Operating Systems Lab
Assignment 4

Name: Aditya Kangune **Batch**: K11
Roll number: 33323 **Date of Submission**: 09/10/2021

Problem Statement:

Thread synchronization using counting semaphores. Application to demonstrate the producer-consumer problem with counting semaphores and mutual exclusion.

Theory:

Producer-Consumer Problem :

- This problem is one of the small collection of standard, well-known problems in concurrent programming: a finite-size buffer and two classes of threads, producers and consumers, put items into the buffer (producers) and take items out of the buffer (consumers).
- A producer must wait until the buffer has space before it can put something in, and a consumer must wait until something is in the buffer before it can take something out.
- A condition variable represents a queue of threads waiting for some condition to be signaled.

Mapping of the problem to the solution

- The first case to consider is the one in which there is a single producer and a single consumer, and there is a finite-size buffer.
- The solution for the producer is to either go to sleep or discard data if the buffer is full.

- The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer empty.
- The next time the producer puts data into the buffer, it wakes up the sleeping consumer.
- The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

Threads

- Thread can be termed as an atomic entity inside a process initialized to perform a designated sub task·
- The process of thread creation, synchronization, communication, and deletion is much faster than that of a process.
- Threads can be of 2 types:
    - User Level Thread
    - Kernel Level Thread ·
- Both types of threads are of the same importance and are intended for similar purposes·
- The difference lies in the creation process and levels of abstractions.
- In the case of the producer-consumer problem, each producer and consumer can be assigned an individual thread.
- The producer thread is responsible for creating the item and placing it on the buffer.
- The consumer thread is responsible for accessing the buffer and consuming the item.

Critical Section

- The critical section is termed as that section where the thread accesses the shared resources.
- The thread that accesses the shared resources can either modify, update or wipe out the underlying data structures which may cause conflicts with other threads if not handled properly.
- Hence, a section of code (Instruction trace) is defined as the critical section of the thread wherein the thread accesses the shared resources.
- In order to avoid conflicts, the principle of mutual exclusion comes into the picture.
- Here, the section where the producer or consumer threads request access to the buffer is termed as the critical section.

Mutual Exclusion

- The critical section, if not handled correctly, can cause conflicts directly associated with the value or existence of the shared resources.
- To avoid such a conflict, the mutual exclusion principle is introduced.
- It states that only 1 thread shall be allowed to be in its critical section.
- Only those threads can requests access to the resources i.e. execute its critical section for which the remaining threads are in their remainder section (non-critical).

Counting Semaphores

- Semaphores are typically used to coordinate access to resources, with the semaphore count initialized to the number of free resources.
- Threads then atomically increment the count when resources are added and atomically decrement the count when resources are removed.

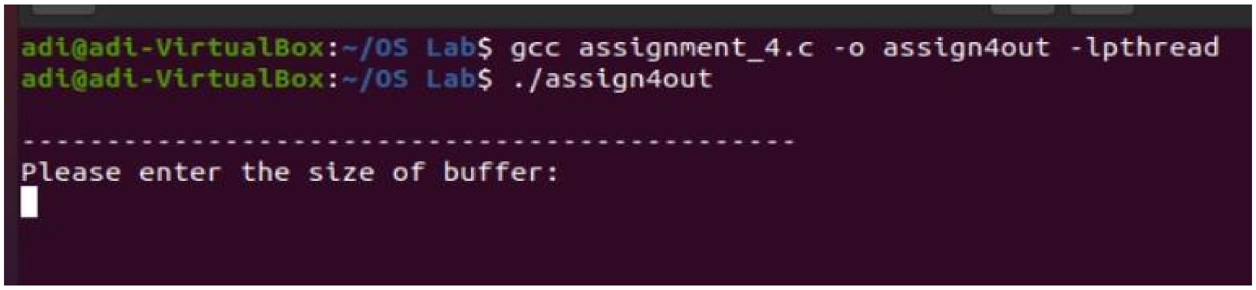
- When the semaphore count becomes zero, indicating that no more resources are present.
- Threads trying to decrement the semaphore block wait until the count becomes greater than zero.
- The life cycle of a thread involves the following operations:
    - Initialize: Initialize a semaphore (allocate memory and assign value).
    - Increment: Increment value of the semaphore.
    - Decrement: Decrement value of the semaphore.
    - Destroy: Destroy Semaphore.

  Here, the semaphores are in charge of keeping a track of the empty and full slots available in the buffer.

  Conclusion:
- The principle of Mutual Exclusion, concepts of threads, and thread synchronization were studied.
- Producer-Consumer problem was understood and solved and implemented by using the pthread library.
- Critical Section, lock and unlock were covered in order to solve the producer-consumer problem.

Main program:

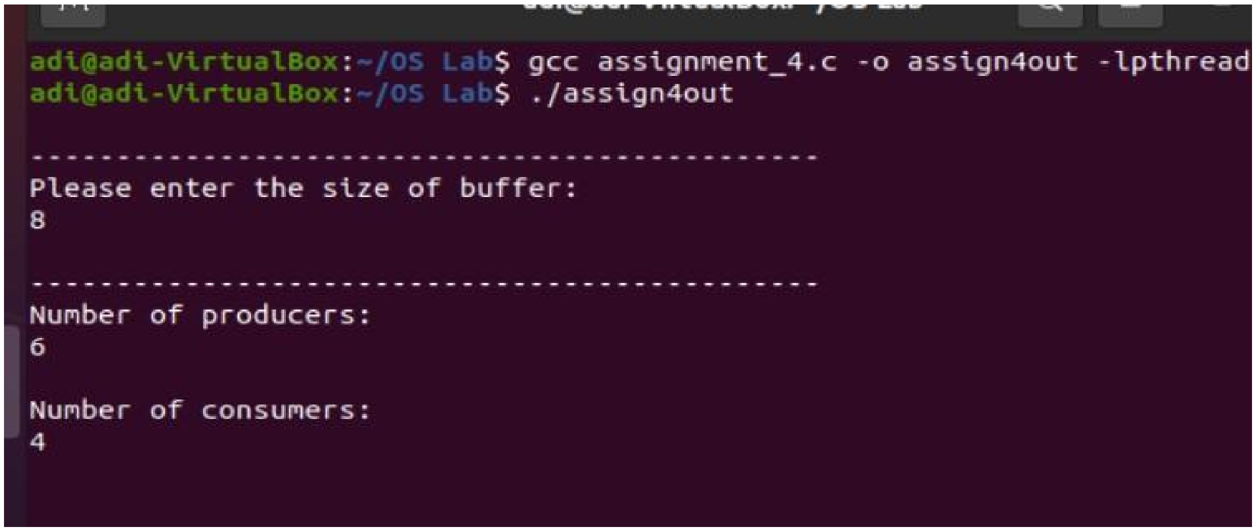
After compiling and executing:



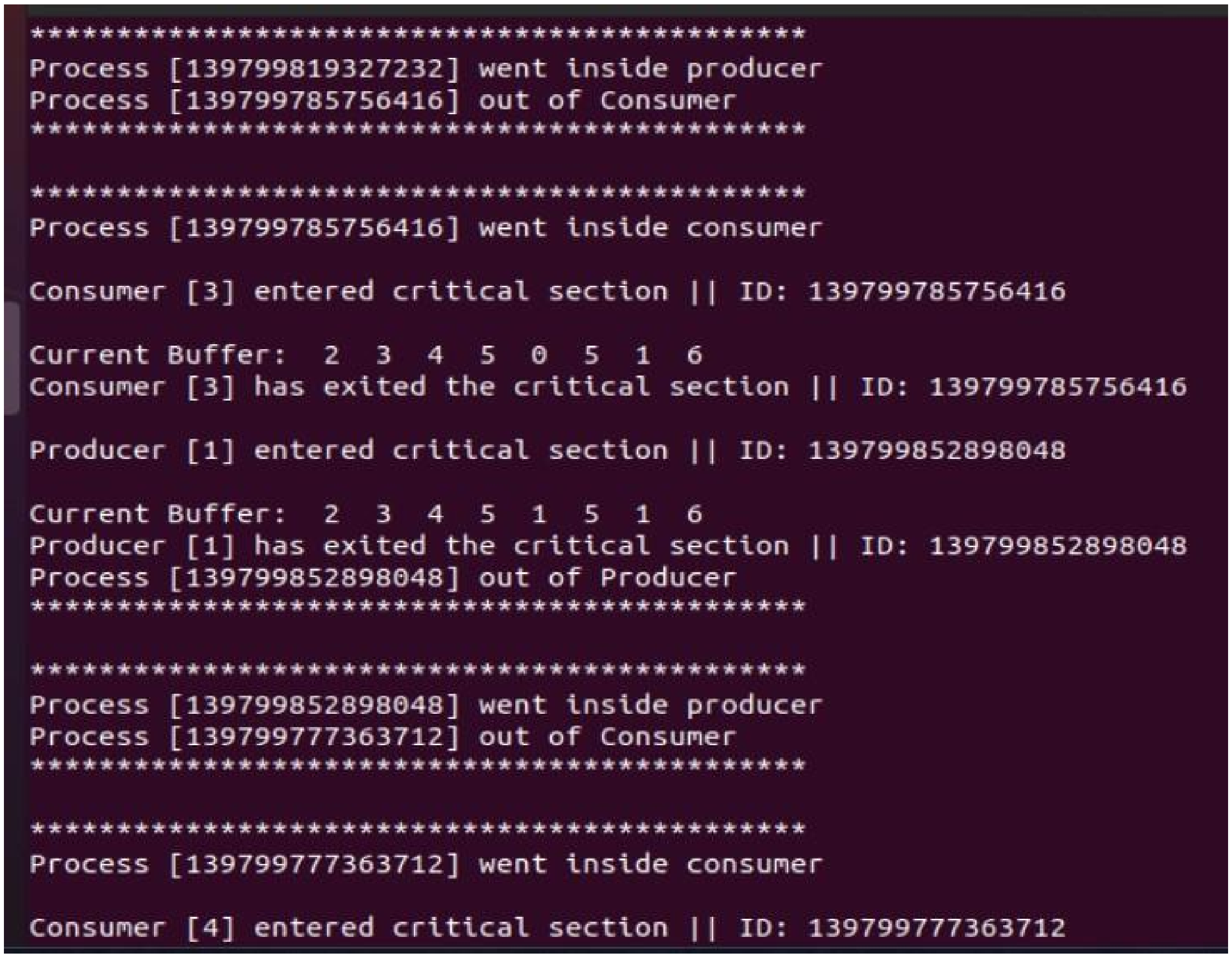
```
adi@adi-VirtualBox:~/OS Lab$ gcc assignment_4.c -o assign4out -lpthread
adi@adi-VirtualBox:~/OS Lab$ ./assign4out

-------------------------------------------
Please enter the size of buffer:
```

Buffer size = 8

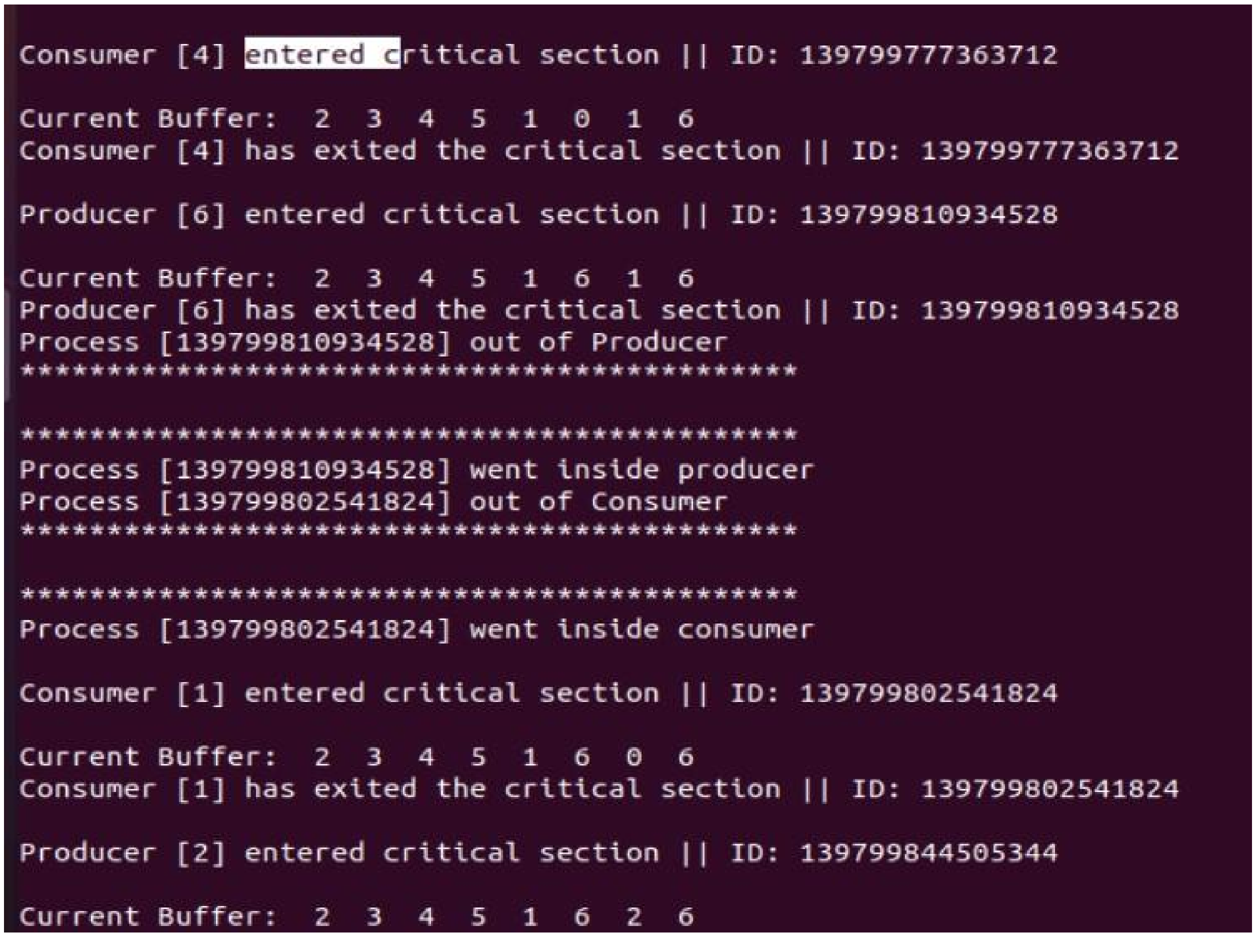
Producers = 6
Consumers = 4



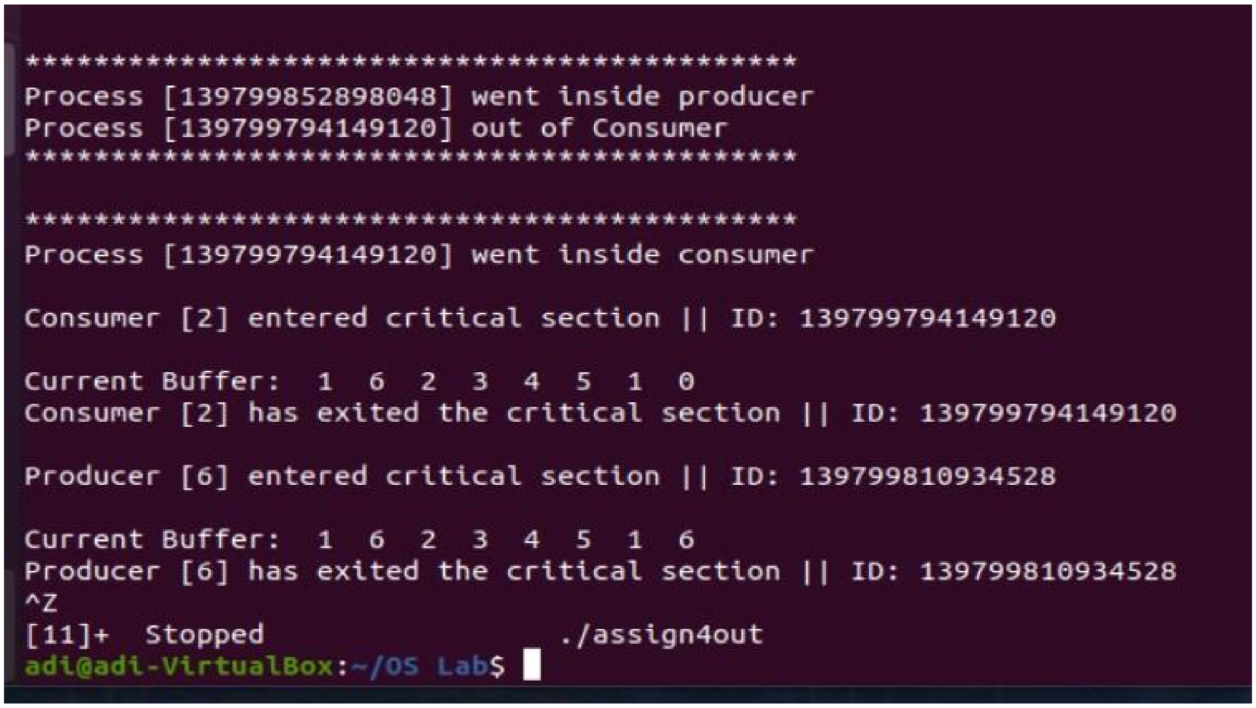
```
adi@adi-VirtualBox:~/OS Lab$ gcc assignment_4.c -o assign4out -lpthread
adi@adi-VirtualBox:~/OS Lab$ ./assign4out

---------------------------------------------
Please enter the size of buffer:
8

---------------------------------------------
Number of producers:
6

Number of consumers:
4
```



```
*********************************************
Process [139799819327232] went inside producer
Process [139799785756416] out of Consumer
*********************************************

*********************************************
Process [139799785756416] went inside consumer

Consumer [3] entered critical section || ID: 139799785756416

Current Buffer:  2  3  4  5  0  5  1  6
Consumer [3] has exited the critical section || ID: 139799785756416

Producer [1] entered critical section || ID: 139799852898048

Current Buffer:  2  3  4  5  1  5  1  6
Producer [1] has exited the critical section || ID: 139799852898048
Process [139799852898048] out of Producer
*********************************************

*********************************************
Process [139799852898048] went inside producer
Process [139799777363712] out of Consumer
*********************************************

*********************************************
Process [139799777363712] went inside consumer

Consumer [4] entered critical section || ID: 139799777363712
```



```
Consumer [4] entered critical section || ID: 139799777363712

Current Buffer:  2  3  4  5  1  0  1  6
Consumer [4] has exited the critical section || ID: 139799777363712

Producer [6] entered critical section || ID: 139799810934528

Current Buffer:  2  3  4  5  1  6  1  6
Producer [6] has exited the critical section || ID: 139799810934528
Process [139799810934528] out of Producer
*********************************************

*********************************************
Process [139799810934528] went inside producer
Process [139799802541824] out of Consumer
*********************************************

*********************************************
Process [139799802541824] went inside consumer

Consumer [1] entered critical section || ID: 139799802541824

Current Buffer:  2  3  4  5  1  6  0  6
Consumer [1] has exited the critical section || ID: 139799802541824

Producer [2] entered critical section || ID: 139799844505344

Current Buffer:  2  3  4  5  1  6  2  6
```



```
*********************************************
Process [139799852898048] went inside producer
Process [139799794149120] out of Consumer
*********************************************

*********************************************
Process [139799794149120] went inside consumer

Consumer [2] entered critical section || ID: 139799794149120

Current Buffer:  1  6  2  3  4  5  1  0
Consumer [2] has exited the critical section || ID: 139799794149120

Producer [6] entered critical section || ID: 139799810934528

Current Buffer:  1  6  2  3  4  5  1  6
Producer [6] has exited the critical section || ID: 139799810934528
^Z
[11]+  Stopped                 ./assign4out
adi@adi-VirtualBox:~/OS Lab$ 
```