Assignment 2:
Priority Inversion and Monitoring

1DT056: Programming Embedded Systems
Uppsala University

January 30, 2015

You can achieve a maximum number of 20 points in this assignment. 8 out of 20 points are required to pass the assignment. You will need to achieve a minimum of 60 points across all six assignments. Earning 80/90/100/110/120 points will earn you 1/2/3/4/5 points in the final exam. Assignments are to be solved by students *individually*.

Exercise 1 Resolving Priority Inversion Problem (12p)

First part of this exercise is to implement three tasks that will suffer from the priority inversion problem. The processing using the shared resource is not relevant and should be abstracted away, e.g., by using a loop with empty processing. You should use a binary semaphore to ensure mutual exclusion over the shared resource (Note: do **NOT** use a mutex, because in that case the FreeRTOS will solve the problem automatically using the priority inheritance algorithm). In order to set up the priority inversion problem, use delays before the periodic parts of the task.

Next, you will solve the problem by implementing a priority boosting algorithm which will process the attempts to access the shared resource in a first-come, first-served fashion (regardless of their priorities in the system). Then you need to encapsulate the calls to semaphore give and take in your own functions in order to include this additional processing. Whenever a task tries to take the semaphore it should be put into a pool of tasks waiting on the resource. Priority of the tasks already in the pool should be boosted up at need, using the *vTaskPrioritySet*(). When a task is done with the shared resource its priority is returned to normal. Think of whether you need to use critical sections somewhere (there are macros in the FreeRTOS which can be used if needed). For simplicity reasons assume that each task has at most one shared resource. The first-come, first-served processing requires an array which adds elements to the end and removes them from the beginning, there is no need to use a FreeRTOS queue or a queue as you implemented in the Assignment 1.

Exercise 2 Implementing a Utilization Monitor (8p)

In this exercise your task is to implement a monitor task that ideally should be executed at the beginning of each time slice and determine the utilization factor during the previous time slice. If the utilization factor is too high it should give warning. To measure idle time you should use the Idle task. In order to set up this monitor you should first performa a one time estimate how many idle operations can be performed in a time slice. This estimate should be used as a reference point by your monitor. The monitor task should print a warning on the output if the utilization factor is higher than 90%.

Submission

Solutions to this assignment are to be submitted
via Student Portal at latest on **Friday, February 13th, 2015**.