Processes and Threads

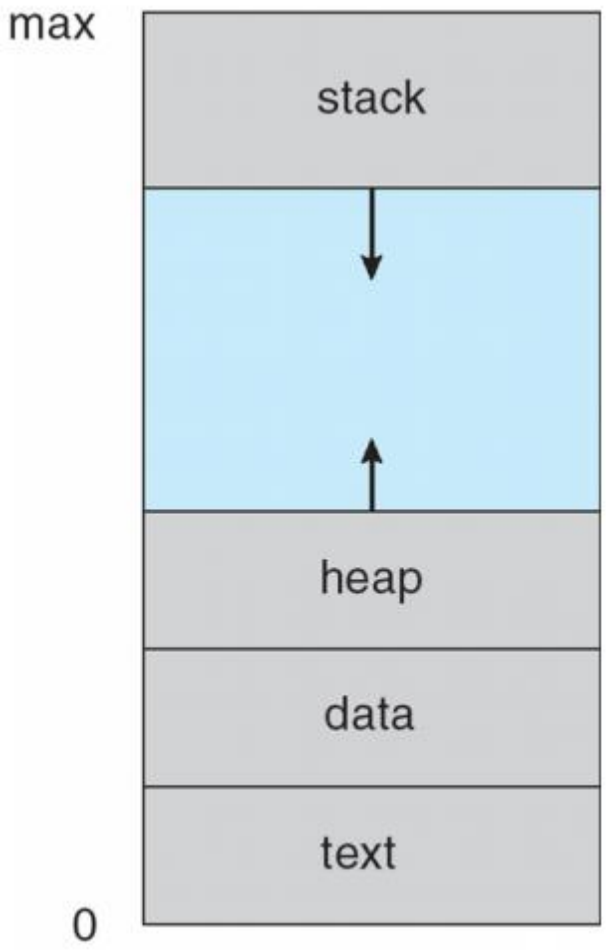
Processes: definition, states, process life cycle, Process Control Block (PCB), process queues and context switching.

Threads: motivations, definition, benefits.

What is a process?

- A process is a program in execution
  - It contains
    - Program code — **text section**
    - **Data**
      - **Data section** -- static variables and global variables
      - **Stack** –Temporary data (method parameters, return addresses and local variables)
      - **Heap** – dynamical memory allocation
    - The contents of the various **CPU registers**
      - **Data/address/general purpose registers**
      - **Special purpose registers**
        - **Program counter** – indicates what instruction the process is to execute next.
    - **State**

Process in Memory



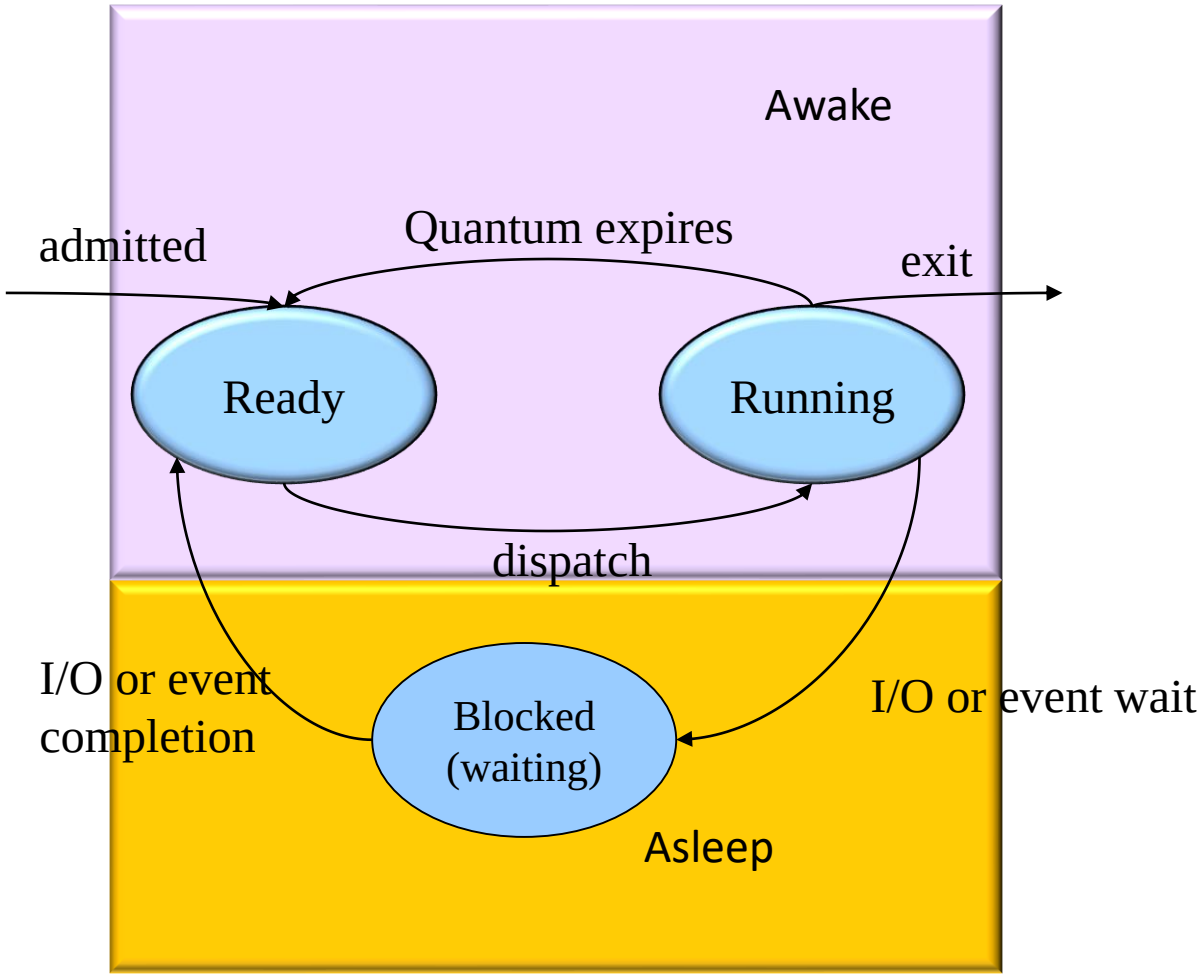
Process States

- Normally there are more processes than CPUs in a system
  - At any given time, only one process can be running on a CPU.
- OS uses an **interrupt clock (interval timer)** to prevent any monopolization of resources
  - It allows a process to run for a specific time interval or **quantum**

Process States

- A process moves through a series of discrete process states
  - **Running** State – the process is executing on a CPU
  - **Ready** State – the process could execute on a CPU when one is available
    - A ready queue for ready processes
  - **Blocked (waiting)** State – the process is waiting for some event to happen before it can proceed.
    - device queues for blocked processes
  - Also others: **new** (being created), **terminated** (finished execution), …

The Life Cycle of a Process

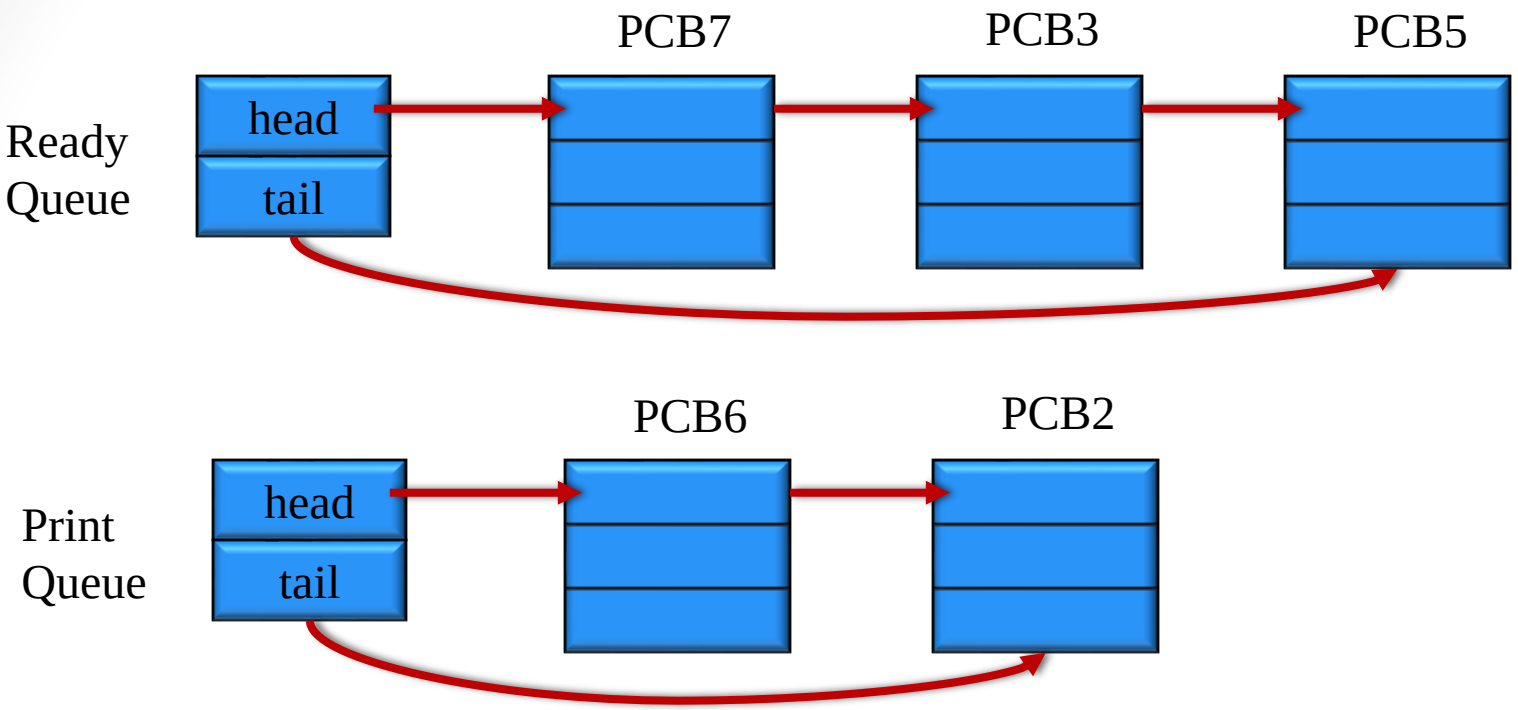


Process Control Block (PCB)

- Each process is represented in the OS by a PCB.
- PCBs typically include
  - Process state
  - Process ID
  - The values of the CPU registers
    - Program counter
    - Stack pointers
    - General purpose registers
    - Other registers
  - Memory-management information
    - Base and limit registers
  - Scheduling and resource allocation information
    - Open files

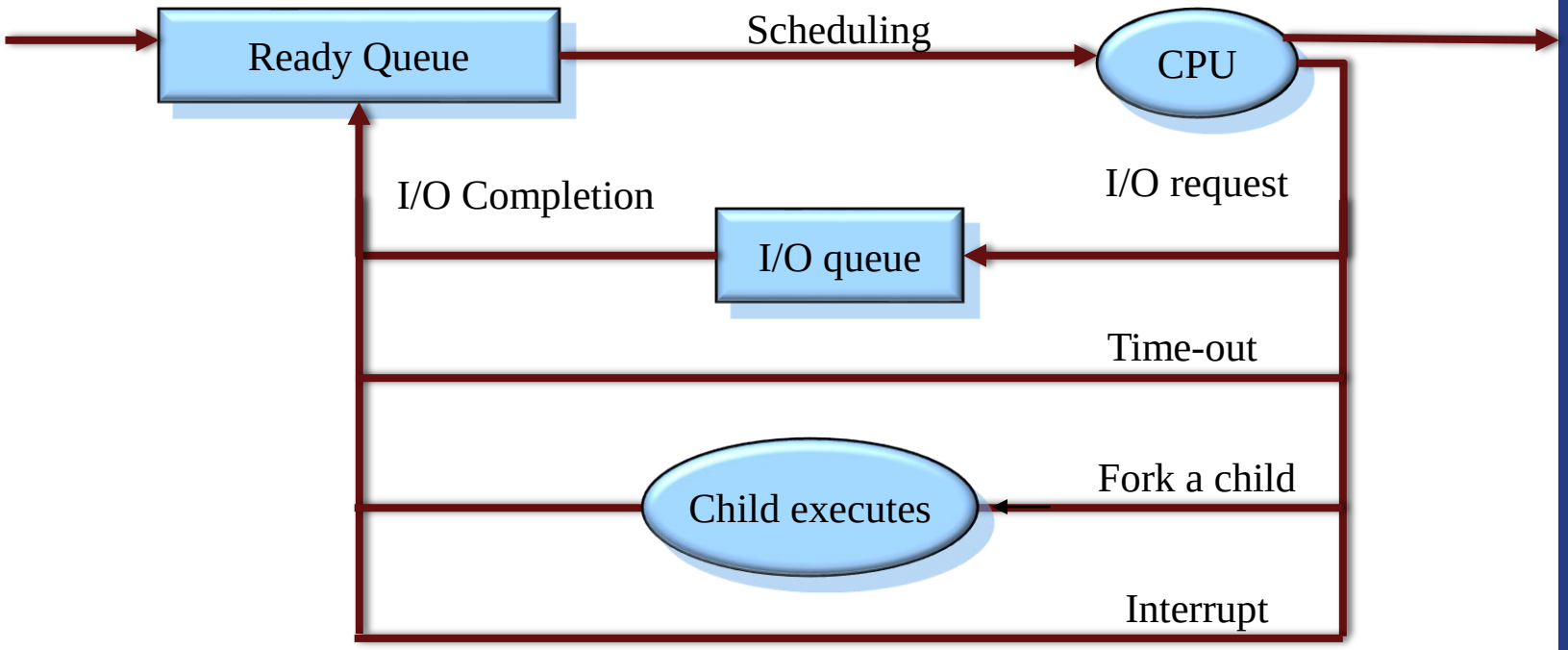
Process Queues

- **Ready queue**: a list of processes that reside in memory and are ready and waiting to execute
- **Device queue**: a list of processes waiting for a particular I/O device. Each device has its own device queue



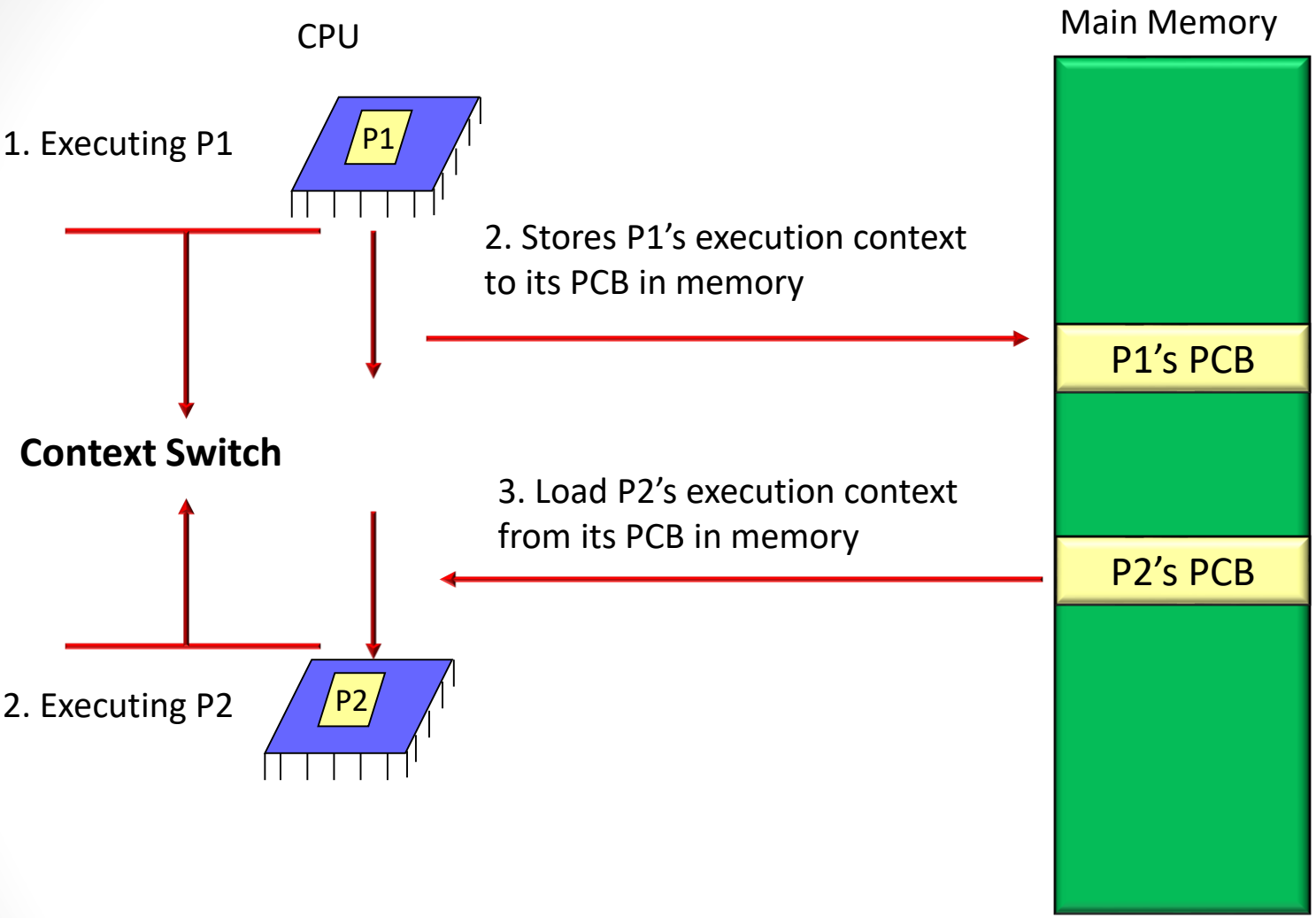
Process queues

Scheduling Queues



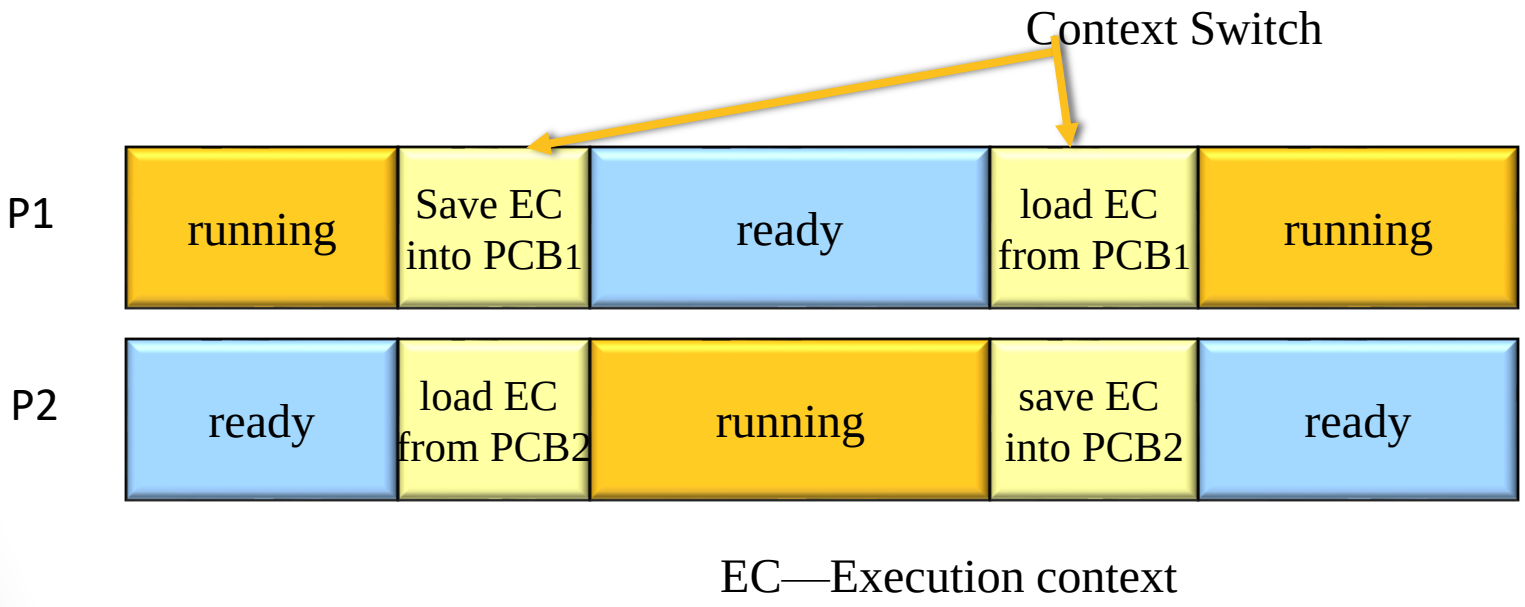
Context Switching

- One major use of the PCBs is in conjunction with a **context switch** operation
- A **context switch** is performed by the OS to stop executing a running process and begin executing a previously ready process
- Switching the CPU to another process requires saving the execution context of the old process into its PCB and loading the execution context of the new process.


CPU
Main Memory
1. Executing P1
P1
2. Stores P1's execution context to its PCB in memory
P1's PCB
Context Switch
3. Load P2's execution context from its PCB in memory
P2's PCB
2. Executing P2
P2

Context Switching

- Context switching is pure overhead.



EC—Execution context

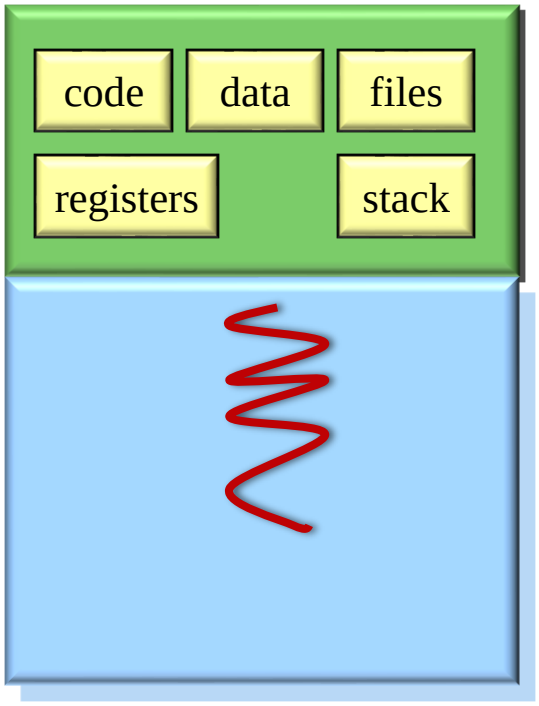
Threads

- Why Thread?
  - A single application may be required to perform several tasks
- A thread, also called a **lightweight process (LWP)**, is a basic unit of CPU utilization that is under the control of a process.
  - A traditional process (or **heavyweight process (HWP)**) has a single thread of control
  - A multithreaded process has multiple threads of control and it can do more than one task at a time.

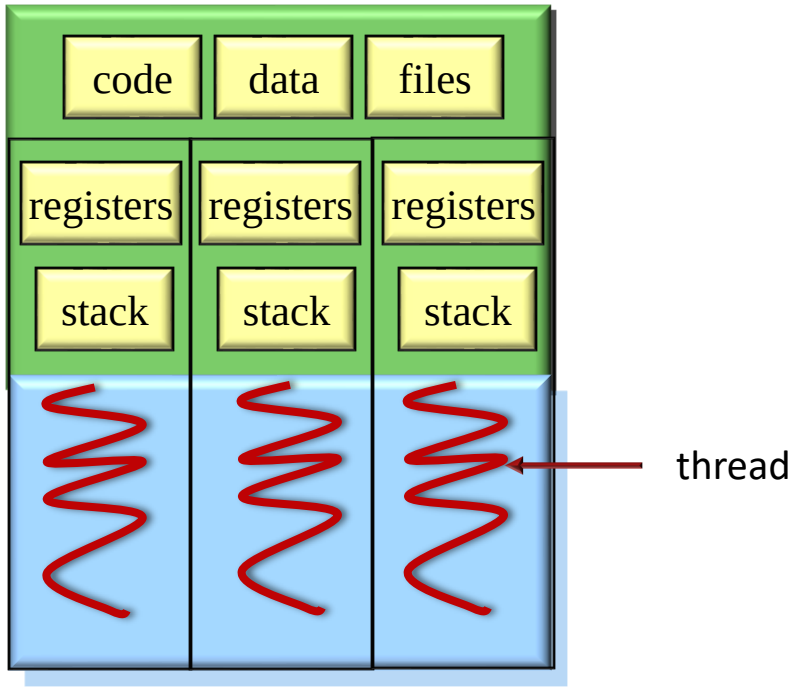
What is a thread?

- All the threads in the same process
  - Sharing
    - memory context (code and data) – no memory protection between the threads
    - Other resources (e.g. open files)
  - Not sharing
    - register context --including PC (Program Counter) and SP (Stack Pointer)
    - Each thread has its own stack
      - All the stacks for the various threads are located in the same data space
      - Each has its own SP and uses a different part of this space for its stack.

Difference between a Single-Threaded Process and a Multi-Threaded Process



Single-threaded process



Multi-threaded process

Benefits of Multithreaded Programming

- Interactivity
  - An application may continue to run even if it is blocked or is performing a lengthy operation
- Resource sharing
  - Threads share the same address space and the resources of the process to which they belong

Benefits of Multithreaded Programming

- Economy
  - Faster to create threads
    - It is costly to allocate memory and resources for process creation
  - Faster to context-switch threads
- Increased Concurrency in a multiprocessor architectures
  - A single-threaded process can only run on one CPU.
  - The threads in a multithreaded process may be running in parallel on different CPUs.

Summary

- Processes
  - Definition
  - Status and life cycle
  - PCB and context switch
  - Process Queues
- Threads
  - What are threads and why threads?
  - Benefits of multithreading