CPSC 213: Assignment 8

Due: UPDATED: Sunday March 27, 2011

Goal

In this assignment we extend the uthread.c package to include synchronization. The new version of uthread.c includes a complete implementation of Spinlocks and Semaphores and a partial implementation of Monitors and Condition Variables. You will complete the implementation and then use these primitives to solve a few problems.

Notes

The uthreads package runs on Intel x86 machines running Linux, MacOS or Cygwin. You can use the department linux machines by connecting to lulu.ugrad.cs.ubc.ca.

To compile on Linux or Cygwin it is necessary to explicitly include the pthread library by adding “-lpthread” to the gcc command line.

Requirements

Here are the requirements for this week’s assignment.

1. Read and comment the implementations of Spinlocks and Monitors inside uthread.c.
2. Implement a version of a Monitor, called a Multi-Reader-Single-Writer Monitor. This Monitor has three operations: enter_for_reading, enter_for_writing, and exit. The monitor allows multiple threads to enter the monitor for reading, but only a single thread for writing.
3. Test your new monitor using a program with four reader threads and one writer thread competing to enter a single monitor. The monitor protects a counter. The writer increments the counter. The readers read and print the value of the counter. Each thread loops entering the monitor, exiting the monitor and then yielding with probability 0.5. Compute the probability using the UNIX random() operation (see its man page).
4. Implement condition variables. Their state is stored in the struct uthread_cv and they have four operations uthread_cv_create, uthread_cv_wait, uthread_cv_notify, and uthread_cv_notify_all.
5. Test your implementation using a single processor.
6. Implement a bounded queue of integers using a fixed-sized array. The queue will be shared among multiple threads and so must be synchronized; use Monitors. Since this queue is fixed sized, it is possible that an enqueue operation will find the queue full and thus have no room at add the new element; use a Condition Variable to block in this case until there is room for the new element. Similarly, a dequeue operation might find the queue empty; use a Condition Variable to block in this case until there is an element in the queue for dequeue to return.
7. Test your implementation using a single processor and four threads: two “producers” that loop enqueueing integers and two “consumers” that loop dequeueing integers and printing them. Create the threads is such a way that both producer threads and both consumer threads get to run during the test.

8. Test your implementation with 2 and 4 “processors” by changing the argument to uthread_init. If you can run this on a real multi-processor (e.g., a dual-core CPU) that is great. But, you can also run the multi-threaded version on a uniprocessor. In this case, the multiple kernel threads created in uthread_init will be multiplexed across the single processor by the operating system using its scheduling policy (i.e., preemptive, round-robin) which provides a sufficient emulation of a true multi-processor for testing purposes. Explain the differences you see among the two multi-processor executions and the uniprocessor execution from question 5.

9. Read and comment the implementation of Semaphores in uthread.c.

Material Provided

The files uthread.h, uthread.c, and racedemo.c are provided in the file code.zip.

What to Hand In

Use the handin program. The assignment directory is a8.

1. A version of uthread.c with comments for Spinlocks, Monitors, and Semaphores and with implementations of Condition Variables and Multiple-Reader, Single-Writer Monitors.
2. A program that tests Multiple-Reader, Single-Writer Monitors and a description of the results of your tests.
3. Your implementation of the bounded queue and an associated test program.