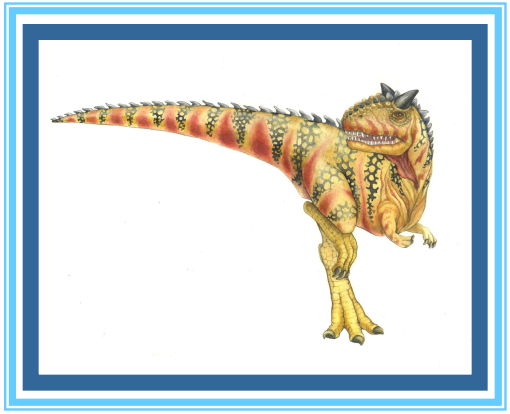
# Chapter 6: CPU Scheduling



# Previous Lecture

## Process Synchronization

- Background
- The Critical-Section Problem
- Peterson's Solution
- Synchronization Hardware
- Mutex Locks
- Semaphores
- Classic Problems of Synchronization
- Monitors
- Synchronization Examples
- Alternative Approaches

# Chapter 6: CPU Scheduling

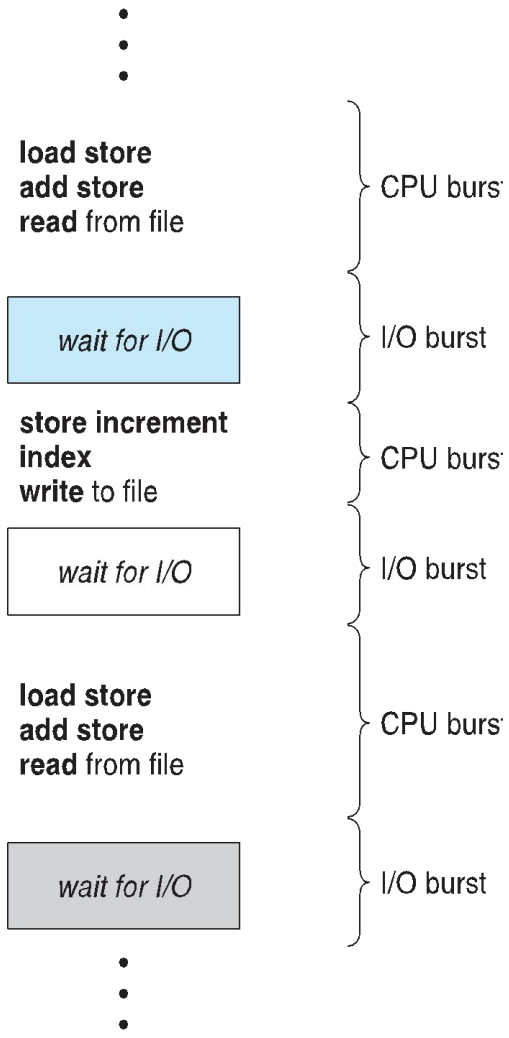
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling

# Objectives

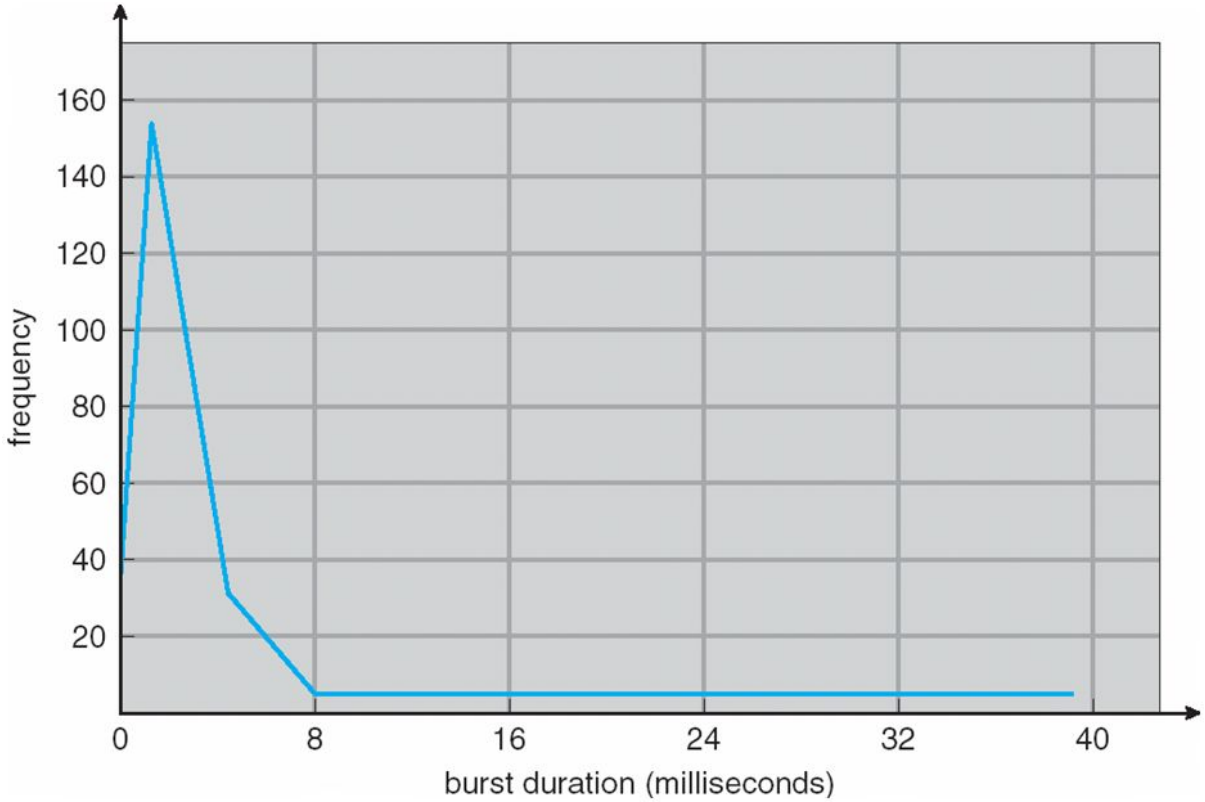
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

# 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

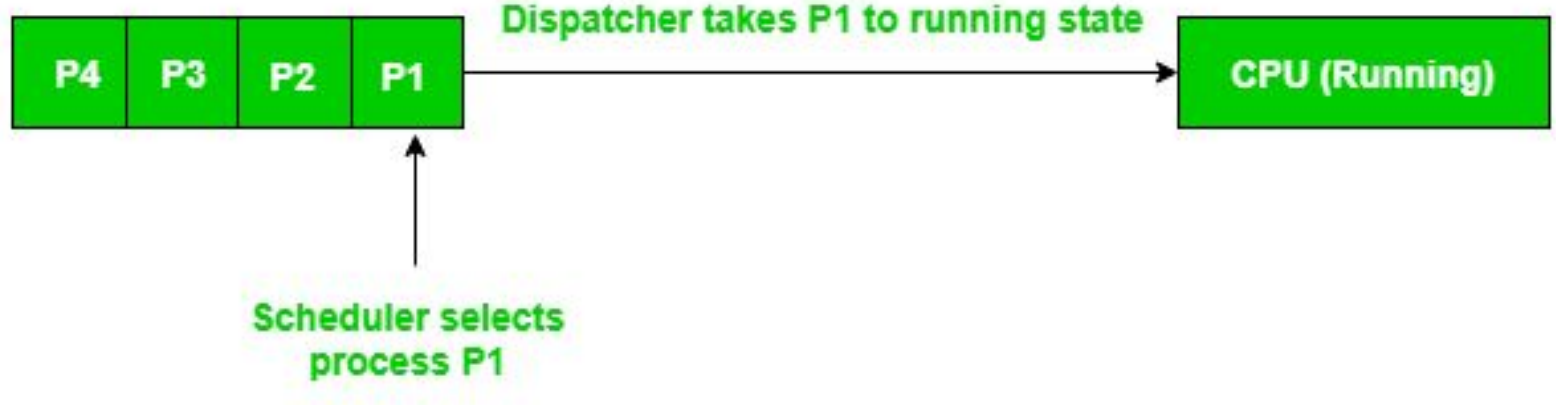



# CPU Scheduler

- **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 **non-preemptive**
- 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

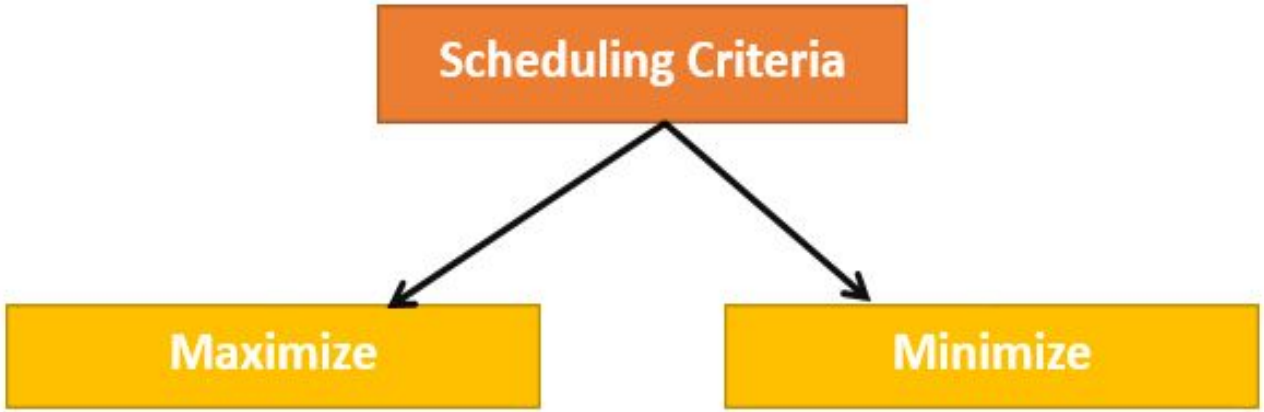



# Scheduling Criteria

- **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 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

- A CPU scheduling algorithm tries to maximize and minimize the following:




Maximize:
CPU Utilization
Throughput

Minimize:
Turnaround time
Waiting time
Response time

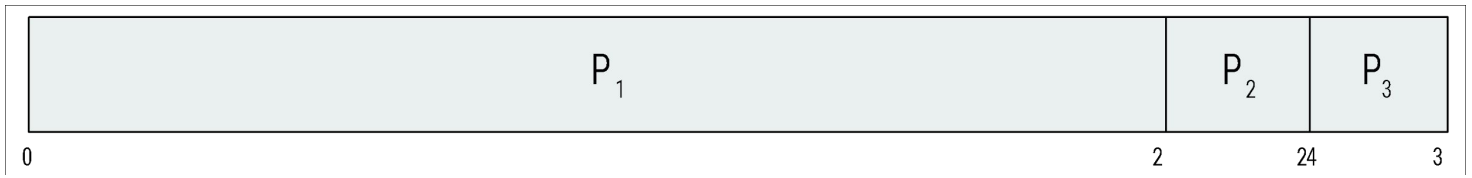
# First- Come, First-Served (FCFS) Scheduling

- **First come first serve** (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time.
- The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU.
- FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

# 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:

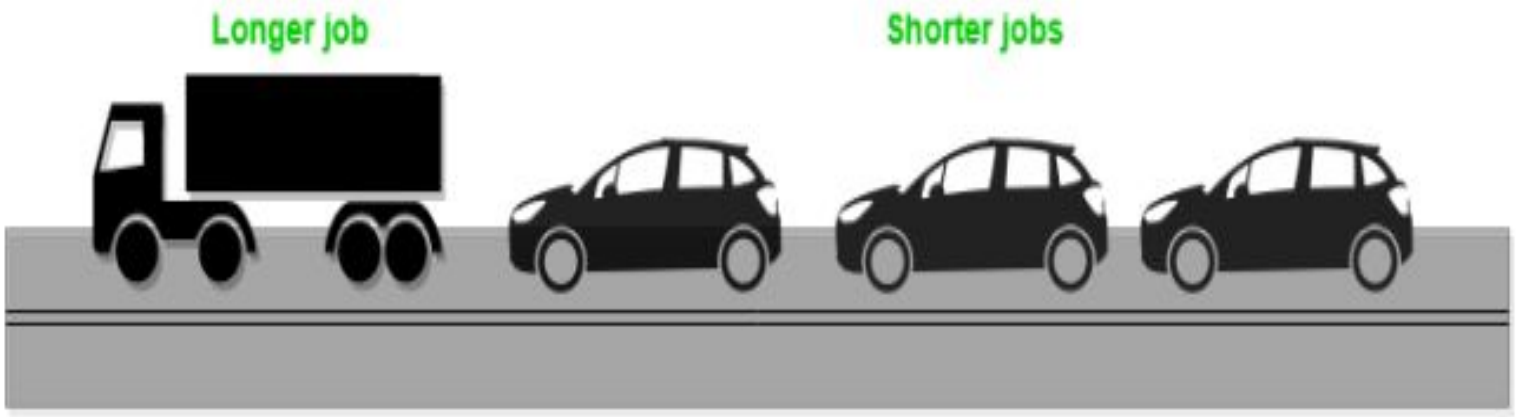
$P_2$ , $P_3$ , $P_1$

- 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

# Convey Effect

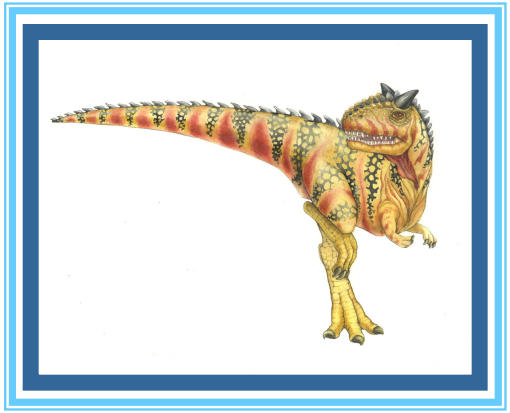
- Convoy Effect is phenomenon associated with the First Come First Serve (FCFS) algorithm, in which the whole Operating System slows down due to few slow processes.



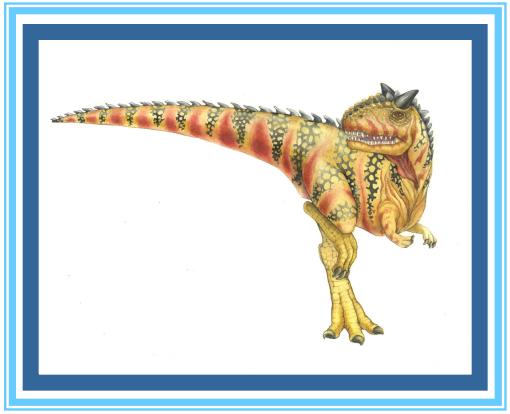

- FCFS algorithm is non-preemptive in nature, that is, once CPU time has been allocated to a process, other processes can get CPU time only after the current process has finished. This property of FCFS scheduling leads to the situation called Convoy Effect.

# FCFS Advantages and Disadvantages

- **Advantages**
  - It is simple and easy to understand.
- **Disadvantages**
  - The process with less execution time suffer i.e. waiting time is often quite long.
  - Favors CPU Bound process then I/O bound process.
  - Here, first process will get the CPU first, other processes can get CPU only after the current process has finished its execution. Now, suppose the first process has large burst time, and other processes have less burst time, then the processes will have to wait more unnecessarily, this will result in *more average waiting time*, i.e., Convey effect.
  - This effect results in lower CPU and device utilization.
  - FCFS algorithm is particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.



# Chapter 6: CPU Scheduling
# Lecture: 08



# Previous Lecture

- Basic Concepts
- Histogram of CPU-burst Times
- CPU Schedular
- Dispatcher
- Scheduling Criteria
- Scheduling Algorithm Optimization Criteria
- FCFS Scheduling
- Convey Effect

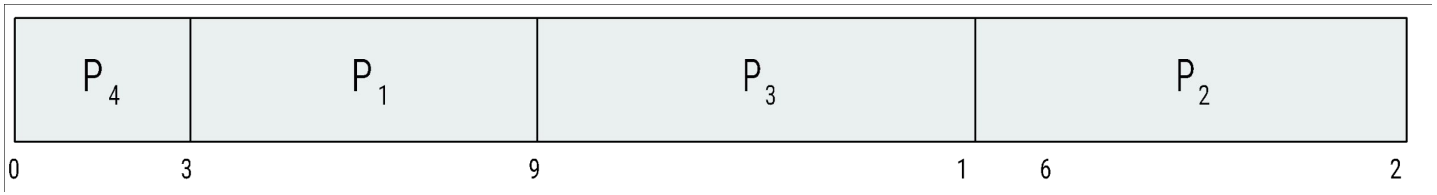
# Shortest-Job-First (SJF) Scheduling

- Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive/preemptive algorithm.
- **Algorithm:**
  - Sort all the process according to the arrival time.
  - Then select that process which has minimum arrival time and minimum Burst time.
  - After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

# 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

# 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. $\alpha, 0 \le \alpha \le 1$
  4. Define : $\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n$.

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**

# Prediction of the Length of the Next CPU Burst

- Calculate the exponential averaging with T1 = 10, α = 0.5 and the algorithm is SJF with previous runs as 8, 7, 4, 16.

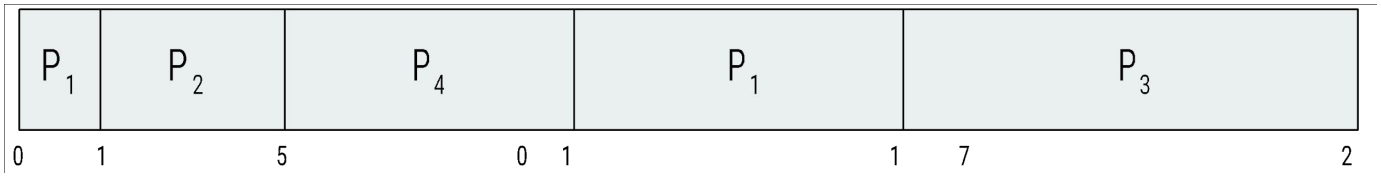
- **Explanation :**
  Initially T1 = 10 and α = 0.5 and the run times given are 8, 7, 4, 16 as it is shortest job first, formula is: $\tau_{n=1} = \alpha\ t_n + (1-\alpha)\tau_n$.
  So the possible order in which these processes would serve will be 4, 7, 8, 16 since SJF is a non-preemptive technique.
  So, using formula: T2 = α*t1 + (1-α)T1
  so we have,
  T2 = 0.5*4 + 0.5*10 = 7, here t1 = 4 and T1 = 10
  T3 = 0.5*7 + 0.5*7 = 7, here t2 = 7 and T2 = 7
  T4 = 0.5*8 + 0.5*7 = 7.5, here t3 = 8 and T3 = 7
  So the future prediction for 4th process will be T4 = 7.5 which is the option(c).

# Example of Shortest-remaining-time-first

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

| Process | *Arrival* Time | Burst Time |
|---|---|---|
| $P_1$ | 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

# SJF Advantages and Disadvantages

- **Advantages**
  - Shortest jobs are favored.
  - It is provably optimal; in that it gives the minimum average waiting time for a given set of processes.
- **Disadvantages**
  - SJF may cause starvation, if shorter processes keep coming. This problem is solved by aging.
  - It cannot be implemented at the level of short-term CPU scheduling.

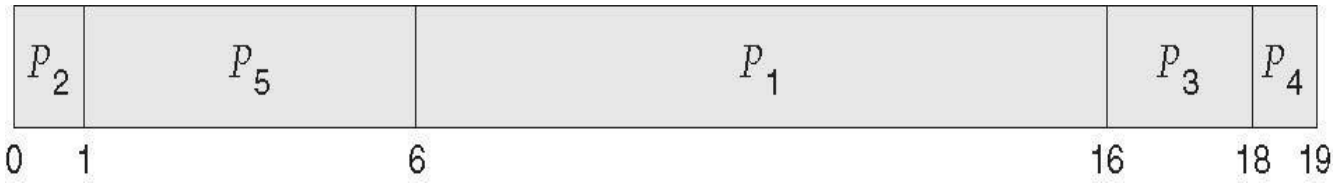
# Priority Scheduling

- 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 inverse of predicted next CPU burst time
- 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 msec

# Priority Scheduling Advantages and Disadvantages

- **Advantages**
  - The priority of a process can be selected based on memory requirement, time requirement or user preference.
- **Disadvantages:**
  - A second scheduling algorithm is required to schedule the processes which have same priority.
  - In preemptive priority scheduling, a higher priority process can execute ahead of an already executing lower priority process. If lower priority process keeps waiting for higher priority processes, starvation occurs.

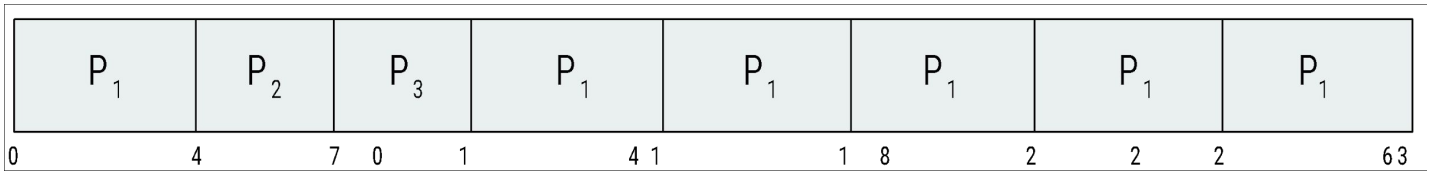
# Round Robin (RR)

- 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 ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

# Example of RR with Time Quantum = 4

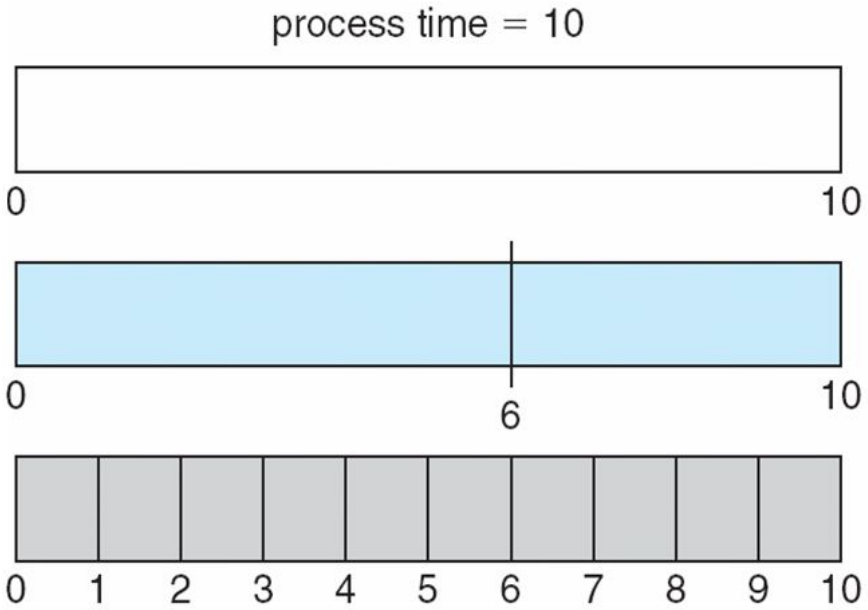
| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- The Gantt chart is:



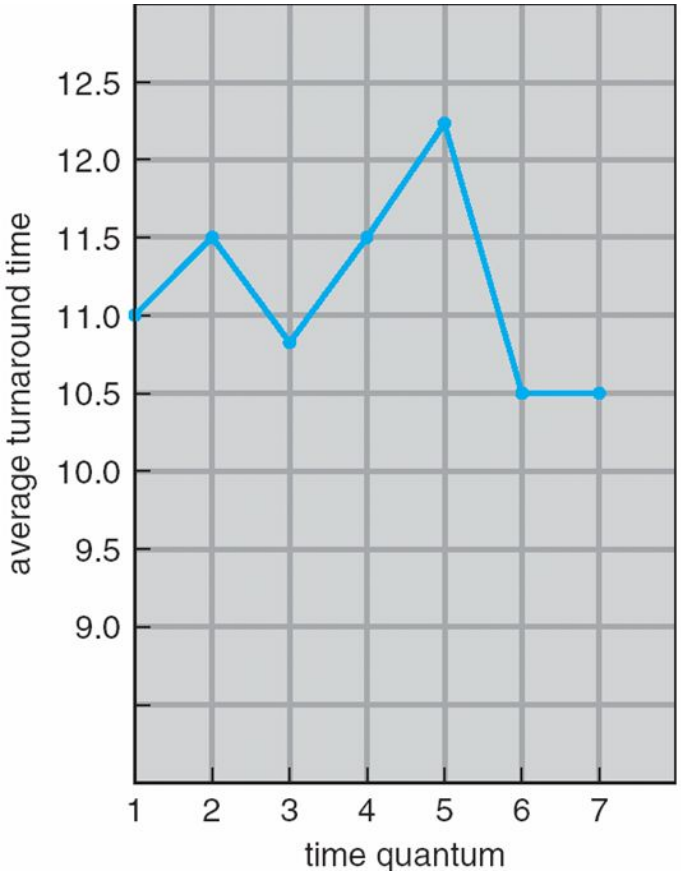

- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

# Time Quantum and Context Switch Time




| quantum | context switches |
|---|---|
| 12 | 0 |
| 6 | 1 |
| 1 | 9 |

# 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

# Round Robin (RR)

- **Advantages:**
  - Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority.
  - Starvation doesn't occur because for each round robin cycle, every process is given a fixed time to execute. No process is left behind.
- **Disadvantages:**
  - The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS.
  - If time quantum is shorter than needed, the number of times that CPU switches from one process to another process, increases. This leads to decrease in CPU efficiency.