Earliest Deadline Late Server

Zafirul Izzat Bin Mohamad Zaidi
Hochschule hamm-Lippstadt
Lippstadt, Germany
zafirul-izzat-bin.mohamad-zaidi@stud.hshl.de

***Abstract*—We are not running the risk of making a mistake, no matter how precisely constructed it is. Any real-time system, for example, is vulnerable to perturbations caused by slight errors in software coding or failures owing to hardware design flaws in input channels. The repercussions of such failures not only jeopardize human life but may also impair high-value equipment in numerous systems, resulting in massive losses, particularly in embedded systems. As a result, it must be required to develop a real-time system. The server plays an enormous role in real-time system to make sure the process of the system run smoothly and efficiently. In this article, the main discussion is about how Earliest Deadline Late server overcome the matter arise due to the scheduling period and aperiodic task to make sure the system run placidly. The reader will first introduce with terminology and background of the real time system. After acquiring a common understanding, this article will help the reader by describing type of scheduling that is implemented nowadays and also know a specific term that will use for the entire paper. Following that, this study will get into priority server to have some knowledge on where Earliest Deadline Late severs located and play a role. Then, the reader may know the problem derive by the other server and how Earliest Deadline Late server tackle the problem. In addition, properties, mechanism, and application of how it works are going to be discussed in this study. Furthermore, the reader may discover the implementation of the topic based on the modelling. The conclusion mark as end of the article.**

***Index Terms*—real time system, earliest deadline late server**

I. Introduction

Currently, there is a lot of research interest in the field of real-time systems with a focus on scheduling. This is because digital computers are used so frequently in real-time applications. Many systems have seen a rise in attention in recent years. We can observe that the system's performance and dependability are improving day by day, as is the sophistication of real-time software. The controlling system in real-time monitoring and control applications is designed around one or more computers. The controlling system is integrated into its environment and acts on it in real time through suitable instrumentation to achieve the desired state. For example, the closed loop controlling system may read data from sensors at regular intervals and then analyse the data in accordance with a specified law. The outcome of the actuators will be the output to the environment [1]. Real-time systems are computer systems that have timing constraints imposed on their actions, such as a start time, a deadline, and a frequency of executions. The validity of a real-time system's output is determined not only by the logical calculation performed, but also by the time at which the result is given. There are two kinds of real-time systems. The first is hard real time, followed by soft real time [2]. The reader may be aware of this later in the article, which will be explained in the topics below. In our life as example, we have a lot to do and to be done in a specific time. This analogy same as a deadline in real time system.

In this article, I will discuss the Earliest Deadline Late server in detail. Before proceeding to the main issue, the reader may discover an intriguing portion to have a thorough grasp of the Earliest Deadline Late server. The first part discusses scheduling types and a few terms that will be used. Then comes the sort of server that is commonly used in daily life in the context of a real-time system. Furthermore, the reader will be exposed to the challenge of other servers and how Earliest Deadline Late dealt with it. The model's implementation is also shown below.

II. Scheduling algorithm

We have previously covered the fundamentals of real-time operating systems, and now we will go further into one of the most critical aspects of creating an embedded system. The scheduling of tasks and the algorithms utilized are critical in order to realize any system that runs efficiently. To understand what Earliest Deadline Late server is about, the reader needs to understand the scheduling aspect. According to my understanding, scheduling is the process of choosing which task should run first in the sequence. The task with the highest priority should be executed first. Essentially, the main idea of scheduler acts as a boss, deciding which tasks should execute in the system. In a real-time system, scheduling is critical for a designer to understand how the system operates and the expected output depending on the scheduling used. Various scheduling strategies may be supported by different real-time operating system releases. It is critical that we select the algorithm before we begin developing the user application. As with many other aspects of engineering, there is no universal method that is suited for every situation. There will always be trade-offs. In this situation, they are primarily concerned with speed, sometimes known as response times, implementation complexity, and so on. The chosen method should always be able to meet the time requirements of the tasks, otherwise, the algorithm is unsuitable.

There are several scheduling techniques available for task execution. Hard and soft real time are the two primary categories. When we talk about hard real time, the first thing that comes to mind is that if a system's job fails to fulfil the

University of applied science Hamm-Lippstadt

deadline, the consequences might be disastrous. One of the scenario examples is in a safety sensitive system, such as a temperature controller in a nuclear facility, if the temperature of the regulated environment surpasses a certain threshold, the damage to the apparatus and potentially the plant might be cataclysmic. It is acceptable for soft real-time systems to occasionally miss some deadlines. Even though it is late, it is still beneficial for the system to complete the work. For example, SMS text message that we communicate throughout the day is classifies as soft real time. Tasks are permitted to miss some of their deadlines in weakly-hard real-time systems. As consequence, there is no associated value if they finish after the deadline. Multimedia systems are typical illustrative instances weakly hard real-time requirements since it is not required to achieve all job deadlines as long as the deadline breaches are suitably spaced. It should be noted that while discussing real-time systems, the emphasis is usually on hard real-time [3]. “Fig. 1” shows the classification of scheduling.

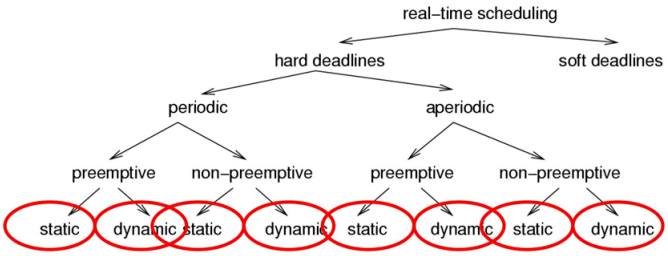

Fig. 1. Flow of scheduling [5]

.

A. Periodic and Aperiodic tasks

Periodic tasks are the primary computational requirement in many real-time control systems. When it comes to scheduling, the term periodic refers to tasks that run on a regular basis with a set run time and deadline. In real life, one example of employing periodic is when data is acquired via a sensor. The example that is nearby while we are utilizing a heater. The data is gathered on a regular basis to accommodate the temperature that we specified. While on the other side, aperiodic tasks arrive at irregular periods and have either soft or hard deadlines. One example is when a pedestrian presses a button at a traffic signal. The activity of the pedestrian when the button is pressed is referred to as an interrupt, and it is aperiodic [4].

B. Preemptive and Non-preemptive

Based on the “Fig.1”, both periodic and aperiodic task is separated into two types which are preemptive and non-preemptive. Preemptive scheduling permits a presently running job to be interrupted so that another with a higher ”urgency” level can be executed. The scheduler moves the interrupted task involuntarily from the running to the ready state. This algorithm’s dynamic switching between tasks is, in reality, a sort of multitasking. It is necessary to designate a priority level to each task. A running task may be stopped if a higher priority item joins the queue [6]. The scheduler has less control over the tasks in non-preemptive scheduling. It can only begin a job and then wait for it to complete or for the task to freely restore control. The scheduler cannot stop a running job. Non-preemptive scheduling can make task synchronization easier, but it comes at the expense of longer reaction times to events. This limits its practical application in sophisticated real-time systems.

III. Type of server

The scheduling algorithm might be either periodic or aperiodic. Both types of scheduling are being used in real today’s world, particularly in real-time control applications. The algorithms employed differ in how the systems operate. There are two priority servers. The server is divided to fix priority and dynamic priority. The technique used to schedule with fix priority and dynamic priority server is explained below.

A. Fix priority server

As explained above, periodic tasks are often time driven and conduct important control actions with hard timing limits to ensure consistent activation rates. While aperiodic task are frequently event-driven and, depending on the application. According on the system’s requirements, aperiodic tasks can be used as hard real time or soft. When we combine periodic and aperiodic tasks, we call it a hybrid task. In a regular situation, including aperiodic and periodic tasks, the kernel’s main purpose is to ensure that the running time of the tasks follows the deadline according to its period. As a result, it provides appropriate average reaction times for non-real-time and soft tasks. Fix priority server is used to schedule both periodic and aperiodic tasks. Fix means that tasks must be scheduled based on a fixed priority assignment. One of the way to assign them is by Rate Monotonic Algorithm. Firm refers to aperiodic actions that require online assurance on particular occurrences. When a firm aperiodic request to run, the kernel can perform an acceptance test to see whether the request can be served within its deadline. If such a promise is not possible, the request is denied [4].

B. Dynamic priority server

We will now look at dynamic priority servers. Dynamic priority in a real-time system can change the CPU need for a job throughout run time. For example, when a task approaches its deadline, the priority can be dynamically increased to ensure that it does not miss the deadline. That seems to be, the deadline can be extended. The good thing about having a dynamic priority server is that it allows you to fit more tasks into a shorter time frame compared, to fix priority server. Priorities are then redistributed depending on the scheduling algorithm, which considers the system’s specific characteristics. For illustration, the time remaining till the deadline, the period, slack time, or the release time. Some users choose dynamic priority servers because they provide

greater flexibility. Earliest Deadline Late server is one of the example of dynamic priority server that will be discussed further below [4].

IV. Problem

Now, we will discuss about the problem occur specially in dynamic priority sever since Earliest Deadline Late is related to this topic. Then for the solution, we will see how Earliest Deadline Late improvise the scheduling of other servers.

First we will schedule based on the Dynamic Sporadic Server2 (DSS) to see the problem occur. A little history about this dynamic server is that it is an aperiodic service strategy proposed by Spuri and Buttazzo. This server is an enhanced version of the Sporadic Server [4], designed to operate effectively under a dynamic Earliest Deadline First scheduler. The Dynamic Sporadic Server has a T_s time and a C_s capacity. For prospective aperiodic inquiries, both time and capacity must be kept, just as on other servers. When scheduling tasks, the idea of having Dynamic Sporadic server is to accommodate a combination of periodic and aperiodic activities. The periods of replenishment are determined by a replenishment rule, allowing the system to attain full processor usage. The primary distinction between dynamic and static versions for readers who are already familiar with Sporadic Server in fixed priority server is how the priority is provided to the server. When we mention Sporadic Server, we are referring to the server fixed priority, which is decided by the Rate Monotonic algorithm. To help the reader understand this part, the explanation will be delivered with the help of the "Fig. 2".

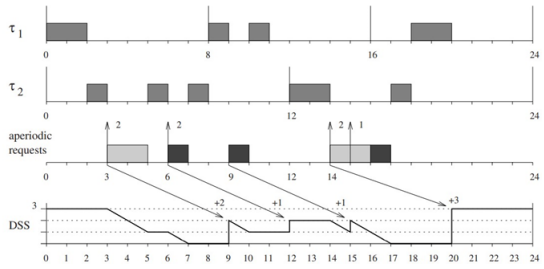

Fig. 2. Example of Dynamic Sporadic Server [4]

.

In "Fig. 2" above, the Earliest Deadline First schedule is utilized, and the tasks consist of two periodic tasks. The period for the first task is τ_1 = 8, while the period for the second periodic task is τ_2 = 12. C_1 = 2 and C_2 = 3 are the execution time for both periodic jobs. DSS likewise has its own time, T_s = 6, . The value of the capacity is, C_s = 3.

The schedule begins at time t_1 = 0, and the server capacity value is C_s = 3. For the reason there is no aperiodic task at beginning, t_1 will execute first due to priority given and has the earliest deadline. Owing to the fact that there is no aperiodic request, t_2 is executed. At time t = 3, an aperiodic request is received. At the moment, the execution time of an aperiodic task is 2. Because the rule of $C_s > 0$ is followed, the initial replenishment time and deadline are determined. RT_1 = d_s = 3 + T_s = 9, where d_s is the earliest deadline. Essentially, when there is capacity, the aperiodic task takes precedence. It will continue to run till it is finished. The periodic task request is completed at time t = 5. In the view of the fact that there were no other pending requests from the aperiodic task, the replenishment of two units of time is planned at time RT_1 = 9. This is how the schedule acts in relation to the rest of the system. It should be noted that the aperiodic task cannot be assigned as the highest priority if the capacity is not available.

After analyzing the figure above, I can conclude that there is a flaw when scheduling with Sporadic Server. We can see from the diagram above that the Sporadic server technique may take a long time. As a result, the execution time for aperiodic requests may be altered. We can see that the aperiodic request may be significantly delayed. There are a few methods to enhance this version based on my findings. However, because the main issue is about the Earliest Deadline Late server, I will just discuss one of them. Each aperiodic request can be improved by designating the earliest deadline. This is critical since it may have an impact on the system's result. When used with Dynamic Sporadic Server with CPU utilization U_s, the amount of total processor usage of the aperiodic load must be less than 1. It is important to note that it must never surpass a preset maximum value Us. Based on the theorems of Spuri and Buttazo. [4]

Based on the theorem proposed by Spuri and Buttazzo in the book that I have refered is [4]*"Given a set of n periodic tasks with processor utilization Up and a TBS with processor utilization U_s, the whole set is schedulable by EDF if and only if"*.

$$U_p + U_s \leq 1 \tag{1}$$

Total Bandwidth Server is used for the improvise version. Based on the stated theory, this server is constructed to improve the efficiency of executing tasks while dealing with aperiodic tasks. There is a significance behind the given name. Whole bandwidth server is so named because anytime a periodic task is requested, the server's total bandwidth is immediately provided to it. It should be noted that it will only allocate if it is possible. The aperiodic job is subsequently added to the ready queue, indicating that it is ready to execute. The aperiodic task, like any other periodic work, is scheduled by Earliest Deadline First. [4]

"Fig. 3" depicts an Eearliest Deadline First schedule generated by two periodic tasks. The first task has τ_1 = 6 periods, whereas the second task has τ_2 = 8 periods and C_1 = 3, C_2 = 2 execution times. The first aperiodic request arrives at time t = 3 and is handled with a deadline of d_1 = 7 based on the equation d_1 = r_1 + C_1/U_s = 3+1/0.25 = 7. Because the aperiodic task's deadline is sooner than task τ_2, the aperiodic task may run first, followed by periodic task τ_2. Similarly, the second and third requests arrive at times t = 9 and t = 14, respectively.

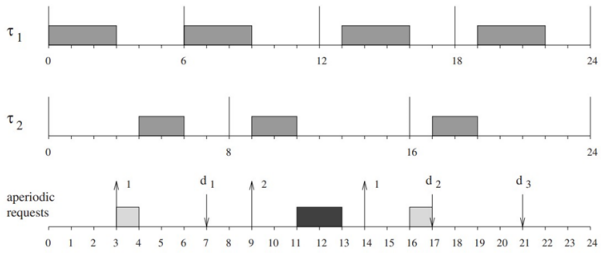

Fig. 3. Example of Total Bandwidth Server [4].

V. Approach

As we can see in the problem part, we can say that Total Bandwidth Server is a better version to maximise the capacity of the task in the schedule compared to Dynamic Sporadic Server. This is due to its ability to deliver good aperiodic responsiveness while being extremely simple. However, there is another sophisticated algorithm that may be used to improve Total Bandwidth Server and can still produce superior performance. For instance, looking at the preceding "Fig. 3", we can see that it is feasible to run or serve the second and third aperiodic tasks as soon as they come. Even if the aperiodic task is completed a little early, it has no effect on the overall system. This is achievable because the active periodic instances have enough slack time to be securely preempted when the requests come. So, the solution to this problem is Earliest Deadline Late Server. The advantage of employing Earliest Deadline Late Server is that it completely utilizes the available slack of periodic tasks for aperiodic task execution.

The Eearliest Deadline Late server core idea is to leverage idle intervals in an Eearliest Deadline Late schedule to perform aperiodic requests as quickly as feasible. We created a scheduling algorithm A and a task set J to simplify the description of the Earliest Deadline Late server below. [4]

$$W_J^A(t) = \begin{cases} 1 \text{ if the processor is idle at t} \\ 0 \text{ otherwise} \end{cases} \quad (2)$$

Two significant things may be expressed by the equation above. The first represents the time when idle time occurs, E = (e_0, e_1,..., e_p). The second might indicate the durations of idle times, D = (0, 1,..., p). The table case study is presented below.

TABLE I
IDLE TIME UNDER EARLIEST DEADLINE LATE SERVER

i	0	1	2	3
e_i	0	8	12	18
Δ_i	3	1	1	1

Figure following depicts the behaviour of both periodic and aperiodic activities to assist the reader comprehend the table mentioned above. As previously stated, aperiodic tasks will run whenever there is idle time. That seems to be, periodic tasks are not running at the moment. As a result, when no aperiodic activities are present in the system, periodic tasks are scheduled as soon as feasible using the EDF algorithm. In another scenario, when a new aperiodic task is requested to execute, it must first wait for the preceding aperiodic task to become inactive, which we refer to as being in a pending state. When no prior aperiodic task is running and there is idle time, the requested aperiodic task will begin to execute. "Fig. 4"and "Fig. 5" depicts an example of the EDL service mechanism and its relationship to the table above.

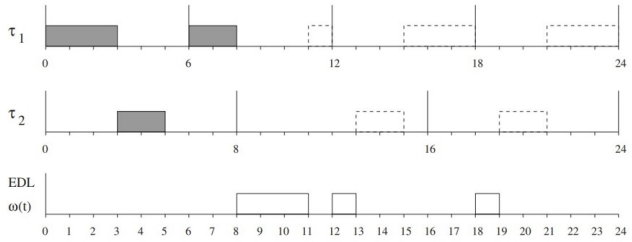

Fig. 4. Availabilty of idle time at t = 8 [4] .

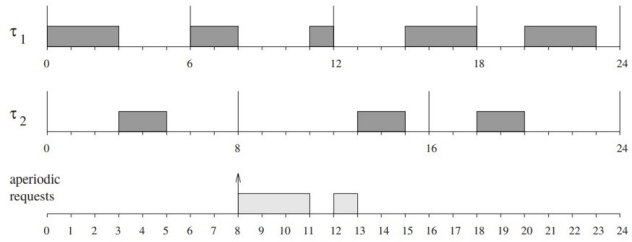

Fig. 5. Schedule aperiodic request under EDL server [4].

"Fig. 4" depicts the Earliest Deadline Late service approach. The first periodic task τ_1 arrives at t = 0, followed by the second τ_2 at time t = 3. Because there is no aperiodic request at this time, the periodic tasks continue to execute. The first aperiodic request arrives at time t = 8. The aperiodic request takes 4 units to execute. There is a gap between time t = 8 and t = 11 in "Fig. 4". The gap indicates that there is free time. Aperiodic requests for further information are automatically assigned a bandwidth of 1-U_p by the server. Because the reaction times attained by this technique are optimal, they can no longer be reduced. It should be noted that there is no deadline for aperiodic tasks. The periodic task τ_2 begins at time t = 6 and continues until t = 8. Then an aperiodic job will take its place. We know that the periodic task τ_1 has one execution time remaining and must complete before the deadline shown in "Fig. 5". As a result, the periodic task will be given top priority, and once completed, the aperiodic task will be handled again.

VI. EDL PROPERTIES

The Eearliest Deadline Late server's schedulable analysis is basic. We say its Earliest Deadline Late when there is an idle time, and the aperiodic task will execute whenever the idle time is available. When there is no aperiodic task available, the periodic task will run based on Earliest Deadline First schedule. This is to ensure the viability of the periodic task set is not jeopardized.

According to the theorem prposed by Spuri and Butazzo [4]-*"Given a set of n periodic tasks with processor utilization U_p and the corresponding EDL server (whose behavior strictly depends on the characteristics of the periodic task set), the whole set is schedulable if and only if"*

$$U_p \leq 1 \tag{3}$$

The equation also is proof according to [4]-*"Proof. If. Since the condition ($U_p \leq 1$) is sufficient for guaranteeing the schedulability of a periodic task set under EDF, it is also sufficient under EDL, which is a particular implementation of EDF. The algorithm schedules the periodic tasks according to one or the other implementation, depending on the particular sequence of aperiodic requests. When aperiodic requests are pending, they are scheduled during precomputed idle times of the periodic tasks. In both cases the timeliness of the periodic task set is unaffected and no deadline is missed."*

Based on the theorem and the proof above,what I understood is that it can ensure the scheduling of periodic tasks under Earliest Deadline First. If it is possible to meet the Earliest Deadline First, it is also sufficient to meet the Earliest Deadline Late. This is due to the fact that the Earliest Deadline Late is listed under the Earliest Deadline First. So, if an aperiodic task is pending, it will execute when idle time becomes available, and as the periodic task's period approaches the deadline, it will move the priority back to the periodic task to complete the work. In conclusion, it is possible to prevent missing a deadline.

VII. MODEL WITH UPPAAL

We will move on to modeling after obtaining a better understanding of the Earliest Deadline Late server. Based on what I have learnt, I will create an Earliest Deadline Late schedule via Uppaal. The model is divided into three components, each of which has specific states. As indicated below, the model consists of a task, a request handler, and idle time.

A. Task model

In "Fig. 6"The first component, task, consists of three states,two periodic tasks and one aperiodic task. Both periodic tasks will continue to run as long as no aperiodic request is received.

B. request handler model

When an aperiodic task makes a request for example in "Fig. 7", the status will change from no request to request. It should be noted that an aperiodic task cannot operate if there is

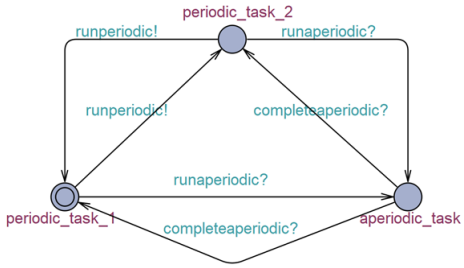

Fig. 6. Schedule of periodic and aperiodic task.

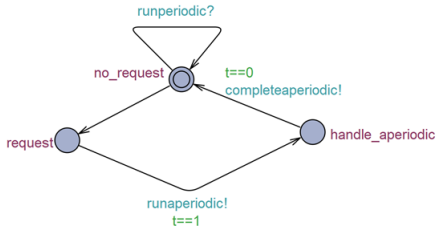

Fig. 7. Task handler.

no available time for idle time. When the idle time is changed to available, the aperiodic task is handled. The periodic task will then continue to run even if idle time is not available and if it close to the deadline and need to be complete.

C. Idle time model

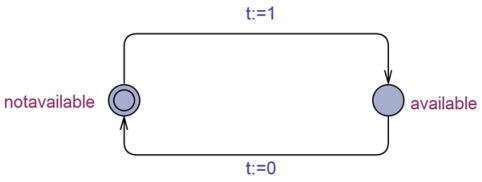

Fig. 8. Availability of idle time.

Since the request state move to handle aperiodic state, the idle time change to 1, $t = 1$ meaning that it is available for aperiodic task to run. The overview of the process is presented in "Fig. 8".

D. Overview of the process

The combination of the model, produce each running state in "Fig. 9". The state start at periodic task τ_1. Then periodic task τ_2 is execute next followed again by periodic task τ_1 since it has no aperiodic request. After that, the aperiodic request and waiting for the idle time to be available. Note that the aperiodic task has no specific deadline, and it will execute as soon as possible. After no idle time available, periodic task τ_1 continue is task execution.

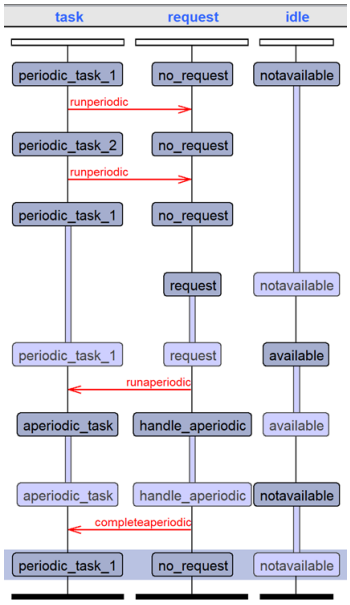

Fig. 9. Schedule of periodic and aperiodic task.

VIII. Conclusion

When it comes to real-time systems, task schedulers are quite important. It is critical to design a task scheduler to ensure that the system runs smoothly. The study has detailed a few scheduling algorithms that are specifically used in real-time systems. The reader may gain comprehension and distinguish which algorithms are most appropriate. The use of graphics to explain a few algorithms should assist the reader in visualizing and having a good imagination on this issue. It should be noted that the primary goal of this paper is to describe the behaviour of the Earliest Deadline Late server in a dynamic priority server. The reader has also been exposed to terminology that are commonly used in real-time systems. Finally, I hope that this explanation has shed some light on the properties of the Earliest Deadline Late server and the modelling part that I mentioned in the previous section, and I can say that with the help of the Earliest Deadline Late server, it can maximize capacity use while not jeopardizing any of the tasks, as we know that both periodic and aperiodic tasks will be fully executed without missing a single deadline.

IX. Acknowledgement

I am eternally thankful to my dear professor, Prof. Dr. Henkler, Stefan, whose inspiration, encouragement, advice, and support from the beginning to the end helped me to build awareness and open my eyes to the importance of scheduling in real-time systems in general. I would also want to offer my appreciation, respect, consideration, and advantages to any and all persons that assisted me in any way throughout the execution of the task. I did everything in my power to gain a better knowledge and share my understanding in this paper and I hope the reader can increase their knowledge specially in this topic.

References

[1] H. Chetto and M. Chetto, “Some results of the earliest deadline scheduling algorithm,” IEEE Transactions on Software Engineering, vol. 15, no. 10, pp. 1261–1269, 1989.

[2] M. Spuri and G. C. Buttazzo, “Efficient aperiodic service under earliest deadline scheduling,” Proceedings Real-Time Systems Symposium REAL-94, 1994.

[3] M. Silly-Chetto and A. Marchand, “Dynamic scheduling of skippable periodic tasks: Issues and proposals,” Journal of Software, vol. 2, no. 5, 2007.

[4] G. C. Buttazzo, Hard real-time computing systems predictable scheduling algorithms and applications. Johanneshov, Stockholm: MTM, 2013.

[5] J. J. Chen and P. Marwedel, “Aperiodic Task Scheduling,” 2014.

[6] O. Team, “RTOS scheduling algorithms,” Open4Tech, 10-Dec-2019. [Online]. Available: https://open4tech.com/rtos-scheduling-algorithms/. [Accessed: 18-May-2022]. pp.354-366.

[7] drey Marchand, Maryline Chetto. Quality of Service Scheduling in Real-Time Systems. International Journal of Computers, Communications and Control, Agora University of Oradea, 2008, 3 (4),