Operating System Design

Process Scheduling

Neda Nasiriani

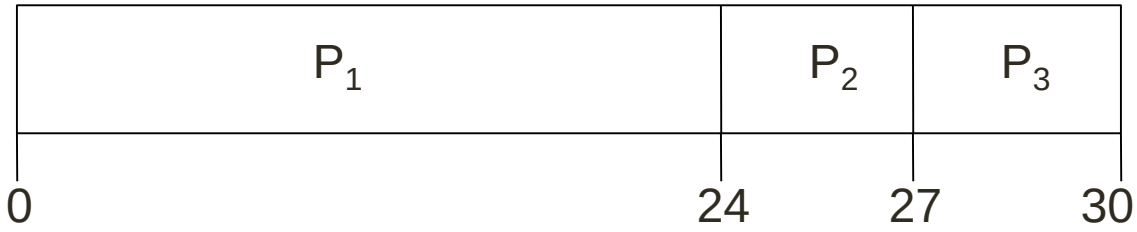
Fall 2018

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:



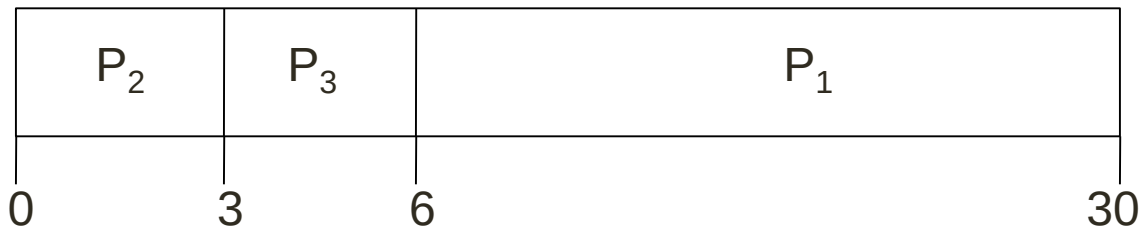
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

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)

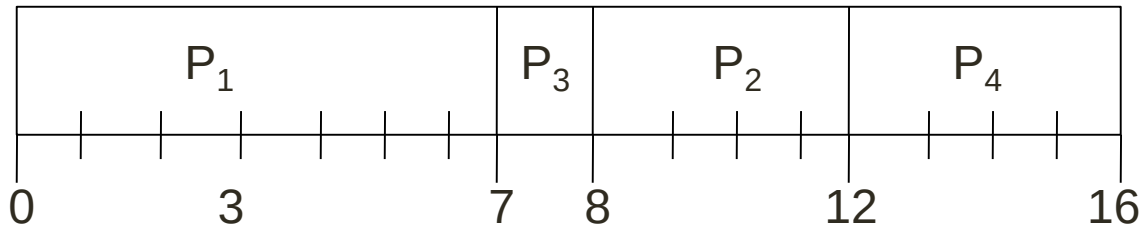
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - Preemptive – if a new process arrives with CPU burst 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)

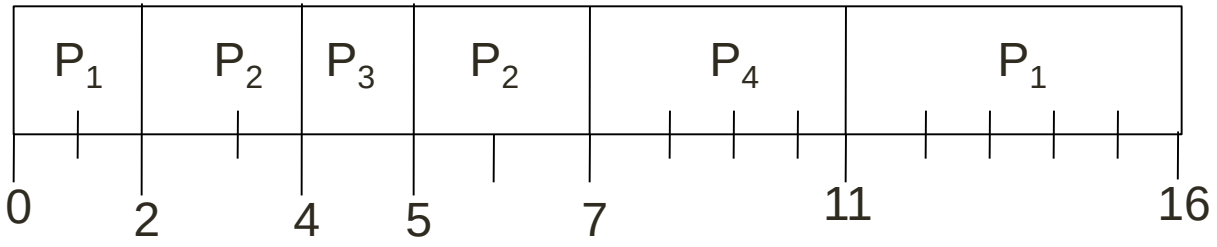


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Preemptive SJF: Shortest Remaining Time First

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 the Next CPU-Burst

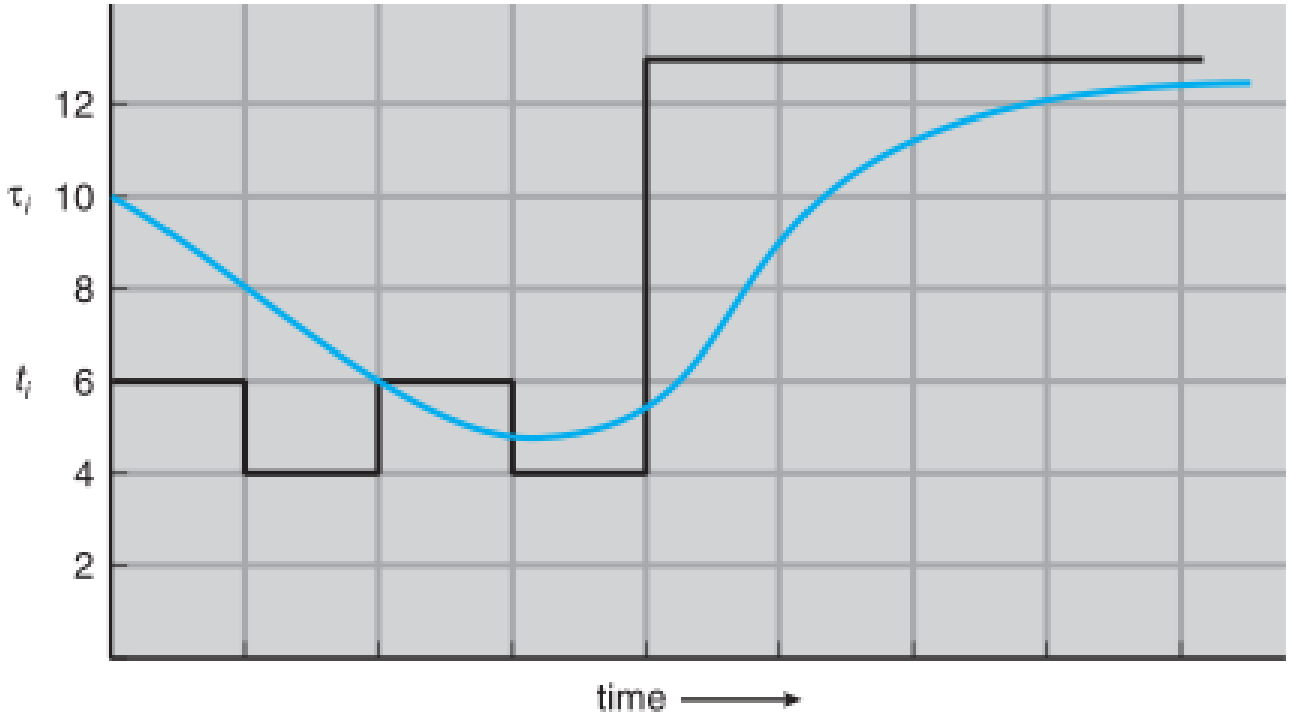
- We can only *estimate* the length.
- This can be done by using the length of previous CPU bursts, using exponential averaging:

$$t_n = \text{actual length of the } n^{th} \text{ CPU burst}$$

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

$$0 \leq \alpha \leq 1$$

Prediction of the Length of the Next CPU-Burst



i	0	1	2	3	4	5	6	7	
CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Priority Scheduling

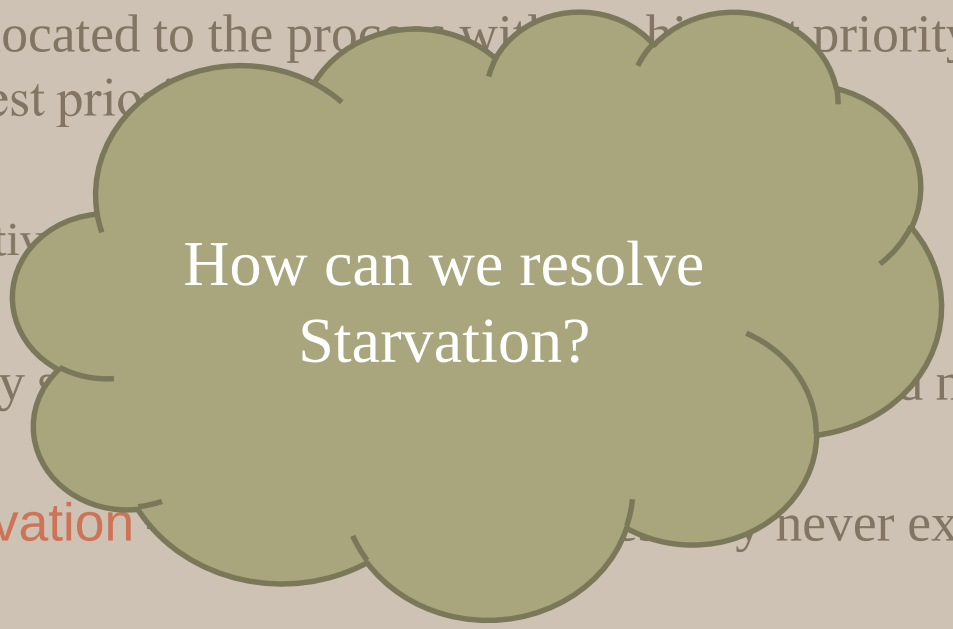
- A priority number (integer) is associated with each process.

- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive

- SJF is a priority scheduling where priority is the predicted next CPU-burst time.
- Problem: Starvation – low priority processes may never execute.

Priority Scheduling

- A priority number (integer) is associated with each process.
- The CPU is allocated to the proc[illegible] priority (smallest integer ≡ highest prio[illegible]
  - Preemptive
  - Nonpreemptiv[illegible]
- SJF is a priority [illegible] next CPU-burst time.
- Problem: Starvation [illegible] never execute.



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 a priority scheduling where priority is the predicted next CPU-burst time.
- Problem: Starvation – low priority processes may never execute.
- Solution: Aging – as time progresses increase the priority of the process.

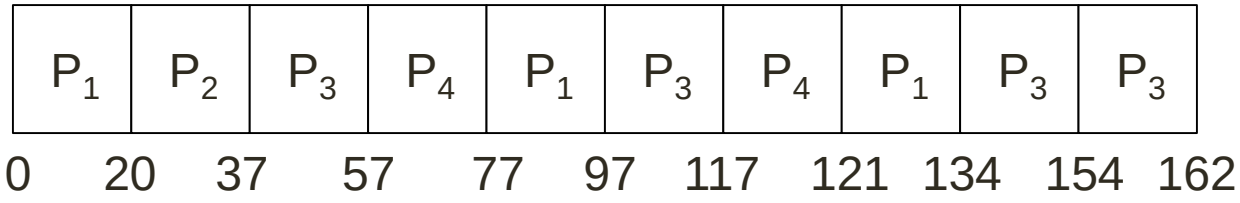
Round Robin (RR)

- Each process gets a small unit of CPU time (time ***quantum***), 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*, then 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.
- Performance:
  - *q* large
    - FCFS.
  - *q* small
    - *q* must be large with respect to context switch, otherwise overhead is too high.

RR with Time Quantum = 20

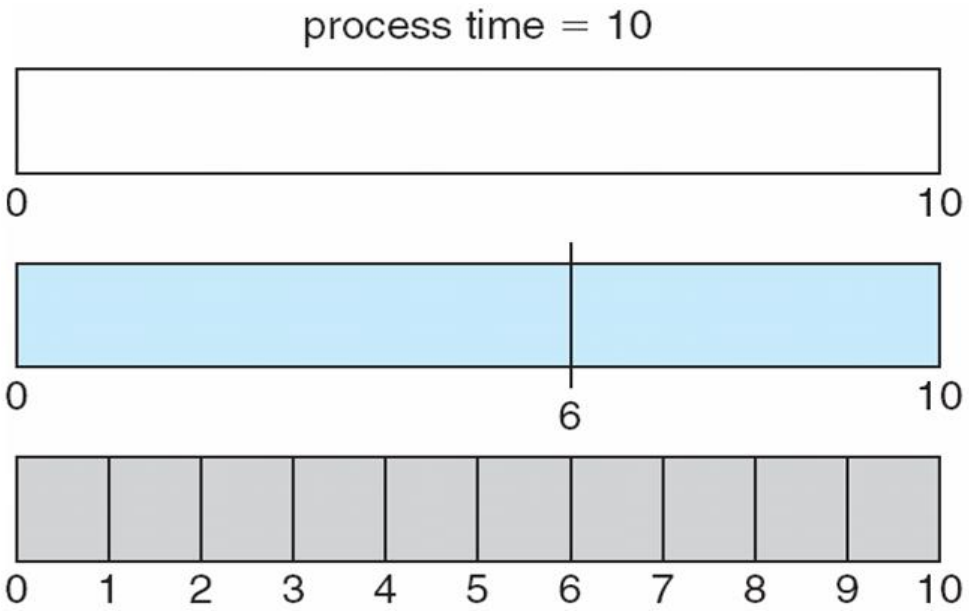
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response*.

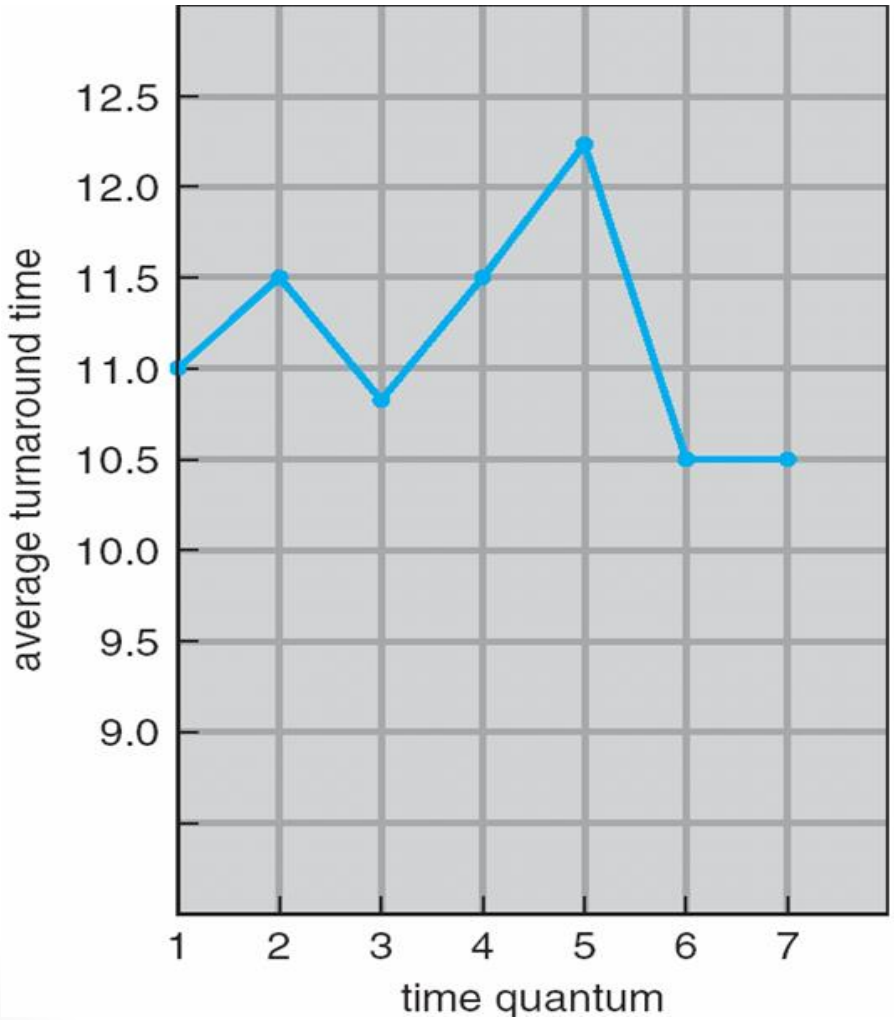
Time Quantum and Context Switch Time



quantum	context switches
12	0
6	1
1	9

Question: What considerations influence the choice of value for the quantum?

Turnaround Time Varies with the Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

What Scheduling to use?

- In a real system, you have an array of different processes with different natures
  - System Backup
  - Web browser
  - Music player
  - Updates
  - …
- How should we schedule all these processes, using the scheduling techniques that we have seen so far?
  - FCFS?
  - SJF?
  - SRJT?

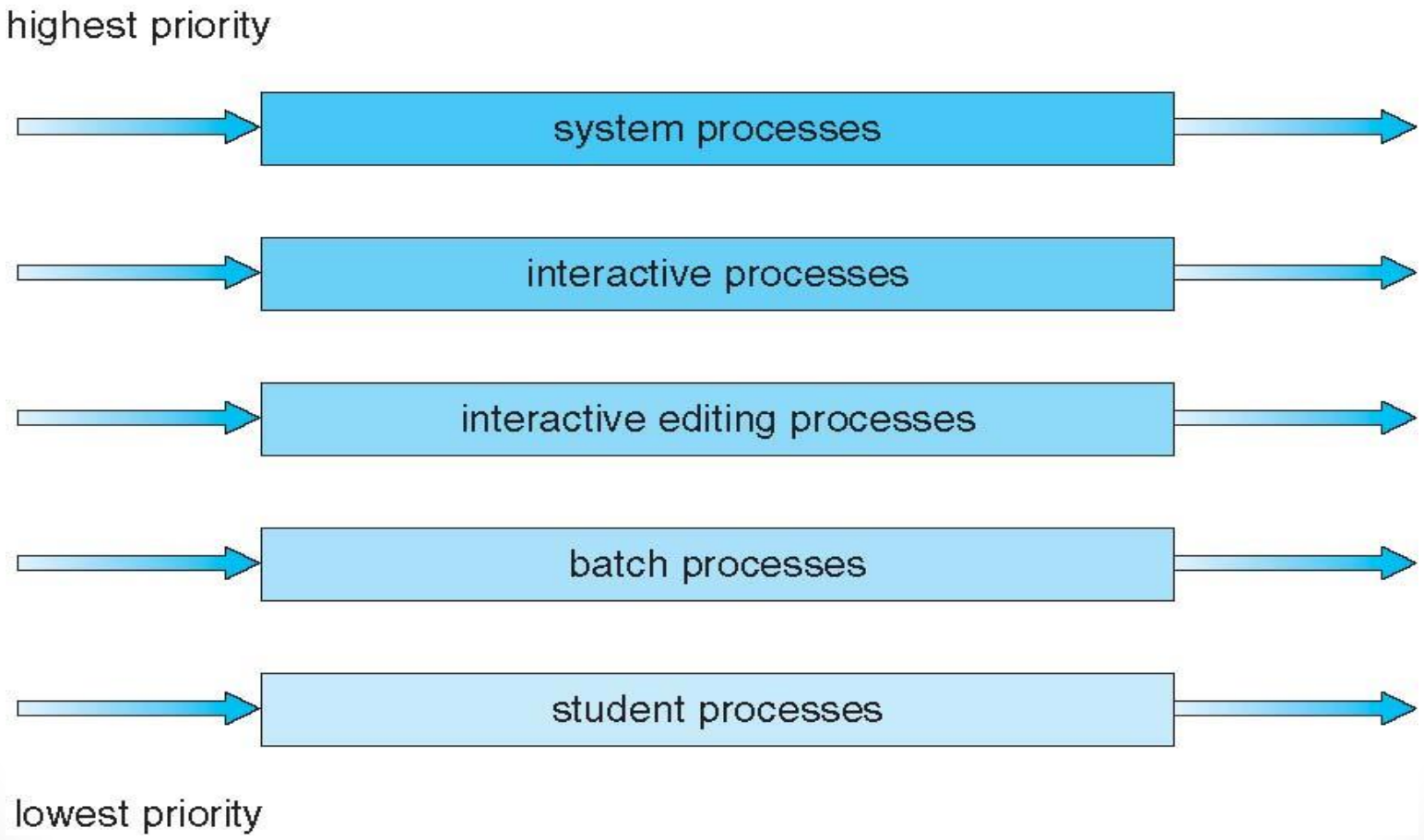
Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)

- Each queue has its own scheduling algorithm.
  - foreground: RR
  - background: FCFS

- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR.
  - 20% to background in FCFS .

Multilevel Queue Scheduling



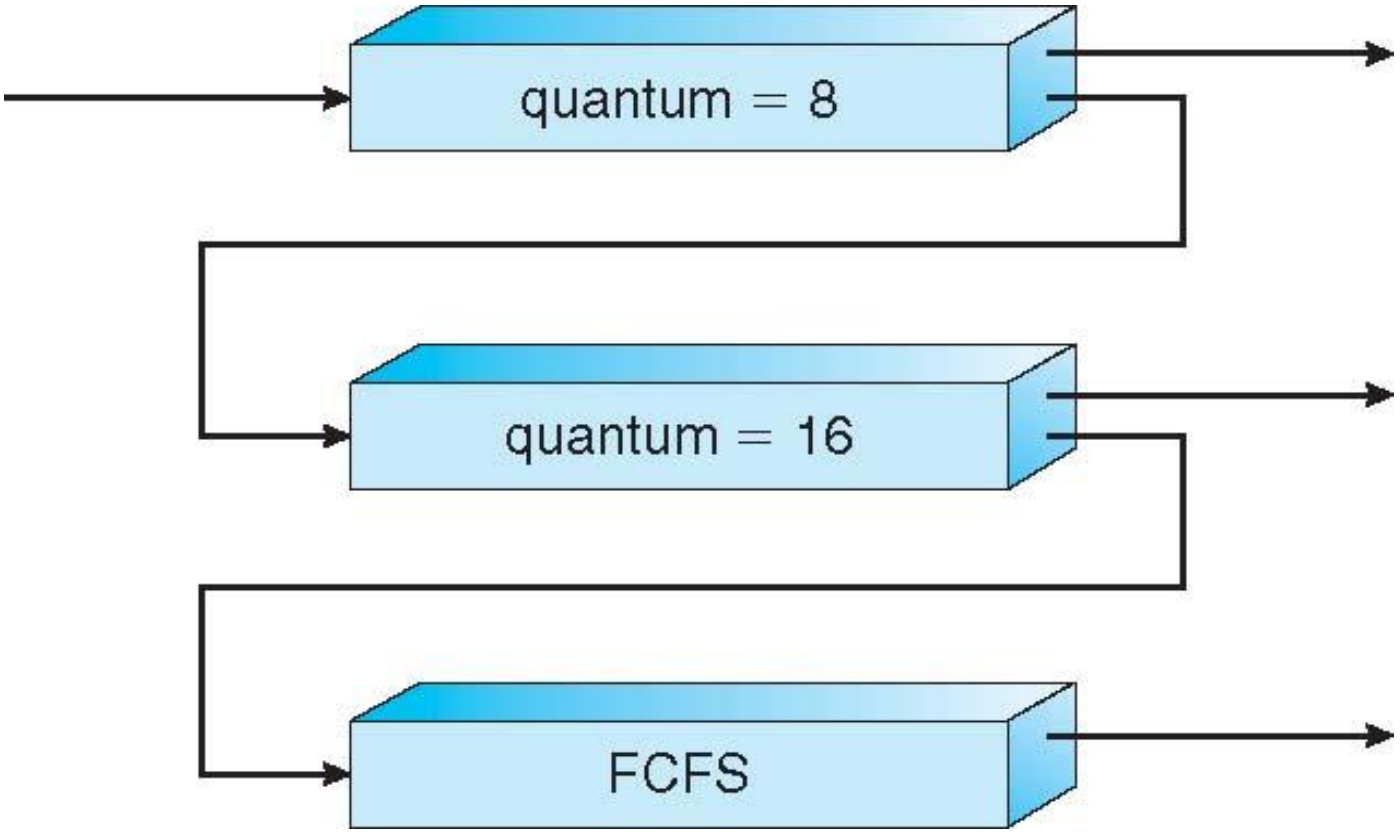
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.

- Multilevel-feedback-queue scheduler 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.

Example of Multilevel Feedback Queue

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

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

Multilevel Feedback Queues