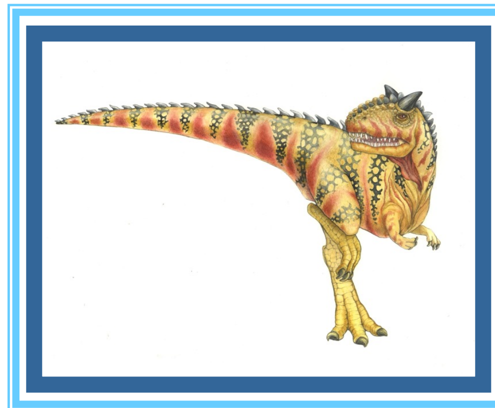
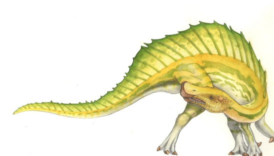
Chapter 6: CPU Scheduling



Chapter 6: CPU Scheduling

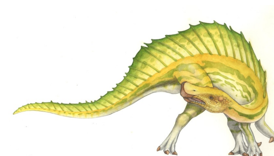
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

- Sections from the textbook: 6.1, 6.2, and 6.3



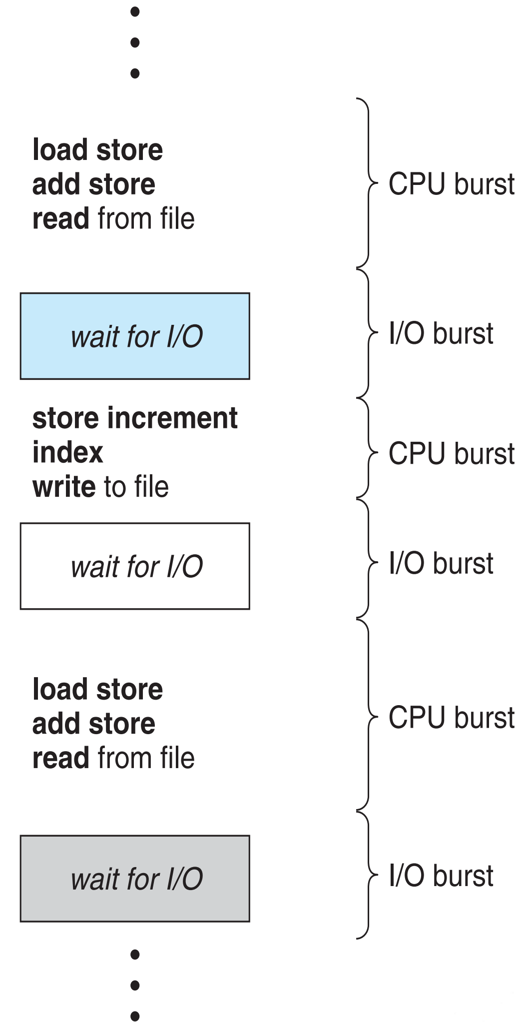
Objectives

- 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



Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- Process execution consists of **cycles** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst,** and so on.
- Process execution begins and ends with **CPU burst.**
- CPU burst distribution is of main concern



Histogram of CPU-burst Times

Short CPU-bursts

Long CPU-bursts

frequency

140
120
100
80
60
40
20

0 8 16 24 32 40

burst duration (milliseconds)

CPU Scheduler

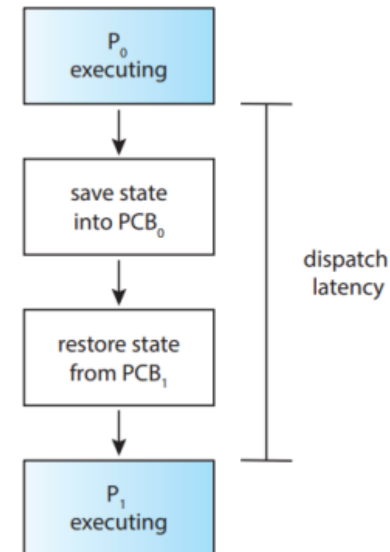
- **When the CPU is idle, Short-term scheduler (CPU scheduler)** selects from the processes in ready queue (memory), and allocates the CPU to one of them
  - Queue may be ordered in various ways (not only “FIFO”).
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state (I/O request or wait())
  2. Switches from running to ready state (interrupts)
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive (cooperative)**
- All other scheduling is **preemptive, which can result in race conditions:**
  - Consider access to shared data
  - 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.**
- 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
- The dispatcher should be fast!
- **Dispatch latency**

– time it takes for the dispatcher to stop one process

and start another running



Scheduling Criteria

- Criteria have been suggested for comparing CPU-scheduling algorithms.
  - **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, from submission to completion (TAT=CT-AT)
  - **Waiting time** – amount of time a process spends waiting in the ready queue (WT=CT-AT-BT)➔ (WT=TAT-BT)
  - **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output.

Completion Time (CT): This is the time when the process completes its execution.
Arrival Time (AT): This is the time when the process has arrived in the ready state.
Burst Time (BT): This is the time required by the process for its execution.

Scheduling Algorithm Optimization Criteria

- It is better to:
  - Maximizing CPU utilization ⇧
  - Maximizing throughput ⇧
  - Minimizing turnaround time ⇩
  - Minimizing waiting time ⇩
  - Minimizing response time ⇩

Scheduling Algorithms

- CPU scheduling algorithm decides on which of the processes in the ready queue is to be allocated the CPU.

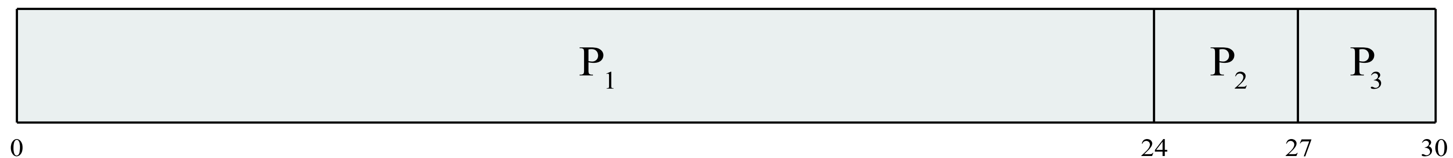
  - First- Come, First-Served (FCFS) Scheduling.
  - Shortest-Job-First (SJF) Scheduling.
  - Priority Scheduling.
  - Round Robin (RR).
  - Multilevel Queue.

First- Come, First-Served (FCFS) Scheduling

- The simplest, the process that requests the CPU first is allocated the CPU first.
- With the help of FIFO queue.
- *nonpreemptive*

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 milliseconds
- Average waiting time: (0 + 24 + 27)/3 = 17 milliseconds

First- Come, First-Served (FCFS) Scheduling

- Turnaround Time for P_1 = 24; P_2 = 27; P_3 = 30
- Average Turnaround Time : (24 + 27 + 30)/3 = 30

First- Come, First-Served (FCFS) Scheduling

Process	Arrive Time	Burst Time
P_1	0	24
P_2	1	3
P_3	2	3

Waiting time?

- Turnaround Time for P_1 = 24-0; P_2 = 27-1; P_3 = 30-2
- Average Turnaround Time: (24 + 26 + 28)/3 = 26

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➔ Lower CPU utilization
  - Consider one CPU-bound and many I/O-bound processes
- Troublesome for time-sharing systems.

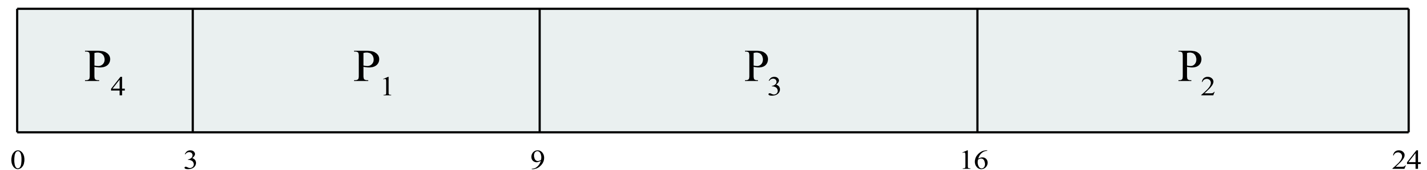
Shortest-Job-First (SJF) Scheduling

- 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

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
- Turnaround Time for P_1 = 9 ; P_2 = 24; P_3 = 16; $P4$ = 3
- Average Turnaround Time: (9+24+16+3)/4 = 13
- What is the average waiting time using FCFS scheduling ?

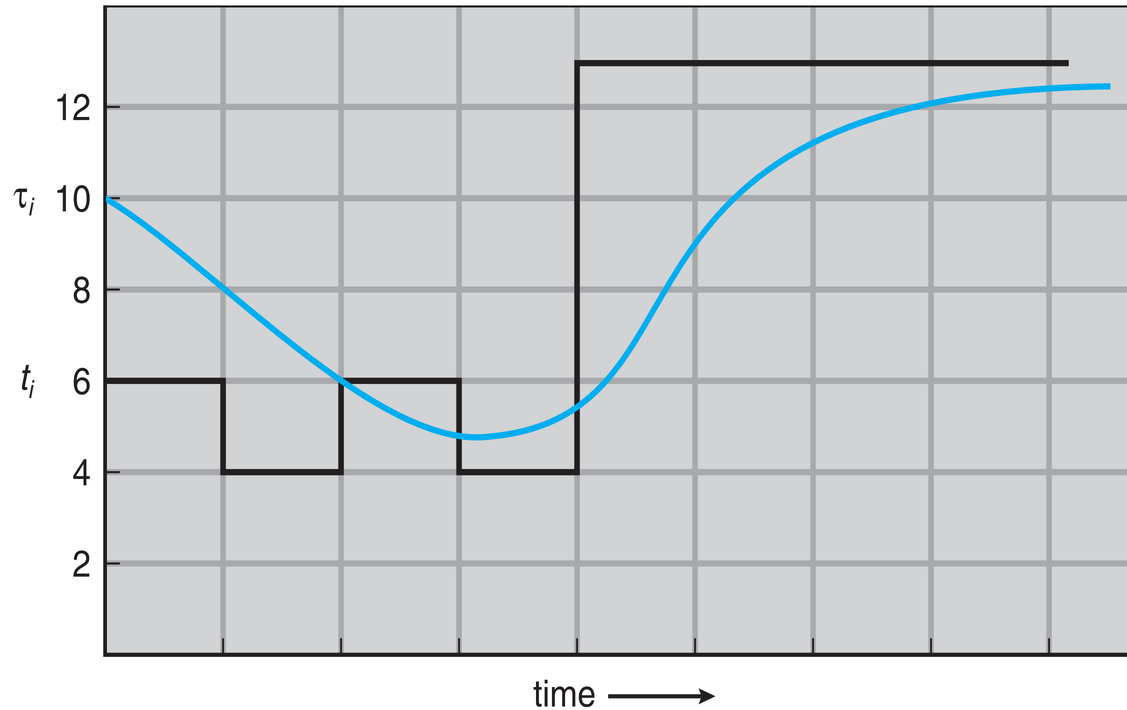
Determining Length of Next CPU Burst

- 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 <u>exponential averaging</u>

1. $t_n =$ actual length of nth CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$

- t_n contains our most recent information, while τ_n stores the past history.
- Commonly, α set to ½

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	…

Prove that diagram !!!

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:

$$\begin{aligned}\tau_{n+1} &= \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \dots \\ &\quad + (1 - \alpha)^j \alpha\, t_{n-j} + \dots \\ &\quad + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

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

Examples of Exponential Averaging

- **Problem:**

Calculate the predicted burst time using exponential averaging for the fifth process if the predicted burst time for the first process is 10 units and actual burst time of the first four processes is 4, 8, 6 and 7 units respectively. Given α = 0.5.

- **Solution:**

- **Predicted Burst Time for 2nd Process-**

= α x Actual burst time of 1st process + (1-α) x Predicted burst time for 1st process

= 0.5 x 4 + 0.5 x 10 = 2 + 5 = 7 units

- **Predicted Burst Time for 3rd Process-**

= α x Actual burst time of 2nd process + (1-α) x Predicted burst time for 2nd process

= 0.5 x 8 + 0.5 x 7 = 4 + 3.5 = 7.5 units

- **Predicted Burst Time for 4th Process-**

= α x Actual burst time of 3rd process + (1-α) x Predicted burst time for 3rd process

= 0.5 x 6 + 0.5 x 7.5 = 3 + 3.75 = 6.75 units

- **Predicted Burst Time for 5th Process-**

= α x Actual burst time of 4th process + (1-α) x Predicted burst time for 4th process

= 0.5 x 7 + 0.5 x 6.75 = 3.5 + 3.375 = 6.875 units

https://www.gatevidyalay.com/predicting-burst-time-sjf-scheduling/

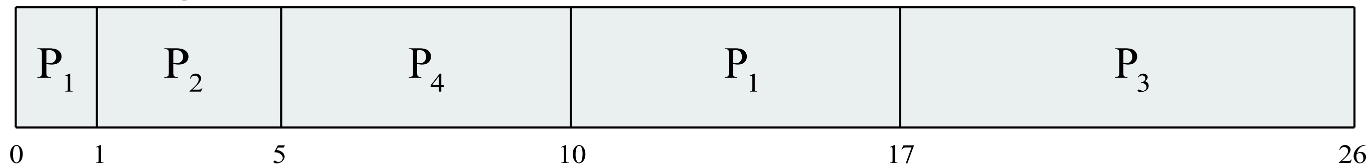
Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis
- SJF can be *preemptive* or *nonpreemptive*.
- The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process.
  - A preemptive SJF algorithm will preempt the currently executing process,.
  - a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF ➔ **shortest-remaining-time-first**

Example of Shortest-remaining-time-first

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



- Waiting time(WT)= total waiting time – # of milliseconds process executed – arrival time
  - WT for P_1= (10-1-0)= 9 ms
  - WT for P_2=(1-0-1)= 0 ms
  - WT for P_3=(17-0-2)= 15 ms
  - WT for P_4=(5-0-3)= 2 ms
- Average waiting time = (9+0+15+2)/4 = 26/4 = 6.5 ms
- Turnaround Time for P_1 = 17-0 ; P_2 = 5-1; P_3= 26-2; *P4* = 10-3
- Average Turnaround Time: (17+4+24+7)/4

Priority Scheduling

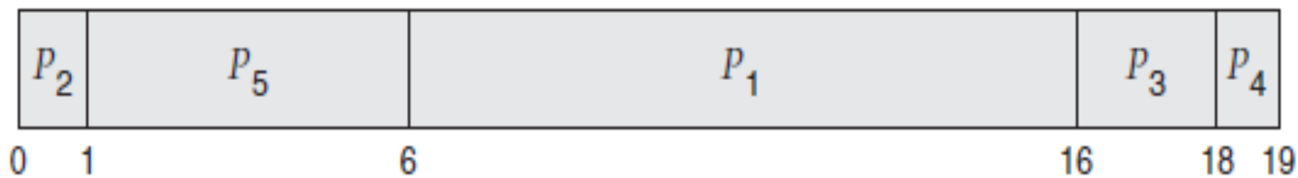
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority

  (smallest integer ≡ highest priority)
- Processes with Equal-priority are scheduled in FCFS.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
  - ***Preemptive***, preempts the CPU if the priority of the newly arrived process is higher
  - ***Nonpreemptive***, puts the new process at the head of the ready queue.
- SJF is a special case of priority scheduling where priority is the inverse of predicted next CPU burst time (i.e., the larger the CPU burst, the lower the priority, and vice versa).
- Problem ≡ **Starvation (indefinite blocking)** – 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



WT for P_1=6 ms
WT for P_2=0 ms
WT for P_3=16 ms
WT for P_4=18 ms
WT for P_5=1 ms

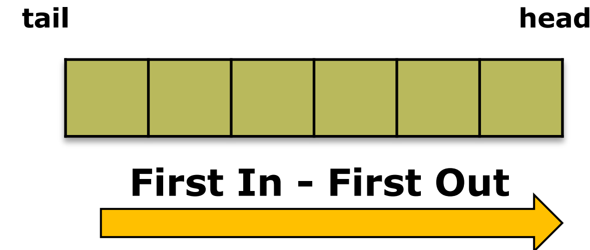
- Average waiting time = (6+0+16+18+1)/5= 8.2 ms
- What is the average waiting time using FCFS and SJF?

Round Robin (RR)

- Designed for time-sharing systems.
- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- 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
  - The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.
- Performance
  - q large $\Rightarrow$ FCFS
  - q small $\Rightarrow$ more context switches, overhead is too high

Round Robin (RR)

- New processes are added to the tail of the ready queue.



One of two things will then happen

The process has a CPU burst of less than 1 time quantum

- process will release the CPU voluntarily.
- The scheduler will then proceed to the next process in the ready queue

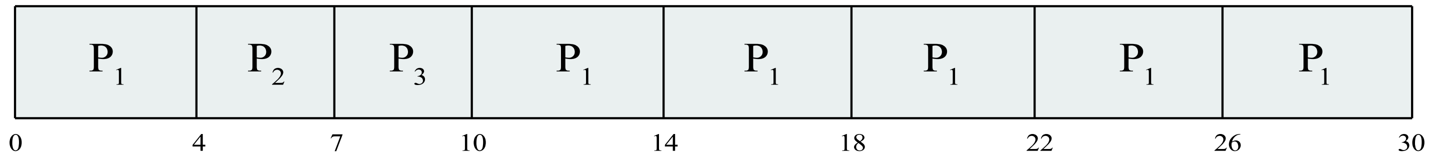
The process has a CPU burst longer than 1 time quantum

- the timer will go off and will cause an interrupt to the operating system
- A context switch will be executed, and the process will be put at the tail of the ready queue.
- The CPU scheduler will then select the next process in the ready queue.

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

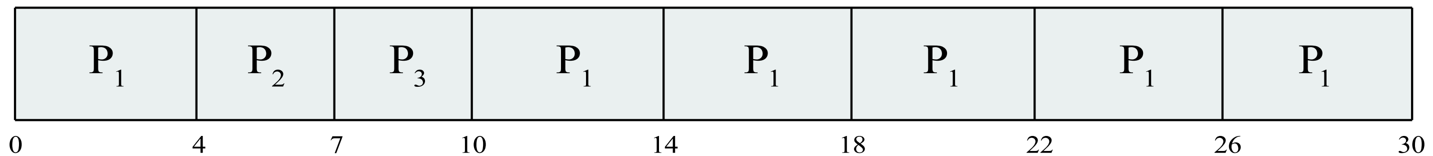


- Average turnaround time= (30+ 7+ 10)/3= 15.67 msec
- Typically, **higher average turnaround time** than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 microseconds (μs)

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



Turnaround time= Completion time - Arrival time

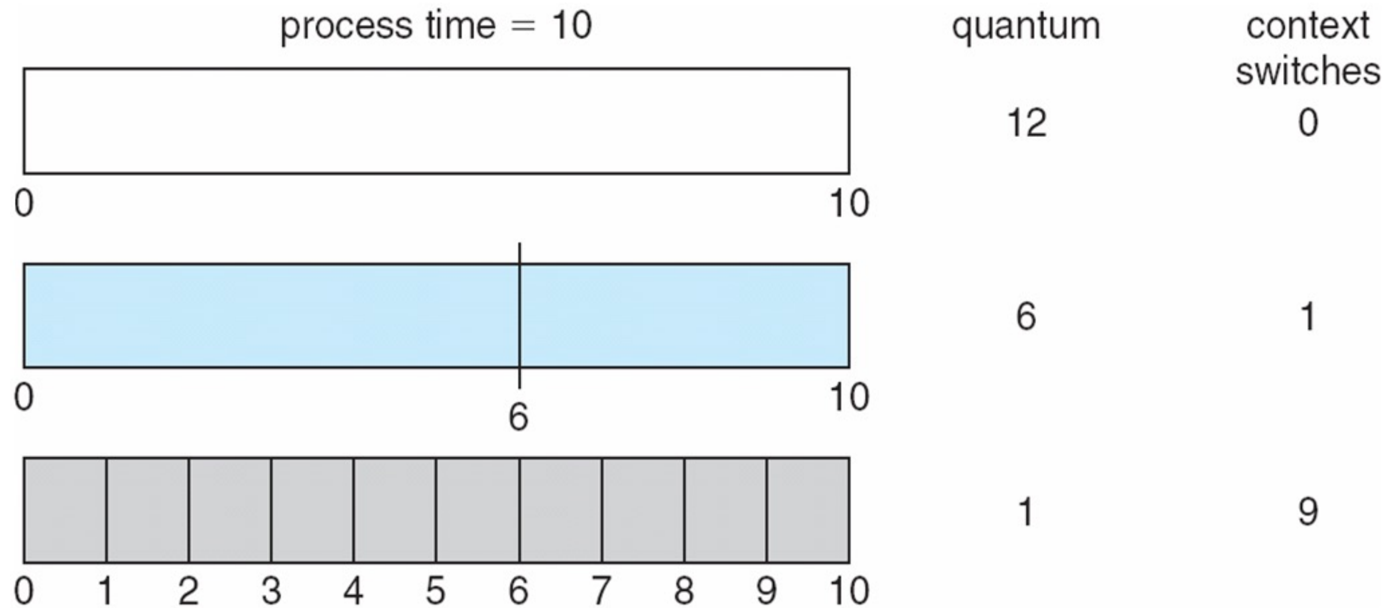
Waiting time= Turnaround time – Burst time

Process	Completion time	Turnaround time	Waiting time
P_1	30	30-0=30	30-24=6
P_2	7	7-0=7	7-3=4
P_3	10	10-0=10	10-3=7

Average Waiting time=(6+4+7)/3

=17/3= 5.66 ms

Time Quantum and Context Switch Time

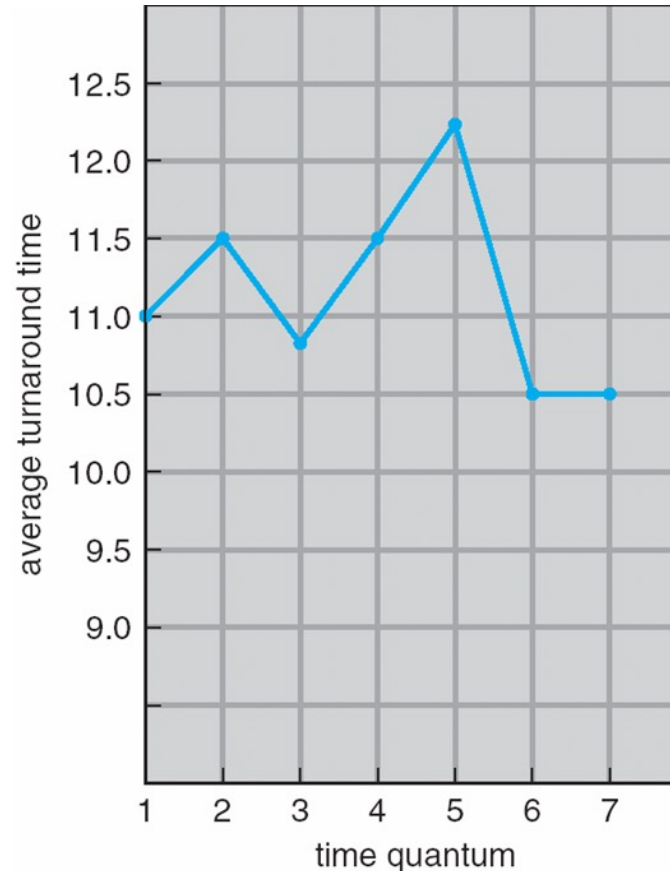


Turnaround Time Varies With The Time Quantum

For Q=1
Average turn around time = 11 msec (prove)

For Q=5
Average turn around time = 12.25 msec (prove)

and so on.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

- Note that the average turnaround time of a set of processes does not necessarily improve as the time-quantum size increases.
- In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum

Preemptive/ Non preemptive

Scheduling	Preemptive/ nonpreemptive
FCFS	nonpreemptive
SJF	may be either preemptive/ nonpreemptive
Priority	may be either preemptive/ nonpreemptive
RR	preemptive

Multilevel Queue

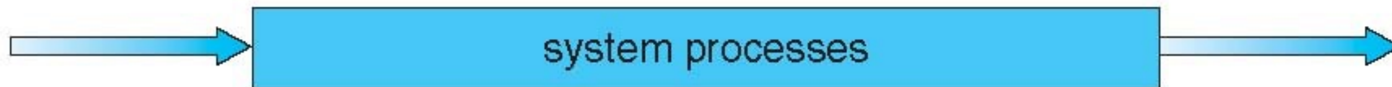
- Ready queue is partitioned into separate queues based on their properties, for example:
  - **foreground** (interactive),
  - **background** (batch)

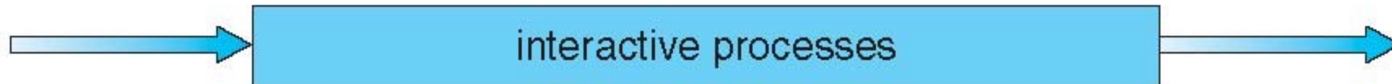
  These two have different response-time requirements ➔ different scheduling needs. Also, foreground processes may have higher priority over background processes.
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- <u>Scheduling must be done between the queues</u>:
  - 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

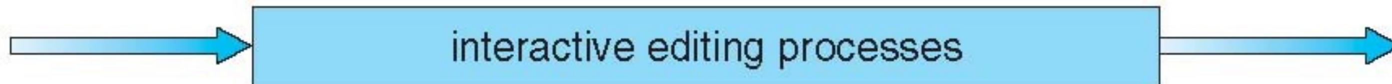
    20% to background in FCFS

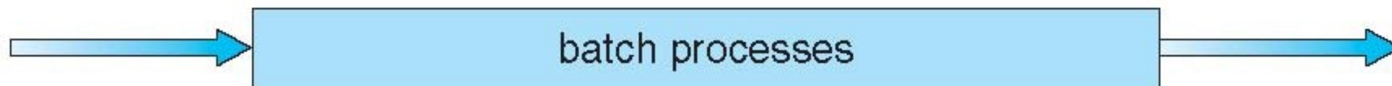
Multilevel Queue Scheduling

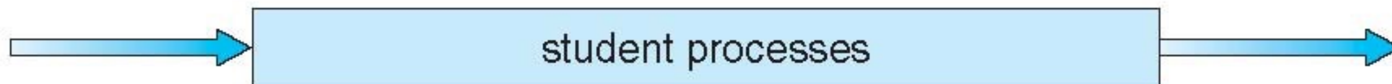
highest priority











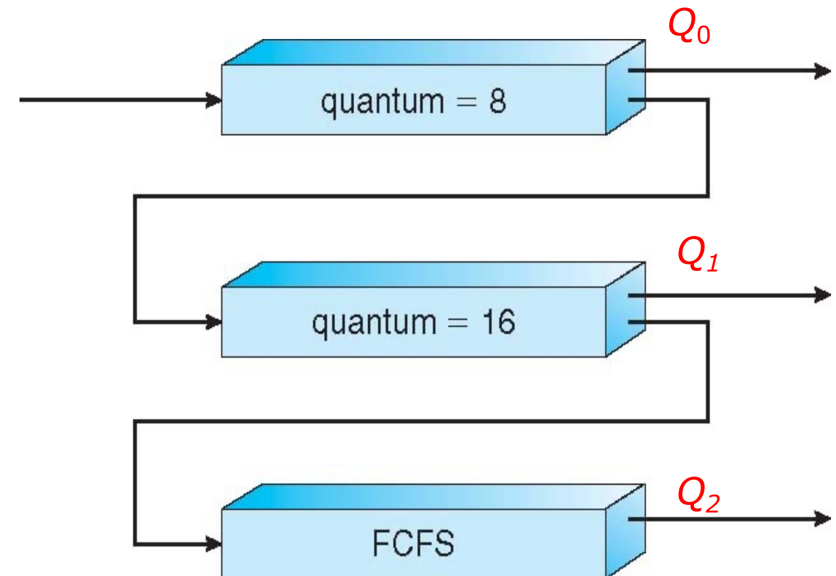
lowest priority

Multilevel Feedback Queue

- **Multilevel feedback queue** scheduling algorithm allows a process to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts.
  - If a process uses too much CPU time, it will be moved to a lower-priority queue →I/O-bound and interactive processes in the higher-priority queues.
  - A process that waits too long in a lower-priority queue may be moved to a higher-priority queue → this form of aging prevents starvation.
- 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

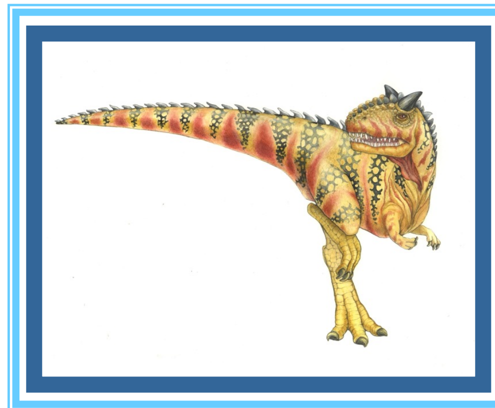
Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 ms
  - Q_1 – RR time quantum 16 ms
  - Q_2 – FCFS



- Scheduling
  - A new job enters queue Q_0
    - When it gains CPU, job receives 8 ms
    - If it does not finish in 8 ms, job is moved to queue Q_1
  - At Q_1 job is again served RR and receives 16 additional ms
    - If it still does not complete, it is preempted and moved to queue Q_2

End of Chapter 6



Silberschatz, Galvin and Gagne ©2013