CSE 422
Real Time Operating System

Real-Time Scheduling

- **Real-Time Scheduling** is designed to ensure that tasks are executed in a predictable and timely manner, with minimal delay or latency.
- The scheduler assigns priorities to tasks based on their importance and urgency and allocates system resources accordingly to ensure that critical tasks are executed on time.

Real-time scheduling can be classified into two categories: hard real-time scheduling and soft real-time scheduling.

1. **Hard real-time scheduling** requires that tasks are completed within a specific deadline, and any delay can result in system failure or loss of data.
2. **Soft real-time scheduling**, on the other hand, allows for some degree of delay but still requires that tasks are completed within a reasonable timeframe.

Common Approaches to Real-Time Scheduling: There are several common approaches to real-time scheduling that are used to ensure that tasks are executed in a timely and efficient manner.

These approaches include

1. Clock-Driven Approach:

- The Clock Driven approach is a real-time scheduling technique that uses a fixed clock to divide time into equal intervals and assigns tasks to each interval based on their deadline and priority.
- In this approach, tasks are assigned to a fixed time slot, and the scheduler ensures that each task is executed within its allocated time slot.
- If a task misses its deadline, it is either rescheduled or dropped, depending on the criticality of the task.
- The Clock Driven approach is helpful in systems that have a high degree of predictability and where tasks have fixed execution times.
- It is commonly used in embedded systems, where tasks are executed in a deterministic and predictable manner.
- One advantage of the Clock Driven approach is that it provides a simple and efficient way to allocate system resources, as tasks are scheduled based on their priority and deadline.
- it can be less flexible than other scheduling techniques, as it assumes that all tasks have fixed execution times and cannot adapt to changing system conditions.

Here's an example of how clock-driven scheduling works:

Execution Time (c)	Task A	Task(B)	Task(C)
0 ms			
10 ms	A		
20 ms		B	
30 ms			C
40 ms	A		
50 ms		B	
60 ms			C
70 ms	A		
80 ms		B	
90 ms			C
100 ms	A		

In this example,

- The system clock is divided into time slots of 10 ms, and three tasks, A, B, and C, are scheduled to run.
- Task A is assigned the first time slot, task B is assigned the second time slot, and task C is assigned the third time slot.
- At time 0 ms, no tasks are running.
- At time 10 ms, task A starts running and continues until the end of its time slot at 20 ms.
- At time 20 ms, task B starts running and continues until the end of its time slot at 30 ms.
- At time 30 ms, task C starts running and continues until the end of its time slot at 40 ms.
- The process repeats until all tasks have completed their execution.

2. Weighted Round Robin (WRR):

- Weighted Round Robin (WRR) is a real-time scheduling algorithm that is an extension of the Round Robin approach.
- In this approach, tasks are assigned a weight value that determines the amount of CPU time they receive during each round.
- Tasks with higher weight values are allocated more CPU time than tasks with lower weight values.

Here is an example diagram of the weighted round-robin approach:

Task A: weight 3, requires 15 ms to complete
Task B: weight 2, requires 10 ms to complete
Task C: weight 1, requires 5 ms to complete
Time quantum: 5 ms

Execution Time (C)	Task A	Task B	Task C
0 - 5 ms	A		
5 - 10 ms		B	
10 -15 ms			C
15 -20 ms	A		-
20 -25 ms		B	-
25 -30 ms	A	-	-

- At time 0 ms, the scheduler starts with Task A since it has the highest weight.
- Task A is executed for the first time quantum of 5 ms, until time 5 ms.
- Since Task A has not been completed, it is moved to the end of the queue, and the scheduler switches to Task B, which is executed for the next time quantum of 5 ms, until time 10 ms.

- Since Task B has not been completed, it is moved to the end of the queue, and the scheduler switches to Task C, which is executed for the next time quantum of 5 ms, until 15 ms.
- Since Task C has been completed, it is removed from the schedule and the scheduler switches back to Task A, which is executed for the next time quantum of 5 ms, until time 20 ms.
- Since Task A has not been completed, it is moved to the end of the queue, and the scheduler switches back to Task B, which is executed for the next time quantum of 5 ms, until time 25 ms.
- Since Task B has been completed, it is removed from the schedule, and the scheduler switches back to Task A, which is executed for the final time quantum of 5 ms, until time 30 ms.

Priority-Driven Approach:

- **The priority-driven approach is a real-time scheduling algorithm that assigns a priority to each task, based on its importance or urgency.**
- **The scheduler then schedules tasks based on their priority, with higher-priority tasks being executed before lower-priority tasks**

Here is an example of the priority-driven approach:

Task A: priority 1, requires 20 ms to complete
Task B: priority 2, requires 15 ms to complete
Task C: priority 3, requires 10 ms to complete

Task A	Task B	Task C
20 ms	15 ms	10 ms
0-20	20-35	35-45

- At time 0 ms, the scheduler starts with Task A since it has the highest priority.
- Task A is executed until it completes at 20 ms.
- Since Task A has been completed, the scheduler switches to Task B, which is the next highest priority task.
- Task B is executed until it completes at 35 ms.
- Finally, the scheduler switches to Task C, which is the lowest-priority task.
- **Task C is executed until it completes at 45 ms.**

The EffectiveDeadlineFirst (EDF) algorithm:

- The Effective Deadline First (EDF) algorithm is a real-time scheduling algorithm that schedules tasks based on their deadline.
- The task with the earliest deadline is given the highest priority and is scheduled first.
- This algorithm ensures that tasks with the shortest deadlines are executed first, which can help to meet critical timing requirements in real-time systems.

Here's an example of how the EDF algorithm works:

Task A: deadline 5 ms, requires 10 ms to complete
Task B: deadline 3 ms, requires 5 ms to complete
Task C: deadline 7 ms, requires 15 ms to complete

Execution Time (C)	Task A	Task B	Task C
0 - 5 ms		B	
5 - 15 ms	A		
15 - 30 ms			C

At time 0 ms, the scheduler starts with Task B since it has the earliest deadline.

- **Task B is executed until it completes at 5 ms.**
- **Since Task B has been completed, the scheduler switches to Task A, which has the next earliest deadline.**
- **Task A is executed until it completes at 15 ms.**
- **Finally, the scheduler switches to Task C, which has the latest deadline. Task C is executed until it completes at 30 ms.**

Least-Slack-Time-First (LST) Algorithms:

- Least-Slack-Time-First (LST) is a real-time scheduling algorithm that is used to optimize the scheduling of tasks based on their deadlines and processing times.
- LST chooses the task with the least slack time as the next task to be executed.
- Slack time is defined as the difference between a task's deadline and the time remaining until the task must be completed.

  - For **example**, if a task has a deadline of 100ms and has already been running for 80ms, then its slack time is 20ms (i.e., the difference between 100ms and 80ms).

The LST algorithm works as follows:

1. Initialize the slack time for all tasks in the system.
2. Select the task with the smallest slack time as the next task to be executed.
3. Execute the selected task for the remaining slack time.
4. Recalculate the slack time for all tasks that have not been completed.
5. Repeat steps 2-4 until all tasks have been completed.

Here's an example of how the LST algorithm works:

Task A: deadline 50 ms, requires 20 ms to complete
Task B: deadline 30 ms, requires 15 ms to complete
Task C: deadline 40 ms, requires 10 ms to complete

Execution Time (C)	Task A	Task B	Task C
0 - 15 ms		B	
15 - 25 ms			C
25 - 45 ms	A		

- At time 0 ms, all tasks are ready to be executed, so the scheduler selects the task with the smallest slack time, which is Task B.
- Task B is executed until it completes at 15 ms.
- At 15 ms, the scheduler checks the slack times of the remaining tasks.
- Task A has a slack time of 35 ms, and Task C has a slack time of 25 ms.
- Since Task C has the smallest slack time, it is given the highest priority and is executed next.

- Task C is executed until it completes at 25 ms.
- Finally, the scheduler switches to Task A, which has the largest slack time.
- Task A is executed until it completes at 45 ms.

EffectiveDeadlineFirst (EDF) Vs Least-Slack-Time-First (LST) Algorithms:

Feature	EDF	LST
Definition	Prioritizes tasks based on their relative deadlines.	Prioritizes tasks based on the amount of time remaining until their deadlines.
Type of scheduling	Preemptive	Preemptive
Algorithm complexity	Moderate	Moderate
Real-time systems	Suitable for hard real-time systems	Suitable for Soft real-time systems
Deadline Requirement	Each task must have a deadline specified.	Each task must have a deadline specified.
Selection Criteria	Tasks with the earliest deadlines are given priority.	Tasks with the least slack time (time remaining until the deadline) are given priority.
Utilization	is Better for high-utilization systems.	Better for low utilization systems.

Rate Monotonic Algorithm:

- The Rate Monotonic (RM) algorithm is a real-time scheduling algorithm that assigns priorities to tasks based on their periods, with shorter-period tasks having higher priority.
- This algorithm assumes that the tasks are periodic and that their execution times are constant and known in advance.

Here's an example of how the RM algorithm works:

Task A: period 50 ms, requires 20 ms to complete
Task B: period 30 ms, requires 15 ms to complete
Task C: period 40 ms, requires 10 ms to complete

Execution Time (C)	Task A	Task B	Task C
0 - 15 ms		B	
15 - 25 ms			C
25 - 45 ms	A		

- At time 0 ms, all tasks are ready to be executed, so the scheduler selects the task with the highest priority, which is Task B.
- Task B is executed until it completes at 15 ms.
- At 15 ms, the scheduler checks which task is ready to be executed next.
- Since Task C has a period of 40 ms, it is prepared to be executed next.
- Task C is executed until it completes at 25 ms.
- At 25 ms, the scheduler checks which task is prepared to be executed next.
- Since Task A has a period of 50 ms, it is prepared to be executed next.
- Task A is executed until it completes at 45 ms.

The scheduler then repeats the cycle, executing the tasks in the order of priority.

Thank You!