Name: Dhruvil Panchamia

Enrolment: AU1940285

End Sem Code

Q1

(b)

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<semaphore.h>
sem_t semaphore;
int loc = 0;

void *thread_function(void *arg){
   int i;
   for(i=0;i<26;i++){
      sem_wait(&semaphore);
      if(loc%3 != 1){
         i--;
      }else{
         printf("%d ",1+i);
         loc++;
      }
      sem_post(&semaphore);
   }
   pthread_exit(NULL);
}
```

```
void *thread_function2(void *arg){
   int i;
   for(i=0;i<26;i++){
      sem_wait(&semaphore);
      if(loc%3 != 0){
         i--;
      }else{
         printf("%c ",'A'+i);
         loc++;
      }
      sem_post(&semaphore);
   }
   pthread_exit(NULL);
}

void *thread_function3(void *arg){
   int i;
   for(i=0;i<26;i++){
      sem_wait(&semaphore);
      if(loc%3 != 2){
         i--;
      }else{
         printf("%c ",'a'+i);
         loc++;
      }
```

```
        sem_post(&semaphore);
    }
    pthread_exit(NULL);
}

int main(){



    pthread_t thread1, thread2, thread3;
    char start1 = 'A';
    char start2 = 'a';
    sem_init(&semaphore, 0, 1);
    pthread_create(&thread1,NULL,thread_function2,NULL);
    pthread_create(&thread2,NULL,thread_function,NULL);
    pthread_create(&thread3,NULL,thread_function3,NULL);
    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);
    pthread_join(thread3,NULL);
    sem_destroy(&semaphore);
    return 0;
}
```

Q2

(a)

Round Robin Process Scheduling

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

```
void main()
{
   int i, j, N, sum=0,count=0, q, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Enter Total number of process: ");
   scanf("%d", &N);
   j = N;
for(i=0; i<N; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is:");
scanf("%d", &at[i]);
printf(" \nBurst time is:");
scanf("%d", &bt[i]);
temp[i] = bt[i];
}

printf("Enter the Time Quantum for the process:");
scanf("%d", &q);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; j!=0; )
{
if(temp[i] <= q && temp[i] > 0)
{
   sum = sum + temp[i];
   temp[i] = 0;
```

```
    count=1;
    }
    else if(temp[i] > 0)
    {
        temp[i] = temp[i] - q;
        sum = sum + q;
    }
    if(temp[i]==0 && count==1)
    {
        j--;
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i],
sum-at[i]-bt[i]);
        wt = wt+sum-at[i]-bt[i];
        tat = tat+sum-at[i];
        count =0;
    }
    if(i==N-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
```

```
   }
}

avg_wt = wt * 1.0/N;
avg_tat = tat * 1.0/N;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}
```

Modified Round Robin Process Scheduling

```
#include <stdio.h>
#include <conio.h>
#include <stdio.h>
#include <conio.h>
#include<math.h>
#include<string.h>
 int wt[100],bt[100],at[100],tat[100],n,p[100];
 float awt[5],atat[5];
 int temp1,temp2,temp3,sqt,avg;
int main(){
printf("Enter Total Number of processes:");
       scanf("%d",&n);
       int x;
for(x=0;x<n;x++)
p[x]=x+1;
       for(x=0;x<n;x++)
```

```
        {
          printf("Enter Arrival Time of process %d:",x+1);
          scanf("%d",&at[x]);
          printf("Enter Burst Time of process %d:",x+1);
          scanf("%d",&bt[x]);
        }
for(x=0;x<5;x++)
{
awt[x]=0.0;
atat[x]=0.0;
}
int bt1[n],i,j,temp,qt;
  int b[n];
float twt,ttat;
for(i=0;i<n;i++)
bt1[i]=bt[i];
for(i=0;i<n;i++)
b[i]=bt[i];
int num=n;
int time=0;
int max;
int sum,t,a,ap;
ap=0;
        for (i = 0; i < n; i++)
        {
           for (j = 0; j < n-i- 1; j++)
```

```
   {
      if (bt[j] > bt[j + 1])
      {
          temp1 = bt[j];
          temp2 = p[j];
          temp3 = at[j];
          bt[j] = bt[j + 1];
          p[j] = p[j + 1];
          at[j] = at[j + 1];
          bt[j + 1] = temp1;
          p[j + 1] = temp2;
          at[j + 1] = temp3;
       }
     }
  }
max=bt[n-1];
sum=0;
for(i=0;i<n;i++)
{
   sum=sum+bt[i];
}
avg=sum/n;

qt=(avg+max)/2;
printf("\nDynamic Quantum time calculated: %d\n",qt);
```

```
while(num>0){
a=0;
max=0;
sum=0;
t=0;
for(i=0;i<n;i++){
if(at[i]<=time && b[i]!=0)
{
if(b[i]<qt)
{
t+=b[i];
b[i]=0;
}
else
{
t+=qt;
b[i]-=qt;
}
if(b[i]<qt && b[i]!=0)
{
t+=b[i];
b[i]=0;
}
if(b[i]==0){
wt[i]=(time+t)-bt1[i];
tat[i]=time+t;
```

```
num--;
}
}
}
time+=t;
}
printf("Processes\tWaitingtime\tTurnAroundTime\n");
for(j=1;j<=n;j++)
        {
            for(i=0;i<n;i++)
            {
                if(j==p[i])
printf("process %d\t%d\t\t%d\n",p[i],wt[i],tat[i]);
}
}

      twt=0;
      ttat=0;
for(i=0;i<n;i++)
{twt=twt+wt[i];}
awt[4]=twt/n;
for(i=0;i<n;i++)
{ttat=ttat+tat[i];}
atat[4]=(ttat/n);
}
```

Q3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
#include <semaphore.h>

#define THREAD_NUM 2
sem_t semEmpty;
sem_t semOccupied;

pthread_mutex_t mutexBuffer;
// Arbitrary sized buf
int buf[20];
int count = 0;

void* producer(void* args) {
   while (1) {

      int x = rand() % 100;
      sleep(0.1);
      printf("Produced %d",x);

      sem_wait(&semEmpty);
```

```
        pthread_mutex_lock(&mutexBuffer);
        buf[count] = x;
        count++;
        pthread_mutex_unlock(&mutexBuffer);
        // Post on the Occupied buffer
        sem_post(&semOccupied);
    }
}

void* consumer(void* args) {
    while (1) {
        int y;
        sem_wait(&semOccupied);
        pthread_mutex_lock(&mutexBuffer);
         // Remove from the buffer
        y = buf[count - 1];
        count--;
        pthread_mutex_unlock(&mutexBuffer);
         // Post on the empty buffer
        sem_post(&semEmpty);

        printf("Consumed %d\n", y);
        sleep(0.1);
    }
}
```

```
int main(int argc, char* argv[]) {
    srand(time(NULL));
    // Declaring a Thread array
    pthread_t threads[THREAD_NUM];
    pthread_mutex_init(&mutexBuffer, NULL);
    // INitializing semaphores
    sem_init(&semEmpty, 0, 10);
    sem_init(&semOccupied, 0, 0);
    int i;
    for (i = 0; i < THREAD_NUM; i++) {
        if (i > 0) {
            if (pthread_create(&threads[i], NULL, &producer, NULL) != 0) {
                perror("Thread creation failed");
            }
        }
        else {
            if (pthread_create(&threads[i], NULL, &consumer, NULL) != 0) {
                perror("Thread creation failed");
            }
        }
    }
    // Joining the Threads
    for (i = 0; i < THREAD_NUM; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("Failed to join thread");
        }
```

```
    }
    sem_destroy(&semEmpty);
    sem_destroy(&semOccupied);
    pthread_mutex_destroy(&mutexBuffer);
    return 0;
}
```