ECE 4550/5550G Project 2

Milestone Deadline: Thursday March 23, 2023 at 11:59PM in Canvas (50 Points)
Final Deadline: Thursday March 30, 2023 at 11:59PM in Canvas (250 Points)

Objective

To design, implement, and measure the performance of the rate monotonic (RM) and deadline monotonic scheduling algorithms.

1 Project Overview

In this project, you will be implementing the rate monotonic (RM) and deadline monotonic (DM) scheduling algorithms, as discussed in class. You will also measure the overhead of these scheduling algorithms and its impact on timeliness.

2 Tasks & Rubrics

Please see the next section for some hints and notes.

Milestone (50 points): Please submit (a preliminary version of) tasks 1 and 2 below before Thursday March 23, 2023. You may continue modifying the code of these two tasks until the final deadline, but the submitted version for milestone 1 should be reasonably working.

Final (250 points): Please submit the final versions of all tasks before Thursday March 30, 2023. The final versions of tasks 1 and 2 can be different from the ones submitted earlier at the milestone.

1. (70 points) Implement the RM scheduling algorithm with the following requirements:
   - Your algorithm does not need to perform any schedulability test.
   - In the event that a job misses its deadline, it should be suspended and run as a new job at its next release time. In other words, the current (missed) instance of the task is dropped.
   - Your code should print out relevant information to the serial monitor, e.g., which task is executing, if a task misses its deadline, etc. The easier you make it for the GTA to grade your work, the better!

   Note that if a job executes for more than its worst-case execution time (i.e., an execution overrun occurs), it is suspended until its next release time. Again, conceptually, this overrun job is dropped. We have already implemented this part for you.

   What to include in your report: A description of your RM implementation. This description should be detailed enough that a competent peer of yours can follow along and implement your work.

2. (70 points) Implement the DM scheduling algorithm with the same requirements discussed above.

   What to include in your report: A description of your DM implementation.

3. (60 points) Test your RM and DM scheduling algorithms using the following two task sets, as shown in Tables 1–2. The time units are in milliseconds. All task instances will require their worst-case execution times. When constructing the task set, make sure that each task parameter is a multiple of your scheduler task's period (defined as `schedSCHEDULER_TASK_PERIOD` in `scheduler.h`).

Table 1: Task set #1

Task	C	D	T
τ_1	100	400	400
τ_2	200	700	800
τ_3	150	1000	1000
τ_4	300	5000	5000

Table 2: Task set #2

Task	C	D	T
τ_1	100	400	400
τ_2	150	200	500
τ_3	200	700	800
τ_4	150	1000	1000

What to include in your report: In 1-3 paragraphs, compare and contrast the performance of RM against that of DM when used to schedule these task sets.

4. (50 points) For each of the algorithms, augment your scheduler task to account for scheduler overhead. Rerun the task sets above.

What to include in your report: Describe your approach, findings and observations in 1-2 paragraphs.

3 Hints and Notes

- Note that you are **NOT** required to use (all of) the code provided. As long as your code works and you document your implementation, you are good to go. That said, if you do decide to work with the provided code, below are some information that you may find useful.
- The key to implementing RM in FreeRTOS is determining how to update the priority of a task/job. You can debug/test your code with an arbitrary task set or you can use the ones given above. In the provided code, search for "/* your implementation goes here */" to see where you might need to add code. Of course, depending on your implementation, some parts may not be necessarily.
- There are several FreeRTOS APIs that you may find helpful. See: `https://www.freertos.org/a00021.html` and `https://www.freertos.org/a00112.html`.
- Use the serial port to better understand/debug code inside `scheduler.cpp`. However, since the print buffer capacity is limited, use `Serial.println()` judiciously.
- The scheduler task will need to check for timely completion/deadline misses. Set the priority of this task accordingly.
- One task structure that will be useful is `xExtended_TCB` in `scheduler.cpp`. Feel free to add members to `xExtended_TCB` to help with maintaining a list of per-task parameters.

- One function that will be useful to you is `vApplicationTickHook()`, which can be found in `FreeRTOS/src/tasks.c`. This is a user-defined function that is called at every system tick. Your job is to somehow integrate this function with the scheduler task (and rest of the given code in `scheduler.cpp`) to formally implement RM. Ideally, you should not do heavy computation inside `vApplicationTickHook()` since it works within the interrupt context. Note that you will also need to implement all the supplied stub functions in `scheduler.cpp` to make it work.

4 Turning In Your Work

You are to upload the following documents on Canvas.

- Your report in PDF format.
- Your RM and DM code.

At the beginning of your report, please include the following statement along with your signature.

`‘‘I have neither given nor received unauthorized assistance on this assignment.’’`

Please remember that you must turn in your own work. Failure to do so will result in you being reported to the university.