OPERATING SYSTEMS

CPU Scheduling

Venkatesh Prasad
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Process Scheduling

Venkatesh Prasad
Department of Computer Science

Basic Concepts

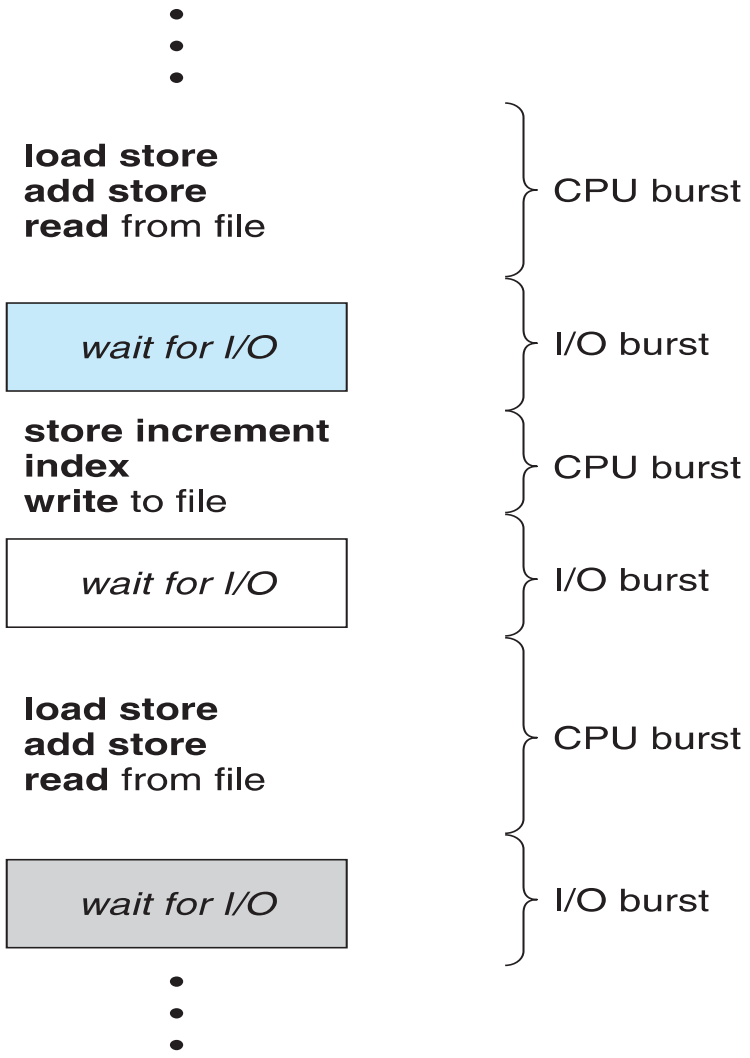
- In a system with a single CPU core, only one process can run at a time. Others must wait until the CPU is free and can be rescheduled.
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- Several processes are kept in memory at one time.
- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues.
- Every time one process has to wait, another process can take over use of the CPU. On a multicore system, this concept of keeping the CPU busy is extended to all processing cores on the system.

Basic Concepts (Contd)

- The idea is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request.
- In a simple computer system, the CPU then just sits idle. All this waiting time is wasted; no useful work is accomplished. With multiprogramming, we try to use this time productively.
- Scheduling of this kind is a fundamental operating-system function.
- Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design.

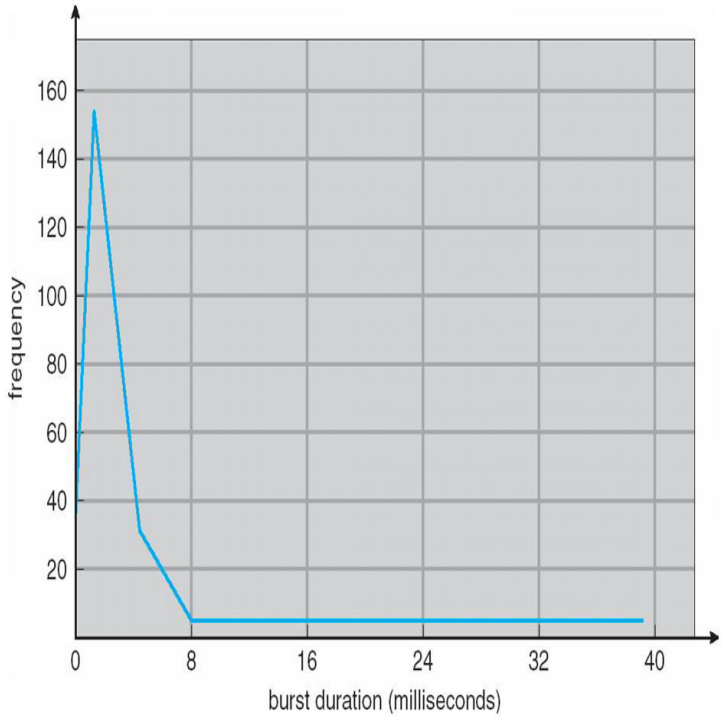
Alternating Sequence of CPU and I/O bursts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



Histogram of CPU-burst Times

- The durations of CPU bursts have been measured extensively. Although they vary greatly from process to process and from computer to computer, they tend to have a frequency curve similar to that shown in the Figure.
- An I/O-bound program typically has many short CPU bursts. A CPU-bound program might have a few long CPU bursts. This distribution can be important when implementing a CPU-scheduling algorithm.



OPERATING SYSTEMS

CPU Scheduler

- **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**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Preemptive vs Non-Preemptive Scheduling

- Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state.
- Virtually all modern operating systems including Windows, macOS, Linux, and UNIX use pre-emptive scheduling algorithms.
- Unfortunately, pre-emptive scheduling can result in race conditions when data are shared among several processes. Ex: While one process is updating the shared data, it is pre-empted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state.
- A pre-emptive kernel requires mechanisms such as mutex locks to prevent race conditions when accessing shared kernel data structures. Most modern operating systems are now fully pre-emptive when running in kernel mode.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process (performance metric)
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

Waiting time for P_1 = 0; P_2 = 24; P_3 = 27

- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

- Suppose that the processes arrive in the order: P_2 , P_3 , P_1
  The Gantt Chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu