CPU Scheduling

Unit-II

Lecture -1

Session 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

Session Outcomes

At the end of this session, participants will be able to

- Discuss various CPU-scheduling algorithms and their evaluation criteria

Agenda

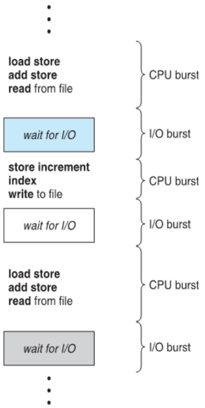
- Basic Concepts
- Scheduling Criteria
- First- Come, First-Served
- Shortest-Job-First
- Priority
- Round Robin
- Multilevel Queue

Presentation Outline

- Basic Concepts
- Scheduling Criteria
- First- Come, First-Served
- Shortest-Job-First
- Priority
- Round Robin
- Multilevel Queue

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- **CPUI/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



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

Presentation Outline

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

Presentation Outline

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: *P1, P2, P3*. The Gantt chart for the schedule is

- Waiting Time for *P1* =0; *P2* = 24; *P3*= 27
  Average Waiting Time : $(0 + 24 + 27)/3 = 17$

FCFS Scheduling

- Suppose that the processes arrive in the order: P2, P3, P1 The Gantt chart for the schedule is

- Waiting time for P1 $= 6$; P2 $= 0$; P3 $= 3$
- 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

Presentation Outline

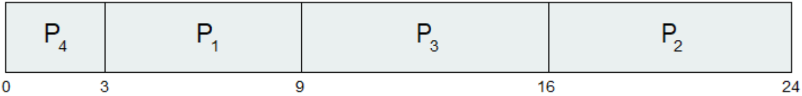
Shortest-Job-First (SJF) Scheduling

- 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

Process	Burst Time
P1	6
P2	8
P3	7
P4	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. τ_{n+1} = predicted value for the next CPU burst
  3. $\alpha, 0 \leq \alpha \leq 1$
  4. Define $\tau_{n=1} = \alpha t + (1 - \alpha)\tau_n$
- Commonly, α set to $\frac{1}{2}$
- Preemptive version called **shortest-remaining-time-first**

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 thExamples of Exponential Averaginge 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

Presentation Outline

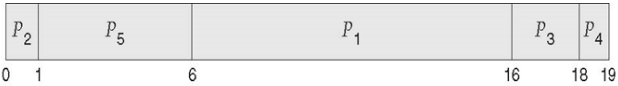
Priority Scheduling

- 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

Presentation Outline

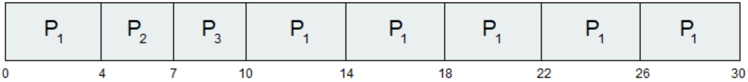
Round Robin (RR)

- 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

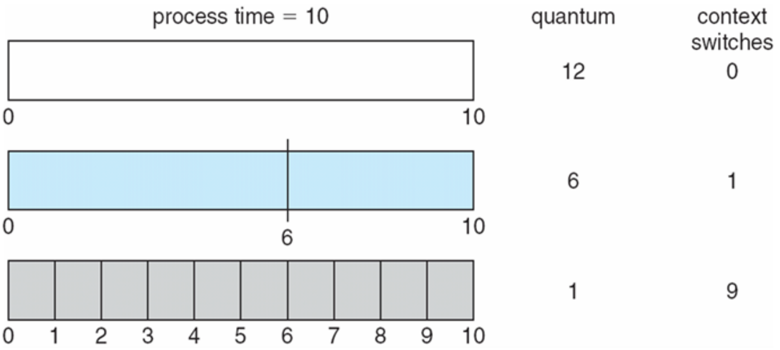
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

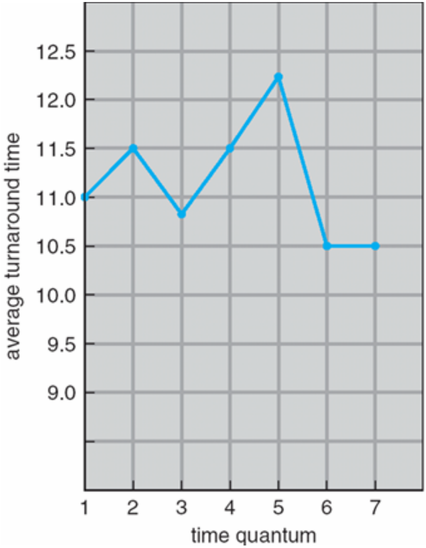


- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

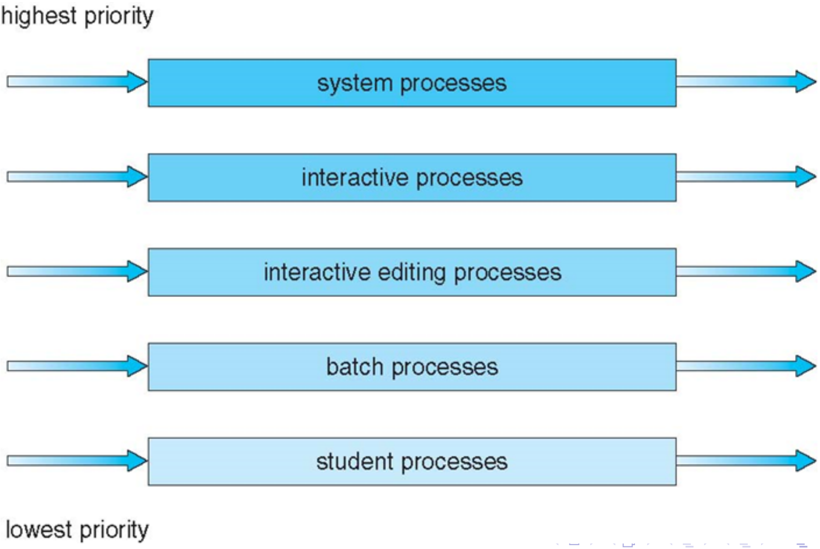
80% of CPU bursts
should be shorter than q

Presentation Outline

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

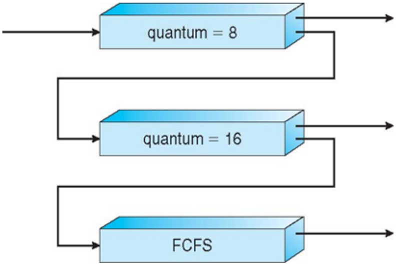


Multilevel Feedback Queue

- 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

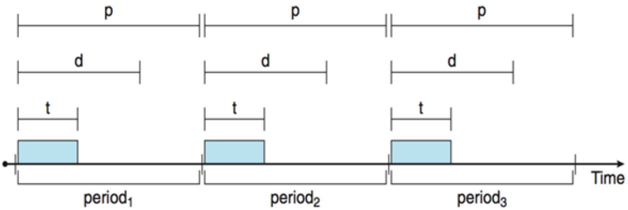
Example of Multilevel Feedback Queue

- 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



Priority-based Scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: periodic ones require CPU at constant intervals
  - Has processing time t, deadline d, period p
  - $0 \leq t \leq d \leq p$
  - Rate of periodic task is $1/p$

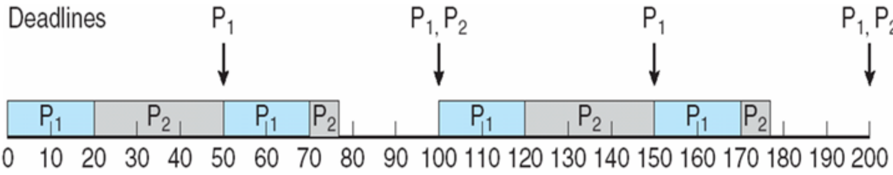


Rate Montonic Scheduling

- Rate Montonic Scheduling assumes that the processing time of a periodic process is the same for each CPU burst
- static priority policy with preemption
- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P1 is assigned a higher priority than P2.

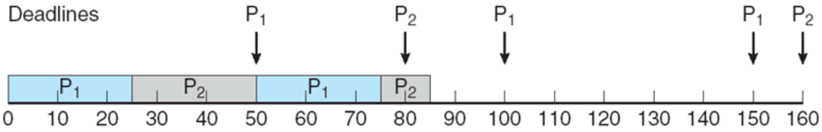
Example

- Processes, P1 and P2.
- Periods p1 = 50 and p2 = 100.
- t1 = 20 for P1 and t2 = 35 for P2.
- The deadline for each process requires that it complete its CPU burst by the start of its next period.
- CPU utilization of a process P_i : the ratio of its burst to its periodt_i/p_i the CPU utilization of P1 is 20/50 = 0.40 and that of P2 is 35/100 = 0.35, **total CPU utilization** : 75



Missed Deadlines with Rate Monotonic Scheduling

- Assume that process P1 has a period of $p1 = 50$ and a CPU burst of $t1 = 25$.
- For P2, the corresponding values are $p2 = 80$ and $t2 = 35$.
- Rate-monotonic scheduling would assign process P1 a higher priority, as it has the shorter period.
- The total CPU utilization : $(25/50)+(35/80) = 0.94$



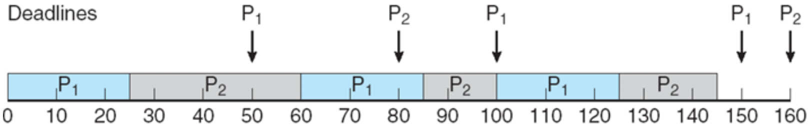
Rate Monotonic Scheduling cannot guarantee that they can be scheduled so that they meet their deadlines

Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:
- The earlier the deadline, the higher the priority;
- The later the deadline, the lower the priority

Earliest Deadline First Scheduling (EDF)

- EDF scheduling allows process P2 to continue running.
- P2 now has a higher priority than P1 because its next deadline (at time 80) is earlier than that of P1 (at time 100). Thus, both P1 and P2 meet their first deadlines.
- Process P1 again begins running at time 60 and completes its second CPU burst at time 85, also meeting its second deadline at time 100.
- P2 begins running at this point, only to be preempted by P1 at the start of its next period at time 100.
- P2 is preempted because P1 has an earlier deadline (time 150) than P2 (time 160).
- At time 125, P1 completes its CPU burst and P2 resumes execution, finishing at time 145 and meeting its deadline.
- The system is idle until time 150, when P1 is scheduled to run once again.


Deadlines
P1
P2
P1
P1
P2
P1
P2
P1
P2
P1
P2
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160

Summary

- CPU-scheduling algorithms :task of selecting a waiting process from the ready queue and allocating the CPU to it
- FCFS, SJF, RR, Priority
- Real time scheduling

Test your understanding

- Explain the difference between preemptive and nonpreemptive scheduling.
- What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?