CPU SCHEDULING

Objectives

- CPU Scheduling, which is the basis of multiprogrammed operating systems

- Objective is to maximize CPU utilization.
  - By switching the CPU among processes, the operating system can make the computer more productive.

- In this lecture we learn;
  - **Some basic CPU scheduling concepts**
  - **Scheduling Criteria**
  - **Several CPU Scheduling Algorithms**
  - **Selecting an algorithm for a particular system.**

- ***On modern operating systems it is kernel level threads –not processes– that are in fact being scheduled by operating system.***

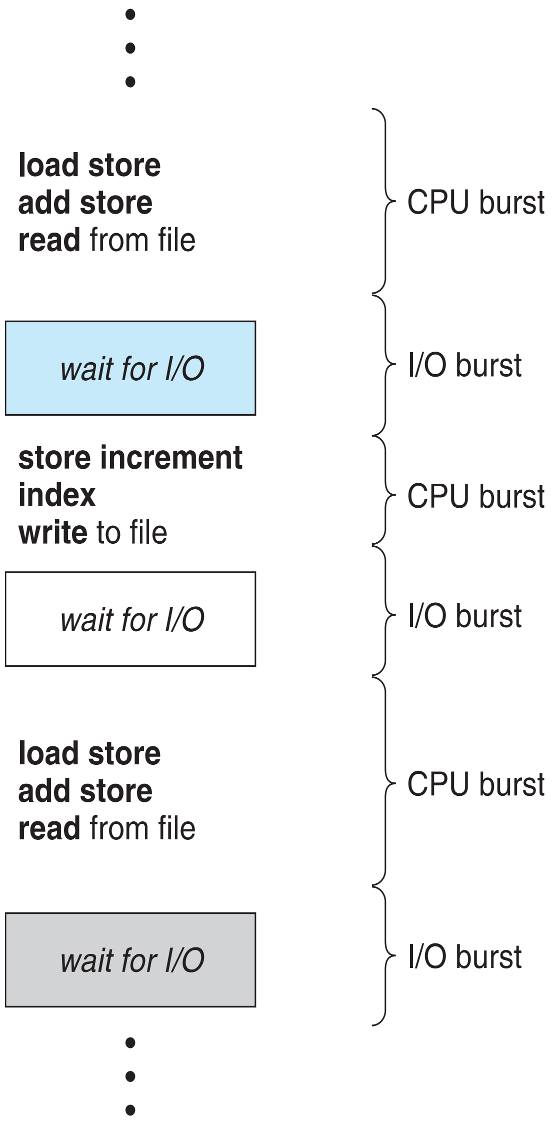
- However, the terms “ process scheduling” and thread scheduling are often used interchangeably.

Basic Concepts

- CPU-I/O Burst Cycle
- CPU Scheduler
- Preemptive and Non-preemptive scheduling
- Dispatcher

CPU-I/O Burst Cycle

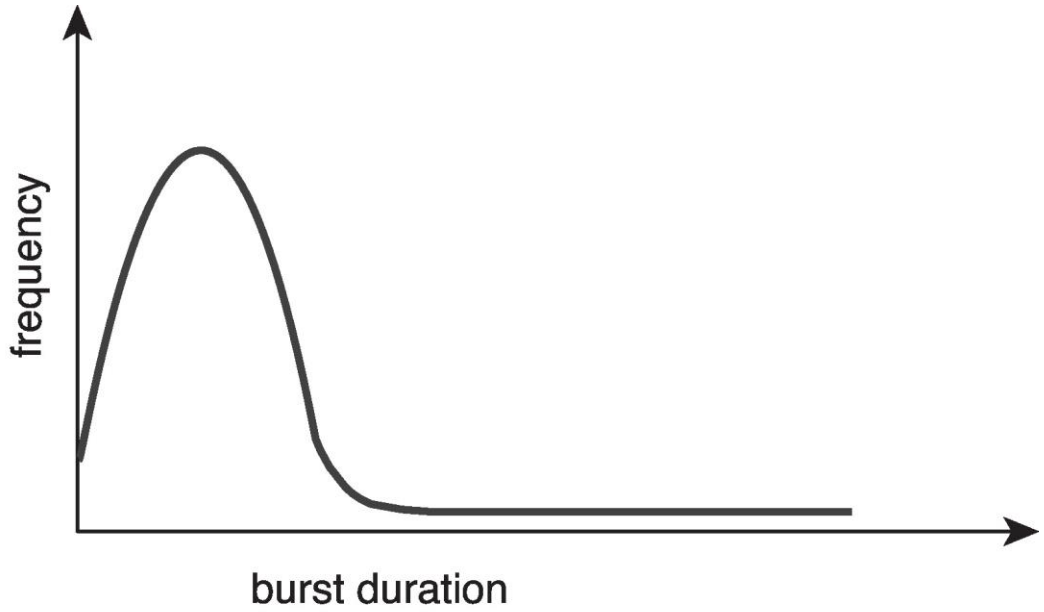
- The success of CPU scheduling depends on an observed property of processes:
  - ***process execution consists of a cycle of CPU execution and I/O wait.***
- Processes alternate between these two states.
  - Process execution **begins with a CPU burst**.
  - That is **followed by an I/O burst**,
  - which is followed by another CPU burst, then another I/O burst, and so on.
  - Eventually, the **final CPU burst ends with a system request** to terminate execution
  - An I/O bound program typically has many short CPU bursts. And a CPU-bound program might have few long CPU bursts.
  - **Duration of the CPU bursts have been measured extensively.**



Histogram of CPU-burst Times

Large number of short bursts

Small number of longer bursts



CPU Scheduler

- Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed.
  - **The *CPU scheduler* also called *short-term scheduler* , selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.**

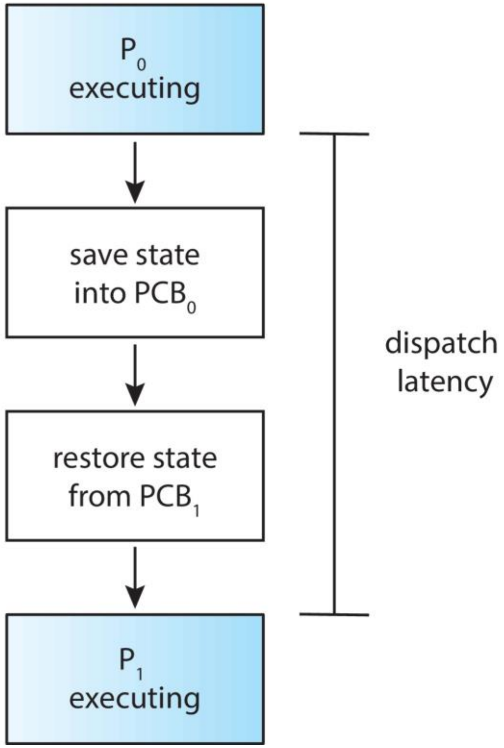
- The ready queue is not necessarily a first-in, first-out (FIFO) queue.
  - It can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list.

- The records in the queues are generally process control blocks (PCBs) of the processes.

Preemptive and non-preemptive scheduling

- 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** or **cooperative**
  - once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
- All other scheduling is **preemptive**
  - Consider access to shared data
    - preemptive scheduling can result in race conditions
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Dispatcher



- Another component involved in the CPU-scheduling function is the **dispatcher**.
- The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.
- This function involves the following:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- The dispatcher should be as fast as possible, since it is invoked during every process switch.
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

An interested question is, how often do context switches occur?

```
        $ vmstat 1 3
Output:
----cpu-----
        24
        225
        339
        cat /proc/2166/status
Output:
voluntay_ctxt_switches                150
nonvoluntay_ctxt_switches             8
```

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
  - CPU utilization can range from 0 percent to 100 percent
  - In real systems it should range from 40 percent to 90 percent.
- **Throughput** – number of processes that complete their execution per time unit
- **Turnaround time --**The interval from the time of submission of a process to the time of completion is the turnaround time.
    - From the point of view of a particular process, it is an important criterion.
    - It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
- **Waiting time** – amount of time a process has been waiting in the ready queue
    - The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O. It affects only the amount of time that a process spends 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)
    - In an interactive system, turnaround time may not be the best criterion.

Scheduling Algorithm Optimization Criteria

- **Maximize →CPU utilization and throughput**
- **Minimize → turnaround time, waiting time, and response time.**

- In most cases, we optimize the average measure. However, under some circumstances, we prefer to optimize the minimum or maximum values rather than the average.
  - For example, to guarantee that all users get good service, we may want to minimize the maximum response time not average response time.
- Investigators have suggested that, for interactive systems (such as desktop systems), it is more important to minimize the variance in the response time than to minimize the average response time.
  - A system with reasonable and predictable response time may be considered more desirable than a system that is faster on the average but is highly variable.

Scheduling Algorithms

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

Shortest-Job-First (SJF) Scheduling

- *Shortest Next CPU Burst algorithm*
- 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

	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart

P_4	P_1	P_3	P_2

0 3 9 16 24

- 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

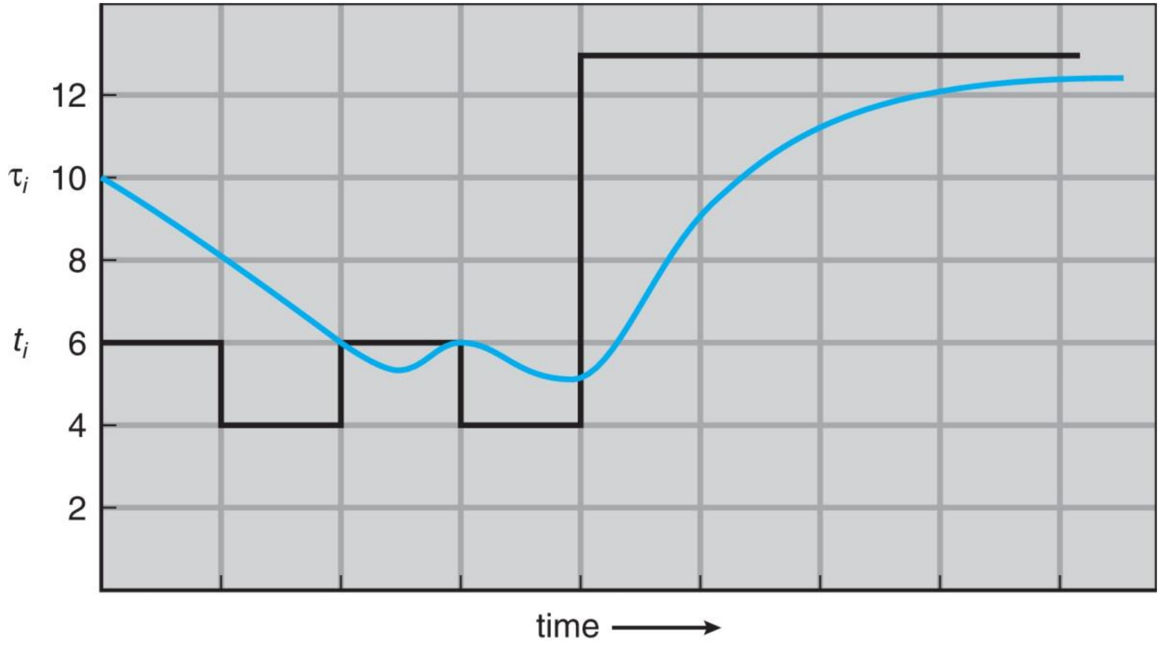
- Can be done by using the length of previous CPU bursts, using exponential averaging

  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



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_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:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

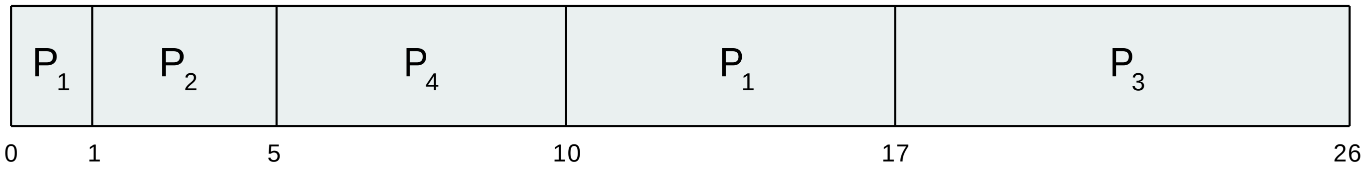
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

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

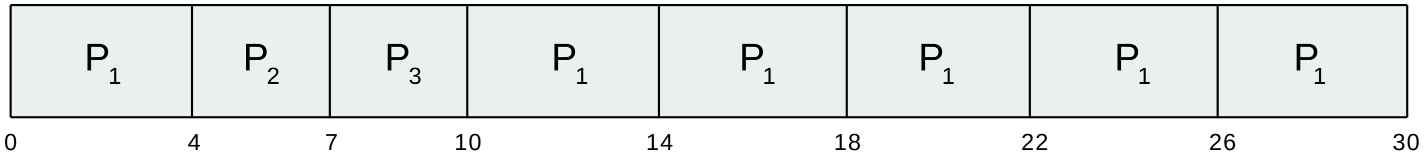
Round Robin (RR)

- 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

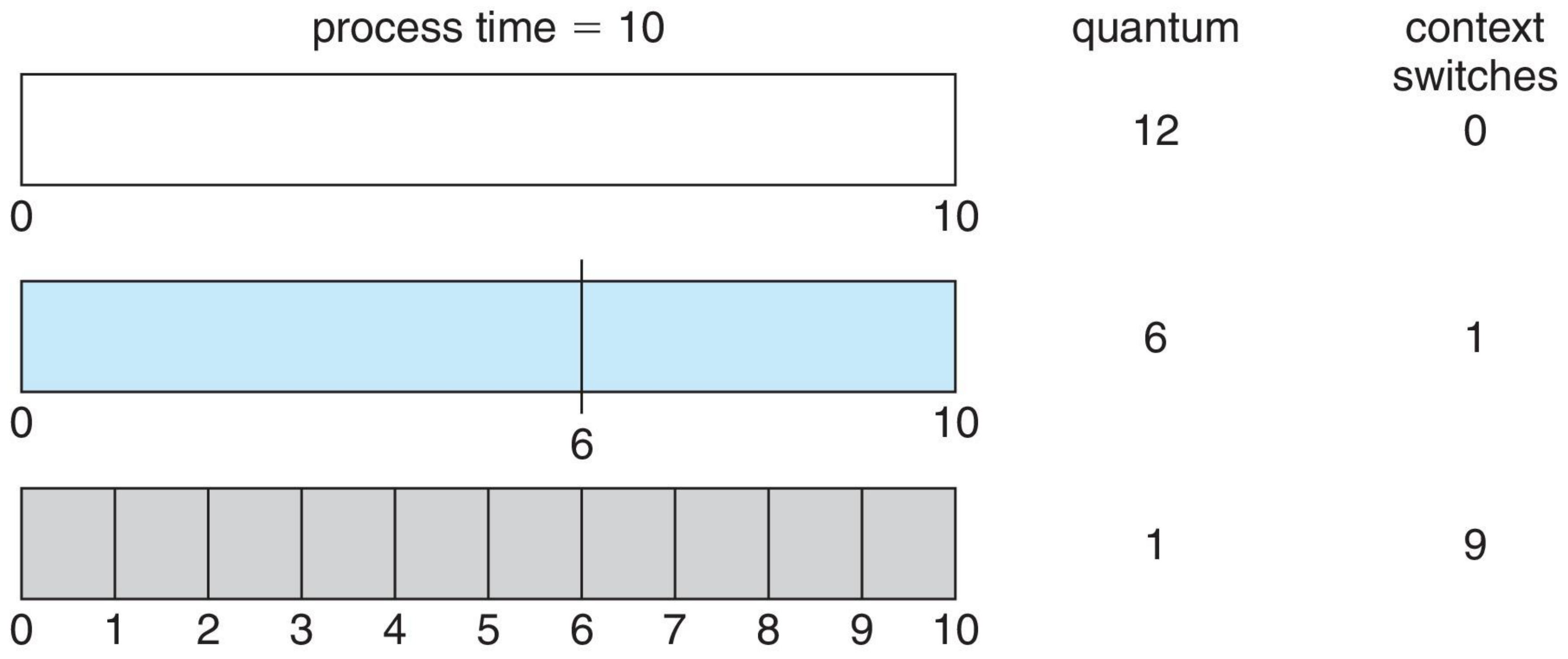
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

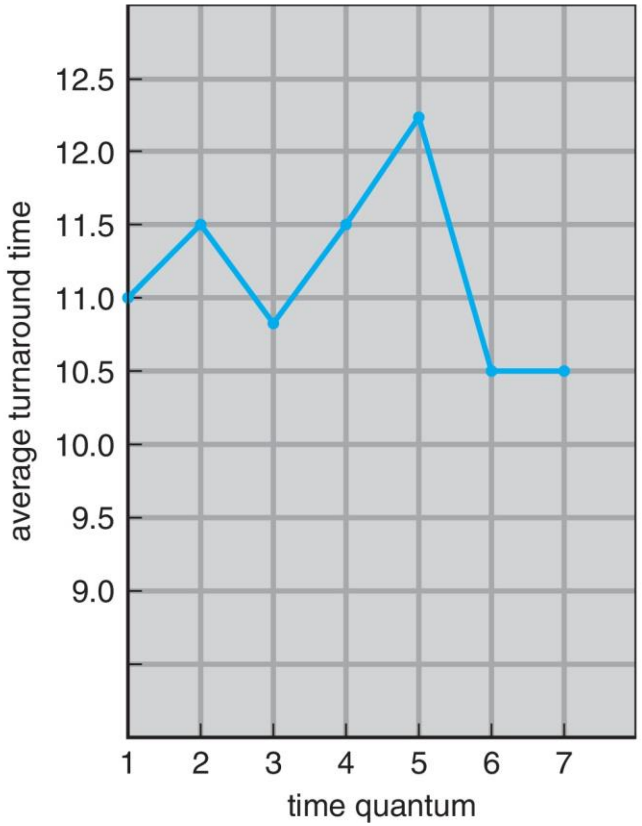


- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 μsec

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

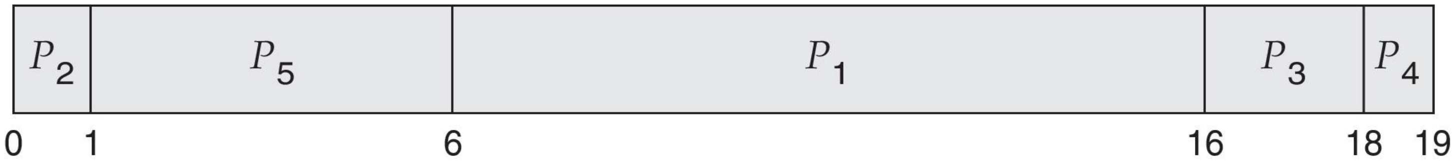
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ 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 T	Priority
P1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart

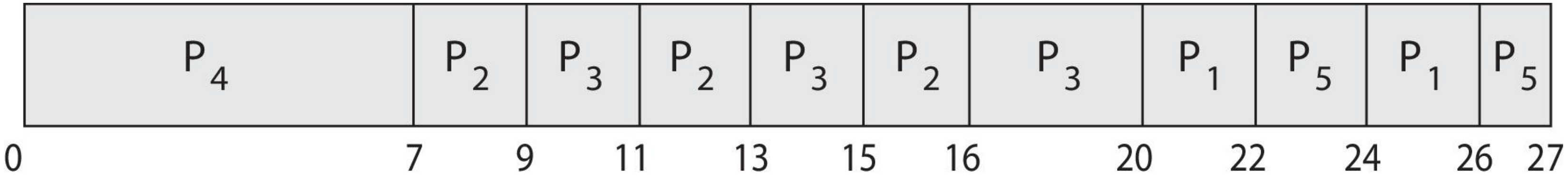


- Average waiting time = 8.2 msec

Priority Scheduling w/ Round-Robin

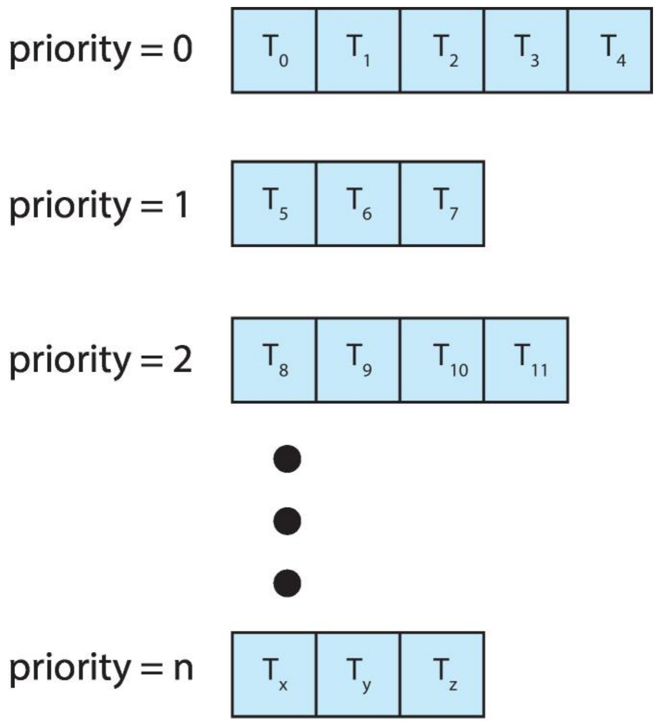
Process	Burst Time T	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes with the same priority run round-robin (q=2)



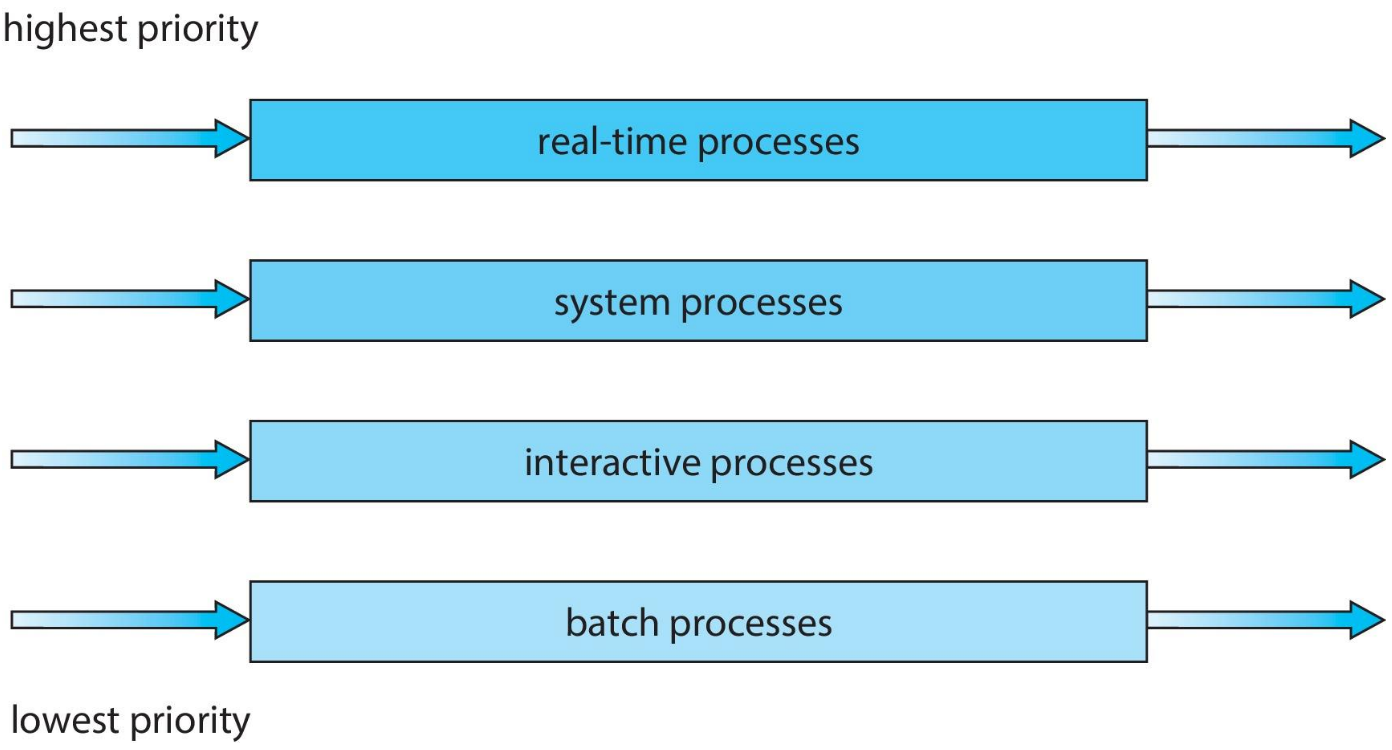
Multilevel Queue

- 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; 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 $Q1$
  - 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