ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-F-404: Real-Time Operating Systems

Uniprocessor Scheduling

Herma ELEZI
Zexin ZHANG

Prof. Joël Goossens

PhD Student. Yannick Molinghen

2024

Contents

Introduction

This project implements and analyzes three classical real-time scheduling algorithms: Deadline Monotonic (DM), Earliest Deadline First (EDF), and Round Robin (RR). The scope encompasses several key aspects:

Algorithm Implementation

- Implementation of three scheduling algorithms with different priorities:
  - DM: Fixed-priority scheduling where tasks with shorter relative deadlines get higher priorities
  - EDF: Jobs are scheduled based on their absolute deadline, with earlier deadlines having higher priorities
  - RR: Time-sharing scheduling where each task gets a fixed time quantum (1 time unit) in a cyclic manner
- Development of both quick tests and simulation-based schedulability analysis
- Implementation of feasibility interval calculations for different algorithms

Dataset Considerations

Due to the computational intensity of the original dataset, particularly for task sets with high utilization or large numbers of tasks, we utilized a second dataset with more manageable parameters.

The project aims to provide insights into the relative performance of these scheduling algorithms under different conditions, their success rates, and the factors that influence schedulability. The results and analysis serve to validate theoretical expectations and provide practical insights into real-time scheduling algorithm behavior.

Methodology

Our methodology focuses on implementing and analyzing three different scheduling algorithms: Deadline Monotonic (DM), Earliest Deadline First (EDF), and Round Robin (RR). The implementation follows a systematic approach combining both theoretical analysis and practical simulation.

Scheduling Algorithms Implementation

For each scheduling algorithm, we implemented a two-phase schedulability test approach. The first phase consists of a quick test that can quickly determine if a task set is definitely schedulable or unschedulable, avoiding unnecessary simulation. The quick test logic varies for each algorithm based on their theoretical properties.

For DM (Deadline Monotonic), the quick test is based on two conditions. First, it checks the necessary condition that the total utilization must not exceed 1 ($\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$). If this condition is violated, the task set is definitely unschedulable. However, meeting this condition alone is not sufficient to guarantee schedulability under DM.

The second part of DM's quick test involves response time analysis. For each task τ_i, we calculate its worst-case response time R_i using the iterative formula:

$R_i^{(k+1)} = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil C_j$

where $hp(i)$ is the set of tasks with higher priority than τ_i (i.e., tasks with shorter deadlines in DM). The iteration starts with $R_i^{(0)} = C_i$ and continues until either:

- $R_i^{(k+1)} = R_i^{(k)}$ (convergence), in which case we check if $R_i \leq D_i$
- $R_i^{(k+1)} > D_i$ (task is unschedulable)

If any task's response time exceeds its deadline, the task set is unschedulable. If all tasks' response times are within their deadlines, the task set is schedulable.

For EDF (Earliest Deadline First), the quick test strategy differs based on the relationship between tasks' deadlines and periods. The test leverages EDF's optimality property for implicit deadline systems.

For implicit deadline task sets (where $D_i = T_i$ for all tasks), EDF is optimal and the quick test only needs to verify the processor utilization bound condition: $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$. If this condition is met, the task set is guaranteed to be schedulable under EDF without needing simulation. This is because EDF can schedule any synchronous implicit deadline task set as long as the total utilization does not exceed 100%.

If the taskset is not an implicit deadline taskset, we will calculate the feasibility interval, then simulate the scheduler in the feasibility interval to determine if the taskset is schedulable. The feasibility interval length L is the smallest positive solution to this equation:

$L = \sum_{i=1}^{n} \left\lceil \frac{L}{T_i} \right\rceil C_i$

We can compute this value using the method below:

$$\begin{cases} w_0 \stackrel{\text{def}}{=} \sum_{i=1}^{n} C_i & \text{(initialisation)} \\ w_{k+1} = \sum_{i=1}^{n} \left\lceil \frac{w_k}{T_i} \right\rceil C_i & \text{(iteration)} \end{cases}$$

For Round Robin (RR), the implementation uses a time-quantum based approach. The quick test is simpler than both DM and EDF, only checking the necessary utilization condition ($\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$). If the utilization test fails, the task set is definitely unschedulable. after passing the utilization test, we will do simulation for RR by using the hyperperiod P (least common multiple of all task periods) as the feasibility interval.

Algorithm-Specific Implementations

The Deadline Monotonic algorithm assigns fixed priorities to tasks based on their relative deadlines, where shorter deadlines receive higher priorities. The implementation maintains these fixed priorities throughout the execution and uses them to select the next task to run.

For EDF, the implementation calculates priorities based on absolute deadlines (task.next_deadline). it's important to note that EDF is actually making scheduling decisions based on these calculated absolute deadlines at each scheduling point.

The Round Robin implementation uses a time-quantum based approach with a quantum size of 1 time unit. To ensure fair rotation among tasks, we maintain a record of each task's last execution time (self.last_execution) and use this information to determine the next task to execute. This prevents any single task from monopolizing the processor while ensuring that each task gets its fair share of processing time.

Experimental Setup

Dataset Description

The experiments were conducted using two main datasets:

- **Task Number Analysis:** 500 tasksets for each task count from 4 to 20 tasks, all with 80% utilization
- **Utilization Analysis:** 500 tasksets with 10 tasks each, for utilizations ranging from 10% to 100% in 10% increments

Analysis Metrics

For each configuration, we calculate several key metrics:

1. **Feasibility Analysis:** - Total tasksets: 500 for each configuration - Infeasible tasksets: Count of tasksets where no algorithm can schedule them - Feasible tasksets: Count of tasksets where at least one algorithm can schedule them - Feasibility ratio = Feasible tasksets / Total tasksets

2. **Algorithm Success Rate:** For each algorithm (DM, EDF, RR), we calculate: - Schedulable count: Number of tasksets the algorithm can schedule - Success rate = Schedulable count / Feasible tasksets

Visualization

We generate four plots to visualize the results:

1. **Feasibility Ratio vs Number of Tasks:** - X-axis: Number of tasks (4-20) - Y-axis: Feasibility ratio (0-1) - Shows how taskset feasibility changes with increasing task count at 80% utilization

2. **Success Rate vs Number of Tasks:** - X-axis: Number of tasks (4-20) - Y-axis: Success rate (0-1) - Three lines showing success rate of each algorithm - Shows how each algorithm's performance changes with increasing task count

3. **Feasibility Ratio vs Utilization:** - X-axis: Utilization percentage (10-100) - Y-axis: Feasibility ratio (0-1) - Shows how taskset feasibility changes with increasing utilization for 10-task sets

4. **Success Rate vs Utilization:** - X-axis: Utilization percentage (10-100) - Y-axis: Success rate (0-1) - Three lines showing success rate of each algorithm - Shows how each algorithm's performance changes with increasing utilization

Results

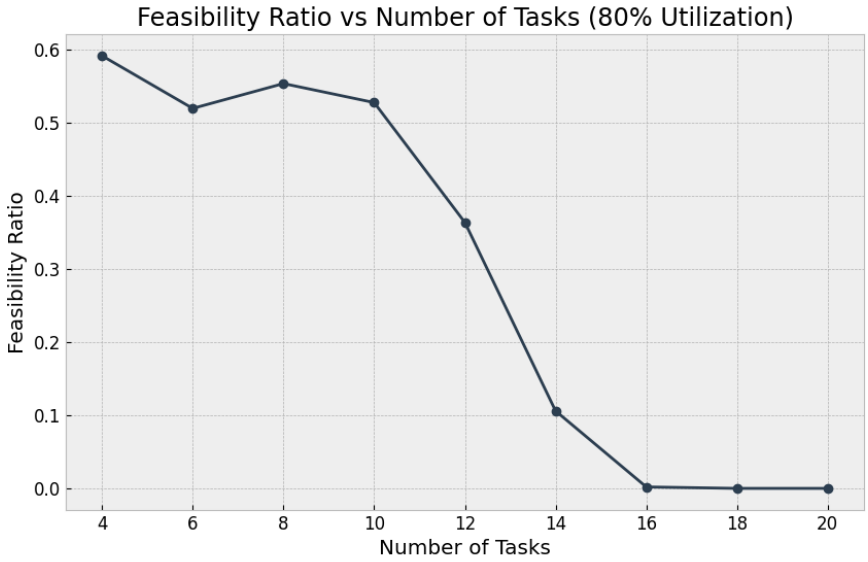

Figure 1: Feasibility ratio versus number of tasks at 80% utilization. The feasibility ratio represents the proportion of task sets that are schedulable by at least one algorithm.

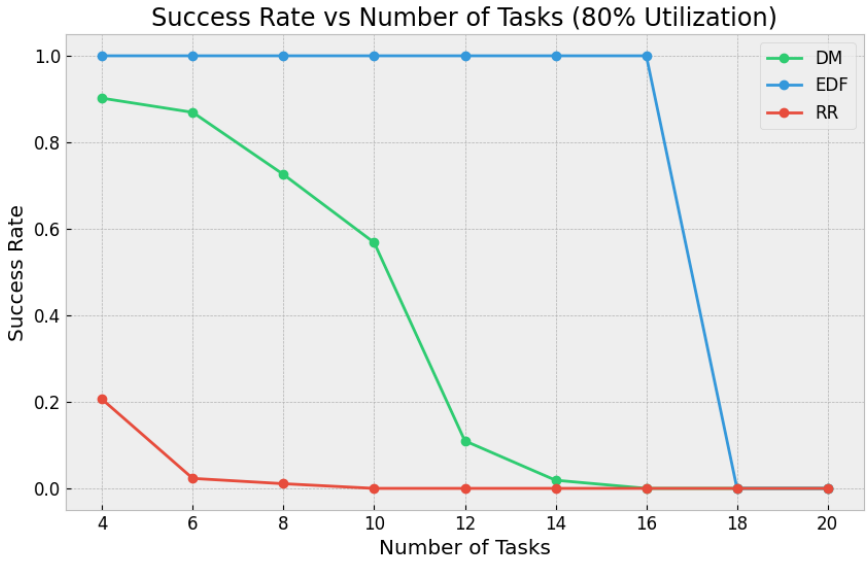

Figure 2: Success rate versus number of tasks at 80% utilization for each scheduling algorithm (DM, EDF, RR). Success rate is calculated as the proportion of feasible task sets that each algorithm can successfully schedule.

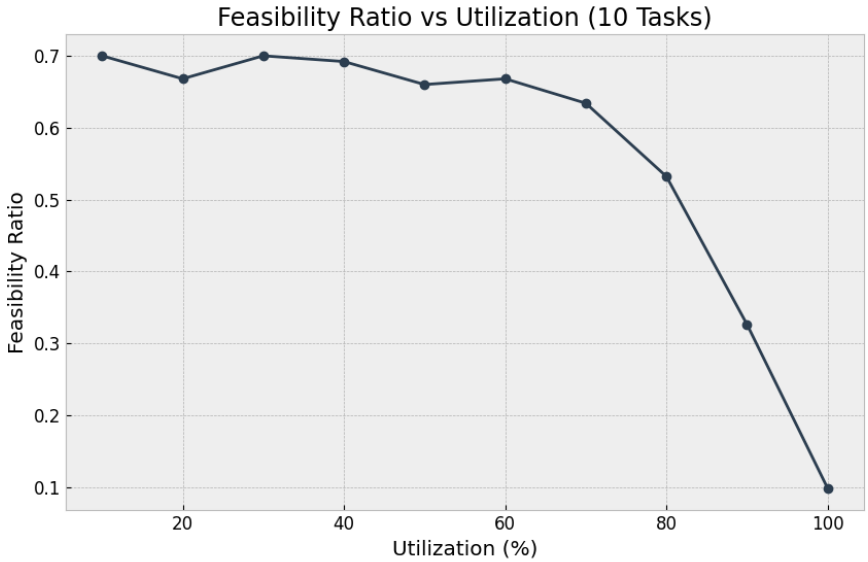

Figure 3: Feasibility ratio versus utilization percentage for task sets with 10 tasks. Shows how the overall schedulability changes as system utilization increases.

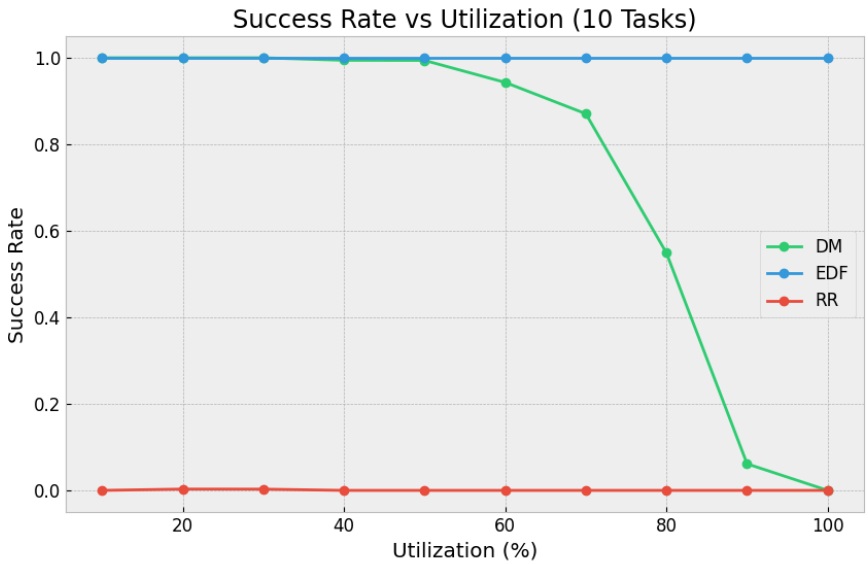

Figure 4: Success rate versus utilization percentage for each scheduling algorithm (DM, EDF, RR) with 10 tasks per set. Demonstrates how each algorithm's performance varies with system load.

Discussion

Analysis of Task Number Impact

The feasibility ratio versus number of tasks plot (Figure 1) reveals a clear downward trend as the number of tasks increases, even while maintaining a constant 80% utilization. The data shows a decline from 59.2% feasible task sets at 4 tasks to 52.0% at 6 tasks, continuing to decrease to 0 at 20 tasks. This decline in feasibility, despite maintaining the same total utilization, occurs because distributing a fixed utilization across more tasks introduces additional complexity to the scheduling problem. When the same 80% utilization is split among more tasks, each task gets a smaller share of processor time, requiring more context switches and creating a more fragmented schedule.

The increased number of tasks also leads to more complex interference patterns and a higher density of scheduling points. With more tasks, there are more release times and deadlines to consider within the same time interval, and each task can be interfered with by more higher-priority tasks. For instance, in a 16-task system, a lower-priority task might need to wait for up to 15 other tasks, compared to only 3 other tasks in a 4-task system.

The success rate versus number of tasks plot (Figure 2) shows distinct patterns for each algorithm at 80% utilization. EDF demonstrates superior performance, maintaining a 100% success for feasible tasksets. This means that whenever a task set is feasible (can be scheduled by any algorithm), EDF is always able to schedule it. This empirically validates EDF's optimality property for deadline-constrained scheduling.

DM shows a clear downward trend as the task count increases. Starting with a success rate close to EDF at 4 tasks (90.2%, scheduling 267 out of 296 feasible sets), its performance gradually declines. The success rate drops to 86.9% at 6 tasks, and continues to decrease more sharply as task count increases, reaching approximately 72.6% at 8 tasks. This decline can be attributed to DM's fixed-priority nature - as the number of tasks increases, there's a higher chance that the fixed priority ordering based on deadlines becomes suboptimal for some task sets.

RR exhibits the poorest performance among the three algorithms, with a dramatic decline in success rate as task count increases. At 4 tasks, RR can only schedule 20.6% of the feasible task sets, indicating that even in relatively simple scenarios, its round-robin approach struggles with real-time constraints. The performance deteriorates rapidly with more tasks, dropping to just 2.3% at 6 tasks and becoming nearly ineffective at 8 tasks with only 1.1% success rate. This severe degradation occurs because RR's time-quantum-based scheduling becomes increasingly inefficient with more tasks - each task must wait longer for its next

quantum as the number of tasks grows, making it harder to meet deadlines even when theoretical schedulability exists.

Analysis of Utilization Impact

The feasibility ratio versus utilization plot (Figure 3) shows how system load affects overall schedulability for 10-task sets. At low utilizations (10-30%), around 70% of task sets are feasible, with about 350 out of 500 task sets being schedulable. This high feasibility ratio remains relatively stable up to 50% utilization. However, as utilization increases beyond 60%, we observe a gradual decline in feasibility. The decline becomes more pronounced at higher utilizations, dropping sharply from 53.2% at 80% utilization to 32.6% at 90% utilization, and finally reaching just 9.8% at 100% utilization. This pattern demonstrates how increasing system load makes it inherently more difficult to find any feasible schedule, regardless of the scheduling algorithm used.

The success rate versus utilization plot (Figure 4) reveals distinct behaviors for each scheduling algorithm. EDF maintains optimal performance across all utilization levels, achieving a 100% success rate for feasible task sets. This means that EDF can successfully schedule all task sets that are schedulable in our tasksets, even at high utilizations. For instance, at 100% utilization, while only 49 task sets are feasible, EDF successfully schedules all of them.

DM performs identically to EDF at low utilizations, successfully scheduling all feasible task sets up to 40% utilization (344/346 at 40%). However, its performance begins to diverge from EDF as utilization increases. At 60% utilization, DM's success rate drops to 94.3% (315/334 feasible sets), and the degradation accelerates at higher utilizations. At 80% utilization, DM can only schedule 54.9% (146/266) of feasible task sets, and at 90% utilization, this drops dramatically to 6.1% (10/163). DM completely fails at 100% utilization, unable to schedule any of the 49 feasible task sets. This degradation pattern reflects DM's increasing difficulty in handling high-utilization scenarios with its fixed-priority approach.

RR shows extremely poor performance across almost all utilization levels. It manages to schedule only a tiny fraction of feasible task sets at very low utilizations, and completely fails to schedule any task sets at utilizations of 40% or higher. This consistent failure demonstrates RR's fundamental unsuitability for real-time scheduling, as its time-quantum-based approach cannot effectively handle the timing constraints of periodic tasks, even at moderate utilization levels.