CS 6210 Pre-lab Questions
Alicia Hoag

The main function contains calls to exit() (line 66) and pthread_exit() (line 80). How will the effect of these two calls differ when they are executed?

The regular exit will remove all the resources that were created by main. This includes any shared memory between processes, and is generally not advisable when you're working with threads. Pthread_exit() leaves process-shared resources (eg mutexes), and after the last thread in a process terminates, it will call regular exit with a 0 status, so that the regular exit routines can be processed.

The main function calls pthread_join() (line 77) with the parameter thread_return. Where does the value stored in thread_return come from when the consumer_thread is joined?

The value stored in thread_return is assigned by the exit function to a field within the p_thread type structure for the thread we're exiting from. That structure is keyed to a PID, so we will be able to find it that way (or if the process asking for the join is the parent by using the variable), and pull out the return value.

Where does the value stored in thread_return come from if the joined thread terminated by calling pthread_exit instead of finishing normally? On the same call to pthread_join() (line 77), what will it do if the thread being joined (consumer_thread, in this case) finishes before the main thread reaches the that line of code (line 77)?

Because the value is placed in a thread structure created elsewhere, that structure will still contain the return value of that thread. Therefore, when the code reaches the end, if consumer has already finished, that value will be sitting within the structure waiting.

In this program, the main thread calls pthread_join() on the threads it created. Could a different thread call pthread_join() on those threads instead? Could a thread call pthread_join() on the main thread (assuming it knew the main thread's thread ID - i.e. pthread_t)?

Yes, it could. The function waits on the finishing of a thread, which can be identified by a thread created in the context of that function, or a pointer to a thread from another process. Either way, when that thread is determined to be finished, the pthread_exit routine will place a return value in the pthread_join call. This is possible with any thread, including main.

The consumer_routine function calls sched_yield() (line 180) when there are no items in the queue. Why does it call sched_yield() instead of just continuing to check the queue for an item until one arrives?

Sched_yield() allows other threads to take CPU time. If we just kept attempting to grab the lock, then one thread could feasibly retain all the CPU uptime and the other threads would never process.