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

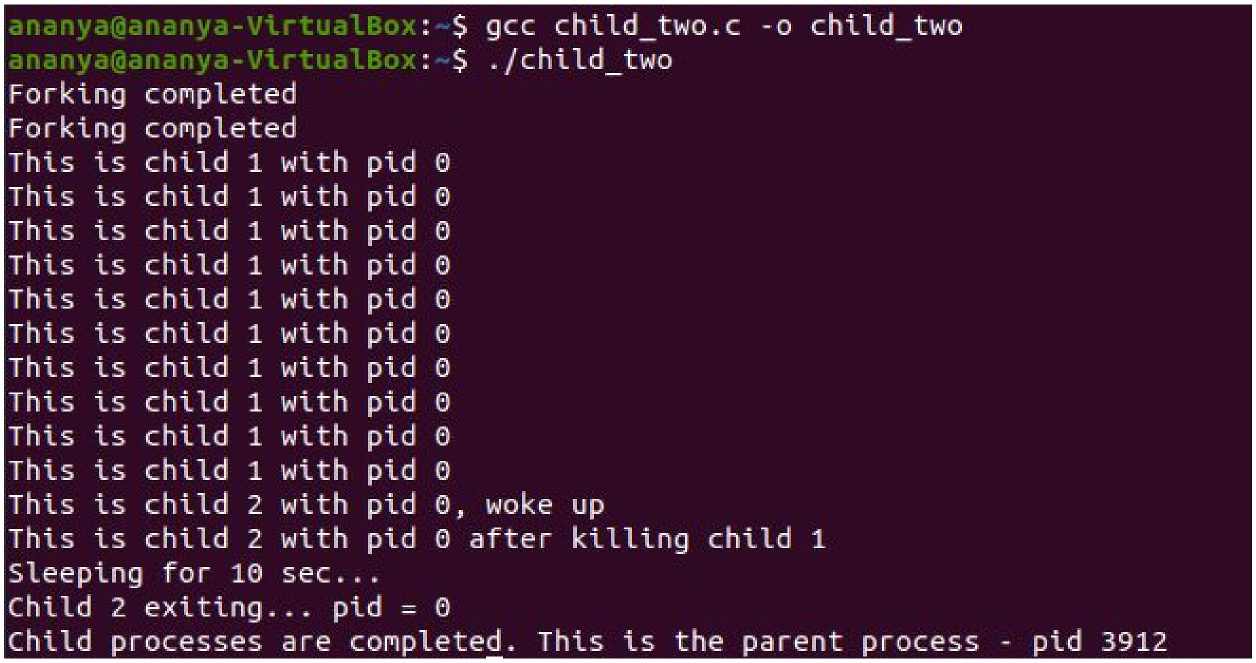
assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles 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: Ananya Mantravadi

Date: 22 Nov 2020

Signature: AM

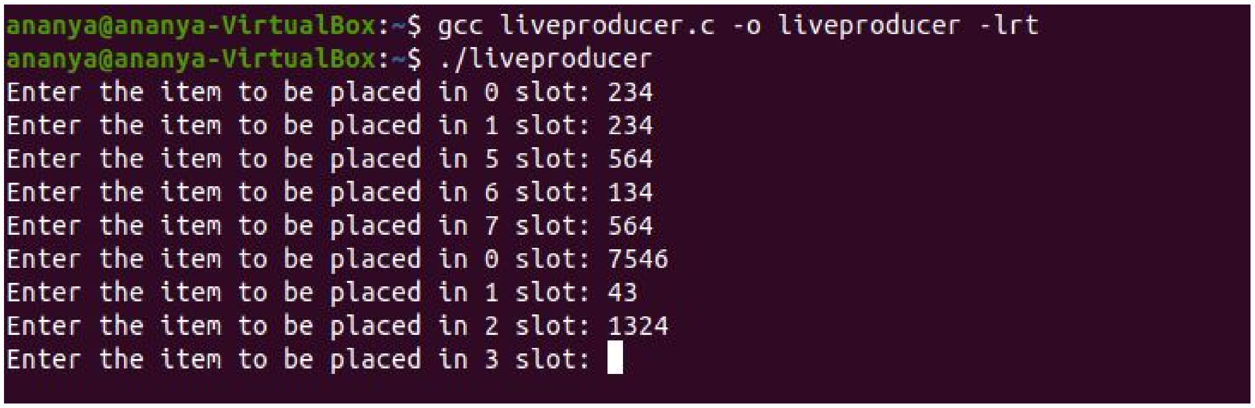
1. ***Forking two child processes***



```
ananya@ananya-VirtualBox:~$ gcc child_two.c -o child_two
ananya@ananya-VirtualBox:~$ ./child_two
Forking completed
Forking completed
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 1 with pid 0
This is child 2 with pid 0, woke up
This is child 2 with pid 0 after killing child 1
Sleeping for 10 sec...
Child 2 exiting... pid = 0
Child processes are completed. This is the parent process - pid 3912
```

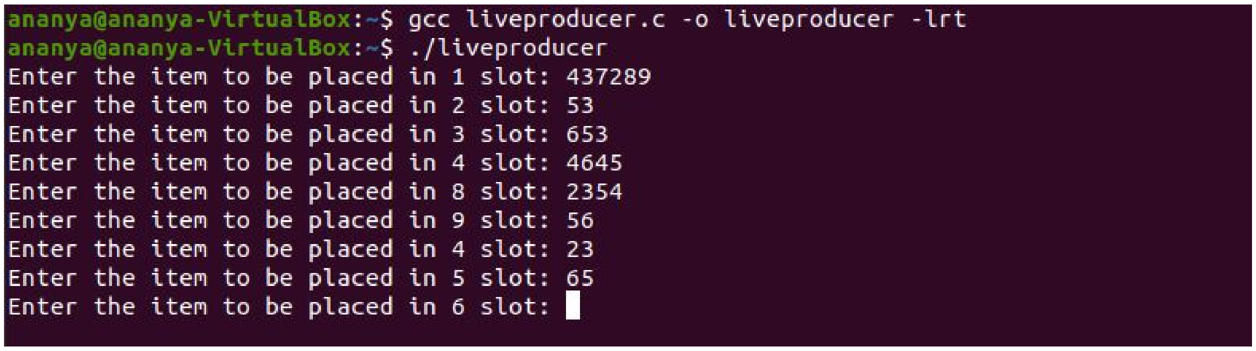
We fork the parent process twice to create two child processes. As instructed in the assignment, the first child process keeps printing a message every second while the second child process sleeps for the first ten seconds. Second child process then kills the first child process using SIGKILL. This means that the first child process gets to print only 10 statements before it gets killed. Second child process then sleeps for another 10 seconds before it gets terminated. To ensure that the child processes are executed before the parent process, we use waitpid to ensure that the child state changes. Finally, the parent process also gets terminated.

2. **Live Producer(s) and Live Consumers**



```
ananya@ananya-VirtualBox:~$ gcc liveproducer.c -o liveproducer -lrt
ananya@ananya-VirtualBox:~$ ./liveproducer
Enter the item to be placed in 0 slot: 234
Enter the item to be placed in 1 slot: 234
Enter the item to be placed in 5 slot: 564
Enter the item to be placed in 6 slot: 134
Enter the item to be placed in 7 slot: 564
Enter the item to be placed in 0 slot: 7546
Enter the item to be placed in 1 slot: 43
Enter the item to be placed in 2 slot: 1324
Enter the item to be placed in 3 slot: 
```

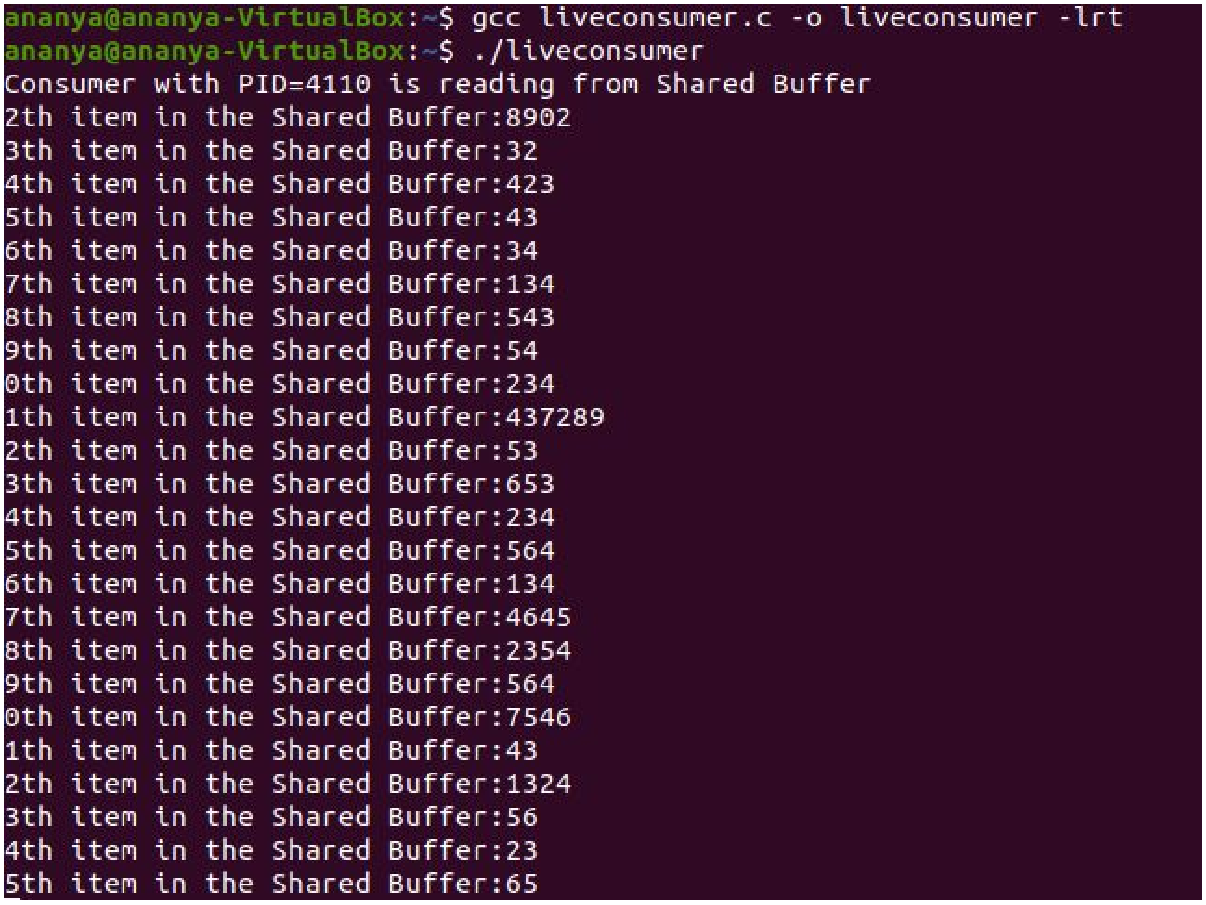
Above is a screenshot of a liveproducer from terminal 1



```
ananya@ananya-VirtualBox:~$ gcc liveproducer.c -o liveproducer -lrt
ananya@ananya-VirtualBox:~$ ./liveproducer
Enter the item to be placed in 1 slot: 437289
Enter the item to be placed in 2 slot: 53
Enter the item to be placed in 3 slot: 653
Enter the item to be placed in 4 slot: 4645
Enter the item to be placed in 8 slot: 2354
Enter the item to be placed in 9 slot: 56
Enter the item to be placed in 4 slot: 23
Enter the item to be placed in 5 slot: 65
Enter the item to be placed in 6 slot: 
```

Above is a screenshot of a liveproducer from terminal 2

These two producers result in simultaneous output from liveconsumer as can be seen below (It includes the output of the shared memory which was already in the buffer and hasn't yet been given as an output by the liveconsumer program). I have given inputs in both the liveproducer terminals randomly.



```
ananya@ananya-VirtualBox:~$ gcc liveconsumer.c -o liveconsumer -lrt
ananya@ananya-VirtualBox:~$ ./liveconsumer
Consumer with PID=4110 is reading from Shared Buffer
2th item in the Shared Buffer:8902
3th item in the Shared Buffer:32
4th item in the Shared Buffer:423
5th item in the Shared Buffer:43
6th item in the Shared Buffer:34
7th item in the Shared Buffer:134
8th item in the Shared Buffer:543
9th item in the Shared Buffer:54
0th item in the Shared Buffer:234
1th item in the Shared Buffer:437289
2th item in the Shared Buffer:53
3th item in the Shared Buffer:653
4th item in the Shared Buffer:234
5th item in the Shared Buffer:564
6th item in the Shared Buffer:134
7th item in the Shared Buffer:4645
8th item in the Shared Buffer:2354
9th item in the Shared Buffer:564
0th item in the Shared Buffer:7546
1th item in the Shared Buffer:43
2th item in the Shared Buffer:1324
3th item in the Shared Buffer:56
4th item in the Shared Buffer:23
5th item in the Shared Buffer:65
```

These programs use a shared memory which is implemented as a circular buffer so that it never overflows. Basic implementation while writing the codes for liveproducer.c and liveconsumer.c is to create another shared memory, an array of size two to store the “in" and “out” pointers across the programs. This would be given a read-write access in liveconsumer.c also so that it can update the “out” pointer. While “in” = “out” the liveconsumer pauses. Producer keeps producing as long as ((in+ 1) % BUFFER_SIZE) == out)