Threads

CPEN333 – System Software Engineering
2021 W2
University of British Columbia

Electrical and Computer Engineering

Introduction

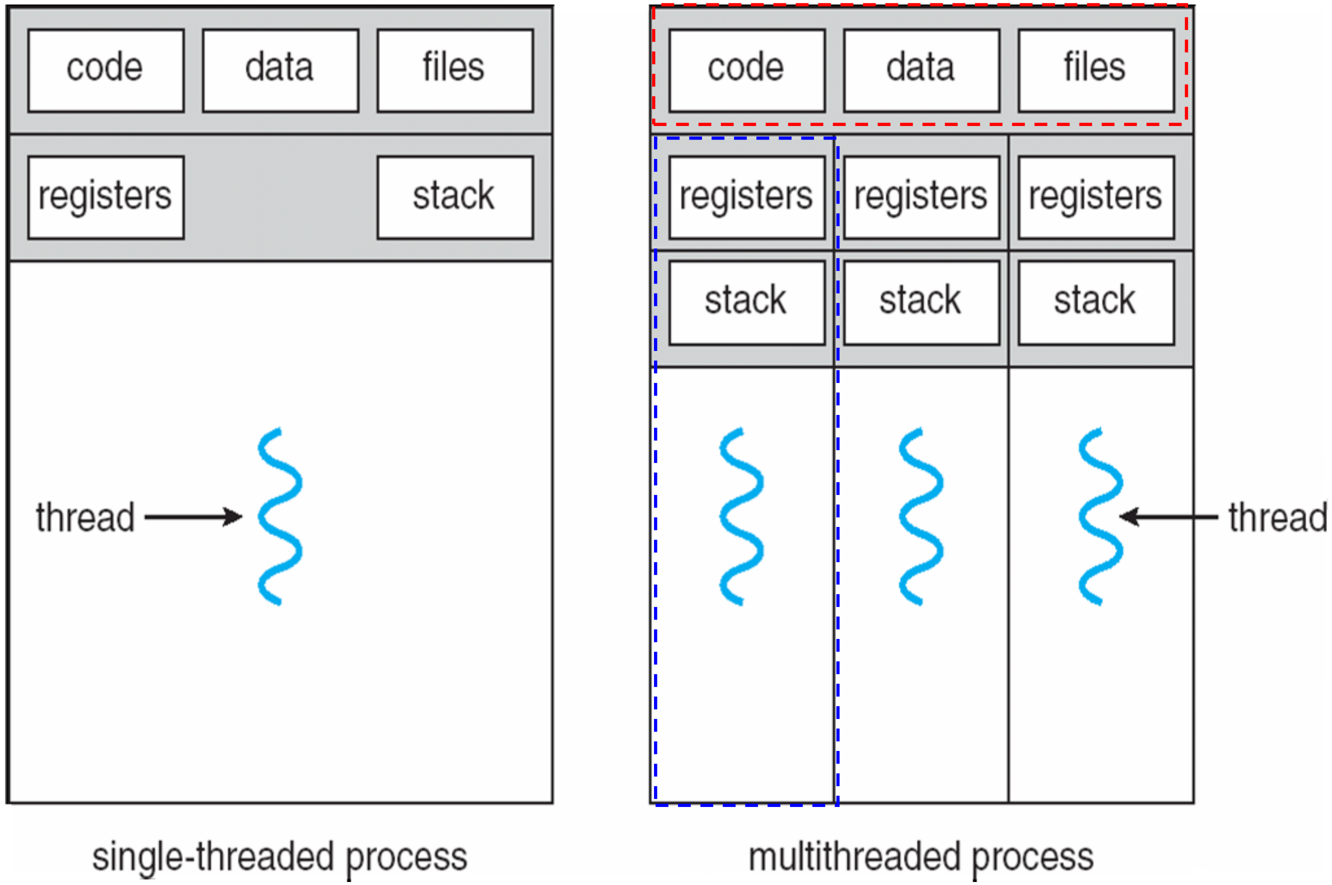
- We have discussed the notion of a process, but we assumed that a process was an executing program with a single thread of control.
- All modern OS enable a process to contain multiple threads of control.
- In this set of slides, we introduce the thread concept and some fundamental concepts associated with multithreaded systems.

Objectives

- To introduce the notion of a thread
  - a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
- To discuss the motivation
- To examine issues related to multithreaded programming

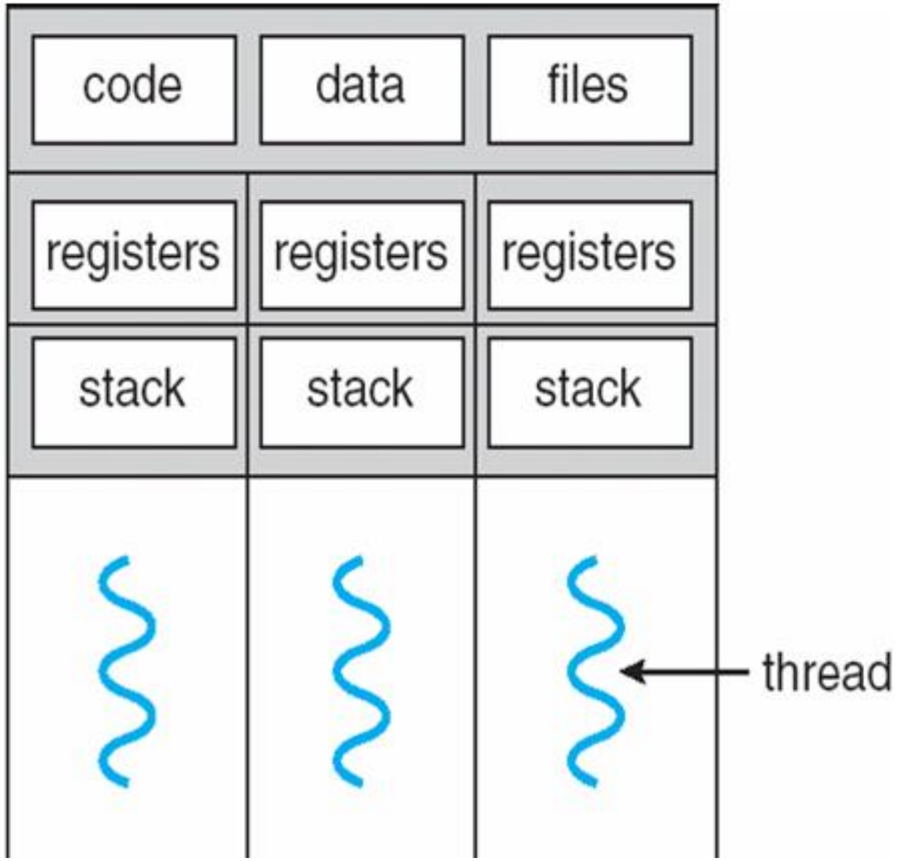
Multithreaded process

- A process can have multiple threads of control, allowing it to perform more than one task concurrently.



Threads

- A thread is a basic unit of CPU utilization
  - It comprises a thread ID, a program counter, a register set, and a stack
  - It shares with other threads belonging to the same process its code section, data section and possibly some other OS resources (e.g. open files)

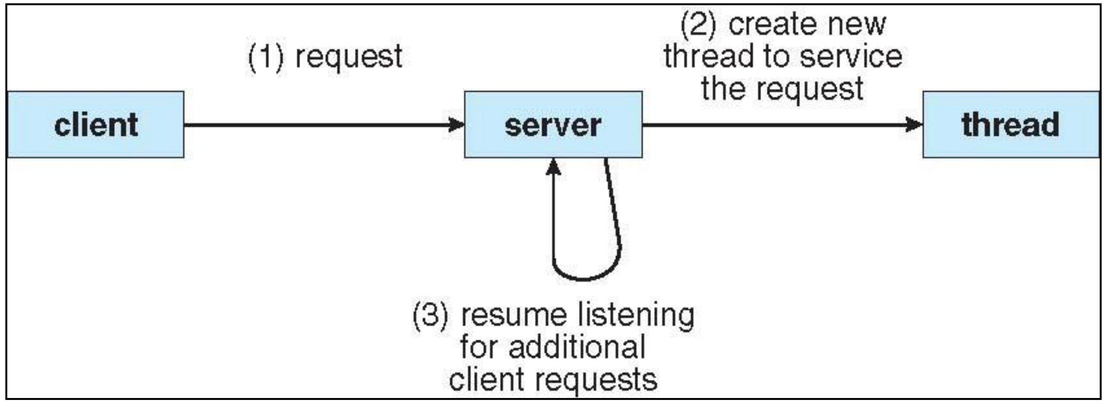


- Many software packages are multithreaded
  - An application typically is implemented as separate processes with several threads of control

Motivation

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

A multithreaded web server:


client
(1) request
server
(2) create new thread to service the request
thread
(3) resume listening for additional client requests

Multithreaded Processes

Q: why not use process-creation method always?

- Process creation is time consuming and resource intensive

- If they are to perform similar tasks, it is generally more efficient to use one process that contains multiple threads instead.

- Note: that the programming language, multi-tasking framework, … can also influence the decision.

Benefits of multithreaded programming

- **Responsiveness**: may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing**: threads share some resources of process, 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 multiprocessor architectures

Amdahl’s Law

- Admahl’s law identifies the theoretical performance gains from adding additional cores to an application that has both serial and parallel components
  - ***S*** is serial portion of the code
  - ***N*** is # of processing cores

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

- i.e. if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As ***N*** approaches infinity, speedup approaches ***1 / S***
  - That is, the serial portion has disproportionate effect on performance gained by having additional cores
- Some argue that the law does not take into account the hardware performance enhancements of the contemporary multicore systems.

Concurrency

- Concurrency is, essentially, the practice of doing multiple things at the same time, but not, specifically/necessarily, in parallel.

- Example, when you multitask, you are working on multiple tasks seemingly at the same time, but you focusing on one task at a time but quickly switch back and forth between the tasks.

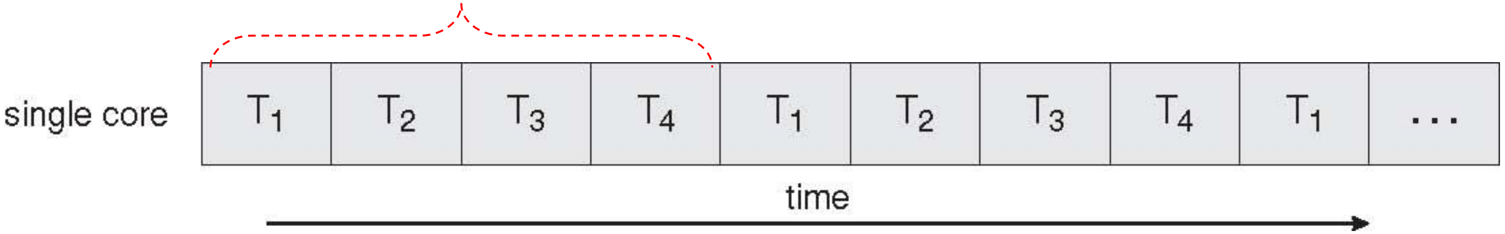
Parallelism

- Parallelism is the art of executing two or more actions simultaneously as opposed to concurrency in which you make progress on two or more things at the same time.

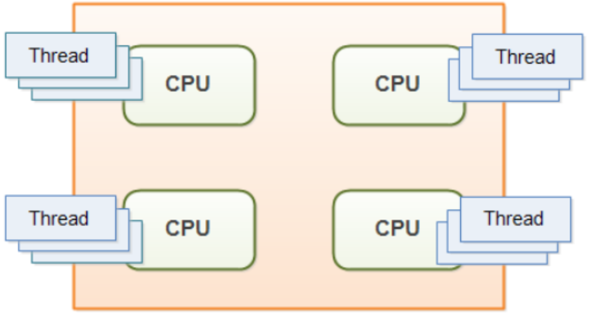
- It implies a system can perform more than one task simultaneously
  - We need multiple available processors to execute code in parallel, but we may achieve concurrency even with one CPU.
- Types of parallelism
  - Data parallelism – distributes subsets of the same data across multiple cores, same operation on each
  - Task parallelism – distributing threads across cores, each thread performing unique operation

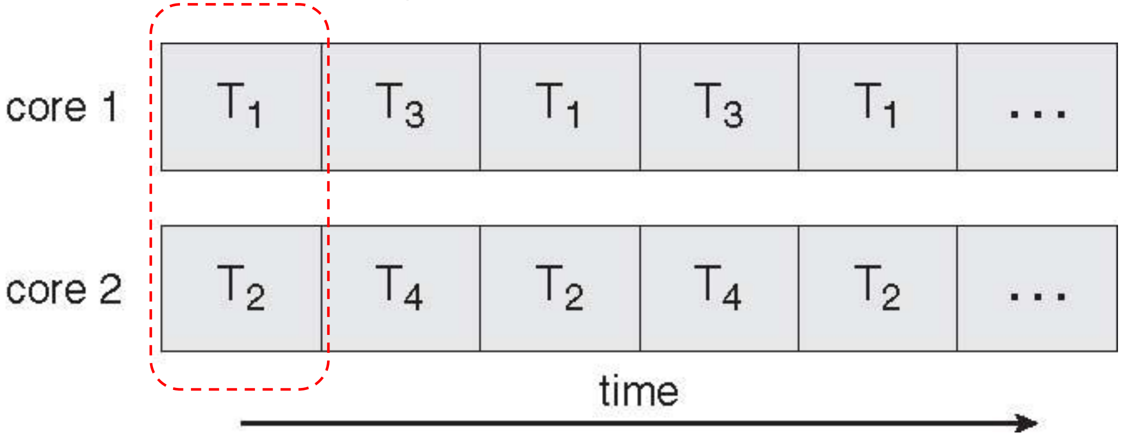
Multicore Programming

- Multithreaded programming provides a mechanism for more efficient use of multiple core and improved concurrency.
  - On a system with a single computing core, concurrency merely means that the execution of the threads will be interleaved over time (time-sharing).



  - On a system with multiple cores, concurrency means that the threads can run in parallel.

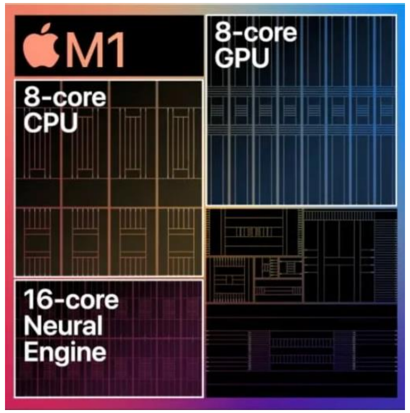




core 1
T_1 T_3 T_1 T_3 T_1 . . .
core 2
T_2 T_4 T_2 T_4 T_2 . . .
time

GPU

- GPU (Graphics Processing Unit): used not only for graphics, video rendering and gaming, but also for artificial intelligence due to their parallel processing capabilities.
- e.g. nVidia’s **RTX 3090** has 10496 GPU cores.
- e.g. Apple’s **M1 Max** (a system on chip) has up to 32 GPU cores.



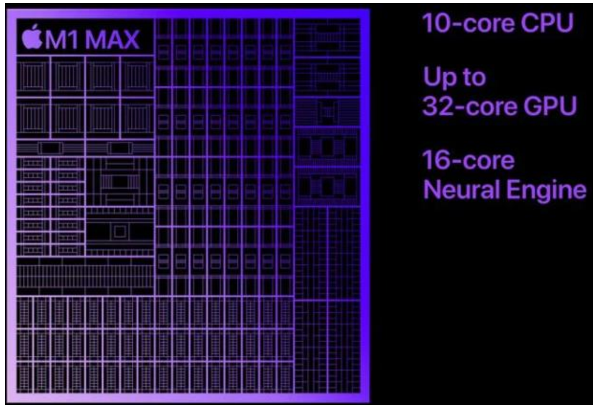
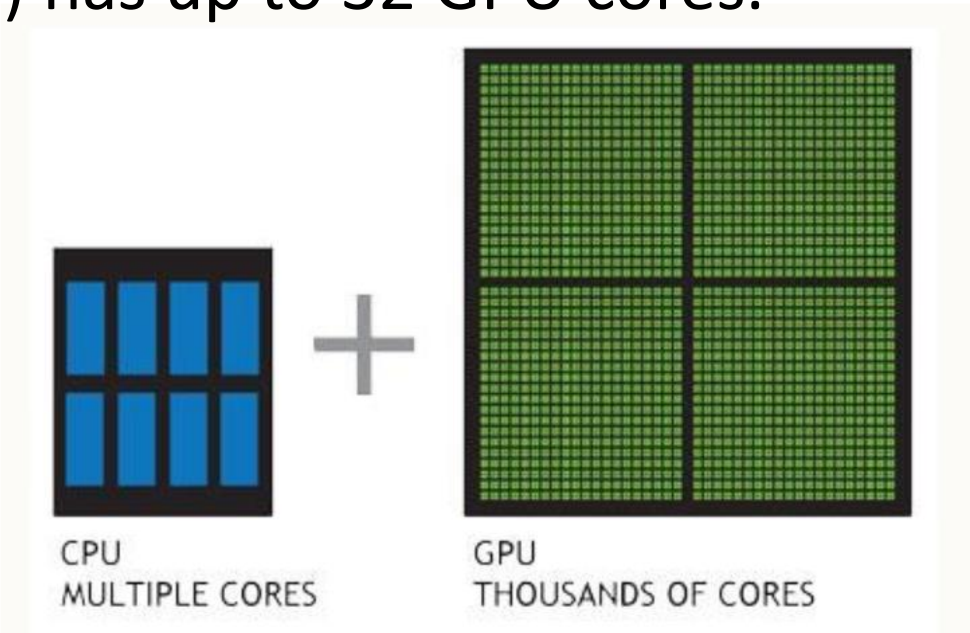


Image: apple.ca

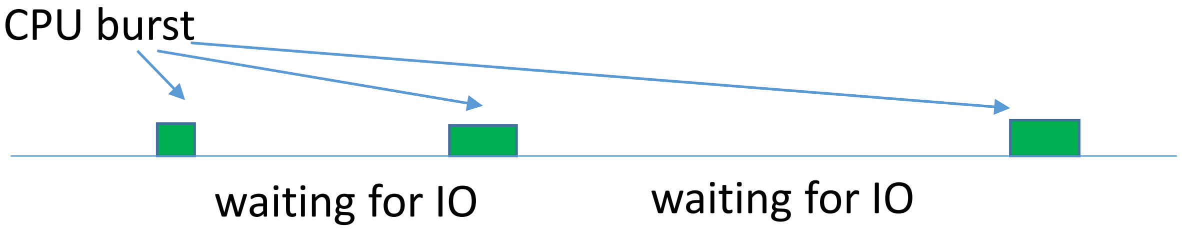


Multicore Programming

- The trend towards multicore systems has placed pressure on system designer and application programmer to make better use of the multiple computing cores
- Some of the areas of challenges in programming for multicore systems:
  - **Dividing activities**: how to divide into separate/concurrent tasks
  - **Balance**: ensuring tasks perform roughly equal work
  - **Data splitting**: how to divide data to run on separate cores
  - **Data dependency**: difficulty arising when a task depends on data from another task
  - **Testing and debugging**: inherently more difficult due to many possible execution paths

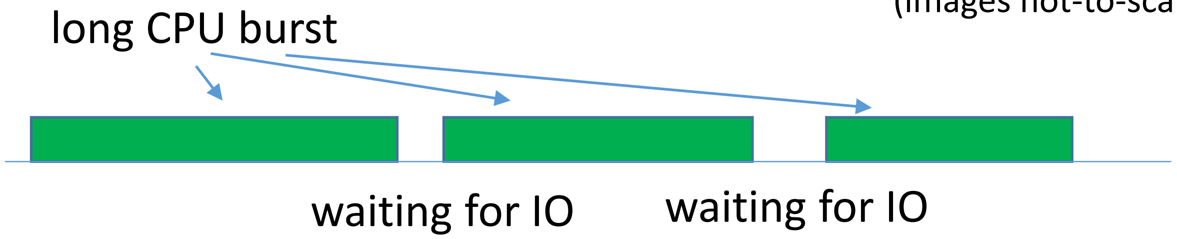
CPU-bound vs I/O bound

- I/O-bound task spends more time doing I/O than computations, many short CPU bursts



- CPU-bound task spends more time doing computations; few long CPU bursts

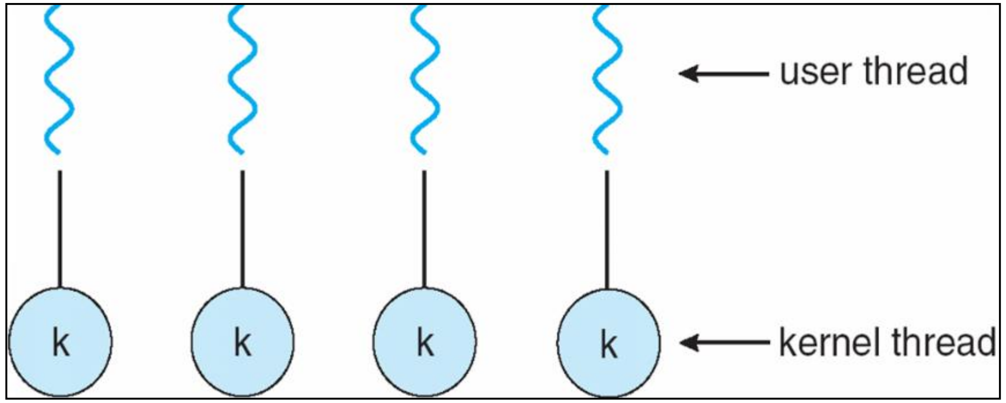
(images not-to-scale necessarily)



- Why is this important?
  - We will soon see that the multitasking model we choose may depends on whether the tasks are primarily CPU-bound or IO bound.

Support for Threading

- Support for threads may be provided either at the user level or by the OS kernel.
- At the user level for user threads: supported above the kernel and are managed without kernel support, done by a user-level thread library
- By the kernel for kernel threads: supported and managed directly by the OS
- Ultimately, a relationship exists between user threads and kernel threads
  - For example: a one-to-one model



There are also many-to-one, many-to-many, … models

References

- Some sections from chapter 4 of Operating Systems Concepts

Acknowledgement: This set of slides is partly based on the PPTs provided by the Wiley’s companion website for the operating system concepts book (including textbook images, when not explicitly mentioned/referenced).