EGH456 Embedded Systems

Laboratory Exercise 6

Priority Inversion

Learning Objectives

After completing this laboratory you should be able to:

1. Implement and schedule tasks with different priorities using FreeRTOS.
2. Investigate the priority inversion problem in real-time systems.
3. Use semaphores to manage access to shared resources between tasks.
4. Analyse task execution timing and the effects of preemption.
5. Apply priority inheritance via mutexes to resolve priority inversion.

Reference Documents

- FreeRTOS Reference Manual
- TivaWare Peripheral Driver Library User's Guide
- EK-TM4C1294XL User's Guide
- TM4C1294NCPDT Datasheet
- OPT3001 Ambient Light Sensor Datasheet
- BOOSTXL-SENSORS Sensors BoosterPack User's Guide

Preparation Activities

Read about the priority inversion problem from Lecture 10 notes (and reference book or www sources). Read the FreeRTOS user guide sections on semaphores.

Laboratory Tasks

Task A - Priority Inversion (4 Marks)

You are provided with an example project *priority_inversion* on canvas. This is a FreeRTOS program which has 3 tasks and is simulating priority inversion. Unzip the file and import the project into VSCode. Read through *main.c* and *led_task.c*. The code creates three tasks of different priority that do work on a shared resource that is protected by a semaphore. There are small delays in the medium and high priority tasks to ensure the low priority task runs first.

Check the messages printed via UART and find out how much time the high priority task took to go through its critical section (between entering and leaving).

1. Implement a method to count the number of system ticks taken by the high priority task within its critical section. How much time in seconds does it take to run the high priority task critical section? **(1 Mark)**
2. Which tasks are sharing the semaphore and which task pre-empts the low-priority task while it is holding the semaphore? **(1 Mark)**
3. Show your understanding of this problem by drawing a timing diagram highlighting each task and its status and when they switch contexts. See example diagrams in the lectures on the concurrent access model **(1 Mark)**
4. Which task is potentially at risk for missing a critical deadline and why? **(1 Mark)**

Task B - Priority Inheritance (4 Marks)

Edit the code to change the computations (iterations) in the medium priority task. Change the number of iterations in the medium priority task by factors 0.5, 2, 4, 8 and observe what happens to the time taken by the high priority task to complete its execution of the critical section. Complete the following tasks and answer the questions based on your observations

1. Does the time taken by the high-priority task depend on the computation time for the low or medium priority task? Why? **(1 Mark)**

Edit the code to create a mutex instead of a binary semaphore via the function *xSemaphoreCreateMutex()*. Repeat the same changes to the number of iterations as before and note the times that the high priority task spends in the critical section – first iteration and in subsequent iterations. Note the messages printed out that show the sequence of execution. Read about the priority inheritance protocol from lecture notes and the reference book. Find out how the use of mutex provides a solution to the priority inversion problem.

3. Show your results and show if more time was spent in the high priority critical section with the semaphore or with the mutex. Explain why this is the case? **(1 Mark)**

Edit the code to print to the serial terminal the current priority of the low task before, during and after it completes its critical section.

4. Show the output and explain what you observed. Which task is the priority inherited from? **(1 Mark)**

5. Why does the medium priority task not pre-empt the high priority task when a mutex is used as it did when a semaphore was used? **(1 Mark)**

Assessment

Show evidence of having performed the modifications and run the programs as required to the tutor. Answers to questions must also be written down and shown in class. You may work in a group of 2. Each member must be present in the laboratory at the workstation and be participating in the exercise to receive credit.