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)

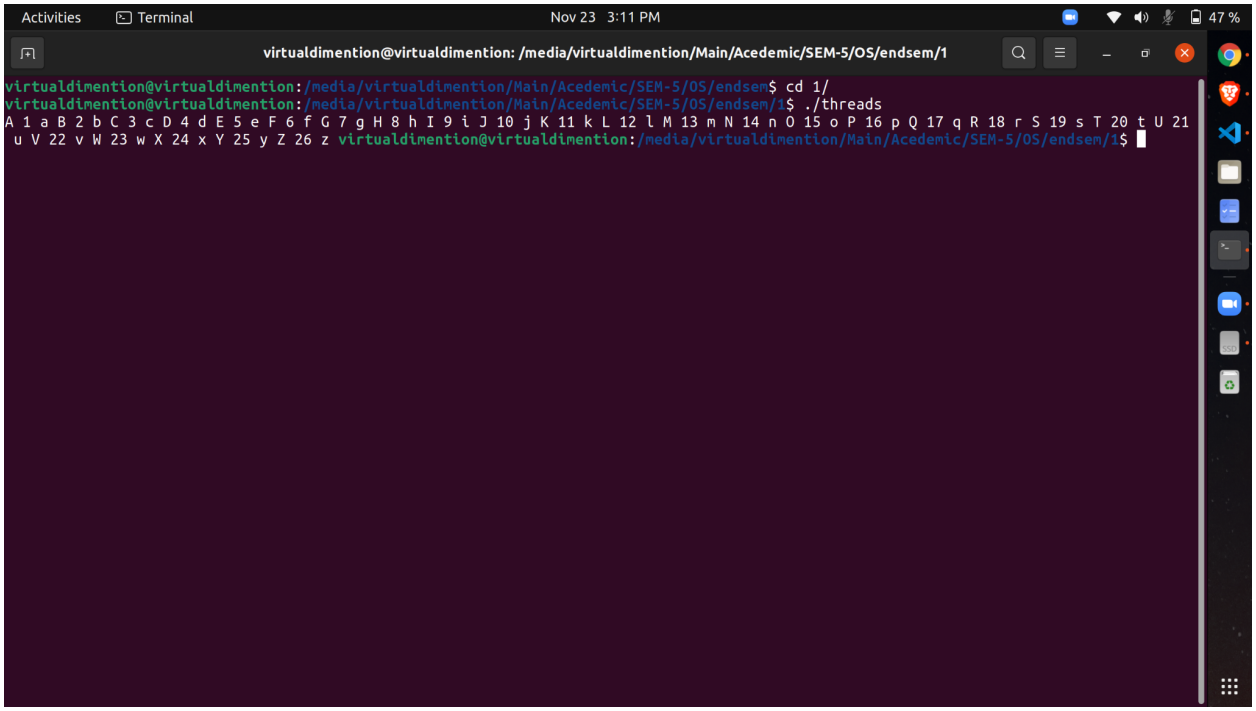
I choose b:

Code :

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>

void *printc(void *i){
    int *data = (int *)i;
    printf("%c ",*data);
}

void *printd(void *i){

   int *data = (int *)i;
    printf("%d ",*data);
}

int main(){


    pthread_t thread1,thread2,thread3;

    int it1,it2,it3;
```

```
    int i=0;


    for(i=1;i<27;i++){

        int small = i+96;
        int cap = i+64;

    it1 = pthread_create(&thread1,NULL,printc,(void*) &cap);
    pthread_join(thread1,NULL);

    it2 = pthread_create(&thread2,NULL,printd,(void*) &i);
    pthread_join(thread2,NULL);

    it3 = pthread_create(&thread3,NULL,printc,(void*) &small);
    pthread_join(thread3,NULL);

    }

    exit(0);

    return 0;
}
```

Results :



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
Code :

Round Robin :

```
#include<stdio.h>
#include<stdlib.h>

int max(int num1, int num2){
   return (num1 > num2 ) ? num1 : num2;
}

void waitTime(int proc[],int n,int bust[],int wait[]){
   int q = 4;
   int remain_bust[n] ;
   int remain_p = n;

   int i=0;
   int tw = 0;

   wait[0] = 0;
   remain_bust[0] = max(bust[0]-q,0);
   tw=bust[0] - remain_bust[0];

   bust[0] = remain_bust[0];


   if(!remain_bust[0]){
      remain_p--;
   }


   while (remain_p>0){
       i++;
      if(i==n){
         i=0;
      }

      if(!bust[i]){
```

```
            continue;
        }

        wait[i] = tw;

        remain_bust[i] = max(bust[i]-q,0);
        tw += (bust[i] - remain_bust[i]);

        bust[i] = remain_bust[i];

        if(!remain_bust[i]){
            remain_p--;
        }


    }

}

void tatTime(int proc[],int n,int bust[],int wait[],int tat[]){
 int i;
   for ( i = 0; i < n ; i++)
              tat[i] = bust[i] + wait[i];          // calculating turnaround time by adding
//    return 0;
}


void averageTime(
int proc[],int n,int bust[]

){
int wait_time[n],tat[n],total_wt=0,total_tat=0;
int bust_c[n];

int i=0;
for(i=0;i<n;i++){

    bust_c[i] = bust[i];
}
    waitTime(proc,n,bust_c,wait_time);
    tatTime(proc,n,bust,wait_time,tat);

    printf("Processes  Burst   Waiting Turn around \n");
```

```
    for ( i=0; i<n; i++) {
      total_wt = total_wt + wait_time[i];
      total_tat = total_tat + tat[i];              // Calculate total waiting time and total turn around time
      printf(" %d\t  %d\t\t %d \t%d\n", i+1, bust[i], wait_time[i], tat[i]);
    }
    printf("Average waiting time = %f\n", (float)total_wt / (float)n);
    printf("Average turn around time = %f\n", (float)total_tat / (float)n);



}



int main(){

    int pid[5] = {1,2,3,4,5};
    int n = 5;
    int bust_time[5] = { 3,6,4,5,2};
    int arrival[5] = {0,0,0,0,0};

    averageTime(pid,n,bust_time);

    return 0;
}
```

Modified Round Robin :

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int max(int num1, int num2){
    return (num1 > num2 ) ? num1 : num2;
}

float mean(int data[],int n){

    int j=0;
    float total=0;
    for(j=0;j<n;j++){
total +=data[j];
    }
    return total/n;
}
```

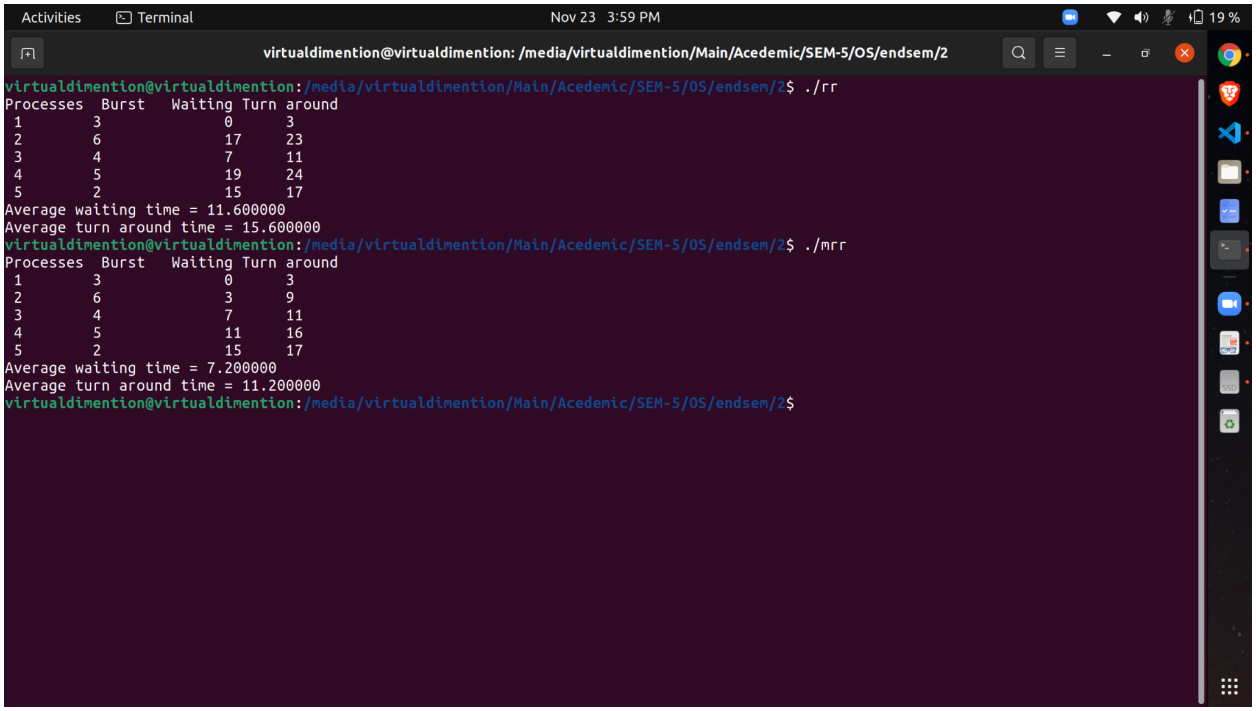
```
float median(int data[],int n){
    float median = 0;
    float mid=0;

    if(n%2 == 0){

        int temp=(n/2)-1;
        for(int i=0;i<n;i++){

            if(temp==i || (temp+1)==i) {
                mid=mid+data[i];
            }
        }
        mid=mid/2;
        }else  {
        int temp=(n/2);
            for(int i=0;i<n;i++) {
                if(temp==i){
                int mid=data[i];
                }
            }
        }


return mid;
}

int timeQ(int busts[],int n){
    int i=0;
    int max=busts[0];
    for(i=1;i<n;i++){
        if(max < busts[i] ){
            max = busts[i];
        }
    }

    float avg = mean(busts,n);
    float mid = median(busts,n);
    int tq = 0;
    if(avg>mid){
        tq= (int)sqrt((avg * max));
    }else{
        tq= (int)sqrt((mid * max));
```

```
    }

return tq;

}

void waitTime(int proc[],int n,int bust[],int wait[]){
    int tq = 4;
    int remain_bust[n] ;
    int remain_p = n;

    int i=0;
    int tw = 0;

    wait[0] = 0;
    remain_bust[0] = max(bust[0]-tq,0);
    tw=bust[0] - remain_bust[0];

    bust[0] = remain_bust[0];


    if(!remain_bust[0]){
        remain_p--;
    }



    while (remain_p>0){
         i++;
        if(i==n){
            i=0;
            tq = timeQ(bust,n);

        }

        if(!bust[i]){
            continue;
        }

        wait[i] = tw;

        remain_bust[i] = max(bust[i]-tq,0);
        tw += (bust[i] - remain_bust[i]);
```

```
        bust[i] = remain_bust[i];

        if(!remain_bust[i]){
            remain_p--;
        }


    }

}

void tatTime(int proc[],int n,int bust[],int wait[],int tat[]){
 int i;
   for ( i = 0; i < n ; i++)
              tat[i] = bust[i] + wait[i];          // calculating turnaround time by adding
}


void averageTime(
int proc[],int n,int bust[]

){
int wait_time[n],tat[n],total_wt=0,total_tat=0;
int bust_c[n];

int i=0;
for(i=0;i<n;i++){

    bust_c[i] = bust[i];
}
    waitTime(proc,n,bust_c,wait_time);
    tatTime(proc,n,bust,wait_time,tat);

    printf("Processes  Burst   Waiting Turn around \n");

    for ( i=0; i<n; i++) {
      total_wt = total_wt + wait_time[i];
      total_tat = total_tat + tat[i];              // Calculate total waiting time and total turn around time
      printf(" %d\t  %d\t\t %d \t%d\n", i+1, bust[i], wait_time[i], tat[i]);
    }
   printf("Average waiting time = %f\n", (float)total_wt / (float)n);
   printf("Average turn around time = %f\n", (float)total_tat / (float)n);
```

```
}


int main(){

    int pid[5] = {1,2,3,4,5};
    int n = 5;
    int bust_time[5] = { 3,6,4,5,2};
    int arrival[5] = {0,0,0,0,0};

    averageTime(pid,n,bust_time);

    return 0;
}
```

Result:



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

Semaphores :

```
#include<stdlib.h>
#include<stdio.h>
```

```
int S=0;
int out,in;// in used by procuder to put data and out used by consumer to consume the data
int n=7; //total buffer legth
int full=0; //full keep trake of no of full locations
int empty=7;// , empty keep track of no of empty slots
int P ;// it is process id.
int buffer[7]= {0};

int itemC;

void wait(int sema )  {
   while( sema <= 0) ;
   sema--;
}

void signal( int sema) {
   while(sema>n);
   sema++;
}

int Produce_item(int p){
return rand();
}

void producer( void )
{
  wait ( empty );
  wait(S);

  buffer[in] = Produce_item(P);
  in = (in + 1)% n  ;
  signal(S);
  signal(full);

}

void consumer(void)
{
  wait ( empty );
  wait(S);
  itemC = buffer[ out ];
out = ( out + 1 ) % n;
  printf("Consumer %d",itemC);
```

```
  signal(S);
  signal(empty);
}


int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");



    for (i = 1; i > 0; i++) {

        printf("\nEnter your What to do:");
        scanf("%d", &n);


        switch (n) {
        case 1:


            producer();
            break;

        case 2:

            consumer();
            break;

        case 3:
            exit(0);
            break;
        }
    }
}
```

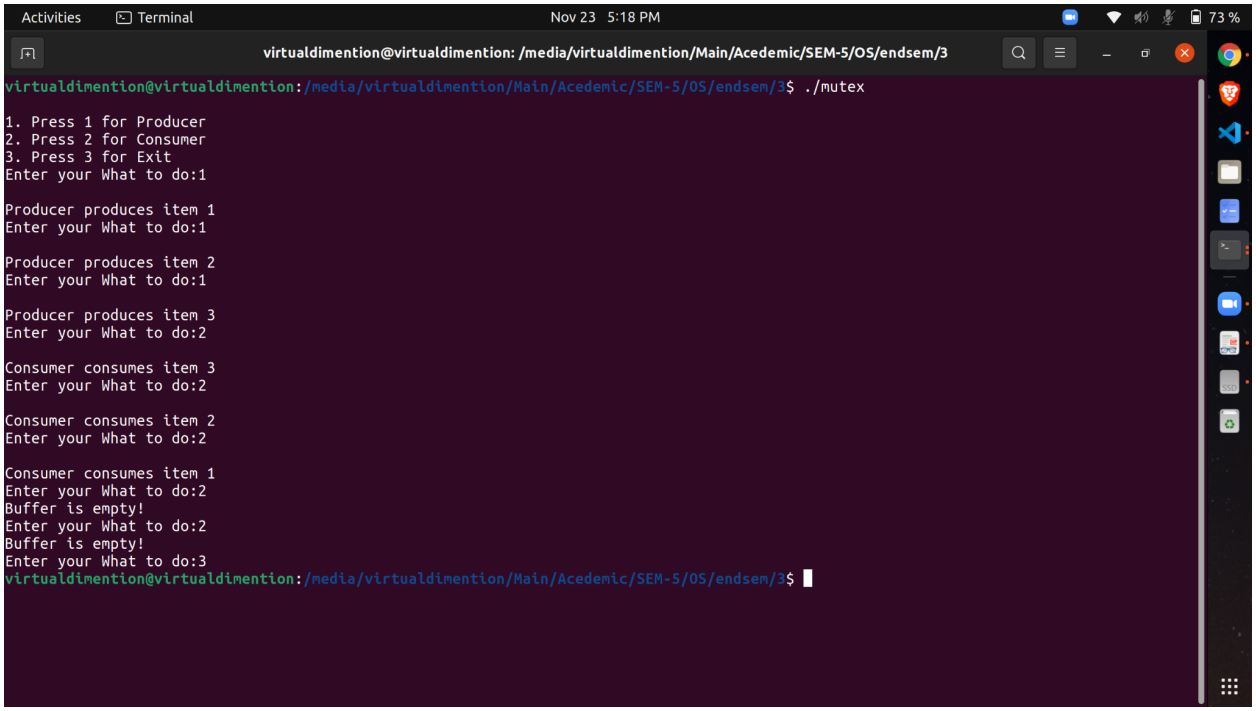
Mutex :

```
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10;

int n = 0;

void producer()
{
    --mutex;

    ++full;

    --empty;

    n++;
    printf("\nProducer produces item %d",n);

    ++mutex;
}


void consumer()
{
    --mutex;

    --full;

    ++empty;
    printf("\nConsumer consumes item %d",n);
    n--;

    ++mutex;
}

int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
```

```
        "\n3. Press 3 for Exit");



for (i = 1; i > 0; i++) {

    printf("\nEnter your What to do:");
    scanf("%d", &n);


    switch (n) {
    case 1:


        if ((mutex == 1)
            && (empty != 0)) {
            producer();
        }

        else {
            printf("Buffer is full!");
        }
        break;

    case 2:

        // consumer can  use
        if ((mutex == 1)
            && (full != 0)) {
            consumer();
        }


        else {
            printf("Buffer is empty!");
        }
        break;


    case 3:
        exit(0);
        break;
    }
}
```

}

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$