EECS 678: Lab 5 - Introduction to pthreads

Gehrig Keane
2727430

Monday 7th March, 2016

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?*

Without a mutex structure in place the iteration within each loop operates on the global variable ***count*** without regard for concurrent modification of said variable. Therefore, during the lengthy concurrent ***for loop*** execution obfuscation is wholly expected. Specifically, after several million iterations we expect undefined behavior with concurrent modification of a shard value, such is the behavior of poorly handled race conditions. Additionally notice, the local variable within each thread ***loc*** increments successfully due to its encapsulation within each thread.

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?*

With small iteration values the chance of undefined race condition behavior decreases. If the concurrent threads are iterating ten times, collisions during the modification of ***count*** are very unlikely amidst the plethora of machine instructions. This phenomenon extends well into the thousands of ***for loop*** iterations.

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

As mentioned above local variables, akin to their nomenclature, reside within each thread respectively. Thereby isolating themselves from the troubles of concurrent thread execution. For three threads in our case, each thread increments its own version of the local variable ***loc***. Ergo, mutual exclusion is inherent for local variables such as these.

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

The trivial addition of ***pthread_mutex_lock*** preceding, and ***pthread_mutex_unlock*** post-ceding the incrementation of the global variable ***count*** allows safe manipulation of the global variable. Note, the placement of mutex signals within the ***for loop*** drastically impacts execution time. If we redefine the ***critical section*** and relocate the mutex signals such that they encompass each thread's ***for loop*** execution, we minimize excessive lock and unlock signals.

Mutex within ***for loop*** execution	Mutex encapsulating ***for loop*** execution
Thread: 2 finished. Elapsed Time: 845.145996	Thread: 0 finished. Elapsed Time: 3.939000
Thread: 1 finished. Elapsed Time: 854.382996	Thread: 1 finished. Elapsed Time: 10.54200
Thread: 0 finished. Elapsed Time: 859.919006	Thread: 2 finished. Elapsed Time: 16.21799

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?*

When considering the time differential between ***No Mutex*** and ***Mutex in for loop*** its apparent that the locking and unlocking of ***count*** incurs notable overhead. As mentioned above, we can remove a substantial amount of such overhead by moving the entire for loop within the protected section. Note, this method seemingly contradicts the purpose of multi-threading in that each thread basically executes their computations in serial. See timings below ("***Elapsed Time***" units are roughly milliseconds)

No Mutex

```
[gkeane@1005b-15 Lab5]$ time ./ptcount_raw 1000000 1
Thread: 0 finished. Counted: 1000000
Thread: 2 finished. Counted: 1000000
Thread: 1 finished. Counted: 1000000
Main(): Waited on 3 threads. Final value of count = 1430064. Done.

real 0m0.016s
user 0m0.034s
sys 0m0.001s
```

Mutex within ***for loop***

```
[gkeane@1005b-15 Lab5]$ time ./ptcount 1000000 1
Thread: 2 finished. Counted: 1000000 Elapsed Time: 894.069031
Thread: 1 finished. Counted: 1000000 Elapsed Time: 910.477966
Thread: 0 finished. Counted: 1000000 Elapsed Time: 911.804993
Main(): Waited on 3 threads. Final value of count = 3000000. Done.

real 0m0.324s
user 0m0.361s
sys 0m0.552s
```

Mutex encapsulating ***for loop***

```
[gkeane@1005b-15 Lab5]$ time ./ptcount 1000000 1
Thread: 0 finished. Counted: 1000000 Elapsed Time: 2.232000
Thread: 1 finished. Counted: 1000000 Elapsed Time: 4.420000
Thread: 2 finished. Counted: 1000000 Elapsed Time: 6.692000
Main(): Waited on 3 threads. Final value of count = 3000000. Done.

real 0m0.010s
user 0m0.007s
sys 0m0.001s
```