# Comparison of EDF with RM and DM

Srilalith Nampally
*Dept. of Electrical and Computer Engineering*
*Virginia Tech*
Blacksburg, USA
nsrilalith@vt.edu

Priyatam Annambhotla
*Dept. of Electrical and Computer Engineering*
*Virginia Tech*
Blacksburg, USA
priyatam@vt.edu

***Abstract*—This project aims to compare three real-time scheduling algorithms: Earliest Deadline First (EDF), Rate Monotonic (RM), and Deadline Monotonic (DM). While RM and DM are relatively straightforward to implement, EDF poses more challenges due to its dynamic priority scheduling nature. The project will delve into theoretical analysis, simulation environments, real-time operating system (RTOS) implementations, and evaluation metrics to understand and compare the performance of these algorithms. The hypothesis suggests that EDF will outperform RM and DM in scenarios where dynamic priority scheduling is required. Metrics such as schedulability, processor utilization, deadline miss rate, context switches, and worst-case response time will be used to evaluate the algorithms' performance.**

***Index Terms*—Real-time scheduling algorithms, EDF, RM, DM, RTOS implementation.**

## I. Objective

### A. Background

*1) EDF Scheduling:* Earliest Deadline First (EDF) scheduling is a dynamic priority scheduling algorithm primarily used in real-time operating systems. It's distinguished by its method of assigning priorities based on the deadlines of tasks or processes, where tasks with closer deadlines are given higher priority. The core principle behind EDF is to minimize the risk of missing deadlines by prioritizing tasks in such a way that the one with the earliest deadline is selected for execution first. If multiple tasks share the same deadline, the one with the highest priority is executed. EDF is considered optimal for scheduling a set of periodic tasks, assuming the total task utilization doesn't exceed the system's capacity, making it widely applicable in various real-time domains such as distributed systems, networking, embedded systems, multimedia, and control systems. [1]

*2) RM Scheduling:* Rate Monotonic Scheduling (RM) is a well-known real-time scheduling algorithm utilized in real-time operating systems (RTOS) for managing tasks with fixed or periodic deadlines. It's a priority-based approach where tasks are given priorities according to their periods, with tasks having shorter periods receiving higher priorities. This method assumes that a task's period remains constant, thereby making the assigned priorities fixed and unchanging throughout the execution. RM operates on a preemptive basis, meaning that if a higher-priority task (a task with a shorter period) becomes ready to run, it will preempt the currently running lower-priority task. [4]

*3) DM Scheduling:* Deadline Monotonic (DM) Scheduling is a fixed-priority scheduling algorithm used in real-time systems, where tasks are assigned priorities based on their deadlines. Unlike Rate Monotonic Scheduling, where task priorities are determined by their periodicity, in DM scheduling, tasks with shorter deadlines are given higher priority. This priority assignment is fixed and does not change over time, making DM a static priority scheduling scheme. [4]

### B. Motivation

Although the implementation of Rate Monotonic and Deadline Monotonic Algorithms is relatively straightforward from a programming perspective, the Earliest Deadline First algorithm is a bit more challenging to implement. And the EDF algorithm is more optimal for dynamic based priority scheduling, scheduling many task sets which both the DM and RM scheduling algorithms are unable to.

## II. Experimental Setup

For Implementing the Earliest Deadline First scheduling algorithm, we had to pick a setup which would have real meaning to implement it on. It also needed to be modular so as to enable us to experiment and compare with our implementaion of EDF with existing implementations of RMS and DMS algorithms. After careful thought we decided to build our project using the popular **FreeRTOS** library and implement EDF in C++, since we already have built RMS and RMS with Priority Inheritance and Priority Ceiling Protocol. In addition to that, we have also decided to recycle our implementations of RMS and DMS from project 2, and enhance it to achieve the aforementioned goals. With this we can confidently built an efficient algorithm for EDF, easily switch between scheduling policies, and test our code on an actual real-time system; the MEGA2560 board.

## III. EDF Implementation

### A. Overview of EDF Scheduling

EDF scheduling dynamically prioritizes tasks based on their deadlines—the task with the closest deadline gets the highest priority. This model is ideal for real-time systems where meeting time constraints is critical. In the provided system, EDF is implemented with enhancements to manage tasks

efficiently, potentially with a focus on reducing overhead and handling task states (ready, blocked, etc.) dynamically.

### B. Key Components and Functions

1) Task Control Block (TCB): Each task has a TCB that stores its attributes, such as execution time, deadlines, and priority. For EDF, `xAbsoluteDeadline` is critical as it determines the task's current deadline relative to the system start time.
2) Task Creation and Initialization: `vSchedulerPeriodicTaskCreate`: This function initializes new tasks and adds them to a management list. It sets up the task's periodic attributes, including phase, period, execution time, and deadlines.
3) Priority Management and Task Scheduling: `prvInitEDF`: Initializes the list and priorities based on the EDF principle. This function is crucial at system start-up to prepare the task list for execution. `prvUpdatePrioritiesEDF`: This function dynamically updates priorities of all tasks based on their deadlines to ensure that the scheduler always runs the task with the nearest deadline. This is central to EDF's operation.
4) Task Switching Logic:
   - `prvInsertTCBToReadyList`: Manages task states, particularly handling ready tasks and ensuring they are prioritized correctly.
   - Scheduler Hooks (e.g., `vSchedulerBlockTrace`, `vSchedulerReadyTrace`): These hooks intervene during task state transitions to update scheduler data structures accordingly.
5) Scheduler Task: `prvSchedulerFunction`: This is a dedicated task that runs continuously to manage timing errors and update task priorities based on the scheduling policy.

### C. Priority Handling in EDF

We have implemented two types scheduler EDF implementations has: Naive and Efficient.

*1) **Naive EDF**:* Naive EDF is a straightforward implementation of the EDF scheduling algorithm.

*a) **Key Characteristics**:*

- Priority Update: Priorities of all periodic tasks are updated whenever there is a need to schedule a task. This involves scanning through all tasks, comparing their deadlines, and assigning priorities accordingly.
- Task List: Tasks are managed using a sorted linked list based on their absolute deadlines.
- Priority Assignment: After sorting the list, tasks are assigned priorities in descending order, with the task having the earliest deadline getting the highest priority.
- Overflow Handling: If a task's absolute deadline overflows, it is moved to an overflowed list.
- Task Switch: The highest priority task is selected to run, and a context switch occurs if it is different from the currently running task.

*b) **Implementation**:* In Naive EDF, the following key functions are used:

1) `prvUpdatePrioritiesEDF()`:
   - Updates the priorities of all tasks.
   - Tasks are re-sorted and assigned new priorities.
2) `prvSwapList()`: Handles overflow by swapping the current list with the overflowed list when necessary.
3) `prvSchedulerFunction()`: Contains the logic for the scheduler task, which handles priority updates.

*2) **Efficient EDF**:* Efficient EDF is an optimized implementation of the EDF scheduling algorithm.

*a) **Key Characteristics**:*

- Task Lists: Efficient EDF uses multiple lists to categorize tasks:
  - TCBReadyList - A sorted linked list for all ready periodic tasks.
  - xTCBBlockedList - A list for blocked periodic tasks.
  - xTCBListAll - A list for all periodic tasks, used for administrative purposes.
  - xTCBOverflowedReadyList - A sorted linked list for all ready periodic tasks with overflowed deadlines.
- Priority Assignment: Tasks are assigned a priority based on their absolute deadline, similar to Naive EDF, but the sorting and switching between lists is more efficiently managed.
- Priority Update: Instead of recalculating priorities for all tasks every time a scheduling decision is made, the Efficient EDF implementation only adjusts the necessary tasks.
- Task Switch: The highest priority task is selected to run, but the context switch is minimized to only occur when necessary.

*b) **Implementation**:* In Efficient EDF, the following key functions are used:

1) `prvInsertTCBToReadyList()`:
   - Inserts a task into the appropriate ready list.
   - Handles deadline overflow by placing the task into the appropriate list.
2) `prvSwapList()`: Handles overflow by swapping the current ready list with the overflowed ready list when necessary.
3) `prvSchedulerFunction()`:
   - Contains the logic for the scheduler task.
   - Handles context switching based on task states.

### D. Flow of Functions

The flow of EDF scheduling can be visualized in several stages:

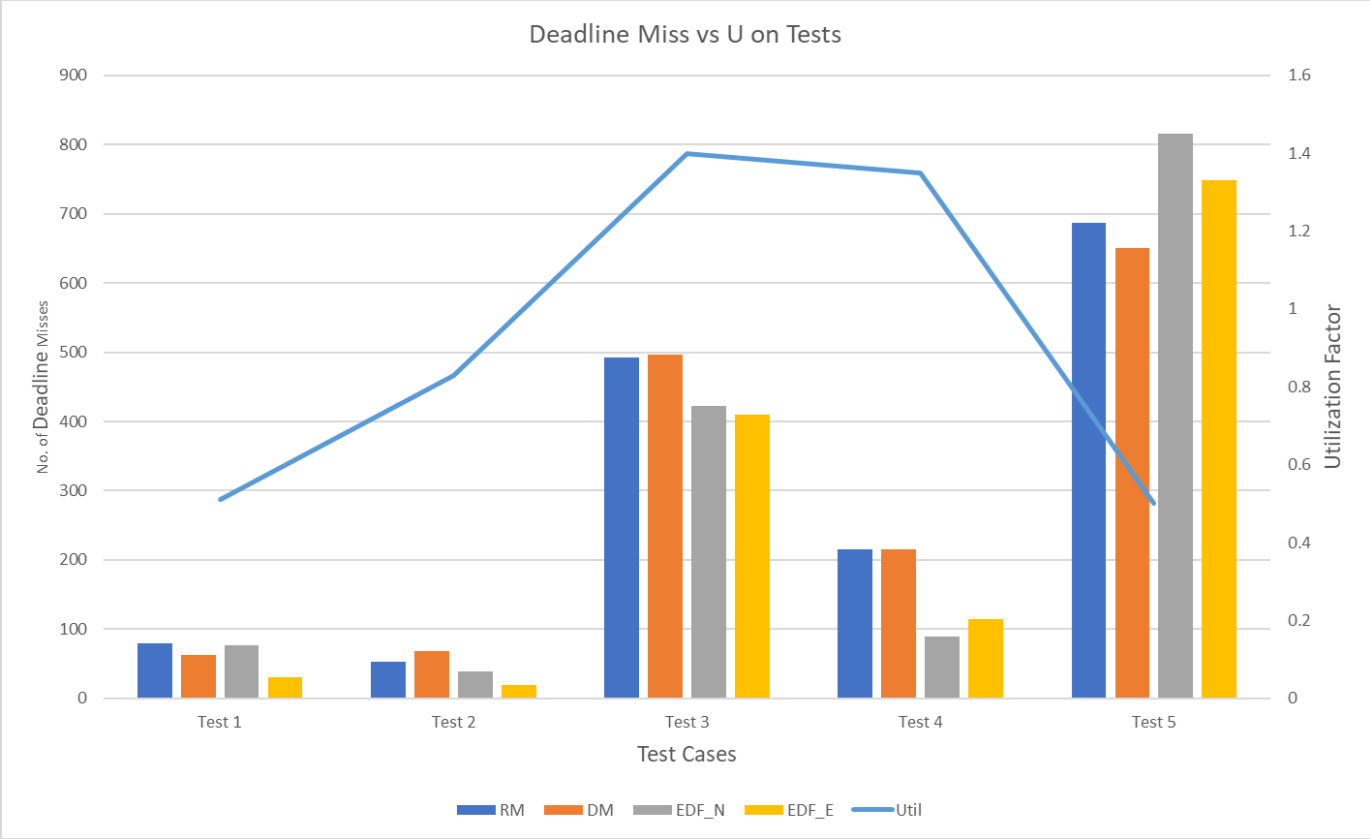


Fig. 1. Number of Deadline Misses for each policy

*a) Initialization Phase:*

- Start: Initialize scheduler, task lists, and system variables.
- Create Tasks: Set up each task's TCB with deadlines and priorities.
- Calculate Priorities: Based on absolute deadlines.
- Scheduler Start: Begin task execution based on EDF logic.

*b) Runtime Phase:*

- Task Execution: Execute tasks based on scheduler decisions.
- Priority Update: Adjust priorities as time progresses or when tasks complete/block/suspend.
- Deadline Management: Check and handle missed deadlines or execution overruns.

### E. Important Functions in Action

- `vSchedulerStart`: Boots the scheduler, sets initial conditions, and starts task execution based on prepared lists and priorities.
- `prvSchedulerFunction`: Monitors and adjusts task priorities, handling state transitions and checking for timing errors.

In our implementation, ensuring that the `prvUpdatePrioritiesEDF` function and related deadline management strategies are correctly implemented is crucial. These components are responsible for the dynamic nature of EDF and its ability to handle real-time task prioritization effectively.

## IV. Results

### A. Setup

When working with Real-Time systems it is important to consider the importance of "time" itself, and ideally Real-Time Systems will never have a down time, and will continue to run until an interrupt is given through software or hardware means. But in order to effectively evaluate and compare results across 3 different algorithms, we need to limit the execution time. For all results we have limited the total execution times of all runs to 1 minute. And no parameter inside a task set exceeded 5000ms, meaning there is plenty of time for even 4 tasks to complete each job at least once.

### B. Tasksets

We have used several tasksets which will provide meaningful results to our comparison experimentation. We tried to use tasksets which are in theory more favourable to certain policies than the others.

### C. Metrics and Evaluation

We have included some helper functions within the scheduler.cpp file of FreeRTOS library, which will print out the

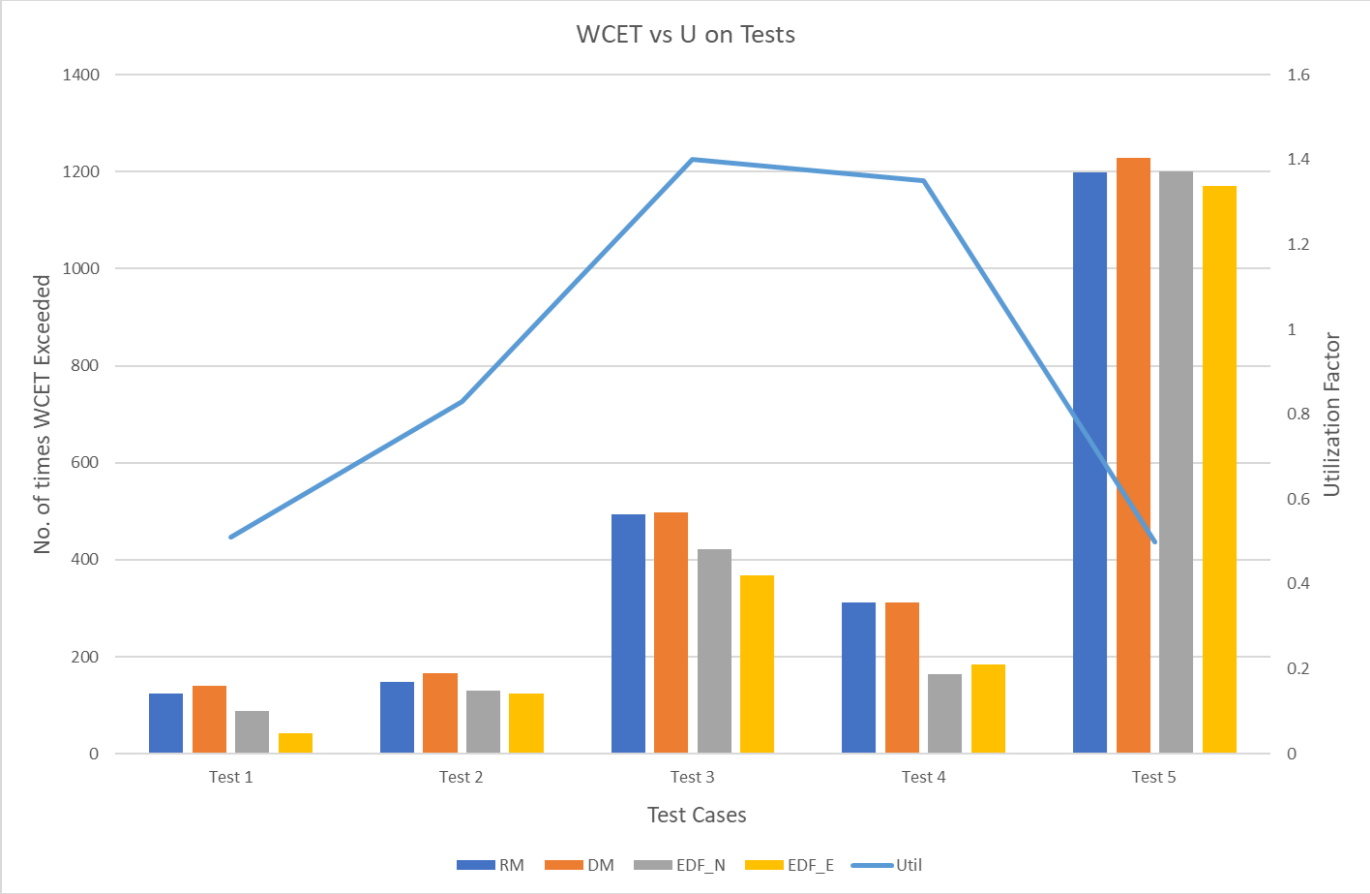


Fig. 2. Number of Deadline Misses for each policy

following metrics for each task once we force interrupt the scheduler to stop after 1 minute time:

- Task Name
- Number of Deadlines missed by the task
- Number of times the Task has exceeded its maxExecution time

With these results it would give insight to each policy's performance when they are plotted against the processor utilization count for each task set.

### *D. Plots Inference*

*1) Legend:* The Figure 1 aims to visualize the changes in total number of times all tasks of a task set miss their deadline, across each policy.

The Figure 2 aims to visualize the changes in total total number of times all tasks of a task set have exceeded their worst case execution time.

In both the plots the blue line is used to indicate the Utilization factor of that task set.

*2) Evaluation:* From Figures 1 and 2, we can clearly see that in most of the cases the EDF-Naive and EDF-Efficient policies perform relatively same and much better than the RM and DM policies. In the final task set however, it might appear that they are performing slightly worse. But, we specifically crafted a task set with selective deadlines and hyper periods so that the RM and DM perform better than EDF.

Our results clearly matched our expectations before starting this project.

## V. Conclusion

In this project, we have implemented and compared three real-time scheduling algorithms: Earliest Deadline First (EDF), Rate Monotonic (RM), and Deadline Monotonic (DM). Each algorithm offers unique advantages and challenges, with EDF generally excelling in dynamic priority scheduling scenarios. Through theoretical analysis, simulation, and real-time operating system (RTOS) implementations, we evaluated the performance of these algorithms using key metrics such as schedulability, processor utilization, deadline miss rate, context switches, and worst-case response time.

The findings reveal that:

1) EDF:
   - Offers optimal performance in scenarios with varying task periods and dynamic priority requirements.
   - Exhibited lower deadline miss rates and better processor utilization in most scenarios, aligning with its theoretical advantages.

2) RM:

- Provides simplicity and predictability through fixed priority scheduling.
- Performed well in scenarios with consistent task periods but struggled with varied task periods.

3) DM:
   - Similar to RM but prioritizes tasks based on deadlines, making it effective for applications where deadlines are critical but periods may vary.
   - Generally performed better than RM but fell short of EDF in dynamic scenarios.

Overall, the experiments validated the hypothesis that EDF would outperform RM and DM in dynamic scheduling scenarios, though the choice of algorithm ultimately depends on the specific requirements and constraints of the application.

## VI. Future Work

Future work in this domain could focus on several areas to further enhance and evaluate real-time scheduling algorithms:

1) Algorithm Enhancements:
   - Develop and evaluate hybrid scheduling algorithms that combine the strengths of multiple algorithms.
   - Investigate the integration of additional scheduling parameters, such as jitter and energy consumption.
2) Simulation and Benchmarking:
   - Expand testing to more complex and diverse task sets, including sporadic tasks and mixed criticality systems.
   - Utilize benchmarking suites to compare algorithms against industry-standard workloads.
3) RTOS and Embedded Systems:
   - Implement the scheduling algorithms in different RTOS environments to assess portability and performance.
   - Explore the use of real-time scheduling in power-constrained and safety-critical embedded systems.
4) Real-World Applications:
   - Apply the scheduling algorithms to practical real-time applications, such as robotics, automotive systems, and multimedia processing.
   - Evaluate the impact of scheduling on application-specific metrics, such as latency, throughput, and quality of service.

By pursuing these future directions, we can deepen our understanding of real-time scheduling and develop more robust and efficient algorithms for emerging real-time systems.

## References


[1] F. Zhang and A. Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling," in IEEE Transactions on Computers, vol. 58, no. 9, pp. 1250-1258, Sept. 2009.

[2] C. Lu, J. A. Stankovic, G. Tao and S. H. Son, "Design and evaluation of a feedback control EDF scheduling algorithm," Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054), Phoenix, AZ, USA, 1999, pp. 56-67.

[3] V. Prajapati, A. Shah and P. Balani, "Design of new scheduling algorithm LLF_DM and its comparison with existing EDF, LLF, and DM algorithms for periodic tasks," 2013 International Conference on Intelligent Systems and Signal Processing (ISSP), Vallabh Vidyanagar, India, 2013, pp. 42-46.

[4] Hemang Thakar, "Comparison between EDF_RM and EDF_DM in dynamic scheduling algorithm with sporadic task", May 2016 IJSDR, Volume 1, Issue 5

[5] Buttazzo, G.C. (2011). Dynamic Priority Servers. In: Hard Real-Time Computing Systems. Real-Time Systems Series, vol 24. Springer, Boston, MA. https://doi.org/10.1007/978-1-4614-0676-1_6

[6] https://www.freertos.org/index.html