IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Comparitive Evalution of Priority and Deadline based Scheduling in RTOS

1st Rahul Rawat
M.Tech Student, Dept. of ECE
CVR College of Engineering
Hyderabad

2nd *Dr. Dhruva R. Rinku*
Dept. of ECE
CVR College of Engineering
Hyderabad,India

Abstract— This paper presents a comparative analysis of two widely used real-time scheduling algorithms—Earliest Deadline First (EDF) and Rate Monotonic (RM)—with the objective of evaluating their performance and suitability for real-time embedded applications on FreeRTOS. To simulate realistic execution scenarios, synthetic task sets were generated. The performance of each scheduler was assessed based on key metrics such as average waiting time, number of context switches, and CPU utilization.

The results indicate that EDF consistently outperforms RM in terms of meeting deadlines, especially under high system loads. EDF also tends to reduce average waiting time; however, it incurs a higher number of context switches. In contrast, RM scheduling leads to increased average wait times but generally results in fewer context switches, which can be advantageous for power-sensitive or resource-constrained systems.

Interestingly, for periodic tasks where deadlines are equal to their periods, the performance gap between EDF and RM narrows. In such cases, the efficiency of the scheduling strategy becomes more dependent on the total number of tasks being managed.

Overall, the study provides valuable insights to embedded system designers, enabling them to make more informed decisions when selecting the appropriate scheduling strategy based on application-specific requirements.

Keywords— *FreeRTOS, Kernel, EDF, RM, Scheduler*

I. INTRODUCTION

Real-time systems play a critical role in various domains such as aerospace, automotive, medical devices, and industrial automation, where timely and predictable task execution is fundamental to ensuring system reliability, safety, and performance. In these systems, meeting task deadlines is not optional—it is a core functional requirement, especially in hard real-time systems that demand deterministic behavior to avoid catastrophic failures [1].

Determinism refers to a system's ability to exhibit predictable and consistent behavior, particularly in terms of timing and response. In hard real-time applications, determinism ensures that tasks execute within well-defined time bounds, thus enabling reliable scheduling and accurate performance estimation.

Given the constraints of embedded platforms—such as limited memory and processing power—a dedicated Real-Time Operating System (RTOS) is essential. RTOSs provide a lightweight, task-oriented environment with fine-grained control over scheduling, resource management, and system responsiveness. In contrast, General-Purpose Operating Systems (GPOS) prioritize fairness and multitasking by allocating processor time equitably among processes, making them less suitable for real-time applications [2].

At the heart of any operating system lies the scheduler, which determines the sequence and timing of task execution [3]–[6]. The scheduler's primary function in real-time systems is to allocate computational resources efficiently, ensuring that critical tasks meet their timing constraints. Goals such as minimizing latency, maximizing throughput, maintaining fairness, and respecting task priorities often conflict, requiring the scheduler to find a balance based on application-specific requirements [7]–[9].

Schedulers for real-time systems assign priorities based on timing characteristics like task periods, deadlines, and response times. Traditional scheduling algorithms used in generic operating systems, such as Cooperative Scheduling and First-Come-First-Served (FCFS), do not guarantee deadline adherence. In contrast, Rate Monotonic (RM) and Earliest Deadline First (EDF) are well-established algorithms designed to support real-time guarantees on uniprocessor systems [3], [5], [7]–[10].

This paper presents a comparative evaluation of EDF and RM scheduling strategies implemented within the FreeRTOS framework. FreeRTOS was chosen due to its open-source nature, portability, and support for real-time features such as task management, synchronization, and memory allocation [14]. Its Windows port enables simulation of scheduling behavior in a controlled environment, and its modular design allows easy extension and customization.

Although EDF is theoretically optimal for uniprocessor task scheduling, it is less commonly used in resource-constrained embedded systems due to perceived overheads. However, recent studies have shown that EDF can be efficiently implemented even on low-end microcontrollers, particularly when task models and deadlines are well-structured [15].

In this study, synthetic task sets were developed to emulate realistic workloads, enabling controlled and reproducible testing. The objective is to analyze the performance of EDF and RM scheduling using key metrics such as average wait time, context switch count, CPU utilization, and deadline miss ratio. The findings aim to guide embedded system designers in selecting the most appropriate scheduling algorithm for their specific real-time application needs.

II. IMPLEMENTATION

2.1 *Earliest Deadline First (EDF)*

2.1.1 – *user level implementation*

A significant contribution of this work is the implementation of the Earliest Deadline First (EDF) scheduling algorithm within the FreeRTOS environment. The objective is to evaluate the feasibility and effectiveness of EDF in meeting task deadlines while optimizing performance metrics such as average waiting time and CPU utilization.

Two approaches were employed to integrate the EDF scheduler into FreeRTOS. The first approach involves implementing EDF at the user level, using standard FreeRTOS APIs to manually control task prioritization based on deadlines. The second approach entails modifying the FreeRTOS kernel itself, by introducing new data structures and function prototypes to support native EDF scheduling. This kernel-level modification allows EDF to operate as a core scheduling mechanism within the RTOS.

The logic for the EDF implementation is illustrated through a flowchart shown in Fig. 1, which outlines the decision-making process used to assign and manage task priorities based on their respective deadlines.

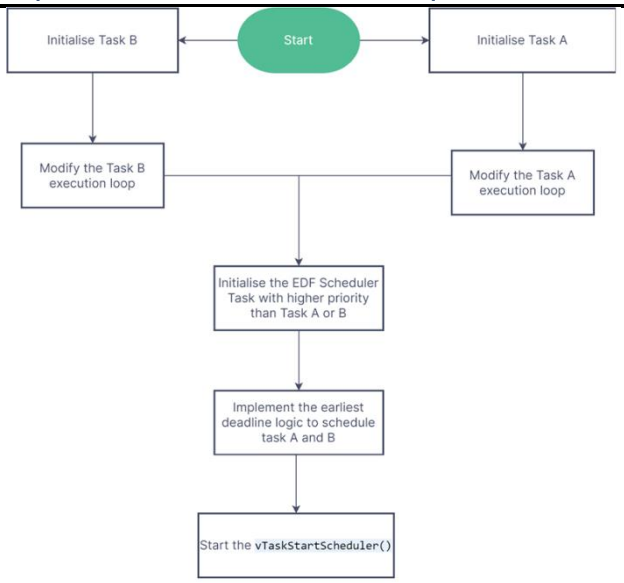

Figure 1. Implementation of EDF Scheduling for Tasks A and B Maintaining the Integrity of the Specifications

2.1.2 *EDF – kernel level implementation*

In contrast to the user-level implementation of the EDF scheduler, the kernel-level integration necessitates structural modifications within the FreeRTOS kernel. The primary objective is to construct a customized Ready List capable of supporting dynamic priority scheduling based on task deadlines. This new Ready List maintains tasks in ascending order of their absolute deadlines, such that the task at the top of the list—i.e., with the earliest deadline—receives the highest scheduling priority.

The remainder of the FreeRTOS architecture, including components such as the Waiting List and system clock mechanism, remains largely intact with only minimal adjustments. All kernel-level changes are encapsulated within the task.c file, and are conditionally compiled using the configuration macro configUSE_EDF_SCHEDULER. When this macro is set to 1, the EDF scheduling functionality is enabled; otherwise, the system defaults to the standard FreeRTOS scheduler.

Prior to adding a task to the Ready List, its absolute deadline must be computed based on the current system tick and the task's relative deadline. Additionally, a context switch is triggered each time a new task is inserted into the Ready List, ensuring that the task with the earliest deadline is always selected for execution.

The EDF scheduling algorithm demonstrates optimal performance under the following assumptions:

Periodic Task Releases: All task requests occur at fixed intervals, with each task having a known and constant period.

Deadline Constraints: Deadlines serve as hard constraints, requiring each task to complete execution before the next activation of the same task.

Task Independence: Each task is self-contained, meaning that its execution is independent of the state, execution, or completion of other tasks in the system.

Constant Execution Time: Each task has a fixed worst-case execution time (WCET), which remains consistent across all instances.

These assumptions help maintain the predictability and schedulability of the system, allowing the EDF algorithm to provide optimal CPU utilization and guarantee deadline adherence in real-time applications.

2.2. RATE MONOTONIC SCHEDULING (RMS)

Rate Monotonic (RM) is a widely adopted pre-emptive, fixed-priority scheduling algorithm used in real-time systems to schedule periodic tasks. In RM scheduling, priorities are statically assigned to tasks based solely on their periods: tasks with shorter periods are given higher priorities, while those with longer periods are assigned lower priorities. This deterministic nature makes RM both analyzable and predictable, making it ideal for safety-critical applications.

RM scheduling assumes a set of independent periodic tasks, each defined by its period (i.e., the time between successive releases) and worst-case execution time (WCET). These tasks are expected to execute repeatedly, maintaining the same timing parameters throughout system operation. The fundamental scheduling principle in RM ensures that when multiple tasks are ready, the one with the highest priority (shortest period) pre-empts the others and executes first. This pre-emptive behavior allows the system to respond quickly to high-frequency tasks.

In this work, the RM scheduling algorithm was implemented at the user level using FreeRTOS primitives. The following pseudocode outlines the core logic of the RM scheduler:

Pseudocode: Rate Monotonic Scheduling
Initialize System Parameters:
Define the number of tasks NUM_TASKS.
Assign periods and capacities (WCETs) to each task.
Calculate Hyperperiod:
Compute the Least Common Multiple (LCM) of all task periods to determine the hyperperiod of the task set.
Sort Tasks by Period:
Apply Rate Monotonic logic: order tasks in ascending order of periods.
Assign Priorities:
Create task handles.
Assign priorities inversely proportional to the periods (shortest period → highest priority).
Define Task Function (TaskFunction):
Simulate task execution within the real-time environment.
Task Execution Loop:
For each task:
Retrieve task index, period, and capacity.
Initialize xLastWakeTime with the current system tick.
Loop indefinitely:
Print tick count, task ID, and remaining execution capacity.
Decrement capacity counter.
If capacity reaches zero, reset it to its initial value.
Use vTaskDelayUntil() to delay until the start of the next period.
Start Scheduler:
Initialize and create all tasks using FreeRTOS APIs.
Start the scheduler with vTaskStartScheduler().
Main Program Loop:
The application enters a continuous execution state, simulating real-time behavior indefinitely.

The RM implementation provides a benchmark for comparison with dynamic scheduling strategies like Earliest Deadline First (EDF). While RM offers simplicity and predictability, its schedulability is limited by Liu and Layland's utilization bound, which may result in suboptimal processor usage in certain task sets.

III RESULTS AND DISCUSSION

3.1 *EDF Implementation in FreeRTOS Results*

To validate the developed EDF scheduler, we run two tasks with known EDF scheduling sequences and compare the run-time scheduling sequence to the expected one.

Consider two Task 1 and Task 2 with the following parameters as seen in Table 1.

Table 1. Task Parameters for testing EDF Kernel Implementation

Task Name	Time Period	Capacity
Task 1	5	3
Task 2	8	3

EDF guarantees the tasks are schedulable if the following condition is satisfied:

$$\Sigma(Ci / Ti) \leq 1 \quad (1)$$

Where Σ denotes summation, Ci is the capacity (or) more specifically, worst-case execution time (WCET) of task `i`, and Ti is the task period. The proof involves the concept of demand bound function (DBF). DBF is the cumulative demand imposed by the tasks on the system's resources within a given interval. For EDF scheduling, the DBF can be calculated as follows:

$$DBF(t) = \Sigma(Ci * ceil(t / Ti)) \quad (2)$$

where ceil() is the ceiling function and `t` represents the time duration.

The set of tasks are guaranteed to be schedulable, if the DBF never exceeds the available resources, i.e., $DBF(t) \leq t$ for all t. By using the above inequality and applying the EDF scheduling algorithm, it can be proven that the tasks will meet their deadlines and the system will be schedulable.

According to the utilization formula as in equation (1), for the tasks defined in the table 1, the utilization factor for this task set is calculated to be:

$$\text{Utilization Factor} = 3/5 + 3/8 = 0.6+0.375 = 0.975 \quad (3)$$

The hyper period is defined as the least common multiple (LCM) of the periods of all periodic tasks in the task set. It represents the interval after which all periodic tasks simultaneously return to their initial states, effectively repeating their execution patterns.

The concept of the hyperperiod is crucial in EDF scheduling because it defines the maximum scheduling interval over which the entire set of tasks completes a full cycle of executions. Denoted by *H*

H, the hyperperiod establishes the timeframe within which the EDF scheduler guarantees that each task meets its deadline at least once. The scheduler then repeats this process for every hyperperiod, ensuring all jobs are completed timely and predictably across the entire execution timeline.

$$H = LCM(8,5) = 40 \quad (4)$$

Since the Utilization factor was 0.975 as in equation (3), the tasks are guaranteed to be schedulable. Over the hyper period of 40, there are multiple instances of pre-emption. At t=0, the deadline of task 1 is earlier than the deadline of task B and hence, task 1 gets the higher priority and is executed. At t=3, task 1 has been executed for its entirety and task B can start execution. Task 1 now only enters the Running state after t=5. Even at t=5, task 2 has the higher priority because its deadline is at t=8, compared to t=10 for the second instance of task 1. Task 1 is scheduled to run at t=6 and completes execution at t=9. At t=9, task B runs for 1 unit of capacity and then gets pre-empted by task 1. This is because the deadline of

task 1 is t=15 while for task B its t=16. After task 1 executed from t=10 to t=13 and now the remaining capacity of the task 2 is scheduled to run from t=13 to t=15. Another instance of pre-emption is seen at t=25, as seen in figure 2.

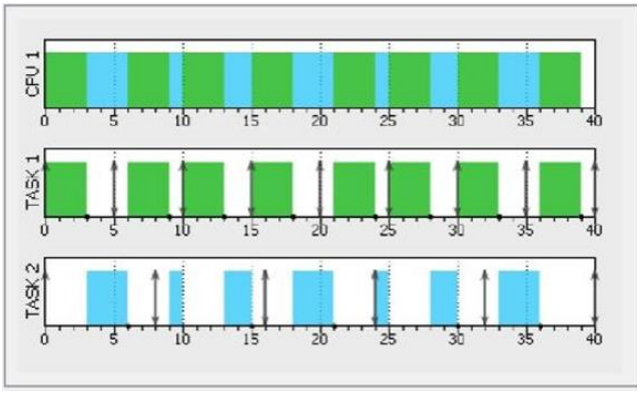

Figure 2: Gantt Chart created from the schedule data returned by the EDF Kernel Implementation

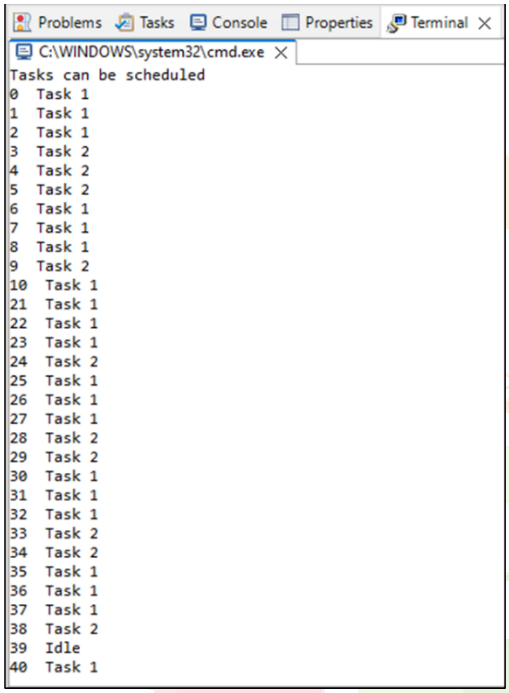

Figure 3. Output of the test application for the EDF implementation in FreeRTOS

III ANALYSIS

Both EDF and RM implementation were tested for number of iterations ranging from one to ten.
Synthetic data have been taken as per table 3.

Table 3 :Task Parameters for testing EDF and RM Implementation

Task Name	Arrival Time	Execution Time	Deadline
T1	0	12	33
T2	4	2	28
T3	9	10	29
T4	16	5	29

research results, Arrival Time is defined as the instant when the task state is ready. Execution Time defines the period for which the task must run to completely executed. Deadline is the absolute instant of time before which the task must be executed completely. This paper assumes Deadline is equal to Periodicity for all the tasks.

From Figure 4(a) and 4(b), it is evident that EDF is better for this particular task set because the context switches are lower, and the waiting time is also less when compared to RM. Waiting Time is defined as the duration between the task arrival time and executing instant.

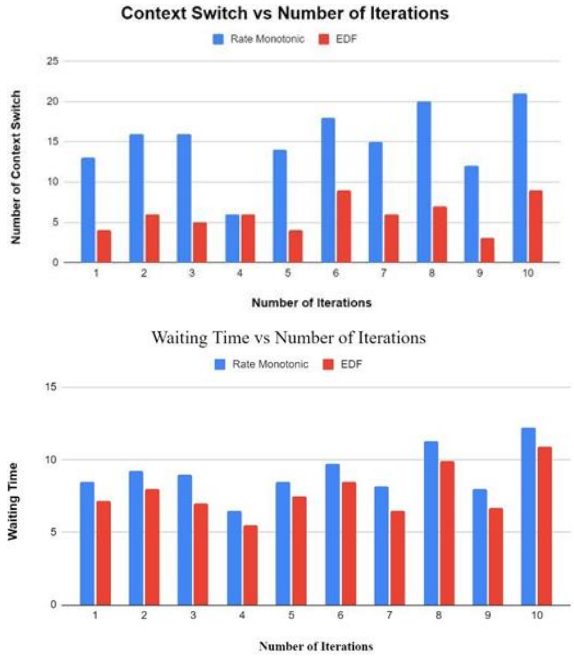

Figure 4. Comparison between RM vs EDF for the task parameters mentioned in Table 3. (a) Compares Number of context switches vs iterations (b) Compares Waiting Time vs iterations

CONCLUSION

This paper conducted a thorough comparison of two key real-time scheduling algorithms, Earliest Deadline First (EDF) and Rate Monotonic (RM), within the FreeRTOS environment. The effectiveness and applicability of these algorithms for real-time applications were determined using synthetic task set and performance was analyzed. The basic yet crucial implementations of EDF at the kernel level and RM at the user level in FreeRTOS represent valuable additions to the real-time embedded systems toolkit. They cater to specific needs and simplify task management, but it is imperative to recognize their inherent limitations and the need for further development to handle more intricate real-world scenarios. These implementations serve as stepping stones towards building more robust and capable real-time systems. The kernel-level integration of EDF, while powerful, can introduce complexity to the system, potentially impacting resource usage and context switching overhead. RM scheduling at the user level, while lightweight, lacks the centralized control that kernel-level scheduling offers. Furthermore, both implementations are rudimentary in their current form and may not cover the full spectrum of features found in dedicated real-time operating systems. They may not address advanced scenarios involving resource sharing, synchronization, or dynamic task creation, which are critical in complex real-time systems.

REFERENCES

[1] L. B. Das, Embedded Systems: An Integrated Approach. 2013.
[2] A. B. Tucker, Computer Science Handbook, Second Edition. 2004.
[3] C. L. Liu and W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”, Journal of the ACM, vol. 20, number 1,pp-46-61, January 1973.

[4] Dhruva R. Rinku, Dr. M. Asha Rani "Evaluation of Scheduling Algorithms on Linux OS"-ICETE 2019, LAIS 4, pp.210 – 217, 2020.https://doi.org/10.1007/978-3-030-24318-0_25.(Springer Book).

[5] J. Cho, Ravindran, "An optimal real-time scheduling algorithm for multiprocessors," IEEE Computer Society, 2007.

[6] Dhruva R. Rinku, Dr. M. AshaRani,Y. Krishna Suhruth "Exploring the Scheduling Techniques for the RTOS" ICT Infrastructure and Computing, Lecture Notes in Networks and Systems 520,pp-11-18. https://doi.org/10.1007/978-981-19-5331-6_2

[7] Y. Oh and S. H. Son, "Preemptive Scheduling of Periodic Tasks on Multiprocessor: Dynamic Algorithms and Their Performance", Tech. Report CS-93-26 Univ. Of Virginia. CS Dept. May 1993.

[8] Dhruva R. Rinku, Dr. M. AshaRani "Reinforcement Learning Based Multi Core Scheduling (RLBMCS)For Real Time Systems", IJECE, Vol 10, Issue 2 April 2020.pages:1805 -1813.

[9] Dhruva R. Rinku, M. Asha Rani, Y. Krishna Suhruth "RTOS schedulers for periodic and aperiodic taskset" Lecture Notes in Networks and Systems 765, pp. 247-257, https://doi.org/10.1007/978-981-99-5652-4

[10] Jiwen Dong, Yang Zhang "A modified Rate Monotonic algorithm for scheduling periodic tasks with different importance in Embedded Systems" International Conference on Electronic Measurement & Instruments, IEEE Xplore,pp.4-606 - 4-609, September 2009.

[11] T. P. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis", IEEE Real-Time Systems Symposium, Dec, 2003.

[12] S. Baruah, "Robustness Results Concerning EDF Scheduling upon Uniform Multiprocessor", Euromicro Conf. on Real-Time Systems 2002.

[13] T. P. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis", IEEE Real-Time Systems Symposium, Dec, 2003.

[14] F. ltd., "Freertos official website." http://www.freertos.org/RTOS.html, Feb. 2016.

[15] Oliveira, G., & Lima, G. (2020). Evaluation of Scheduling Algorithms for Embedded FreeRTOS-based Systems. Brazilian Symposium on Computing System Engineering, SBESC,2020-November. https://doi.org/10.1109/SBESC51047.2020.9277851