Astha Patel - AU1940312
CSE-332 : Operating System
Section-2
End sem Lab exam - Code

Question 1 (b):

Code:

```
// ASTHA PATEL - AU1940312
//QUESTION 1 (b)

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

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t* cond = NULL;

int threads;
volatile int cnt = 0;


void* foo(void* arg) //to synchronize threads
{

   int turn = *(int*)arg; //to identify thread


      pthread_mutex_lock(&mutex);

      if (turn != cnt) //which thread to enter into critical action
      {

         // put all thread except one thread in waiting
         // state
         pthread_cond_wait(&cond[turn], &mutex);
      }
```

```
        printf("%d ", turn + 1);


        if (cnt < threads - 1) //thread schedular
        {
            cnt++;
        }
        else
        {
            cnt = 0;
        }


        pthread_cond_signal(&cond[cnt]);
        pthread_mutex_unlock(&mutex);


    return NULL;
}

int main()
{
    pthread_t* tid;
    volatile int i;
    int* arr;

    printf("\nEnter number of threads: ");
    scanf("%d", &threads);

    // allocate memory to cond (conditional variable),
    // thread id's and array of size threads
    cond = (pthread_cond_t*)malloc(sizeof(pthread_cond_t)
                                   * threads);
    tid = (pthread_t*)malloc(sizeof(pthread_t) * threads);
    arr = (int*)malloc(sizeof(int) * threads);

    // Initialize cond(conditional variable)
    for (int i = 0; i < threads; i++) {
```

```
        if (pthread_cond_init(&cond[i], NULL) != 0) {
            perror("pthread_cond_init() error");
            exit(1);
        }
    }

    // create threads
    for (i = 0; i < threads; i++)
    {
        arr[i] = i;
        pthread_create(&tid[i], NULL, foo, (void*)&arr[i]);
    }

    // wait for thread
    for (i = 0; i < threads; i++)
    {
        pthread_join(tid[i], NULL);
    }

    return 0;
}
```

Question 2 (a):

Code:

ROUND ROBIN

```
//ASTHA PATEL - AU1940312
//QUESTION-2 (a)
//ROUND ROBIN

#include<stdio.h>

int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int WT = 0, TAT = 0, AT[10], BT[10], temp[10];
    float average_wait_time, average_turnaround_time;
```

```
printf("Enter total number of processes:");
scanf("%d", &limit);
x = limit;

printf("\nPROCESS DETAILS ");

printf("\n-----------------------------------------------------");

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

      printf("\n Process : %d", i + 1);

     printf("\nArrival Time: ");

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

     printf("Burst Time: ");

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

     temp[i] = BT[i];

     printf("\n-----------------------------------------------------");
}

printf("\nTime Quantum: ");

scanf("%d", &time_quantum);

printf("\n-----------------------------------------------------");

printf("\nProcess ID \t Burst Time \t Turnaround Time \t Waiting Time \n");

for(total = 0, i = 0; x != 0;)
{
     if(temp[i] <= time_quantum && temp[i] > 0)
     {
          total = total + temp[i];
          temp[i] = 0;
```

```
            counter = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - time_quantum;
            total = total + time_quantum;
        }

        if(temp[i] == 0 && counter == 1)
        {
            x--;
            printf("\n%d\t\t%d\t\t %d\t\t\t %d", i + 1, BT[i], total - AT[i], total - AT[i] - BT[i]);
            WT = WT + total - AT[i] - BT[i];
            TAT = TAT + total - AT[i];
            counter = 0;
        }

        if(i == limit - 1)
        {
            i = 0;
        }
        else if(AT[i + 1] <= total)
        {
            i++;
        }
        else
        {
            i = 0;
        }
    }

    printf("\n---------------------------------------------------------------------");

    average_wait_time = WT * 1.0 / limit;
    average_turnaround_time = TAT * 1.0 / limit;
    printf("\n\nAverage Waiting Time:  %f", average_wait_time);
    printf("\nAverage Turnaround Time:  %f\n", average_turnaround_time);
    return 0;

}
```

MODIFIED ROUND ROBIN

```
//ASTHA PATEL - AU1940312
//QUESTION-2 (a)
//Modified RR

#include<stdio.h>
struct process
{
   int WT,AT,BT,TAT,PT;
};

struct process a[10];

int main()
{
   int n,temp[10],t,count=0,short_p;
   float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
   printf("Enter total number of the processes: ");
   scanf("%d",&n);
   printf("\nEnter the arrival time,burst time and priority of the process");
   printf("\n-------------------------------------------------");

   for(int i=0;i<n;i++)
   {
      printf("\n Process : %d", i + 1);

         printf("\nArrival Time: ");

         scanf("%d", &a[i].AT);

         printf("Burst Time: ");

         scanf("%d", &a[i].BT);

         printf("Priority: ");

         scanf("%d", &a[i].PT);
```

```
   // copying the burst time in
   // a temp array fot futher use
   temp[i]=a[i].BT;
}


a[9].PT=10000;

for(t=0;count!=n;t++)
{
   short_p=9;
   for(int i=0;i<n;i++)
   {
      if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
      {
         short_p=i;
      }
   }

   a[short_p].BT=a[short_p].BT-1;


   if(a[short_p].BT==0)   // if any process is completed
   {
       count++; // one process is completed : count is incremented

      a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
      a[short_p].TAT=t+1-a[short_p].AT;

      // total calculation
      total_WT=total_WT+a[short_p].WT;
      total_TAT=total_TAT+a[short_p].TAT;

   }
}

Avg_WT=total_WT/n;
Avg_TAT=total_TAT/n;

// printing of the answer
```

```
    printf("\nProcess ID \t Burst Time \t Turnaround Time \t");
    for(int i=0;i<n;i++)
    {
        printf("\n%d\t\t%d\t\t %d",i+1,a[i].WT,a[i].TAT);
    }

    printf("\n--------------------------------------------------");
    printf("\nAverage waiting time %f\n",Avg_WT);
    printf("\nAverage turnaround time %f\n",Avg_TAT);

    return 0;
}
```

Question 3:

Code:

PRODUCER-CONSUMER PROBLEM

```
//ASTHA PATEL  - AU1940312
//QUESTION 3

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

#define MaxItems 5    // Max items a producer can produce/consumer can consume
#define BufferSize 5 // Size of the buffer

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *p_no)
{
    int item;
    for(int i = 0; i < MaxItems; i++) {
```

```
        item = rand();  // Produces a random item
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)p_no),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
void *consumer(void *c_no)
{
    for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)c_no),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}

int main()
{

    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);

    int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

    for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }
```

```
    for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;

}
```