CS 360 Special Topics: Computer Networking

Spring 2007 **DUE: Feb 22, 2007**
Assignment #2 **15 points**

Introduction Multithreaded Programming

The focus of this assignment is to become familiar with the Pthreads library. The key components of this assignment are thread creation and the use of mutexes to guard access to data shared between two or more threads.

Multithreaded Random Numbers

This assignment consists of an application that contains four threads. One thread will randomly generate numbers, two threads (the even and the odd thread) will print numbers to the screen, and the final thread will act as a user interface thread.

The **random number thread** should generate a random number (between 0-1000, inclusive) every ¼ second and alternate between storing the generated number in a location where the even thread and odd thread can access it, respectively. The first random number should be stored in a location where the even thread can access it.

When **the even thread** executes, if its number is even the thread should: pause for ½ a second, increment the number by 2 and then print out the number (the number should be preceded by **one** tab and followed by a newline). No matter if the number is even or odd, this thread should then pause for ½ a second.

The **odd thread** works the same way, if its number is odd the thread should: pause for ½ a second, increment the number by 2 and then print out the number (the number should be preceded by **two** tabs and followed by a newline). No matter if the number is even or odd, this thread should then pause for ½ a second.

The **UI thread** should prompt the user for an integer to use as a seed for the random number generator and kick off all the other threads. The UI thread should then wait for the user to input the character 'q' to terminate the application. Each thread needs to be stopped gracefully and the application should not terminate until all the spawned threads have exited.

The *format* of the output from your program must look exactly the like sample output below, with some variation in the random numbers generated.

What and How to Submit

The source code should be well documented and conform to the Coding Standards. You must submit ONE .c file named prand.c, ONE Makefile that will build your executable, and the .svn directory for the project. The name of the makefile target for this executable should be prand, which should also be the name of the executable file. The Makefile should also build prand by default if no command line arguments are given to the `(g)make` command.

Also, answer the questions below and write up a small document (ASCII text) named answers.txt.

You will need to roll all THREE of these files (and one directory and some number of screenshots) into a gzipped tar file named **PUNetID_cs360s07_PA2.tar.gz** and submit that electronically using the submit script on **zeus** as detailed in lecture. You also need to turn in a hard copy of all THREE files at the beginning of class on Feb 22th. Both the electronic and paper submissions must be made by 1pm, Feb 22th. You do not need to print out the screenshots discussed below.

Questions

Modify your code to see what happens when the changes below are made. Be sure to submit your ORIGINAL, UNMODIFIEDcode. Illustrate your answers below with screenshots of your running program. These screen shots should be .png files included in the **PUNetID_cs360s07_PA2.tar.gz** file.

1. What happens if you remove the unconditional ½ second pause from both the even and odd thread?
2. What happens if you remove the conditional ½ second pause from both the even and odd thread?
3. How are the above two results the same or different, and why?
4. What happens if you remove all the pthread_mutex_lock/unlock calls from the even thread?
5. What happens if you remove the unconditional ½ second pause from only the odd thread?

Hints/Clarifications

You will need to look at the `man` pages for `pthread_create, pthread_join, usleep, sleep, pthread_mutex_[lock|unlock|init|destroy],` etc.

All of the print statements in your program should be `fprintf(stderr,...` statements.

The even thread should have its number initialized to 0 and the odd thread should have its number initialized to 1.

The srand() command should be called (once) from within the random number generating thread, the argument should be passed as the parameter to the function that starts the thread.

Neither the even nor odd thread should print anything until the user enters the random seed. The threads should be started in the following order: UI, even, odd, rand. The threads should be terminated in the order: even, odd, rand, UI. Note that when you enter `main()` you have one thread active.

You should use **ksnapshot** on the CS lab machines to take screenshots of your running code.

Any data accessed by more than one thread needs to be protected by a mutex.

Sample Output

```
[chadd@zeus]$ ./prand
Please enter the seed> 1
        2
                3
                523
        222
                971
        186
                889
q
Terminating...
[chadd@zeus]$
```

Extra Credit

+2: Create a make target called answers that will display the file answers.txt to the screen when `make answers` is run from the Linux command line.

Start on this today! Come see me with questions ASAP! Be prepared to demo your project in class!