LAB #6: Threads

Lab writeup is due to your TA at the beginning of your next scheduled lab. Don’t put this off to the last minute! There is pre-lab work to complete before the start of the next lab. **NO LATE LAB REPORTS WILL BE ACCEPTED.**

1 Objectives

- To gain experience with preemptive schedulers and semaphores.

2 Reading

- Read Chapter 5 on threads

3 Tasks

In this lab, you are going to experiment with a thread scheduler and using a semaphore to lock access to an I/O port. In particular, you are to write a simple thread which counts to $FFFF. When it wraps around, it should then try to obtain a semaphore, $S$. After it obtains the semaphore, the thread should then write its ID (stored in its thread control block) to PORTC. It then should wait for a delay. This delay should be implemented with one of the hardware counters rather than a software counter (for example, you can add $1FFF to TCNT and wait for TCNT to reach this value). In this way, other threads may be executing as long as they do not wish to access PORTC. After the delay has completed, the thread should set PORTC back to 0, and it should release the semaphore, $S$. It should then loop back and start counting again. In designing this thread, be aware that it must be reentrant, since many copies of this thread will be running at the same time.

Next, you should then implement a thread scheduler, and create thread control blocks for at least three copies of your simple counting thread (try to put in as many copies as your memory can handle). Finally, run your program and watch PORTC on the oscilloscope. This will allow you to see which thread is in its critical section. Ideally, you would like one thread to always be in its critical section. While you can start with the simple thread scheduler from the book, feel free to be creative to try to optimize performance. For example, having threads which are waiting for their display delay time out could release the CPU. Or having blocked threads release the CPU. If you are feeling very adventurous, you can try to use blocked lists, but this is not required.

4 Prelab

1. You should have your simple thread written and running in one of the simulators.

2. You should have at least a basic version of your thread scheduler written though it need not be thoroughly debugged.

5 Lab Tasks

1. Complete your thread scheduler and watch it execute using an oscilloscope on PORTC.
2. Attempt to optimize your scheduler to improve the proportion of time that threads are in their critical section.

6 Writeup

Include the following items. In this lab, only one writeup per team is required.

1. A printout of your assembly code.
2. Annotated scope output for PORTC.
3. Did you have any problems? For example, did you have any problems in how you implemented your display delay?
4. What percentage of the time do your threads spend in their critical section?
5. What types of things did you try to achieve this level of efficiency?