// Using Barriers
// George F. Riley, Georgia Tech, Spring 2012
// This illustrates how a barrier would be implemented (both a "buggy" version
// and a good one).

#include <iostream>
#include "gthread.h"
#include "math.h"
#include <sys/time.h>
#include "complex.h"
#include "InputImage.h"

using namespace std;

// Implement a "buggy" barrier for illustration
class BuggyBarrier {
public:
  BuggyBarrier(int P0); // P is the total number of threads
  void Enter(int); // Enter the barrier, don’t exit till a ll there
private:
  int P;
  int count; // Number of threads presently in the barrier
  int FetchAndIncrementCount();
  pthread_mutex_t countMutex;
};

BuggyBarrier::BuggyBarrier(int P0)
  : P(P0), count(0)
  {
    // Initialize the mutex used for FetchAndIncrement
    pthread_mutex_init(&countMutex, 0);
  }

void BuggyBarrier::Enter(int)
  {
    // This is buggy! Why?
    // Also, we include the "int" parameter, but it’s not neede for this
    // implementation. It is needed for the GoodBarrier, so we add a
    // dummy parameter to make switching between the good and buggy one
    // easier.
    int myCount = FetchAndIncrementCount();
    if (myCount == (P - 1))
      {
        // All threads have entered, reset count and exit
        count = 0;
      }
    else
      {
        // Spin until all threads entered
        while(count != 0) { } // Spin waiting for others
      }
  }

int BuggyBarrier::FetchAndIncrementCount()
  {
    // We don’t have an atomic FetchAndIncrement, but we can get the
    // same behavior by using a mutex

Program barriers.cc
Program barriers.cc (continued)
gthread_mutex_t printingMutex;

// BuggyBarrier, barrier;    // The barrier for thread coordinatino
GoodBarrier* barrier;    // The barrier for thread coordinatino

void MyThreadThreeArgs(int myId, int count1, int count2)
{
    for (int i = 0; i < count1; ++i)
    {
        LockMutex(printingMutex);
        cout << "Hello from thread " << myId << " count1 " << i << endl;
        UnlockMutex(printingMutex);
        // Use a random count2, so that each thread takes a different
        // amount of time. drand48() returns a random number between
        // zero and one.
        count2 = count2 * drand48();
        for (int j = 0; j < count2; ++j)
        {
        }
        // Here we want to wait for all other threads to get to this
        // point, hence we need a barrier.
        barrier->Enter(myId);
    }
    EndThread(); // Required by the GThreads library
}

int main( int argc, char** argv)
{
    if (argc < 4)
    {
        cout << "Usage: testGthread nThreads count1 count2" << endl;
        exit(1);
    }
    int nThreads = atol(argv[1]);
    int count1 = atol(argv[2]);
    int count2 = atol(argv[3]);
    // barrier = new BuggyBarrier(nThreads + 1);
    barrier = new GoodBarrier(nThreads);
    for (int i = 0; i < nThreads; ++i)
    {
        // Start each thread
        cout << "Creating thread " << i << endl;
        thread t = thread(MyThreadThreeArgs, i, count1, count2);
    }
    // Now wait for all to complete
    WaitAllThreads();
    cout << "Main exiting" << endl;
}