ASSIGNMENT #4

*Producer-Consumer using POSIX API*

CSCI 480 90 points

Check Blackboard for due date. Due by 11:59 PM.

**Specifications**

In this assignment, you will solve the Producer-Consumer problem using the PThreads library. You will get to practice using mutex and semaphores in the PThreads library.

**Preparation**

See the "PreparationForPThreads.doc" about PThreads programming.

**Project Description**

Your program will create 4 consumer threads and 4 producer threads. Each thread loops for 5 iterations. Producers insert cookies to a buffer while consumers remove cookies from the buffer.

For this project, standard counting semaphores will be used for *empty* and *full*, and a mutex lock, rather than a binary semaphore, will be used to represent *mutex*. The producer and consumer – running as separate threads -- will add cookies to and remove cookies from a buffer that is synchronized with these *empty*, *full*, and *mutex* structure.

The buffer will be manipulated with two functions, insert_item() and remove_item(), which are called by the producer and consumer threads, respectively. In this assignment, you will simply use an *int cookieCount*, which represents the total number of cookies. The insert_item(..) and remove_item(..) will just increment and decrement the cookieCount. The arguments of insert_item(..) and remove_iterm(..) will be the id of the thread. The return value indicates if there is any error (-1 means error, 0 means no error). The functions insert_item(int) and remove_item(int) will print out the thread id as well as the current cookie count. *The printing needs to be done in the critical section*.

A skeleton outlining these functions (without synchronization logic) is shown here:

```
int insert_item(int threadid)
{
    /* increment cookie count by 1.
    Print out a line indicating which thread is inserting a cookie and current cookie count.
    Return a value (0 or -1).
   */
}

int remove_item(int threadid)
{
  /* decrement cookie count by 1.
    Print out a line indicating which thread is removing a cookie and current cookie count.
    Return a value (0 or -1).
   */
}
```

The main() function will do the initialization to initialize the *mutex* lock along with *empty* and *full* semaphores. It also creates the separate producer and consumer threads, 4 each. Once it has created the producer and consumer threads, the main() function will wait for the 8 threads to finish (using pthread_join()),

A skeleton for main() is:

```
int main(int argc, char *argv[])
{
    /*  Initialization.
        Create producer threads.
        Create consumer threads.
        Wait for producer threads to finish.
        Wait for consumer threads to finish.
        Cleanup and exit.
    */
}
```

The producer thread will alternate between inserting an item and sleeping for 1 second. It will do so for 5 times. Call sleep(1) to sleep for 1 second between iterations. A skeleton for producer is:

```
void *producer(void *param)
{
   //some local variables
   //loop for 5 times:
   {
        //insert an item
        //sleep for 1 second
   }
}
```

The consumer thread can be coded similarly.

The above skeletons do *NOT* yet have any synchronization logic. You need to add mutex locks and semaphores at the correct places.

**Pthreads mutex locks**

You will use one mutex for protecting the critical section in insert_item(..) and remove_iterm(..). You need to check the return value in your program to know if the operation was successful.

**Pthreads semaphores**

You will need two semaphores for thread synchronization between producers and consumers:

```
sem_t full;
sem_t empty;
```

Semaphore *empty* is initialized to BUFFER_SIZE, which is set to 10. Semaphore *full* is initialized to 0. You can do so using the following (assuming unnamed semaphore):

```
sem_init(&empty, 0, BUFFER_SIZE);
sem_init(&full, 0, 0);
```

The code structure of using full and empty semaphores to solve the producer-consumer problem is provided in slides and textbook. Make sure you call the related wait and signal/post functions in correct order.

**Cleanup etc.**

You need to clean up the resources in main() by destroying the semaphore and mutex which can be done by calling `pthread_mutex_destroy()` and `sem_destroy()`.

You need to define the NUM_THREADS and BUFFER_SIZE as constants. You will set NUM_THREADS to 4 in this assignment. By setting them as constants, you can change them easily. If you will use printf to print out information, use fflush(stdout) after it to flush the output buffer.

**Example Output**

*producer #0 inserted a cookie. Total: 1*
*producer #2 inserted a cookie. Total: 2*
*consumer #1 removed a cookie. Total:1*
*producer #1 inserted a cookie. Total: 2*
*producer #3 inserted a cookie. Total: 3*
*consumer #2 removed a cookie. Total:2*
*consumer #0 removed a cookie. Total:1*
*consumer #3 removed a cookie. Total:0*
*producer #0 inserted a cookie. Total: 1*
*producer #2 inserted a cookie. Total: 2*
*consumer #1 removed a cookie. Total:1*
*producer #3 inserted a cookie. Total: 2*
*consumer #0 removed a cookie. Total:1*
*consumer #2 removed a cookie. Total:0*
*producer #1 inserted a cookie. Total: 1*
*consumer #3 removed a cookie. Total:0*
*producer #0 inserted a cookie. Total: 1*
*producer #3 inserted a cookie. Total: 2*
*consumer #1 removed a cookie. Total:1*
*producer #1 inserted a cookie. Total: 2*
*consumer #2 removed a cookie. Total:1*
*consumer #3 removed a cookie. Total:0*
*producer #2 inserted a cookie. Total: 1*
*consumer #0 removed a cookie. Total:0*
*producer #0 inserted a cookie. Total: 1*
*producer #1 inserted a cookie. Total: 2*
*consumer #2 removed a cookie. Total:1*
*producer #3 inserted a cookie. Total: 2*
*consumer #1 removed a cookie. Total:1*
*producer #2 inserted a cookie. Total: 2*
*consumer #3 removed a cookie. Total:1*
*consumer #0 removed a cookie. Total:0*
*producer #0 inserted a cookie. Total: 1*
*producer #1 inserted a cookie. Total: 2*
*producer #3 inserted a cookie. Total: 3*
*consumer #2 removed a cookie. Total:2*
*consumer #0 removed a cookie. Total:1*
*consumer #1 removed a cookie. Total:0*
*producer #2 inserted a cookie. Total: 1*
*consumer #3 removed a cookie. Total:0*

*All threads are done.*
*Resources cleaned up.*

Note that the order of printout is based on CPU scheduling so it can be different from example, but the cookie count needs to be consistent. In other words, you should not have a producer producing one cookie but the output shows that the cookie count suddenly increases more than one.

**Suggestions and other optional test**

1. Tackle the problem step by step. First of all make sure that your threads are all created successfully. Then let each thread do something trivial, e.g. print out its own id. Finally add the mutex and semaphores to solve the producer/consumer problem.
2. As an experiment, you may try to test printing the cookieCount outside of critical section, check if you would see inconsistency printout. You don't need to submit the result of this test.

**Requirements**

The programs should 1) work according to the specifications; 2) be comprehensible and well commented; 3) check error conditions and have proper error handling.

**Submission**
The github classroom project is at: https://classroom.github.com/a/Vo9fHS3V