Jash Shah-AU1940286

Q-1.b

Code:

```
#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <semaphore.h>


sem_t semaphore;

int pos = 0;

int i=1;

char Ch='A';

char ch='a';


void *PrintpatternCH(void *arg)

{

   int i;

   for (i = 0; i < 26; i++){

      sem_wait(&semaphore);

      if (pos % 3 != 0){

         i--;

      }

      else{

         printf("%c ", 'A' + i);

         pos++;

      }

      sem_post(&semaphore);

   }

   pthread_exit(NULL);

}
```

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


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

int main()
{

    pthread_t thread1, thread2, thread3;
    // INitializing the semaphore
    sem_init(&semaphore, 0, 1);
    pthread_create(&thread1, NULL, PrintpatternCH, NULL);
    pthread_create(&thread2, NULL, Printpattern, NULL);
    pthread_create(&thread3, NULL, Printpatternch, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    sem_destroy(&semaphore);
    return 0;
}
```

Q-2.

RoundRobin (RR)

Code:

```
#include <stdio.h>

#include <conio.h>


void main()

{


   int i, n_process, sum = 0, cnt = 0, y, t_quantam, wt = 0, tat = 0, arrival_time[10], burst_time[10],
temp[10];

   float avg_wt, avg_tat;

   printf("Enter the number of process in the system: ");

   scanf("%d", &n_process);

   y = n_process;


   for (i = 0; i <n_process; i++)

   {

      printf("\nEnter the Arrival and Burst time of the Process %d\n", i + 1);

      printf("Arrival time is: ");

      scanf("%d", &arrival_time[i]);

      printf("Burst time is: ");

      scanf("%d", &burst_time[i]);

      temp[i] = burst_time[i];

   }


   printf("\nEnter the Time Quantum for the process: ");

   scanf("%d", & t_quantam);


   printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

   for (sum = 0, i = 0; y != 0;)
```

```
    {
      if (temp[i] <=  t_quantam && temp[i] > 0)
      {
        sum = sum + temp[i];
        temp[i] = 0;
        cnt = 1;
      }
      else if (temp[i] > 0)
      {
        temp[i] = temp[i] -  t_quantam;
        sum = sum +  t_quantam;
      }
      if (temp[i] == 0 && cnt == 1)
      {
        y--;
        printf("\nProcess No %d \t\t %d\t\t\t %d\t\t\t %d", i + 1, burst_time[i], sum -arrival_time[i], sum - arrival_time[i] - burst_time[i]);
        wt =wt + sum - arrival_time[i] - burst_time[i];
        tat = tat + sum - arrival_time[i];
        cnt = 0;
      }
      if (i == n_process - 1)
      {
        i = 0;
      }
      else if (arrival_time[i + 1] <= sum)
      {
        i++;
      }
      else
      {
            i = 0;
        }
    }

    avg_wt =wt * 1.0 / n_process;
    avg_tat = tat * 1.0 /n_process;
    printf("\nAverage Turn Around Time: %f", avg_wt);
    printf("\nAverage Waiting Time: %f", avg_tat);
}
```

Modified RoundRobin (MRR)

Code:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
int wt[100], burst_time[100], arrival_time[100], tat[100], n, p[100];
float avg_wt[5], avg_tat[5];
int temp1, temp2, temp3, sqt, avg;
int main()
{
   printf("Enter Number of processes in the system: ");
   scanf("%d", &n);
   int i;
   for (i = 0; i < n; i++)
      p[i] = i + 1;
   for (i = 0; i < n; i++)
   {
      printf("\nEnter the Arrival and Burst time of the Process %d\n", i + 1);
      printf("Arrival time is: ");
      scanf("%d", &arrival_time[i]);
      printf("Burst time is: ");
      scanf("%d", &burst_time[i]);
   }
   for (i = 0; i < 5; i++)
   {
      avg_wt[i] = 0.0;
      avg_tat[i] = 0.0;
   }
   int bt1[n], j, temp, t_quantum;
   int b[n];
```

```
float total_wt, total_tat;
for (i = 0; i < n; i++)
    bt1[i] = burst_time[i];
for (i = 0; i < n; i++)
    b[i] = burst_time[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 (burst_time[j] > burst_time[j + 1])
        {
            temp1 = burst_time[j];
            temp2 = p[j];
            temp3 = arrival_time[j];
            burst_time[j] = burst_time[j + 1];
            p[j] = p[j + 1];
            arrival_time[j] = arrival_time[j + 1];
            burst_time[j + 1] = temp1;
            p[j + 1] = temp2;
            arrival_time[j + 1] = temp3;
        }
    }
}
max = burst_time[n - 1];
sum = 0;
```

```
for (i = 0; i < n; i++)
{
    sum = sum + burst_time[i];
}
avg = sum / n;



t_quantum = (avg + max) / 2;
printf("\nQuantum time calculated is : %d\n", t_quantum);

while (num > 0)
{
    a = 0;
    max = 0;
    sum = 0;
    t = 0;
    for (i = 0; i < n; i++)
    {
        if (arrival_time[i] <= time && b[i] != 0)
        {
            if (b[i] < t_quantum)
            {
                t += b[i];
                b[i] = 0;
            }
            else
            {
                t += t_quantum;
                b[i] -= t_quantum;
            }
            if (b[i] < t_quantum && 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 no %d\t%d\t\t%d\n", p[i], wt[i], tat[i]);
    }
}

total_wt = 0;
total_tat = 0;
for (i = 0; i < n; i++)
{
    total_wt = total_wt + wt[i];
}
```

```
    avg_wt[4] = total_wt / n;

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

    {

        total_tat = total_tat + tat[i];

    }

    avg_tat[4] = (total_tat / n);

}
```