Operating System

Chapter 4: Scheduling

Prepared By:
Amit K. Shrivastava
Asst. Professor
Nepal College Of Information Technology

Scheduling:

- When a computer is multiprogrammed, it frequently has multiple processes competing for the CPU at the same time. When more than one process is in the ready state and there is only one CPU available, a choice has to be made which process to run next. The part of the operating system that makes the choice is called the scheduler, and the algorithm it uses is called the scheduling algorithm and the mechanism is called scheduling.

Scheduling Criteria :

The criteria include the following:

- CPU utilization: The CPU should keep as busy as possible. CPU utilization may range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).
- Throughput: If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes completed per time unit, called throughput.
- Turnaround time: From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. 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: Waiting time is the sum of the periods spent waiting in the ready queue.
- Response Time: Response time is the time it takes to start responding.

Preemptive Scheduling and Non Preemptive Scheduling

- **Non preemptive:** In this case, once a process is in the running state, It continues to execute until (a) it terminates or (b) blocks itself to wait for I/O or to request some operating system service.

- **Preemptive:** The currently running process may be interrupted and moved to the ready state by the operating system. The decision to preempt may be performed when a new process arrives or periodically based on a clock interrupt OR when a new process switches from the waiting state to the ready state(for example, at completion of I/O)

Scheduling Technique

- **First-Come-First-Served(First-In-First-Out) Scheduling:** With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue.

Consider the following set of processes that arrive at the time 0, with the length of the CPU burst given in milliseconds.

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:

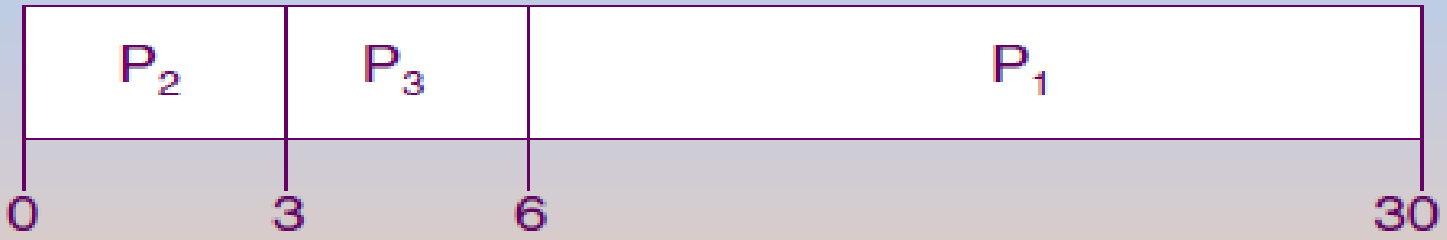
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

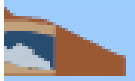
$$P_2 , P_3 , P_1 .$$

- 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

Shortest-Job-First (SJR) Scheduling

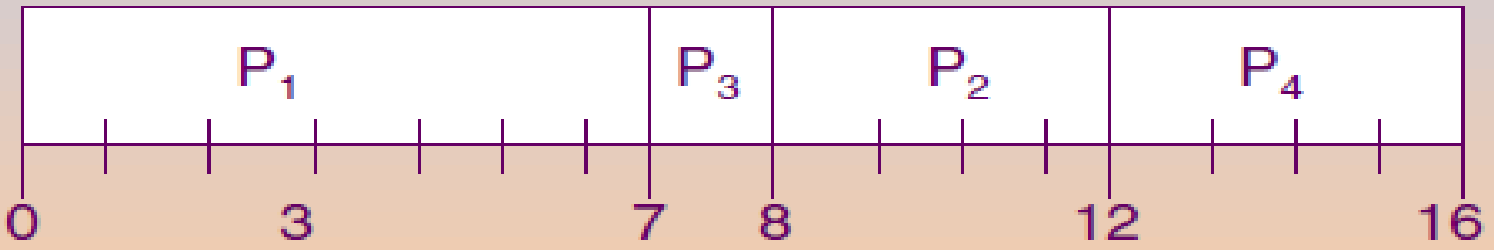


- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst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

Example of 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)

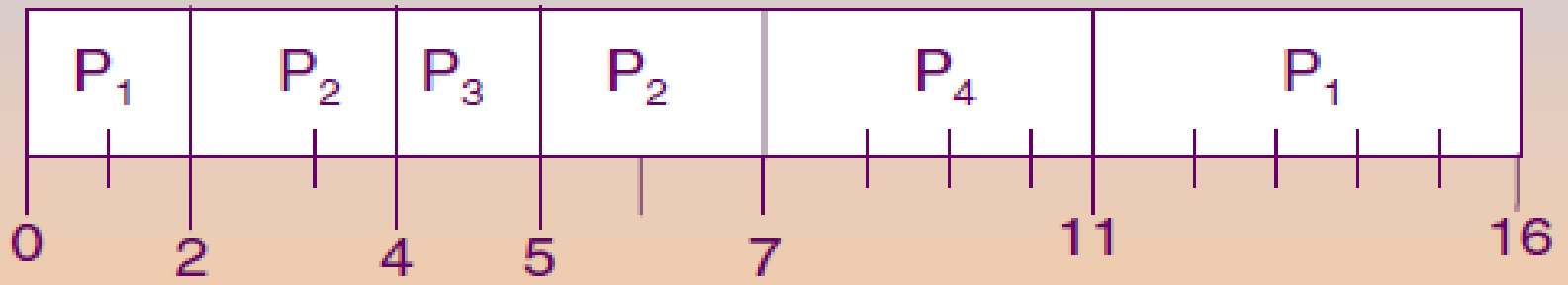


- Average waiting time = (0 + 6 + 3 + 7)/4 - 4

Example of 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 (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 - 3

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

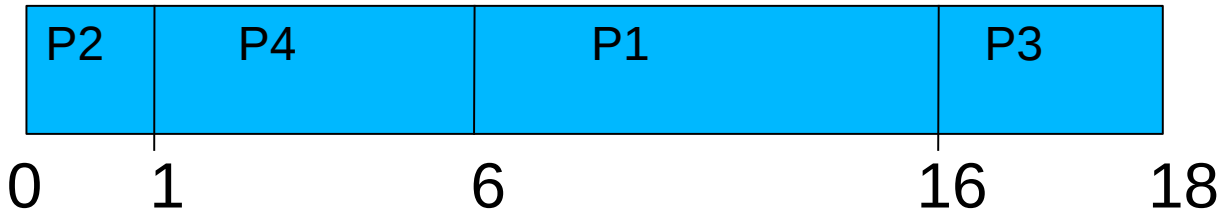
- SJF is a priority scheduling where priority is the predicted next CPU burst time.

Problem ≡ Starvation – low priority processes may never execute.

Solution ≡ Aging – as time progresses increase the priority of the process.

Consider following set of process arrived at time 0

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	5	2



Average Waiting Time=8.2 ms

Round Robin (RR)

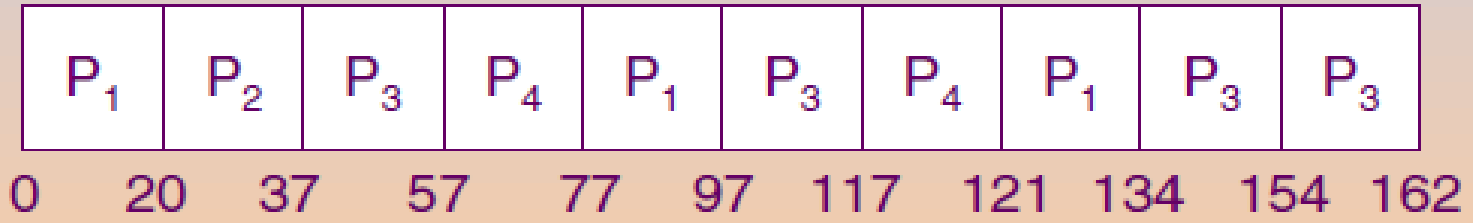
RR is always Pre-emptive

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

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

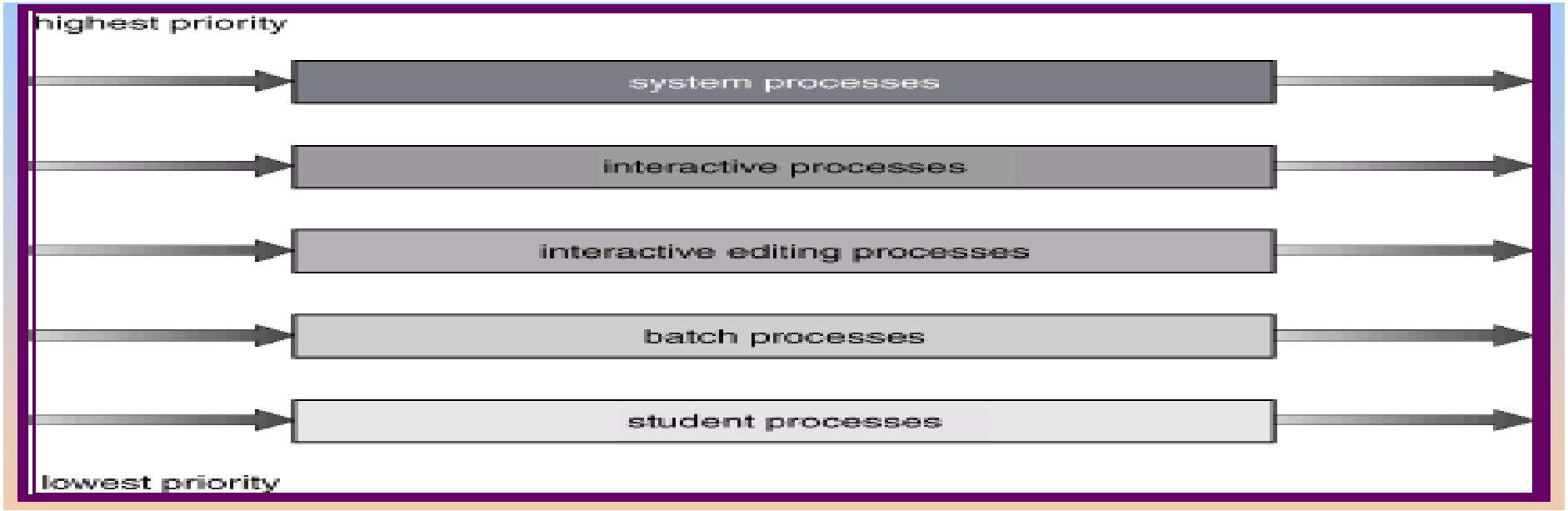
- The Gantt chart is:



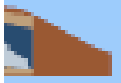
- Typically, higher average turnaround than SJF, but better *response*.

Multilevel Queue Scheduling:

- 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 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.

Highest-Response-Ratio-Next(HRN) Scheduling

- HRN is a nonpreemptive scheduling discipline in which the priority of each job is a function not only of the job's service time but also of the amount of the time the job has been waiting for service. Once a job gets the CPU, it runs to completion. Dynamic priorities in HRN are calculated according to the formula

$$\text{priority} = \frac{\text{time waiting} + \text{Service Time}}{\text{Service Time}}$$

- Because the service time appears in the denominator, shorter job will get preference. But because time waiting appears in the numerator, longer jobs that have been waiting will also be given favorable sum.

Deadline Scheduling

- In deadline scheduling certain jobs are scheduled to be completed by a specific time or deadline. These jobs may have very high value if delivered on time and may be worthless if delivered later than the deadline. Deadline scheduling is complex for many reasons.

• The user must supply the precise resource requirements of the job in advance. Such information is rarely available.

• The system must run the deadline job without severely degrading service to other users.

• If many deadline jobs are to be active at once, scheduling could become so complex that sophisticated optimization methods might be needed to ensure that deadlines are met.