CSCI 340 Operating Systems Project

Due Tuesday December 10th by 11:59 PM EST

This project must be completed individually.

**Overview:** Implement and test a correct solution for the bounded-buffer, producer-consumer problem that satisfies mutual exclusion and avoids deadlocks. Demonstrate that the solution is correct. Modify the working solution so that mutual exclusive access to the buffer is violated and document the result. Modify the working solution to swap the order of the semaphore calls both before and after the critical section access in both the producers and consumers. Run tests to identify and document the problem that arises.

**Part 1.** Implement, test, and document a correct solution to the bounded-buffer, producer-consumer problem that satisfies mutual exclusion and avoids deadlocks. Think carefully about how producers can insert values into the buffer that make it easy to see whether or not you have a correct result – no lost items, no overwrites, etc. For example, you may make each producer generate 100 numbers, where producer A outputs only the number 1, producer B outputs only the number 2, etc. Use mutexes to help maintain an accurate total of all produced items and all consumed items. When the simulation ends, both totals should match. There may be other mechanisms you might use, but the requirement is to design a scheme that allows you to at a moments glance determine whether or not a run was correct. Save a screenshot of the output to demonstrate a successful run. Repeat the test multiple times to confirm that your correct result is deterministic (will happen every time).

**Part 2.** Modify your correct producer and consumer thread code from part 1 to swap the order of the semaphore commands both before and after the critical section (insert / remove) buffer access. For example, this code snippet swaps the order of the before and after critical section statements for the producer. Also, make the corresponding swap for the consumer.

```
mutex.wait();
empty.wait();
put(i); // insert item into buffer
full.post();
mutex.post();
```

Run the program multiple times. Is the result always the same? Does something unexpected happen? If so, what happened? Document your result with screenshot and a brief text write up.

**Part 3.** Modify your correct producer and consumer thread code from part 1 so that you deliberately forget to enforce mutual exclusion when producers and consumers access the buffer. Do multiple runs and save screenshots of the printed output that document the fact that the results are not deterministic due to a race condition.

**Deliverables:** C source code file and text file that explains how to compile and link the program. You will need to read about how to link the pthread library when you run the compiler. Include three separate source codes for parts 1, 2, and 3. Clearly name the source files so I can tell which code goes to which

part. Include clearly labeled screen grab images that document the results of tests run for each part. Include a written response to what was observed for parts 2 and 3. Create a single ZIP file that includes all requested deliverables.

Grading Rubric: under construction