SYSC 4001 Operating Systems, Fall 2013
Assignment 2
Due: Thursday Nov. 7th at 2pm

Purpose: There are two objectives of assignment 2. The first one is to implement three variations of the finite circular buffer for the producer/consumer problem by using Unix/Linux shared memory and semaphores. The second objective is to compare the performance for those three variations.

Assignment description:
You need to understand the producer/consumer (PC) problem discussed in class and the programming details, including semaphores and shared memory, for Linux. The following sub-sections describe the tasks for the assignment.

A. Implementation of three variations of the PC problem
Three alternatives of the PC problem were discussed in class. They differ only slightly.

> **Alternative 1**. This variation is the standard solution that was discussed in class and in most books, which uses three semaphores, S, N, and E. Semaphore S is used for mutual exclusion. Before the producer or the consumer can access the shared data or enter the critical section, it has to execute *wait(S)* operation. Based on this approach, the entire buffer will be locked, even if only one element of the array is needed every time.
>
> **Alternative 2**. The second variation is to verify if semaphore S can be removed if only one producer and one consumer are involved.
>
> **Alternative 3**. After verification of alternative 2, this variation is to test the program using multiple producers and/or multiple consumers. Create multiple instances of the producer and consumer processes and test if the results will be consistent. Modify the producer and consumer programs separately for mutual exclusion using two different semaphores, one for producers and the other one for consumers.

Part 2 + each group of consumer/ producer has it's own lock?

Use integer numbers and a for loop to verify the program. The producer generates one number and puts it into the buffer at a time. The consumer reads from the buffer and prints it to another output array for verification. After the producer finishes or the loop terminates, it puts a special number, -1, into the buffer. When the consumer reads the special value, it stops reading from the buffer. Finally, to verify if the consumer reads the numbers properly, the consumer prints the output.

Producer?

To make the output readable, each producer generates unique numbers. For instance, if there are two producers; one generates odd numbers; the other, even numbers. Alternatively, each producer generates numbers in a specific range, e.g., one producer generates numbers from 1 to 99,999; another one generates numbers from 100,000 to 199,999. The main point is to clearly distinguish the numbers generated from different producers. You can design your own method to demonstrate the output clearly.

The first few digits is the number of the producer (or pid), eg. producer 2 is could give: 201234, producer 1 could give: 101234. This is like 20000 + (rand() <= 9999)

There is another requirement for the output. The item put into or taken from the buffer each time in the shared memory must be clearly labeled or indexed. For example, when a producer puts an item into the shared memory, it should print something like:

SM 63963145: pid 1882 puts number 88 at position 7

This example shows that process 1882 generates an integer value of 88 and puts it into position 7 of the buffer. SM 63963145 indicates the shared memory ID obtained from the system when the shared memory segment is created. In other words, you need to keep track of the numbers for each producer. For this assignment, handling of overflow of the number is not required. We assume that overflow will not happen.

Likewise, when a consumer gets an item, a message like the following should be printed out on the screen.

SM 63963145: pid 1883 takes number 278 from position 5

This example says that a consumer retrieves a number 278 at position 5 in the buffer from shared memory segment 63963145.

Implementation issues:

1. The critical section for this assignment is very short and the computers these days are very fast. You may add some dummy statements, e.g., a for loop that simply counts to a big number in the critical section or sleep() *outside* of the critical section.
2. Make sure the producers and consumers are running concurrently for your testing. If a producer terminates early, the semaphores it has allocated may be released. To avoid this issue, keep the producer alive using sleep() or replace SEM_UNDO with 0 in semaphore operations.

B. Performance comparison of three variations

Performance comparison is critical to computing systems. The second part of the assignment is to conduct preliminary performance comparison for alternative 1 and alternative 3. In general, it is important to identify the parameters that potentially have impact on the system performance. Examples for this assignment include: buffer size, the time required for critical section, the number of producers/consumers, etc. This assignment only focuses on the first parameter, e.g., buffer size. Use at least three different values for the buffer size, representing small, medium, and large. Other parameter values can be fixed, but multiple producers or multiple consumers need to be used, e.g., 1pmc, mp1c, or mpmc scenarios, where 1: single, p: producer, c: consumer, and m: multiple.

Run the programs using a large fixed number of items to write/read and get the timestamps. Calculate the time that is needed to finish writing/reading all the items. Times should be measured only for actual computations, i.e., the initialization, artificial delay using *sleep()*, etc., should *not* be included. Timestamps can be obtained using *time()* system call and the *difftime()* function can be used to calculate the difference between two time values. See page 146 of the Linux book for *time()* and *difftime()* functions.

Grading criteria:
Marking is based on the following criteria:

- Correctness (including error checking, message labeling): 80%
- Documentation: 10%
- Program structure: 5%
- Style and readability: 5%

Additional notes:
Finally, as we know that customer requirements are constantly changing in practice, I also reserve the right to make changes (only minor ones) to the assignment, but the requirements will be finalized as least one week before the due date.