Operating Systems
Producer-Consumer Problem

Lobna ElHawary
900160270

I. Code

- ***Initialize function***

This function initializes the mutual exclusion object, and the semaphores empty and full.
The mutex is initialized with its default attributes, and the empty semaphore is initialized with the value BUFFER_SIZE, since initially, the buffer is fully empty. The full semaphore is initialized with the value zero, seeing that there are no items filling the buffer initially.

- ***Insert_item function***

First, ***sem_wait(&empty)*** is executed to decrement the empty semaphore, and if the value is greater than zero, indicating that there is an empty slot in the buffer, then the decrement proceeds, otherwise if it already has the value zero, indicating that the buffer has no empty slots, the call blocks until it is no longer zero (full) and can decrement the semaphore. Next, ***pthread_mutex_lock(&mutex)*** is executed to lock the mutex, and ensure that the access to the buffer is mutually exclusive. Now we can insert the item into the buffer. Once this is done, the mutex is unlocked and ***sem_post(&full);*** is called to increment the full semaphore, seeing that we just insert an item, so there is one more full spot Finally, we return 0 to indicate failure to insert the item, and -1 incase of success.

- ***Remove_item function***

The remove item function is very similar to that of the ***insert_item*** function with the differences being that instead of ***sem_wait(&empty), sem_wait(&empty)*** is executed, and instead of inserting an item an item in the buffer, we remove it. Finally, instead of the ***sem_post(&full),*** we have ***sem_post(&empty).***

- ***Producer function***

The producer function creates a random item, and sleeps for a random period of time. This period of time was restricted to 1 - 10 seconds in the code, so that the thread does not sleep for too long, since the rand( ) function generates numbers ranging from 0 to at least 32767. Next, the randomly generated item is attempted to be inserted into the buffer using the ***insert_item*** function, and if the insertion is successful, the producer ID and the item itself is output. Otherwise, if the insertion was unsuccessful ***report error condition*** is output.

- ***Consumer function***

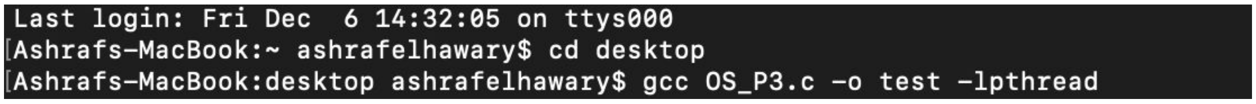
The consumer function sleeps for a random period of time that was restricted to 1 - 10 seconds in the code, so that the thread does not sleep for too long, since the rand( ) function generates numbers ranging from 0 to at least 32767. Next, an item from the buffer is attempted to be removed by calling the ***remove_item*** function, and if the removal is successful, then the consumer ID and the item it produced are output, otherwise if it was unsuccessful ***report error condition*** is output.

- ***Main function***

An error message is produced in case the input arguments input by the user are not 4 (1. The name of executable 2. How long to sleep before terminating 3. The number of producers 4. The number of consumers). Next, the inputs are converted from a string into a long integer using the ***strtol*** function, and then the absolute value of that is taken in the case that the user incorrectly input a negative number. Then, the ***initialize*** function is called. After that, two arrays containing the producer and consumer thread identifiers are created based on the user's input. Then for the number of producers and consumers, we create their threads.

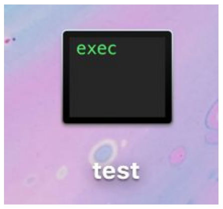
II. Running the code

First, on the terminal the command ***cd desktop*** is run, followed ***by gcc OS_P3.c -o test -lpthread.***

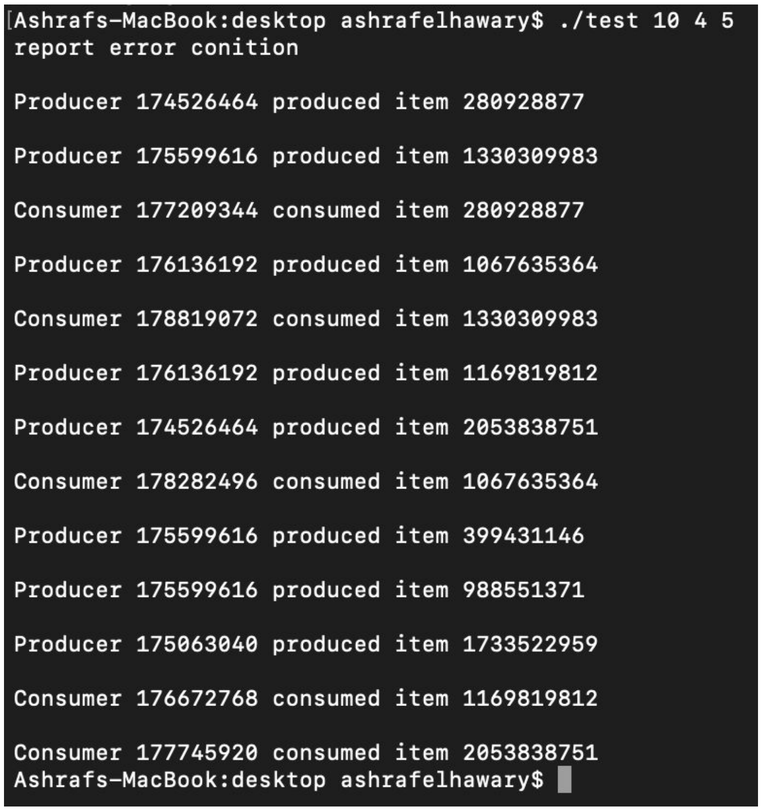


```
Last login: Fri Dec  6 14:32:05 on ttys000
[Ashrafs-MacBook:~ ashrafelhawary$ cd desktop
[Ashrafs-MacBook:desktop ashrafelhawary$ gcc OS_P3.c -o test -lpthread
```

This generates the executable file named test

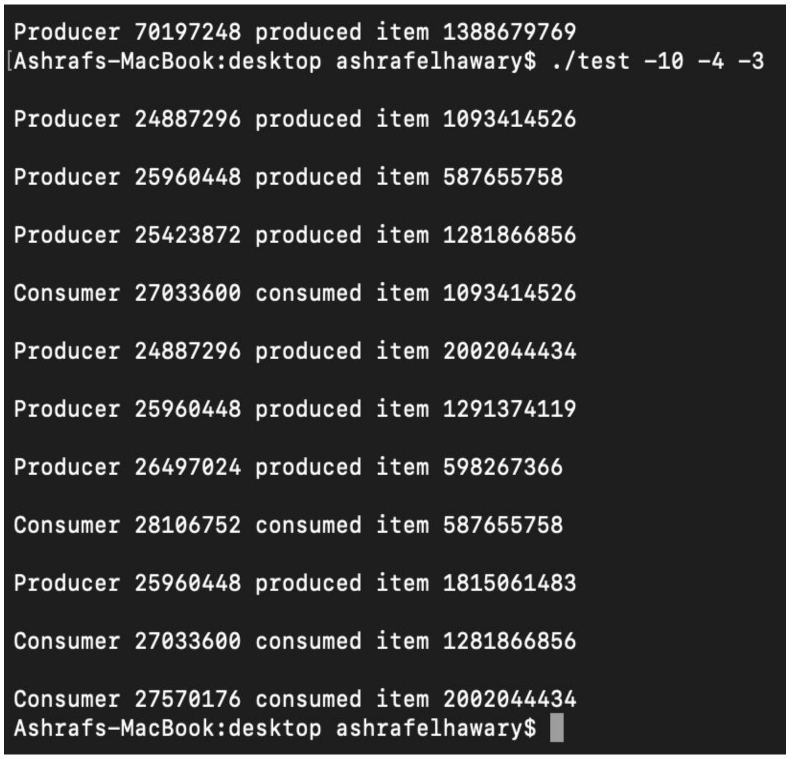


Next, we provide the input to this executable file by entering the command: ***./test sleep_time producer_num consumer_num***, and by doing so, the code runs. Here the sleep_time: 10, number of producers: 4, number of consumers is 5



```
[Ashrafs-MacBook:desktop ashrafelhawary$ ./test 10 4 5
report error conition

Producer 174526464 produced item 280928877

Producer 175599616 produced item 1330309983

Consumer 177209344 consumed item 280928877

Producer 176136192 produced item 1067635364

Consumer 178819072 consumed item 1330309983

Producer 176136192 produced item 1169819812

Producer 174526464 produced item 2053838751

Consumer 178282496 consumed item 1067635364

Producer 175599616 produced item 399431146

Producer 175599616 produced item 988551371

Producer 175063040 produced item 1733522959

Consumer 176672768 consumed item 1169819812

Consumer 177745920 consumed item 2053838751
Ashrafs-MacBook:desktop ashrafelhawary$ 
```

In case the user enters a negative number as input, the code works on the absolute value of the input, and runs normally. This is shown here:



```
Producer 70197248 produced item 1388679769
[Ashrafs-MacBook:desktop ashrafelhawary$ ./test -10 -4 -3

Producer 24887296 produced item 1093414526

Producer 25960448 produced item 587655758

Producer 25423872 produced item 1281866856

Consumer 27033600 consumed item 1093414526

Producer 24887296 produced item 2002044434

Producer 25960448 produced item 1291374119

Producer 26497024 produced item 598267366

Consumer 28106752 consumed item 587655758

Producer 25960448 produced item 1815061483

Consumer 27033600 consumed item 1281866856

Consumer 27570176 consumed item 2002044434
Ashrafs-MacBook:desktop ashrafelhawary$ 
```