COS : Day 3

CPU Scheduling

Dr Kiran Waghmare
CDAC Mumbai

Start Slide

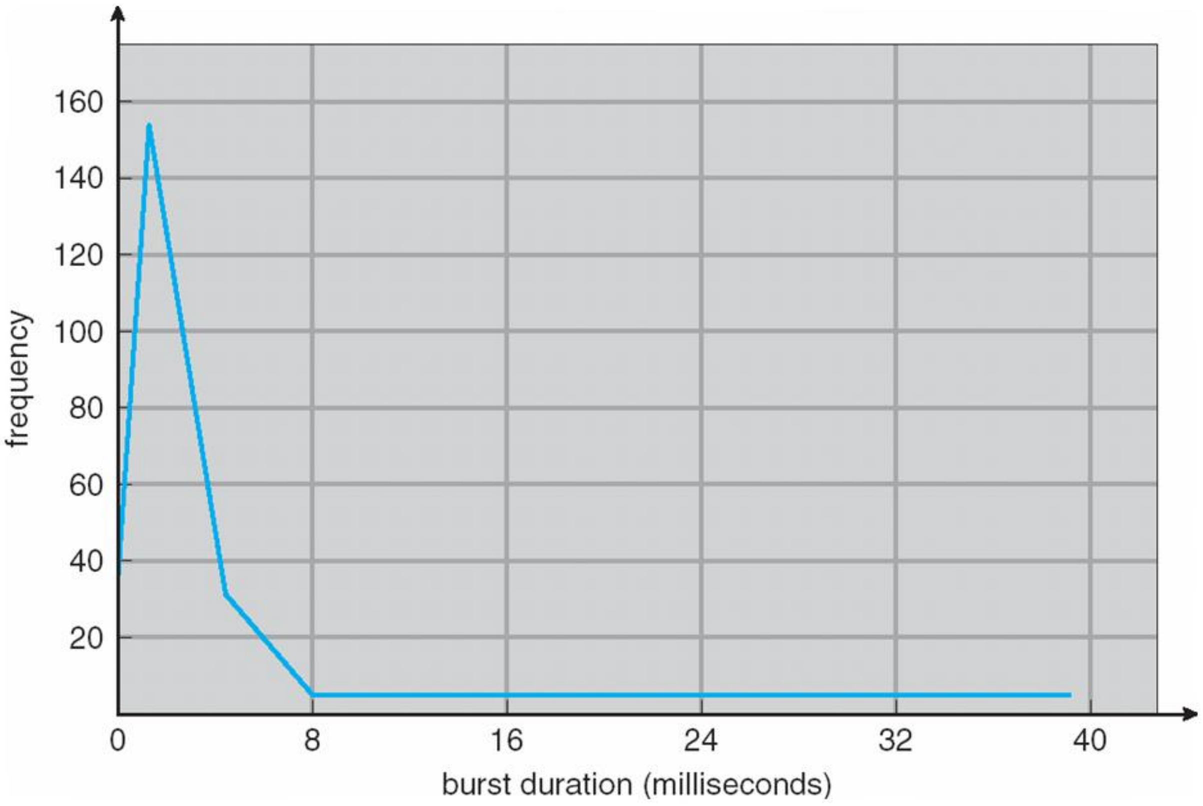
Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Algorithm Evaluation

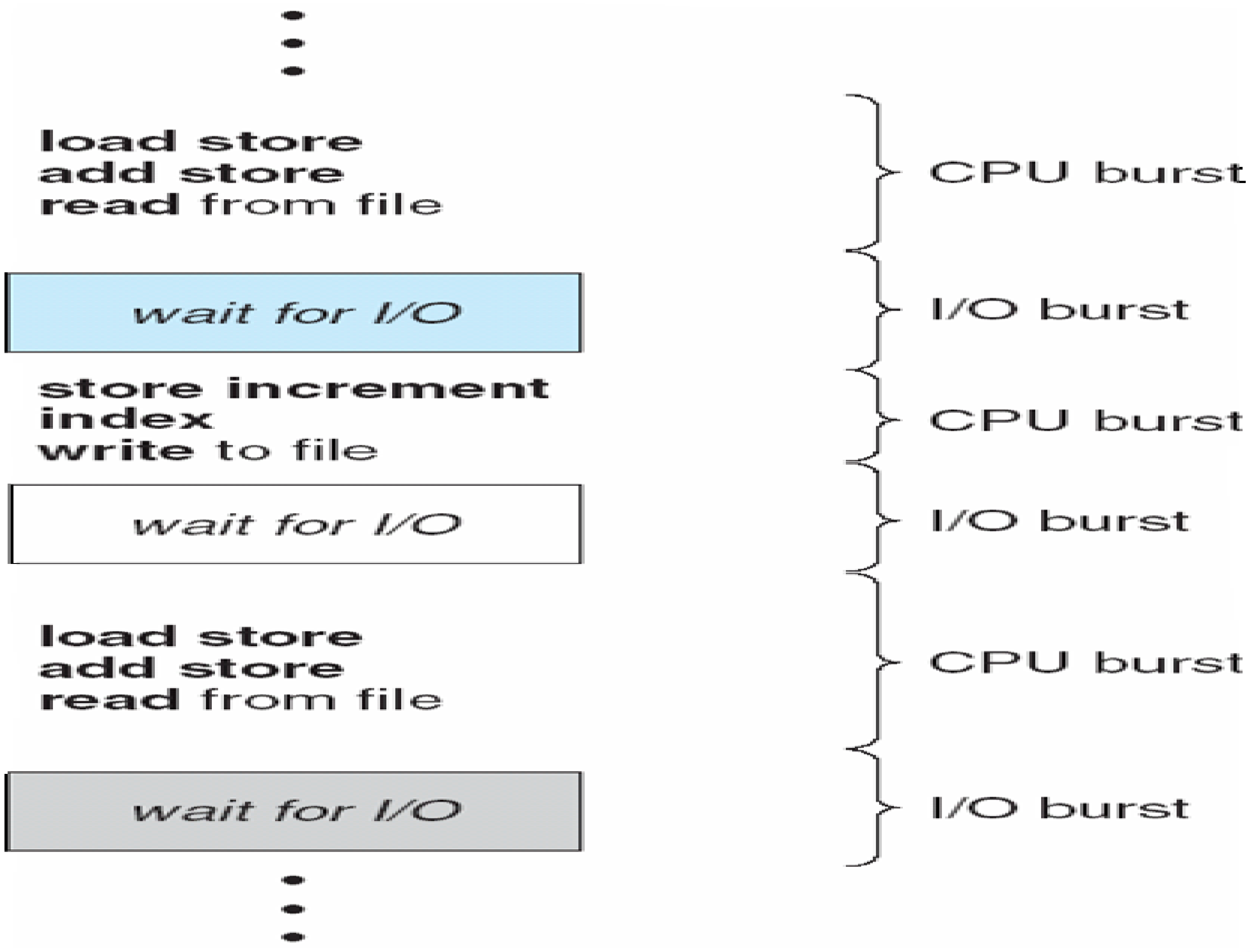
Basic Concepts

- 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

Histogram of CPU-burst Times



Alternating Sequence of CPU And I/O Bursts



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**
  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**

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
- **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

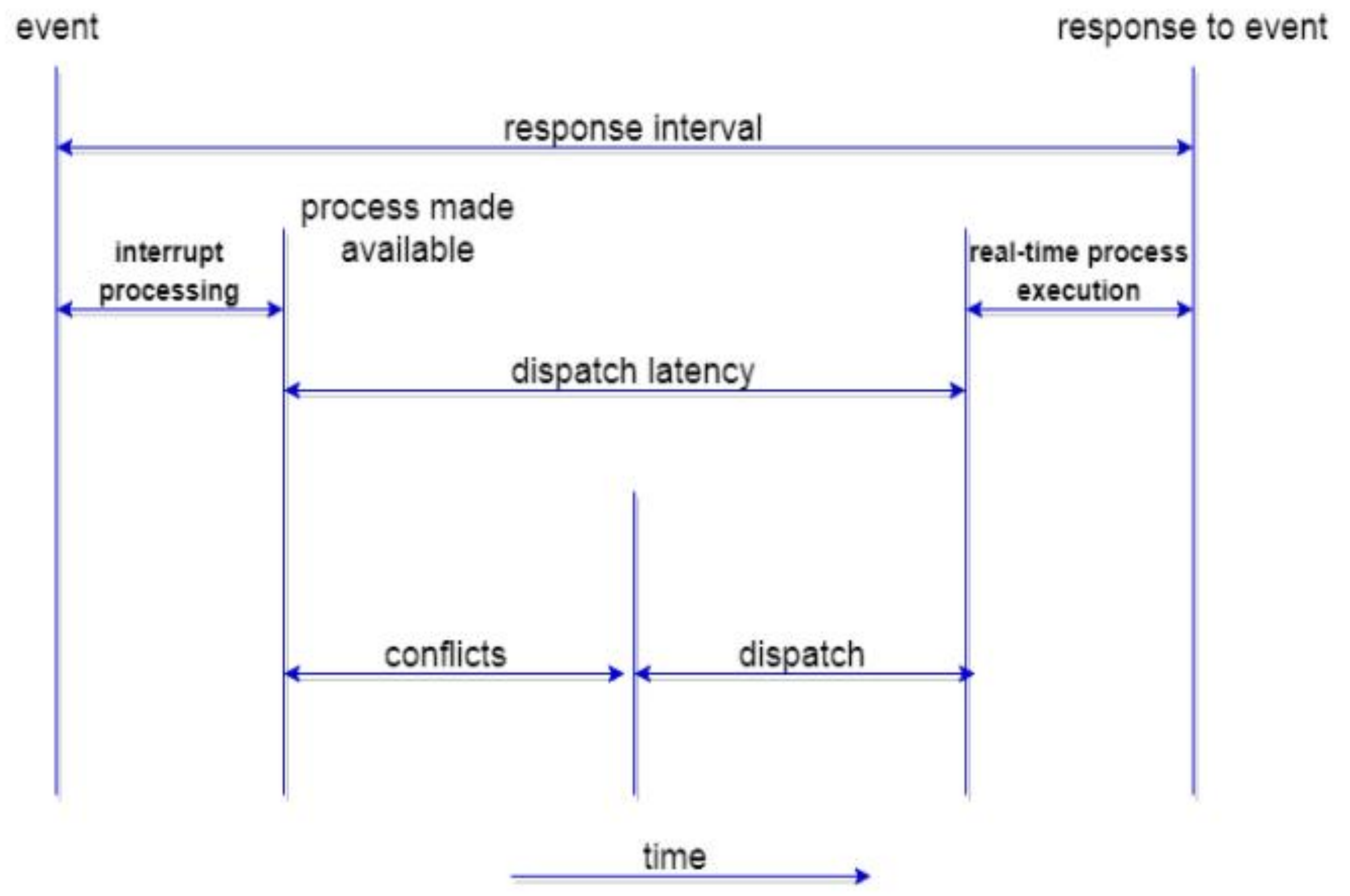
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

CPU Scheduling in Operating System

- CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU.
- The aim of CPU scheduling is to make the system efficient, fast, and fair.
- Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed.
- The selection process is carried out by the short-term scheduler (or CPU scheduler).
- The scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

CPU Scheduling: Dispatcher

- Another component involved in the CPU scheduling function is the **Dispatcher**.
- The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:
  - Switching context
  - Switching to user mode
- Jumping to the proper location in the user program to restart that program from where it left last time.
- The dispatcher should be as fast as possible, given that it is invoked during every process switch.
- The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency**.
- Dispatch Latency can be explained using the below figure:


event
response to event
response interval
process made available
interrupt processing
real-time process execution
dispatch latency
conflicts
dispatch
time

Types of CPU Scheduling

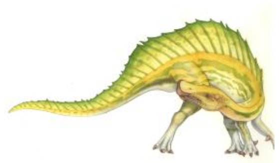
- CPU scheduling decisions may take place under the following four circumstances:

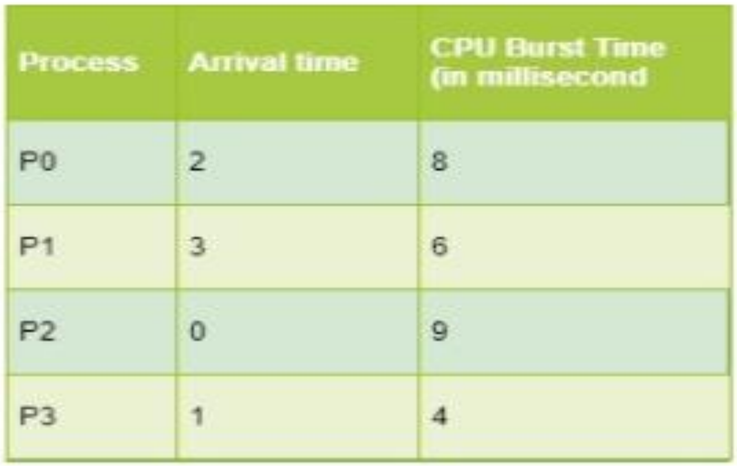
1. When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
4. When a process **terminates**.

- In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.
- When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise, the scheduling scheme is **preemptive**.

Non-Preemptive Scheduling

- Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
- This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.
- It is the only method that can be used on certain hardware platforms because It does not require the special hardware(for example a timer) needed for preemptive scheduling.
- In non-preemptive scheduling, it does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then after that it can allocate the CPU to any other process.
- Some Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non-preemptive) Scheduling and Priority (non- preemptive version) Scheduling, etc.





Process	Arrival time	CPU Burst Time (in millisecond
P0	2	8
P1	3	6
P2	0	9
P3	1	4

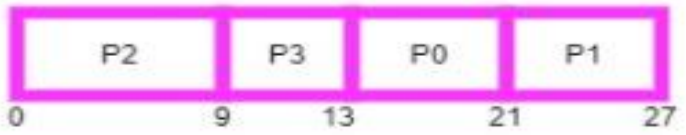
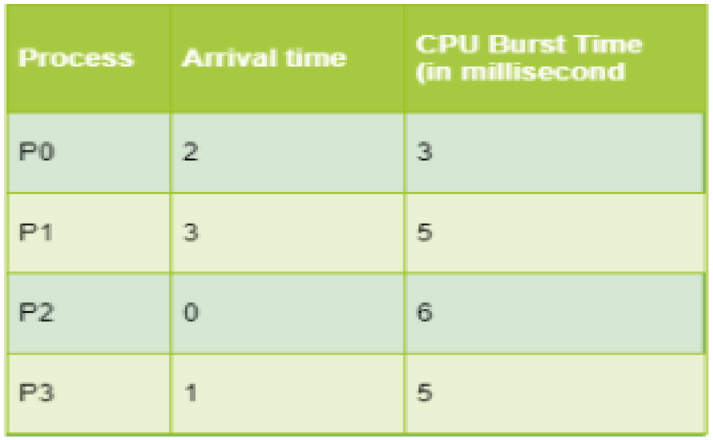


Figure: Non-Preemptive Scheduling



Process	Arrival time	CPU Burst Time (in millisecond
P0	2	3
P1	3	5
P2	0	6
P3	1	5

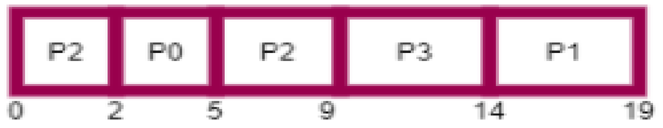


Figure: Preemptive Scheduling

Preemptive Scheduling

- In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.
- Thus this type of scheduling is used mainly when a process switches either from running state to ready state or from waiting state to ready state. The resources (that is CPU cycles) are mainly allocated to the process for a limited amount of time and then are taken away, and after that, the process is again placed back in the ready queue in the case if that process still has a CPU burst time remaining. That process stays in the ready queue until it gets the next chance to execute.
- Some Algorithms that are based on preemptive scheduling are Round Robin Scheduling (RR), Shortest Remaining Time First (SRTF), Priority (preemptive version) Scheduling,

CPU Scheduling: Scheduling Criteria

- There are many different criteria to check when considering the **"best"** scheduling algorithm, they are:
- **CPU Utilization**
- To make out the best use of the CPU and not to waste any CPU cycle, the CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- **Throughput**
- It is the total number of processes completed per unit of time or rather says the total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.
- **Turnaround Time**
- It is the amount of time taken to execute a particular process, i.e. The interval from the time of submission of the process to the time of completion of the process(Wall clock time).
- **Waiting Time**
- The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
- **Load Average**
- It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- **Response Time**
- Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).
- In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

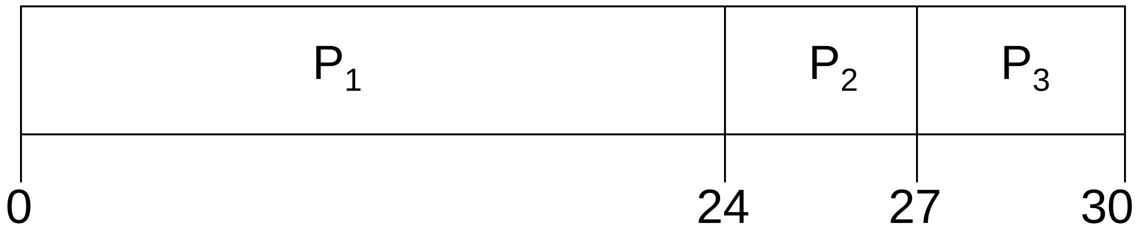
Scheduling Algorithms

- First Come First Serve(FCFS) Scheduling
- Shortest-Job-First(SJF) Scheduling
- Priority Scheduling
- Round Robin(RR) Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling
- Shortest Remaining Time First (SRTF)
- Longest Remaining Time First (LRTF)
- Highest Response Ratio Next (HRRN)

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:



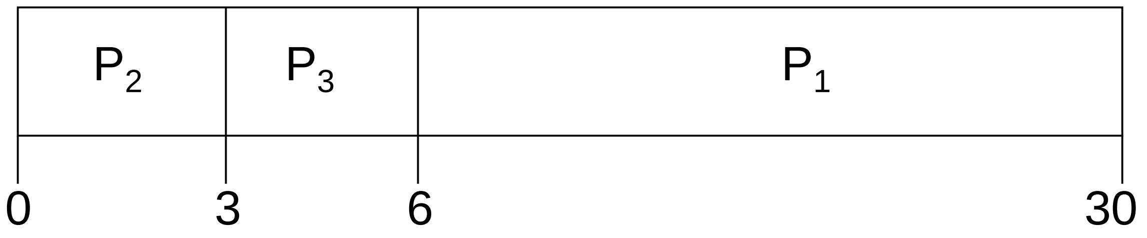
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

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

First Come First Serve Scheduling

- In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.
- First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the head of the queue.
- A perfect real life example of FCFS scheduling is **buying tickets at ticket counter**.

Calculating Average Waiting Time

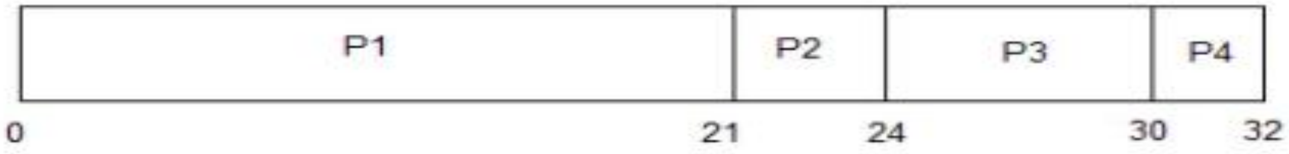
- For every scheduling algorithm, Average waiting time is a crucial parameter to judge it's performance.

- AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution.

- Lower the Average Waiting Time, better the scheduling algorithm.
- Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time, let's find the average waiting time using the FCFS scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/ 4 = 18.75 ms



This is the GANTT chart for the above processes

Problems with FCFS Scheduling

- Below we have a few shortcomings or problems with the FCFS scheduling algorithm:

1. It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter.If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.
2. Not optimal Average Waiting Time.
3. Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource(CPU, I/O etc) utilization.

- **What is Convoy Effect?**
- Convoy Effect is a situation where many processes, who need to use a resource for short time are blocked by one process holding that resource for a long time.
- This essentially leads to poort utilization of resources and hence poor performance.

- Here we have simple formulae for calculating various times for given processes:
- **Completion Time:** Time taken for the execution to complete, starting from arrival time.
- **Turn Around Time:** Time taken to complete after arrival. In simple words, it is the difference between the Completion time and the Arrival time.
- **Waiting Time:** Total time the process has to wait before it's execution begins. It is the difference between the Turn Around time and the Burst time of the process.
- For the program above, we have considered the arrival time to be 0 for all the processes, try to implement a program with variable arrival times.

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

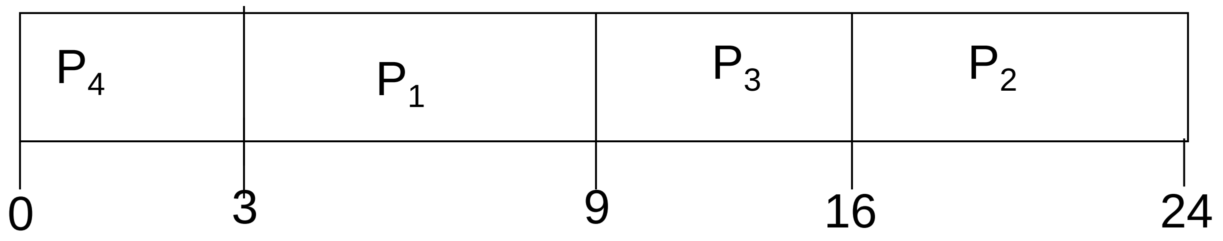
Example of SJF

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

Example of SJF

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

- SJF scheduling chart



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

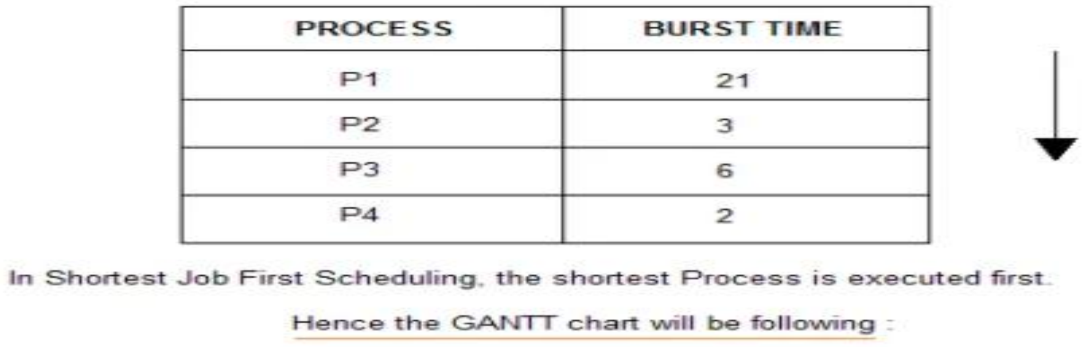
Shortest Job First(SJF) Scheduling

- Shortest Job First scheduling works on the process with the shortest burst time or duration first.

- This is the best approach to minimize waiting time.
- This is used in Batch Systems.
- It is of two types:
  - Non Pre-emptive
  - Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

Non Pre-emptive Shortest Job First

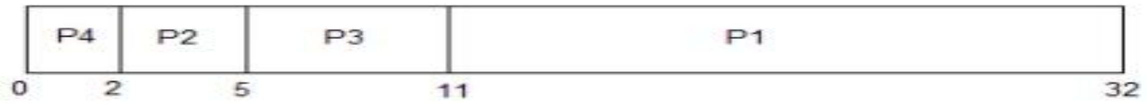
- Consider the below processes available in the ready queue for execution, with arrival time as 0 for all and given burst times.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

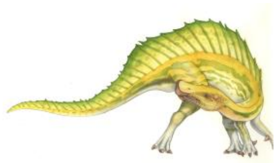
Problem with Non Pre-emptive SJF

- If the arrival time for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.

- This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

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



Priority CPU Scheduling

In this tutorial we will understand the priority scheduling algorithm, how it works and its advantages and disadvantages.

In the Shortest Job First scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.

In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on memory requirements, time limits ,number of open files, ratio of I/O burst to CPU burst etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, makrte factor etc.

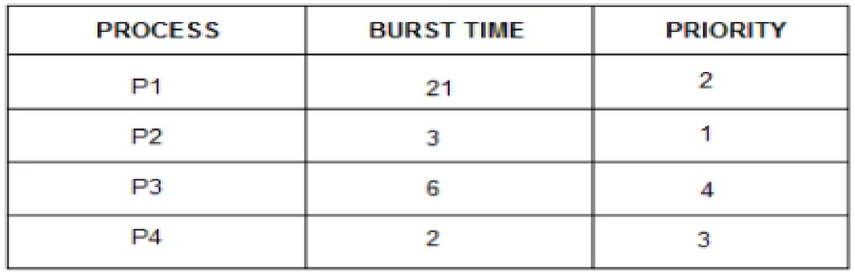
Types of Priority Scheduling Algorithm

- Priority scheduling can be of two types:

1. **Preemptive Priority Scheduling**: If the **new process arrived at the ready queue has a higher priority** than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.
2. **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a **new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue,** which means after the execution of the current process it will be processed.

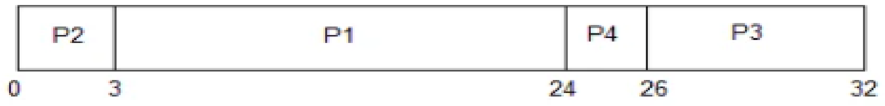
Example of Priority Scheduling Algorithm

Consider the below table fo processes with their respective CPU burst times and the priorities.



PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

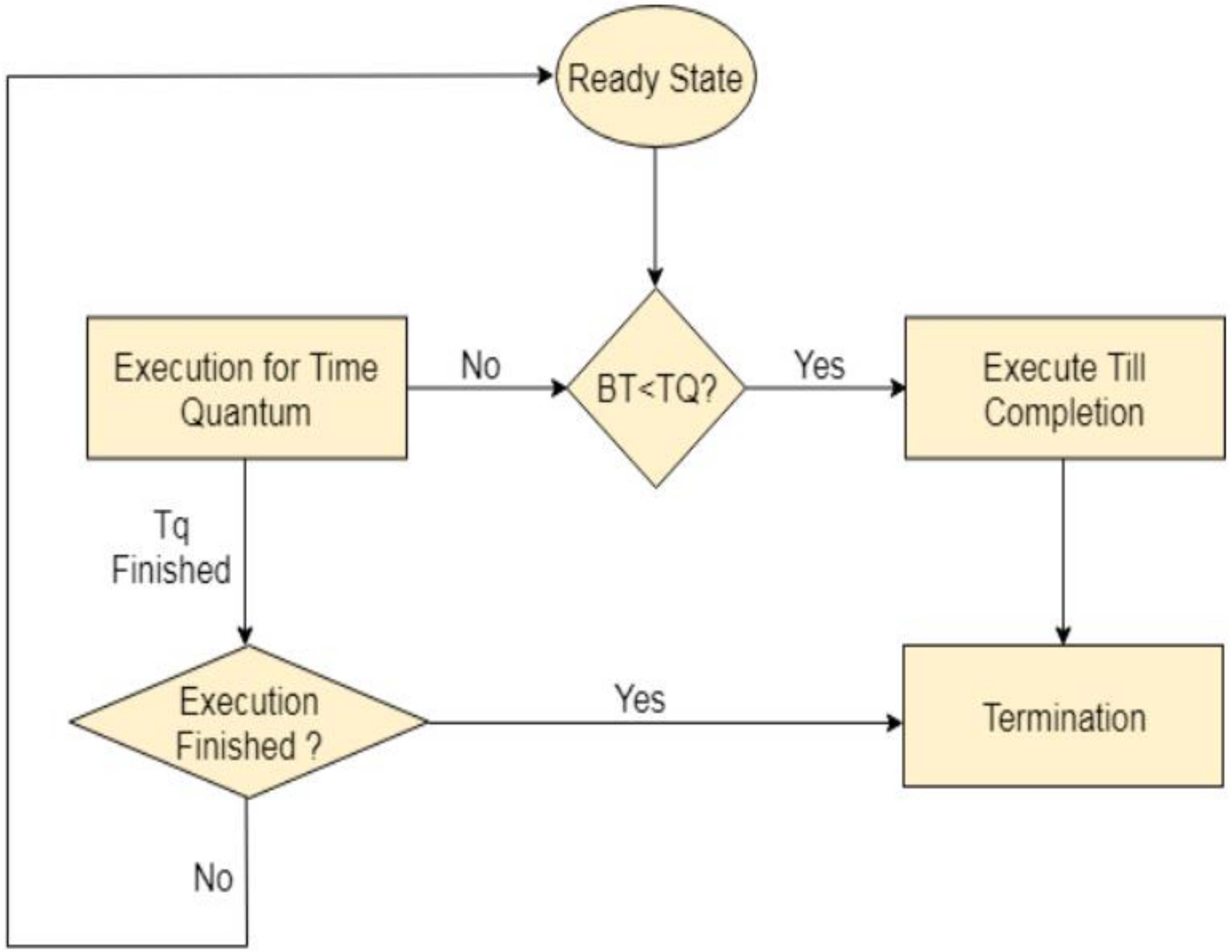
The GANTT chart for following processes based on Priority scheduling will be,



The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

Round Robin Scheduling

- Round Robin(RR) scheduling algorithm is mainly **designed for time-sharing systems**. This algorithm is similar to FCFS scheduling, but in Round Robin(RR) scheduling, preemption is added which enables the system to switch between processes.
- A fixed time is allotted to each process, called a **quantum**, for execution.
- Once a process is executed for the given time period that process is preempted and another process executes for the given time period.
- **Context switching is used to save states of preempted** processes.
- This algorithm is simple and easy to implement and the most important is thing is this **algorithm is starvation-free** as all processes get a fair share of CPU.
- It is important to note here that the length of time quantum is generally from 10 to 100 milliseconds in length.

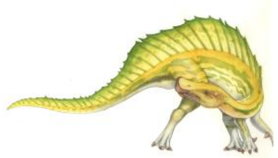

Ready State
Execution for Time Quantum
No
BT<TQ?
Yes
Execute Till Completion
Tq Finished
Execution Finished ?
Yes
Termination
No

Some important characteristics of the Round Robin(RR) Algorithm are as follows:

1. Round Robin Scheduling algorithm resides under the category of Preemptive Algorithms.
2. This algorithm is one of the oldest, easiest, and fairest algorithm.
3. This Algorithm is a real-time algorithm because it responds to the event within a specific time limit.
4. In this algorithm, the time slice should be the minimum that is assigned to a specific task that needs to be processed. Though it may vary for different operating systems.
5. This is a hybrid model and is clock-driven in nature.
6. This is a widely used scheduling method in the traditional operating system.

Advantages of Round Robin Scheduling Algorithm

- Some advantages of the Round Robin scheduling algorithm are as follows:
- While performing this scheduling algorithm, a **particular time quantum is allocated** to different jobs.
- In terms of average response time, this **algorithm gives the best performance.**
- With the help of this algorithm, **all the jobs get a fair allocation** of CPU.
- In this algorithm, there are **no issues of starvation or convoy effect**.
- This algorithm deals with all processes without any priority.
- This algorithm is **cyclic in nature.**
- In this, the newly created process is added to the end of the ready queue.
- Also, in this, **a round-robin scheduler generally employs time-sharing** which means providing each job a time slot or quantum.
- In this scheduling algorithm, each process gets a chance to reschedule after a particular quantum time.



Disadvantages of Round Robin Scheduling Algorithm

- Some disadvantages of the Round Robin scheduling algorithm are as follows:

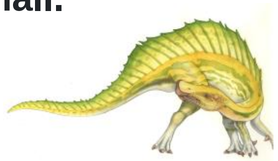
- This algorithm spends **more time on context switches**.
- For **small quantum**, it is time-consuming scheduling.
- This algorithm **offers a larger waiting time and response time**.
- In this, there is **low throughput.**
- If time quantum is less for scheduling then its Gantt chart seems to be too big.

Some Points to Remember

- **1.Decreasing value of Time quantum**
- With the decreasing value of time quantum
  - The number of context switches increases.
  - The Response Time decreases
  - Chances of starvation decrease in this case.
- For the **smaller value of time quantum,** it becomes better in terms of **response time.**

- **2.Increasing value of Time quantum**
- With the increasing value of time quantum
  - The number of context switch decreases
  - The Response Time increases
  - Chances of starvation increases in this case.
- For the higher value of time quantum, it becomes better in terms of the **number of the context switches.**

- 3. If the value of **time quantum is increasing** then Round Robin Scheduling tends to **become FCFS Scheduling.**
- 4.In this case, when the value of time quantum **tends to infinity** then the Round Robin Scheduling **becomes FCFS Scheduling.**
- 5. Thus the performance of Round Robin scheduling mainly depends on the **value of the time quantum.**
- 6.And the value of the **time quantum** should be such that it is neither **too big nor too small.**



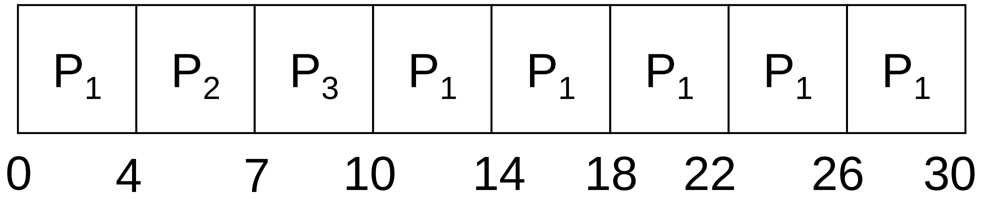
Round Robin (RR)

- 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

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

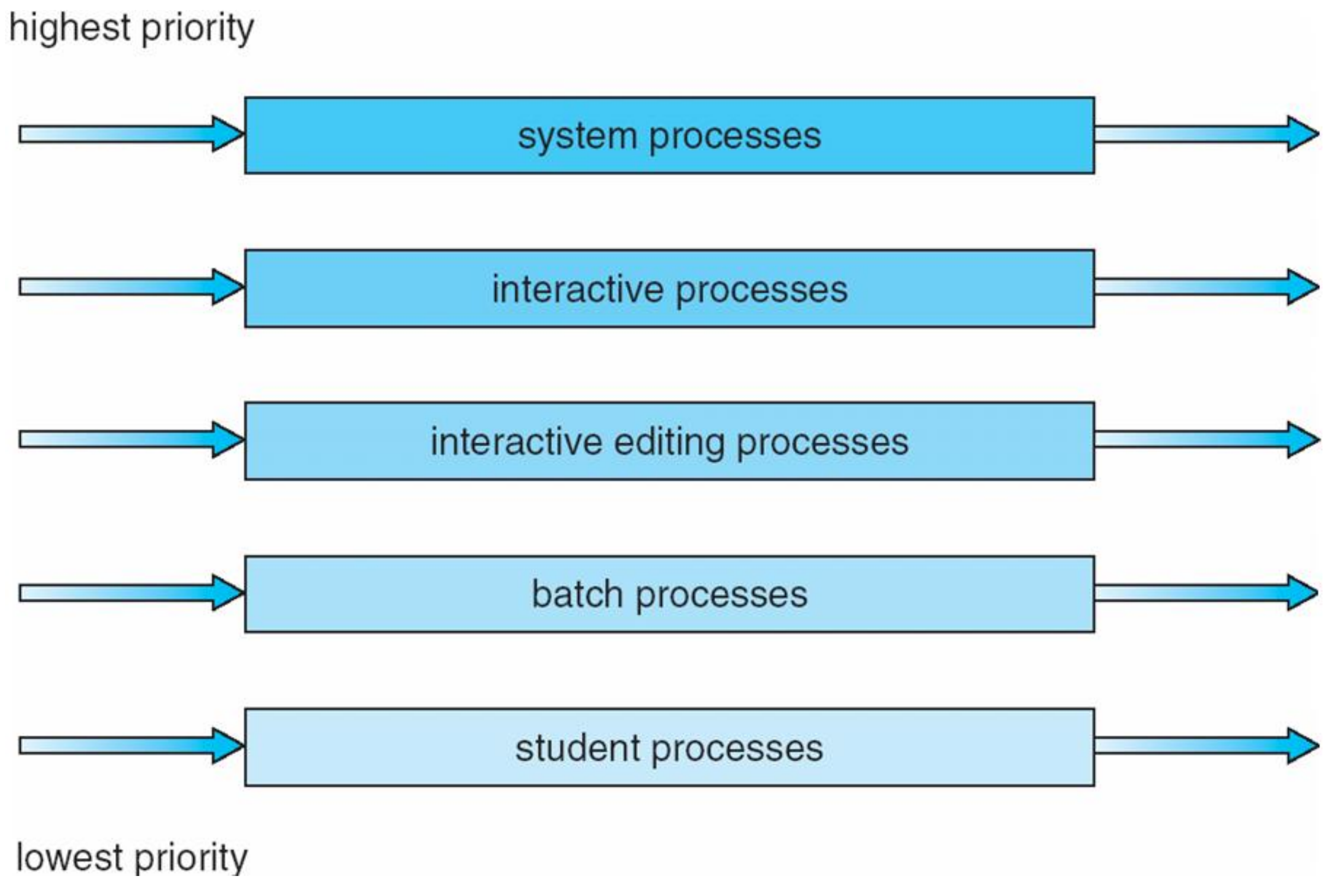


- Typically, higher average turnaround than SJF, but better *response*

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.
  - 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



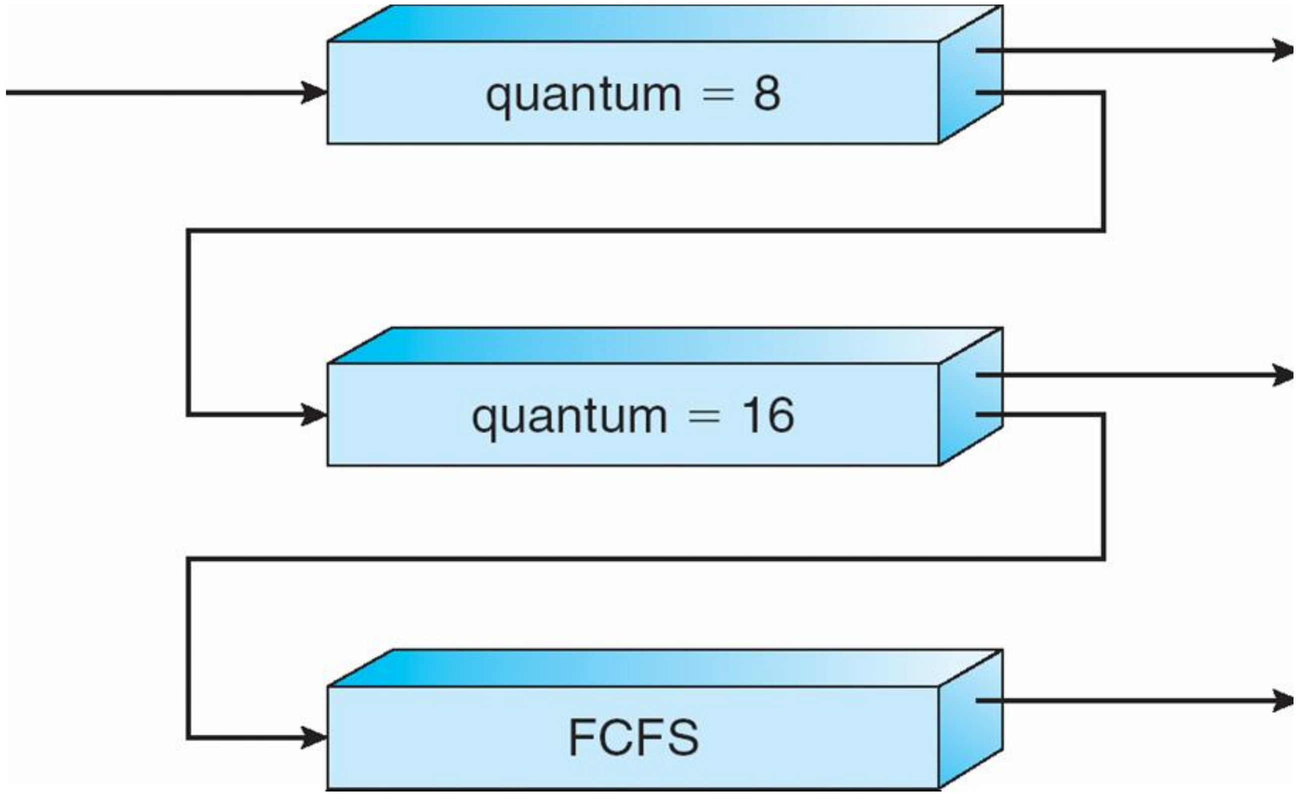
Multilevel Feedback Queue

- 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


quantum = 8
quantum = 16
FCFS