

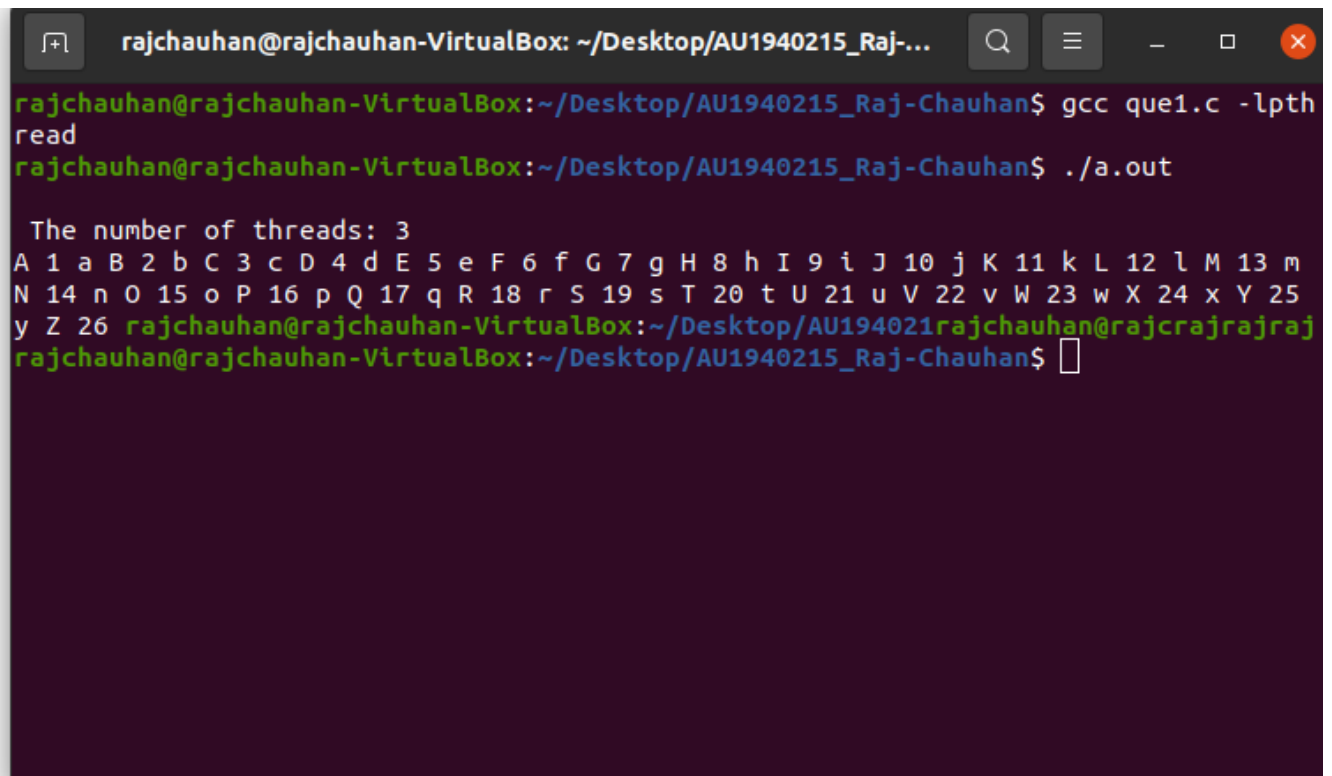
OS LAB- End Semester Description

Name- Raj Chauhan

Roll no – AU1940215

-----Question-1 -----

Firstly, we will generate 3 different threads. What we will do is that we will display Capital letter in thread 1, number in thread-2 and lower letter in thread-3. So we will run these 3 threads multiple times using thread synchronization to display the desired output in the terminal. We have created a function to synchronize threads by name **foo**. Depending upon the thread we will name the turn of the thread and then according we will display the output. For example, when the count is 0 first thread is running and we will display A, in count 2, second thread is running so we will display 1, in third thread we will display a. By this algorithm we will iterate this function for 26 times to display 26 letters.



```
rajchauhan@rajchauhan-VirtualBox: ~/Desktop/AU1940215_Raj-...  
rajchauhan@rajchauhan-VirtualBox:~/Desktop/AU1940215_Raj-Chauhan$ gcc que1.c -lpthread  
rajchauhan@rajchauhan-VirtualBox:~/Desktop/AU1940215_Raj-Chauhan$ ./a.out  
  
The number of threads: 3  
A 1 a B 2 b C 3 c D 4 d E 5 e F 6 f G 7 g H 8 h I 9 i J 10 j K 11 k L 12 l M 13 m  
N 14 n O 15 o P 16 p Q 17 q R 18 r S 19 s T 20 t U 21 u V 22 v W 23 w X 24 x Y 25  
y Z 26 rajchauhan@rajchauhan-VirtualBox:~/Desktop/AU1940215_Raj-Chauhan$
```

-----Question-2-----

In Normal Round robin we can see that it takes more time for scheduling but however in the Modified round robin we see that Average waiting time and Average Turn around time both are less respectively. So MRR works better than RR. MRR shows the lesser number of CS, lesser Average TAT and lesser Average WT than the other two algorithms. The proposed algorithms actively monitors the burst time of the executing processes and updates the TQ accordingly. Therefore, it overcomes all the drawbacks of the RR algorithm as the proposed algorithm has lesser context switches, lesser turnout and waiting time.

```
rajchauhan@rajchauhan-VirtualBox: ~/Desktop/AU1940215_Raj-Chauhan
rajchauhan@rajchauhan-VirtualBox:~/Desktop/AU1940215_Raj-Chauhan$ gcc ques2.c
rajchauhan@rajchauhan-VirtualBox:~/Desktop/AU1940215_Raj-Chauhan$ ./a.out
Total number of process in the system: 5Enter the Time Quantum for the process: 10
Process No      Burst Time      TAT      Waiting Time
Process No[5]   36              777      741
Process No[2]   18              66300    66282
Process No[3]   27              31573201 31573174
```

-----Question-3-----

In producer consumer, problem producers are generating certain types of data and putting them in a place called buffer. Buffer may be of an infinite or finite length. Consumers are the ones who are taking data out of the buffer. They are taking out data one by one. We also need to take care that when the buffer is full, producer do not add more data, and also when the consumer is empty it should not try to remove the data from the empty buffer. To solve the producer consumer problem, binary semaphores are used. Firstly we need 2 semaphores namely full and empty, which keeps track of no. of full and empty slots in the buffer.

In consumer's code we will down the full as it consumes the data item. Value of binary semaphore s or mutex is reduced so that the producer cannot activate the critical section right now, then after completing the task consumer will increase the mutex and also the empty.

In the producer's code the reverse will happen. It will down the empty and also the mutex, so consumer cannot pre-empt the process. After completing his process, the producer will increase the mutex as well as full.

