Amirkabir University of Technology
(Tehran Polytechnic)

Operating Systems

Scheduling Algorithms

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Spring 2023

First-Come, First-Served

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**

FCFS Scheduling and Convoy effect

- **Short process behind long process.**
  - Consider one CPU-bound and many I/O-bound processes.

- **What is the important side-effect?**

FCFS Scheduling and Convoy Effect (Cont.)

- **Short process behind long process.**
  - Consider one CPU-bound and many I/O-bound processes.

- What is the side-effect?
  - Results in **lower CPU and device utilization** than might be possible if the shorter **processes were allowed to go first.**

Further Explanation of the Side-effect

- From the source book (Silbershatz)

"In addition, consider the performance of FCFS scheduling in a dynamic situation. Assume we have one CPU-bound process and many I/O-bound processes. As the processes flow around the system, the following scenario may result. The CPU-bound process will get and hold the CPU. During this time, all the other processes will finish their I/O and will move into the ready queue, waiting for the CPU. While the processes wait in the ready queue, **the I/O devices are idle**. *…"*

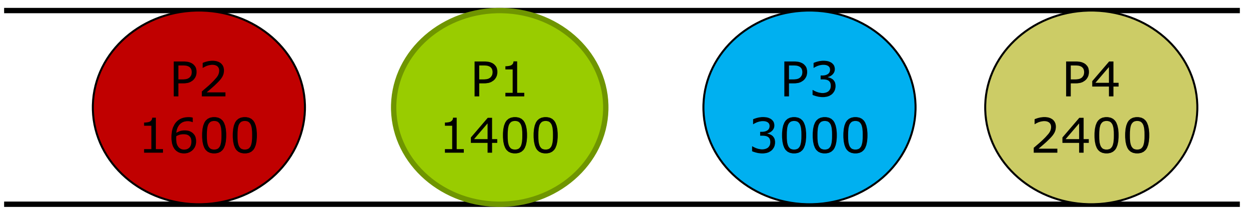
Further Explanation of the Side-effect

- *"Eventually, the CPU-bound process finishes its CPU burst and moves to an I/O device. All the I/O-bound processes, which* ***have short CPU*** *bursts, execute quickly and move back to the I/O queues. At this point, the* ***CPU sits idle****. The CPU-bound process will then move back to the ready queue and be allocated the CPU. Again, all the I/O processes end up waiting in the ready queue until the CPU-bound process is done. There is a* ***convoy effect*** *as all the other processes wait for the one big process* ***to get off the CPU****. This effect results in lower CPU and device utilization than might be possible if the shorter Processes were allowed to go first."*

Shortest-Job-First

Shortest-Job-First (SJF) Scheduling

- **Associate** with each process the length of its **next CPU burst**.
  - Use these lengths to schedule the process with the shortest time.



Ready Queue

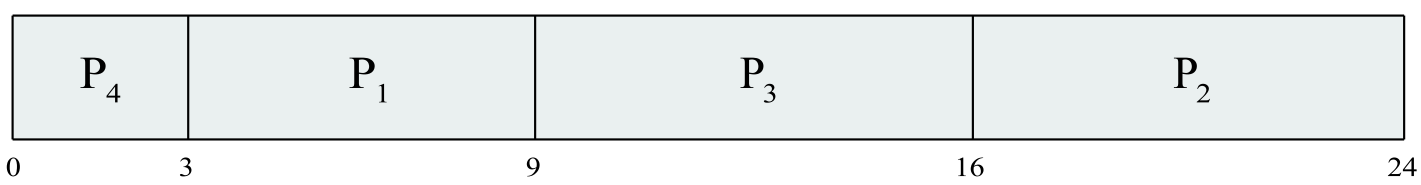
Shortest-Job-First (SJF) Scheduling (cont.)

- **SJF is optimal**
  - **Gives minimum average waiting time** for a given set of processes.
- Preemptive version called **shortest-remaining-time-first**
- **The difficulty is knowing the length of the next CPU request**
  - Could ask the user
  - Estimate (we do not cover this in the class and the exams)

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Overview

	FCFS	SJF	SRTF
Preemptive or Non-preemptive?			
Does it suit interactive systems?			
Is it possible to have starvation?			
Is it possible to apply priorities for different processes?			
Is it possible to implement it in practice?			

Overview

	FCFS	SJF	SRTF
Preemptive or Non-preemptive?	NP	NP	P
Does it suit interactive systems?	NO	Somehow	YES
Is it possible to have starvation?	NO	YES	YES
Is it possible to apply priorities for different processes?	NO	NO	NO
Is it possible to implement it in practice?	YES	NO	NO

Round-Robin

Round Robin (RR)

- **Each process gets a small unit of CPU time (time quantum q)**
  - Usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- **If there are n processes in the ready queue and the time quantum is q:**
  - Each process gets 1/n of the CPU time in chunks of at most *q* time units at once.
  - No process waits more than *(n-1)q* time units.

Round Robin (RR) (cont.)

- Timer *interrupts* every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

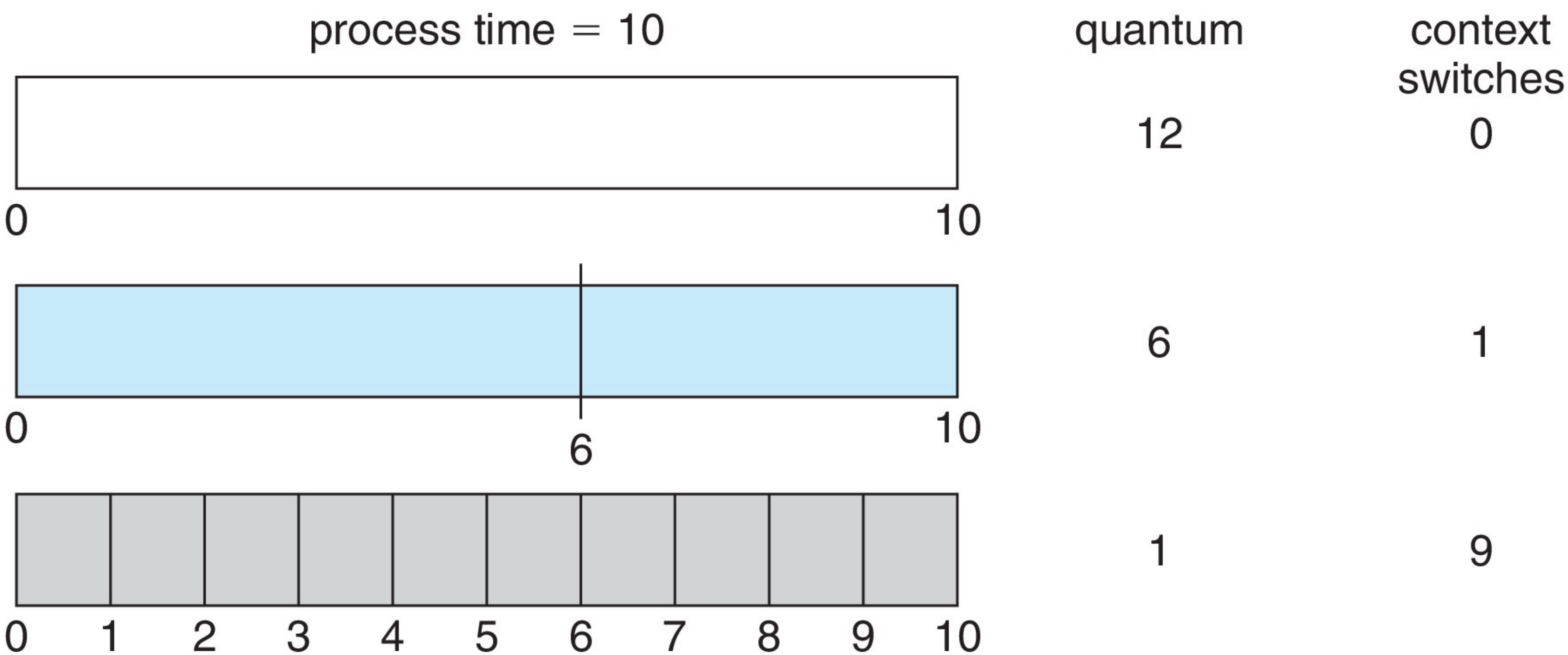
- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

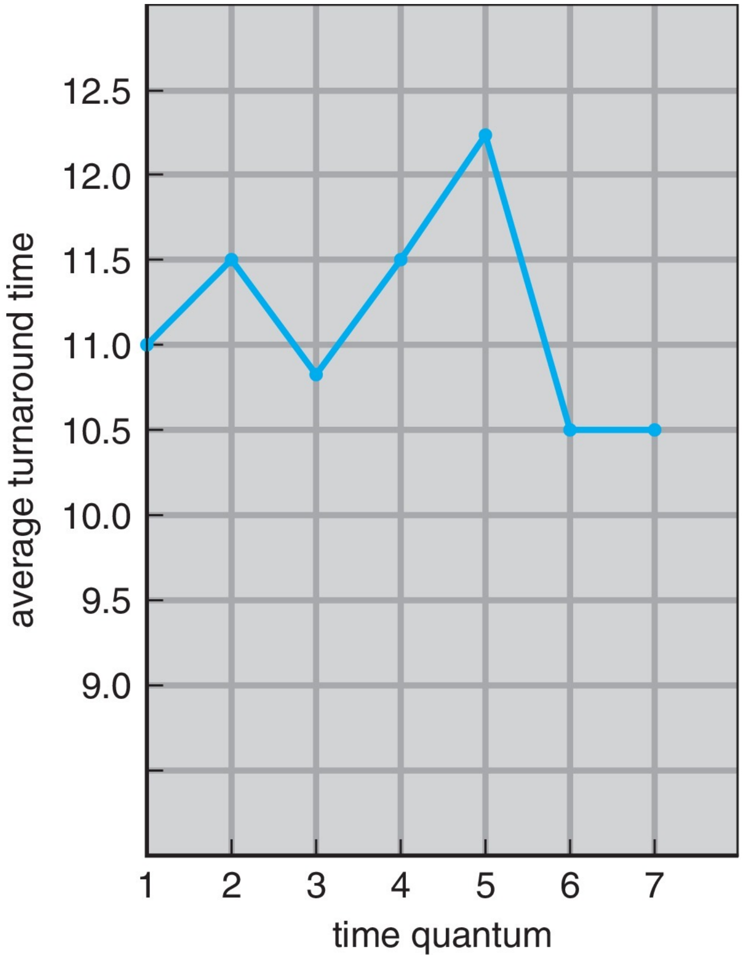
0 4 7 10 14 18 22 26 30

- Typically, higher average turnaround than SJF, but better ***response***
- *q* should be large compared to context switch time
  - **q** usually 10 milliseconds to 100 milliseconds,
  - Context switch < 10 microseconds

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

A rule of thump is that 80% of CPU bursts should be shorter than q

Priority Scheduling

Priority Scheduling

- A priority number (integer) is associated with each process
- **How OS/we decide on priorities?**

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (**smallest integer ≡ highest priority**)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the ***next predicted CPU burst time***

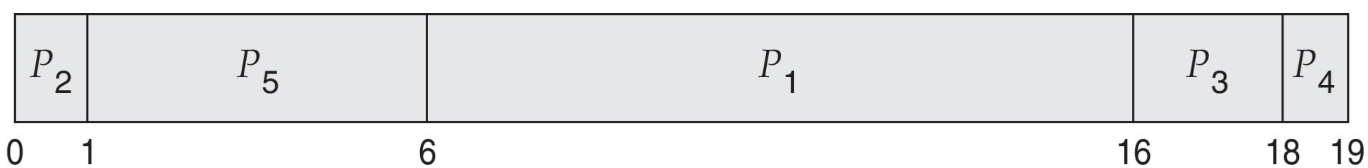
Priority Scheduling

- **Problem** ≡ **Starvation**
  - Low priority processes may never execute
- **Solution** ≡ **Aging**
  - As time progresses increase the priority of the process

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2

Priority Scheduling w/ Round-Robin

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes with the same priority run round-robin

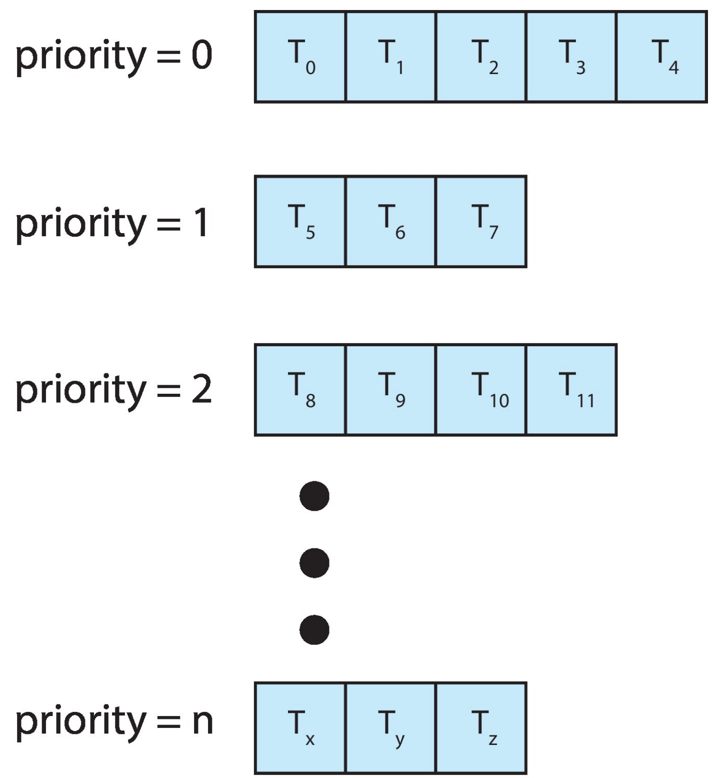
- Gantt Chart with time quantum = 2

P_4	P_2	P_3	P_2	P_3	P_2	P_3	P_1	P_5	P_1	P_5
0–7	7–9	9–11	11–13	13–15	15–16	16–20	20–22	22–24	24–26	26–27

Multilevel Queue Scheduling

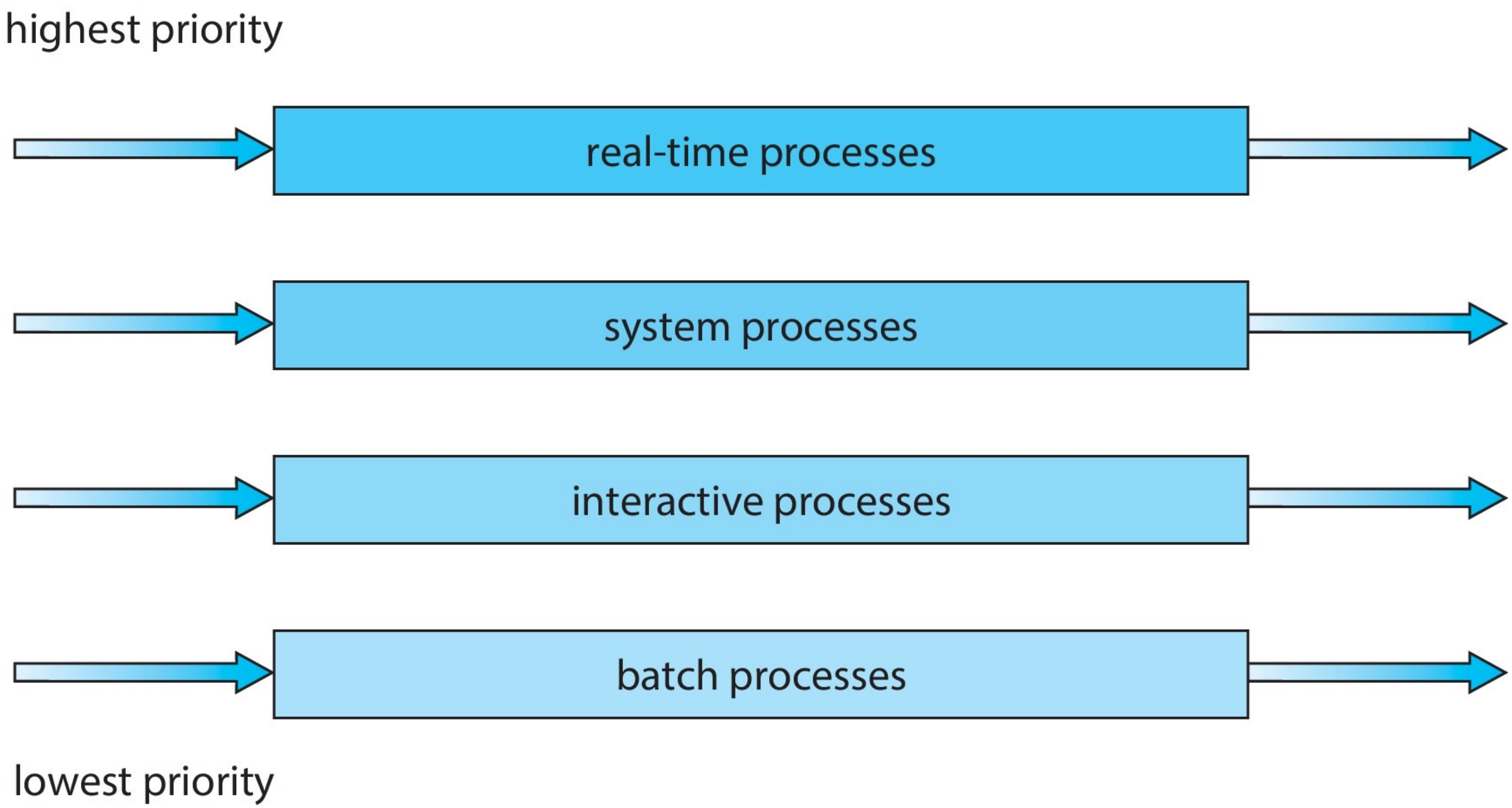
Multilevel Queue

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



Multilevel Queue

- Prioritization based upon process type



Multilevel Feedback Queue

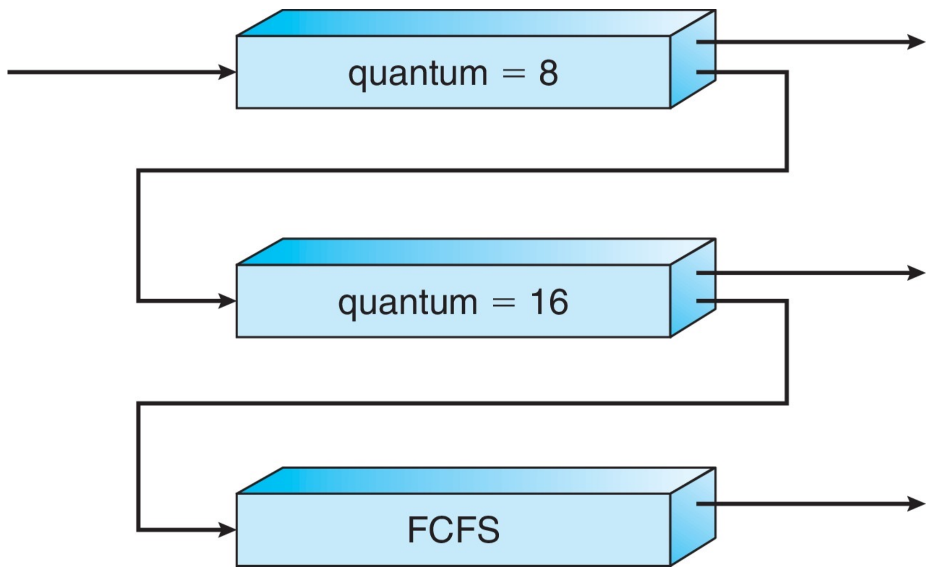
- A process can move between the various queues.

- Multilevel-feedback-queue defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service

- **Aging** can be implemented using multilevel feedback queue

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS



Example of Multilevel Feedback Queue (cont.)

- Scheduling
  - A new process enters queue Q_0 which is served in RR
    - When it gains CPU, the process receives 8 milliseconds
    - If it does not finish in 8 milliseconds, the process is moved to queue Q_1
  - At Q_1 job is again served in RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2