The picture can't be displayed.

Chapter 5: CPU Scheduling

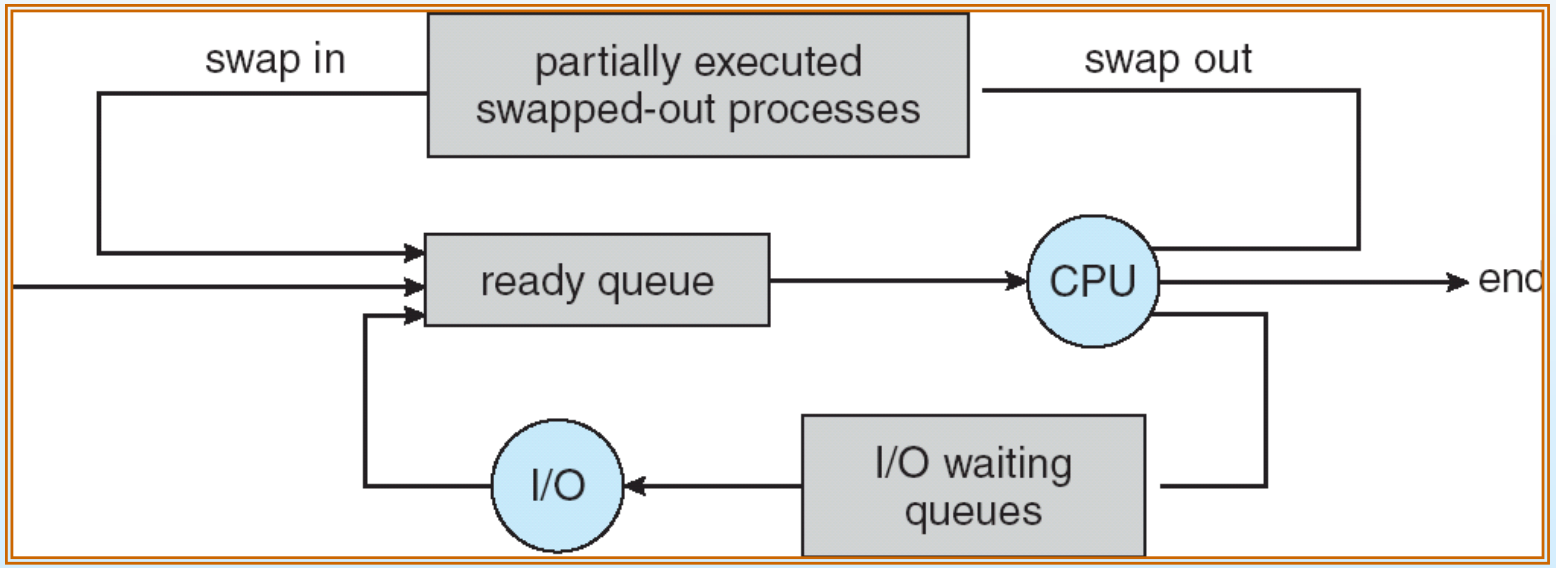
http://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/

(Contains edits for our course)

Schedulers

- **Long-term scheduler** (or job scheduler) – selects processes from pool on mass storage device that should be brought into the ready queue. Loads into memory. (invoked in mins)
  - provides balanced mix of I/O and processor bound jobs.
- **Short-term scheduler** (or CPU scheduler or dispatcher) – selects which process should be executed next and allocates CPU.
  - Faster than long term scheduler. (invoked in milliseconds)
- Medium Term scheduler: Used when you might want to swap a process waiting for I/O to secondary storage to make place for other processes. Done sometimes to improve the process mix

Addition of Medium Term Scheduling



Schedulers (Cont.)

- The long-term scheduler controls the *degree of multiprogramming*

I/O or CPU bound

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in PCB so that a process execution can be resumed from the same point later

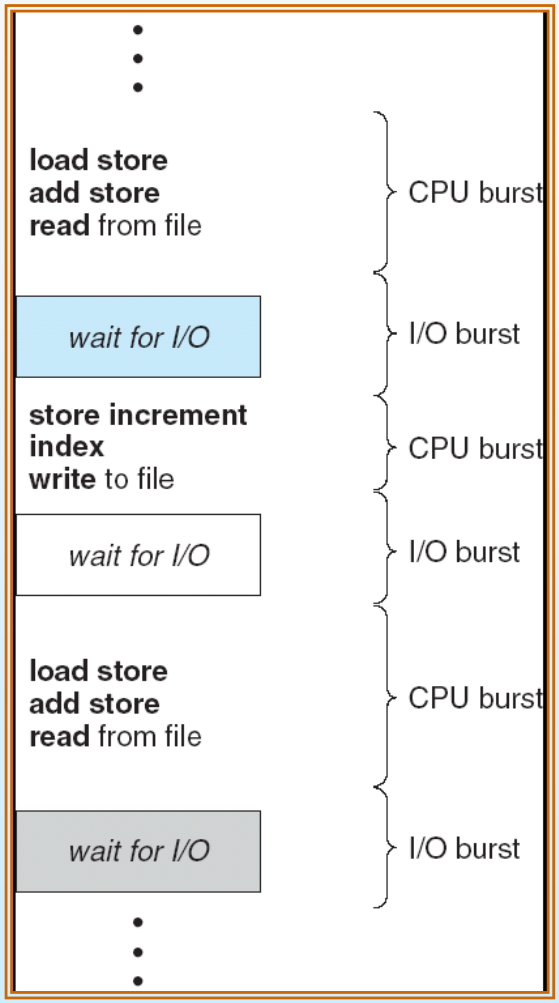
Context-switch time is overhead; the system does no useful work while switching. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers

Time dependent on hardware support

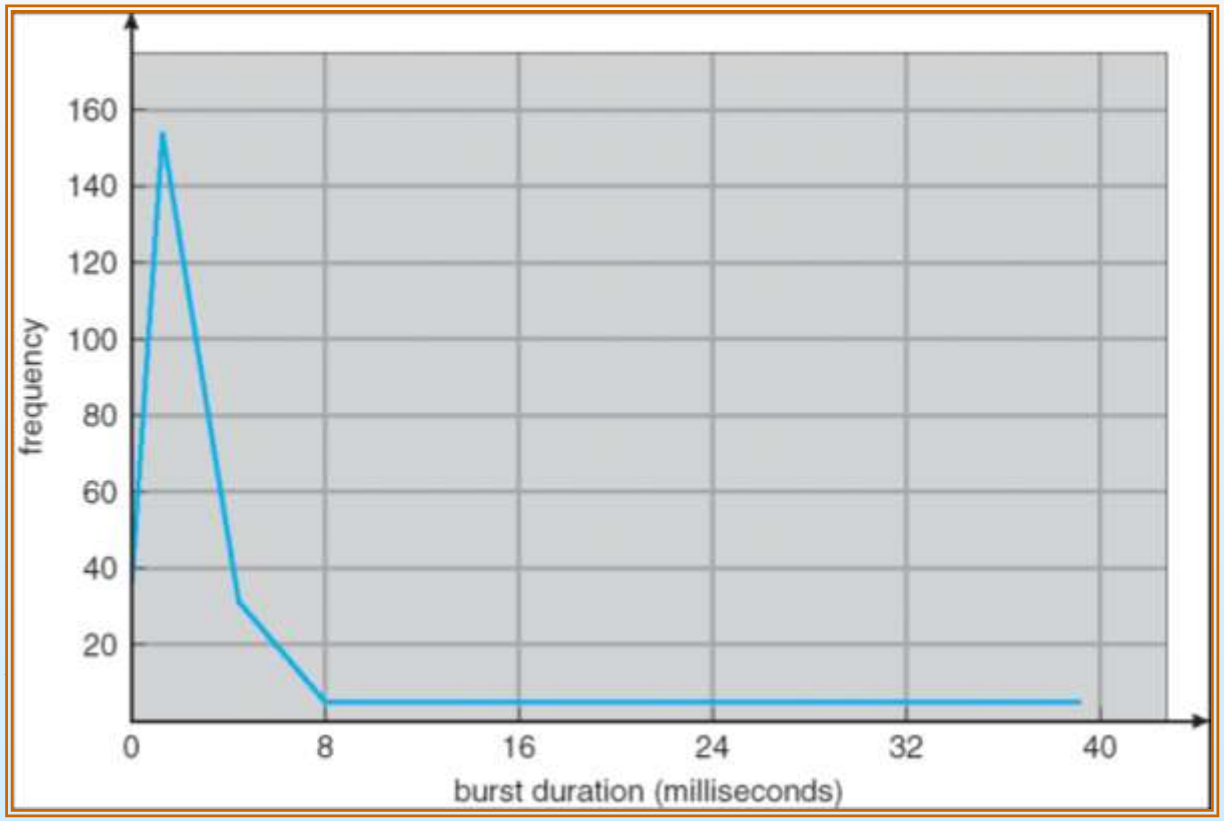
Basic Concepts

- 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 distribution: All processes mostly alternate between cpu bursts of performing calculations and I/O burst for data transfer in and out of the system.

Alternating Sequence of CPU And I/O Bursts



Histogram of CPU-burst Times



CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state (due to an interrupt or I/O)
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates

** An **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An **interrupt** alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context (issuing interrupt and updating PCB)
  - switching to user mode(processor calculates next process to run and the process is woken up)
  - jumping to the proper location in the user program to restart that program (loading PCB and moving to run state)
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

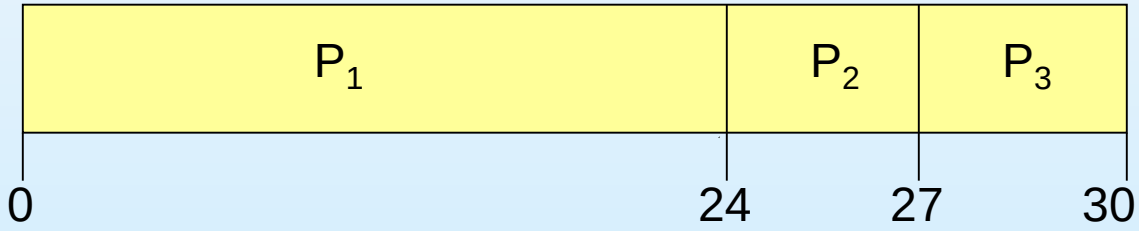
- CPU utilization(maximise) – keep the CPU as busy as possible
- Throughput(maximise) – # of processes that complete their execution per time unit
- Turnaround time(minimise) – amount of time to execute a particular process – from time of submission till finish
- Waiting time(minimise) – amount of time a process has been waiting in the ready queue.
- Response time(minimise) – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

The response time is equal to the waiting time if the process never went from running to ready. Else, the waiting time is the sum of the time it spent in the ready queue.

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:



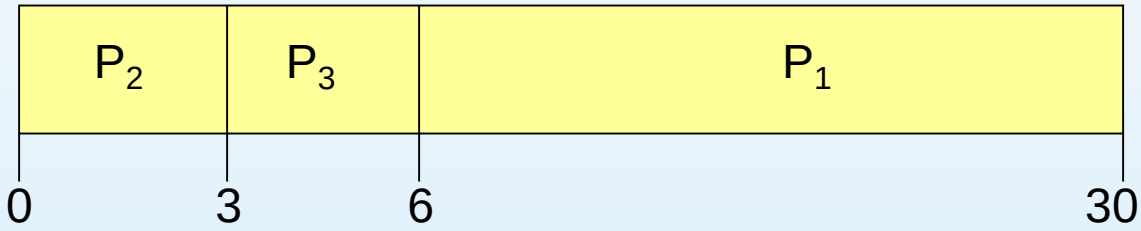
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

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

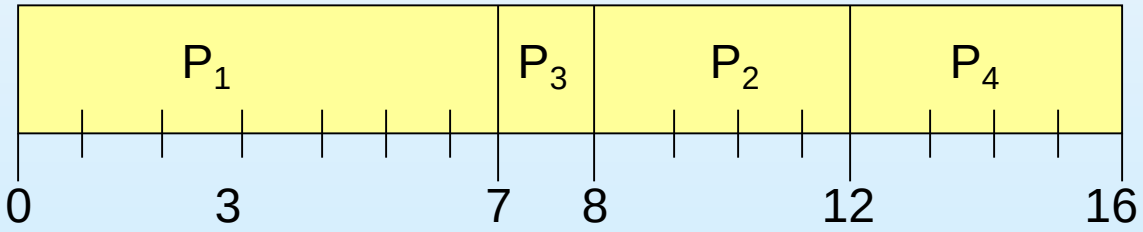
Shortest-Job-First (SJR) Scheduling

- 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)

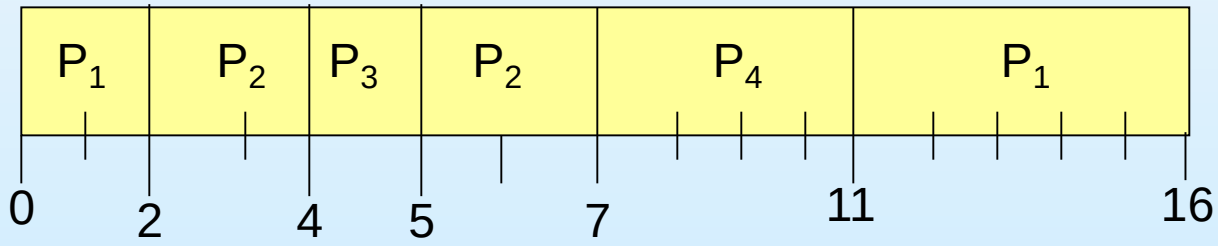


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

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

Determining Length of Next CPU Burst

- Can estimate next burst by using the length of previous CPU bursts, using exponential averaging
  1. t_n = actual lenght of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next CPU burst
  3. $\alpha, 0 \le \alpha \le 1$
  4. Define: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n.$

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

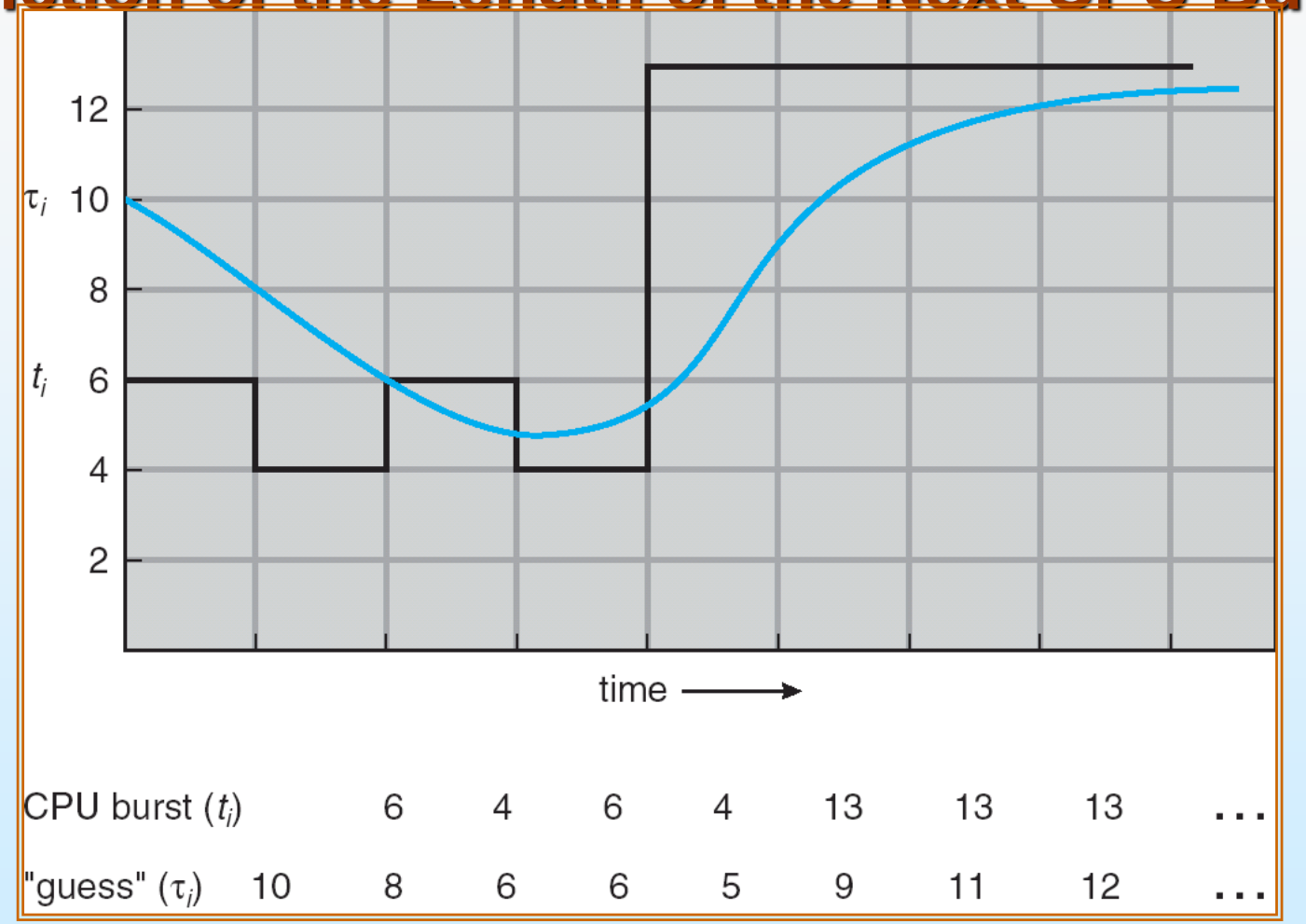
In this scheme the previous estimate contains the history of all previous times, and alpha serves as a weighting factor for the relative importance of recent data versus past history.

If alpha is 1.0, then past history is ignored

If alpha is 0.0, then all measured burst times are ignored, and we just assume a constant burst time.

Most commonly alpha is set at 0.5, as illustrated in Figure 5.3:

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

In-Class Problems

ID	AT	BT
1	5	5
2	4	6
3	3	7
4	1	9
5	2	2
6	6	3

Find the avg waiting time, avg response time, avg turn around time for the above for the below cases

a. FCFS

b. SJF non pre-emptive

c. SJF pre-emptive

Round Robin (RR)

- 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 $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

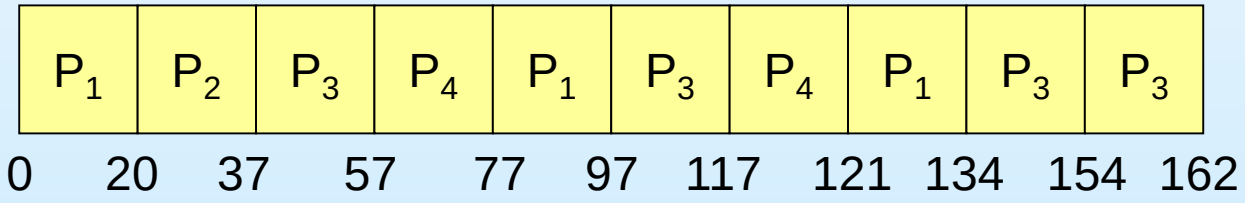
Round Robin (RR)

- Typically 20-40ms is the quantum time allocated
- Attempt to pick quantum time such that about 80% jobs finish in one quantum of time

Example of RR with Time Quantum = 20

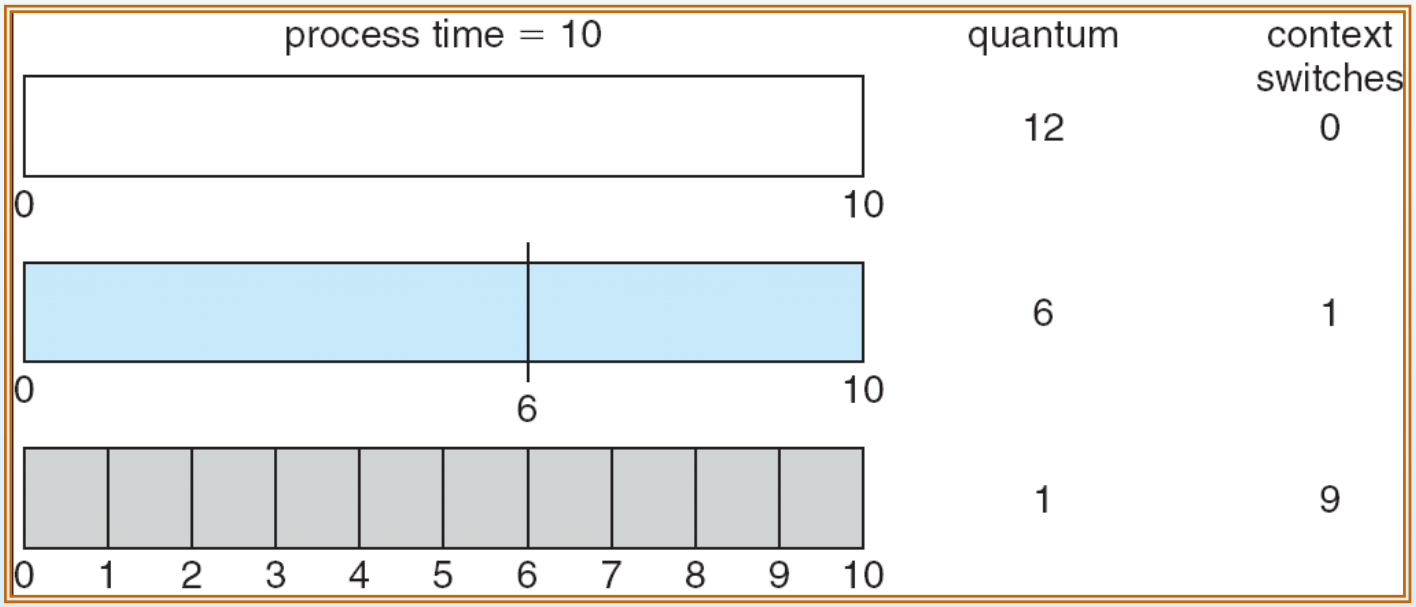
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



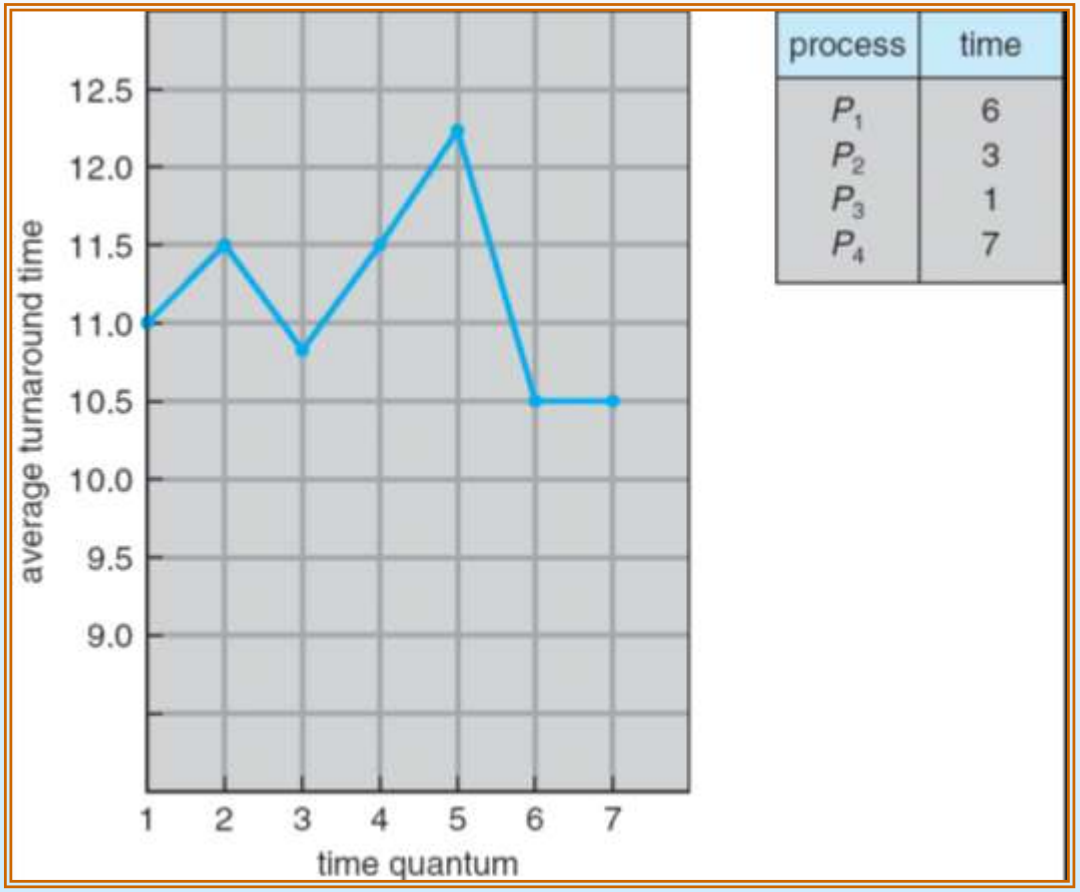
- Typically, higher average turnaround than SJF, but better *response*

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum

Does not necessarily improve with larger time quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Example of irregular TAT

Example for decrease in the average turn round time :

Suppose we have two process P1 and P2 with burst time 20 sec and 2 sec respectively. P1 arrives at time t= 0 sec and P2 at t=2 sec.
When time quantum for RR = 1 sec.
Then TAT for P1 = 22 sec. and for P2 = 3 sec.
So AVG TAT = **(22 + 3) /2 sec**.
Now do same with time quantum for RR = 2 sec.
You will get AVG TAT = **(22 + 2)/2 sec**.

Example for increase in the average turn round time :

Suppose we have two process P1 and P2 with burst time 20 sec and 5 sec respectively. P1 arrives at time t= 0 sec and P2 at t=5 sec.
When time quantum for RR = 5 sec.
Then TAT for P1 = 25 sec. and for P2 = 5 sec.
So AVG TAT = **(25 + 5) /2 sec.**
Now do same with time quantum for RR = 6 sec.
You will get AVG TAT = **(25 + 6)/2 sec.**

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 shortest CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

In-Class Problems

ID	AT	BT	Priority
1	5	5	1
2	4	6	3
3	3	7	4
4	2	9	1
5	1	2	2
6	6	3	3

Find the avg waiting time, avg response time, avg turn around time for the above for the below cases

a. Priority Schedule non pre-emptive

(higher priority dominates)

b. Priority Schedule pre-emptive

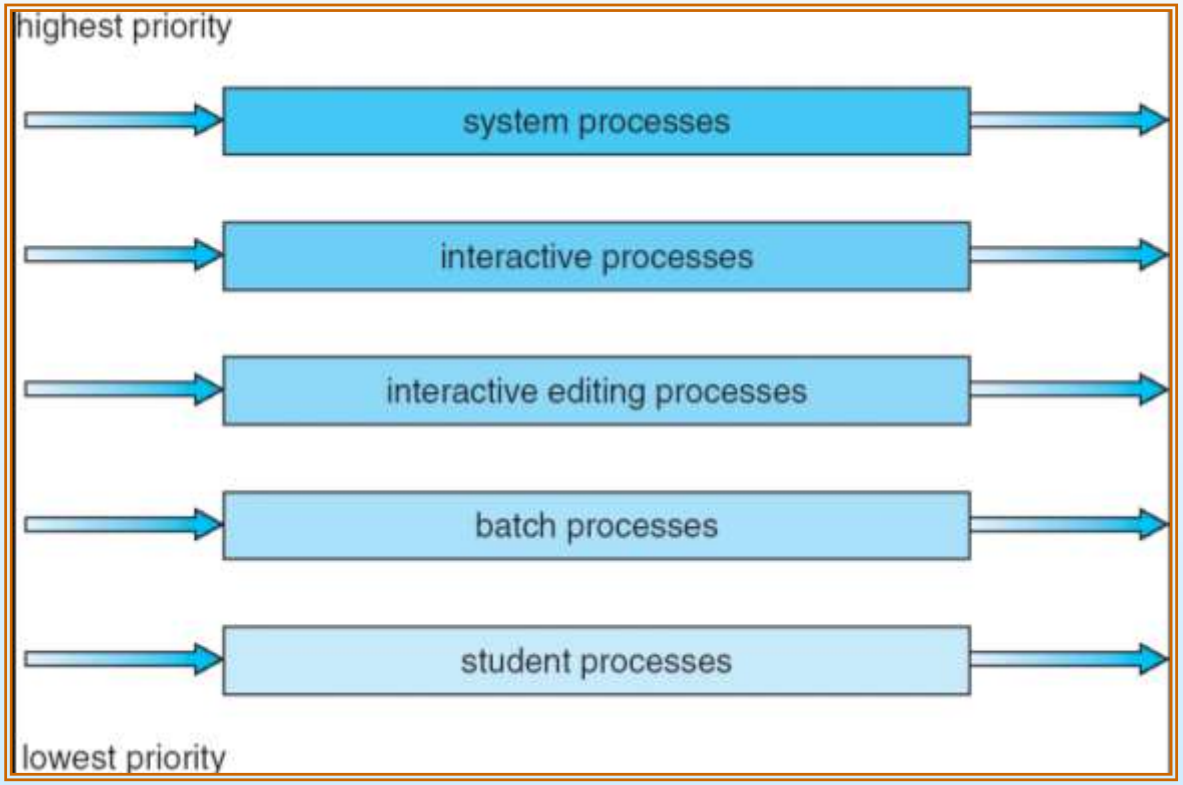
c. Round Robin with time quantum as 3

Multilevel Queue

- 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.
  - RoundRobin with varying 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.

Multilevel Feedback Queues