OPERATING SYSTEMS
CSE-4041

Lecture-6

CPU Scheduling

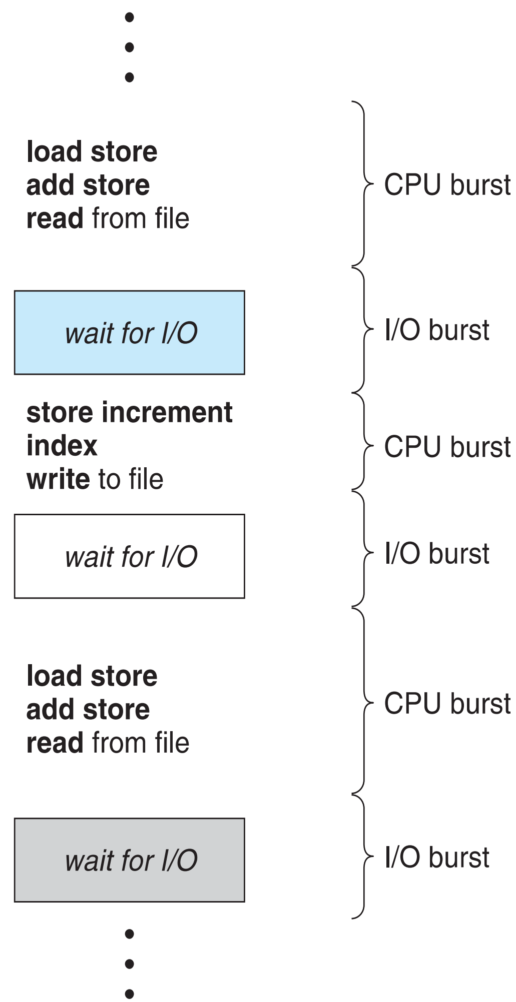
Nitesh Kumar Jha

Assistant Professor

ITER S'O'A (Deemed to be University)

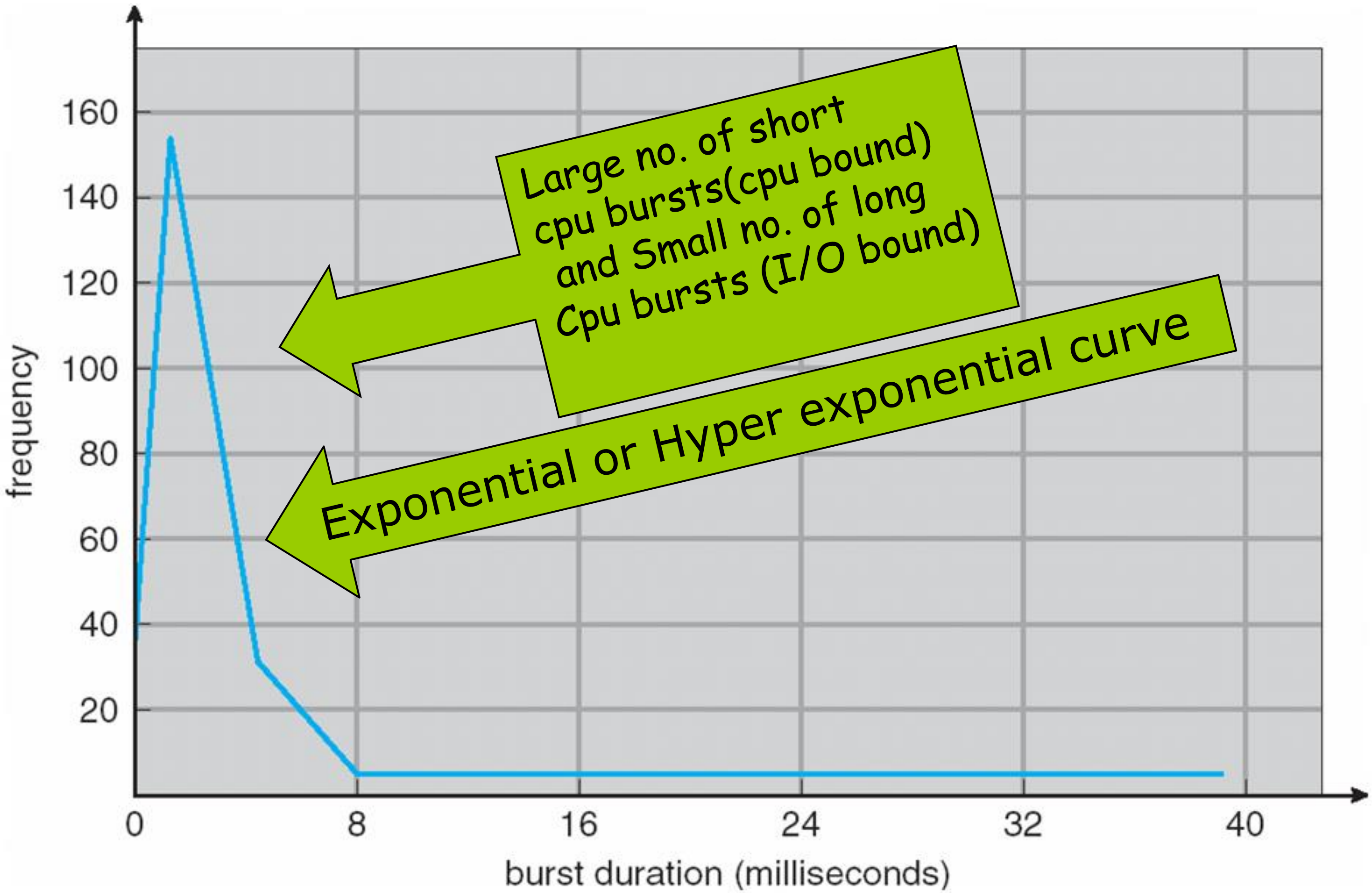
CPU SCHEDULING

- 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** followed by **I/O burst**
- CPU burst distribution is of main concern



ALTERNATE sequence for CPU and I/O bursts

Histogram of CPU-burst Times



Histogram of CPU-burst durations

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(I/O request or execution of fork())
  2. Switches from running to ready state(Interrupt occurs)
  3. Switches from waiting to ready(completion of I/O)
  4. Process Terminates
- Scheduling under 1 and 4 is **non preemptive**
- 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

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible, 40% lightly loaded-90% heavily loaded.
- **Throughput** – # of processes that complete their execution per time unit. For long process it may be 1 process per hour or may be 10 processes per second(short processes).
- **Turnaround time** – amount of time to execute a particular process i.e the time starting from the submission of process upto its termination.
  - TAT = completion time – Arrival time
- **Waiting time** – amount of time a process has been waiting in the ready queue.
  - WT = TAT – Burst time
- **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

Ideally for any CPU scheduling Algorithm it must satisfy below:

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

CPU scheduling Algorithms:

It is the task of selecting any one process from the ready queue and allocating the cpu for execution.

First- Come, First-Served (FCFS) Scheduling

- Simplest of all scheduling algorithms.
- First process to enter the ready queue shall be allocated the CPU.
- This algorithm is non preemptive and doesn't suit time sharing system.

Consider three process P_1, P_2 and P_3 with their respective burst times

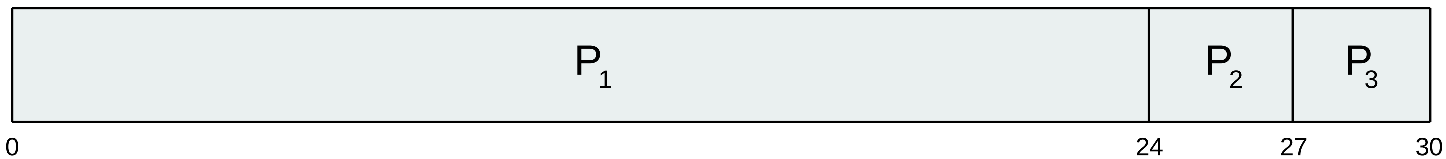
Process	Burst Time
P_1	24
P_2	3
P_3	3

What are the possible order in which processes arrive?

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:



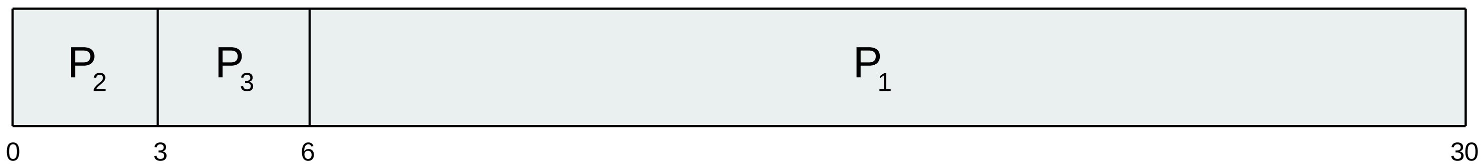
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
- Turn around time P_1 = 24; P_2 = 27; P_3 = 30
- Average turn around time: (24+27+30)/3=27

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

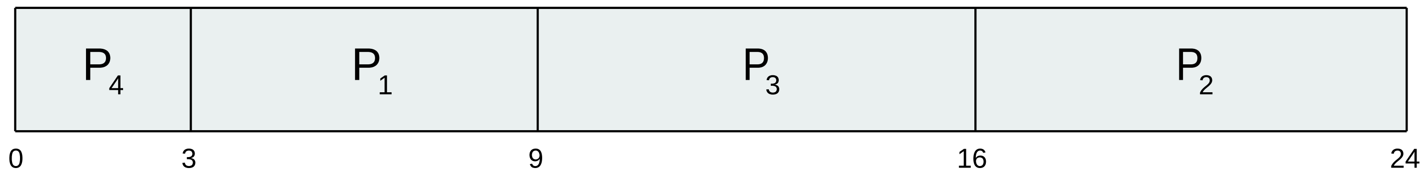
Shortest-Job-First (SJF) Scheduling

- 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
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

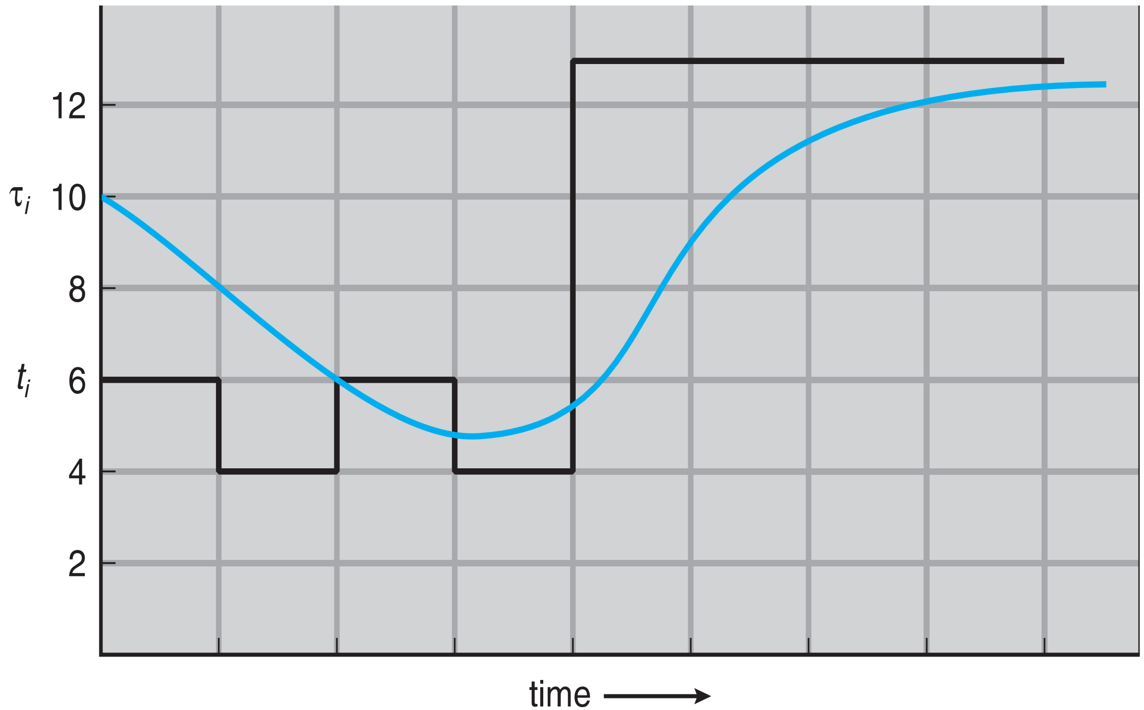
Implementation of SJF

- This Algorithm is optimal in terms of average waiting time for a set of processes.
- It is not suitable for CPU scheduling cause it is impossible to calculate or predict the next CPU time of the process waiting in the ready queue.
- The SJF algorithm is somehow used for long term scheduling performed by the job scheduler where the burst time of a process can be calculated.
- Although SJF is impossible one approach of approximation 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. $\alpha, 0 \leq \alpha \leq 1$
4. Define: $\tau_{n=1} = \alpha\, t_n + (1 - \alpha)\tau_n.$

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first(SRTF)**

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:

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

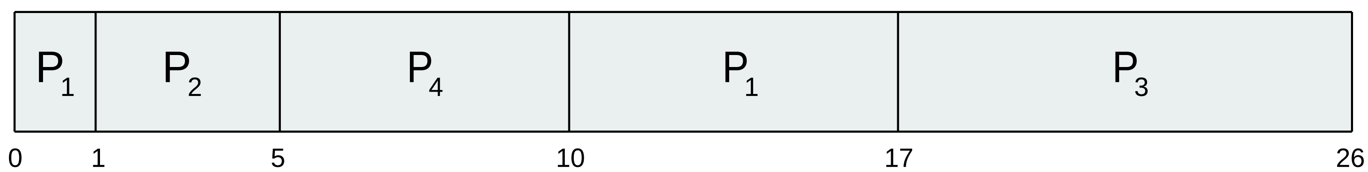
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

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 milli seconds

Priority Scheduling

- A priority number (integer) is associated with each process, The CPU is allocated to the process with the highest priority (smallest integer $\equiv$ highest priority)
- Priority assigned to a process may either be
  - internally defined (by the OS)
    - ratio of CPU to I/O burst
    - Overall memory requirements
  - Externally defined
    - Based on importance of process
    - Funding or sponsorship or other administrative factors
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time

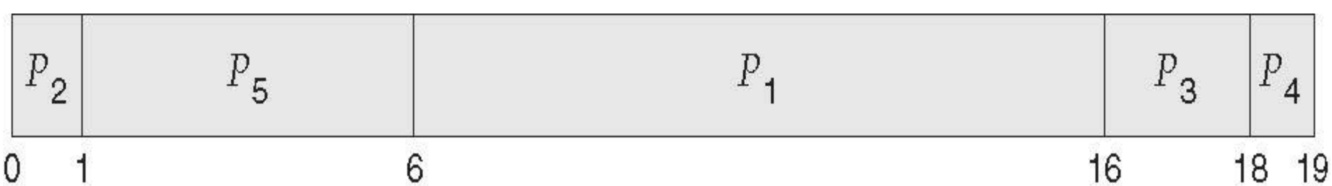
Priority Scheduling

- Priority scheduling can either be
  - Preemptive
  - Non preemptive
- In cases where two or more processes are found to have same priority then FCFS logic is used to break the tie.
- Problem ≡ **Starvation** – In certain cases low priority processes may never execute as there is a flow of higher priority processes.
- Solution ≡ ***Aging*** – as time progresses increase the priority of the process.

Priority Scheduling(Non Preemptive)

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 m.sec

Priority Scheduling(Preemptive)

Process	Arrival Time	Priority	Burst time
P_1	0	3	10
P_2	1	1	1
P_3	2	4	2
P_4	4	2	3

- Priority scheduling Gantt Chart

- Average waiting time = 4 m.sec

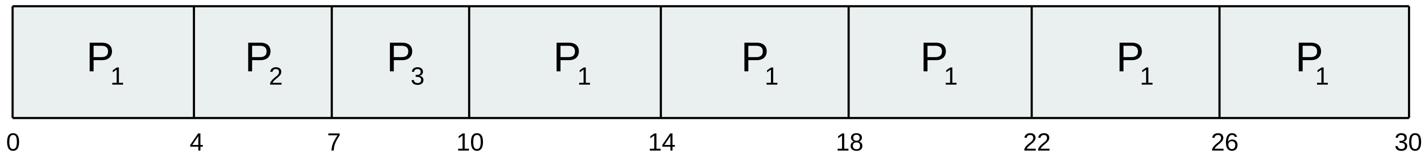
Round Robin (RR)

- 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

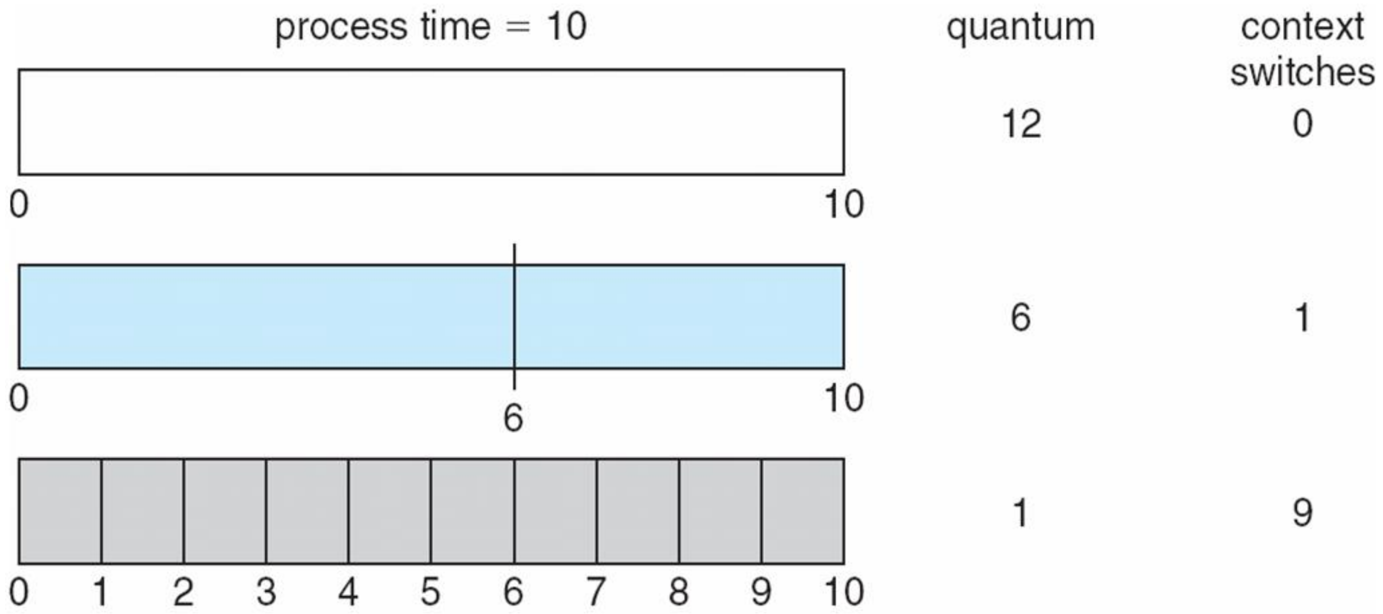
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

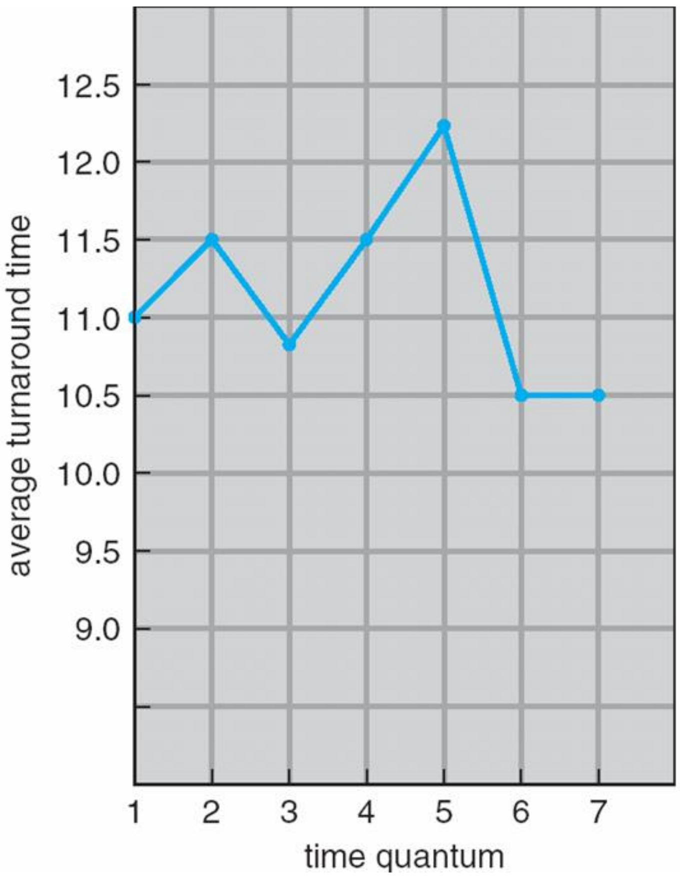


- Typically, higher average turnaround than SJF, but better response
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



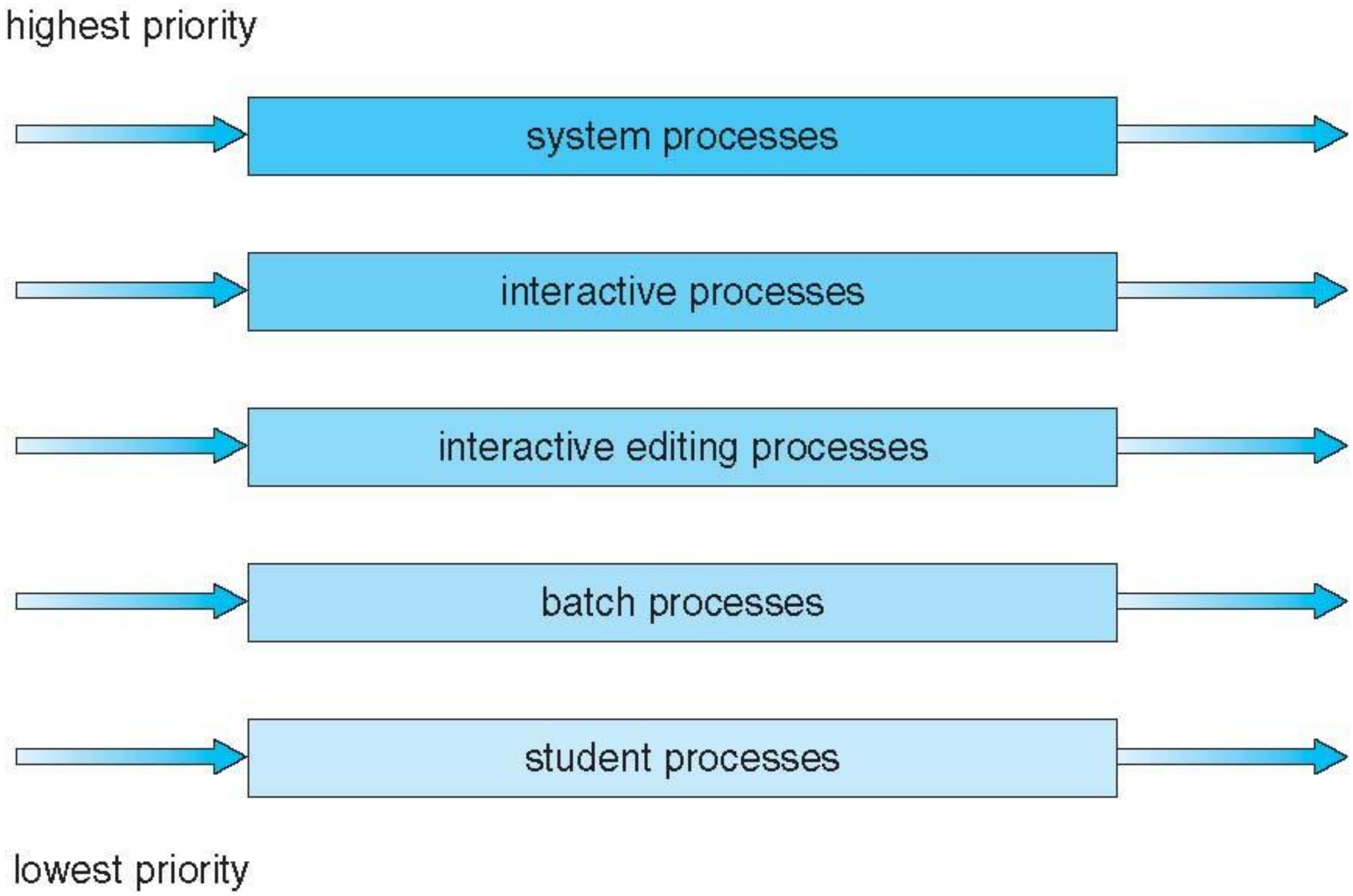
process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q

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

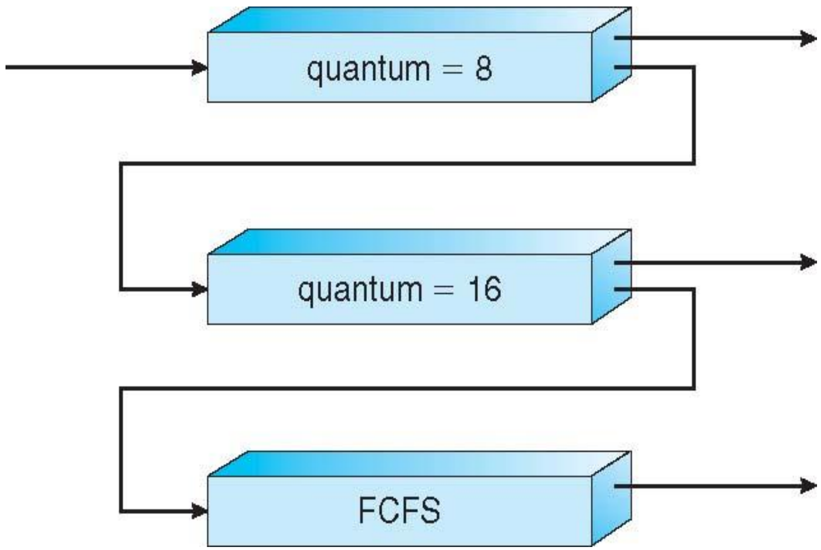


Multilevel Feedback Queue

- 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

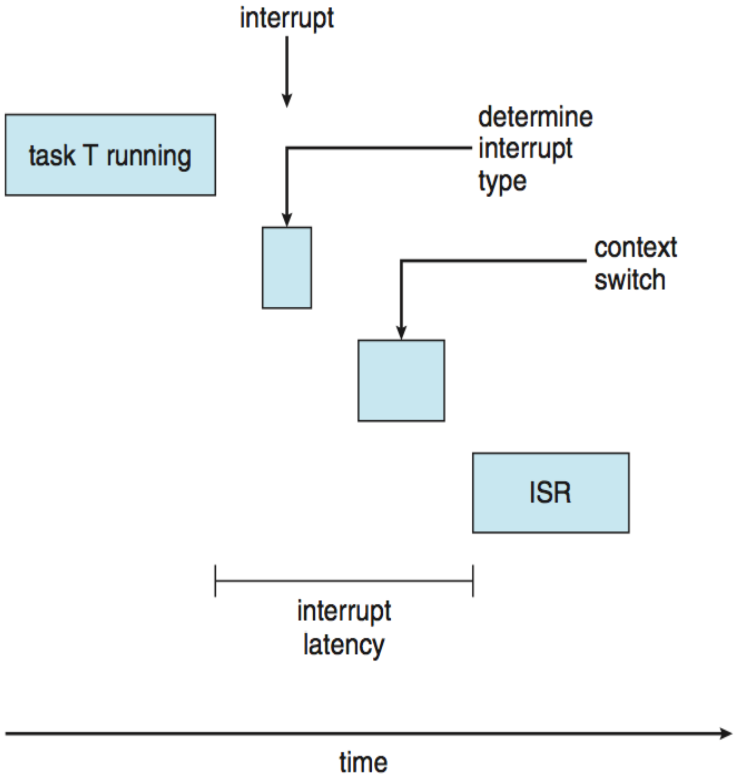


Thread Scheduling

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as **process-contention scope** (**PCS**) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope** (**SCS**) – competition among all threads in system

Real-Time CPU Scheduling

- Can present obvious challenges
- **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline
- Two types of latencies affect performance
  1. Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for schedule to take current process off CPU and switch to another



End of Lecture

Thank You