Real-Time Systems - Assignment 01 Group 9

Zhengtao Huang (5833469, zhengtaohuang) Barry Tee Wei Cong (5662834, btee)

1 Introduction

In this lab assignment, two different schedulers are required to be implemented, namely the Rate-Monotonic(RM) scheduling algorithm and the Earliest-Deadline-First(EDF) scheduling algorithm. This report will seek to answer the questions listed in the lab manual, including the results of the code scheduling output, which will be documented in each section. Our report will also explain about the occurrence of overloading and how each of the scheduling algorithms will handle such occurrences.

2 Understanding the System

2.1 The Example Scheduler

The example scheduler is not a real-time scheduler. The example scheduler does not take into account execution time, deadlines, and other factors related to whether the tasks would meet the deadlines or not. The scheduler is also not preemptive. Once a task has the time slot, it would continue executing until it finishes. The order in which the tasks are assigned time slots is dependent only on their positions in the ready queue. Such a design has no guarantee in meeting the deadlines, and hence should not be considered a real-time scheduler.

2.2 Hyper-Period of the Task set in "main.c"

The task set defined in "main.c" has a hyper-period of 120 ms, as it is the least common multiple(LCM) of all the periods of the three tasks, i.e. task 1, 2, and 3 having a period of 20, 40, 60 ms, respectively.

2.3 Two New Task sets

In this section, the report will provide two task sets. Task set A will have a utilisation factor of > 1 while Task set B will have a utilisation factor of 1.

2.3.1 Task set A - Utilisation Factor >1

	C_i	T_i
τ_1	2	20
τ_2	10	40
τ_3	40	60

Table 1: Task set A

The processor utilisation factor can be calculated with the formula below

$$U = \sum_i \frac{C_i}{T_i} = \frac{2}{20} + \frac{10}{40} + \frac{40}{60} = 1.0167$$

Following the information in Table 1, Set A of non-trivial tasks has a processor utilisation factor of 1.0167.

2.3.2 Task set B - Utilisation Factor =1

	C_i	T_i
τ_1	4	20
τ_2	10	40
τ_3	33	60

Table 2: Task set B

The processor utilisation factor can be calculated with the formula below

$$U = \sum_i \frac{C_i}{T_i} = \frac{4}{20} + \frac{10}{40} + \frac{33}{60} = 1$$

Following the information in Table 2, Set B of non-trivial tasks has a processor utilisation factor of 1.

3 Rate-Monotonic Scheduler

3.1 How It Works

Rate-Monotonic(RM) Scheduling is a static priority scheduling algorithm that operates on the cycle duration of the task in the task set. For the sets of task, the shorter the cycle duration, the higher the task's priority i.e. priorities are inverse to the period duration. This life cycle for RM Scheduling will remain throughout the entire run-time of the scheduling operation. If a higher priority task becomes ready, the current lower priority task is put back into the queue. This means that scheduling is deterministic and predictable and will always schedule tasks in the same way for any reruns (repeated sequence for every hyper-period).

Two variable fields were added to the Task struct. The first is next_arrival_time with the data type integer(int). The second is is_finished with the data type boolean(bool). The first variable field keeps records of the arrival time of the next job of the task in the form of a timestamp referenced to the system's uptime i.e. time elapsed since system's initial boot-up. The second variable field is a Boolean to indicate whether the job has been completed within the particular cycle.

3.2 Output from Logic Analyser

	C_i	T_i
τ_1	2	20
τ_2	10	40
τ_3	30	60

Table 3: Task set for RM

Based on the assignment, the set of task was given for RM as shown in Table 3. The RM scheduler was coded in C and flashed on the STM32F4Discovery microcontroller board. The Logic Analyser with the Logic 2 software generated the results of the RM Scheduler as shown in Figure 1.

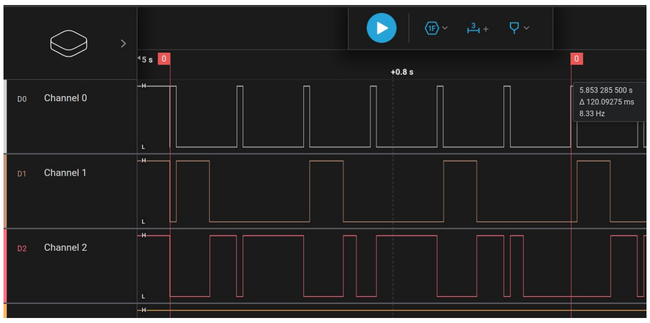


Figure 1: RM - Actual Output

Before coding for the RM scheduler, theoretical calculations were performed and displayed via a Python graph shown in Figure 9. The output of the logic analyser matched our theoretical calculations and hence verify the validity of our designed RM scheduler.

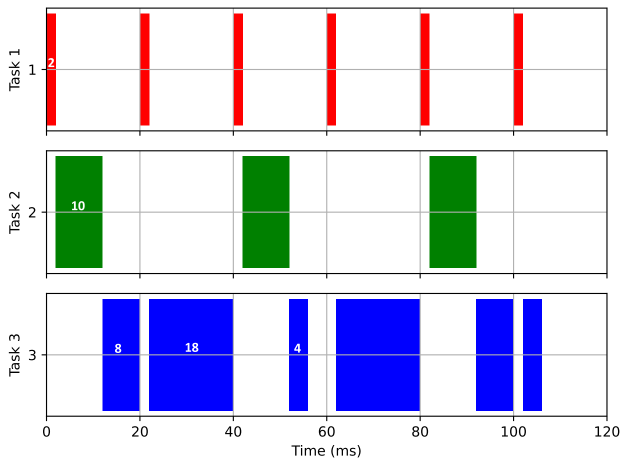


Figure 2: RM - Theoretical Output

3.3 Overhead of RM Scheduler

To calculate overheads of the RM scheduler, time was measured for all the occurrence from the falling edge of one job to the rising edge of the next job it transited to, as shown by the example in Figure 3. Table 3 shows all the overhead timings within a single 120ms hyper-period. With a total of 0.580 ms, the percentage of overhead is 0.483% for a single 120ms hyper-period.

It must be noted that the measurements of overhead was done starting from the second hyper-period onwards for two main reasons. The first reason is to show the falling edge of Task 3 at the end of the first hyper-period and the rising edge of Task 1 at the start of the second hyper-period. It would not have been accurate to use the rising edge of Task 1 during the start of the first hyper-period as the STM32 microcontroller was just flashed with the program. The second reason is because to obtain the data for serial number 0 and 8, the calculation is not as simple as taking the gap as the overhead time but there is a need to take the measured time to deduct the idle time and the spare time from the previous job that finished earlier.

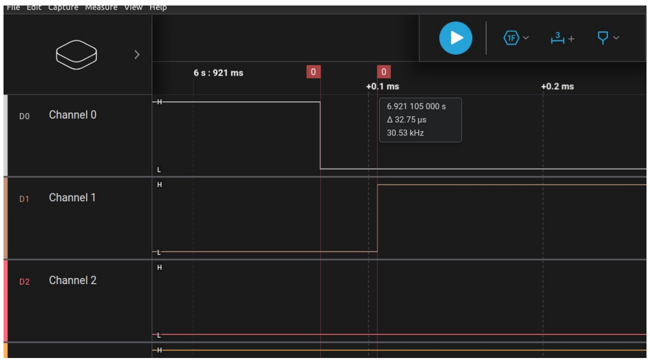


Figure 3: RM - Example of Overhead

S/N	End	Start	Time(μ s)
0	3	1	107.75
1	1	2	32.75
2	2	3	32.25
3	3	1	22.25
4	1	3	28.50
5	3	1	22.25
6	1	2	32.50
7	2	3	28.00
8	3	1	107.50
9	1	3	32.75
10	3	1	22.25
11	1	2	32.50
12	2	3	28.00
13	3	1	22.00
14	1	3	28.75
Total			580.00

Table 4: Overhead Timings Within 120ms Hyper-Period

3.4 Handling Overloading

When there is an overload due to missed deadlines, the designed RM scheduler continues to execute the set of tasks without panic, i.e., without stopping the execution process entirely. However, the overloaded task was handled with caution to achieve continuous execution of the task set, which will be explained below.

As shown by Task set A in Table 1, processor utilisation was calculated and proven to be more than 1. Figure 4 shows the theoretical graph of the RM scheduler for Task set A. It can be seen that the new job for Task 3 in Task set A has the lowest priority and will overshoot permanently by six milliseconds every second to eighth millisecond during the last half of the 120ms hyper-period. The second cycle will then not have enough time to complete before the end of the hyper-period of 120 ms. This event will be deemed permanent overloading. Following the theoretical knowledge of RM scheduling gained from the CESE4025 Real-

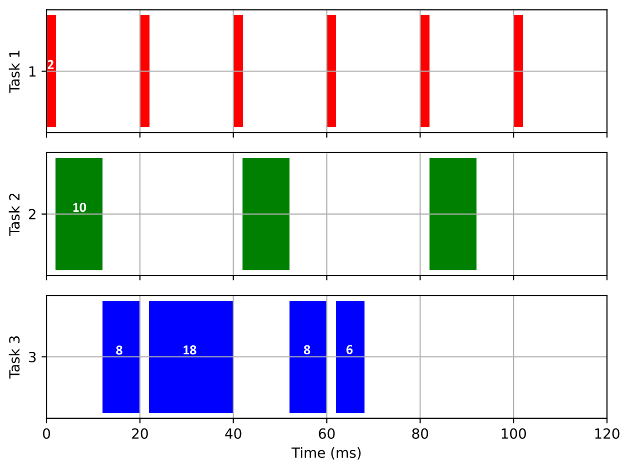


Figure 4: RM - Overshoot for Utilisation > 1

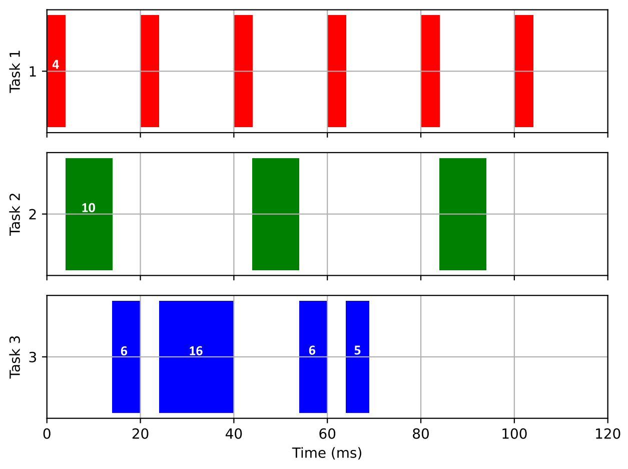


Figure 6: RM - Overshoot for Utilisation = 1

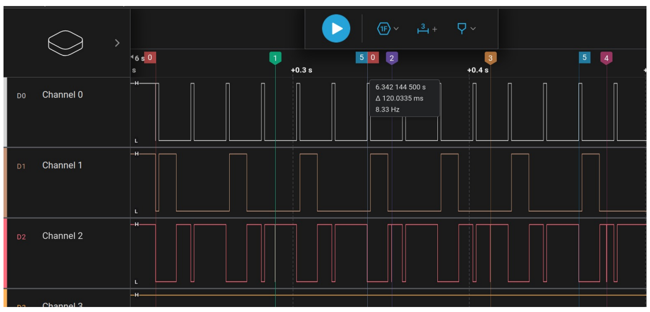


Figure 5: RM - Utilisation > 1

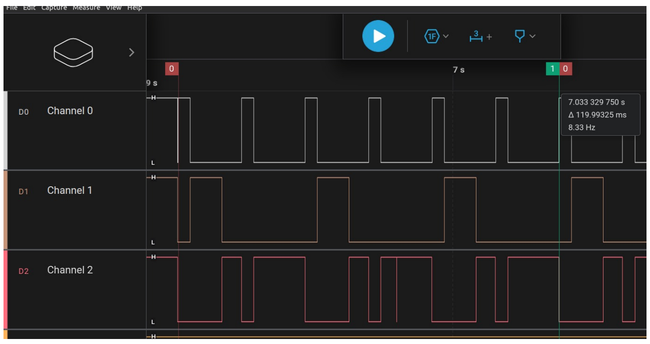


Figure 7: RM - Utilisation = 1

Time Systems course, i.e., Lecture 7 - Scheduling Periodic Tasks (Part 3), the second job for Task 3 will be blocked for every second half of the 120ms hyper-period. Hence, Task 1 will have the highest priority to be preempted followed by Task 2 with the second highest priority.

After coding on the STM32 microcontroller, Figure 5 presents the actual output of the logic analyser. When compared with utilisation < 1, the processor utilisation of more than 1 will show that Task C with the lowest priority will have its job blocked following the logic coded into the STM32 microcontroller. It can be noted that the graph output of the logic analyser differs from the Python plot in Figure 4 because upon clarification with the teaching assistants, it is not ideal for processor to be idle. Thus, the STM32 microcontroller will continue to run overran jobs as block does not mean dropping jobs in Task 3 while ensuring Tasks 1 and 2 with higher prioritisation preempt Task 3.

Although Task 1 will be prioritised over Tasks 2 and 3 and Task 2 will be prioritised over Task 3 i.e. adhering to the scheduling priority rules of RM scheduler, a problem can be seen is that jobs in Task 3 have perpetually overrun and violate the deterministic nature of RM scheduling. As shown in Figure 5, marker pairs 0, and 5 show the first and second 120ms hyper-period, respectively, while single marker 2 (job 2 of first hyper-period) and single 4 (job 2 of second hyper-period) exceeded marker pairs 0, and 5 respectively.

As shown by Task set B in Table 2, processor utilisation was calculated and proven to be 1. Figure 6 shows the theoretical graph of the RM scheduler for Task set B. It can be seen that Task 3 in Task set B has the lowest priority and will overshoot permanently by five milliseconds every fourth to ninth millisecond during the last half of the 120ms hyper-period, that is, during the new job for Task 3. The second cycle will then not have enough time to complete before the end of the hyper-period of 120 ms. This event of blocking of lower priority Task 3 for permanent overloading case follows the same logic as per what happened when RM has utilisation factor of more than 1.

After coding on the STM32 microcontroller, Figure 7 presents the actual output of the logic analyser. When compared with utilisation < 1, the processor utilisation of 1 will show that Task C with the lowest priority will have its job blocked following the logic coded into the STM32 microcontroller. It can be noted that the graph output of the logic analyser differs from the Python plot in Figure 6 because upon clarification with the teaching assistants, it is not ideal for processor to be idle. Thus, the STM32 microcontroller will continue to run overran jobs as block does not mean dropping jobs in Task 3 while ensuring Tasks 1 and 2 with higher prioritisation preempt Task 3. The second job of Task 3 was slotted into idle time slots of the process and still finished within its own hyper-period cycle(120ms).

3.5 Difference Between Theory and Task set B where U =1

The theory where the utilisation factor equals 1 for a single-core processor means that a processor has the ability to be fully utilised in terms of its full computational capacity to run continuously over the observed time frame. This means that processor is able to achieve 0 idle moments and execute tasks without any downtime.

Although RM scheduler shows permanent overloading of Task 3's job 1 exceeding its own period within the first half of the hyper-period in the given task set, RM is still able to schedule and execute Task 1, 2, and 3 all within the 120ms hyper-period, which matches the theory that a processor can be fully utilised despite having overheads in Task set B. Task set B behaviour fulfils theoretical expectations of utilisation of 1 but defies the theoretical expectation of RM scheduling. Overloading did not occur in this Task set B because it is noted that most of the task took abit lesser time than the implemented schedule time. Lab instructors explained the phenomenon that the time decrease for task equals to the context-switching overhead because their tasks have a resolution of less than 1ms. But in the scenario that tasks strictly follow the stipulated execution time, the entire Task set B will overrun akin to the scenario in Task set A where utilisation factor is more than 1. So any task set with utilisation factor of 1 does not always guarantee feasibility to meet all deadlines within the hyper-period and needs to be further verified, as shown the specific case of Task set B having additional overheads that threatened to make RM scheduling infeasible.

4 Earliest Deadline First Scheduler

4.1 How It Works

Earliest Deadline First (EDF) is a dynamic priority scheduling algorithm that operates on the cycle duration of the task in the task set. For the sets of task, the task's with the highest priority is the task with the closest deadline. So, it is different from RM because instead of taking task with the shortest period to have priority, task are now prioritised based on tasks with shortest time available to its next cycle. This means, when compared to the duration of period, which is static, the time available for each task is different and dynamically computed on the go.

EDF has an added feature called preemptive scheduling which means if there is a new task that just started, it can preempt the current task if the deadline of the new task is shorter than the deadline of the currently running task, which is clearly unpredictable and hence deemed a dynamic scenario. EDF is particularly useful in systems where tasks have varying arrival times, as it can adapt to these changes and generally provides better utilisation of system resources than fixed-priority schedulers like RM Scheduling.

Two variable fields were added to the Task struct. The first is next_arrival_time with the data type integer(int). The second is is_finished with the data type boolean(bool). The first variable field keeps records of the arrival time of the next job of the task in the form of a timestamp referenced to the system's uptime i.e. time elapsed since system's initial boot-up. The second variable field is a Boolean to indicate whether the job has been completed within the particular cycle.

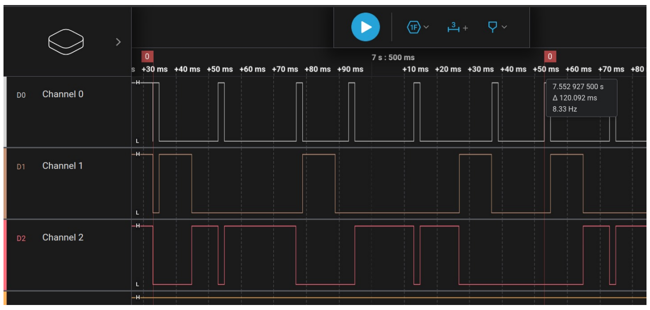
The difference in the programming logic from RM to EDF lies in EDF utilising the shortest time to its next cycle, whereas in RM, the allocation of the task prioritises task with the shortest time period. This is true when we assume that the deadlines are the same as the periods, which means that the deadline of a job is the arrival time of the next new job.

4.2 Output from Logic Analyser

	C_i	T_i
τ_1	2	20
τ_2	10	40
τ_3	30	60

Table 5: Task set for EDF

Based on the assignment, the set of tasks was given for EDF as shown in Table 5. The EDF scheduler was coded in C and flashed on the STM32F4Discovery microcontroller board. The Logic Analyser with the Logic 2 software generated the results of the EDF Scheduler as shown in Figure 8.



7.552 927 500 s
Δ 120.092 ms
8.33 Hz

Figure 8: EDF - Actual Output

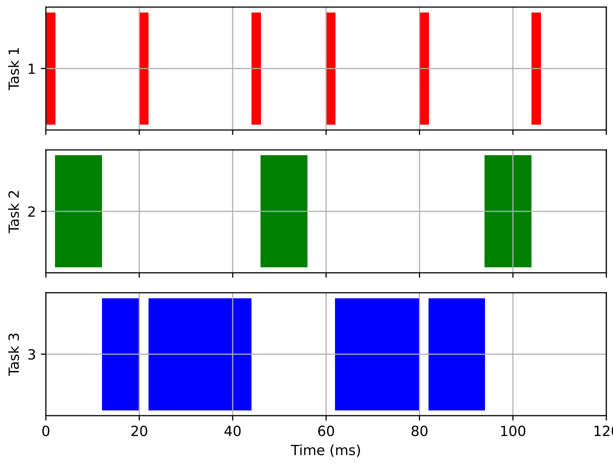


Figure 9: EDF - Theoretical Output

Before coding for the EDF scheduler, theoretical calculations were performed and displayed via a Python graph shown in Figure 10. The output of the logic analyser matched our theoretical calculations and hence verify the validity of our designed EDF scheduler.

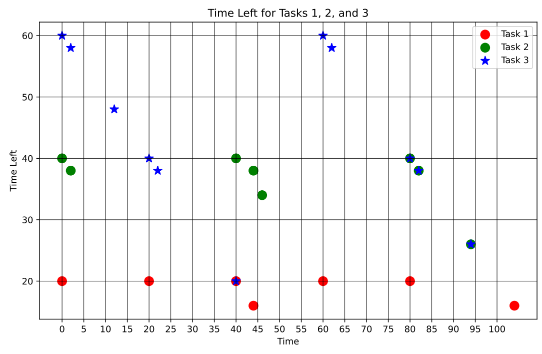


Figure 10: EDF - Time Left for Tasks

In Figure 10, it shows a theoretical calculation of the time left for each task prior to deciding which task gets preempted. An interesting point of interest is evident at the 40ms mark. Tasks 1 and 3 have the same time left to its deadline. However, since Task 3 is in the process of execution, Task 3 has been selected to continue running rather than switching to Task 1, which reduces overhead. This observation repeats again at the 82ms mark where usually in RM, Task 2 runs first but because of EDF scheduling, Task 3 is allowed to preempt Task 2 and at the 94ms, Task 3 is able to preempt Task 2 again. This saves 2 occurrences of overheads as seen by less switching to and from Task 3, which is more efficient than RM scheduling.

4.3 Handling Overloading

When there is an overload due to missed deadlines, the designed EDF scheduler continues to execute the set of tasks, that is, without stopping the execution process entirely. However, the overloaded task was handled with caution to achieve continuous execution of the task set, which will be explained below.

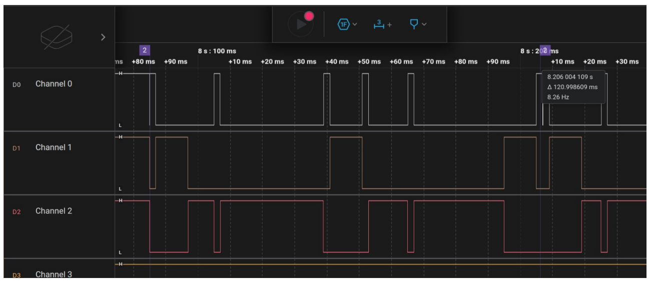

8.206 004 109 s
Δ 120.998609 ms
8.26 Hz

Figure 11: EDF - Utilisation > 1

As shown by Task set A in Table 1, processor utilisation was calculated and proven to be more than 1.

After coding on the STM32 microcontroller, Figure 11 presents the actual output of the logic analyser. When compared with utilisation < 1, the processor utilisation of more than 1 will show that Task 1 will overrun by 1ms after the 120ms hyper-period. This is expected as Task set A is not feasible to be scheduled with EDF when the processor utilisation is more than 1.

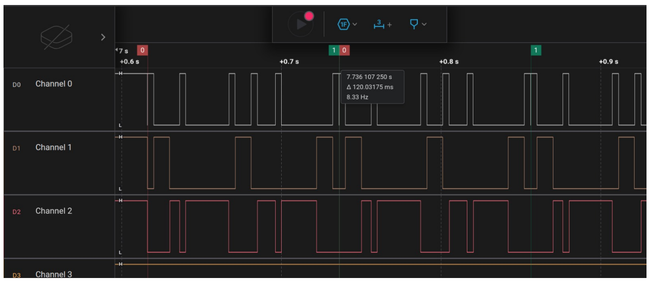

7.736 107 250 s
Δ 120.03175 ms
8.33 Hz

Figure 12: EDF - Utilisation = 1

As shown by Task set B in Table 2, processor utilisation was calculated and proven to be 1.

After coding on the STM32 microcontroller, Figure 12 presents the actual output of the logic analyser. Compared to utilisation < 1, the result shows that it meets the theoretical expectation of full processor utilisation of 1 and also the theoretical expectation of EDF scheduling. By meeting both theoretical expectations, this means that none of the theoretical expectations are defied, unlike RM processor has static scheduling rules that make Task set B unschedulable and resort to squeezing remaining tasks into all idle time slots.

However, a thorough of the graph output has shown that the end of the hyper-period is at 120.03175 ms. Although 0.03 ms is negligible since the subsequent tasks within each subsequent hyper-period are repeated, this is still worth noting as such small overloads may compound over time and have serious ramifications in real-world scenarios like aeroplanes that cannot allow such overload to affect its flight safety operations.

5 Conclusion

In conclusion, this lab assignment was to learn the concept of RM and EDF scheduling in Real-Time Systems with the aid of actual hardware, a STM32F4 microcontroller and a logic analyser. By studying the logic analyser output, one can glean the detailed differences between both scheduling algorithms and have a deeper analysis of the benefits of EDF over RM, especially in numerous real-world situations where scenarios are mostly dynamic i.e. automobiles, aircraft and even smart-home sensing.

References

[1] Buttazzo, Giorgio C. $https://doi.org/10.1023/b:time$.0000048932.30002.$d9$ Rate Monotonic vs. EDF: Judgment Day. Real-Time Systems, vol. 29, no. 1, Jan. 2005, pp. 5–26. (accessed Dec. 07, 2023)