CS3840 POSIX Synchronization and multithreading homework assignment

In this assignment, you are to use POSIX or C++11 threads to create a system which implements the producer – consumer problem. In essence, you are to have n producer threads. Each producer thread will do the following:

1. Using random numbers, randomly populate a structure with a random number, a, a second random number b, and an operation (+, -, /, *) (Hint: Look at rand in math.h.)
2. Enqueue the operation into a single queue. (Note: If the queue is full, the thread should block until there is space in the queue.)
3. Increment a variable that keeps track of how many operations have been created by all producers.
4. Increment a thread specific variable that keeps track of how many operations have been created by this thread.
5. Sleep for a random period of time.

You are also to create a set of n consumer threads which do the following:

1. If there is an operation to do in the queue, dequeue the operation from the queue. Otherwise, yield.
2. Perform the operation, printing the result out to the console, as well as the PID and TID performing the operation.
3. Increment a variable that keeps of the number of operations consumed by this thread.
4. Increments a global variable which keeps track of how many operations have been consumed by all threads.
5. Repeats

The program is started by the user entering the command from the command line with the following syntax:

threadsimulator 100 100000 5 8

This command will indicate that you want to have 5 producer threads each creating 100000 operations. You will create 8 consumer threads. The queue itself can hold a maximum of 100 operations in it. The program will then start running.

At any time, the user can send a signal SIGUSR1 signal and it will print out how many operations have been produced and how many are in the queue. If the user sends a SIGUSR2 signal, the machine will print out how many operations have been consumed and how many are still pending in the queue.

When each producer has produced the correct number of operations, the producer thread will exit. The main thread, which will block on the producer threads to complete, will then inform the consumer threads to exit when the queue is empty and then will block on the consumer threads to exit. Once all consumer threads have exited, the main thread will print out how many operations have been produced and how many have been consumed to the console before exiting.

If using C++11, your queue can use the deque STL implementation. You will need to wrap around this a maximum size for the queue, and you will need to implement some form of synchronization to prevent race conditions, as the native STL implementation is not thread safe.

If you are using POSIX C, then your queue data structure is best implemented as a simple circular array. You will need appropriate synchronization around shared variables and the queue itself.

When finished, your system should ensure that threads can execute concurrently to the best of their ability and that no race conditions exist within the code. You should also verify that when you run your program, the process status information

Sample Execution:

cs3844@SECEDebian:/media/sf_lwshare/20162017CS3844Repo/homework/Syncw/Debug$ $ ./Syncw 100000 5 8 > log2.txt

Operation 1 processed by PID 1598 TID 1602: 89383 - 30886 = 58497 queue size: 51

Operation 2 processed by PID 1598 TID 1603: 47793 * 38335 = 1832144655 queue size:76

Operation 3 processed by PID 1598 TID 1602: 16649 * 41421 = 689618229 queue size: 75

.

.

.

Operation 51891 processed by PID 1598 TID 1608: 70681 * 93979 = -1947404893 queue size: 8

Operation 51892 processed by PID 1598 TID 1602: 40143 + 92878 = 133021 queue size: 7

Status: Produced 51898 operations and 6 operations are in the queue.

.

.

.

Operation 61051 processed by PID 1598 TID 1608: 65494 * 63115 = -161313486 queue size: 25

Operation 61052 processed by PID 1598 TID 1603: 4485 * 67487 = 302679195 queue size: 42

Status: Consumed 61052 operations and 44 operations are in the queue.

Operation 61053 processed by PID 1598 TID 1608: 62438 / 49678 = 1 queue size: 51

.

.

.

Operation 499994 processed by PID 1598 TID 1603: 88092 + 12944 = 101036 queue size: 0

Operation 499995 processed by PID 1598 TID 1608: 32005 + 72059 = 104064 queue size: 3

Operation 499996 processed by PID 1598 TID 1603: 6842 + 99521 = 106363 queue size: 2

Operation 499997 processed by PID 1598 TID 1602: 46161 - 27922 = 18239 queue size: 1

Operation 499998 processed by PID 1598 TID 1608: 39688 / 35484 = 1 queue size: 0

Operation 499999 processed by PID 1598 TID 1603: 16647 - 54586 = -37939 queue size: 0

Operation 500000 processed by PID 1598 TID 1602: 73252 - 92299 = -19047 queue size: 0