

# Chapter 5: CPU Scheduling

---





# Outline

---

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation





# Objectives

---

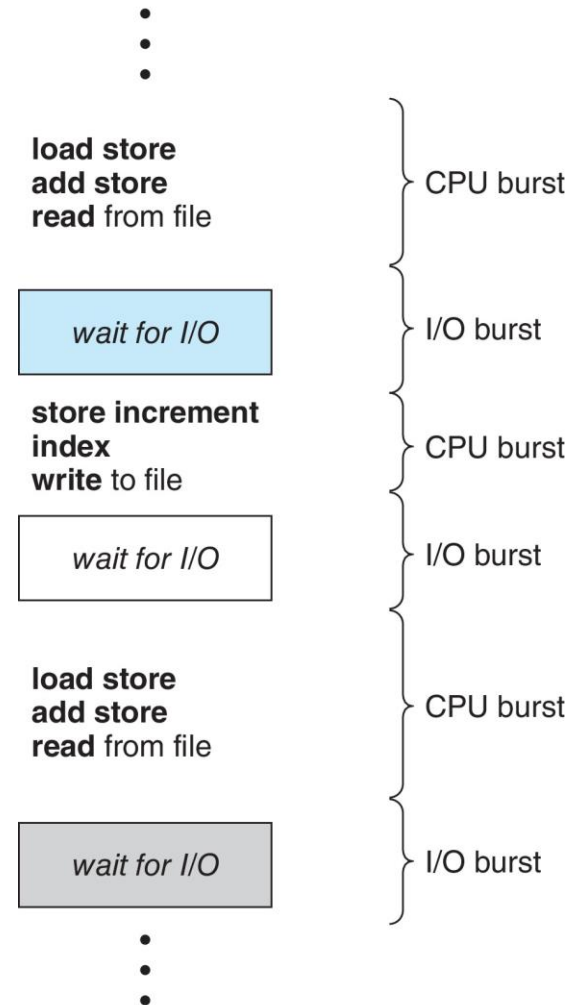
- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on scheduling criteria
- Explain the issues related to multiprocessor and multicore scheduling
- Describe various real-time scheduling algorithms
- Describe the scheduling algorithms used in the Windows, Linux, and Solaris operating systems
- Apply modeling and simulations to evaluate CPU scheduling algorithms





# 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

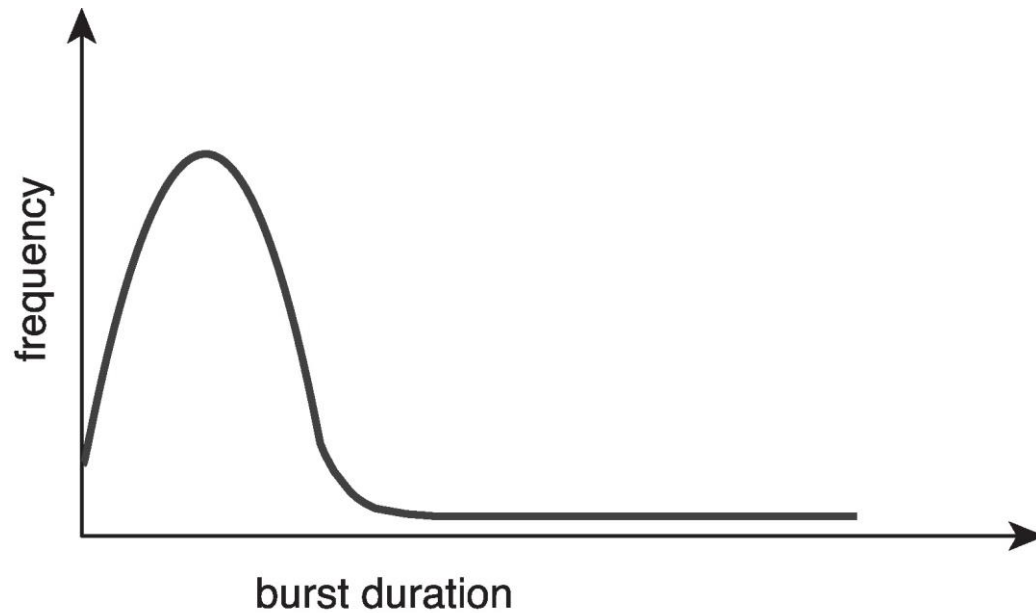




# Histogram of CPU-burst Times

Large number of short bursts

Small number of longer bursts





# CPU Scheduler

- The **CPU scheduler** selects from among the processes in ready queue, and allocates a CPU core 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
- For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.
- For situations 2 and 3, however, there is a choice.





# Preemptive and Nonpreemptive Scheduling

---

- When scheduling takes place only under circumstances 1 and 4, the scheduling scheme is **nonpreemptive**.
- Otherwise, it is **preemptive**.
- Under Nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state.
- Virtually all modern operating systems including Windows, MacOS, Linux, and UNIX use preemptive scheduling algorithms.





# Preemptive Scheduling and Race Conditions

---

- Preemptive scheduling can result in race conditions when data are shared among several processes.
- Consider the case of two processes that share data. While one process is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state.
- This issue will be explored in detail in Chapter 6.

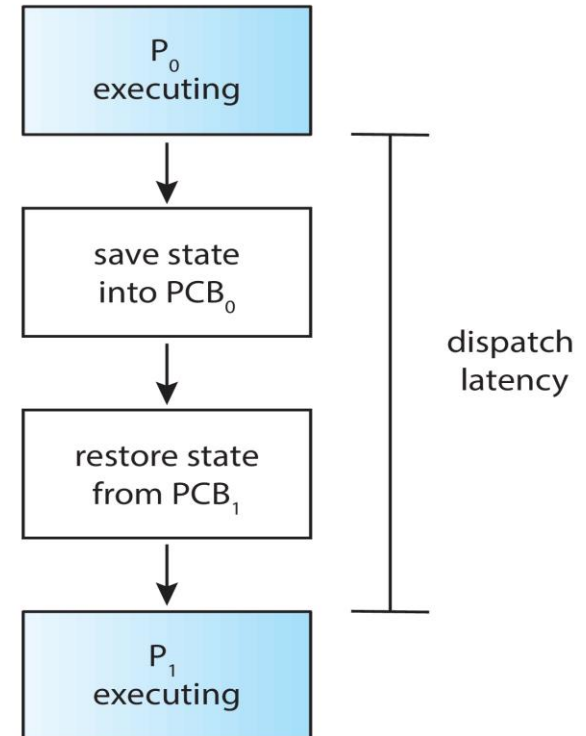






# Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU 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.





# 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:  $(0 + 24 + 27)/3 = 17$
- Turnaround Time of a process = Completion Time – Arrival Time
- Compute the average turnaround time?





# 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





# FCFS Scheduling (Cont.)

---

- Jobs are executed on first come, first serve basis.
- "H" It is a non-preemptive, pre-emptive scheduling algorithm.
- "H" Easy to understand and implement.
- "H" Its implementation is based on FIFO queue.
- "H" Poor in performance as average wait time is high.





# 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
- Preemptive version called **shortest-remaining-time-first**
  
- How do we determine the length of the next CPU burst?
  - Could ask the user
  - Estimate





- Shortest Job First (SJF)
- "H" This is also known as shortest job next, or SJN
- "H" This is a non-preemptive, pre-emptive scheduling algorithm.
- "H" Best approach to minimize waiting time.
- "H" Easy to implement in Batch systems where required CPU time is known in advance.
- "H" Impossible to implement in interactive systems where required CPU time is not known.
- "H" The processor should know in advance how much time process will take.



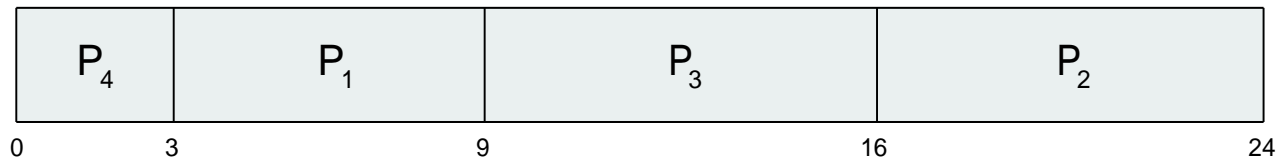




# Example of SJF

<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 =  $(3 + 16 + 9 + 0) / 4 = 7$





# 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:

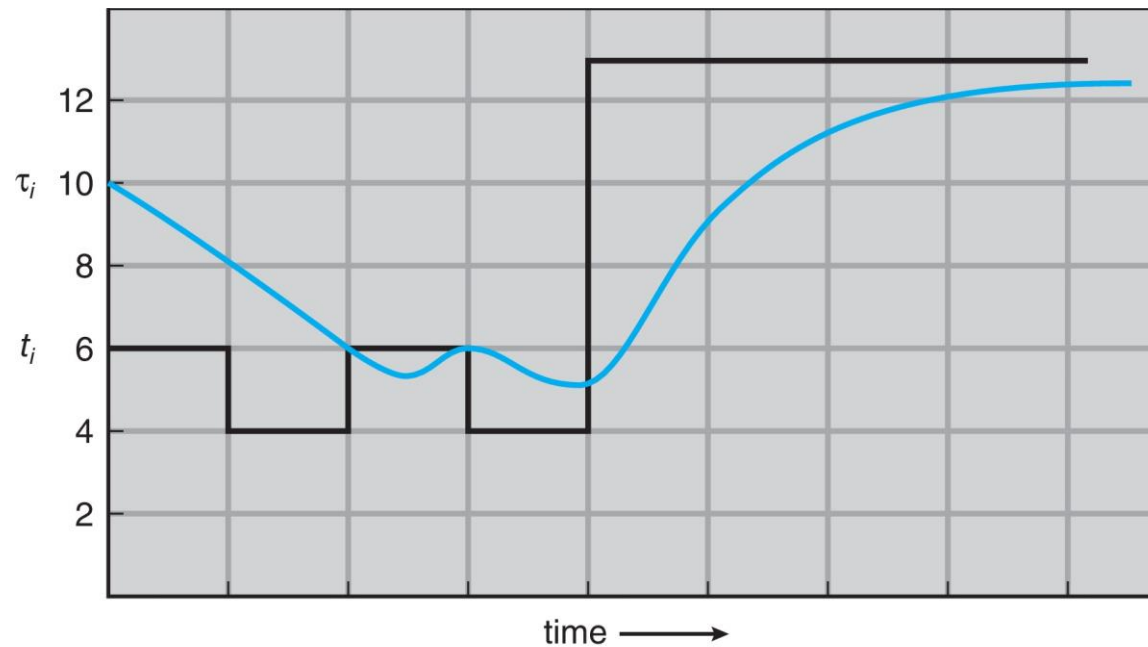
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n.$$

- Commonly,  $\alpha$  set to  $\frac{1}{2}$





# 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	5	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 successor predecessor term has less weight than its predecessor





# Shortest Remaining Time First Scheduling

---

- Preemptive version of SJN
- Whenever a new process arrives in the ready queue, the decision on which process to schedule next is redone using the SJN algorithm.
- Is SRT more “optimal” than SJN in terms of the minimum average waiting time for a given set of processes?



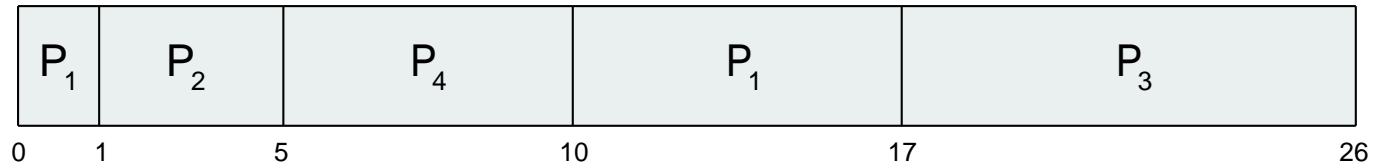


# Example of Shortest-remaining-time-first

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





# 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 (FCFS)
  - $q$  small  $\Rightarrow$  RR
- Note that  $q$  must be large with respect to context switch, otherwise overhead is too high

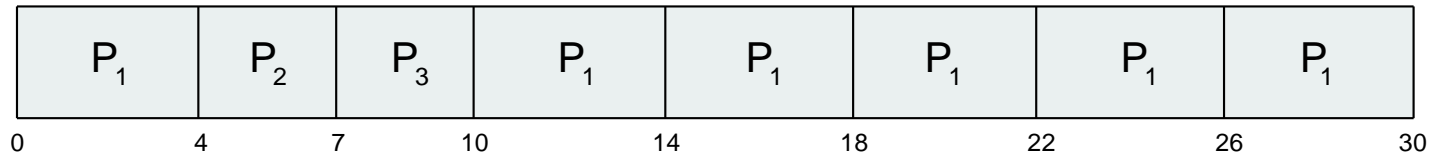




# Example of RR with Time Quantum = 4

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

- The Gantt chart is:



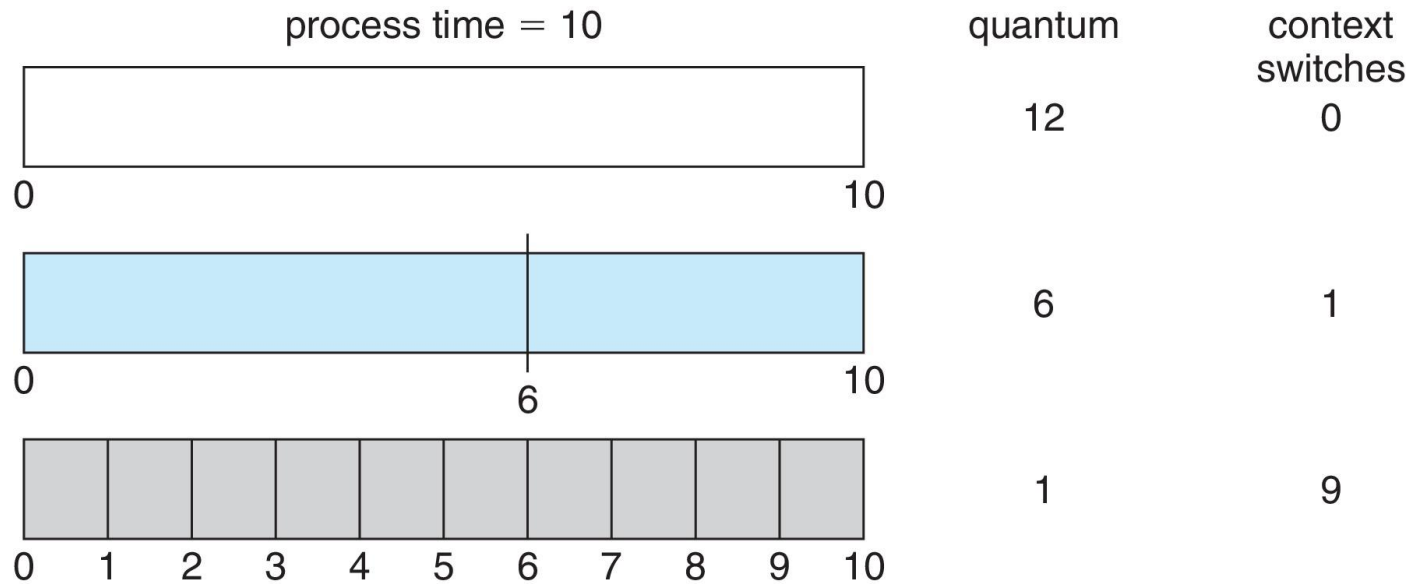
- Typically, higher average turnaround than SJF, but better **response**
- q should be large compared to context switch time
  - q usually 10 milliseconds to 100 milliseconds,
  - Context switch < 10 microseconds





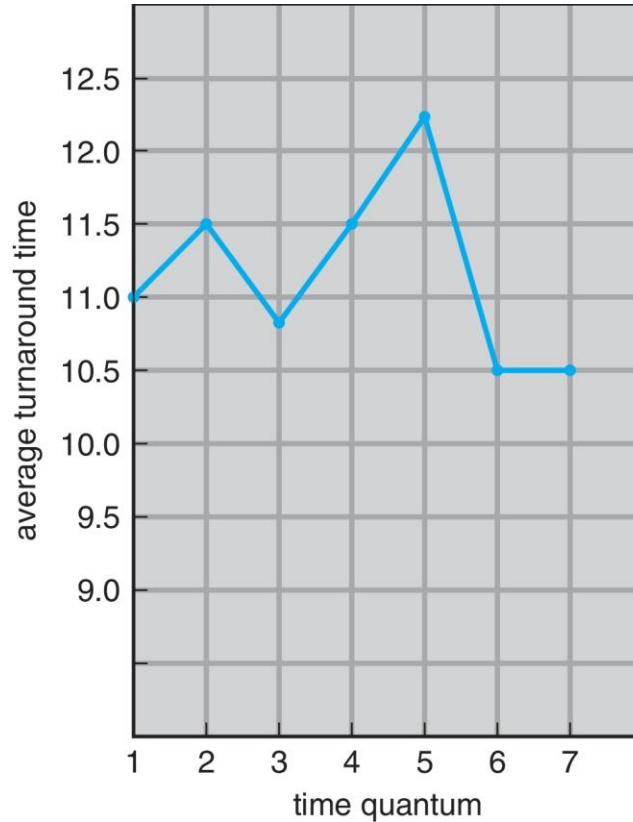


# Time Quantum and Context Switch Time





# Turnaround Time Varies With The Time Quantum



process	time
$P_1$	6
$P_2$	3
$P_3$	1
$P_4$	7

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





# 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
- Priority can be associated with each process depending on how urgently a process should execute. The CPU is then allocated to the process with the highest priority. Equal priority processes are scheduled by the FCFS algorithm.
- Problem  $\equiv$  **Starvation** – low priority processes may never execute
- Solution  $\equiv$  **Aging** – as time progresses increase the priority of the process





# Priority Scheduling

---

- **Priorities can be defined either internally or externally.**
- **Internally** defined priorities use some measurable quantity or quantities to compute the priority of a process, for example: time limits, memory requirements, the number of open files etc.
- **External** priorities are set by criteria that are external to the operating system, such as the type and the amount of funds being paid for computer use and other external, often political factors.

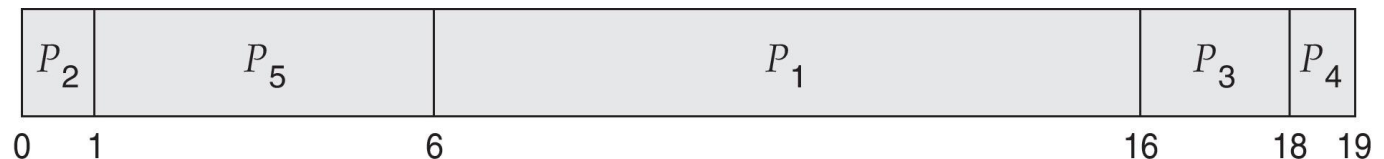




# Example of Priority Scheduling

<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





# Round Robin Scheduling

---

- Round Robin is the preemptive process scheduling algorithm.
- "H" Each process is provided a fix time to execute, it is called a quantum.
- "H" Once a process is executed for a given time period, it is preempted
- and other process executes for a given time period.
- "H" Context switching is used to save states of preempted processes.



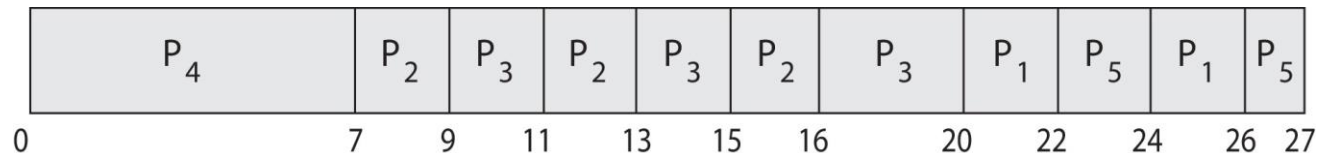


# Priority Scheduling w/ Round-Robin

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

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
$P_1$	4	3
$P_2$	5	2
$P_3$	8	2
$P_4$	7	1
$P_5$	3	3

- Gantt Chart with time quantum = 2





# Multilevel Queue

---

- The ready queue consists of multiple queues
- Multilevel queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine which queue a process will enter when that process needs service
  - Scheduling among the queues

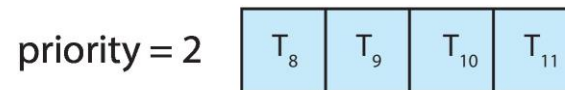
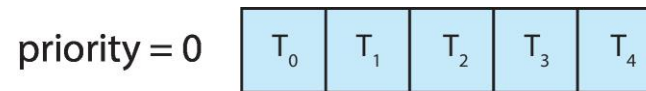






# Multilevel Queue

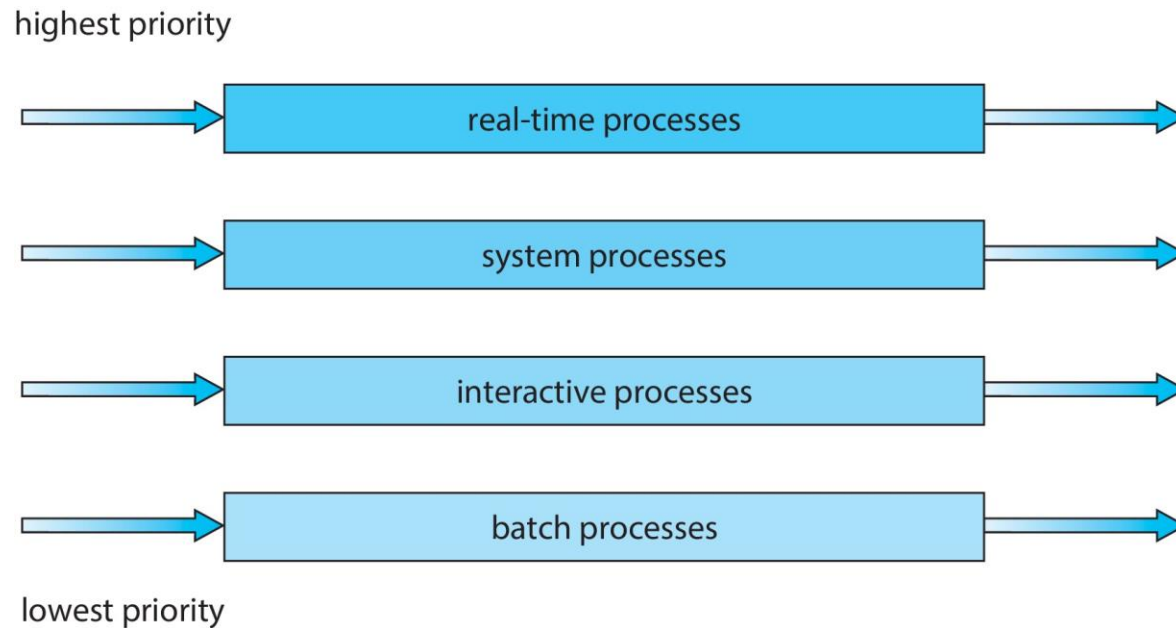
- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!





# Multilevel Queue

- Prioritization based upon process type





# Multilevel Feedback Queue

---

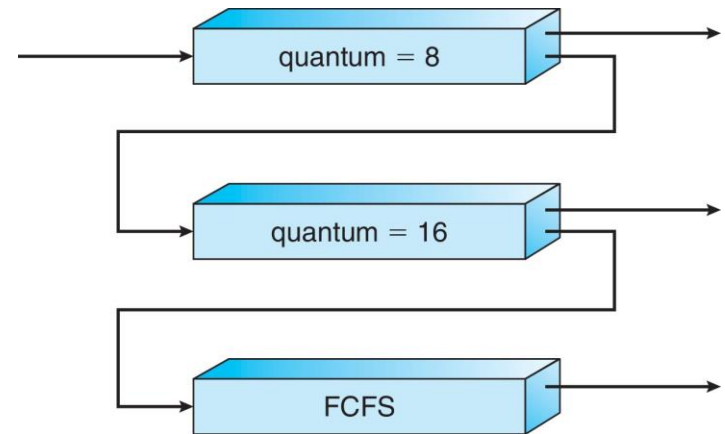
- A process can move between the various queues.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service
- 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
- 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  $Q_2$



# End of Chapter 5

---

