Lab 5: Introduction to PThreads

This lab introduces basic synchronization concepts through the use of PThreads. We discuss basic usage of PThreads (creating, destroying, waiting on) and also illustrate (and debug) a simple race condition.

Lab Materials

1. Slides
2. Lab Files

Assignment

You need to complete the implementation of ptcount_atomic.c and ptcount_mutex.c. When you are finished, the main process of ptcount should create three pthreads and wait for these pthreads to complete execution. Each thread should increment (in a loop) a shared variable named count as well as a local counter. When the child threads are finished executing, the main thread should print out the value of count. The value reported by the main process should be consistent with what you would expect with the given loop bound and increment values. Your program should match the following output:

```
bash$ make test
./ptcount_mutex 100000000 1
Thread: 0 finished. Counted: 100000000
Thread: 1 finished. Counted: 100000000
Thread: 2 finished. Counted: 100000000
Main(): Waited on 3 threads. Final value of count = 300000000. Done.
./ptcount_atomic 100000000 1
Thread: 0 finished. Counted: 100000000
Thread: 1 finished. Counted: 100000000
Thread: 2 finished. Counted: 100000000
Main(): Waited on 3 threads. Final value of count = 300000000. Done.
```

After you have finished your implementation, go over the following questions (you don't need to turn them in, but you may be quizzed over them):

1. What accounts for the inconsistency of the final value of the count variable compared to the sum of the local counts for each thread in the version of your program that has no lock/unlock calls?
2. If you test the version of your program that has no lock/unlock operations with a smaller loop bound, there is often no inconsistency in the final value of count compared to when you use a larger loop bound. Why?
3. Why are the local variables that are printed out always consistent?
4. How does your solution ensure the final value of count will always be consistent (with any loop bound and increment values)?
5. Consider the two versions of your ptcount.c code. One with the lock and unlock operations, and one without. Run both with a loop count of 1 million, using the *time* time command: "bash> time ./ptcount 1000000 1". Real time is total time, User time is time spent in User Mode. SYS time is time spent in OS mode. User and SYS time will not add up to Real for various reasons that need not concern you at this time. Why do you think the times for the two versions of the program are so different?

You also need to archive your lab for submission. For this step, you should use the 'zip' target included in the lab's Makefile. Change the STUDENT_ID variable in the Makefile to your student ID and type:

```
make zip
```

Turn this zip file into Blackboard before the beginning of the next lab session.

< Back to the Lab Home Page