EECS 678 - Lab 05

Theodore Lindsey

April 24, 2015

`http://people.eecs.ku.edu/~vvivekan/lab/pthreads_intro/pthreads_intro.html`

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?*

   Several threads can update the value at the same time, eg

T1	T2	T2
load count		
	load count	
		load count
increment count		
	increment count	
		increment count
save count		
	save count	
		save count

   When each of the threads get around to saving their value of count, then they haven't taken into account that the value of count may have been updated by one of the other threads.

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?*

   Looping less gives less of a chance for threads to modify common resources between the time another thread loads and then saves that resource.

3. *Why are the local variables that are printed out always consistent?*

   The printed values are stored as a local variable and can't be modified by the other threads. There is no way for the local variable to be updated unless it is done so by the thread who owns it.

4. *How does your solution ensure the final value of count will always be consistent (with any loop bound and increment values)?*

   Only one thread has permission to modify the value of count at a time (via the use of a resource lock).

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 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?*

   Without lock:

```
$ ./ptcount 1000000 1
Thread: 1 finished. Counted: 1000000
Thread: 0 finished. Counted: 1000000
Thread: 2 finished. Counted: 1000000
Main(): Waited on 3 threads. Final value of count = 1151266. Done.

real    0m0.024s
user    0m0.052s
sys     0m0.001s
```

   With lock:

```
$ ./ptcount 1000000 1
Thread: 1 finished. Counted: 1000000
Thread: 0 finished. Counted: 1000000
Thread: 2 finished. Counted: 1000000
Main(): Waited on 3 threads. Final value of count = 3000000. Done.

real    0m0.369s
user    0m0.430s
sys     0m0.554s
```

   I suspect that the version without lock is able to run everything in parallel rather than having to wait for the other threads to unlock the shared resource. If the code is run on a multi-core machine, then each thread in the no-lock version could run on its own core at the same time without waiting for an unlock. In the case of the lock version, even though each thread has pleanty of computing resrouces, they still have to wait their turns.