

# OS Endsem

Name : Dhruv Kabariya

Enrollment No : AU1940188

Q1:Print the following Pattern

A 1 a B 2 b C 3 c ... Y 25 y Z 26 z

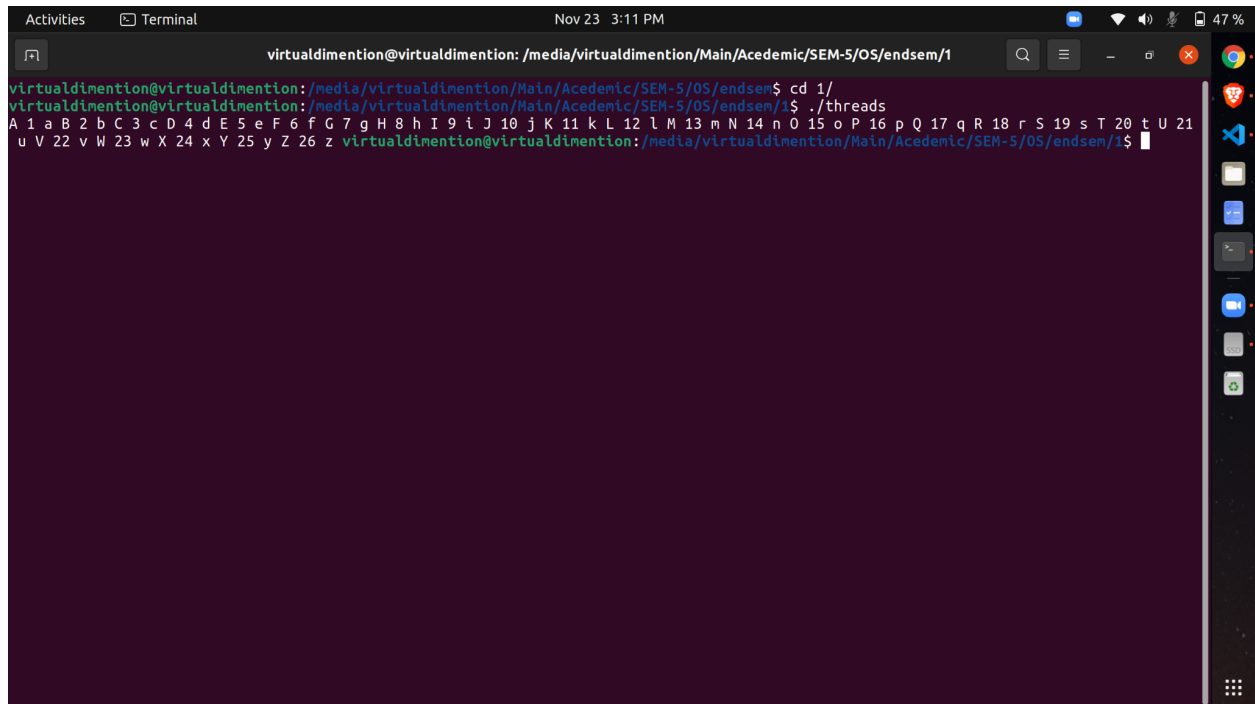
Using any one of the following concepts

- a. Multiprocesses (Hint: using 3 child processes)
- b. Multithreads (Hint: using 3 Threads)

Description : I choose b)

So to generate a given pattern in c language with thread I use the POSIX thread library (pthread as we are using Linux). We first declare 3 threads with data type "pthread\_t" and then create a thread by "pthread\_create" function. After creation we wait for the thread to complete and to do that we use the "pthread\_join" function which waits for the thread to complete. I do these process for 26 timer with for loop and pass loop counter in thread 2, counter + 64 in thread 1 and counter + 96 in thread 3 so we get first Capital then digit and then small letter.

Result:



Q2:Describe and implement any one of the following

a. Describe the RoundRobin (RR) and Modified RoundRobin (MRR) Algorithm. Also mention the difference between the results of both the algorithms and implement them both in C.

Reference: <https://ieeexplore.ieee.org/document/8392238>

Desc:

- For round robin we set process id in proc array , bust time in bust array then set time quantum of 4. We first find whether the bust is greater than tq if so we subtract the tq and assign remainder to bust and we separately count wait time in the variable which we use to increment the wait of the process. After calculating wait time for all processes we move to calculate turn around time.and last we print the result.
- For modified round robin we set process id in proc array , bust time in buts array then set time quantum of 4. Bust here when we complete the all process then we calculate time quantum. First we calculate mean and median bust time of the remaining process and if the mean is big then we multiply with max of remaining bust time and then take square root else we multiply median with max of remaining bust time and then take square root. Then we do the same process to calculate the wait time as we did in Round Robin .After that we calculate turnaround time by adding wait time and bust time of processes.

So in Round Robin we use static time quantum and in modified Round Robin we calculate dynamic time quantum when we complete all processes.So in MRR we get TQ as per process need. It help to reduce waiting time and reduce average turn around time.

Result :

```
Activities Terminal Nov 23 3:59 PM
virtualdimention@virtualdimention: /media/virtualdimention/Main/Acedemic/SEM-5/OS/endsem/2
virtualdimention@virtualdimention:/media/virtualdimention/Main/Acedemic/SEM-5/OS/endsem/2$ ./rr
Processes Burst Waiting Turn around
1 3 0 3
2 6 17 23
3 4 7 11
4 5 19 24
5 2 15 17
Average waiting time = 11.600000
Average turn around time = 15.600000
virtualdimention@virtualdimention:/media/virtualdimention/Main/Acedemic/SEM-5/OS/endsem/2$ ./mrr
Processes Burst Waiting Turn around
1 3 0 3
2 6 3 9
3 4 7 11
4 5 11 16
5 2 15 17
Average waiting time = 7.200000
Average turn around time = 11.200000
virtualdimention@virtualdimention:/media/virtualdimention/Main/Acedemic/SEM-5/OS/endsem/2$
```

Q3: Producer Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C.

Producer and Consumer Problem :

Let's say the producer process writes data to the buffer and the consumer process uses data from the buffer. Here both processes should not access the resource at same time. Also producer should not put data if buffer is full. Consumer can only access the buffer if it is not empty.

Now to solve the problem we use semaphore:

Semaphore is method where we use wait and signal method where it reduce the variable so if it is less then 0 then consumer can not use it. And wait for producer to fill the buffer. Whereas the job of the signal is to increment the value of variable. So when it is greater than define length of buffer produce can not write to buffer.

Now to solve the problem we use Mutex:

Mutex is a method where we keep track of full and empty also we prevent the mutual exclusion by mutex variable which is set to 0 if producer is writing or consumer is using. and when their process completes mutex set to 1. So We rise the full as we produce the data and we reduce the full as we consume the data.

Result:

```
Activities Terminal Nov 23 5:18 PM 73 %
virtualdimention@virtualdimention: /media/virtualdimention/Main/Acedemic/SEM-5/OS/endsem/3
virtualdimention@virtualdimention:/media/virtualdimention/Main/Acedemic/SEM-5/OS/endsem/3$ ./mutex
1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your What to do:1
Producer produces item 1
Enter your What to do:1
Producer produces item 2
Enter your What to do:1
Producer produces item 3
Enter your What to do:2
Consumer consumes item 3
Enter your What to do:2
Consumer consumes item 2
Enter your What to do:2
Consumer consumes item 1
Enter your What to do:2
Buffer is empty!
Enter your What to do:2
Buffer is empty!
Enter your What to do:3
virtualdimention@virtualdimention:/media/virtualdimention/Main/Acedemic/SEM-5/OS/endsem/3$
```