CSC 415 Operating Systems Principles
Fall 2017, Anthony J Souza

Programming Assignment #5: Basic Synchronization; Producer‐Consumer

Part 1: Add synchronization to your solution **to HW #4** to eliminate all the race conditions. You should not remove any of the nanosleep commands that you used to produce the race conditions, but simply add the necessary mutex synchronization calls for Linux threads. Your implementation should use the synchronization primitives of the programming API you are working with (Posix). You only need to re‐implement your solutions with the most number of race conditions (i.e. incorrect sums for threads).

Part 2. This assignment is (very roughly) based on Programming Project 6.40 in Silbershatz. You will be implementing a Producer‐Consumer program with a bounded buffer queue of N elements, P producer threads and C consumer threads (N, P and C should be command line arguments to your program, along with three additional parameters, X, Ptime and Ctime, that are described below). Each Producer thread should Enqueue X different numbers onto the queue (***spin‐waiting*** for Ptime cycles in between each call to Enqueue). Each Consumer thread should Dequeue P*X/C items from the queue (***spin‐waiting*** for Ctime cycles in between each call to Dequeue). The main program should create/initialize the Bounded Buffer Queue, print a timestamp, spawn off C consumer threads & P producer threads, wait for all of the threads to finish and then print off another timestamp & the duration of execution.

Step 1: Write high level pseudocode for the Producer and Consumer threads, as well as for the Bounded Buffer Queue (Init, Enqueue, Dequeue). Use semaphores to describe synchronization logic in your pseudocode, and put all of the calls to P/V in the Enqueue/Dequeue implementations. ***Submit this pseudocode in a file called pandcpseudo.txt .*** Design a testing strategy for verifying that all of the threads are collectively executing correctly. One possible testing strategy is to have a single atomic counter (i.e. a counter with mutex synchronization so it is guaranteed to produce different numbers) to generate numbers for Producer threads, then have the main routine combine the output from all of the Consumer threads, sort it and verify that all of the input numbers appeared as output. ***Submit this testing strategy as part of your design documentation.***

Step 2. Implement your Producer-Consumer program ***using Linux threads.***

NEXT, modify your code by adding appropriate synchronization (mutex locks and/or signal semaphores and/or count semaphores) so that your code always executes correctly. Submit well-commented source code and annotated output to demonstrate that your code is executing correctly. ***<u>Run your final program with 7 buffers (each of which can store one integer), 5 producers, 3 consumers and values of X, Ptime and Ctime that are large enough so your program executes for a total of about 45 seconds on your VirtualBox.</u>***

<u>Extra Credit Option:</u> Implement your Producers/Consumers program using Linux Processes plus System V IPC Shared Memory .

What to submit:

1. Updated source code for pthread_race.c for part 1.
2. pandcpseudo.txt file containing pseudo code from step 1 of Part 2.
3. Copy-n-paste program output from completed Part 2
4. Please fill in readme.txt file given for both Parts 1 and 2.
5. Push all completed code to your given repository by the deadline.

How to submit:

- To submit your work please do the following:
    - git add .
    - git commit -m "message"
    - git push