COE 4DS4 – Lab #6 Report

Group Number: 33
Moshiur Howlader / 001316948 / howlam@mcmaster.ca
Ryan Ganeshan / 001322407 / ganesr3@mcmaster.ca
March 7th, 2018

Exercise 1:

This exercise was built off of experiment 2. The two periodic tasks task 2 and 3 were deleted. We have removed custom_taskdel, and custom_task_create. We use the custom_scheduler function to create the two periodic task, and initialize the appropriate execution_time and idle_time for the two tasks. Task 1 was assigned higher priority than task 2. Macros used to determine the scheduling protocol (preemptive or nonpreemptive scheduling) and the OSTimeGet() print statements were added into the two periodic tasks. As expected the nonpreemptive run of the program waits for a given running task to be completed before the next task is run. For the preemptive case, we notice that there are instances of task 1 with higher priority to halt task 2.

Exercise 2:

This exercise was completed by modifying experiment 2. The experiment was modified as described by the following: First, we removed the code that initially created tasks and the code that marked them for deletion. Then we heavily modified the task creation function. It now operates as a task handling function. We implemented a system to keep track of multiple button presses by saving the OS time at the press in a 4 index array, where each index corresponds to a button. If multiple buttons were pressed, then we service the button press for an individual button that was furthest away in time FIRST and work our way down from there. We also had a mechanism to detect whether a button was pressed multiple times before creation, indicating that it should not be created with the highest priority at the next run of the scheduler, but instead, at the lowest priority when the associated task is handled. Thus, in our handler, there were 3 cases that would be serviced: 1) the edge case when a task is newly created but should be at lowest priority (ie button is pressed multiple times before its creation), 2) New task is created with Highest Priority, and 3) A task is given lowest priority. This method of implementation does not limit the amount of button presses between runs, but was a little tricky to implement. We got it to a 100% working condition using our test cases.

Exercise 3:

Four mutexes were created in order to manage the resource managements of the 4 peripherals grpA_SW, grpB_SW, red_LED, and green_LED.
Below are the results for the three test cases when SW 0, 1, 15, and 16 are on.
i) greenLED 0 and 1 turns on, then redLED 0 and 1 turns on. Then when PB 2 is pressed, greenLED 6 and 7 stays on for 1.45 second, while redLED 0 and 1 wait for greenLED 6 and 7 to turn off before turning itself off. Then after that, redLED 6 and 7 turns on for 1.25 seconds.
ii) redLED 6 and 7 turns on for 1.25 seconds, then greenLED 6 and 7 turns on for 1.45 seconds, then redLED 0 and 1 turns on for 1.45 seconds, then greenLED 0 and 1 turns on for 1.65 seconds.
iii) redLED 0 and 1 turns on for 1.45 seconds, then redLED 6 and 7 turns on for 1.25 seconds, then greenLED 0 and 1 turns on for 1.5 seconds, then greenLED 6 and 7 light up time of 1.45 seconds, then greenLED 0 and 1 turns off after 0.15 sec.