EECS 678 - Lab 06

Theodore Lindsey

April 24, 2015

`http://people.eecs.ku.edu/~vvivekan/lab/pthreads_pc/pthreads_pc.html`

1. *Briefly describe the problem you are trying to solve and describe the design of your solution.*

   We want to allow a consumer and producer to run concurrently rather than the producer first followed by the consumer. We want the consumer to not consume a item that hasn't been produced yet and we want the data buffer to not be overflowed (if full) by the producer. In order to achieve this, we use an item queue and we track information regarding if the queue is empty and if the queue is full. We protect each with a pthread mutual exclusion lock so that the consumer must wait until the producer threads are ready and vice-versa.

2. *Discuss why the busy-wait solution is inefficient for threads running on the same CPU, even though it produces the "desired behavior".*

   The busy-wait solution forces threads to spin while they wait for the conditions necessary for them to continue with their action to be met (either queue sitting at EMPTY and the producer waiting to kick in or the queue sitting at FULL and the consumer waiting to kick in). Lots of CPU cycles are wasted with this method.

3. *Why are you confident your solution is correct? You will need to argue from your narrated output as to why your solution is correct. Note, your output will likely not match the output listed here exactly. Two successive runs of your application will probably not match even vaguely, due to random variations in how threads are scheduled. However, your report should discuss each of the following points and discuss how your output supports your discussion of each:*

   - *Are each producer and consumer operating on a unique item? Is each item both produced and consumed?*
   - *Do the producers produce the correct number of items and the consumers consume the correct number of items?*
   - *Does each thread exit appropriately?*
   - *Does your solution pass the above tests with the different numbers of producer and consumer threads in the Makfile tests?*

   The critical section is protected. This means that when pulling from the queue or adding to the queue, the queue and the full and empty tracking is protected. We see that this is the case because in our narratives, no producer processed the same integer as any other producer and no consumer processed the same integer as any other consumer.

   Again, because we lock the critical sections, we can be sure that all involved parties are working with accurate counts of processed items. We see this reflected in fact that never in the narratives was WORK_MAX exceeded.

   Each thread exits correctly, given that the program as a whole terminates cleanly and with the anticipated output. In the case of each test (as evidenced by the narratives), the correct number of consumers are used and in each case, each one has produced or consumed unique values.