

# CPU Scheduling

# Chapter 6: CPU Scheduling

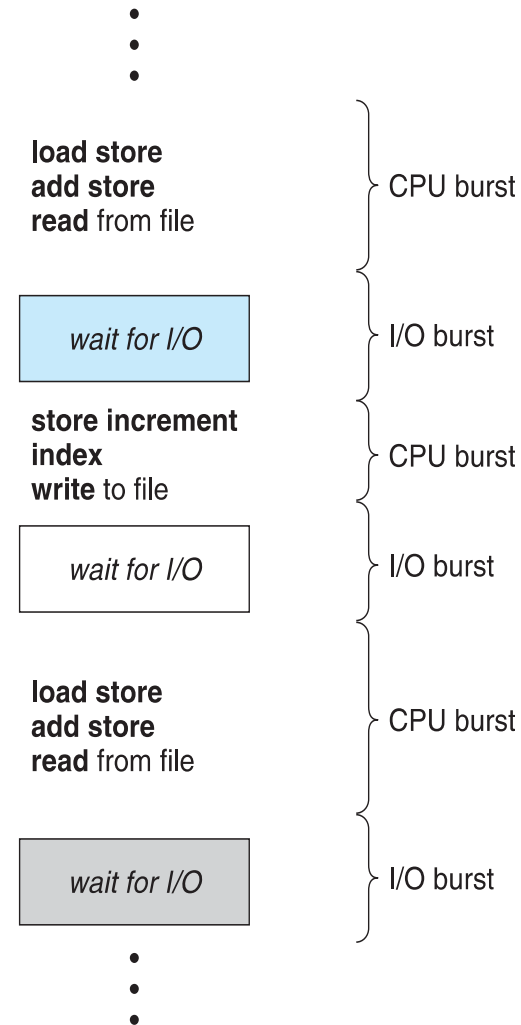
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

# Objectives

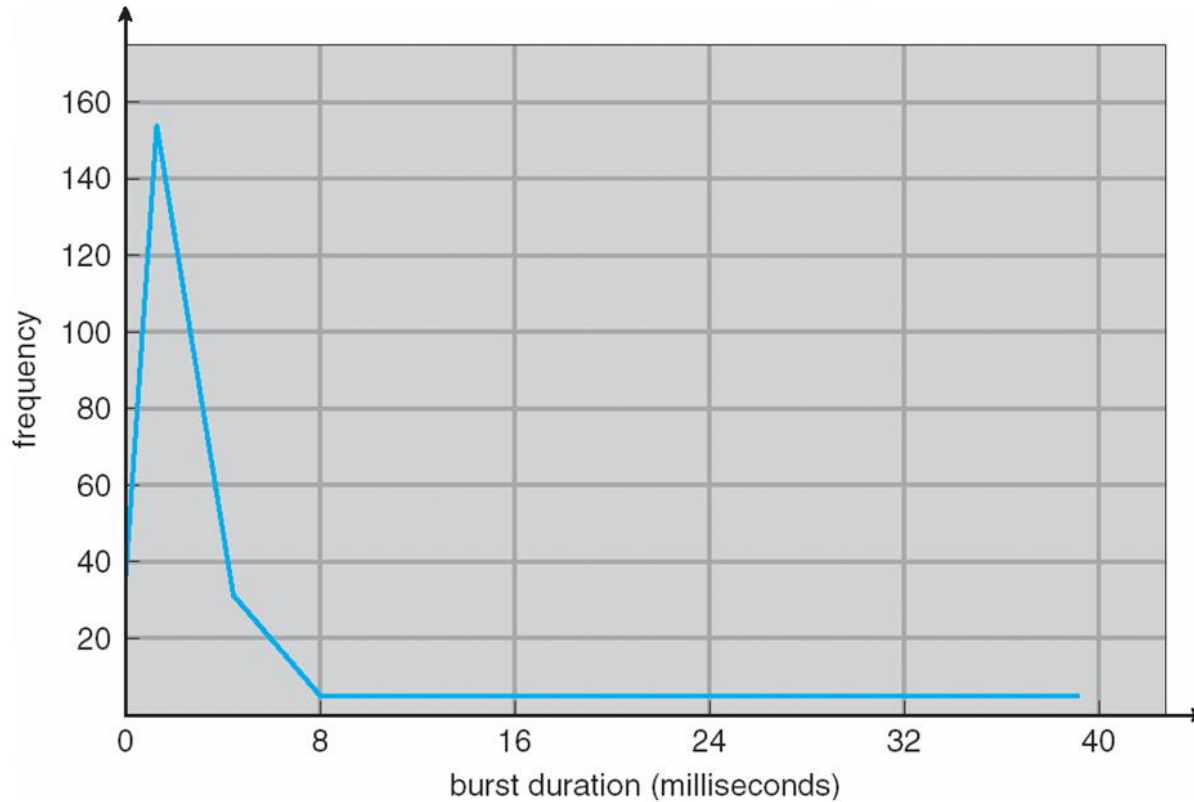
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

# Basic Concepts

- 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

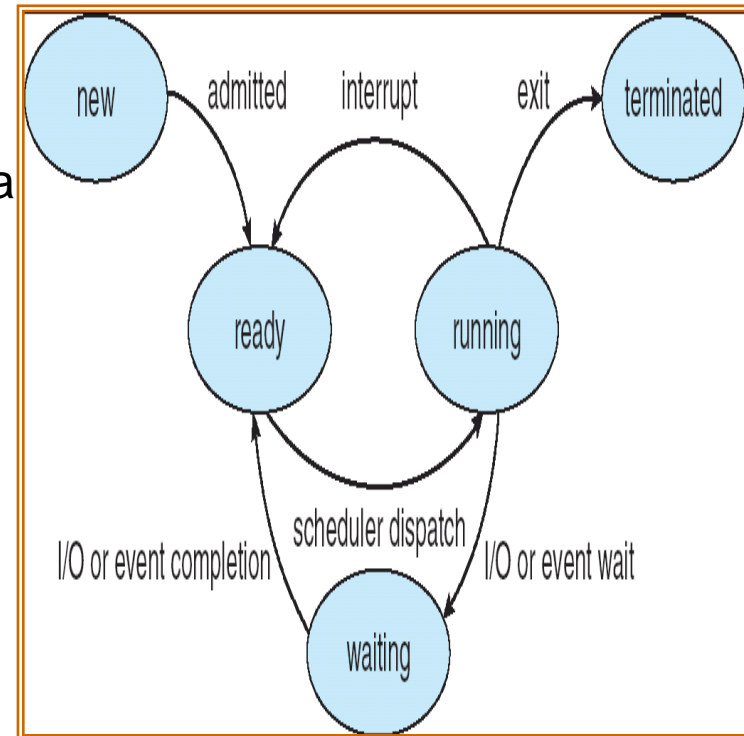


# Histogram of CPU-burst Times



# 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



# 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
- **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

<u>Process</u>	<u>Burst Time</u>
$P_1$	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

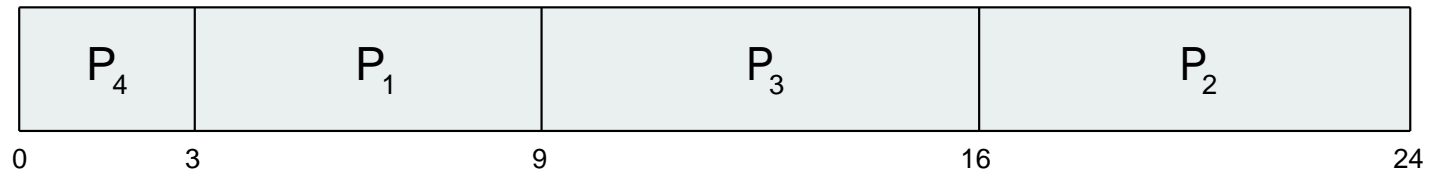
# Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

# Example of SJF

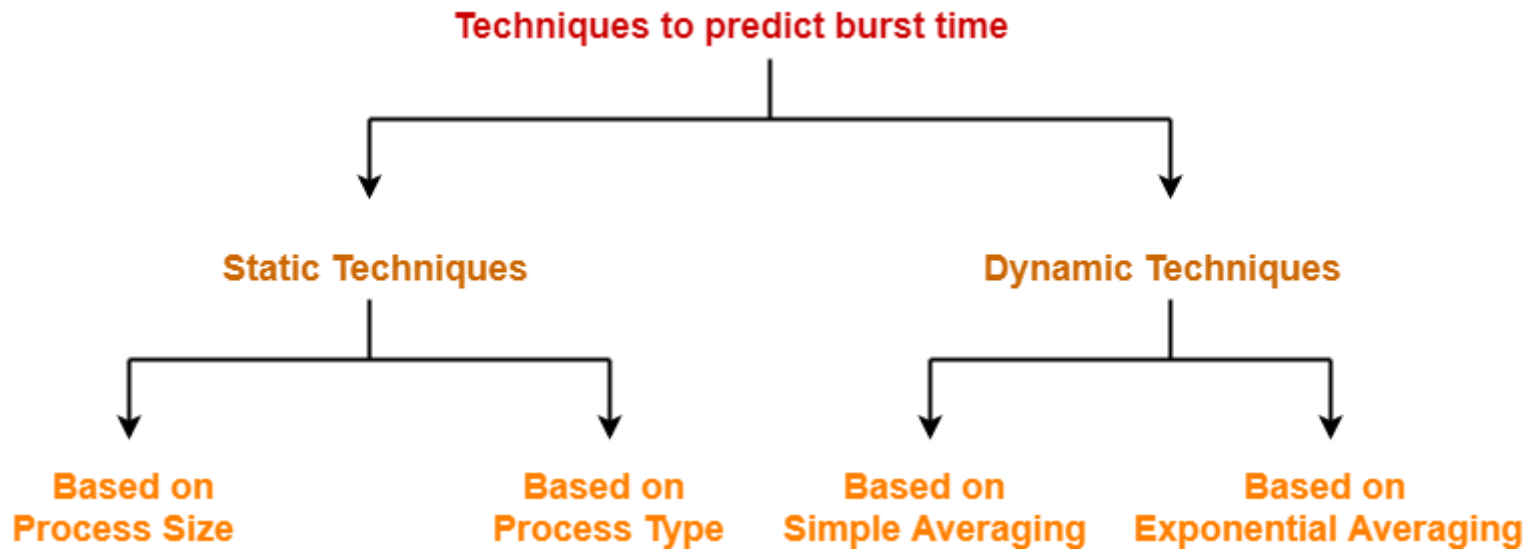
<u>Process</u>	<u>Burst Time</u>
$P_1$	6
$P_2$	8
$P_3$	7
$P_4$	3

## ■ SJF scheduling chart



## ■ Average waiting time = $(3 + 16 + 9 + 0) / 4 = 7$

# Techniques to predict next burst time



Process size based : Eg: Consider a process of size 200 KB took 20 units of time to complete its execution. Then, burst time for any future process having size around 200 KB can be taken as 20 units.

Process type can be either Operating system or User process  
User process can be interactive, foreground or background

# Dynamic techniques

## ■ Simple averaging

Burst time for the process to be executed is taken as the average of all the processes that are executed till now.

Given  $n$  processes  $P_1, P_2, \dots, P_n$  and burst time of each process  $P_i$  as  $t_i$ , then predicted burst time for process  $P_{n+1}$  is given as-

$$T_{n+1} = \frac{1}{n} \sum_{i=1}^n t_i$$

## ■ Exponential averaging

$$T_{n+1} = \alpha t_n + (1 - \alpha) T_n$$

where-

$\alpha$  is called smoothing factor ( $0 \leq \alpha \leq 1$ )

$t_n$  = actual burst time of process  $P_n$

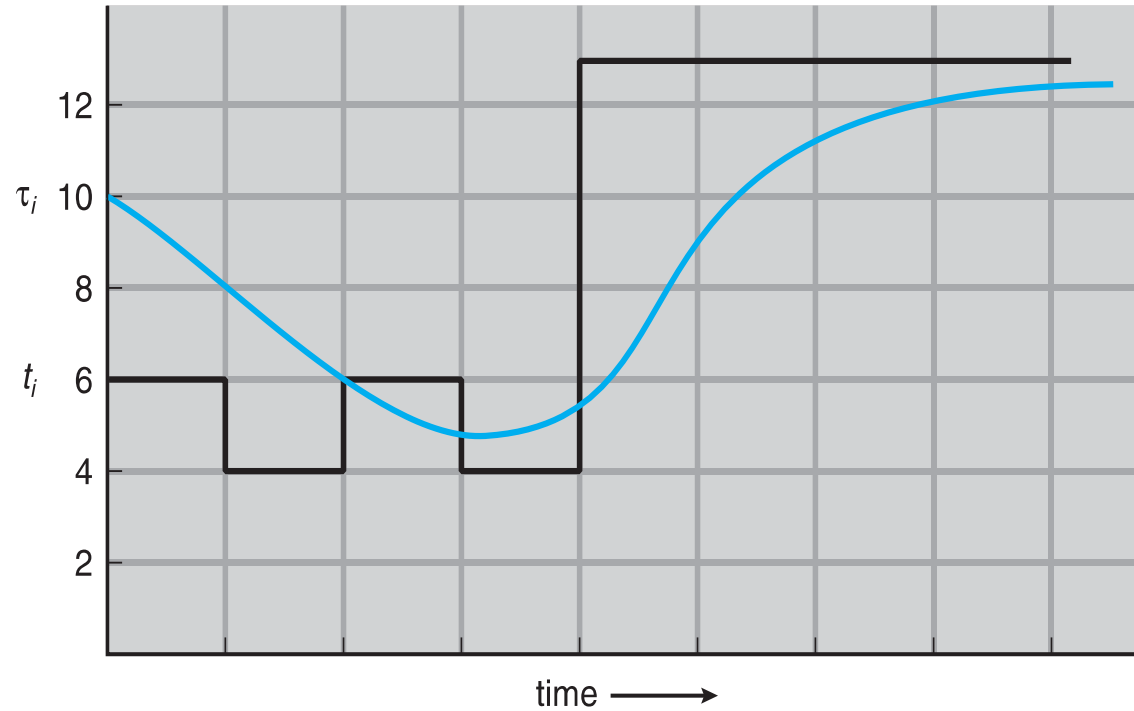
$T_n$  = Predicted burst time for process  $P_n$

# Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
  1.  $t_n$  = actual length of  $n^{th}$  CPU burst
  2.  $\tau_{n+1}$  = predicted value for the next CPU burst
  3.  $\alpha, 0 \leq \alpha \leq 1$
  4. Define:  $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$ .
- Commonly,  $\alpha$  set to  $\frac{1}{2}$
- Preemptive version called **shortest-remaining-time-first**



# Prediction of the Length of the Next CPU Burst



CPU burst ( $t_i$ )	6	4	6	4	13	13	13	...
"guess" ( $\tau_i$ )	10	8	6	6	9	11	12	...

# Examples of Exponential Averaging

## ■ $\alpha = 0$

- $\tau_{n+1} = \tau_n$
- Recent history does not count

## ■ $\alpha = 1$

- $\tau_{n+1} = \alpha t_n$
- Only the actual last CPU burst counts

## ■ If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} = & \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots \\ & + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- ## ■ Since both $\alpha$ and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

- Calculate the predicted burst time using exponential averaging for the fifth process if the predicted burst time for the first process is 10 units and actual burst time of the first four processes is 4, 8, 6 and 7 units respectively.  
Given  $\alpha = 0.5$ .

Exponential averaging

$$T_{n+1} = \alpha t_n + (1 - \alpha) T_n$$

where-

$\alpha$  is called smoothing factor ( $0 \leq \alpha \leq 1$ )

$t_n$  = actual burst time of process  $P_n$

$T_n$  = Predicted burst time for process  $P_n$

Given-

Predicted burst time for 1st process = 10 units

Actual burst time of the first four processes = 4, 8, 6, 7

$\alpha = 0.5$

Predicted burst time for 2nd process

$= \alpha \times \text{Actual burst time of 1st process} + (1-\alpha) \times \text{Predicted burst time for 1st process}$

$= 0.5 \times 4 + 0.5 \times 10$

$= 2 + 5$

$= 7 \text{ units}$

Predicted burst time for 3rd process

$= \alpha \times \text{Actual burst time of 2nd process} + (1-\alpha) \times \text{Predicted burst time for 2nd process}$

$= 0.5 \times 8 + 0.5 \times 7$

$= 4 + 3.5$

$= 7.5 \text{ units}$

Predicted burst time for 4th process

$= \alpha \times \text{Actual burst time of 3rd process} + (1-\alpha) \times \text{Predicted burst time for 3rd process}$

$= 0.5 \times 6 + 0.5 \times 7.5$

$= 3 + 3.75$

$= 6.75 \text{ units}$

Predicted Burst Time for 5th Process-

Predicted burst time for 5th process

$= \alpha \times \text{Actual burst time of 4th process} + (1-\alpha) \times \text{Predicted burst time for 4th process}$

$= 0.5 \times 7 + 0.5 \times 6.75$

$= 3.5 + 3.375$

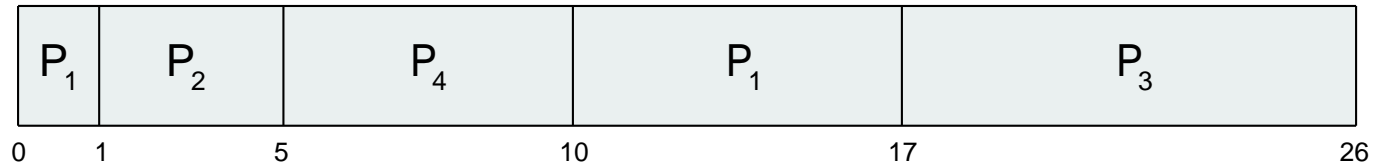
$= 6.875 \text{ units}$

# Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0	8
$P_2$	1	4
$P_3$	2	9
$P_4$	3	5

- Preemptive* SJF Gantt Chart



- Average waiting time =  $[(10-1)+(1-1)+(17-2)+5-3]/4 = 26/4 = 6.5$  msec

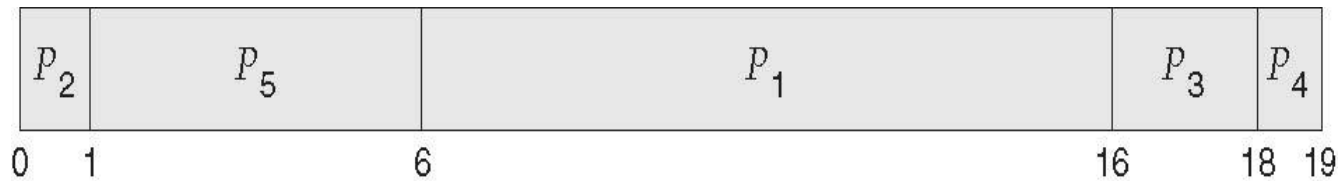
# Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer  $\equiv$  highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem  $\equiv$  **Starvation** – low priority processes may never execute
- Solution  $\equiv$  **Aging** – as time progresses increase the priority of the process

# Example of Priority Scheduling

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
$P_1$	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 msec

# Round Robin (RR)

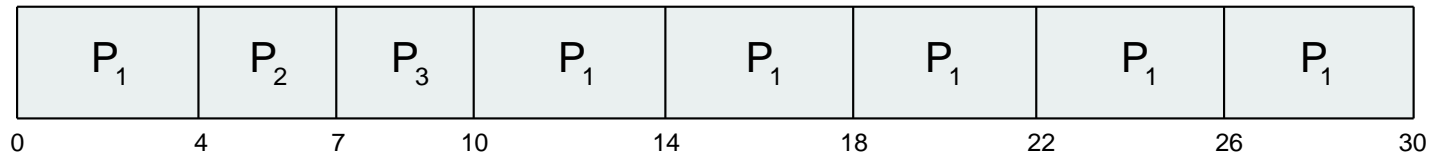
- Each process gets a small unit of CPU time (**time quantum**  $q$ ), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are  $n$  processes in the ready queue and the time quantum is  $q$ , then each process gets  $1/n$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

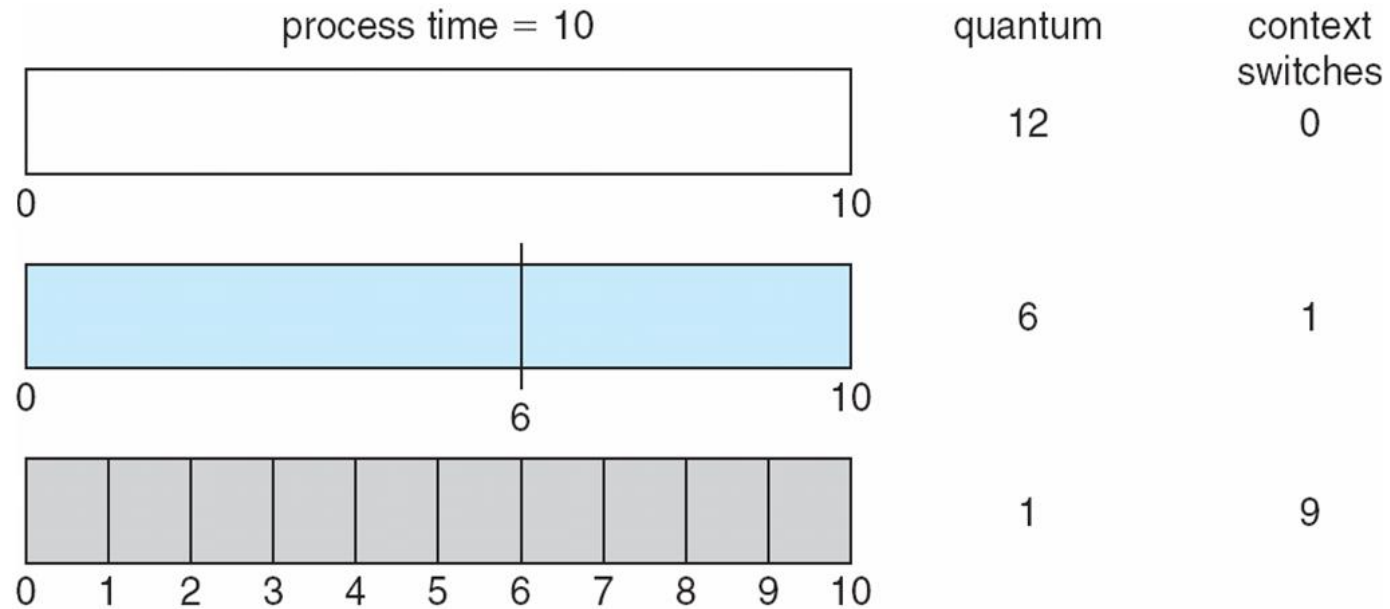
<u>Process</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

- The Gantt chart is:

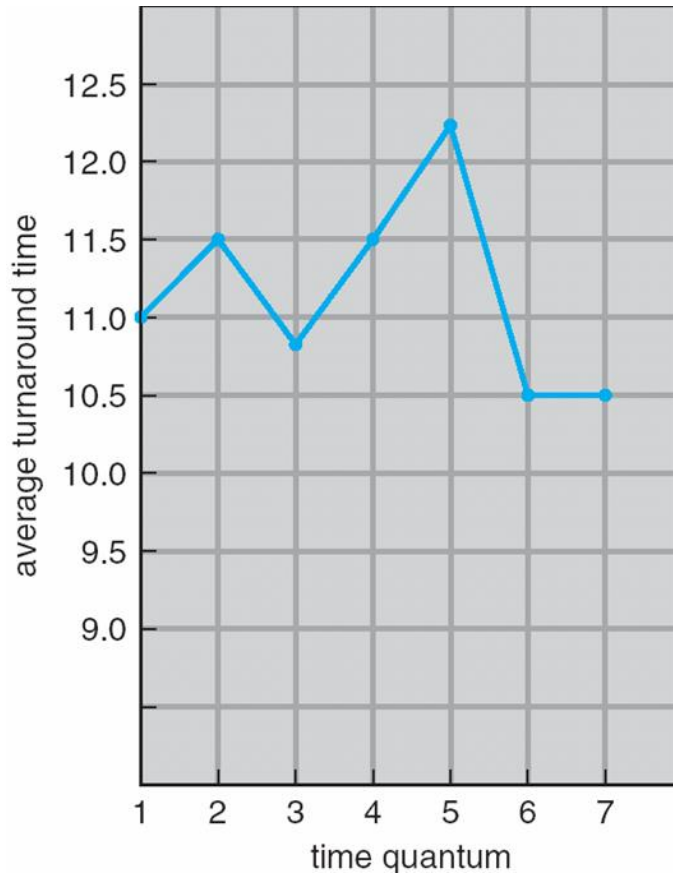


- Typically, higher average turnaround than SJF, but better **response**
- $q$  should be large compared to context switch time
- $q$  usually 10ms to 100ms, context switch < 10 usec

# 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

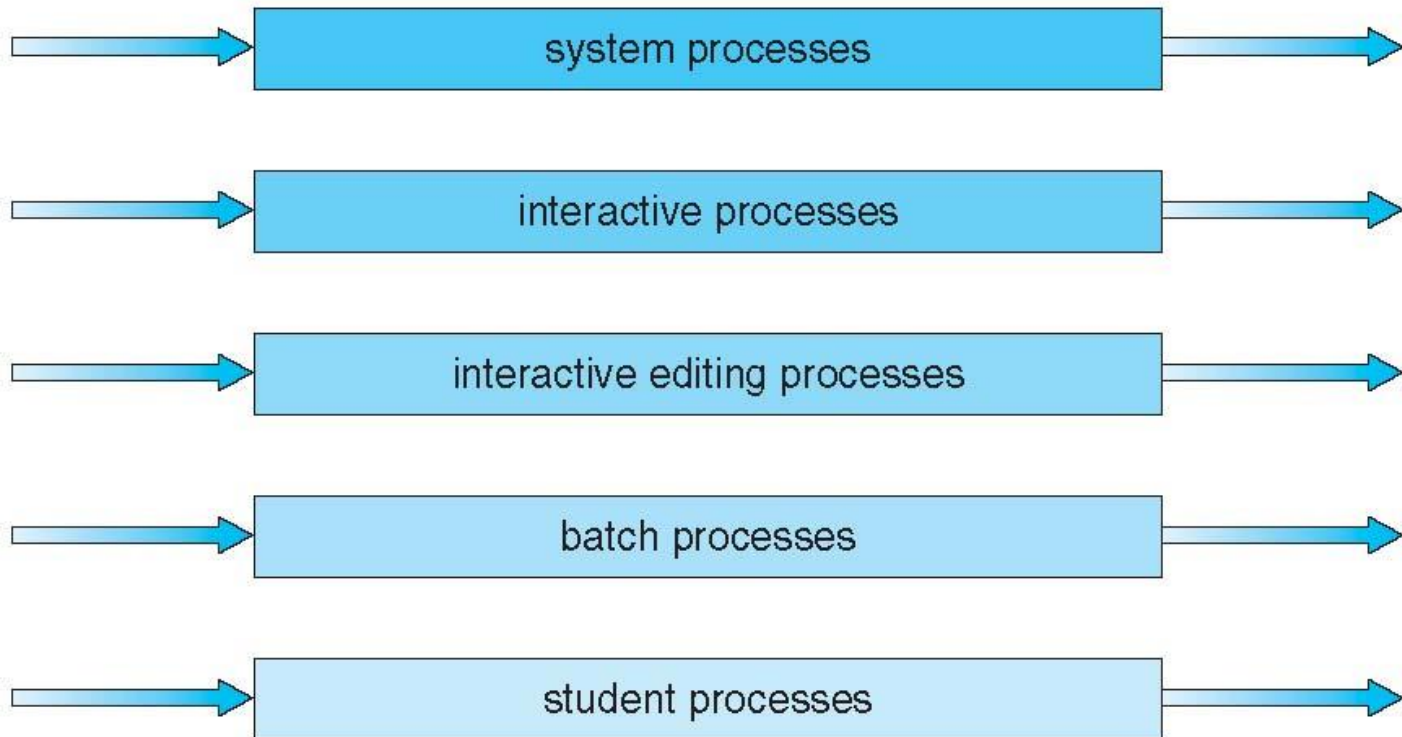
80% of CPU bursts  
should be shorter than  $q$

# Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- 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

highest priority



lowest priority

# 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

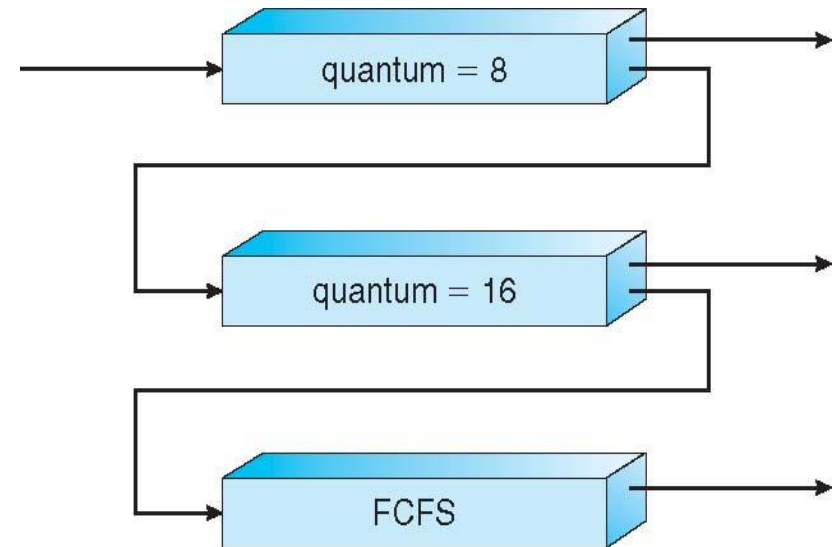
# Example of Multilevel Feedback Queue

## ■ Three queues:

- $Q_0$  – RR with time quantum 8 milliseconds
- $Q_1$  – RR 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  $Q_2$



# Tutorial Question

- Draw the Gantt Chart and schedule the processes using non-preemptive priority and RR algorithm with quantum 2msec. Calculate the waiting time and turn around time in each case.

Process	Burst Time	Priority
P1	10	2
P2	01	1
P3	02	2
P4	01	4
P5	05	3

Nonpreemptive priority:

Average waiting time = TAT=

RR with  $q=2\text{msec}$

Average waiting time = TAT=