ECE 4550/5550G Project 4

Due on Thursday April 20, 2023 at 11:59PM in Canvas (200 Points)

Objective

To implement resource sharing protocols in FreeRTOS.

1 Project Overview

In Project 3, you learned how to implement the rate-monotonic algorithm (RM) in FreeRTOS. For this project, you will leverage your RM implementation to support the original ceiling priority protocol (OCPP) and immediate ceiling priority protocol (ICPP) in FreeRTOS.

As in Project 3, you will validate your implementation through a systemic debugger (in our case the serial port communication).

2 Notes on Implementation of OCPP and ICPP

In this project, you will implement OCPP and ICPP for RM (hereafter called RM-OCPP and RM-ICPP respectively) in FreeRTOS. We have discussed both protocols in class, but addition details can be found in the textbook: OCPP is documented in Section 7.7 under the name of Priority Ceiling Protocol, and ICPP is described in Section 7.5 under the name of Highest Locker Priority Protocol.

This project requires a working implementation for rate monotonic (RM) scheduling. We provide one in the starter code, but you may use your own solution from project 3.

3 Tasks & Rubrics

1. (100 points) Implement the RM-OCPP scheduling algorithm with the following requirements:
   - Your algorithm does not need to perform any schedulability test. RM-OCPP should follow all the rubrics as listed under RM scheduling in Project 3.
   - Moreover, in the event that a job fails to acquire the resource within its worst-case execution time, 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.
   - Use the `C Structs` to define your shared resources. You are free to use `Struct` members that will record information pertaining to tasks that acquire it.
   - Your code should print out relevant information to the serial monitor, e.g., which task is executing, when it is executing which of its critical sections, if a task misses its deadline, etc.

   The key to implementing RM-OCPP in FreeRTOS is tracking the highest priority ceiling among all the locked resource(s) on the fly. You can debug/test your code with an arbitrary task set of your choice. For example, the task set from homework 4, question 1.

   There are several FreeRTOS APIs that you may find helpful:

- Your first job is to explore how FreeRTOS implements priority inheritance. Consult `semphr.h`, `queue.c`, and `queue.h` in `FreeRTOS/src` to further understand how that works.
- `xSemaphoreCreateMutex` creates a mutex, and returns a handle by which the created mutex can be referenced. Mutexes are acquired using `xSemaphoreTake`, and released using `xSemaphoreGive`. `SemaphoreHandle_t` stores the handle of a mutex/semaphore. (Hint: Mutexes and binary semaphores are very similar but have some subtle differences).
- `xTaskGetHandle` is used to look up the handle of a task from the task's name. Similarly, `xSemaphoreGetMutexHolder` returns the handle of the task that holds the mutex specified by the function parameter.
- `vTaskGetInfo` populates a `TaskStatus_t` structure for just a single task. The `TaskStatus_t` structure contains, among other things, the members for the task handle, task name, task priority, task state, and the total amount of execution time consumed by the task. (Hint: Each task has a member `uxBasePriority` in `FreeRTOS/src/task.h`. Check what is it used for.)
- `uxTaskPriorityGet/vTaskPrioritySet` obtains/sets the priority of a task.

What to include in your report: A description of your RM-OCPP implementation. This description should be detailed enough that a competent peer of yours can follow along and implement your work. In addition, you should include justifications as to why your implementation is correct.

2. (100 points) Implement the RM-ICPP scheduling algorithm with the same requirements as RM-OCPP (see above). Since RM-ICPP is an improvement over RM-ICPP, it follows similar pattern of execution, with specific changes to decision making while acquiring and releasing the mutex. The key to implementing RM-ICPP in FreeRTOS is to change the priority of the critical section to the priority ceiling of the resource, as soon as the task enters the critical section.

   What to include in your report: A description of your RM-ICPP implementation. You do not need to repeat information already described under your RM-OCPP implementation. Again, you should include justifications as to why your implementation is correct.

4 Turning In Your Work

You are to upload the following documents on Canvas.

- Your report in PDF format.
- Your RM-OCPP and RM-ICPP 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.