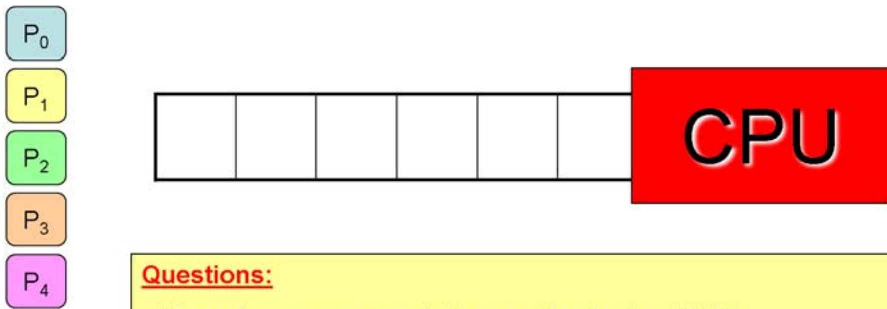
BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

CPU Scheduling Algorithms

<u>Notice:</u> The slides for this lecture have been largely based on those from an earlier edition of the course text *Operating Systems Concepts with Java*, by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source.

Basic Concepts



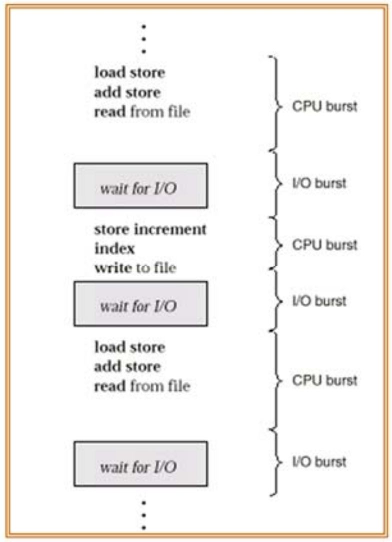
Questions:

- When does a process start competing for the CPU?
- How is the queue of ready processes organized?
- How much time does the system allow a process to use the CPU?
- Does the system allow for priorities and preemption?
- What does it mean to maximize the system's performance?

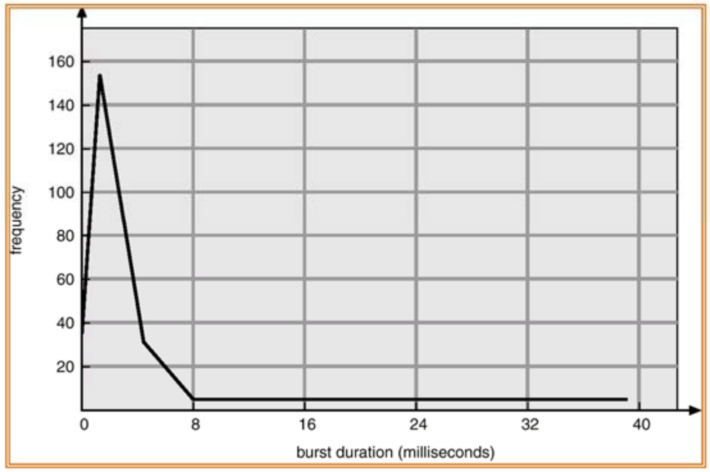
Basic Concepts

- You want to maximize **CPU utilization** through the use of multiprogramming.
- Each process repeatedly goes through cycles that alternate CPU execution (a CPU burst) and I/O wait (an I/O wait).
- Empirical evidence indicates that CPU-burst lengths have a distribution such that there is a large number of short bursts and a small number of long bursts.

Alternating Sequence of CPU And I/O Bursts



Histogram of CPU-burst Times



CPU Scheduler

- AKA *short-term scheduler*.
- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Question: Where does the system keep the processes that are ready to execute?

- 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.

Preemptive Scheduling

- In **cooperative** or **nonpreemptive** scheduling, when a process takes the CPU, it keeps it until the process either enters waiting state or terminates.

- In **preemptive scheduling**, a process holding the CPU may lose it. Preemption causes context-switches, which introduce overhead. Preemption also calls for care when a process that loses the CPU is accessing data shared with another process or kernel data structures.

Dispatcher

- The **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.
- The **dispatch latency** is the time it takes for the dispatcher to stop one process and start another running.

Scheduling Criteria

These are **performance** metrics such as:

- **CPU utilization** – high is good; the system works best when the CPU is kept as busy as possible.
- **Throughput** – the number of processes that complete their execution per time unit.
- **Turnaround time** – amount of time to execute a particular process.
- **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).

It makes sense to look at averages of these metrics.

Optimizing Performance

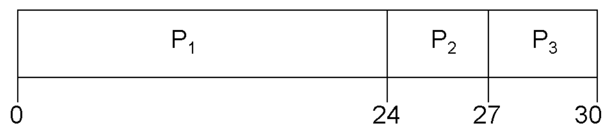
- **Maximize** CPU utilization.
- **Maximize** throughput.
- **Minimize** turnaround time.
- **Minimize** waiting time.
- **Minimize** response time.

Scheduling Algorithms

First-Come, First-Served (FCFS)

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:



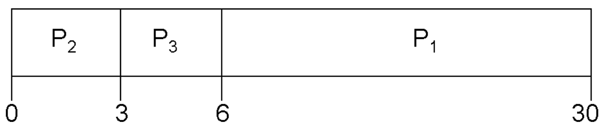
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS

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:* all process are stuck waiting until a long process terminates.

Shortest-Job-First (SJF)

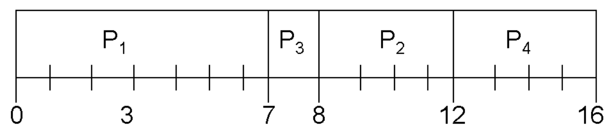
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **Nonpreemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is **optimal** – gives minimum average waiting time for a given set of processes.

Question: Is this practical? How can one determine the length of a CPU-burst?

Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

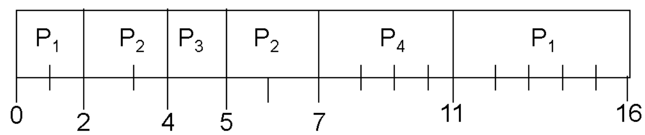


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

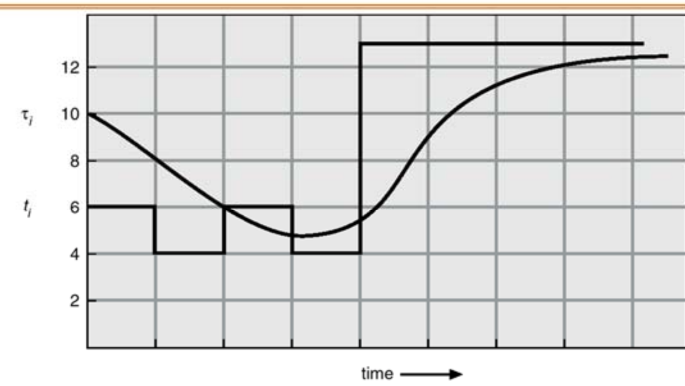
Determining Length of Next CPU-Burst

- We can only *estimate* the length.
- This can be done by using the length of previous CPU bursts, using exponential averaging:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

1. $t_n =$ actual lenght of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $0 \le \alpha \le 1$

Prediction of the Length of the Next CPU-Burst



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .