IC411 pthreads lab Name \_\_\_Brandon Sipes\_\_\_\_\_\_\_\_

Fall AY2019

**Summary**: In this lab, you'll practice using POSIX threads in C.

**Learning Objectives**:

- Understand how POSIX threads are created and terminated (rejoined)

**Estimated Completion Time**: 1.5 hours **Lab Total**: 15 points

**Submission**: Submit gonavy.c to the online submission site and a paper copy of these instructions with answers completed.

**Example Files**

counters.c

gonavy.c

**Tasks**:

Review the contents of counters.c. Examine the pthreads code. Understand how threads are created and terminated (joined).

Compile and run counters.c

gcc -o counters counters.c -lpthread

./counters

Note how the lpthread flag tells the compiler to link against the pthread library. This is necessary because it is not a 'standard' library. If you include this flag at the beginning, as in:

gcc -lpthread -o counters counters.c

it will cause a compiler error, because the source/target arguments are expected first.

(1) 1. Do the counter threads *execute* in strictly numerical order? Why or why not?

They do not because there is nothing telling the parent of the threads to wait for 1 to execute before creating another thread.

(1) 2. If you run the program repeatedly, does the order of thread execution sometimes change? Why or why not?

Yes, because there is nothing telling the parent of the threads to wait for 1 to execute before creating another thread. So occasionally one will execute before the other.

(1) 3. If one of these threads happened to block, would the other threads block? Why or why not? If you're not sure, uncomment the "lucky number" portion of thread\_function to experiment.

No. The other threads continue to execute. Because it is not pure ULT.

(1) 4. Based on your answer to question #3, is the pthreads implementation a 'pure ULT' strategy? Explain.

No. It is a hybrid of the ULT and KLT. This is why it does not block the other threads when one thread blocks.

Complete the gonavy.c code

The gonavy.c code is similar to the counters program. However, we need two sets of threads now: one set for Army, one set for Navy. The Army threads will each generate an individual 'score'. The corresponding Navy threads will each read the appropriate Army score from that array, double it, and save the 'Navy score' to the right place in that array. The Army score will be a random integer between 0 and 100. All scores are printed at the end.

Since the Army scores will be *written* by Army threads but *read* by Navy threads, we want to ensure proper correspondence. For example, an Army threads will put a score in element X of the army\_scores array, and its corresponding Navy thread will put its own score (which will be twice the Army score for that thread number) into element X of the navy\_scores array. Complete the code to make this happen.

Example output:

Final scores:

Thread:0 Navy:166, Army:83

Thread:1 Navy:172, Army:86

Thread:2 Navy:154, Army:77

Thread:3 Navy:42, Army:21

Thread:4 Navy:30, Army:15

Thread:5 Navy:98, Army:49

Thread:6 Navy:186, Army:93

Thread:7 Navy:172, Army:86

Thread:8 Navy:70, Army:35

Thread:9 Navy:184, Army:92

(1) 5. If one of these threads happened to block, but all other threads terminated normally, what would happen to the *main process* at the pthread\_join() loop? If unsure, try adding another "lucky number" section to one of the thread routines as an experiment.

It would continue to wait for the blocking thread but all the other threads would finish.

(1) 6. In our gonavy example, the Army threads all complete before the Navy threads are created, so concurrency is not a problem. Suppose we want to *interleave* execution of the Army and Navy threads -- create them all, let them all run, then terminate them at the same time. Would doing so risk generating an incorrect output sometimes? Why or why not?

It might create some incorrect outputs because there would be a race condition between when army score is set and when navy doubles that score.

(1) 7. Would it be possible to successfully and correctly interleave execution of all the Army and Navy threads using concurrency mechanisms? If so, describe in general terms how it might be done. If not, describe why not.

You could setup a locking mechanisms the stops other threads from executing while the lock is locked.

(+2) Extra credit. Re-code the lab so that execution of the Army and Navy threads is fully *interleaved* (all Army and Navy threads running at the same time) but executes properly, using semaphores, mutexes, or some other type of synchronization. If completed, submit to online site as gonavy\_interleaved.c