

Lab Assignment 1

Roll No : CS18BTECH11001

PLAGIARISM STATEMENT

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name: A. Venkata Sai Mahesh

Date: 16/01/2020

Signature: Venkata Sai Mahesh Abburi

For the producer-consumer problem using multithreaded program I used command line arguments for taking no. of producers, no. of consumers, buffer size as strings which is then converted to integers using atoi function of stdlib.h library and given to p, c, N variables. I declared a long pointer (*buffer) in global and pointed to N long space in heap created using malloc function. Then I declared p threads for the p producers and c threads for c consumers. Then I initialized (created) these threads to their respective function using pthread_create function of pthread.h library. Then I used pthread_join function of pthread.h library for letting main function to wait until all the non-main threads get executed. In the producer function I used produced variable for each producer thread to exit when it has produced 10*N products. In the consumer function I exited when all the products 10*N*p are consumed (as per my assumption).

Without Using Semaphore :

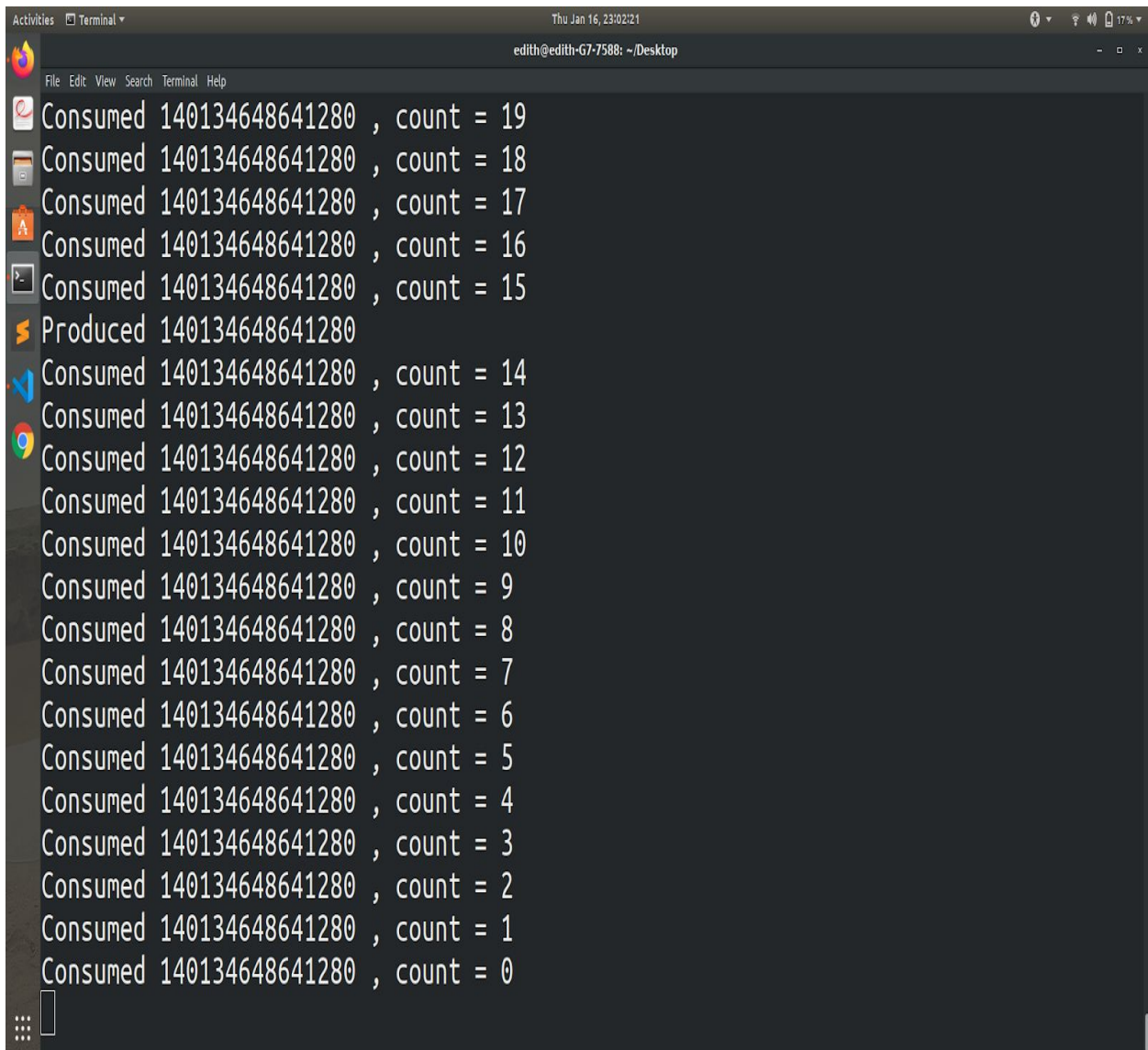
I used an empty while loop in the producer function which waits until there is an empty slot in the buffer for the producer to produce. If there is an available slot then the produced value (considered as the thread id) will be entered into the buffer increasing the produced and tproduced variables which is used to denote the total no. of products produced. I also changed the in variable to the next value in the cyclic buffer. I used count variable to denote no. of slots occupied in the buffer which increased when a product is produced and decreased when consumed.

Data Inconsistency Problems:

When a producer enters its id into the buffer and a context switch occurs then if there is any producer trying to add to the same position as that of before producer which leads to a data inconsistency as the last position was not updated before the context switch

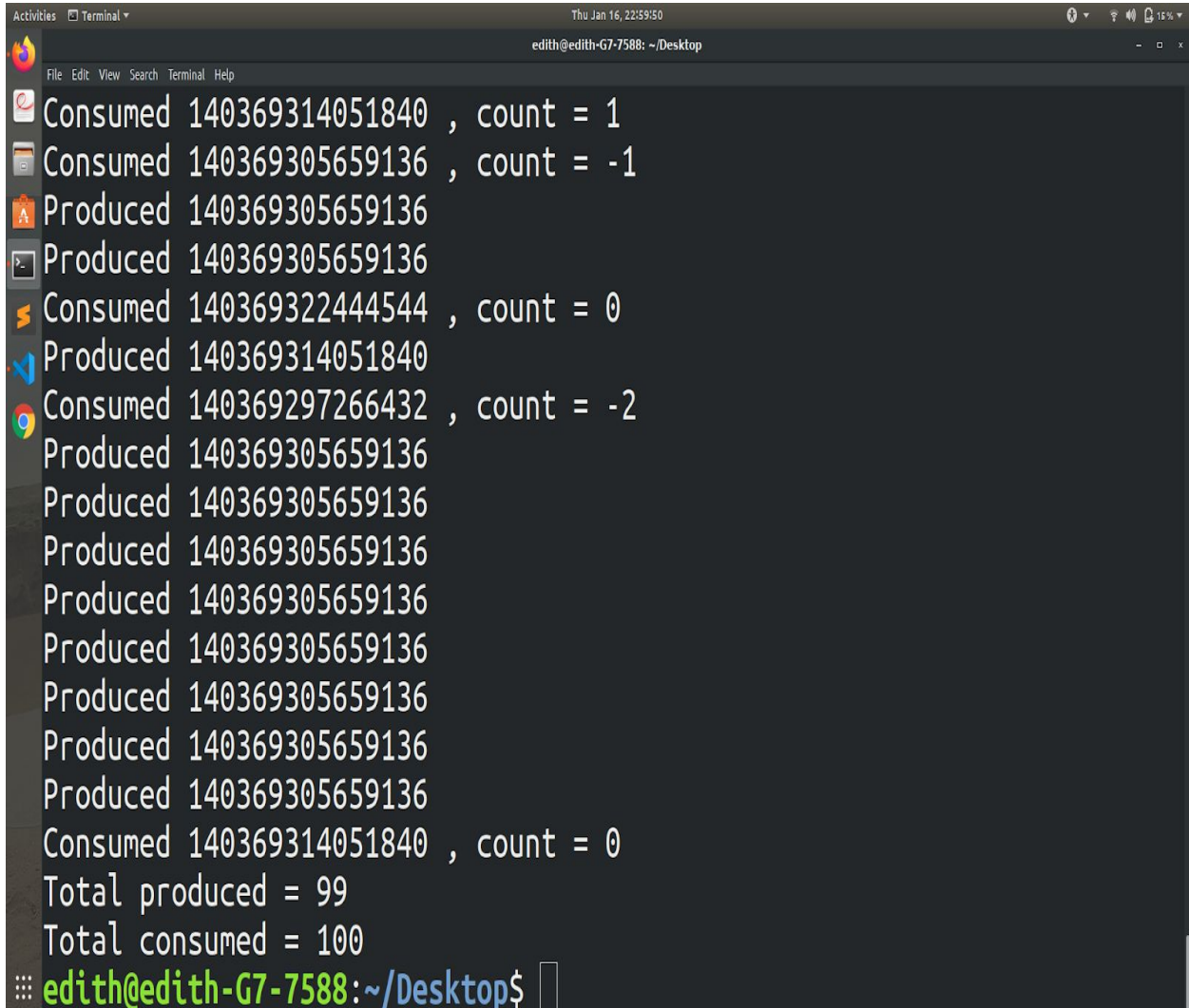
Problem 1 : The program sometimes doesn't exit.

Reason : It is caused when all the producer threads exited and if there is any consumer thread entered into critical section where the value of count doesn't change and it keeps waiting instead of exiting.



```
Thu Jan 16, 23:02:21
edith@edith-G7-7588: ~/Desktop
File Edit View Search Terminal Help
Consumed 140134648641280 , count = 19
Consumed 140134648641280 , count = 18
Consumed 140134648641280 , count = 17
Consumed 140134648641280 , count = 16
Consumed 140134648641280 , count = 15
Produced 140134648641280
Consumed 140134648641280 , count = 14
Consumed 140134648641280 , count = 13
Consumed 140134648641280 , count = 12
Consumed 140134648641280 , count = 11
Consumed 140134648641280 , count = 10
Consumed 140134648641280 , count = 9
Consumed 140134648641280 , count = 8
Consumed 140134648641280 , count = 7
Consumed 140134648641280 , count = 6
Consumed 140134648641280 , count = 5
Consumed 140134648641280 , count = 4
Consumed 140134648641280 , count = 3
Consumed 140134648641280 , count = 2
Consumed 140134648641280 , count = 1
Consumed 140134648641280 , count = 0
```

Problem 2 : The total number of products produced by all the producers isn't $10*N*p$.
Reason : It is due to the switching between the threads and second critical section is not atomic operation.



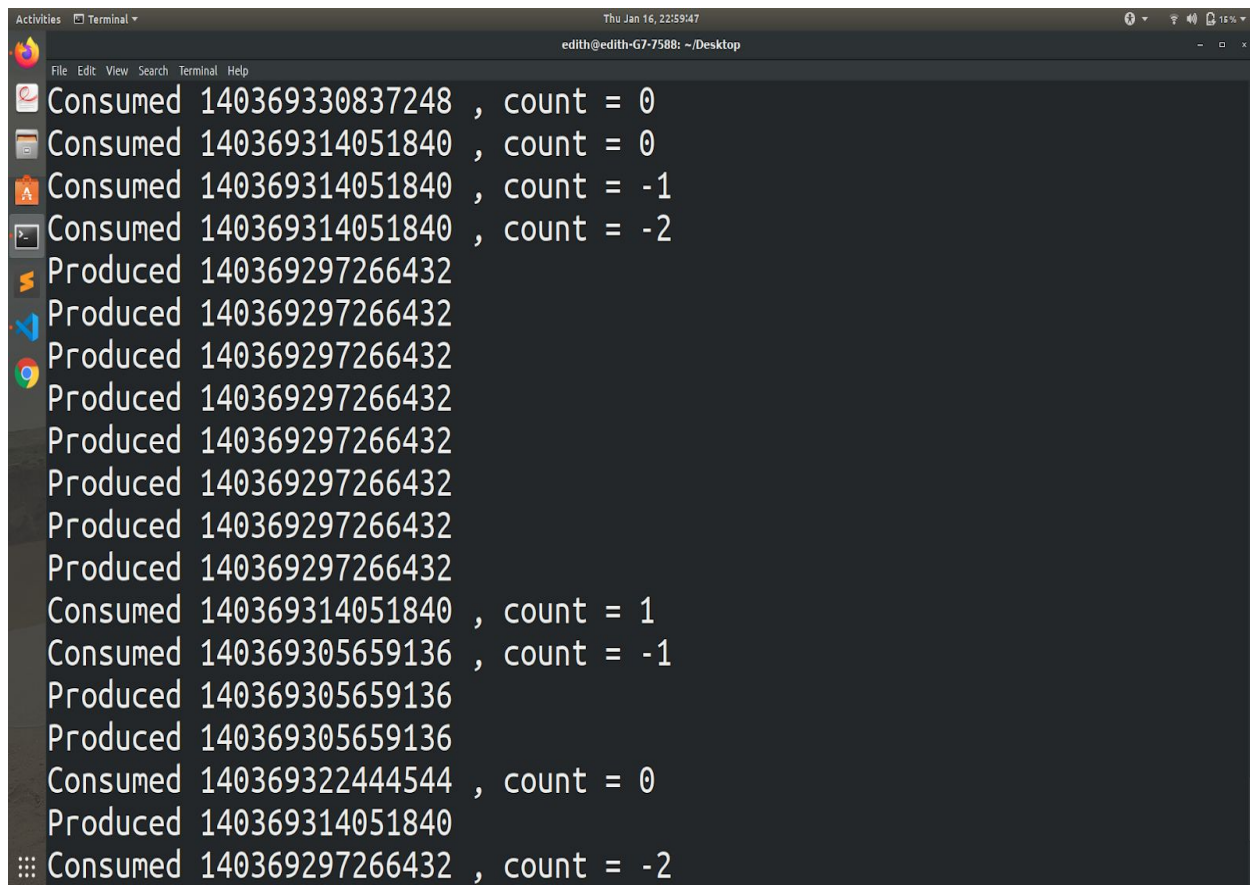
```
Activities Terminal Thu Jan 16, 22:59:50
edith@edith-G7-7588: ~/Desktop
File Edit View Search Terminal Help
Consumed 140369314051840 , count = 1
Consumed 140369305659136 , count = -1
Produced 140369305659136
Produced 140369305659136
Consumed 140369322444544 , count = 0
Produced 140369314051840
Consumed 140369297266432 , count = -2
Produced 140369305659136
Produced 140369305659136
Produced 140369305659136
Produced 140369305659136
Produced 140369305659136
Produced 140369305659136
Produced 140369305659136
Consumed 140369314051840 , count = 0
Total produced = 99
Total consumed = 100
edith@edith-G7-7588:~/Desktop$
```

Problem 3 : The count value becomes negative which indicates there is consumption without products.

Reason : This is also due to switching between the threads.

For Analysing these two conditions we consider an example:

Consider $count=0$. Now let the producer change the count to 1 and we then exit all the consumer threads that are in busy waiting and change decrease the count which leads to negative in count representing consuming without having any product.



```
Activities Terminal
Thu Jan 16, 22:59:47
edith@edith-G7-7588: ~/Desktop
File Edit View Search Terminal Help
Consumed 140369330837248 , count = 0
Consumed 140369314051840 , count = 0
Consumed 140369314051840 , count = -1
Consumed 140369314051840 , count = -2
Produced 140369297266432
Produced 140369297266432
Produced 140369297266432
Produced 140369297266432
Produced 140369297266432
Produced 140369297266432
Produced 140369297266432
Consumed 140369314051840 , count = 1
Consumed 140369305659136 , count = -1
Produced 140369305659136
Produced 140369305659136
Consumed 140369322444544 , count = 0
Produced 140369314051840
Consumed 140369297266432 , count = -2
```

Fairness :

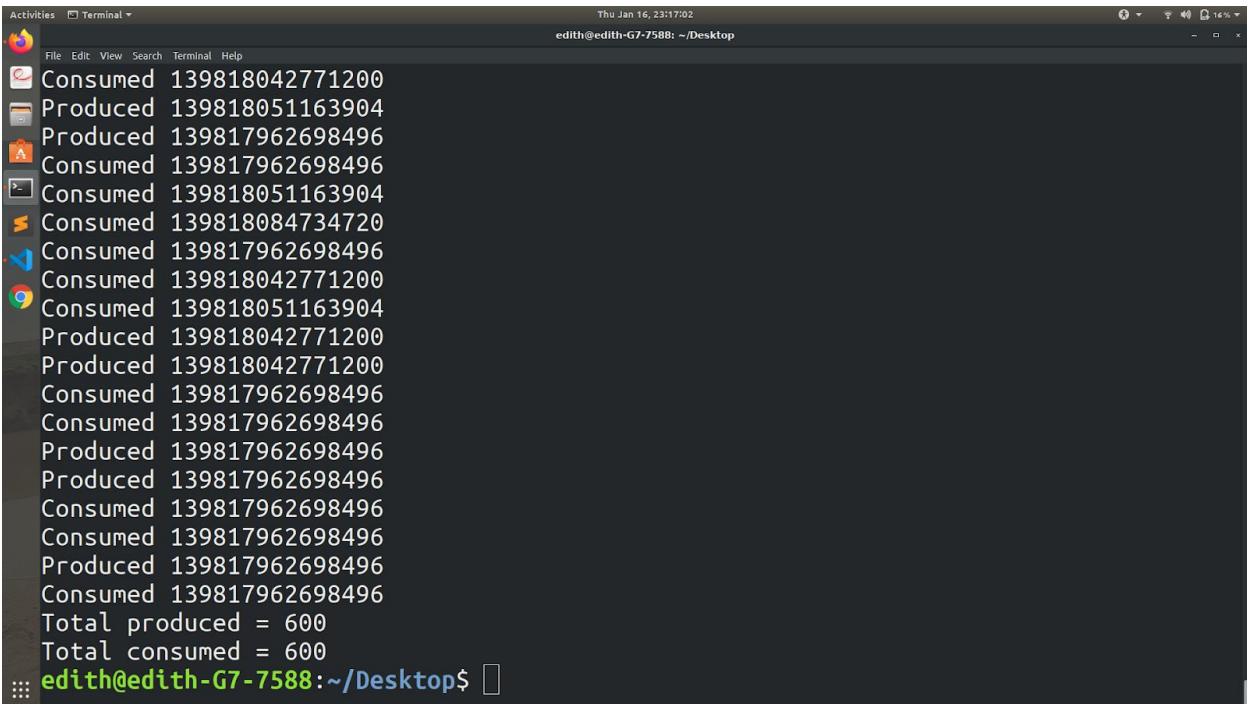
We can observe that there isn't any fairness as some of the producers dominate over other. This is due to the whole dependent on Scheduler for the context switching.

With using Semaphores :

Data Inconsistency Issues Solved :

When semaphores are used, concurrent access to shared data is handled due to the allowing of only one thread to access the shared memory thereby minimizing the chance of inconsistent data which shows the synchronisation.

Solution 1: There is synchronization between threads where only one thread can enter the critical section when we used semaphores.



```
edith@edith-G7-7588: ~/Desktop
Consumed 139818042771200
Produced 139818051163904
Produced 139817962698496
Consumed 139817962698496
Consumed 139818051163904
Consumed 139818084734720
Consumed 139817962698496
Consumed 139818042771200
Consumed 139818051163904
Produced 139818042771200
Produced 139818042771200
Consumed 139817962698496
Consumed 139817962698496
Produced 139817962698496
Produced 139817962698496
Consumed 139817962698496
Consumed 139817962698496
Produced 139817962698496
Consumed 139817962698496
Total produced = 600
Total consumed = 600
edith@edith-G7-7588:~/Desktop$
```

Solution 2 : As there is synchronization between all the threads all are given the same priority and all the producer threads produce $10 \times N$ products which makes $10 \times p \times N$ in total.

Solution 3 : As we are using atomic operations `sem_wait`, `sem_post` for doing busy waiting, there will be no chance of consumption with no products in buffer.

Fairness:

As there is synchronisation between the threads all the threads will execute in fair with the other threads in which there will be complete fairness.