Introduction to threads

Juan Felipe Medina Lee, Ph.D.

Computer Science and Engineering
University of Puerto Rico
Mayagüez Campus

Overview

- Overview
- Multicore Programming
- Multithreading Models
- Thread Libraries

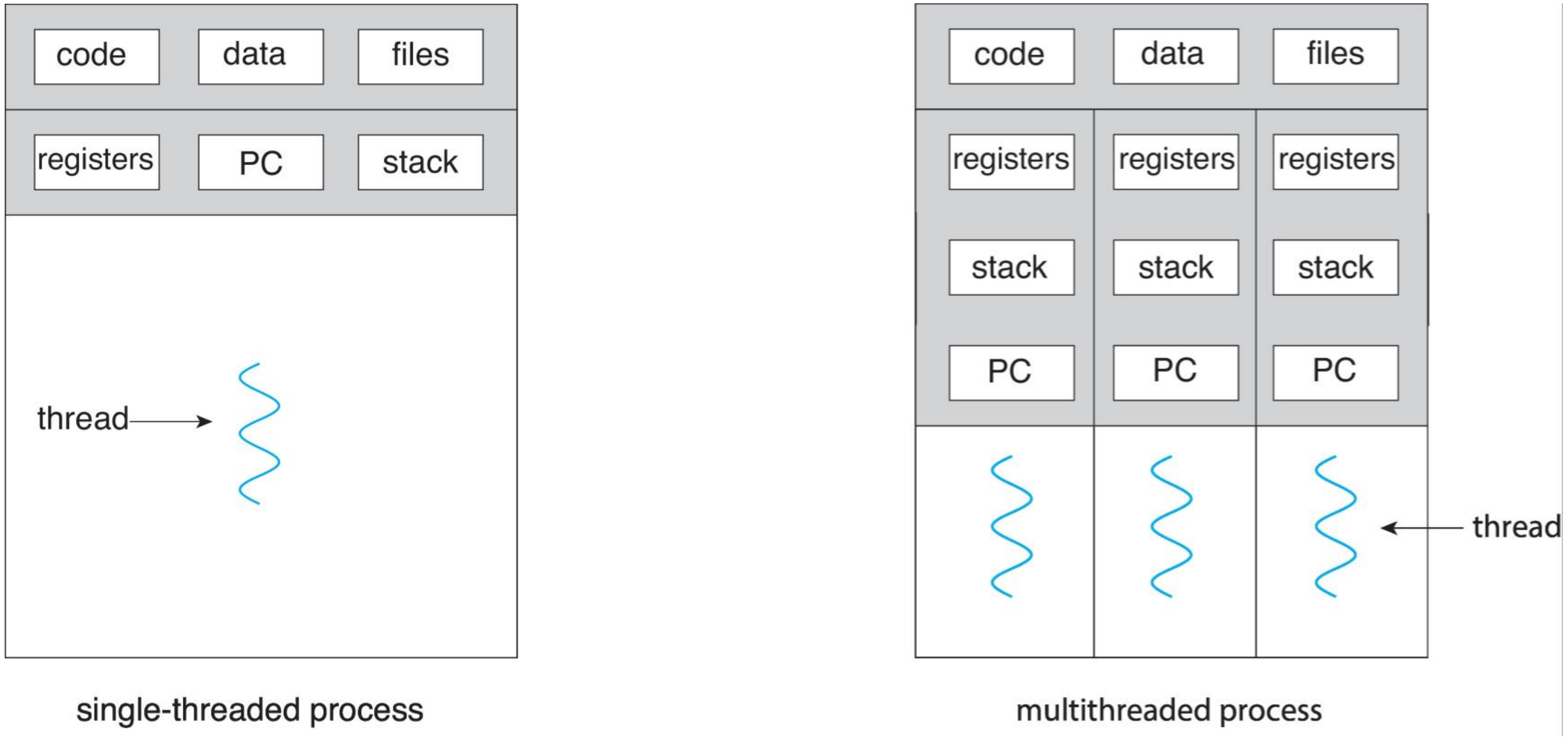
Motivation

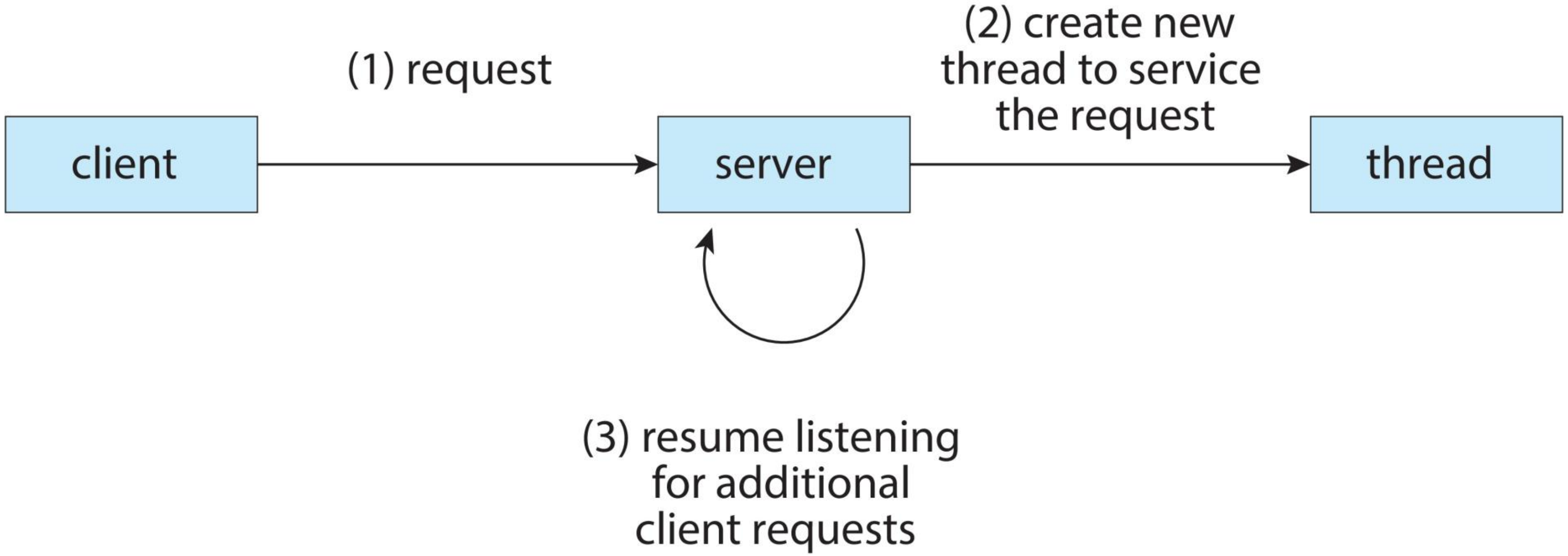
- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks in the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

What is a thread?

- A **thread** is a basic unit of CPU utilization within a process. In computing, it represents a single sequence of executed instructions that can run concurrently with other threads within the same process.
- Each thread shares the same resources with other threads in the process
  - Memory
  - File descriptors
  - Global variables
- Nevertheless, they operate independently in terms of execution flow.

Single and multithreaded processes





Multithreaded server architecture

Benefits

Responsiveness – This may allow continued execution if part of the process is blocked, which is especially important for user interfaces.

Resource Sharing – threads share resources of process, which is easier than shared memory or message passing.

Economy – cheaper than process creation, thread switching lower overhead than context switching.

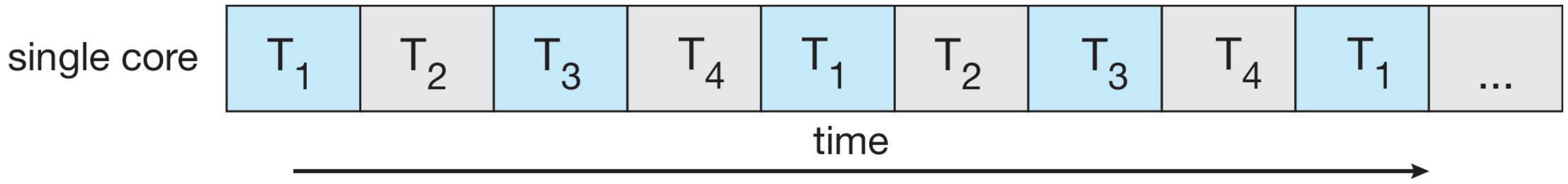
Scalability – process can take advantage of multicore architectures.

Multicore programming

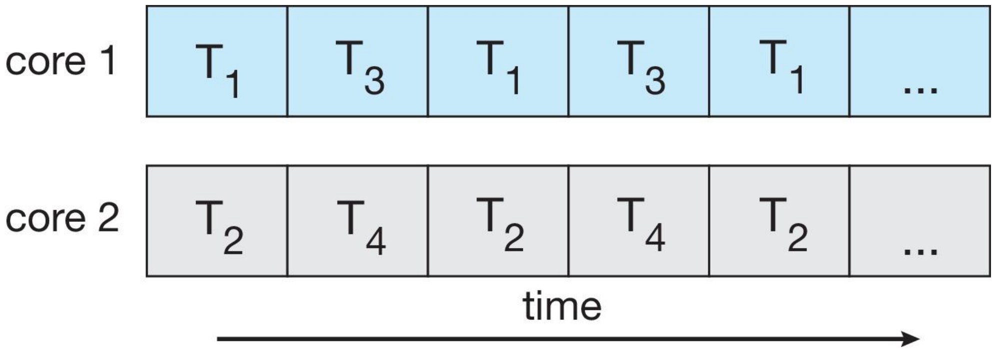
- Multicore or multiprocessor systems put pressure on programmers, challenges include:
    - Dividing activities
    - Balance
    - Data splitting
    - Data dependency
    - Testing and debugging
- *Parallelism* implies a system can perform more than one task simultaneously
- *Concurrency* supports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

Concurrency vs parallelism

- Concurrent execution on a single-core system:



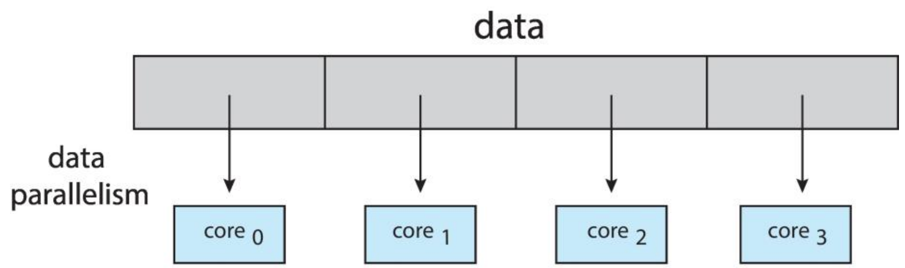
- Parallelism on a multi-core system:



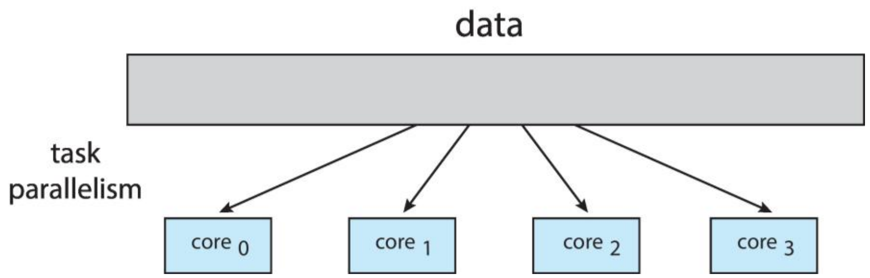
Concurrency vs parallelism

Types of parallelism:

- Data parallelism – distributes subsets of the same data across multiple cores, same operation on each



- Task parallelism – distributing threads across cores, each thread performing unique operation



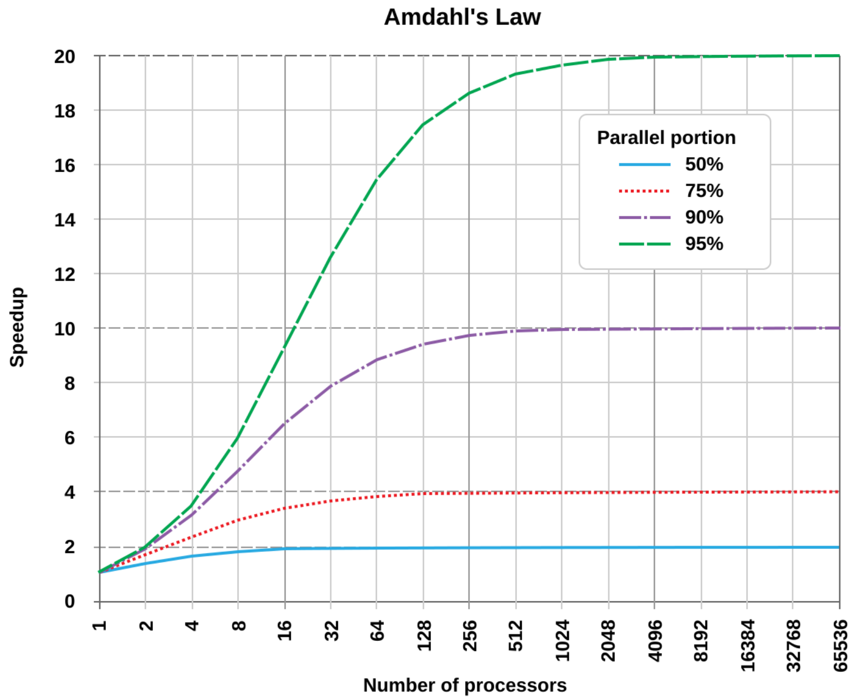
Ahmdal’s Law

Identifies performance gains from adding additional cores to an application that has both serial and parallel components

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

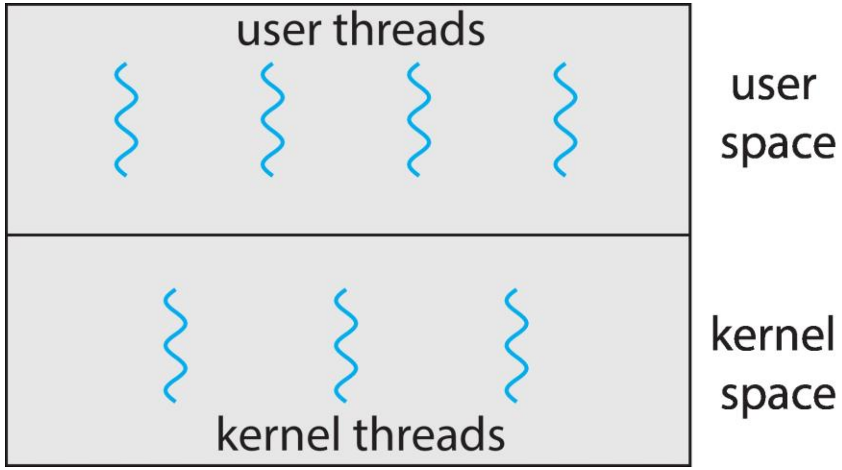
Where:

- S(N) is the speedup with N processors.
- P is the proportion of the program that can be parallelized.
- (1 - P) is the proportion of the program that is sequential and cannot be parallelized.
- N is the number of processors or cores.



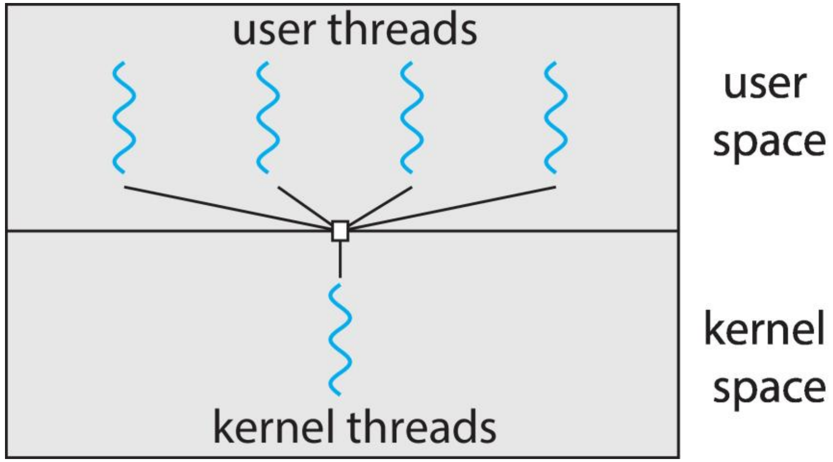
User threads and kernel threads

- User threads - management is done by user-level threads library
- Kernel threads - Supported by the Kernel
- Examples – virtually all general-purpose operating systems, including:
  - Windows
  - Linux
  - Mac OS X
  - iOS
  - Android



Multithreading models: Many to one

- 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
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



Multithreading models: Many to one

Each user-level thread maps to kernel thread

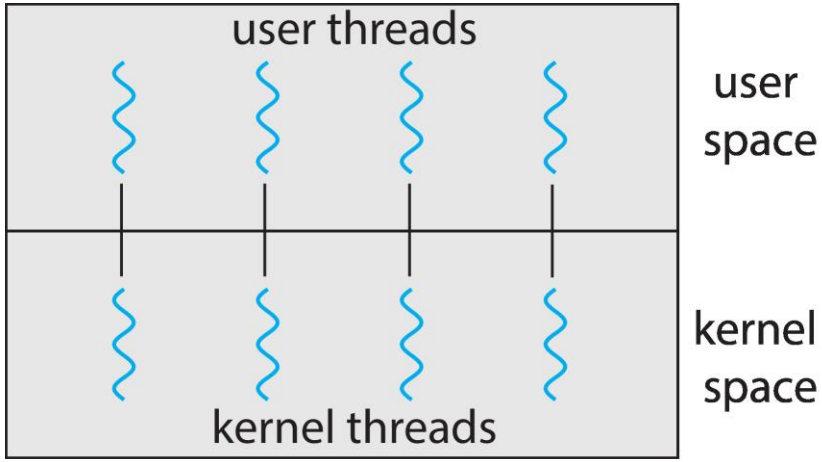
Creating a user-level thread creates a kernel thread

More concurrency than many-to-one

Number of threads per process sometimes restricted due to overhead

Examples

- Windows
- Linux



Posix threads

PTHREADS

Pthreads

- In GNU/Linux, it provides threads at kernel level
- A POSIX standard (IEEE 1003.1c) **API** for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies the behavior of the thread library. Implementation is up to the development of the library
- Common in UNIX operating systems (Linux & Mac OS X)

Pthreads functions (1)

```
#include <pthread.h>

int pthread_create(pthread_t * thread, pthread_attr_t * attr, void
* (*start_routine)(void *), void * arg);
```

- **pthread_create** creates a new thread of control that executes concurrently with the calling thread.
- The new thread applies the function **start_routine**, passing `arg` as the first argument.
- The new thread terminates explicitly by calling `pthread_exit` or implicitly by returning from the **start_routine** function.
    - Returning a value is equivalent to calling **pthread_exit** with the result value as exit code.
- The `attr` argument can also be `NULL`, in which case the default attributes are used: the created thread is joinable (not detached) and has default (non real-time) scheduling policy.

Pthreads functions (2)

```
void* function(void* param)
```

- The function associated with a thread **must** return a void pointer and receive a void pointer

Pthreads functions (3)

- The pthread_attr_t is a data type in the POSIX threads (pthreads) library that is used to specify attributes when creating new threads. It provides a way to define various properties of a thread, such as:
  - its stack size
  - scheduling policy
  - the thread should be joinable or detached.
- By using the pthread_attr_t structure, you can customize thread behavior beyond the default settings when calling pthread_create().

```
pthread_attr_t attr;

pthread_attr_init(&attr);
```

Pthreads functions (4)

```
#include <pthread.h>

int pthread_join(pthread_t th, void **thread_return);
```

`pthread_join` suspends the execution of the calling thread until the thread identified by *th* terminates, either by calling `pthread_exit` or by being canceled.

Thank you

Juan F. Medina

Juan.medina26@upr.edu