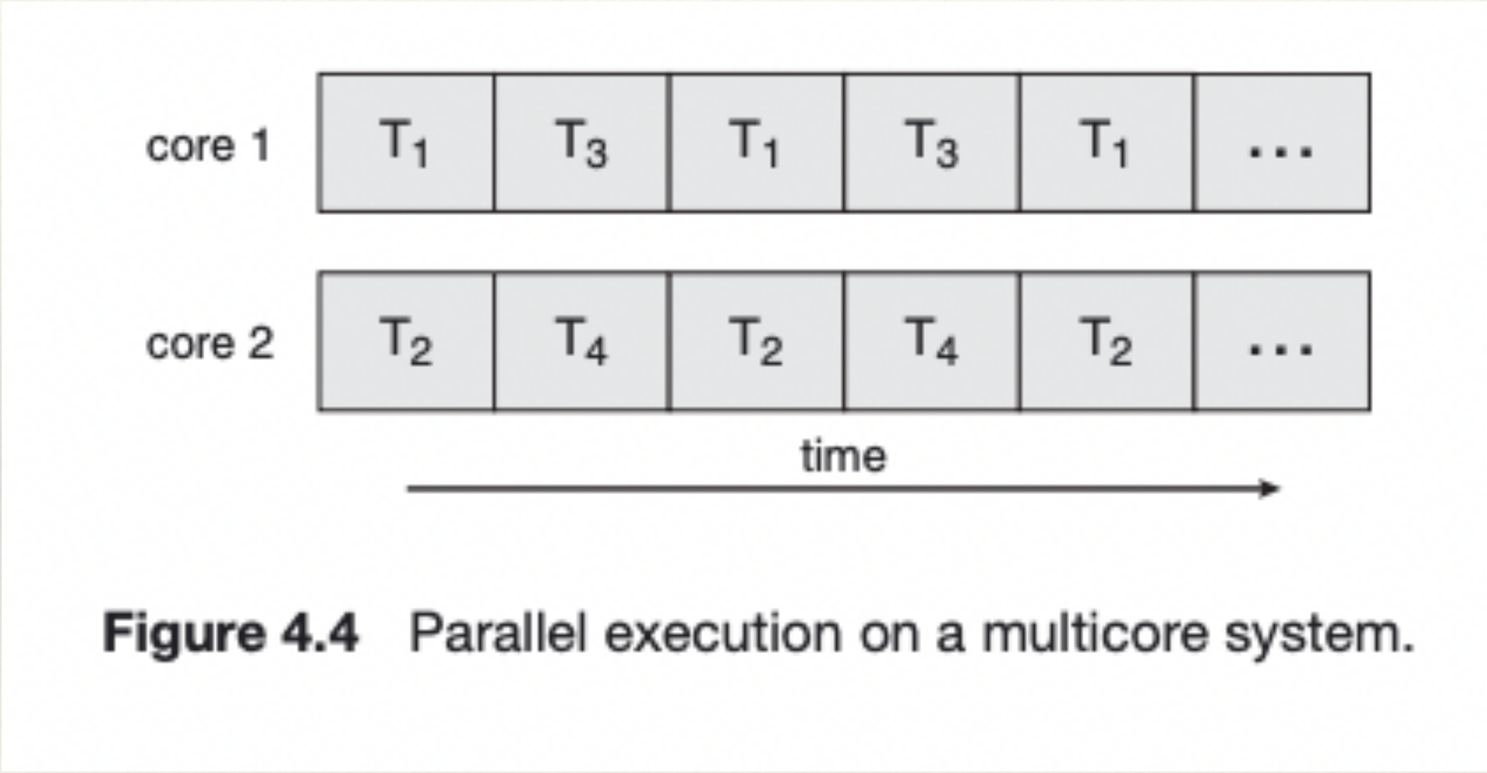


Figure 4.4 Parallel execution on a multicore system.

Multicore Programming

- Amdahl's Law (S is the percent performed Serially, N is the number of processing cores):

$$speedup <= \frac{1}{(S + \frac{(1-S)}{N})}$$

Multicore Programming

- Amdahl's Law (S is the percent performed Serially, N is the number of processing cores):

$$speedup <= \frac{1}{(S + \frac{(1-S)}{N})}$$

Modern computers with increased hardware enhancements may render this law irrelevant

Programming Challenges

1. Identifying tasks - which tasks can be run independently and thus in parallel?
2. Balance - is it worth the cost to separate some tasks?
3. Data splitting - data accessed by tasks must be divided to run on separate cores
4. Data dependency - must ensure synchronization where required
5. Testing and debugging - more difficult given the inconsistency of run-time execution

Types of Parallelism

1. Data parallelism - how to split up the data?
    1. Example: Sum an array
2. Task parallelism - split the work (tasks)

Thread Types

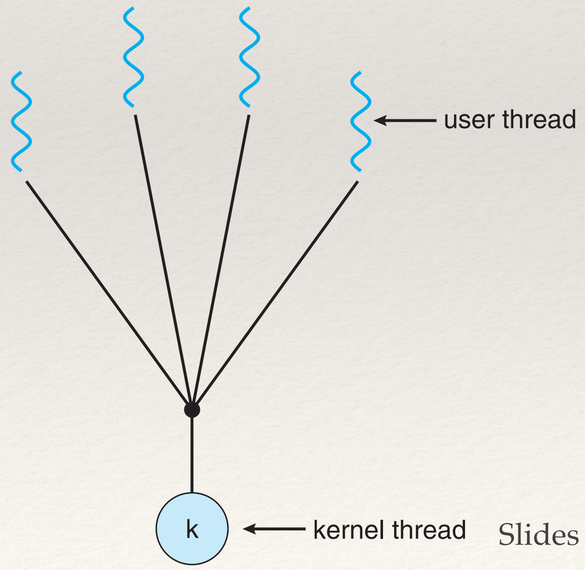
- User Threads - managed without kernel support
- Kernel Threads - managed directly by the operating system

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

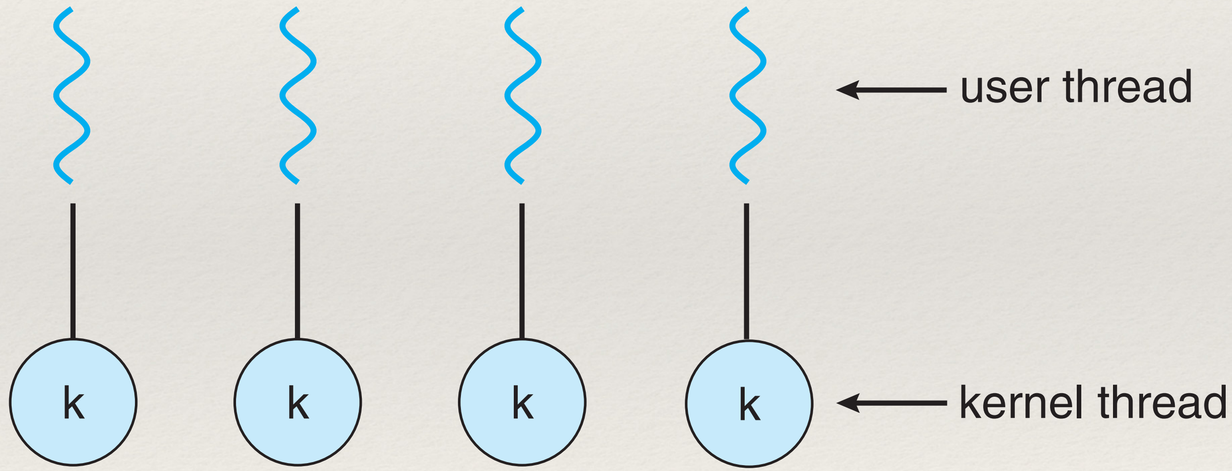
Many-to-One

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



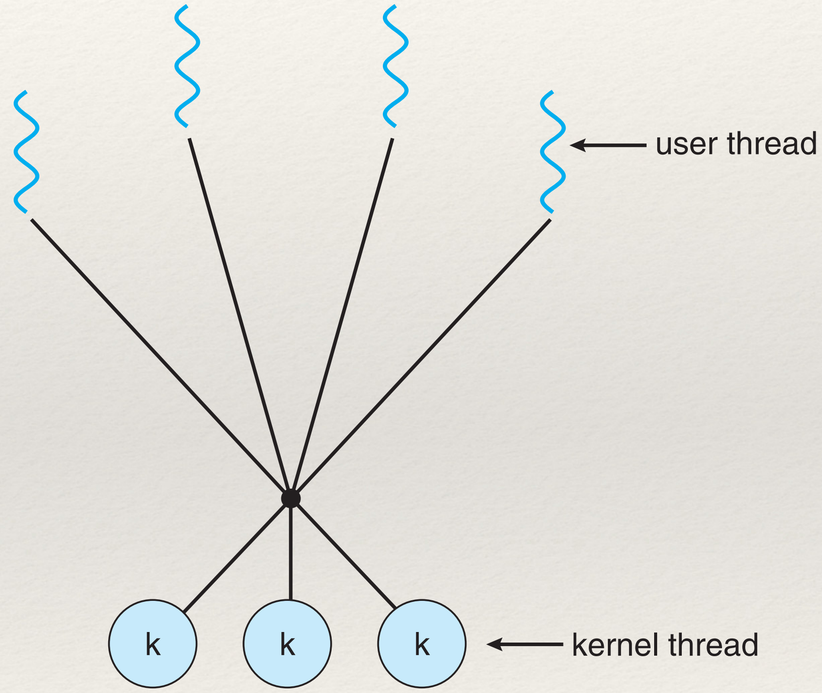
One-to-One

- Each user-level thread maps to a kernel thread
- Creating a user-level thread creates a kernel thread
- Number of threads per process sometimes restricted due to overhead
- Examples:
  - Windows
  - Linux
  - Solaris 9 and later



Many-to-Many 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
- Solaris prior to version 9
- Windows with the ThreadFiber package



Thread Libraries

- Thread Library - provides an API for creating and managing threads
  - User level
  - Kernel level

Thread Libraries

- Three main thread libraries:
  - POSIX Pthreads
  - Windows
  - Java
    - Windows - Typically uses Windows API
    - Unix/Linux - Typically implemented using Pthreads

Thread Libraries

- Three main thread libraries:
  - POSIX Pthreads
  - Windows
  - Java
    - Windows - Typically uses Windows API
    - Unix/Linux - Typically implemented using Pthreads

Thread Libraries

- Two general strategies for creating multiple threads
  - Asynchronous - Once the parent creates the child, the parent resumes it's execution
  - Synchronous - Parent waits for the children to complete, then resumes (can also be referred to as the 'fork-join' strategy)

Thread Libraries - Pthreads

- Pthreads - POSIX standard defining an API for thread creation and synchronization.
  - Specification, not an implementation
  - IEEE 1003.1c

Thread Libraries - Pthreads

- Pthreads - POSIX standard defining an API for thread creation and synchronization.
  - Specification, not an implementation
  - IEEE 1003.1c
    - https://standards.ieee.org/standard/1003_1c-1995.html

Thread Program - Sample

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h> /* atoi */

int sum;

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 */
```

Thread Program - Sample

```
   /* Create an object with the default attributes */
    pthread_attr_init(&attr);

    /* create the thread with the default attributes */
    pthread_create(&tid, &attr, runner, argv[1]);

    /* wait for the thread to exit */
    pthread_join(tid, NULL);

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

Thread Program - Sample

```
void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;

    for(i = 1; i <= upper; i++){
        sum += i;
    }
    pthread_exit(0);
}
```

Thread Attributes - Group Exercise

- Split into small groups
- What type of thread attributes are there?
  - https://man7.org/linux/man-pages/man3/pthread_attr_init.3.html
  - Choose a few to examine and answer the following questions:
    - What function does the attribute serve?
    - What options can be set for it?

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Separating task to be performed from mechanics of creating a task allows different strategies for running task
    - i.e., tasks could be scheduled to run periodically