CPSC 213: Assignment 11

This assignment is provided for review and not to be handed in. Solutions will be posted in less than one week.

Goal

You have seen, in the uthread.c file, complete implementations of Spinlocks, Semaphores, Monitors and Condition Variables. You will now use these primitives to solve an interesting robot problem.

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 Cygqin it is necessary to explicitly include the pthread library by adding "-lpthread" to the gcc command line.

Requirements

Using any combination of Spinlocks, Monitors, Multi-Reader-Single-Writer Monitors, Condition Variables and Semaphores that you like, implement the following "Robot" program.

The Robot consists of a centralized decision-making engine, a set of sensors, and a set of actuators. The decision engine will be modeled by a binary search tree. Nodes of the tree have a key and a value, both are integers; keys range between 0 and 1000. Sensor's will be modeled by threads that generate random key-value pairs that are submitted to the decision engine where they are stored. Sensor updates to the same key are averaged. Actuators are modeled by threads that read the decision tree to determine the average value associated with all keys that fall within a randomly selected range. At random actuators will request that they wait until a sensor has updated at least one node in the range they select. The total number of actuator requests can not be permitted to exceed the total number of sensors updates by more than a fixed value. The program should terminate when either all sensors or all actuators have completed. The following are input parameters: the number of sensor threads, the number of actuator threads, the number of CPUs, the frequency at which actuators perform the wait-for-update version of their access, the maximum allowed delta between the number of sensor updates and actuator requests, and the number of iterations for each actuator and sensor thread.

Testing this code will present some challenges. I suggest the following steps.

a) Test a single-threaded sensor-only version to ensure that search-tree insertion is implemented correct. You may want to use non-random values to simplify testing.

b) Now added a single-threaded actuator phase to follow the sensor phase to ensure that reading works correctly.

c) Now, add whatever mutual exclusion primitives you plan to use and a test with multiple CPUs and multiple sensor threads, but no actuator threads. While its not the best general strategy, you can assume your code works if you can run it lots of threads, lots of iterations and a few CPUs.

d) Now run multiple actuators and sensors together.

e) Add the constraint that limits how far actuator requests can exceed sensor requests and use another synchronized mechanism to check this constraint.

f) Finally, add the ability for actuator threads to wait for sensor threads and test this by first controlling the sequence of sensor and actuator requests that are made using one thread. Then test more generally.

Material Provided

The files uthread.h, uthread.c can be found in the solution from assignment 10.