# MODULE-3: SCHEDULING

## PROCESS SCHEDULING

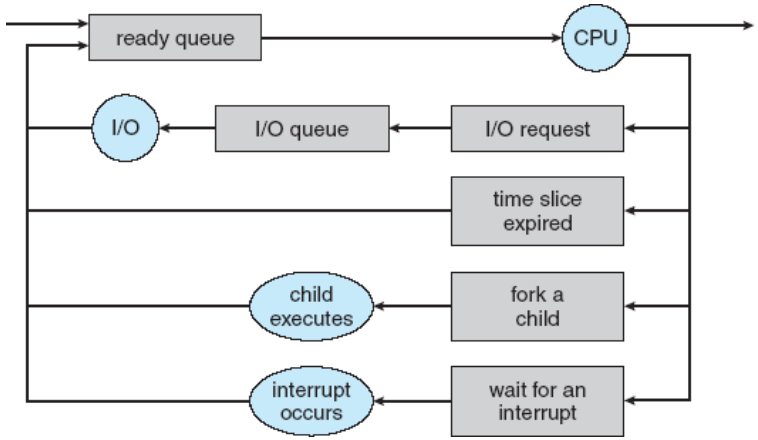
- The objective of multiprogramming is to maximize CPU utilization.
  - Running the process at all times.
- The objective of timesharing (multitasking) is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, **the process scheduler** selects an available process (possibly from a set of several available processes) for program execution on the CPU.
- For single processor system, there will never be more than one running process.
  - If there are more processes, then rest of the processes will have to wait until the CPU is free and can be rescheduled.

### Scheduling Queues

- **Job queue –** As process is created they are put into a job queue, which consists of all processes in the system.

- **Ready queue** – Processes that are residing in main memory are in ready and waiting to execute are kept in the ready queue.
- **Device queues –** Processes waiting for an I/O device. Each device has its own device queue.

## Queuing-diagram representation of process scheduling




- A common representation of process scheduling is a **queuing diagram.**
- Each rectangular box represents a queue.
- Two types of queues are present: ready queue and a set of device queues.
- The circles represent the resources that serve the queues.
- The arrows indicate the flow of processes in the system.
- A ready process is initially put in the ready queue. It waits there until it is selected for execution, or

**dispatched**.

- Once the process is allocated the CPU and is executing, one of several events could occur:
  - The process could issue an I/O request and then be placed in an I/O queue.
  - The process could create a new child process and wait for the child's termination.
  - The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

- In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue.
- A process continues this cycle until it terminates, at whichtime it is removed from all queues and has its PCB and resources deallocated.

**Schedulers**

- A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for

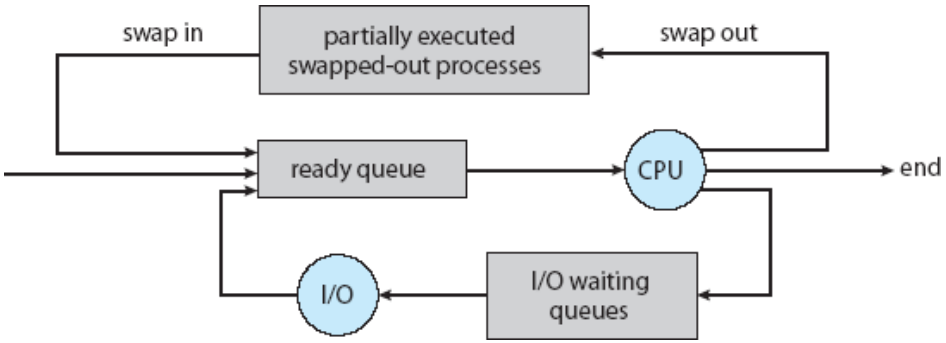
scheduling purposes, processes from these queues in some fashion.

- The selection process is carried out by the appropriate **scheduler**.
- **Long-term scheduler (or job scheduler)**
    - Selects which processes should be brought into the ready queue from the pool.
    - Long-term scheduler is invoked infrequently (seconds, minutes) - (may be slow).
    - The long-term scheduler controls the degree of multiprogramming (the number of processes in memory).
    - Long-term scheduler makes a careful selection. In general, most processes can be described as either I/O bound or CPU bound.
        - An **I/O-bound process** is one that spends more of its time doing I/O than it spends doing computations.
        - A **CPU-bound process**, in contrast, generates I/O requests infrequently, using more of its time

doing computations.

- It is important that the long-term scheduler select a good ***process mix*** of I/O-bound and CPU-bound processes.

- **Short-term scheduler (or CPU scheduler)**
  - Selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) - (must be fast)
- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**




**Medium-term scheduling to the queuing diagram**

## Context Switch

- Interrupts cause the OS to change a CPU from its current task and to run kernel routine.
- Such operation happens frequently on general purpose system.
- When an interrupt occurs, the system needs to save the current **CONTEXT** of the process currently running on the CPU so, that it can restore that **CONTEXT** when it resumes. Means essentially suspending the process and the resuming it.
- The CONTEXT is represented in the PCB of the process.
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.

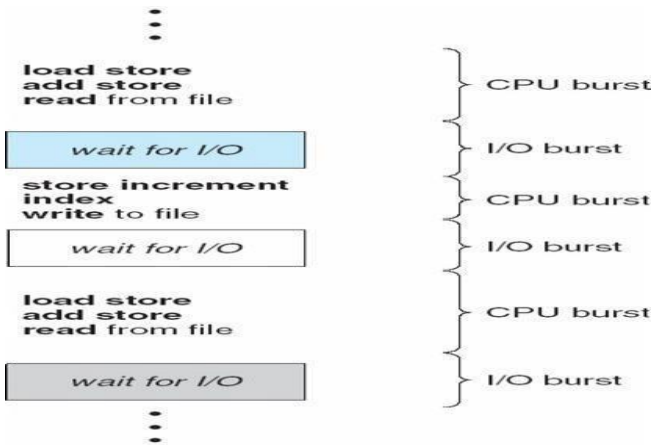
## CPU SCHEDULING

- CPU scheduling is the basis for multi-programmed operating systems.
- By switching the CPU among processes, the OS can make the computer work more productive.
- The main objective of multiprogramming is to have some process running at all times in order to maximize CPU utilization.

- Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. Every time one process has to wait, another process can take over use of the CPU.

## CPU–I/O Burst Cycle

- Process execution consists of a *cycle* of CPU execution and I/O wait i.e. CPU–I/O Burst Cycle –**CPU burst d**istribution followed by I/O burst distribution.
- The alternating sequence of CPU and I/O Bursts is as shown




below.

## CPU Scheduler

- 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 a single process from the memory among various processes 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 **non-preemptive.**
- All other scheduling is **preemptive.**

**Non-Preemptive**: Non-preemptive algorithms are designed so that once a process enters the running state, it is not removed from the processor until it has completed its service time (or it explicitly yields the processor).

**Preemptive**: Preemptive algorithms are driven by the notion of prioritized computation. The process with the highest priority should always be the one currently using the processor. If a process is currently using the processor and a new process with a higher priority enters the ready list, the process on the processor should be removed and returned to the ready list.

## Dispatcher

Dispatcher is a module that gives control of the CPU to the process selected by the short- term scheduler. This function involves the following:

- Switching context.
- Switching to user mode.
- Jumping to the proper location in the user program to restart that program.

  **Dispatch latency** – The time taken for the dispatcher to stop one process and start another running.

## Scheduling Criteria

Many criteria are there for comparing CPU scheduling algorithm. Some criteria required for determining the best algorithm are given below.

- **CPU utilization** – keep the CPU as busy as possible. The range is about 40% for lightly loaded system and about 90% for heavily loaded system.
- **Throughput** – The number of processes that complete their execution per time unit.
- **Turnaround time** – The interval from the time of submission of a process to the time of completion is the Turnaround time.
- **Waiting time** – amount of time a process has been waiting in the ready queue (or) the sum of the periods spent 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 the output (for time-sharing environment) is the response time.

## Best Algorithm consider following:

- Max CPU utilization

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

## Timings in scheduling

**Arrival Time (AT) – Is the time at which the process entered the ready queue.**

- Burst Time (BT) – Is the time required for the processor to complete the process.
- Completion Time (CT) – Is the time when the processor has finished the execution of the process.
- Turn Around Time (TAT) – Is the total amount of time the process was inside the queue.
    - **TAT = CT – AT or TAT = WT + BT**
- Waiting Time (WT) – Is the time the process was waited in the queue but did not get the processor.
    - **Waiting Time = TAT – BT**

## Scheduling Algorithms

1. First-come, First-Served Scheduling
2. Shortest-Job-First Scheduling – SJF & SJF-SRTF
3. Priority Scheduling
4. Round-Robin Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback-Queue Scheduling

### 1. FCFS - First-Come, First-Served

- ✓ **First come first serve** (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time.
- ✓ The job which comes first in the ready queue will get the CPU first.
- ✓ The lesser the arrival time of the job, the sooner will the job get the CPU.
- ✓ FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Example-1: Let us assume the processes are arriving as shown below. Find Average TAT, Average Waiting Time. Show the Timeline representation of each process.

| **Process** | **AT (ms)** | **BT (ms)** |
|---|---|---|

| | | |
|---|---|---|
| $P_1$ | 0 | 4 |
| $P_2$ | 1 | 3 |
| $P_3$ | 2 | 1 |
| $P_4$ | 3 | 2 |
| $P_5$ | 4 | 5 |

**Criteria: Arrival Time**

**Mode: Non-preemptive**

**Gantt Chart:**

| 0 | | 4 | | 7 | | 8 | | 10 | 15 |
|---|---|---|---|---|---|---|---|---|---|
| $P_1$ | | $P_2$ | | $P_3$ | | $P_4$ | | $P_5$ | |

**Calculation:**

| Process | AT (ms) | BT (ms) | CT (ms) | TAT (ms) | WT (ms) |
|---|---|---|---|---|---|
| $P_1$ | 0 | 4 | 4 | 4-0=4 | 4-4=0 |
| $P_2$ | 1 | 3 | 7 | 7-1=6 | 6-3=3 |

| | | | | | |
|---|---|---|---|---|---|
| **$P_3$** | **2** | **1** | **8** | **8-2=6** | **6-1=5** |
| **$P_4$** | **3** | **2** | **10** | **10-3=7** | **7-2=5** |
| **$P_5$** | **4** | **5** | **15** | **15-4=11** | **11-5=6** |

**Total TAT =** 4 + 6 + 6 + 7 +11 = 34 ms

**Avg(TAT) =** 34/5 = 6.8 ms

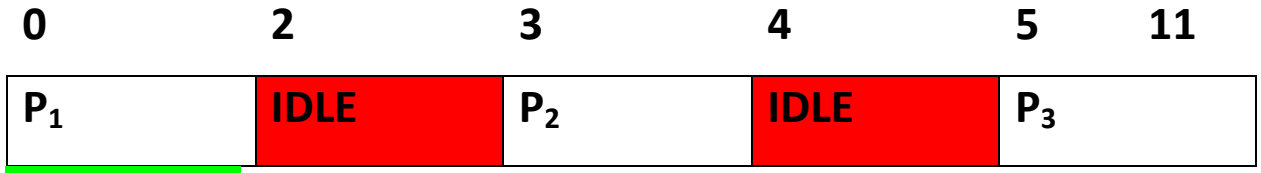
**Total WT =** 0+ 3 + 5 + 5 + 6 = 19 ms

**Avg(WT) =** 19/5 = 3.8 ms

Example-2: Let us assume the processes are arriving as shown below. Find Average TAT, Average Waiting Time. Show the Timeline representation of each process.

| **Process** | **AT** | **BT** |
|---|---|---|
| $P_1$ | 0 | 2 |
| $P_2$ | 3 | 1 |
| $P_3$ | 5 | 6 |

**Gantt chart:**




**Calculation:**

| Process | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| $P_1$ | 0 | 2 | 2 | 2-0=2 | 2-2=0 |
| $P_2$ | 3 | 1 | 4 | 4-3=1 | 1-1=0 |
| $P_3$ | 5 | 6 | 11 | 11-5=6 | 6-6=0 |

**Total TAT =** 2 + 1 + 6 = 9 ms **Avg(TAT) =** 9/3 = 0 ms

**Total WT =** 0+ 0 + 0 =0 ms **Avg(WT) =** 0 ms

## Convoy Effect in FCFS

- ✓ FCFS may suffer from the **convoy effect** if the burst time of the first job is the highest among all.
- ✓ As in the real life, if a convoy is passing through the road then the other persons may get blocked until it passes completely. This can be simulated in the Operating System also.

- ✓ If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time. This is called **convoy effect** or **starvation**.
- ✓ **Starvation** is a condition where a process does not get the resources it needs for a long time because the resources are being allocated to other processes.

**Practice: All the processes arrived at 0 ms. Find Average TAT,** and **Average WT.**

| **Process** | **BT** |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

**Advantages of FCFS**

- Simple
- Easy

**Disadvantages of FCFS**

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, **the problem of starvation may occur**.
3. Although it is easy to implement, but it is **poor in performance** since the **average waiting time is higher** as compare to other scheduling algorithms.

## FCFS with Overhead

- In the above Examples, we are assuming that all the processes are the CPU bound processes only. We were also neglecting the context switching time.
- However if the time taken by the scheduler in context switching is considered then the average waiting time of the system will be **increased** which also affects the **efficiency** of the system.
- Context Switching is always an overhead.

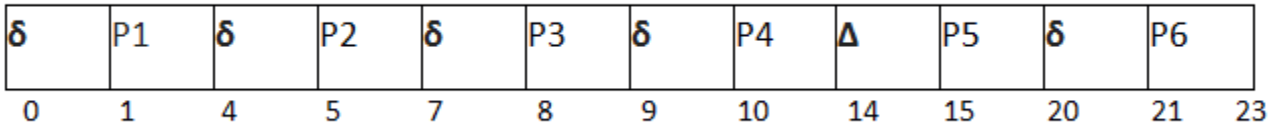
The following example shows how the efficiency will be affected if the context switching time is considered in the system.

We are considering five processes P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below.

| Process ID | Arrival Time | Burst Time |
|---|---|---|
| 1 | 0 | 3 |
| 2 | 1 | 2 |
| 3 | 2 | 1 |
| 4 | 3 | 4 |
| 5 | 4 | 5 |
| 6 | 5 | 2 |

If the context switching time of the system is 1 unit then the Gantt chart of the system will be prepared as follows.

Given **δ=1 unit;**




The system will take extra 1 unit of time (overhead) after the execution of every process to schedule the next process.

Inefficiency= (6/23) X 100 %  = 26.08%

Efficiency = (1-6/23) X 100 %  = 21.73%

## 2. Shortest Job First

- ✓ SJF scheduling algorithm, schedules the processes according to their burst time.
- ✓ In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

**Advantages of SJF**

1. Maximum throughput

2. Minimum average waiting and turnaround time

**Disadvantages of SJF**

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

**Example-1:** Let us assume the processes are arriving as shown below. Find Average TAT, Average Waiting Time. Show the Timeline representation of each process.

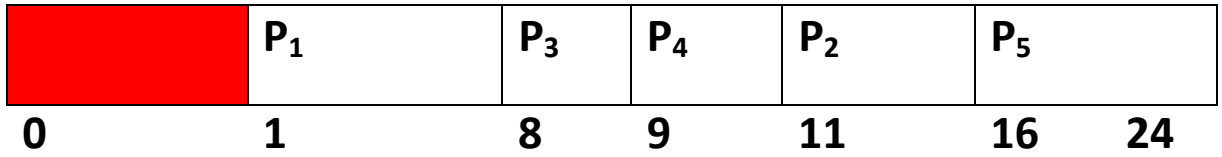
| Process | AT | BT |
|---|---|---|
| $P_1$ | 1 | 7 |
| $P_2$ | 2 | 5 |
| $P_3$ | 3 | 1 |
| $P_4$ | 4 | 2 |
| $P_5$ | 5 | 8 |

**Solution:**

**Criteria – Burst Time**

**Mode – Non-preemptive**

**Gantt chart:**




**Calculation:**

| Process | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| $P_1$ | 1 | 7 | 8 | 8-1=7 | 7-7=0 |
| $P_2$ | 2 | 5 | 16 | 16-2=14 | 14-5=9 |
| $P_3$ | 3 | 1 | 9 | 9-3=6 | 6-1=5 |
| $P_4$ | 4 | 2 | 11 | 11-4=7 | 7-2=5 |
| $P_5$ | 5 | 8 | 24 | 24-5=19 | 19-8=11 |

**Total TAT =** 7 + 14 + 6 + 7 +19 = 53 ms

**Avg(TAT) = 53**/5 = 10.5 ms

**Total WT =** 0+ 9 + 5 + 5 + 11 = 30 ms

**Avg(WT) =** 30/5 = 6 ms

**2. Shortest Job First – Smallest Remaining Time First (SJF-SRTF)**

**Criteria – Burst Time**

**Mode – Preemptive [**Can pause the currently running process and can execute another process]

Example-1: Let us assume the processes are arriving as shown below. Find Average TAT, Average Waiting Time. Show the Timeline representation of each process.

| Process | AT | BT |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 1 | 5 |
| $P_3$ | 2 | 3 |
| $P_4$ | 3 | 1 |
| $P_5$ | 4 | 2 |
| $P_6$ | 5 | 1 |

**Gantt Chart:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 13 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|
| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_3$ | $P_3$ | $P_6$ | $P_5$ | $P_2$ | $P_1$ | |

**Calculation:**

| Process | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| $P_1$ | 0 | 7 | 19 | 19-0=19 | 19-7=12 |
| $P_2$ | 1 | 5 | 13 | 13-1=12 | 12-5=7 |

| | | | | | |
|---|---|---|---|---|---|
| **$P_3$** | **2** | **3** | **6** | **6-2=4** | **4-3=1** |
| **$P_4$** | **3** | **1** | **4** | **4-3=1** | **1-1=0** |
| **$P_5$** | **4** | **2** | **9** | **9-4=5** | **5-2=3** |
| **$P_6$** | **5** | **1** | **7** | **7-5=2** | **2-1=1** |

**Total TAT =** 19 + 12 + 4 + 1 + 5 + 2 = 43 ms

**Avg(TAT) =** 43/6 = 7.166 ms

**Total WT =** 12+ 7 + 1 + 0 + 3 +1 = 24 ms

**Avg(WT) =** 24/6 = 4 ms

**Practice:**

Let us assume the processes are arriving as shown below. Find Average TAT, Average

Waiting Time. Show the Gannt chart of each process.

| Process | AT (ms) | BT (ms) |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 2 | 4 |
| $P_3$ | 4 | 1 |
| $P_4$ | 5 | 4 |

## Priority Scheduling

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Priorities can be defined either internally or externally.
- Internally defined priorities use some measurable quantities to compute the priority of a process.
    - Time limits
    - Memory requirements
    - Number of open files, and
    - Ratio of average I/O burst to average CPU burst

- External priorities are set by criteria outside the OS such as:
    - Importance of the process
    - Type and amount of fund paid for computer use
    - Department sponsoring work, and
    - Other political factors.
- Equal priority processes are scheduled in FCFS order.
- Priorities are generally indicated by some fixed range of numbers.
- There is no general agreement on whether 0 is the highest priority or lowest priority.
- Some systems use low numbers to represent low priority; others use low numbers for higher priority.
- Priority scheduling can be either preemptive or non- preemptive.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
- In Preemptive priority scheduling CPU will preempt the process if the priority of a newly arrived process is higher than the priority of the currently running process.

- In Non-preemptive, the CPU will simply put the new process at the head of the ready queue.
- **Indefinite blocking or Starvation**: A process is ready to run but waiting for the CPU can be considered blocked. Means a priority scheduling can leave some low priority process waiting indefinitely.
- **Solution**: **Aging**→Is a technique in which the priority of the processes are slowly increasing.
- NOTE: We assume that low numbers represent high priority.
- Consider the following set of processes, assumed to have arrived at time 0 in the order P1,P2,….,P5 with the length of the CPU burst in ms.

**Example-1:** Assume the processes are arriving at 0 ms with the following burst values. Calculate average TAT, and average WT.

- **Mode: Non-preemptive**

| Process | BT | Priority |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 4 |
| $P_4$ | 1 | 5 |
| $P_5$ | 5 | 2 |

**Gantt chart:**

| $P_2$ | $P_5$ | $P_1$ | $P_3$ | $P_4$ | |
|---|---|---|---|---|---|
| 0 | 1 | 6 | 16 | 18 | 19 |

**Calculation:**

| Process | BT | Priority | CT | TAT | WT |
|---|---|---|---|---|---|
| $P_1$ | 10 | 3 | 16 | 16-0=16 | 16-10=6 |
| $P_2$ | 1 | 1 | 1 | 1-0=1 | 1-1=0 |
| $P_3$ | 2 | 4 | 18 | 18-0=18 | 18-2=16 |
| $P_4$ | 1 | 5 | 19 | 19-0=19 | 19-1=18 |
| $P_5$ | 5 | 2 | 6 | 6-0=6 | 6-5=1 |

**Total TAT =** 16 + 1 + 18 + 19 + 6 = 60 ms

**Avg(TAT) =** 60/5 = 12 ms

**Total WT =** 6+ 0 + 16 + 18 +1 = 41 ms

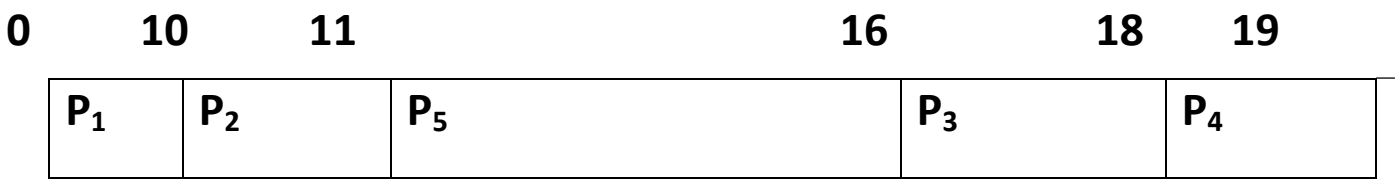
**Avg(WT) =** 41/5 = 8.2 ms

**Example-2:** Assume the processes are arriving at different time to the ready queue with a burst values. Calculate average TAT, and average WT.

**Mode: Non-Preemptive**

| Process | AT | BT | Priority |
|---|---|---|---|
| $P_1$ | 0 | 10 | 3 |
| $P_2$ | 1 | 1 | 1 |
| $P_3$ | 2 | 2 | 4 |
| $P_4$ | 3 | 1 | 5 |
| $P_5$ | 4 | 5 | 2 |

**Gantt chart:**




**Calculation:**

| Process | AT | BT | Priority | CT | TAT | WT |
|---|---|---|---|---|---|---|
| $P_1$ | 0 | 10 | 3 | 10 | 10-0=10 | 10-10=0 |
| $P_2$ | 1 | 1 | 1 | 11 | 11-1=10 | 10-1=9 |
| $P_3$ | 2 | 2 | 4 | 18 | 18-2=16 | 16-2=14 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **$P_4$** | **3** | **1** | **5** | **19** | **19-3=16** | **16-1=15** |
| **$P_5$** | **4** | **5** | **2** | **16** | **16-4=12** | **12-5=7** |

**Total TAT =** 10 + 10 + 16 + 16 + 12 = 64 ms

**Avg(TAT) =** 74/5 = 12.8 ms

**Total WT =** 0 + 9 + 14 + 15 +7 = 45 ms

**Avg(WT) =** 45/5 = 9 ms

**Practice:**

Assume the processes are arriving at different time to the ready queue with a burst values. Calculate average TAT, and average WT. **Mode: Non-Preemptive**

| Process | AT | BT | Priority |
|---|---|---|---|
| $P_1$ | 0 | 4 | 8 |
| $P_2$ | 1 | 2 | 6 |
| $P_3$ | 2 | 3 | 3 |
| $P_4$ | 3 | 1 | 1 |
| $P_5$ | 4 | 5 | 4 |

## Priority Scheduling - Preemptive

Assume the processes are arriving at different time to the ready queue with a burst values. Calculate average TAT, and average WT.

**Mode: Preemptive Criteria : Priority**

| Process | AT | BT | Priority |
|---|---|---|---|
| $P_1$ | 0 | 10 | 2 |
| $P_2$ | 2 | 5 | 1 |
| $P_3$ | 3 | 2 | 0 |
| $P_4$ | 5 | 20 | 3 |

**Calculation:**

| P1 | **P1** | **P2** | **P3** | **P2** | **P1** | **P4** |
|---|---|---|---|---|---|---|

**0 1 2 3 5 9 17 37**

| Process | AT | BT | Priority | CT | TAT | WT |
|---|---|---|---|---|---|---|
| $P_1$ | 0 | 10 | 2 | **17** | **17** | **7** |
| $P_2$ | 2 | 5 | 1 | **9** | **7** | **2** |
| $P_3$ | 3 | 2 | 0 | **5** | **2** | **0** |
| $P_4$ | 5 | 20 | 3 | **37** | **32** | **12** |

**Avg. TAT = 58/4 = 14.5 ms Avg. WT = 21/4 =**

**5.25 ms**

Assume the processes are arriving at different time to the ready queue with a burst values. Calculate average TAT, and average WT. **Mode: Preemptive**

| Process | AT | BT | Priority |
|---|---|---|---|
| $P_1$ | 0 | 10 | 3 |
| $P_2$ | 1 | 1 | 1 |
| $P_3$ | 2 | 2 | 4 |
| $P_4$ | 3 | 1 | 5 |
| $P_5$ | 4 | 5 | 2 |

**Gantt chart:**

| $P_1$ | $P_2$ | $P_1$ | $P_1$ | $P_5$ | $P_1$ | $P_3$ | $P_4$ |
|---|---|---|---|---|---|---|---|

**0 1 2 3 4 9 16 18 19**

**Calculation:**

| Process | AT | BT | Priority | CT | TAT | WT |
|---|---|---|---|---|---|---|
| **$P_1$** | 0 | 10 | 3 | **16** | **16** | **6** |
| **$P_2$** | 1 | 1 | 1 | **2** | **1** | **0** |
| **$P_3$** | 2 | 2 | 4 | **18** | **16** | **14** |
| **$P_4$** | 3 | 1 | 5 | **19** | **16** | **15** |
| **$P_5$** | 4 | 5 | 2 | **9** | **5** | **0** |

**Total TAT =** 16 + 1 + 16 + 16 + 5 = 54 ms ; **Avg(TAT) =** 54/5 = 10.8 ms

**Total WT = 6** + 0 + 14 + 15 +0 = 35 ms ;**Avg(WT) =** 35/5 = 7 ms

## Round Robin (RR)

- Round robin scheduling is designed especially for time-sharing systems.
- It is similar to FCFS, but preemption is added to switch between processes.
- Each process gets a small unit of CPU time called a *timequantum or time slice.*
- A time quantum is generally from 10-100 milliseconds.
- A ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- To implement RR scheduling, the ready queue is kept as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum and dispatches the process.
- If the CPU burst time is less than the time quantum, the process itself will release the CPU voluntarily. Otherwise, if the CPU burst of the currently running process is longer than the time quantum a context switch will be executed and the process will be put at the tail 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**.

- The performance of RR completely depends on the size of the time quantum.
- If the time quantum is very large then FCFS policy is followed.
- If time quantum is small then this approach is called processor sharing.
- The time quantum must be large with respect to context switch, otherwise overhead is too high.

**Example-1**

Consider the following processes are arrived at time 0 ms, with the length of the CPU burst given in ms. Assume the time quantum TQ = 4ms.

| Process | BT |
|---|---|
| P1 | 24 |
| P2 | 3 |
| P3 | 3 |

**Gantt Chart:**

| P1 | P2 | P3 | P1 | P1 | P1 | P1 | P1 |
|---|---|---|---|---|---|---|---|

0 4 7 10 14 18 22 26 30

**Calculation:**

| Process | BT | CT | TAT | WT |
|---|---|---|---|---|
| P1 | 24 | 30 | 30 | 6 |
| P2 | 3 | 7 | 7 | 4 |
| P3 | 3 | 10 | 10 | 7 |

**Average TAT = 30 + 7 + 10/3 = 47/3 = 15.66 ms Average WT = 6**

**+ 4 + 7 = 17/3 = 5.66 ms Example-2:**

Consider the following processes are arrived with the length of the

CPU burst given in ms. Assume the time quantum TQ = 3ms.

| Process | AT | BT |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |

**Gantt Chart:**

| P1 | P2 | P1 | P3 | P2 | P1 | |
|---|---|---|---|---|---|---|
| 0 | 3 | 6 | 9 | 10 | 11 | 12 |

**Calculation:**

| Process | AT | B<br>T | CT | TAT | WT |
|---|---|---|---|---|---|
| P1 | 0 | 7 | 12 | 12 | 5 |
| P2 | 2 | 4 | 11 | 9 | 5 |
| P3 | 4 | 1 | 10 | 6 | 5 |

**Average TAT = 12 + 9 + 6/3 =27/3 = 9 ms Average WT = 5**

**+ 5 + 5 = 15/3 = 5 ms**

**Practice:** With TQ = 4 ms and TQ = 3ms

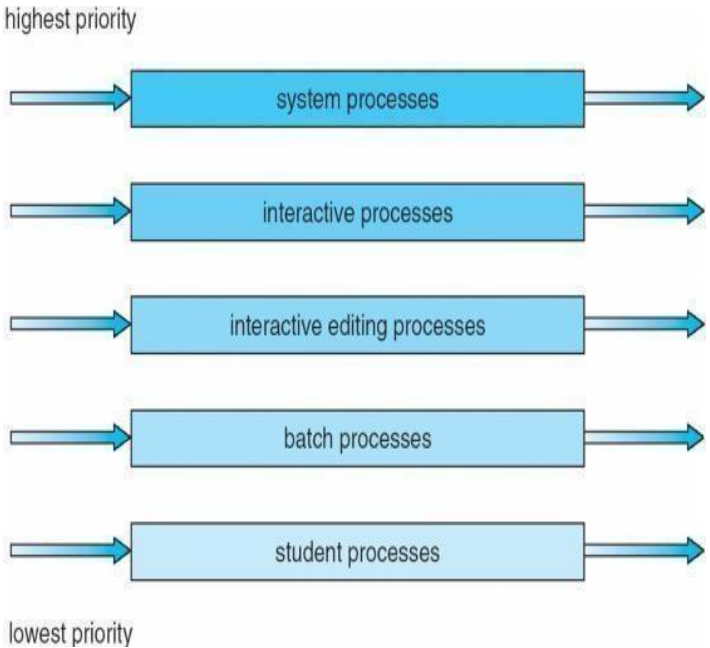
| Process | AT | BT |
|---|---|---|
| P1 | 0 | 4 |
| P2 | 1 | 5 |
| P3 | 2 | 2 |
| P4 | 3 | 1 |
| P5 | 4 | 6 |
| P6 | 6 | 3 |

## Multilevel Queue Scheduling

- This scheduling algorithm is created for situations in which processes are easily classified into different groups.
- There are two types of processes
    - Foreground or interactive processes.
    - Background or batch processes.
- This algorithm partitions the ready queue into several separate queues.

- The processes are permanently assigned to one queue based on properties like process size, process priority or process type.
- Each queue has its own algorithm for example the foreground queue might be scheduled by an RR algorithm and the background queue is scheduled by an FCFS algorithm.
- The foreground queue have absolute priority over background queue.
- Scheduling must be done between the queues

Fixed priority scheduling; (i.e., serve all from foreground then from background). Due to fixed priority scheduling



there are possibility of starvation.

**Multilevel Feedback Queue scheduling**

- In multilevel queue scheduling the processes are fixed with respect to each queue and they cannot navigate between the queues.
- Multilevel Feedback Queue scheduling however allows a process to move between queues.
- A process can move between the various queues; aging can be implemented this way.
- If a process uses too much of CPU time, it will be moved to a low priority queue. If a process waits too long in a lower priority queue it may be moved to a higher priority queue. This form 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 to a higher priority queue.

- method used to determine when to demote a process to a lower priority queue.
- The 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

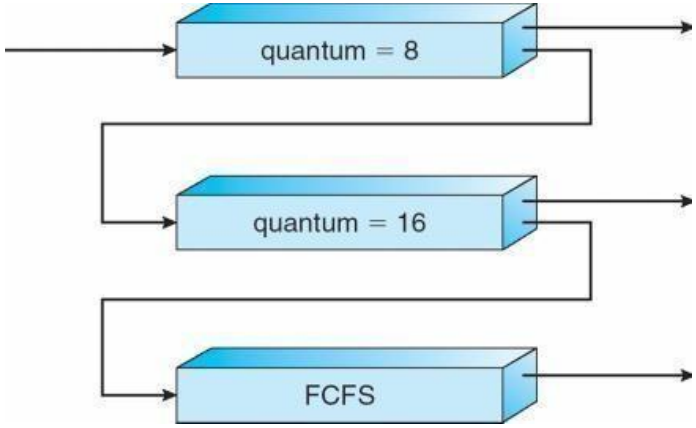
**Scheduling**

A new job enters queue *Q0* 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**




**Practice:**

1. Consider the following process, with the CPU burst tie given in ms.

| Process | BT | Priority |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

Processes are arrived in P1,P2, P3, P4, P5 order of all at time 0.

1) Draw Gantt charts to show execution using FCFS, SJF, non-preemptive priority ( small priority number implies higher priority) and RR (quantum = 1ms) scheduling.
2) Also calculate turnaround time for each scheduling algorithms.
3) What is the waiting time of each process for each one of the above scheduling algorithms?

2. For the following example calculate average TAT and average WT for the following algorithms.

a) FCFS
b) Preemptive SJF
c) Round Robin ( 1ms)

| Process | AT | BT |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |