CMPT 300
Operating System I
CPU Scheduling – Chapter 5

Dr. Hazra Imran

Summer 2022

Admin notes

- Quiz 3 available on canvas

Today's plan

- In class activity 2 (20 min)

- Chapter 5

In-class activity

Learning goals

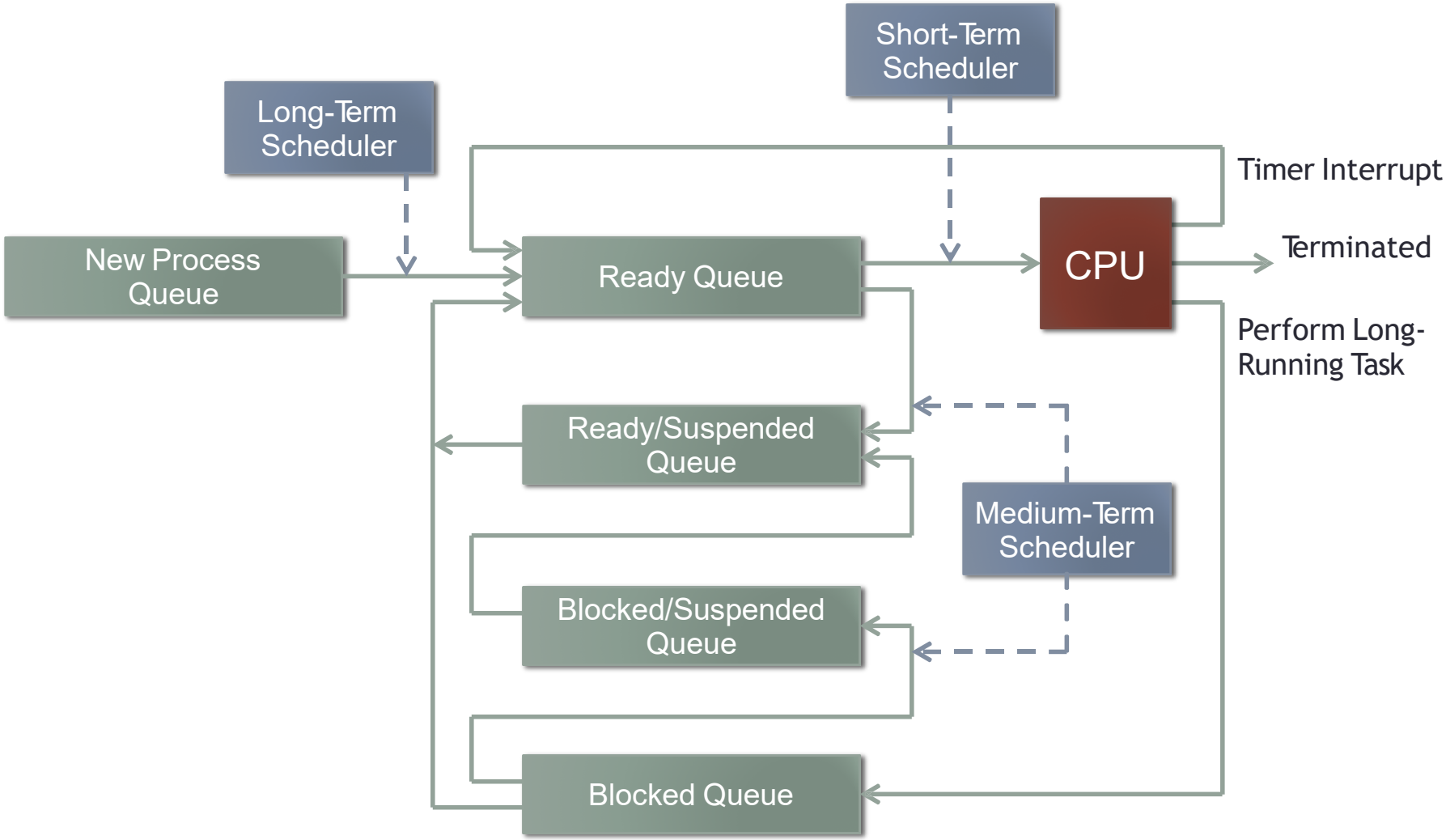
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system

gantt chart

Process Scheduling

- OS must manage the allocation and sharing of hardware resources to the applications that use them
- Most important resource for multitasking OS is the CPU
- The goal is to have multiple concurrently executing processes
- Processes fall into various categories based on their state
  - "Running" processes are on a CPU
  - "Ready" processes don't have a CPU, but could run if they did (i.e. not blocked on I/O).
- How to allocate CPU time to the processes that can run?

Recap of the queues



CPU-bound and I/O-bound

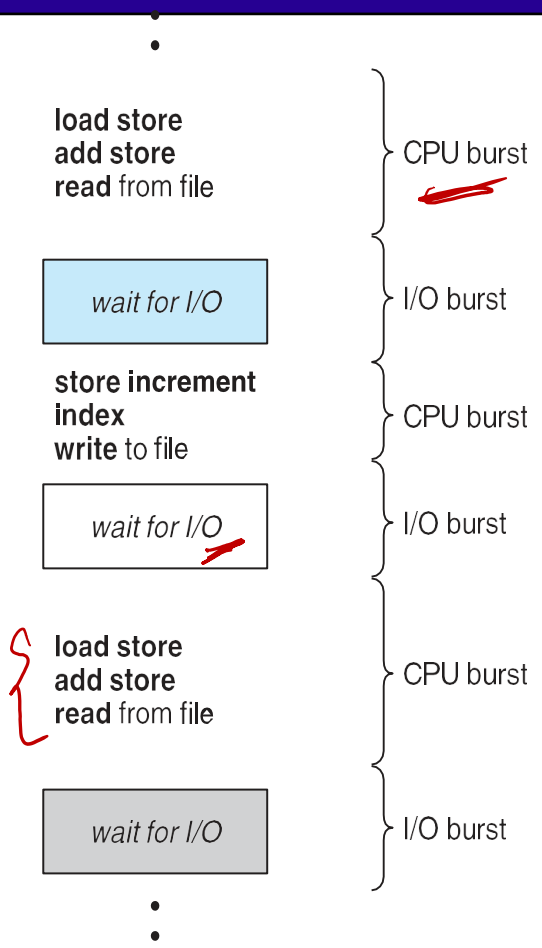
- We can divide processes into I/O-bound or CPU-bound

- **CPU-bound** process spends (almost) all of its time in a ready/running state.
- The bottleneck is the CPU.
- Would a faster CPU speed up this process?
  - Yes
  - No

- **I/O-bound** process spends (almost) all of its time in a sleep/waiting state.
- The bottleneck is I/O
- Would a faster CPU speed up this process?
  - Yes
  - No

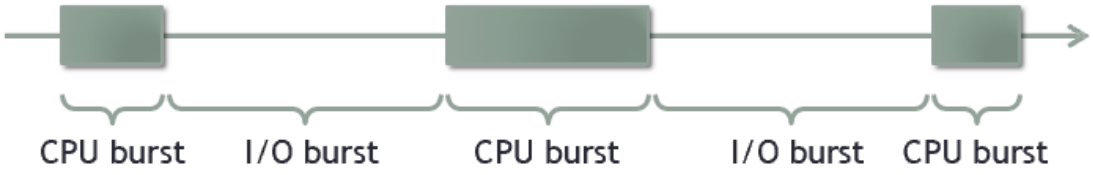
Real processes are different…

- In reality, very few processes are purely computational
- In reality, very few processes are all I/O driven
- A real-life process would be described has having
  - CPU-bound periods or I/O-bound periods
  - Or, it may be described using bursts



Bursts

- A burst is a period where a process will be in a ready/running state
- It indicates how much computation a process will do before reaching its next system call
- **CPU-bound** processes have long burst lengths (more than 10ms)
- **I/O-bound** processes have short burst lengths (less than 5ms)



Scheduling Characteristics

- Some schedulers are **nonpreemptive** or **cooperative**:
  - Only perform a scheduling operation when the current process blocks or terminates
  - Processes with long CPU bursts aren't preempted
- Other schedulers are **preemptive**:
  - Processes with long CPU bursts will be interrupted, to give other processes time to execute

Cooperative multitasking

- From the beginning of operating system design, people started seeing a need for increased throughput
- Even if a computer is not being used interactively, a process which is held up by an I/O request is leaving the CPU idle
- It would be best of we could use the CPU with a different process while the previous process is in a sleeping/waiting state

Preemptive multitasking

- Cooperative multitasking is still used in some domains (especially embedded systems)
- It has a big problem.
  - One process has the ability to starve all other processes
- Under preemptive multitasking, a scheduler will check to see if there is contention for the CPU(s).
- If the CPU is under contention, the kernel will set a hardware timer to generate an interrupt after a certain length of time, called a quantum
- If the hardware timer goes off before the current running process yields to do a system call, then the kernel preempts it by forcing a context switch to the next process

CPU Scheduler… again with the feelings

- Short-term scheduler selects from among the processes in ready queue, and allocates the CPU to one of them
- Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is nonpreemptive
- Cases 2 and 3 require process preemption