ECE 4550/5550G Project 2

Due Thursday March 2, 2023 at 11:59PM in Canvas (200 Points)

Objective

To become familiar with the scheduling infrastructure to be used with FreeRTOS.

1 Project Overview

In Project 1, you learned that FreeRTOS uses a simple priority-based scheduling algorithm. In this project, you will get yourself familiarized with the scheduling infrastructure that is more general. You will leverage knowledge acquired in this project in future projects.

Getting a good understanding of the existing code and (later) creating a new scheduler can be a lot of work and may seem intimidating, but follow along, as we will provide you with starter code and plenty of hints along the way. Besides, you will feel great when you are done!

2 Getting Started

From Canvas, download the zip file. Inside, you will see a set of files (`project2.ino`, `scheduler.cpp` and `scheduler.h`). Move `scheduler.cpp` and `scheduler.h` inside the `FreeRTOS` folder under `Arduino/Libraries/` (this folder is usually located inside your `Documents` or `My Documents` folder).

We will first look at `project2.ino`. This file shows how to create two tasks bodies (`testFunc1` and `testFunc2`). Then, these task bodies are used to create two periodic tasks using `vSchedulerPeriodicTaskCreate()`. Note that before tasks can be created, we need to initialize the scheduler, which itself is a task, using the method `vSchedulerInit()`. Once all the tasks have been created, the system is started and run by calling `vSchedulerStart()`. All of the methods called inside `project2.ino` are implemented in `scheduler.cpp`.

Inside `scheduler.cpp`, one important definition is the extended task control block (`xExtended_TCB`). This is where the attributes of a given periodic task are defined. (You may also add to this TCB as needed.) The method `vSchedulerInit()` initializes the `xExtended_TCB` array. Then, `vSchedulerPeriodicTaskCreate()` adds tasks to said array. We start scheduling tasks with `vSchedulerStart()` where we set the scheduling policy used. Since we're using the TCB array, `schedUSE_TCB_ARRAY = 1`.

At every system tick, the method `vApplicationTickHook( void )` is called. Note that while it would be possible to put the scheduler code inside this method, doing so may be too costly as `vApplicationTickHook( void )` is called *every* tick. We overcome this challenge by implementing the scheduler as a separate task (see `prvSchedulerFunction( void *pvParameters )`).

3 Tasks & Rubrics

1. (100 points) Go through `scheduler.cpp` and (1) give a brief description of each function, and (2) create a flowchart depicting the relationship among the different functions. For example, if

`function A` calls `function B`, you should draw two boxes, one for `function A` and another for `function B`, with a directional arrow going from `function A` to `function B`. More introduction to flowchart can be found at `https://en.wikipedia.org/wiki/Flowchart`.

The purpose of this exercise is for you to really understand the code inside `scheduler.cpp` before you start coding.

2. (100 points) The file `scheduler.cpp` contains the basics which allow you to implement the RM (or RMS) scheduling algorithm. Inside `scheduler.cpp`, there are 20 instances where the following sentence appears:

   ```
   /* your implementation goes here */
   ```

   In your own words, describe what should go there, e.g., initilize an array, check for deadline misses, etc. Note that you do not have to write actual code at this time. That is your task for Project 3. That said, the more details you put down, the easier it will be for you later. Your answer should have the following format:

   ```
   Line 137: Search for unicorns using the RAINBOW data structure.
   ```

   A few of hints on RM implementation that may help you here:

   - The key to implementing RM is determining how to update the priority of a task/job. Think a bit about how you can do this.
   - The scheduler task will need to check for timely completion/deadline misses.
   - 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.
   - 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, by the way.
   - Depending on how you design your code, you may not need to fill in all 20 instances of the fill-in-the-blank parts. This is OK as long as your design achieves the correct functionality for RM.

4 Turning In Your Work

You are to upload your report in PDF on Canvas. 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.