Ahmedabad University

CSE332 : Operating System Lab

Section 2

Monsoon Semester 2021

Sakshi Shah - AU1940213

End Semester Examination Report - description

date: 23th November 2021

Question 1 :

(A) Use Multiple thread

- **Output :**

```
sakshi@sakshi:~$ gcc ques1.c -o ques1
sakshi@sakshi:~$ ./ques1
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 zsakshi@sakshi:~$
```

- **Description :**

The child processes can be spawned using fork and a counter can be used in order to print the sequence where each child process can print one of the needed character in the sequence

The three threads are used here which work concurrently, each printing one of the characters concurrently with proper synchronization.

Question 2 :

(A) Round Robin

- **Output :**

Quantum time = 2

```
sakshi@sakshi:~$ gcc rr.c -o rr
sakshi@sakshi:~$ ./rr
 Total Number of process to be executed in the system with Round Robin method : 5

Process [1]
Arrival Time of Process: 0
Burst Time of the Process : 1

Process [2]
Arrival Time of Process: 1
Burst Time of the Process : 3

Process [3]
Arrival Time of Process: 1
Burst Time of the Process : 2

Process [4]
Arrival Time of Process: 1
Burst Time of the Process : 5

Process [5]
Arrival Time of Process: 1
Burst Time of the Process : 7
Enter the Time Quantum (q) for the RR scheduling for above processes:   2

Process Id :     Burst Time :    TurnAroundTime          Waiting Time
Process Id[1]        1                   1                       0
Process Id[3]        2                   4                       2
Process Id[2]        3                   9                       6
Process Id[4]        5                   14                      9
Process Id[5]        7                   17                      10
Average Turn Around Time : 5.400000
Average Waiting Time : 9.000000
sakshi@sakshi:~$
```

Quantum time =4

```
sakshi@sakshi:~$ ./rr
 Total Number of process to be executed in the system with Round Robin method : 5

Process [1]
Arrival Time of Process: 0
Burst Time of the Process : 1

Process [2]
Arrival Time of Process: 1
Burst Time of the Process : 3

Process [3]
Arrival Time of Process: 1
Burst Time of the Process : 2

Process [4]
Arrival Time of Process: 1
Burst Time of the Process : 5

Process [5]
Arrival Time of Process: 1
Burst Time of the Process : 7
Enter the Time Quantum (q) for the RR scheduling for above processes:   4

Process Id :     Burst Time :    TurnAroundTime          Waiting Time
Process Id[1]        1                   1                       0
Process Id[2]        3                   3                       0
Process Id[3]        2                   5                       3
Process Id[4]        5                   14                      9
Process Id[5]        7                   17                      10
Average Turn Around Time : 4.400000
Average Waiting Time : 8.000000
sakshi@sakshi:~$
```

- **Description :**

Round Robin is a scheduling algorithm used for effective switches among processes for given time quantum, the round robin algorithm can also be used keeping in mind priorities, burst time, arrival time of the processes to arrange them in the ready queue.
For the round robin implementation in C, first the number of processes, their arrival time, their burst time is taken as input from the user. After obtaining the details of process and quantum time the loop is used for the process to switch at each quantum time interval, the burst time of each process is stored into a side variable thus, as the process executes for certain period of time, it helps calculate remaining time of execution for each process. The waiting time can be calculated when any process is completed by wait time = wait time (waited during starting) - burst time - arrival time + time(current -- when process is completed).
Turn around time is obtained by finishing time - arrival time.

Both the values (wait time and turn around time) are averaged using each process value.

(B) Modified RR :

```
sakshi@sakshi:~$ gcc mrr.c -o mrr
sakshi@sakshi:~$ ./mrr
Enter total number of the processes to be executed under RR: 5

Enter the Arrival time , Burst time and Priority for each process :
Process Id : [1]
Arrival Time of Process: 0
Burst Time of Process : 1
Priority of the process: 1

Process Id : [2]
Arrival Time of Process: 1
Burst Time of Process : 3
Priority of the process: 2

Process Id : [3]
Arrival Time of Process: 2
Burst Time of Process : 4
Priority of the process: 3

Process Id : [4]
Arrival Time of Process: 1
Burst Time of Process : 5
Priority of the process: 4

Process Id : [5]
Arrival Time of Process: 1
Burst Time of Process : 6
Priority of the process: 5
Enter quantum time: 3

Process ID      Wait-Time       Burst-Time      Turnaround-Time

Procees ID[1]           0               1               1
Procees ID[2]           0               3               3
Procees ID[3]           2               4               6
Procees ID[4]           7               5               12
Procees ID[5]           12              6               18

Average Turn Around Time : 4.200000
Average Waiting Time : 8.000000
sakshi@sakshi:~$
```

Modified RR : Each Process runs on the basis of the priority and quantum time which help increase the effectiveness of CPUs than traditional methods.

Modified Round Robin scheduling algorithm works on the basis of priority,the processes are selected as per priority. This scheduling increases the CPU efficiency than the traditional round robin scheduling.

Buddy's Memory Allocation : (optional)
The buddy system is implemented as a list of nodes of a tree structure with different possible powers of 2, and is maintained at all times. Thus the memory allocation takes place as 1byte, 2 bytes, 4 bytes, 8 bytes, 16 bytes, 32 bytes and so on. Thus on request the allocation takes place for the next bigger block and if block is available allocation is done else it is more bigger blocks are searched in the list where it can fit.

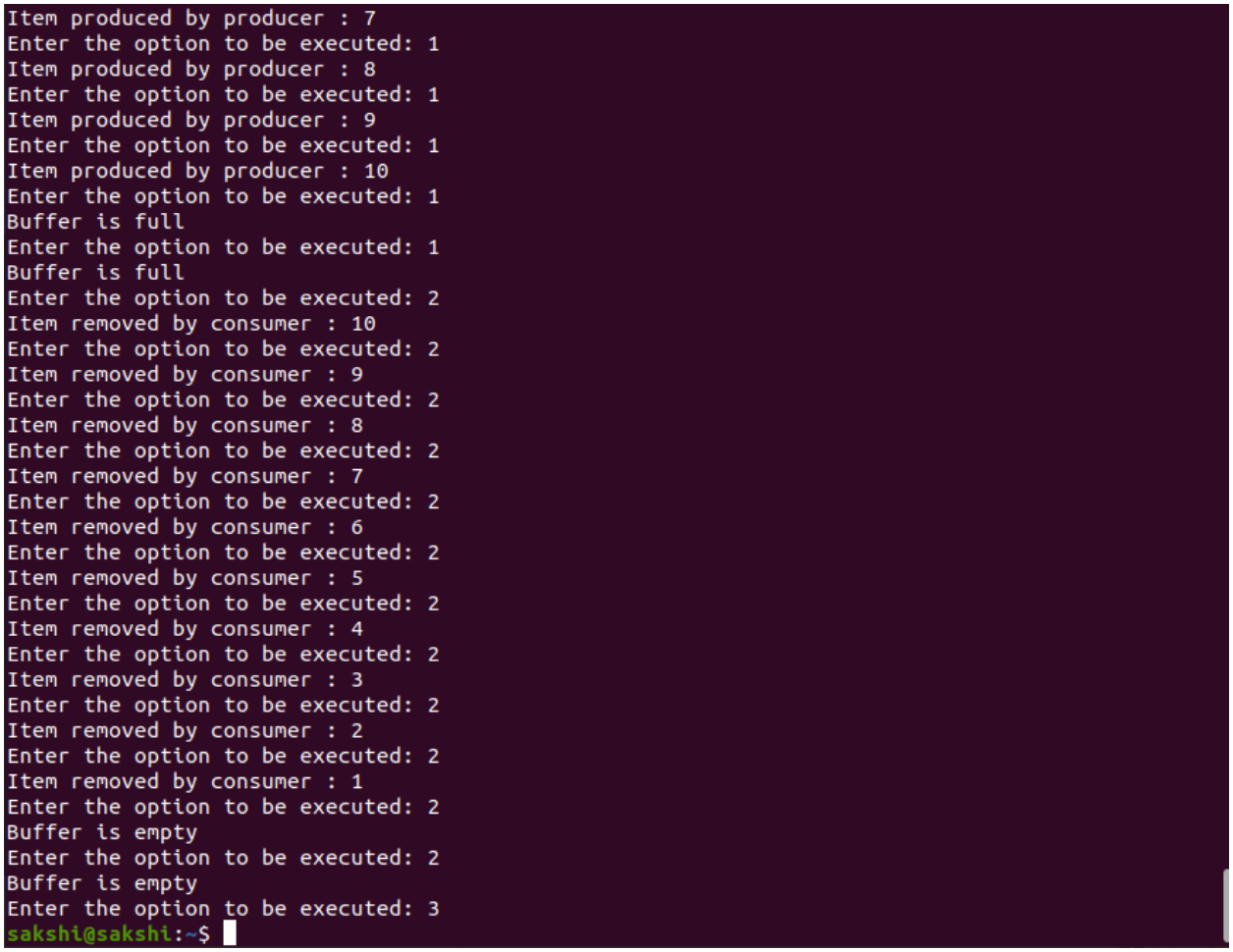
Question 3 :

Bonus Question

- **Output :**

```
sakshi@sakshi:~$ gcc pc.c -lpthread -lrt
sakshi@sakshi:~$ ./a.out

 1. Producer
 2. Consumer
 3. Exit
Enter the option to be executed: 1
Item produced by producer : 1
Enter the option to be executed: 2
Item removed by consumer : 1
Enter the option to be executed: 1
Item produced by producer : 1
Enter the option to be executed: 1
Item produced by producer : 2
Enter the option to be executed: 1
Item produced by producer : 3
Enter the option to be executed: 1
Item produced by producer : 4
Enter the option to be executed: 1
Item produced by producer : 5
Enter the option to be executed: 1
Item produced by producer : 6
Enter the option to be executed: 1
Item produced by producer : 7
Enter the option to be executed: 1
Item produced by producer : 8
Enter the option to be executed: 1
Item produced by producer : 9
Enter the option to be executed: 1
Item produced by producer : 10
Enter the option to be executed: 1
Buffer is full
Enter the option to be executed: 1
Buffer is full
Enter the option to be executed:
```



- **Description :**

Producer consumer problem is a basic problem of synchronization of concurrent processes. The processes that run concurrently using the same resources often face problems like deadlock. Here, both the process producer and consumer share a common fixed sized buffer (queue) In which producer generates the data and put into the buffer and simultaneously consumer deletes the data from the buffer.

Problem : The producer should not generate data when the buffer is full and the consumer should not perform deletion when the buffer is empty. Solution to this is that when the buffer is full, producer goes to sleep or replaces/discards the data not required. When the consumer detes any data , the consumer signals to the producer that the buffer is not full.The producer then can start generating data again . Similarly if the buffer is empty the producer should sleep and when the producer send the signal on adding data , the consumer can resume its work.

These can also be accomplished by two semaphores which signal the empty and full queue to the processes. The solution when not effective leads to deadlock in this situation as both process may wait for another process to be awakened.

Semaphores works with two signal wait(), signal() which helps in syncronisation of two processes and helps avoiding usage of same resources under critical section , here mutex variable is also used which has the same signalising feature while semaphore is more generalised mutex.