



Dynamic Priority Scheduling Algorithm

Content

- ◎ **Priority Scheduling**
 - ◎ **Non-Preemptive Scheduling**
 - ◎ **Preemptive Scheduling**
- ◎ **Dynamic priority scheduling**
- ◎ **Earliest Deadline First**
- ◎ **Least Slack Time**
- ◎ **Difference between EDF and LST**

Priority Scheduling

- ① In priority scheduling, each process has a priority which is an integer value assigned to it. The **smallest integer** is considered as the **highest priority** and the **largest integer** is considered as the **lowest priority**. The process with the highest priority gets the CPU first.
- ① In rare systems, the largest number might be treated as the highest priority also so it all depends on the implementation.
- ① If priorities are internally defined then some measurable quantities such as time limits, memory requirements, the number of open files and the ratio of average I/O burst to average CPU burst are used to compute priorities.
External priorities are assigned on the basis of factors such as the importance of process, the type and amount of funds been paid for computer use, the department sponsoring of the work, etc.



Priority Non-Preemptive Scheduling :

- ⦿ Unlike priority preemptive scheduling, even if a task with higher priority does arrive, it has to wait for the current task to release the CPU before it can be executed. It is often used in various hardware procedures such as timers, etc.

Priority Preemptive Scheduling :

- ⦿ Sometimes it is important to execute higher priority tasks immediately even when a task is currently being executed. For example, when a phone call is received, the CPU is immediately assigned to this task even if some other application is currently being used. This is because the incoming phone call has a higher priority than other tasks. This is a perfect example of priority preemptive scheduling. If a task with higher priority than the current task being executed arrives then the control of the CPU is taken from the current task and given to the higher priority task.

Dynamic priority scheduling

- ◎ **Dynamic priority scheduling** is a type of [scheduling algorithm](#) in which the priorities are calculated during the execution of the system. The goal of dynamic priority scheduling is to adapt to dynamically changing progress and to form an optimal configuration in a self-sustained manner. It can be very hard to produce well-defined policies to achieve the goal depending on the difficulty of a given problem.
- ◎ [Earliest deadline first scheduling](#) and [Least slack time scheduling](#) are examples of Dynamic priority scheduling algorithms.

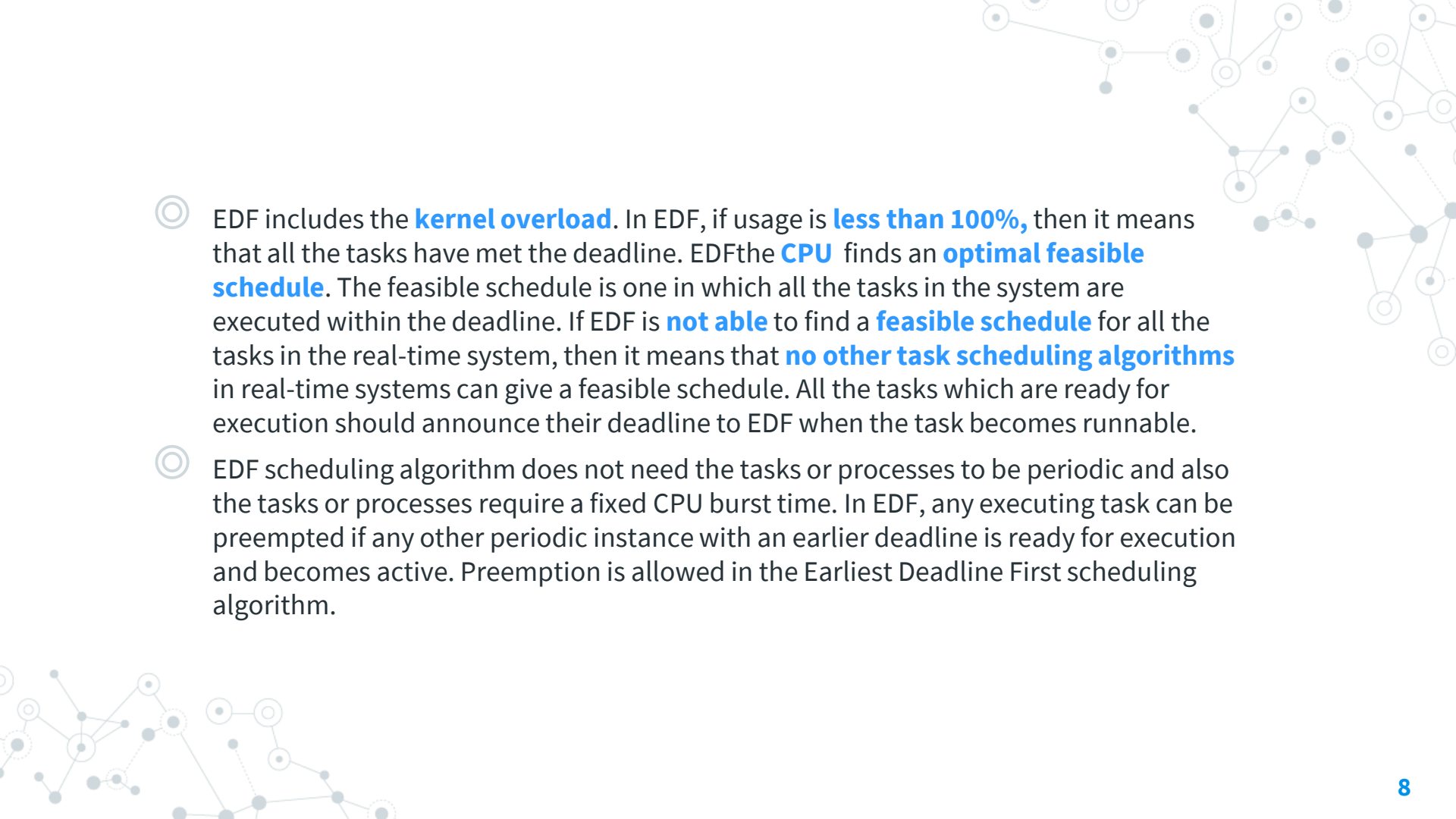


EDF

Earliest Deadline First

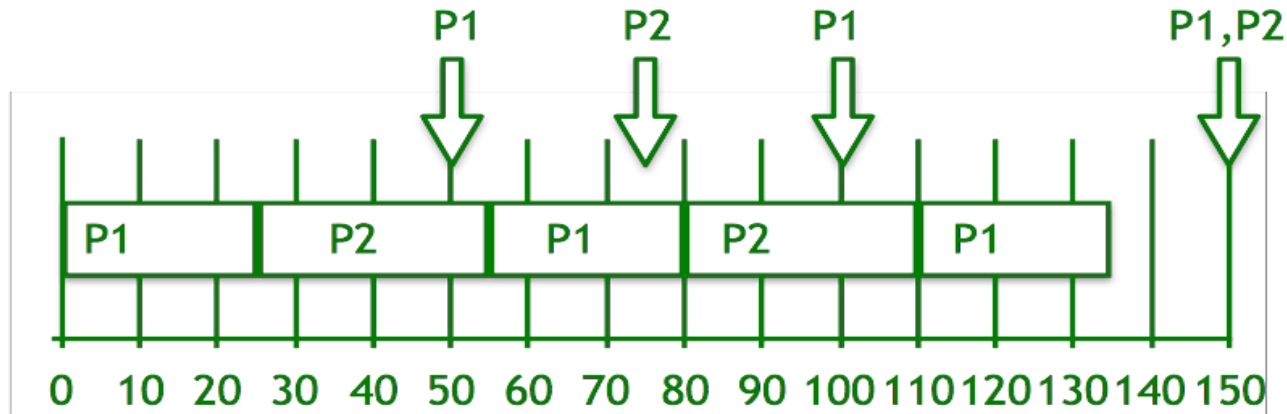
Earliest Deadline First (EDF)

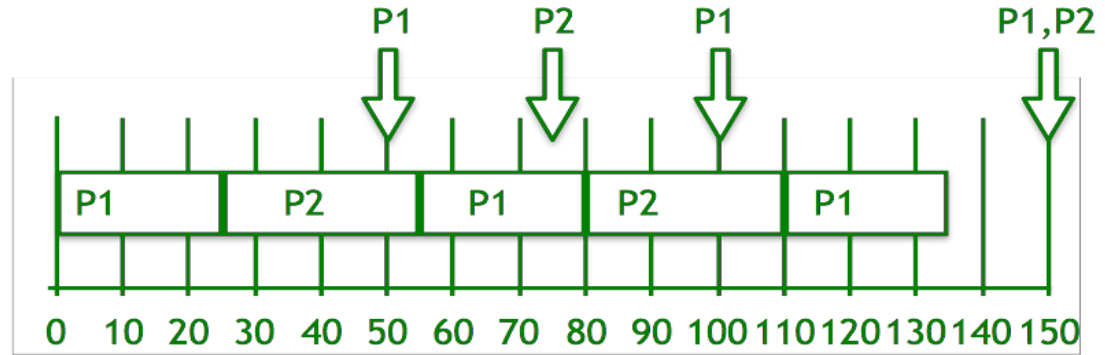
- ◎ **Earliest Deadline First (EDF)** is an optimal dynamic priority scheduling algorithm used in **real-time systems**.
It can be used for both static and dynamic real-time scheduling.
- ◎ EDF uses priorities to the jobs for scheduling. It assigns priorities to the task according to the absolute deadline. The task whose deadline is **closest gets the highest priority**. The priorities are assigned and changed in a dynamic fashion. EDF is **very efficient** as compared to other scheduling algorithms in real-time systems. It can make the **CPU utilization** to about **100%** while still guaranteeing the deadlines of all the tasks.

- 
- ① EDF includes the **kernel overload**. In EDF, if usage is **less than 100%**, then it means that all the tasks have met the deadline. EDFthe **CPU** finds an **optimal feasible schedule**. The feasible schedule is one in which all the tasks in the system are executed within the deadline. If EDF is **not able** to find a **feasible schedule** for all the tasks in the real-time system, then it means that **no other task scheduling algorithms** in real-time systems can give a feasible schedule. All the tasks which are ready for execution should announce their deadline to EDF when the task becomes runnable.
 - ① EDF scheduling algorithm does not need the tasks or processes to be periodic and also the tasks or processes require a fixed CPU burst time. In EDF, any executing task can be preempted if any other periodic instance with an earlier deadline is ready for execution and becomes active. Preemption is allowed in the Earliest Deadline First scheduling algorithm.

Example:

- ⊙ Consider two processes P1 and P2.
- ⊙ Let the period of P1 be $p_1 = 50$, Let the processing time of P1 be $t_1 = 25$
- ⊙ Let the period of P2 be $p_2 = 75$, Let the processing time of P2 be $t_2 = 30$





◎ **Steps for solution:**

- ◎ Deadline of P1 is earlier, so priority of $P1 > P2$.
- ◎ Initially P1 runs and completes its execution of 25 time.
- ◎ After 25 times, P2 starts to execute until 50 times, when P1 is able to execute.
- ◎ Now, comparing the deadline of $(P1, P2) = (100, 75)$, P2 continues to execute.
- ◎ P2 completes its processing at time 55.
- ◎ P1 starts to execute until time 75, when P2 is able to execute.
- ◎ Now, again comparing the deadline of $(P1, P2) = (100, 150)$, P1 continues to execute.
- ◎ Repeat the above steps...
- ◎ Finally at time 150, both P1 and P2 have the same deadline, so P2 will continue to execute till its processing time after which P1 starts to execute.

A decorative network diagram in the top-left corner, consisting of various sized grey circles (nodes) connected by thin grey lines (edges). Some nodes are solid, while others are hollow with a dashed border. The network is dense and irregular.

LST

Least Slack Time

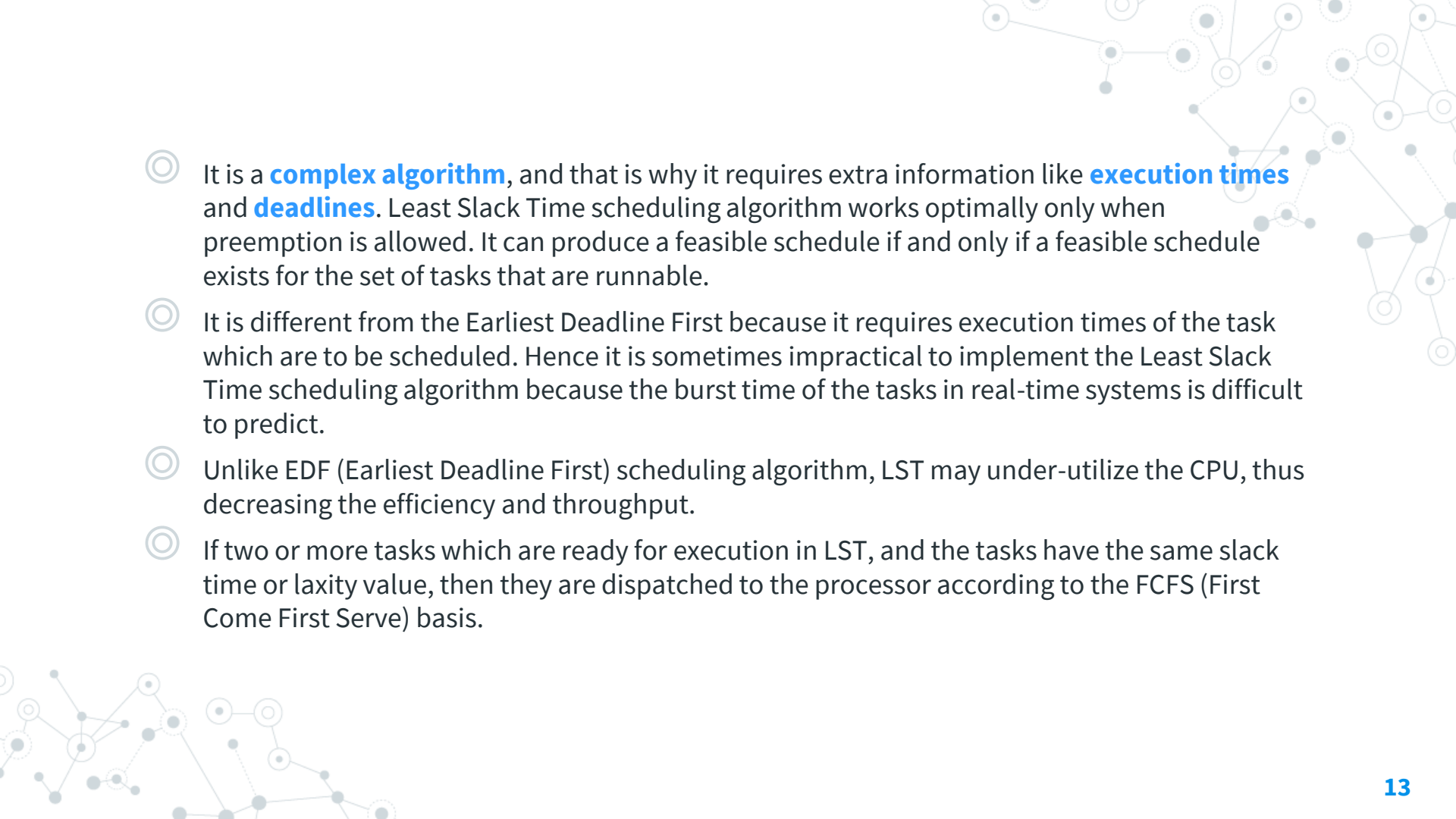
Least Slack Time (LST)

- ◎ **Least Slack Time (LST)** is a dynamic priority-driven scheduling algorithm used in real-time systems.
- ◎ In LST, all the tasks in the system are assigned some priority according to their slack time. The task which has the least slack time has the highest priority and vice versa.
- ◎ Priorities to the tasks are assigned dynamically.
- ◎ Slack time can be calculated using the equation:

$$\text{slack_time} = (D - t - e')$$

- ◎ Here **D** : Deadline of the task
t : Real time when the cycle starts.
e' : Remaining Execution Time of the task.

The task which has the minimal slack time is dispatched to the CPU for its execution as it has the highest priority.

- 
- ◎ It is a **complex algorithm**, and that is why it requires extra information like **execution times** and **deadlines**. Least Slack Time scheduling algorithm works optimally only when preemption is allowed. It can produce a feasible schedule if and only if a feasible schedule exists for the set of tasks that are runnable.
 - ◎ It is different from the Earliest Deadline First because it requires execution times of the task which are to be scheduled. Hence it is sometimes impractical to implement the Least Slack Time scheduling algorithm because the burst time of the tasks in real-time systems is difficult to predict.
 - ◎ Unlike EDF (Earliest Deadline First) scheduling algorithm, LST may under-utilize the CPU, thus decreasing the efficiency and throughput.
 - ◎ If two or more tasks which are ready for execution in LST, and the tasks have the same slack time or laxity value, then they are dispatched to the processor according to the FCFS (First Come First Serve) basis.

Example:

| | arrival | duration | deadline |
|-----|---------|----------|----------|
| T 1 | 0 | 10 | 33 |
| T 2 | 4 | 3 | 28 |
| T 3 | 5 | 10 | 29 |

At time $t=0$: Only task T1, has arrived. T1 is executed till time $t=4$.

At time $t=4$: T2 has arrived.

Slack time of T1: $33-4-6=23$

Slack time of T2: $28-4-3=21$

Hence T2 starts to execute till time $t=5$ when T3 arrives.

At time $t=5$:

Slack Time of T1: $33-5-6=22$

Slack Time of T2: $28-5-2=21$

Slack Time of T3: $29-5-10=12$

Hence T3 starts to execute till time $t=13$

At time $t=13$:

Slack Time of T1: $33-13-6=14$

Slack Time of T2: $28-13-2=13$

Slack Time of T3: $29-13-2=14$

Hence T2 starts to execute till time $t=15$

At time $t=15$:

Slack Time of T1: $33-15-6=12$

Slack Time of T3: $29-15-2=12$

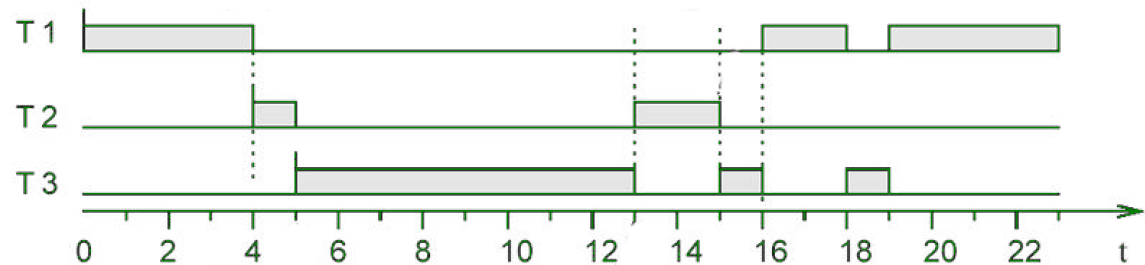
Hence T3 starts to execute till time $t=16$

At time $t=16$:

Slack Time of T1: $33-16-6=11$

Slack Time of T3: $29-16=12$

Hence T1 starts to execute till time $t=18$ and so on..



| EDF | LST |
|--|---|
| Task having shortest deadline is scheduled first in it. | Task having minimum slack time is scheduled first in it. |
| It assigns priority to tasks according to their deadlines. | It assigns tasks according to their slack time. |
| It can be used as both static and dynamic scheduling. | It is used only as dynamic scheduling. |
| Execution time of a task is not required. | It requires execution time of a task. |
| It is a simple and optimal algorithm. | It is a complex algorithm. |
| It can be implemented on any set of tasks. | It can only be implemented on set of tasks having their burst time. |
| It completely utilizes the CPU (even sometimes 100%). | It may under-utilize the CPU. |
| It increases the efficiency and throughput of the processor. | It may decrease the efficiency and throughput of the processor. |

**Thank you
for you
attention**

