**UNIT 2 (Continue)**

**Scheduling**

CPU scheduling is the process which allows one process to use the CPU while execution of another process is in hold due to unavailability of any resources like CPU cycle time, I/O etc. Its aim is to utilize the CPU time up to maximum level. It is the process of giving executing time to another process while one process is waiting or idle such that CPU time is not wasted or dose not sits idle.

In single processor system only one process can run at a time, other must wait until the CPU is free. The objective of the multiprogramming is to have some process running all the time to maximize the CPU utilization. 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. Process alternate between these two burst. Process execution begins with a CPU burst and followed by the I/O burst which is followed by another CPU burst then I/O burst and so on. The final CPU burst ends with a system request to terminate execution.

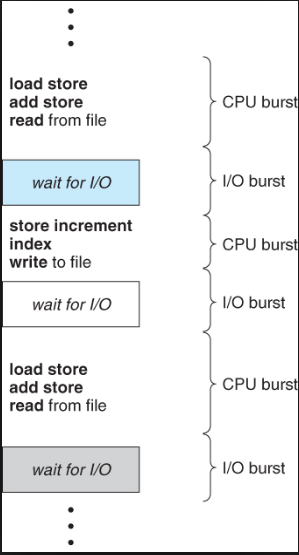


Figure: alternating sequence of CPU and I/O burst

**CPU scheduler:**

Whenever the CPU becomes idle, the operating system must selects one of the process from ready queue to execute such that CPU time is utilized. This selection process is carried out by the short-term scheduler also known as CPU scheduler.

This scheduler selects the process from the group of processes in memory that are ready to execute and allocates the CPU to that process. All the process in the ready queue are lined up waiting for a chance to run on the CPU. The record in the queue are generally process control block of the processes.

CPU scheduling decision may take place under the following circumstance:

1. When a process switches from the running state to the waiting state. For e.g. the result of an I/O request or an invocation of wait() for the termination of child.
2. When a process switches from running state to the ready state for e.g. when an interrupt occur.
3. When a process switches from the waiting state to ready state for e.g. at completion of I/O
4. When a process terminates.

Under the **non-preemptive or cooperative scheduling scheme,** 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 it to the waiting state. This scheduling is the only method that can be used on certain hardware platform because it does not require any hardware such as timer needed for preemptive scheduling. This scheduling is used by Microsoft Windows 3.x. It is rigid because although critical process enter the ready queue it will not disturb the running process.

Under the **preemptive scheduling scheme**, the resources (CPU time) are allocated to the process for limited amount of time and is then taken away and again placed back in the ready queue if that process still have burst time remaining. The process stays in the ready queue till it gets next chance to execute. Here the running process are paused in middle of execution if highest priority process request arrives.

Preemptive mechanism can suffer from the starvation problem because low priority process have to wait for a long time if high priority process frequently arrive. But it is flexible also as it allows critical process to execute first as they arrive in ready queue. It can also suffer from **race condition.** For e.g. window 95 introduce the preemptive scheduling, Mac OS also uses preemptive scheduling etc.

Difference between preemptive and non-preemptive scheduling:

|  |  |
| --- | --- |
| **Preemptive Scheduling** | **Non-Preemptive Scheduling** |
| CPU is allocated to process for limited time | CPU is allocated to process until it finish execution or terminate or switches to waiting state. |
| The executing process are interrupted in middle of execution if highest priority process will arrives | The process are not interrupted in the middle of the execution and other process have to wait until running process terminates |
| There is overhead of switching the process from ready to running state and maintaining the ready queue. | No overhead of switching the process from ready to running state |
| It suffers from starvation problem because the low priority process have to wait if high priority processes frequently arrives. | It may also suffer starvation if a process with longer burst time is running CPU. |
| It is flexible as it allows critical process to execute first | It is rigid |
| It is cost associated as it has to maintain the integrity of shared data | It is not cost associated |

**Dispatcher:**

Dispatcher is the module or component involved in CPU scheduling that gives the control of the CPU to the process selected by the short-term scheduler. This function involves switching the context, switching to user mode, jumping to the proper location in the user program to restart that program. As it is invoked during every process switch, it should be as fast as possible.

The time it takes for dispatcher to stop one process and start another process is known as **dispatch latency**.

**Scheduling Criteria:**

Many criteria have been suggested for comparing CPU scheduling algorithm. Which characteristics are used for comparison can make a difference in which algorithm is judged to be best. The criteria includes the following:

1. **CPU Utilization:**

It refers to keep the CPU as busy as possible. Conceptually CPU utilization can range from 0 to 100 percent. In real system it should ranges from 40 percent (lightly loaded system) to 90 percent (highly loaded system).

1. **Throughput:**

It is the measure of work or measure of number of process that are completed per time. If the CPU is busy executing processes then the work is being done. For long process it rate may be one process per hour and for short transaction it may be ten process per second.

1. **Turnaround Time:**

The interval from the time of submission of the process to the time of completion is turnaround time. It refers to how long it takes to execute a process. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing in the CPU and doing I/O.

Mathematically, Turnaround time = Complete Time (CT) – Burst Time (BT)

1. **Waiting Time:**

It is the sum of the period spent waiting in the ready queue. CPU scheduling algorithm only affects the amount of time that a process spends waiting in the ready queue.

Mathematically, Waiting Time = Total turnaround time – Burst time

1. **Response Time:**

It is measure of the time from the submission of a request until the first response is produced. It is the time process takes to start responding not the time it takes to output the response.

**Scheduling Algorithm:**

CPU scheduling deals with the problem of deciding which of the process in the ready queue is to be allocated to the CPU. Some of the CPU scheduling algorithm are:

1. **First Come First Serve Scheduling (FCFS): (Non-preemptive)**

In this scheme the process that request the CPU first is allocated the CPU first. The implementation of this scheme is managed with a FIFO queue. When the process enters the ready queue its PCB is linked into the tail of the queue and when the CPU is free it is allocated to the process at the head of the queue. The running process is then removed from the queue.

Its disadvantage is that its average waiting time is often quite long. For e.g.:

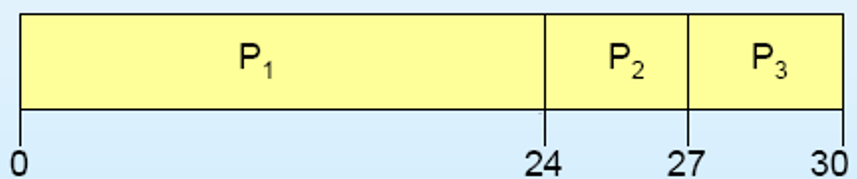
Process Burst Time

P1 24

P2 3

P3 3

Suppose that process arrives in the order P1, P2 and P3 and are served in FCFS order.



Waiting Time for: P1 = 0, P2 = 24 and P3 = 27

Average Waiting Time = (0+24+27)/3 = 17

This scheme is non-preemptive strategy i.e. once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU. So, this scheme is troublesome for time sharing system where each user gets the share of the CPU at regular interval. There is a **convey effect** as all the other process with small burst have to wait for the one big process to get off the CPU.

1. **Shortest Job First (SJF): (Non-preemptive and Preemptive)**

**Non-preemptive Scheme:**

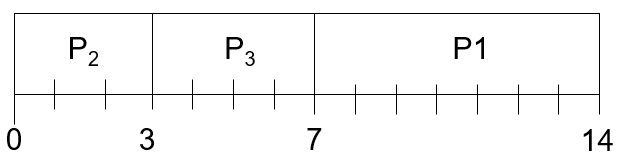
This algorithm associates with each process’s next CPU burst. When the CPU is available it is assigned to the process that has the smallest next CPU burst. If the next two process’s burst are same then FCFS mechanism is used to break the tie. Scheduling depends on the length of the next CPU burst of a process rather than its total length. Example:

Process Burst Time

P1 7

P2 3

P3 4



Waiting time for: P1 = 7, P2 = 0, P3 = 3

Average waiting time = (7 + 0 + 3)/3 = 3.33

**Preemptive Scheme:**

The choices arises when a new process arrives at the ready queue while previous process is still executing and the CPU burst of new arrived process is less than that of executing. This scheme will preempt the currently executing process. This scheme is sometimes called **shortest-remaining-time first scheduling.**

The SJF scheduling is provably optimal as it gives the minimum waiting time. Although it is optimal, the real difficulty with this scheme is knowing the length of the next CPU request.

Example of shortest job first (Non preemptive and preemptive):

Process Arrival Time Burst Time

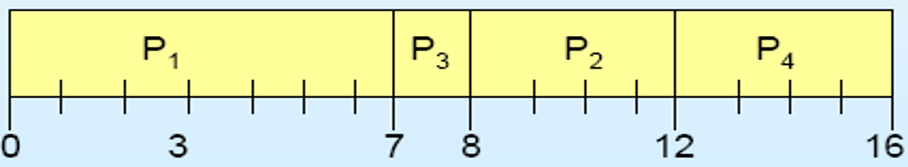
P1 0.0 7

P2 2.0 4

P3 4.0 1

P4 5.0 4

**SJF with non-preemptive scheme**



Process: At first, the process with arrival time 0 will execute after that process with smallest job is executed. In this case P1 is executed first as it has arrival time 0 and after that P3 is executed as it has smallest burst than P2 and P4. If there is tie in burst then the process with shortest arrival time will executed first. In this case P2 and P4 both have burst 4 but arrival time of P4 is first so P2 is executed.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Turnaround Time = Complete Time(CT)-Arrival Time (AT)** | **Waiting time = total turnaround time – burst time** | **Response time = Gantt chart start – Arrival time** |
| P1 | 7 - 0 = 7 | 7 - 7 = 0 | 0 - 0 = 0 |
| P2 | 12 - 2 = 10 | 10 - 4 = 6 | 8 - 2 = 6 |
| P3 | 8 - 4 = 4 | 4 – 1 = 3 | 7 - 4 = 3 |
| P4 | 16 – 5 = 11 | 11 – 4 = 7 | 12 – 5 = 7 |

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

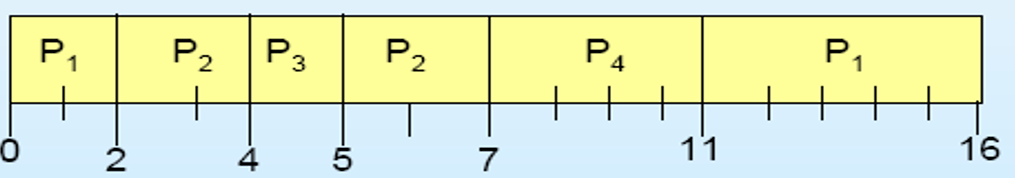
Average Turnaround Time = (7+ 10 + 4 + 11) /4 = 8

Average Response Time = (0+6+3+7)/4 = 4

**Note:**

In Non-Preemptive: waiting time = response time

**SJF with Preemptive scheme:**



**Process:**

First the process with arrival time 0 or with smallest arrival time is executed after that the process executed up to next small arrival time. Then the burst time of executing process is subtracted with arrival time of next process (in this case P1’s burst time = 7-2 =5). P1 is compared with P2’s burst time (here 5 (p1) > 4 (p2)).

The smallest burst is 4 so P2 is executed up to next arrival time and P2’s burst is now 4-2 = 2. Now, P3 arrives its burst is compared with P1 and P2. In this case P3 has smallest burst than P1 and P2 so it is executed up to next arrival. Now all P1, P2, P3 and P4 arrives. The comparison is made and the process with smallest burst is executed.

Calculation is shown in table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Turnaround Time = Complete Time(CT)-Arrival Time (AT)** | **Waiting time = total turnaround time – burst time** | **Response time = Gantt chart start – Arrival time** |
| P1 | 16 – 0 = 16 | 16 – 7 = 9 | 0 - 0 = 0 |
| P2 | 7 – 2 = 5 | 5 – 4 = 1 | 2 – 2 = 0 |
| P3 | 5 – 4 = 1 | 1 – 1 =0 | 4 – 4 = 0 |
| P4 | 11 – 5 = 6 | 6 – 4 = 2 | 7 – 5 = 2 |

Average Turnaround Time = (16 + 5 + 1 +6)/4 = 7

Average Waiting Time = (9 + 1 + 0 +2)/ 4 = 3

Average Response Time = (0 + 0 + 0 +2)/4 = 0.5

1. **Priority Scheduling: (Non-preemptive and Preemptive)**

Here a priority is associated with each process and the CPU is allocated to the process with highest priority. Equal priority process are schedule in FCFS order. Priority are generally indicated by some range of number such as 0 to 7 or 0 to 1000. There is no general agreement on whether 0 is the highest or lowest priority. Some system uses low number to represent low priority, other uses low number for highest priority. But we represent low number as high priority.

Priority can be defined either internally or externally. When define by internally, priority uses measurable quantity or quantities like time limits, memory requirement, the number of files to be open, average I/O and CPU burst etc. external priorities are set by the criteria outside the operating system such as importance of the process, type and amount of fund being paid for computer use etc.

For example:

Process Burst time Priority

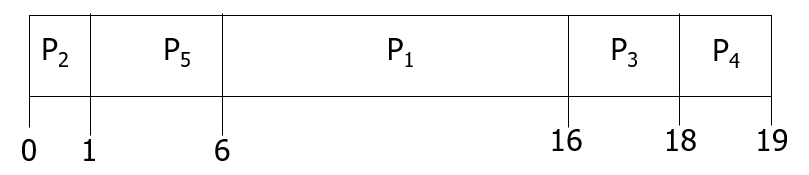
P1 10 3

P2 1 1 (High)

P3 2 4

P4 1 5 (Low)

P5 5 2



Average waiting time = (6+0+16+18+1)/5 = 8.2 ms.

Priority scheduling can be non-preemptive and preemptive. In non-preemptive strategy, when the process arrives at the ready queue, its priority is compared with the currently running process and simply puts the new process at the head of the ready queue.

In preemptive strategy, it will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

**Problem with Priority scheduling:**

It faces indefinite block or **starvation** problem. If there are more high priority process then low priority process have to wait for longer time. This can leave some low priority process waiting indefinitely. That is higher priority process can prevent a low priority process from ever getting the CPU time.

**Solution:**

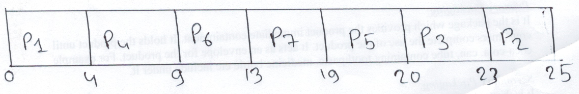
The solution for starvation problem is **aging** which involves gradually increasing the priority of the process that wait in the system for a long time. For example we can increase the priority of waiting process by 1 in every 10 minute.

Example of Priority Scheduling (Non-Preemptive and Preemptive Scheduling):

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Priority | Arrival Time | Burst Time |
| P1 | 2 (Low) | 0 | 4 |
| P2 | 4 | 1 | 2 |
| P3 | 6 | 2 | 3 |
| P4 | 10 | 3 | 5 |
| P5 | 8 | 4 | 1 |
| P6 | 12 (High) | 5 | 4 |
| P7 | 9 | 6 | 6 |

Calculate Turnaround time, waiting time, response time and also average of all.

**Using Non- Preemptive strategy:**



|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Turnaround Time = Complete Time(CT)-Arrival Time (AT)** | **Waiting time = total turnaround time – burst time** | **Response time = Gantt chart start – Arrival time** |
| P1 | 4 – 0 = 4 | 4 – 4 = 0 | 0 – 0 = 0 |
| P2 | 25 – 1 = 24 | 24 – 2 = 22 | 23 – 1 = 22 |
| P3 | 23 – 2 = 21 | 21 – 3 = 18 | 20 – 2 = 18 |
| P4 | 9 – 3 = 6 | 6 – 5 = 3 | 4 – 3 = 1 |
| P5 | 20 – 4 = 16 | 16 – 1 = 15 | 19 – 4 = 15 |
| P6 | 13 – 5 = 8 | 8 – 4 = 4 | 9 – 5 = 4 |
| P7 | 19 – 6 = 13 | 13 – 6 = 7 | 13 – 6 = 7 |

Average Turnaround Time = (4+24+21+6+16+8+23)/7 = 13.14

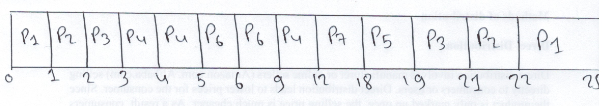
Average Waiting Time = (0+22+18+3+15+4+7)/7 = 9.57

**Process:**

Here, process with smaller arrival time is executed first in this case P1. P1 completes up to its burst time which is 4. Up to 4 ms P2, P3, P4 and P5 has arrived. From this four process P4 have highest priority (10) so it is executed first and process up to its burst time. Up to 9 millisecond all the process have arrived so according to their priority they are given CPU time.

**Using Preemptive Strategy:**

**Note refer your class note for solution:**

****

1. **Round Robin Scheduling: (Preemptive)**

This scheduling algorithm is specially designed for time sharing system. A small unit of time called time-quantum or time-slice is defined. Time quantum is generally from 10 to 100 millisecond in length. The CPU scheduler goes around the ready queue allocating the CPU to each process for a certain time interval. Here ready queue is treated as FIFO queue. New process are added to tail of the queue. Once the running process’s time quantum expired but it still have a CPU burst left then it is added to tail of the ready queue.

The average waiting time of this scheduling is often long. Let us consider following example:

Process Burst

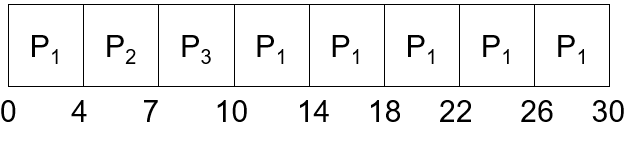
P1 24

P2 3

P3 3

Time quantum: 4 millisecond

Solution:



Waiting time for:

P1 = 10 – 4 = 6 [time it need to wait for second processing].

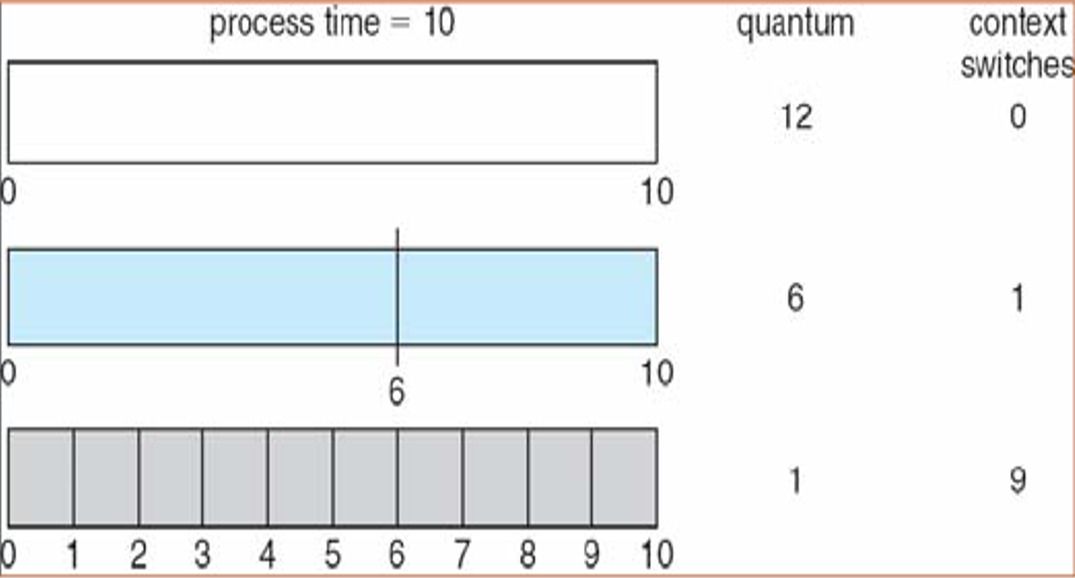
P2 = 4

P3 = 7

Average waiting time: (6+4+7)/3 = 17/3 = 5.66 millisecond

The performance of round robin algorithm depends on size of the time quantum. If the time quantum is extremely large the RR policy is the same as FCFS policy. If the time quantum is extremely small (1 millisecond) the RR policy can results in a large number of context switch.

Let us consider we have one process of 10 unit time and let time quantum be 12, 6 and 1 millisecond. If the time quantum is 12 process finish in less than 1 time quantum with no overhead. If the time quantum is 6 the process complete in 2 quanta resulting in context switch. For the time quantum 1 millisecond, nine context switch will occur which is shown below:



**NOTE: Refer your class note for further example of round robin scheduling**

1. **Multilevel Queue Scheduling:**

In this scheduling algorithm, process are classified into different groups. For e.g. a common division is made between foreground (interactive) processes and background (batch) processes. Such processes may have different response time requirement, so have different scheduling needs. Foreground process may have high priority over background process.

This scheduling algorithm partition the ready queue into several separate queues and process are permanently assigned to one queue based on some property of process such as memory size, priority or process type etc. Each queue has its own scheduling algorithm. For e.g. the foreground queue might be scheduled by a round robin algorithm while the background queue cab be scheduled by an FCFS algorithm.

Here scheduling is performed using two method:

* **Fixed priority preemptive scheduling:**

In this approach, queues are given fixed priority, highest priority queue are executed first and no any process in low priority queue are run unless the queues with highest priority are empty. When the process of low priority queues are running and if another process entered into high priority queue then the process in the low priority queue is preempted. Therefore, possibility of starvation problem can arise. For example:

Let us consider five queues: system processes queue, interactive processes queue, interactive editing processes queues, batch processes queue and student processes queue as shown in figure below.

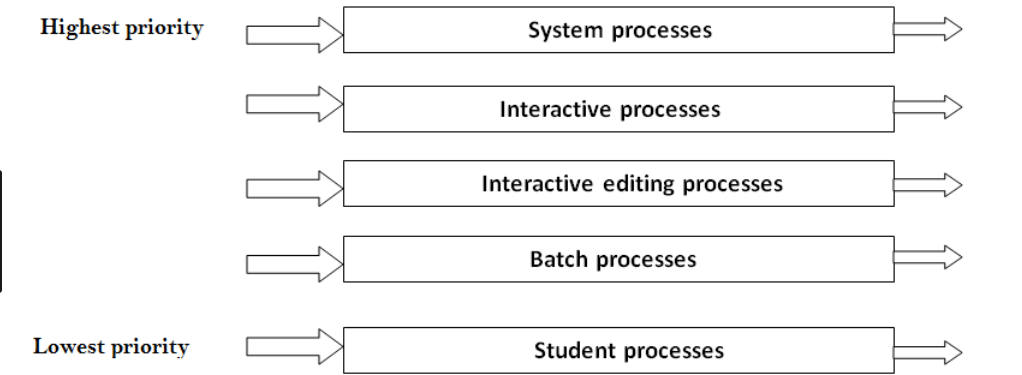


Figure: Multilevel Queue Scheduling

Each queue has absolute priority over lower priority queue. No process in the batch queue can run unless the queue for system processes, interactive processes and interactive editing processes are empty. If the interactive process enter into queue while batch processes queue is running then batch process will be preempted.

* **Time-slice:**

Here each queue gets certain portion of CPU time which it can then schedule among its various processes. For example: foreground queue can be given 80 percent of CPU time for RR scheduling among its process while background queue will receive 20 percent of the CPU time to give it to its process.

1. **Multilevel Feedback Queues Scheduling:**

This algorithm allows a process to move between queues. The process are separated according to the characteristics of their CPU burst. If a process uses too much CPU time it will be moved to a lower priority queue. If a process waits too long in a lower priority queue then it may be moved to a higher priority queue. This prevents the starvation problem.

For example let us consider a multilevel feedback queue scheduler with three queues numbered from 0 to 2 as shown in figure below:

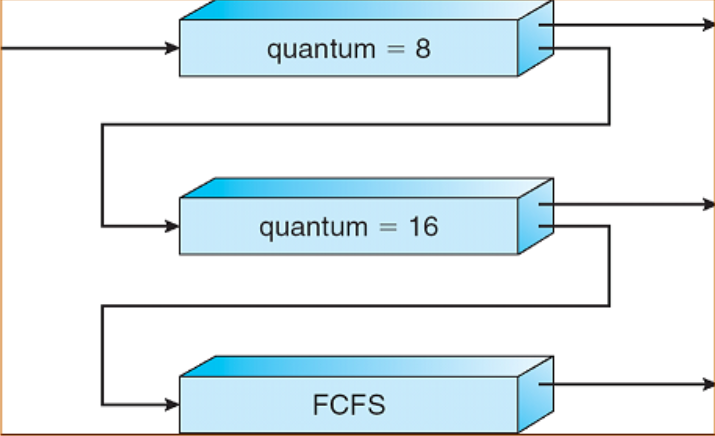


Figure: Multilevel Feedback Queue

Here a process entering the ready queue is put in queue 0 and is given 8 millisecond time. If it does not finish within the time it is moved to the tail of the queue 1. If the queue 0 is empty, the process at head of queue 1 is given a quantum of 16 millisecond. If it does not complete, it is preemptive and is put into queue 2. Process in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.

In general a multilevel feedback queue scheduler is defined by the following parameter:

* The number of queues
* The scheduling algorithm for each queue
* The method used to determine when to upgrade a process to a higher priority queue.
* The 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.

**Real Time Scheduling:**

The feature of a real time system is to respond immediately to a real time process as soon as that process requires CPU. Soft real time system provide no guarantee as to when a critical real time process will schedule. They guarantee only that the process will be given preference over non critical process. Hard real time system have strict requirement where task is serviced by its deadline and service after the deadline has expired is the same as no service at all.

**Minimizing Latency:**

When an event occur, the system must respond to and service it as quickly as possible. Latency or event latency refer to the amount of time elapses from when an event occur to when it is service. It is the time difference between when an event is occurred to response provided by the system for such event. Usually, different event has different latency requirement.

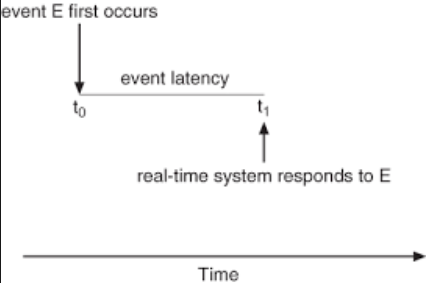


Figure: Event Latency

Two types of latencies affect the performance of real time system:

**Interrupt latency** refers to the period of time from the arrival of an interrupt at the CPU to the start of the routine that service the interrupt. It is crucial for the real time system to minimize interrupt latency to ensure that real time tasks receive immediate attention.

The amount of time required for the scheduling dispatcher to stop one process and start another is known as **dispatch latency**. Providing real time task with immediate access to the CPU ensures that real time operating system minimize this latency as well.

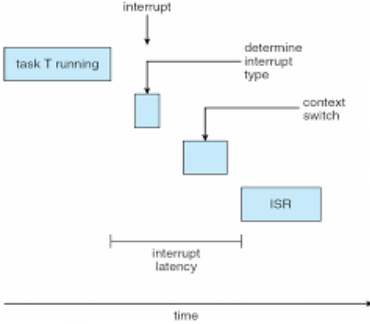


Figure: Interrupt Latency

**Types of Real Time Scheduling:**

1. **Priority Based Scheduling:**

The scheduler for a real time operating system must support a priority based algorithm with preemption. Here the process are consider periodic i.e. they require the CPU at constant intervals (periods). Once the process has acquired CPU it has a fixed processing time t, a deadline d by which it must be served by CPU and a period p. the relationship of the processing time, the dead line and the period can be expressed as 0 <= t <=d <= p.

The process may have to announce its deadline requirement to the scheduler. Then using an admission control algorithm, the scheduler does one of the two thing. It either admit the process guaranteeing that the process will complete on time or rejects the request if it cannot guarantee that the task will be serviced by its deadline.

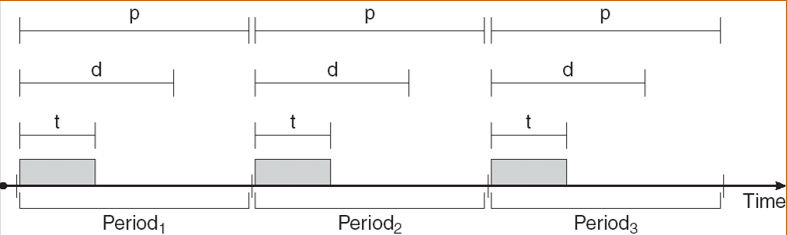


Figure: Periodic Task

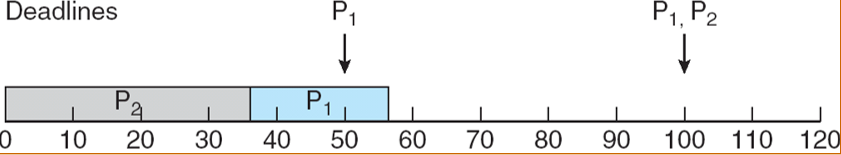


Figure: Scheduling of task when P2 has a higher priority than P1

1. **Rate Monotonic Scheduling:**

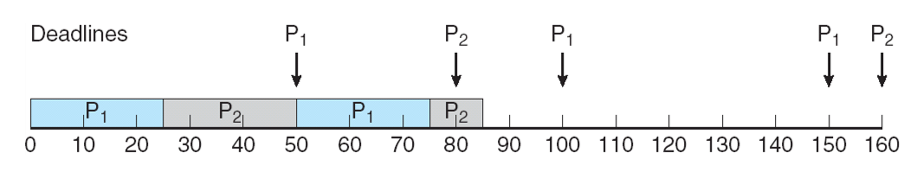
A rate monotonic scheduling algorithm schedule periodic tasks using a static priority policy with preemption. Here each periodic task is assigned a priority inversely based on period i.e. shorter the period higher the priority and higher the period shorter the priority. Every time a process acquires the CPU the duration of its CPU burst is same.

For example

Process Burst period

P1 25 50

P2 35 100



Here, the priority of P1 is higher than P2 because P1’s period is less than P2’s period. First, P1 will execute and it have to finish 25 burst within the period or interval of 50. After 50 interval again P1 will execute and so on. So p1 execute and finish its burst of 25 before 50 period. Now, P2 will start from 25 (where P1 has finished) and execute till 50 period. From 50 to 25 P2 has completed only 25 of its burst (50-25) i.e. still 15 burst is remaining. P2 is stopped in 50 period because P1 arrived at 50 period and P1 has highest priority than P2. So, P2 is preempted and P1 is executed. P1 have to complete 25 burst within 100 period, it will complete in 75 period. Now that remaining 15 burst of P2 is executed. The process continues so on..

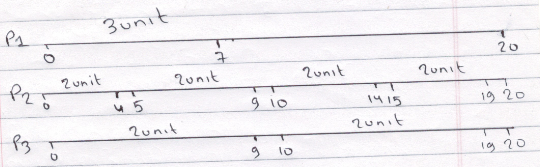
1. **Earliest Deadline First Scheduling:**

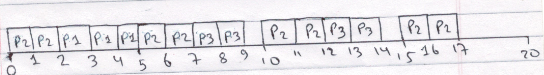
This algorithm dynamically assigns priorities according to deadline. The earlier the deadline the higher the priority, the later the deadline lower the priority. When a process becomes runnable it must announce its deadline requirement to system. It solves the problem of missing deadline of rate monotonic scheduling.

For example:

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst | Deadline | Period |
| P1 | 3 | 7 | 20 |
| P2 | 2 | 4 | 5 |
| P3 | 2 | 9 | 10 |

Solution:





Here, process with less deadline have high priority i.e. (P2>P1>P3). P2 have to complete its 2 unit burst in every 5 interval within the deadline of 4. Similarly, P1 have to complete its 3 unit of burst in every 20 interval within the deadline of 7 and P3 have to complete its 2 unit of burst in every 10 interval within deadline of 9.

First P1 will executes and completes its burst of 2 unit within less than of its deadline thus meeting its deadline. After this P1 will execute as it has less deadline that P3. So, P1 completes its 3 unit of burst within its deadline of 19. Now, P2 arrives at period 5 so it will complete its 2 unit of burst in less than its deadline i.e. 9. Now, P3 will execute and complete its burst of 2 unit within its deadline (9) and so on…