OS End Sem lab exam
Name : Krutin Rathod
Roll No : AU1940261

Question - 1

```c
#include <pthread.h> // need to include compulsarily in order to use Threads
#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 currentThread = 0;
// Threds will be synchronized
void *function(void *arg)
{
   int turn = *(int *)arg;
   pthread_mutex_lock(&mutex);
   if (turn != currentThread)
   {
       pthread_cond_wait(&cond[turn], &mutex);
   }

   char capitalLetter = turn + 65;
   char lowercaseLetter = turn + 97;
   printf("%c %d %c ", capitalLetter, turn + 1, lowercaseLetter);

   if (currentThread > threads - 1)
   {
       currentThread = 0;
   }
   else
   {
```

```
        currentThread++;
    }

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

    return NULL;
}

int main()
{
    pthread_t *tid;
    volatile int i;
    int *arr;
    threads = 26;
    cond = (pthread_cond_t *)malloc(sizeof(pthread_cond_t) * threads);
    tid = (pthread_t *)malloc(sizeof(pthread_t) * threads);
    arr = (int *)malloc(sizeof(int) * threads);

    for (int i = 0; i < threads; i++)
    {
        if (pthread_cond_init(&cond[i], NULL) != 0)
        {
            perror("pthread_cond_init() error");
            exit(1);
        }
    }
    for (i = 0; i < threads; i++)
    {
        arr[i] = i;
        pthread_create(&tid[i], NULL, function, (void *)&arr[i]);
    }
    for (i = 0; i < threads; i++)
    {
        pthread_join(tid[i], NULL);
    }
```

```
    return 0;
}
```

Question - 2

```
#include <stdio.h>
int main()
{
      int totalProcesses, sum, count, temp, timeQuantum, waitingTime,
turnAroundTime;
    int arrivalTime[10], burstTime[10], temporary[10];
    waitingTime = turnAroundTime = 0;
    count = sum = 0;
    printf("Please enter the total no. of processes : ");
    scanf("%d", &totalProcesses);
    printf("\n");
    temp = totalProcesses;
    int i;

    // Now we will ask details like arrival time, burst time from the user
    for (i = 0; i < totalProcesses; i++)
    {
        printf("Please enter arrival and burst time for process no. %d \n",
i + 1);
        printf("\n");
        printf("Enter arrival time : ");
        scanf("%d", &arrivalTime[i]);
        printf("Enter burst time : ");
        scanf("%d", &burstTime[i]);
        printf("\n");
        temporary[i] = burstTime[i]; // Temporary array used to store burst
time
    }
    // Accept the Time qunatum
    printf("Enter the Time Quantum for the process: ");
    scanf("%d", &timeQuantum);
```

```
    printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
    for (sum = 0, i = 0; temp != 0;)
    {
        if (temporary[i] <= timeQuantum && temporary[i] > 0)
        {
            sum = sum + temporary[i];
            temporary[i] = 0;
            count = 1;
        }
        else if (temporary[i] > 0)
        {
            temporary[i] = temporary[i] - timeQuantum;
            sum = sum + timeQuantum;
        }
        if (temporary[i] == 0 && count == 1)
        {
            temp--;
               printf("\nNo =    %d   \t\t %d\t\t\t\t %d\t\t\t %d", i + 1,
burstTime[i], sum - arrivalTime[i], sum - arrivalTime[i] - burstTime[i]);
                    waitingTime  =  waitingTime  +  sum  -  arrivalTime[i]  -
burstTime[i];
            turnAroundTime = turnAroundTime + sum - arrivalTime[i];
            count = 0;
        }
        if (i == totalProcesses - 1)
        {
            i = 0;
        }
        else if (arrivalTime[i + 1] <= sum)
        {
            i++;
        }
        else
        {
            i = 0;
        }
```

```
    }
}
```

Question - 3

```
const int sizeofbuffer = 20; // size of the buffer
semaphore s = 1, n = 0, e = sizeofbuffer;
void producer()
{
    while (1)
    {
        produce();
        semWait(e);
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
void consumer()
{
    while (1)
    {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
void main()
{
    parbegin(producer, consumer);
}
```