Odile SCHAEFFER 4Ts1

FINAL ASSIGNMENT : Schedule search

1. Introduction

This report presents the solution for scheduling a given set of tasks in a real-time system, ensuring the conditions for both schedulability and minimized waiting time are met. The main objectives of this task are to check if the system is schedulable, develop a non-preemptive scheduling algorithm, and analyze the waiting time behavior under different conditions, with a special focus on task τ_5.

2. Task Set Analysis

The given task set consists of the following tasks:

Task	Execution Time (C_i)	Deadline (T_i)
τ_1	2	10
τ_2	3	10
τ_3	2	20
τ_4	2	20
τ_5	2	40
τ_6	2	40
τ_7	3	80

3. Assumptions and Considerations

In order to evaluate the schedulability of the task set, we need to make the following assumptions:

1. **Utilization-based Schedulability Check:**
   We will use the utilization-based approach to check if the total processor utilization does not exceed 100%.
2. **Earliest Deadline First (EDF) Scheduling:**
   EDF is used as the scheduling policy. This is a dynamic priority scheduling algorithm where tasks with the earliest deadline are given priority.
3. **Hyperperiod Calculation:**
   The hyperperiod of the task set is computed using the least common multiple of the periods (deadlines) of all tasks.
4. **Non-preemptive Scheduling:**
   Tasks will be scheduled in a non-preemptive manner, meaning that once a task starts executing, it will run to completion before another task can preempt it.
5. **Task 5 Deadline Missed:**
   In the second part of the analysis, task τ5 is allowed to miss its deadline in order to minimize the total waiting time.

4. Methodology

The approach to solving this problem consists of the following steps:

1. **Schedulability Check:**
   First, we check the system's schedulability by calculating the total utilization of the system. If the total utilization is less than or equal to 1, the system is schedulable. We use the formula:
   $$U = \sum \left(\frac{C_i}{T_i} \right)$$
2. **Hyperperiod Calculation:**
   Next, we calculate the hyperperiod of the task set by finding the least common multiple (LCM) of the task periods (deadlines).
3. **Job Generation:**
   Jobs are generated based on the task periods and execution times, where each task is repeated periodically until the hyperperiod.
4. **Scheduling:**
   Tasks are scheduled using the EDF policy, where at each time point, the task with the earliest deadline is executed.
5. **Minimization of Waiting Time:**
   In the second part of the analysis, we allow task τ_5 to miss its deadline to minimize the total waiting time for the remaining tasks.

5. Results

The results are divided into two parts. The first execution ensures that no deadlines are missed, and the second execution minimizes the waiting time while allowing task τ_5 to miss a deadline.

5.1 First Execution: Normal Schedule (No Deadline Missed)

In the first execution, we ensure that no task misses its deadline. The tasks are scheduled in the order of earliest deadline first, and the execution is performed with the goal of meeting all deadlines.

Execution Order: Task execution order will be displayed, showing the timeline of task completions. For each task, we check whether it meets the deadline or not.

5.2 Second Execution: Minimizing Waiting Time (Allowing Task τ5 to Miss Deadline)

In the second execution, we minimize the total waiting time by allowing task τ5 to miss its deadline. This approach optimizes the overall task schedule by prioritizing the other tasks and reducing the idle time.

6. Computational Complexity

The computational complexity of the algorithm depends primarily on two operations: sorting the jobs by their deadlines and calculating the hyperperiod. The complexity can be summarized as follows:

1. **Sorting Jobs:**
   The sorting of jobs by their deadlines is the most computationally expensive step. The complexity of sorting is $O(n \ \log n)$, where N is the number of jobs.
2. **Hyperperiod Calculation:**
   The calculation of the hyperperiod involves finding the least common multiple of task periods. This can be done in $O(T)$, where T is the number of tasks. Thus, the overall complexity of the algorithm is dominated by the sorting step and is $O(n \ \log n \ )$, where n is the number of jobs generated.

7. Conclusion

In conclusion, the task set was found to be schedulable under the given conditions. The first scheduling ensured that no deadlines were missed, while the second scheduling optimized the waiting time by allowing task τ_5 to miss its deadline. The proposed scheduling algorithm is efficient with a time complexity of $O(n \ \log n)$, and the approach provides a clear method for managing real-time task sets.