Lab 3 Report

Erik Alm, Mario Rios, Group 8

2016 04 19

1 A, Delaying task without sleeping

The main goal of this section is getting familiar with busy waiting. We implemented a function that performs the busy waiting. The way this is done, is first getting a time stamp, and then getting into a while loop. In each loop iteration we update the current time, and we leave the loop when the difference between the initial time stamp and the current time is equal or bigger than the waiting time specified.

2 B1

No matter the order of creation of the threads, the result is the same. We observe that both leds are propertly lit, but they present some flickering. We interpret this flickering as a consequence of both threads sharing the cpu (hence the actual period can't be perfect).

3 B2

Executing alwasy thread 1 first, and then thread2, we observe different behaviours depending on the priority of the threads. With thread 1 having low priority, the result is: Led 1 is lit for approximately a second, (while led 2 is off), and then led 2 gets lit, and after that, both leds are propertly lit, having some flickering, and periodically blinking. The reason of this behaviour, is that once thread 1 is created, it gets executed (preempting main thread), therefore thread 2 is not created yet. Only after the main thread has gets cpu time again, thread 2 can be created, and from this point normal execution proceeds. We belive the blinking efect is due to main thread being periodically resumed, thus kicking out both thread 1 and 2 from execution.

However, when we run thread 1 with high priority and thread 2 with low priority, only led 1 gets lit. This is due to thread 1 not doing actual sleeping (getting CPU usage 100% of the time). Since thread 2 has lower priority than thread 1, it never gets the chance to kick thread 1 out of execution.

4 B3

With thread 1 at high priority, we observe that both leds are properly lit. The reason of this working more smoothly than the previous version, is that both threads are eventually sleeping (not only doing busy waiting). However, when thread 2 has high priority, we notice that sometimes, led 1 is shining at 100% intensity (meaning thread 1 is stopped). This makes us think that the best strategy is to have the task with shortest period to have higher priority.

5 C

We have three tasks, one for capturing commands, with the lowest priority setting, and two for modifying the led status, with higher priority than task 1 (but having both the same priority level). In order to ensure data consistency, we are using mutex for shared memory protection. Task 1 (command control) only gets the lock when is about to write to the shared variable, but the other two tasks, grab the lock when they are about to read the shared variable, and free once they have updated the command status.