AU1940314
Kareena Matwani

END-SEMESTER_PRACTICAL_EXAM CODE

Question1

```
/*
Question-1: Using Multi-threads print pattern
Kareena Matwani AU1940314
*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int threads;
volatile int count = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t* condition = NULL;

void* foo(void* arg)
{
```

```
int turn = *(int*)arg;

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



        if (count < threads - 1) {
                count++;
        }
        else {
                count = 0;
        }


        pthread_cond_signal(&condition[count]);



        pthread_mutex_unlock(&mutex);



return NULL;
```

```
}

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

        printf("\nEnter number 3 for the process to execute further: ");
        scanf("%d", &threads);

        condition = (pthread_cond_t*)malloc(sizeof(pthread_cond_t)
                                                        * threads);
        threadid = (pthread_t*)malloc(sizeof(pthread_t) * threads);

        arr = (int*)malloc(sizeof(int) * threads);

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


                                        if (pthread_cond_init(&condition[i], NULL)
```

```
!= 0) {
                                        perror("pthread_cond_init() error");
                                        exit(1);
                                }
        }

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

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

                                pthread_join(threadid[i], NULL);
        }

        return 0;
}
```

Question2

```
/*
 Question-2: round robin
Kareena Matwani AU1940314
*/
#include<stdio.h>


int main()
{
    int i, limit, total = 0, x, counter = 0, time;



    int WT = 0, TAT = 0, AT[10], BT[10], temp[10];


    float average_wait_time, average_turnaround_time;




    printf("give number of processes");
```

```
scanf("%d", &limit);
x = limit;
for(i = 0; i < limit; i++)
{
                                        printf("\n Process : %d", i + 1);

                                        printf("\n Arrival Time: ");

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

                                        printf("Burst Time: ");
```

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

                                        temp[i] = BT[i];

                                }

    printf("\nTime Quantum: ");

    scanf("%d", &time);
    printf("\nProcess ID  \t Waiting Time \tBurst Time \t Turnaround Time  \n");

    for(total = 0, i = 0; x != 0;)
    {
                                                if(temp[i] <= time && temp[i] > 0)
                                                {
                                                                        total =
total + temp[i];
                                                                        temp[i] =
0;
                                                                        counter =
1;
```

```
                                                    }
                                                    else if(temp[i] > 0)
                                                    {
                                                                temp[i] =
temp[i] - time;
                                                                total =
total + time;
                                                    }
                                                    if(temp[i] == 0 && counter == 1)
                                                    {
                                                                x--;
printf("\n%d\t\t%d\t\t %d\t\t\t %d", i + 1,total - AT[i] - BT[i], BT[i], total - AT[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;
                                        }
    }
    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 RR code

```
/*
 Question-2: modified roundrobin
```

```
Kareena Matwani AU1940314

*/

#include<stdio.h>

struct process

{

   int WaitTime,ArrivalTime,BurstTime,TAT,PT;

};


struct process a[10];


int main()

{

   int n,temp[10],t,count=0,short_p;

   printf("please enter the nuber of processes to be executed: ");

   scanf("%d",&n);

   float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;


   printf("\ngive burst time arrival time and priority to the process");

printf("\n==========================================================
======");
```

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

                                           printf("\nAT: ");

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

                                           printf("Burst Time: ");

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

                                           printf("Process priority ");

                                           scanf("%d", &a[i].PT);
                                        temp[i]=a[i].BurstTime;
}
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].ArrivalTime<=t
&& a[i].BurstTime>0)
                                   {
                                        short_p=i;
                                   }
                                }

                                a[short_p].BurstTime=a[short_p].BurstTime-1;
                                if(a[short_p].BurstTime==0)
                                {

                                   count++;

a[short_p].WaitTime=t+1-a[short_p].ArrivalTime-temp[short_p];
                                   a[short_p].TAT=t+1-a[short_p].ArrivalTime;



                                   total_WT=total_WT+a[short_p].WaitTime;
                                   total_TAT=total_TAT+a[short_p].TAT;
```

```
                                        }
    }

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

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

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

    return 0;
}
```

Question3

```
/* Question-3: Producer-consumer problem using semaphores and mutex

Kareena Matwani AU1940314

*/



//libraries declarations

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#include <semaphore.h>




#define Maximum 5

#define Buffer 5



sem_t empty;

pthread_mutex_t mutex;

sem_t full;
```

```
int in = 0;

int out = 0;

int buffer[Buffer];




void *consumer(void *cusno)

{

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

                            sem_wait(&full);

                            pthread_mutex_lock(&mutex);

                            int item = buffer[out];

                            printf("Consumer %d:removing item number%d from
%d\n",*((int *)cusno),item, out);

                            out = (out+1)%Buffer;

                            pthread_mutex_unlock(&mutex);

                            sem_post(&empty);

                        }

}
```

```
void *producer(void *prono)

{

                        int item;

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

                            item = rand();

                            sem_wait(&empty);

                            pthread_mutex_lock(&mutex);

                            buffer[in] = item;

                            printf("Producer %d: please insert item %d at %d\n", *((int
*)prono),buffer[in],in);

                            in = (in+1)%Buffer;

                            pthread_mutex_unlock(&mutex);

                            sem_post(&full);

            }

}

int main()

{



                               pthread_t pro[5],con[5];
```

```
                pthread_mutex_init(&mutex, NULL);

                sem_init(&empty,0,Buffer);

                sem_init(&full,0,0);


                int a[5] = {1,2,3,4,5};


                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;
}
```