Time Planning in RTS

Real-time systems are systems that carry real-time tasks. These tasks need to be performed immediately with a certain degree of urgency. In particular, these tasks are related to control of certain events (or) reacting to them. Real-time tasks can be classified as hard real-time tasks and soft real-time tasks.

A hard real-time task must be performed at a specified time which could otherwise lead to huge losses. In soft real-time tasks, a specified deadline can be missed. This is because the task can be rescheduled (or) can be completed after the specified time.

In real-time systems, the scheduler is considered as the most important component which is typically a short-term task scheduler. The main focus of this scheduler is to reduce the response time associated with each of the associated processes instead of handling the deadline.

If a preemptive scheduler is used, the real-time task needs to wait until its corresponding tasks time slice completes. In the case of a non-preemptive scheduler, even if the highest priority is allocated to the task, it needs to wait until the completion of the current task. This task can be slow (or) of the lower priority and can lead to a longer wait.

A better approach is designed by combining both preemptive and non-preemptive scheduling. This can be done by introducing time-based interrupts in priority based systems which means the currently running process is interrupted on a time-based interval and if a higher priority process is present in a ready queue, it is executed by preempting the current process.

Based on schedulability, implementation (static or dynamic), and the result (self or dependent) of analysis, the scheduling algorithm are classified as follows.

Static table-driven approaches:
These algorithms usually perform a static analysis associated with scheduling and capture the schedules that are advantageous. This helps in providing a schedule that can point out a task with which the execution must be started at run time.

Static priority-driven preemptive approaches:
Similar to the first approach, these type of algorithms also uses static analysis of scheduling. The difference is that instead of selecting a particular schedule, it provides a useful way of assigning priorities among various tasks in preemptive scheduling.

Dynamic planning-based approaches:
Here, the feasible schedules are identified dynamically (at run time). It carries a certain fixed time interval and a process is executed if and only if satisfies the time constraint.

Dynamic best effort approaches:
These types of approaches consider deadlines instead of feasible schedules. Therefore the task is aborted if its deadline is reached. This approach is used widely is most of the real-time systems.

Firm Real-time Tasks:
Firm real-time tasks are such type of real-time tasks which are associated with time bound and the task need to produce the result within the deadline. Although firm real-time task is different from hard real-time task as in hard real-time once deadline is crossed and task is not completed, system fails but in case of firm real-time task even after the passing of deadline, system does not fail.

Example:
1. Video conferencing;
2. Satellite based tracking.

Soft Real-time Tasks:
Soft real-time tasks are such type of real-time tasks which are also associated with time bound but here timing constraints are not expressed as absolute values. In soft real-time tasks, even after the deadline result is not considered incorrect and system failure does not occur.

Example:
1. Web browsing;
2. Railway Ticket Reservation.

FIRM REAL-TIME TASKS	SOFT REAL-TIME TASKS
It needs to be completed within the deadline.	It also needs to be completed within deadline but not strictly.
The value of associated timing constraint is taken as absolute.	The value of associated timing constraint is taken as average value.
The utility of result becomes zero after the deadline.	The utility of result decreases after the deadline but it gradually becomes zero.
Result obtained after deadline is considered incorrect.	Result obtained after deadline is not incorrect.
It is widely used in multimedia applications.	It is less used in such applications.
It is less used in practical applications.	It is widely used in practical applications.
Example: Satellite based tracking.	Example: Railway ticket reservation.

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:

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.

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.

	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..

THANK YOU