Mutexes and Semaphores for Avoiding Priority Inversion

COE718: Embedded Systems Design
Final Project

1. Introduction

This document contains the manual for COE718 course final project. Since we do not have access to hardware (physical board), the tasks in this final project are designed in a way to require minimum dependence on input and output to the board. However, the project will help you gain the knowledge about the real-time Operating System, ARM Cortex-M3, and uVision.

2. The Project Application

In this project, you will be creating an embedded application, which performs several tasks as described here and listed in Section 3. The application will have a main file that controls the execution of various multiple tasks. In the main function, the application should handle the processing of different tasks **based on the user input**. As we do not have access to the hardware board, you will be simulating the joystick input, as you have explored in the previous labs.

Notes on User Inputs Methodology and Project Report:

- Your project application must ask for user input by displaying or printing the options the user has provided. The examples of options are:
    - Which task to be executed first ?
    - In which mode the user wants to execute a task. For instance, in case of task one, which version of the code to be executed ?

  Please be as creative as possible to get better marks. Moreover, navigating through different parts of your program depends on the user input, which should be enabled.
- You should use either polling or interrupt-based techniques. Explain which method you have used in your report and why.
- The project report must contain a block diagram of your application. Moreover, how each user input corresponds to a specific program to run? You can use any software available to draw the block diagram of your system's structure such as Microsoft Visio, Biorender, Draw.io or any other open software available.

3. Resource Management

In concurrent systems such as a multitasking operating system, simultaneous accessing of shared resources can result in unexpected system behaviors, erroneous output generation or system crashing. A shared resource can be thought of as a certain variable, which is accessed by several tasks. The parts of a program where such shared resources are accessed are known as **critical sections**. In order to prevent such cases, the critical sections of the program must be protected and there are many ways to do it.

3.1 Priority Inversion

An example of operating system behaving un-expectably is the **Priority Inversion**, and you have explored it in **Lab4**. Priority inversion is a common design error within an RTOS. An example of priority inversion occurs when a lower-priority thread or task (say P1) blocks a higher-priority thread/task (say P3) because P1 is accessing a **critical shared resource** before P3 can access the same **critical shared resource** to execute. Within this time, it is possible that a medium priority thread/task (say P2) preempts the lower priority thread or task, P1. Then, the medium priority thread is free to execute until it is blocked by the higher priority thread, where the high-priority thread is still waiting for the lower-priority thread to complete its access of **shared resource**. In this way, the high priority thread will continue to be blocked while a medium priority thread executes (causing the priority inversion).

3.2 Project Assignment -- Priority Inversion

In Lab4, you are introduced to **Signal and Wait** as well as **Mutexes** as solutions to prevent priority inversion type problems from happening. You were asked not to use the built-in **OS_MUT** function as it automatically eliminates the problem. By the end of Lab4, you must have implemented both the problem and solution within a single C code. **In this course project,** you are required to work with mutexes and semaphores as the other ways of preventing the priority inversion or similar problems. More specifically, you must implement the following as part of your project:

- **Mutexes**: Create a new version of your Lab4 code where you use specific mutex functions to prevent priority inversion. In your report, explain how the built-in mutex functions work. For each mutex related function you employ, explain how it works and what parameters it receives, etc.

- **Semaphores**: Then create a 2nd version of your code where you use specific semaphore functions (**osSemaphore** related) available in RTX to prevent the priority inversion from happening. Set breakpoints in different threads; run your code and observe the state of the threads when each breakpoint is reached. In your report, explain how the built-in semaphore

functions work. For each semaphore related function you use, explain how it works and what parameters it receives, etc.

- **Resemble Semaphores**: Once you're familiar with semaphores and how they work, try to simulate them in your code and prevent the priority inversion from happening by implementing (coding) the inner working of semaphores, without using the built-in semaphore related functions of RTX.

4. Joinable Threads

When multithreading, a certain job can be divided into several threads, so that they can run concurrently, or in parallel when we have multiple CPU cores available. We have learned how to do multithreading in **Lab3**. The thread management of RTX operating system allows joining of a thread to another thread. It means that the threads waits until all the joined threads are terminated. In doing so, the thread which tries to join to other threads enters a waiting period until the joined threads are terminated.

4.1 Project Assignment -- Joinable Threads

You are required to create three threads:

- **Thread_1** is responsible for turning on certain LEDs. This thread joins Thread_2 and **Thread_3**, and based on their output, it turns on different LEDs.
- **Thread_2** and **Thread_3** perform some creative calculation and return an output.

Use the event viewer and performance analyzer to monitor the thread behaviors. Take some useful screenshots and include them in your report. Identify the thread scheduling algorithm you have used. How using different scheduling algorithms might affect the functionality of your program? In your project report, include as much elaboration and illustration as necessary to demonstrate the function of joinable threads.

5. Project Design, Implementation and Submission

The project combines three separate components: a written component, a demonstration/presentation component, and an oral component. All projects are to be completed individually. Project will be assessed as per following schedule:

- Interim Report. Week **10.**
- Final demonstration. Weeks **12** and **13** during your Lab sessions.
- Final project report End of Week **13**.

5.1 Project Summary, Interim Report, Demo, Coding and Final Report

Project summary (One page) carries 3% marks
Interim Report: It will help the students to start on the final report. Interim report must have the objectives, block diagrams and/or flow charts, progress made and future plans for the project. Interim report carries 12% of the project marks.
Final Demo: It carries 25% of the project marks.
Code: Project code should be explained and documented well. It carries 35% of the project marks. Code source files can be submitted as Appendix in the Final report.
Final Report: It carries 25% of the project marks.

Final report of the project should be of 10-15 pages with the following IEEE like format.

i. The report must be typed and have some Figures and/or drawings of your own.
ii. Avoid Cut and paste of Figures from other papers or manuals.
iii. A suitable Font (Bookman, Courier, Times New Roman) of size 11 or 12 points.
iv. Single line spacing.
v. Pages of letter size with 1.0" top, bottom, left and right margins.
vi. The report may have the following sections:
Introduction, Past Work or Review, Methodology, Design, Experimental Results, Conclusions, Reference, Appendix, etc.

Submit all the report files through D2L.

Useful Readings

[1] On resource sharing and semaphores: https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF3150/h03/annet/slides/semaphores.pdf
[2] CMSIS-RTOS (in uVision) semaphore related functions:
https://www.keil.com/pack/doc/cmsis/RTOS/html/group__CMSIS__RTOS__SemaphoreMgmt.html
[3] CMSIS-RTOS (in uVision) mutex related functions:
https://www.keil.com/pack/doc/cmsis/RTOS/html/group__CMSIS__RTOS__MutexMgmt.html
[4] On CMSIS-RTOS thread management:
https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__ThreadMgmt.html