Python Scheduling for RMS, LLF, EDF, and DMS

GUI Implementation

Tessa Morgan (458) and Mara Prochaska (458)
CPR E 458//558 Real-Time Systems, Fall 2023
Department of Electrical and Computer Engineering
Iowa State University

Abstract

This paper presents the design and implementation of a graphical user interface (GUI) in Python for simulating and visualizing the performance of several real-time scheduling algorithms, including Rate-Monotonic Scheduling (RMS), Earliest Deadline First (EDF), Least Laxity First (LLF), and Deadline Monotonic Scheduling (DMS). Real-time systems often require deterministic and predictable behavior to meet stringent timing constraints, making efficient scheduling algorithms essential. The GUI provides an intuitive platform for users to interact with and visualize how tasks are scheduled under each algorithm. Key features of the GUI include task parameter inputs, such as task arrival times, deadlines, and periods, as well as visual representations of the task execution timeline and scheduling decisions. Users can observe the behavior of different scheduling strategies in various system configurations, compare their performance, and identify scenarios where each algorithm excels or fails. The Python implementation leverages libraries such as Tkinter for the GUI, along with custom scheduling algorithm modules, to offer real-time task simulation. This approach enables a hands-on understanding of scheduling theory and allows users to experiment with and analyze the impact of different scheduling policies on system performance. The paper discusses the implementation details, challenges encountered during development, and the overall effectiveness of the GUI as an educational and research tool for real-time systems. Additionally, the paper highlights potential extensions, such as supporting dynamic task sets and integrating real-time operating system (RTOS) features for more complex simulations.

1. Introduction

Real time systems are a key component of many fast-moving technologies including hard applications such as airplanes all the way to soft applications such as banking. All real time systems are designed around scheduling algorithms that are optimized for specific conditions to ensure all essential tasks are processed by their deadlines.

Developers or students without extensive real time system knowledge may not be able to determine which algorithms would be most effective for projects they are developing. Therefore, our project is intended to create an application that allows users to easily put in periodic tasks and choose an algorithm to display. If the tasks are schedulable by the chosen algorithm, the task schedule will be displayed.

Our project will implement the Rate-Monotonic Scheduling (RMS), Earliest Deadline First (EDF), Least Laxity First (LLF), and Deadline Monotonic Scheduling (DMS) algorithms. These algorithms are ideal for periodic tasks and may have different strengths depending on the number of input tasks and their costs. By using our tool, developers and students will be able to more easily determine which algorithm will be most effective for the tasks correlating to their real time systems projects.

2. Project Objectives & Scope

The objective of this project is to develop a tool that helps developers and students visualize and understand the performance of various real-time scheduling algorithms. Specifically, we aim to implement the RMS, EDF, LLF, and DMS algorithms, which are commonly used in real-time systems for managing periodic tasks. The tool will allow users to input tasks defined by their period, execution cost, and optionally deadlines, select a scheduling algorithm, and visualize the resulting task schedule, provided the tasks are schedulable by the chosen algorithm.

The problem we are addressing lies in the difficulty that developers, especially those without extensive experience in real-time systems, face when selecting the most appropriate scheduling algorithm for a given set of tasks. Since real-time systems are designed to ensure that all essential tasks meet their deadlines, choosing the right scheduling algorithm can significantly impact the system's performance. This is especially true when there are varying task characteristics such as different periods, execution costs, and deadlines.

Our tool aims to bridge this knowledge gap by providing a user-friendly graphical interface that simplifies the process of comparing and understanding the performance of different algorithms. By inputting different task sets and visualizing the task scheduling behavior, users will gain insight into which algorithm best meets their project's needs. This tool will be beneficial not only for educational purposes but also as a practical resource for real-time system development, enabling users to make more informed decisions about task scheduling in their projects.

2.1. System Model

The problem of task scheduling in real-time systems appears in a wide range of real-world applications, particularly in systems where tasks must be executed within strict timing constraints. These applications span various industries, from safety-critical embedded systems in aerospace, automotive, and medical devices to soft real-time systems used in financial transactions and telecommunications.

Conceptually, this problem resides primarily within embedded systems. These systems are typically characterized by tightly integrated hardware and software components that work together to perform specific functions. In embedded systems, task scheduling is a critical element because the system must ensure that each task is completed by its deadline, which could be crucial for the overall safety, reliability, or performance of the system. For instance, in an aircraft's flight control system, real-time scheduling ensures that sensor data is processed on time to adjust control surfaces or make navigation decisions.

Within the context of this project, the problem is more directly related to embedded systems, where all the components—task generators, schedulers, and processors—are part of the same circuit or hardware architecture. Tasks are typically periodic, with well-defined periods and execution costs, and the system must decide how to allocate processor time to each task to meet the timing constraints.

However, this problem also appears in more distributed environments, such as the Internet of Things (IoT), where multiple devices (e.g., smart sensors, actuators, and control systems) communicate over a network and may rely on real-time scheduling to ensure that critical tasks like data transmission, processing, and control actions are completed on time. In such systems, real-time scheduling algorithms can help manage how tasks are assigned across devices, and how timing constraints are met in a distributed context.

In both embedded systems and IoT environments, the fundamental challenge remains the same: ensuring that all tasks are executed within their deadlines. Our project focuses on providing a tool to simulate and visualize how various scheduling algorithms perform in these environments, offering developers and students the means to evaluate which algorithm is best suited to their specific real-time system's needs. This project's scope is primarily limited to the scheduling of periodic tasks within a single system (either embedded or distributed), but the insights gained can be applied to broader contexts, including more complex IoT or multi-device scenarios.

2.2. Problem Statement

Real-time systems, which are critical for applications such as aerospace, automotive safety, medical devices, and telecommunications, rely on scheduling algorithms to ensure that tasks meet their deadlines. However, selecting the most appropriate scheduling algorithm for a given set of tasks can be challenging, especially for developers or students with limited experience in real-time systems. The problem arises from the fact that different scheduling algorithms—such as RMS, EDF, LLF, and DMS—perform differently based on task characteristics like period, execution cost, and deadline. Without a tool to visualize and compare these algorithms in action, it is difficult for users to understand which algorithm will best meet their system's needs.

This project aims to address this issue by creating an intuitive application that allows users to input periodic tasks (defined by their period, cost, and deadline), select a scheduling algorithm, and visualize the resulting task schedule. The goal is to help users determine if their tasks can be scheduled by the chosen algorithm, and if so, display the task schedule. By doing so, the project seeks to simplify the task selection process, educate users about real-time scheduling algorithms, and provide insights into their performance under various conditions. Ultimately, the tool will empower developers and students to make more informed decisions about task scheduling in real-time systems.

By allowing users to interact with the scheduling algorithms through the GUI, they will gain practical experience and a clearer understanding of the real-time scheduling process. This tool is designed not just as a scheduler but as an educational aid, enabling students and developers to experiment with different task sets, visualize the outcomes, and learn which algorithms are best suited to specific real-time system scenarios.

2.3. Objectives and Scope

The primary goal of this project is to develop a user-friendly Graphical User Interface (GUI) tool that simulates and visualizes real-time task scheduling algorithms, including RMS, EDF, LLF, and DMS. This tool will enable users to input periodic tasks, each with an associated execution cost, period, and determine if these tasks can be scheduled using the selected algorithm. The LLF and DMS algorithms will also have the ability to specify task deadlines which must be less than the associated period. If no deadlines are input, the deadlines will be the same as the periods. RMS and EDF will not have the option to specify deadlines other than the period and if any are input, they will be ignored. If the tasks are schedulable, the tool will generate a schedule and display it graphically. Tasks that are aperiodic will be out of scope for this project implementation since that would require the study of additional related algorithms for aperiodic task scheduling.

The key objectives of this project include:

1. **Algorithm Implementation:** Implement RMS, EDF, LLF, and DMS scheduling algorithms to determine if a set of tasks can be feasibly scheduled within their deadlines and if so, generate a possible schedule. Also implement exact analysis for RMS and EDF scheduling algorithms.
2. **User Interface (UI) Development:** Design a simple, intuitive interface where users can input periodic tasks and choose an algorithm to test their schedulability.
3. **Graphical Visualization:** Display the output schedule in a clear graphical format, showing how tasks are allocated to processor time based on the selected algorithm.
4. **Performance:** Ensure the tool is efficient, responding to user input within 5 seconds and supporting up to five tasks for a manageable display size and processing time.

3. Solution Methodology / Approach

The challenge that many developers and students face when dealing with real-time systems is selecting the right scheduling algorithm for a given task set. The decision-making process can be difficult without a clear understanding of how each algorithm works in practice or how task characteristics (e.g., period, cost, and deadlines) affect the feasibility of the schedule.

The GUI will enable users to input task parameters (such as task period and cost) in an easy-to-understand format. This will eliminate the need for complex manual calculations and ensure that users can focus on the scheduling algorithms themselves.

Once tasks are entered, the backend of the application will use exact analysis techniques to determine if the tasks can be scheduled according to the selected algorithm. If the tasks are schedulable, the system will generate a schedule for each algorithm and display it visually. By implementing algorithms like RMS, EDF, LLF, and DMS, users can quickly see how each algorithm handles the same set of tasks and compare their performance.

The graphical output will provide a visual representation of task execution times on a timeline, allowing users to observe the task scheduling process in real-time. This visual representation will help users better understand how the algorithms allocate processor time, handle task deadlines, and manage task prioritization. For example, users can easily spot task deadline misses in the schedule or see how tasks with shorter periods are prioritized under RMS or EDF.

3.1. Description of Algorithms

This section will provide an overview of the four algorithms implemented in our GUI: RMS, EDF, LLF, and DMS. In these algorithms, each periodic task must have a period (the time value when the task recurs and must be processed again), a cost (the length of time the processor requires to complete the task), and a deadline (the time the task must be completed, same as the period by default). Tasks will be formatted as follows, where p equals task period and c equals task cost: T = (p, c). For LLF and DMS, d equals deadline will be an additional parameter: T = (p, c, d).

3.1.1. Rate-Monotonic Scheduling

RMS is a fixed priority-driven preemptive scheduling algorithm for periodic tasks. In this model, it is assumed that the instance deadline is the end of its period. Tasks are prioritized based on period, with the shortest period tasks being prioritized, regardless of the task instance's deadline. For example, in a task set with tasks T1 = (12, 2), T2 = (6, 1), and T3 = (24, 5), T2 will always be prioritized over other tasks. Being prioritized means that if there is a T2 task available, it will always preempt other task instances. Also, T1 will always be prioritized over T3 task instances.

To schedule a task set, begin at time 0 with all tasks having a "ready" instance. Begin scheduling tasks for their full cost by order of priority. When a multiple of a tasks priority occurs, another instance of that task is added to the "ready" queue. Continue scheduling until the least common multiple of the period for all tasks in the set is reached. An example schedule is shown below for the task set:

	T2	T1	T1	T3	T3	T3	T2	T3
t=	0	1	2	3	4	5	6	7
	T3	--	--	--	T2	T1	T1	--
t=	8	9	10	11	12	13	14	15
	--	--	T2	--	--	--	--	--
t=	16	17	18	19	20	21	22	23

A schedulability check can be performed for RMS by adding together the sum of c/p for all tasks in the set. If this value is less than or equal to n(2^1/n – 1), where n is the number of tasks, the task set is schedulable. For the above example, this calculation would be:

2/12 + 1/6 + 5/24 = 0.542 <= 3(2^(1/3) -1) = 0.78.

Since the schedulability check passes, this task set is schedulable by RMS. However, failing this check does not necessarily mean that the task set is not schedulable by RMS. If this test fails, an exact analysis should be performed to ensure that the task set is not schedulable.

Exact analysis can be performed by progressively adding up costs of all tasks until the amount of work at that time (all costs added up through that time) remains the same, indicating there is enough time to complete the tasks and work does not continue to increase. If the time does not converge to a single value, the task set is not schedulable by RMS. This is referred to as the Completion Time Test.

Consider the task set T1 = (6, 2), T2 = (10, 2), T3 = (5, 1), and T4 = (6, 1). Schedulability check:

2/6 + 2/10 + 1/5 + 1/6 = 0.9 !<= 0.76. Since the sum of utilization is less than 1, there is still a chance that this task set is schedulable. Thus, the exact analysis can be calculated as shown:

- t0 = 2 + 2 + 1 + 1 = 6
- t1 = 2(2) + 2 + 2(1) + 2(1) = 10
- t2 = 2(2) + 2(2) + 3(1) + 2(1) = 13
- t3 = 3(2) + 2(2) + 3(1) + 3(1) = 16
- t4 =
- t5 =
- t6 =

Since the workload converges, this task set is schedulable.

3.1.2. Earliest Deadline First

EDF is a static priority-driven preemptive scheduling algorithm for periodic tasks. In this model, it is assumed that the instance deadline is the end of the period. Please note that while EDF can be performed on a task set with deadlines earlier than the end of the period, this is outside the scope of this paper.

Tasks are prioritized based on instance deadline for each task. Prioritization is unique to each task instance, unlike RMS, where priority is assigned to the task itself. Although EDF and RMS are relatively similar, they can produce very different results, with task sets often being more schedulable in EDF as compared to RMS. This is because EDF is a static scheduling algorithm that puts deadlines ahead of all other components, meaning that it is more likely to be schedulable.

In EDF, the schedulability check is necessarily determined by the sum of c/p for all tasks in the set. If this sum is less than 1, this indicates that the utilization is less than 100%, or that the task set is schedulable. For example, consider the task set T1 = (6, 2), T2 = (12, 3), and T3 = (3, 1). The utilization of this task set is the following:

2/6 + 3/12 + 1/3 = 0.92 <= 1.

Since the total utilization is less than 1, the task set is schedulable. If a task set fails the schedulability check for EDF, it has been proven not schedulable, and no additional analysis is required.

Scheduling for the EDF algorithm is like the procedure for RMS, except that deadline is used as the indicator for which task is prioritized. It is important to keep in mind that different tasks may be prioritized at certain points throughout scheduling depending on when that instance's deadline is compared to other "ready" tasks. If multiple tasks have the same deadline, ties are broken based on the remaining computation time (with the least being prioritized). The example schedule for this task set is shown below:

T3	T1	T1	T3	T2	T2
t = 0	1	2	3	4	5
T3	T1	T1	T3	T2	--
6	7	8	9	10	11

3.1.3. Least Laxity First

The LLF scheduling algorithm is a static priority-driven preemptive algorithm that prioritizes scheduling based on the smallest laxity of each task instance. Laxity of a task refers to the changing time span between the remaining computation time for a task and its instance deadline. Laxity of a task can be calculated with the following formula, where d equals instance deadline, t equals time, and c equals remaining computation time of the instance:

$$L = d - (t + c).$$

The schedulability check for LLF is the same as EDF, with the task set being schedulable if the sum of c/p for all tasks in the set being less than one. Consider the task set from above: T1 = (6, 2), T2 = (12, 3), and T3 = (3, 1). The table below shows the laxity value for each task at a given time (assume the laxity is assigned to the specific "ready" instance of that task). A scheduling table was generated simultaneously at each time point, since laxity is a dynamic parameter. Ties were broken by favoring the task that was already running.

t	0	1	2	3	4	5	6	7	8	9	10	11
T1	4	**3**	**3**	-	-	-	4	**3**	**3**	-	-	-
T2	9	8	7	6	**5**	**5**	5	4	3	**2**	-	-
T3	**2**	-	-	**2**	-	-	**2**	-	-	2	**1**	-

T3	T1	T1	T3	T2	T2
t = 0	1	2	3	4	5
T3	T1	T1	T2	T3	--
6	7	8	9	10	11

This schedule result is very similar to that generated for the task set by EDF, but T2 and T3 ran in a different order at time 9 and 10. Changes such as these that allow for a task to continue running may end up reducing time for context switching in a real time system and thus may be more favorable.

3.1.4. Deadline Monotonic Scheduling

DMS is a fixed priority-driven preemptive scheduling algorithm similar to the RMS algorithm. DMS is a more generic version of RMS which allows a deadline the is less than or equal to the period to be input as an additional parameter. DMS is useful for tasks that have a sooner deadline than their periods and can assist with raising priority for certain tasks. The schedulability check and exact analysis for DMS are the same as RMS, except deadline should be substituted for period as the prioritized parameter.

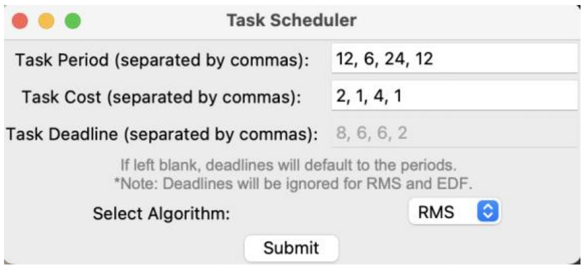
Consider the following task set: T1 = (12, 2, 4), T2 = (6, 1, 6), and T3 = (24, 5, 10). This task set is schedulable by DMS as shown below.

T1	T1	T2	T3	T3	T3	T2	T3
t=0	1	2	3	4	5	6	7
T3	--	--	--	T1	T1	T2	--
8	9	10	11	12	13	14	15
--	--	T2	--	--	--	--	--
16	17	18	19	20	21	22	23

The difference between RMS and DMS schedules are slight, but in this example, T1 is prioritized over T2 despite its longer period. This may be ideal in an application where there is a less frequent task that requires results more urgently than other more frequent tasks.

3.2. Illustrative Example

The following example demonstrates the functionality of the GUI application as well as some differences between all four algorithms. The first figure demonstrates the input window for our implementation.



Task Period (separated by commas): 12, 6, 24, 12

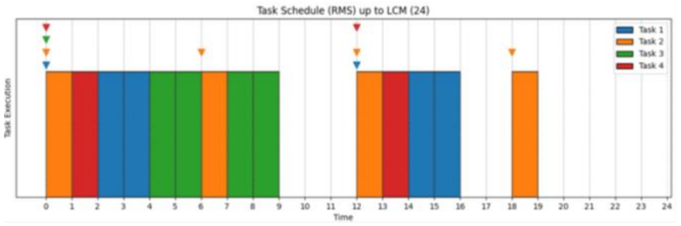
Task Cost (separated by commas): 2, 1, 4, 1

Task Deadline (separated by commas): 8, 6, 6, 2

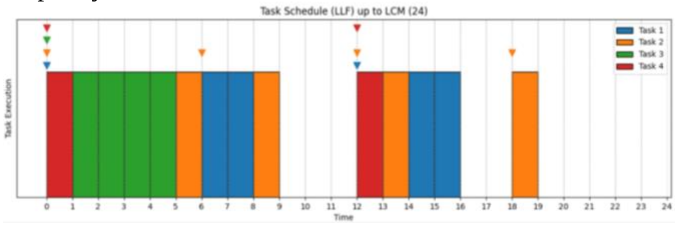
If left blank, deadlines will default to the periods.
*Note: Deadlines will be ignored for RMS and EDF.

Select Algorithm: RMS

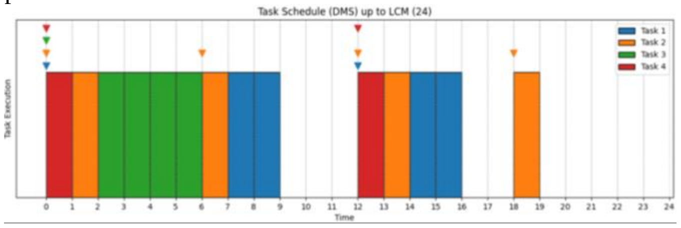
This example contains 4 tasks: T1 = (12, 2), T2 = (6, 1), T3 = (24, 4), and T4 = (12, 1). For the LLF and DMS algorithm, corresponding deadlines of 8, 6, 6, and 2 were also included.



The results of the RMS scheduler are shown in the figure above. With this algorithm, T2 is always prioritized and preempts all other tasks due to its low period. While not pictured, this is the same schedule as output by EDF.



The results of the RMS scheduler are shown in the figure above. With this algorithm, T2 is always prioritized and preempts all other tasks due to its low period.



The results of the RMS scheduler are shown in the figure above. With this algorithm, T2 is always prioritized and preempts all other tasks due to its low period.

4. Implementation Architecture

The system architecture of this implementation leverages a modular software design and integrates various tools, libraries, and programming concepts to enable real-time task scheduling and visualization. The implementation is developed in Python and uses the Tkinter library to create a graphical user interface (GUI). Tkinter provides a lightweight yet effective framework for user input, where users can specify task parameters such as periods, execution costs, and deadlines. The GUI includes dynamic input fields, dropdown menus for scheduling algorithm selection, and interactive feedback through validation messages, ensuring usability and clear interaction.

The system relies on several computational models and techniques for real-time scheduling including RMS, EDF, LLF, and DMS as detailed in previous sections of this paper. Each scheduling algorithm is implemented using Python's functional programming constructs, with tasks represented as objects containing properties such as period, cost, and deadlines. The algorithms are designed to determine task schedulability based on utilization bounds or priority rules, employing mathematical computations such as processor utilization checks and least common multiple (LCM) calculations to simulate task execution over time.

Visualization of task schedules is handled using the Matplotlib library, enabling graphical representation of scheduling results. Tasks are plotted as horizontal bars with the left edge aligned with the start of the time block. For additional clarity, periods are marked by triangular indicators to represent when each task comes back up. The visualization spans the LCM of task periods, ensuring comprehensive representation of periodic behavior over a defined time frame. Customization of plots, including color-coded task labels and gridlines, enhances the readability and interpretation of the results. The use of Matplotlib integrates seamlessly with the Tkinter-based interface, providing a smooth workflow from input submission to graphical output.

For ease of getting started, a default selection is made for the user. This default pre-selects the RMS algorithm and includes a task set that is schedulable under each algorithm. The task deadlines are also filled in but are grayed out and will not be applied to either RMS or EDF. Once the user selects LLF or DMS, the algorithm will schedule based on the specified deadlines and for additional clarity, this default options will produce a different schedule under each algorithm except RMS and EDF.

From a hardware perspective, the implementation is designed to run on general-purpose computing platforms. The workload generation and computations are lightweight, ensuring compatibility with typical consumer-grade laptops or desktops running Python 3.x. The system's modular codebase is structured to allow extensions or modifications, such as adding new scheduling algorithms or refining existing schedulability tests. Libraries such as NumPy are used to handle numerical computations efficiently, supporting tasks like generating time sequences for simulation and plotting.

The system also incorporates basic error handling and user guidance mechanisms, such as validating the consistency of task parameters and ensuring deadlines are within acceptable bounds. This promotes robust operation, preventing runtime errors or incorrect configurations. While the implementation does not directly integrate with hardware platforms like microcontrollers or real-time operating systems, it provides a conceptual foundation for scheduling in real-time systems, suitable for educational or experimental purposes. Overall, the combination of Python's rich ecosystem, intuitive GUI design, and mathematical rigor ensures technical depth and breadth in this task scheduler implementation.

5. Evaluation

The test plan for evaluating the real-time scheduling algorithms was designed to rigorously assess the functionality and correctness of the RMS, EDF, LLF and DMS algorithms. The objective was to ensure that each algorithm could effectively schedule tasks and produce accurate results under a variety of conditions, including scenarios where some algorithms would succeed while others failed. The testing and development process was carried out on both Windows and macOS devices, ensuring compatibility of the graphical user interface (GUI) across these operating systems. This setup provided sufficient resources to execute the algorithms, generate graphical representations, and validate cross-platform functionality.

The testing process began with the implementation of each algorithm in Python, starting with RMS, followed by EDF, LLF, and DMS. After implementing an algorithm, it was validated against initial test cases to ensure correctness. As new algorithms were added, previous implementations were re-tested to confirm they continued to function correctly and produced distinct results. Tasks were defined in terms of periods, computation costs, and deadlines, and the outputs were visualized using Matplotlib to create Gantt-like charts for interpretation. The testing plan incorporated unit tests to verify individual algorithm rules, integration tests to evaluate cross-algorithm consistency, and "real" tests to simulate practical scheduling scenarios.

The first test case involved task periods of 6, 12, and 15 units, with computation costs of 2, 3, and 4 units, respectively. RMS correctly prioritized tasks based on fixed priorities, with higher-frequency tasks receiving higher priority, while EDF dynamically scheduled tasks by their earliest deadlines, producing a valid but distinct schedule. As LLF and DMS were implemented, additional test cases were introduced. One such test case used task periods of 12, 6, 24, and 12 units, computation costs of 2, 1, 4, and 1 units, and deadlines of 6, 6, 12, and 8 units. LLF dynamically adjusted task priorities based on their laxity, ensuring no deadline violations, even when the deadlines were modified to 6, 3, 12, and 8 units. DMS prioritized tasks based on relative deadlines and produced correct schedules in some scenarios but failed in cases where deadline constraints exceeded feasible utilization bounds, which was expected.

The final set of tests evaluated algorithm performance under a more complex task set, with task periods of 4, 5, 8, and 10 units, and computation costs of 1, 2, 1, and 2 units. RMS and DMS failed to produce feasible schedules due to utilization exceeding their bounds, while EDF and LLF successfully scheduled the tasks. To further test LLF's flexibility, deadlines were added as 4, 4, 6, and 8 units, and LLF adapted to these constraints, dynamically reassigning priorities to maintain deadline adherence. These tests validated LLF's robustness in handling dynamic conditions.

The evaluation used metrics such as schedulability, correctness, flexibility, and visualization clarity. Schedulability was assessed based on whether a feasible schedule was produced without deadline violations. Correctness ensured the algorithm adhered to its specific rules, while flexibility examined how well the algorithm adapted to changing constraints. Visualization provided clear graphical confirmation of task execution windows and idle times. Results demonstrated that RMS and DMS performed well for low-utilization scenarios or tasks with consistent relative deadlines, but they struggled under high-utilization or dynamic deadline conditions. EDF and LLF excelled in flexibility and adaptability, with EDF prioritizing deadlines effectively and LLF dynamically adjusting to minimize laxity. The graphical representations were critical in interpreting the results and confirming correctness. Overall, the testing process verified the accuracy of the algorithms and highlighted their strengths and limitations, providing a replicable procedure for future evaluations.

6. Conclusions

In this experiment, the authors successfully created a Python scheduling GUI that implements the RMS, EDF, LLF, and DMS scheduling algorithms. The primary contributions involved designing and coding the GUI to allow users to input task data and then applying each algorithm to visualize the task schedules. Mara worked on the initial input GUI and RMS scheduling algorithm, while Tessa implemented the EDF, LLF, and DMS scheduling algorithms.

Through testing and evaluation, we observed the strengths and weaknesses of each scheduling algorithm under different conditions. For instance, RMS worked well with less saturated task sets, while EDF and LLF showed better adaptability to complicated and saturated task sets. DMS proved effective in minimizing deadline misses for more complex scenarios with different deadlines that were less than the period.

For future experiments, we recommend exploring additional scheduling algorithms, incorporating real-world constraints such as critical section locks, and optimizing the GUI for improved user interaction. This would provide valuable insights into the broader applications of real-time scheduling in various fields.

Overall, this project has provided the authors a deeper understanding of real-time systems and the practical application of different scheduling strategies, while also enhancing our coding and problem-solving skills.

Self-Assessment of Project Completion:

Project learning objectives	**Status (Not/Partially /Mostly/Fully Completed)**	**Pointers in the document**
Self-contained description of the project goal, scope, and relevant requirements	Fully Completed	Section 2, 2.3, page 1, 3
Self-contained description of the solutions (algorithms/protocol /applications/etc.)	Fully Completed	Section 3, 3.1, page 3
Adequate description of the implementation details (data structures, pseudo code segments, libraries used, etc.)	Fully Completed	Section 4, page 4-5
Testing and evaluation – test cases, metrics, test results, any relevant performance results	Fully Completed	Section 5, page 5-6
Overall Project Success assessment	Successful	

7. References

[1] Lui Sha, et al. “Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems.” Proceedings of the IEEE, vol. 82, no. 1, 1994, pp. 68–82, https://doi.org/10.1109/5.259427.

8. Team Members’ Contributions

Tasks/Member	Tessa	Mara
Literature survey	LLF and DMS research	RMS and EDF research
Design	Designed same parts as implemented	Designed same parts as implemented
Implementation	Created visualization, LLF and EDF scheduler	Created input GUI, RMS and DMS scheduler
Testing/Evaluation	Test cases for EDF and LLF	Test cases for RMS and DMS
Preparation of the Report and Presentation	Made slides 6-10	Made slides 1-5
Total percentage of contribution to the project	50%	50%