NAME - SHUBHAM TIWARI

ROLL - 20051488

OS ASSIGNMENT

Question : - Implement the following three question in C programming language using semaphore.

1. In a railway station, there are 3 rest rooms. In each rest room, only one passenger is allowed to take rest at a time. Write a solution using semaphore to synchronize among the passenger to avoid the race condition for accessing the rooms.

CODE : -

```
/*----In a railway station, there are 3 rest rooms. In each rest room, only
one passenger is allowed to take rest at a time. Write a solution using
semaphore to synchronize among the passenger to avoid the race
condition for accessing the rooms.----*/


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


enum {empty,pack};


//Semaphore
```

```
sem_t passenger[10];


//Mutex
pthread_mutex_t mutex;


//Global Variable
int room_state[3] = {empty};

int room_occupied[10] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

int num = 0;


//USER DEFINED FUNCTIONS
void *passenger_work(void *args);
void occupy_room(int );
void check_room_status(int );
void vacant_room(int );
void take_rest(int );

//Main
int main()
{
 int i;
 int *a;

 pthread_t Thread[10];


 //Initializing mutex
 pthread_mutex_init(&mutex,NULL);


 //Initializing semaphore passenger
 for(i=0;i<10;i++)
        sem_init(&passenger[i],0,0);
```

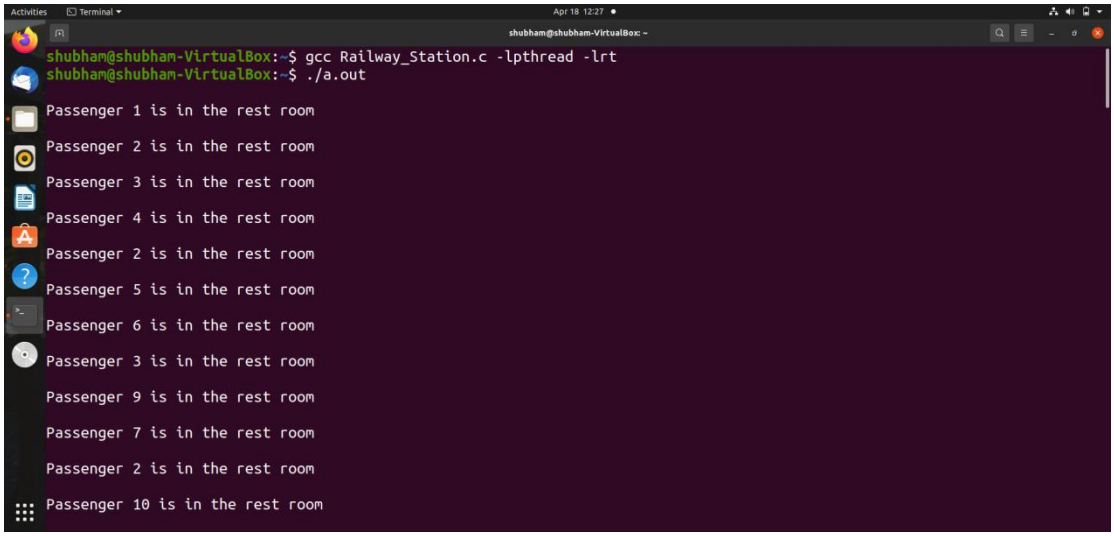
```
 //Thread Creation
 for(i=0;i<10;i++)
 {
  a=(int *)malloc(sizeof(int ));
  *a = i;

  if(pthread_create(&Thread[i],NULL,&passenger_work,a)!=0)
  {
   perror("Failed to create thread\n");
  }

 }


 //Joining the threads
 for(i=0;i<10;i++)
 {
   if(pthread_join(Thread[i],NULL)!= 0)
   {
    perror("Failed to join thread\n");
   }

 }


 //Destroying the semaphore
 sem_destroy(passenger);

 //Destroying the mutex
 pthread_mutex_destroy(&mutex);

 free(a);        //Free the space

 return 0;
}


//Passenger_Work Function
void *passenger_work(void *args)
{
```

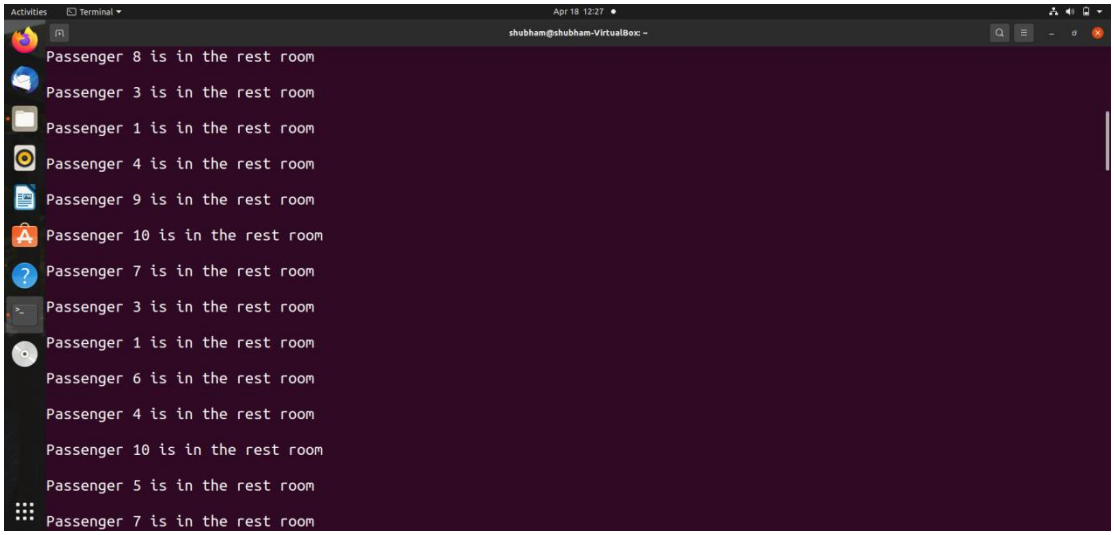
```
 while(1)
 {
  occupy_room(*(int *)args);

  take_rest(*(int *)args);

  vacant_room(*(int *)args);
 }
}


//Occupy_room Function
void occupy_room(int i)
{
 pthread_mutex_lock(&mutex);          //P(mutex)
   check_room_status(i);
 pthread_mutex_unlock(&mutex);     //V(mutex)

 sem_wait(&passenger[i]);         // p(passenger[i])

}


//Check_Room_Status Function
void check_room_status(int i)
{
 int rm_no = -1;      // -1 because no room is occupied

 if(room_state[0] == empty)
 {
   rm_no = 0;          // Room '0' is occupied
 }

 else if(room_state[1] == empty)
 {
   rm_no = 1;          //Room '1' is occupied
 }

 else if(room_state[2] == empty)
 {
```

```
  rm_no = 2;          //Room '2' is occupied
 }

 if(rm_no != -1)
 {
  room_occupied[i] = rm_no;     //It assigns the room , the passenger is
occupying.

  room_state[rm_no] = pack;     //Room state is changed to pack


  sem_post(&passenger[i]);        // v(passenger[i])
 }

}


//Vacant_Room Function
void vacant_room(int i)
{
  int count;

  pthread_mutex_lock(&mutex);                    //p(mutex)

    room_state[room_occupied[i]] = empty;    //Passenger vacated the
                                             //room

    room_occupied[i] = -1;          //Since the passenger vacated the room


    //Looking for the passengers who are waiting....
    for(count=0;count<10;num=(num+1)%10,count++)
    {

     if(room_occupied[num]==-1 && num!=i)
     {
                 check_room_status(num);

                 num=(num+1)%10;      //Similar to queue traversal
```

```
        }

    }

  pthread_mutex_unlock(&mutex);     //v(mutex)

}


//Take_Rest Function
void take_rest(int i)
{

  printf("\nPassenger %d is in the rest room\n",(i+1));

  sleep(1);

}
```

OUTPUT : -





---

2. In a civilized society, a gentle man lives with his spouse and his elderly parents. Due to old age, his parents cannot be left alone in the house. So, at least any one of the spouse must be available in the house. Write a synchronize solution using semaphore for this problem.

CODE : -

```
/*------In a civilized society, a gentle man lives with his spouse and his elderly parents.
Due to old age, his parents cannot be left alone in the house. So, at least any one of the spouse must be available in the house. Write a synchronize solution using semaphore for this problem.----------*/

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

```
#define SPOUSE 2   //Either wife or husband


enum {out,in};        //enum for out and in status;


int spouse_state[2] = {out};        //An array of size 2 for the current
status(out,in) of spouse


//Mutex
pthread_mutex_t mutex;

//Semaphore
sem_t spouse[SPOUSE];


//USER DEFINED FUNCTIONS
void *spouse_work(void *args);
void enter_house(int );
void check_spouse_status(int );
void takecare_parent(int );
void leave_house(int );


//Main
int main()
{
  int i;
  int *a;

  pthread_t spouse_thread[SPOUSE];


  //Initializing mutex
  pthread_mutex_init(&mutex,NULL);
```

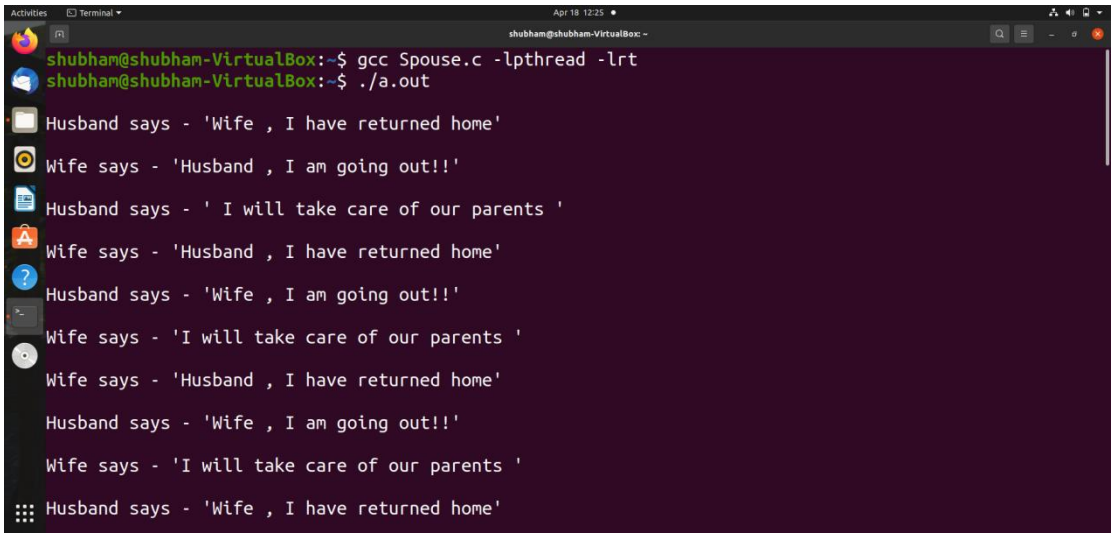
```
//Initializing semaphore spouse
 for(i=0;i<SPOUSE;i++)
        sem_init(&spouse[i],0,0);


 //Thread Creation
 for(i=0;i<SPOUSE;i++)
 {
  a=(int *)malloc(sizeof(int));

  *a = i;

  //Creating thread
  if(i>0)
  {
   if(pthread_create(&spouse_thread[i] ,NULL,&spouse_work ,a) !=0)
   {
     perror("Failed to create thread\n");
   }

  }

  else
  {
    if(pthread_create(&spouse_thread[i],NULL,&spouse_work, a) !=0)
    {
      perror("Failed to create thread\n");
    }
  }

 }


 //Joining the threads
 for(i=0;i<SPOUSE;i++)
 {
  if(pthread_join(spouse_thread[i],NULL) != 0)
  {
    perror("Failed to join thread");
  }
```

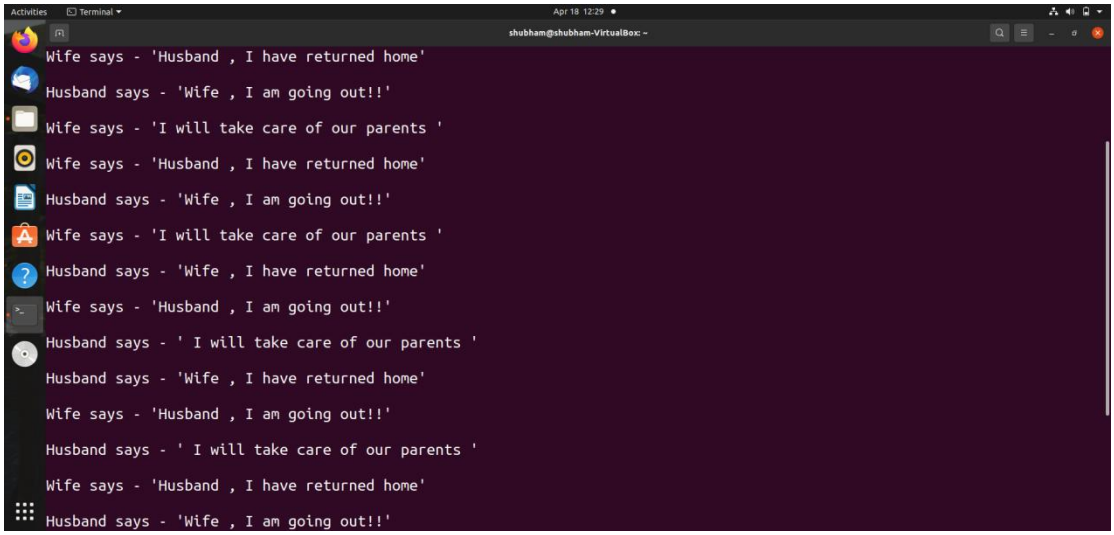
```
  }

  //Destroying the semaphore
  sem_destroy(spouse);

  //Destroying the mutex
  pthread_mutex_destroy(&mutex);


  free(a);       //Free the space

  return 0;
}



//Spouse_Work Function
void *spouse_work(void *args)
{
  while(1)     //Infinite Loop
  {
          enter_house(*(int *)args);
          takecare_parent(*(int *)args);
          leave_house(*(int *)args);

  }

}


//Enter_house Function
void enter_house(int i)
{
  pthread_mutex_lock(&mutex);           //P(mutex)

          spouse_state[i]=in;          //Spouse 'i' returned back to home

          if(i==0)
                  printf("\nHusband says - 'Wife , I have returned home'\n");
```

```
        else
                printf("\nWife says - 'Husband , I have returned home'\n");


        sleep(1);

        check_spouse_status((i+1)%2);

  pthread_mutex_unlock(&mutex);      //V(mutex)

}


//check_spouse_status Function
void check_spouse_status(int i)
{
  if(spouse_state[(i+1)%2]==in)
  {
        spouse_state[i]=out;       //Spouse 'i' went out

        if(i==0)
          printf("\nHusband says - 'Wife , I am going out!!'\n");

        else
          printf("\nWife says - 'Husband , I am going out!!'\n");


        sem_post(&spouse[i]);     //V(spouse[i])

  }

  sleep(1);

}


//leave_house Function
void leave_house(int i)
{
```

```
  pthread_mutex_lock(&mutex);          //P(mutex)

      check_spouse_status(i);

  pthread_mutex_unlock(&mutex);     //V(mutex)

  sem_wait(&spouse[i]);     //P(spouse[i])

}

//takecare_parent Function
void takecare_parent(int i)
{
  if(i==0)
        printf("\nHusband says - ' I will take care of our parents '\n");

  else
        printf("\nWife says - 'I will take care of our parents '\n");

}
```

OUTPUT : -





---

3. In a railway ticket booking office, maximum 10 persons, either male, female, or both are allowed to go inside. There are three ticket counters in the booking office. Among these 10 persons, a maximum of 3 persons are allowed to book the ticket at a time with a restriction that all these 3 persons can neither be male nor be female. It means that maximum of 2 males with 1 female or maximum of 2 females with 1 male is allowed to book the ticket. Write a solution using semaphore to synchronize among the males and females to book their ticket.

CODE : -

/*------In a railway ticket booking office, maximum 10 persons, either male, female, or both are allowed to go inside. There are three ticket counters in the booking office. Among these 10 persons, a maximum of 3 persons are allowed to book the ticket at a time with a restriction that all these 3 persons can neither be male nor be female. It means that maximum of 2 males with 1 female or maximum of 2 females with 1 male is allowed to book the ticket. Write a solution using semaphore to synchronize among the males and females to book their ticket.---------*/

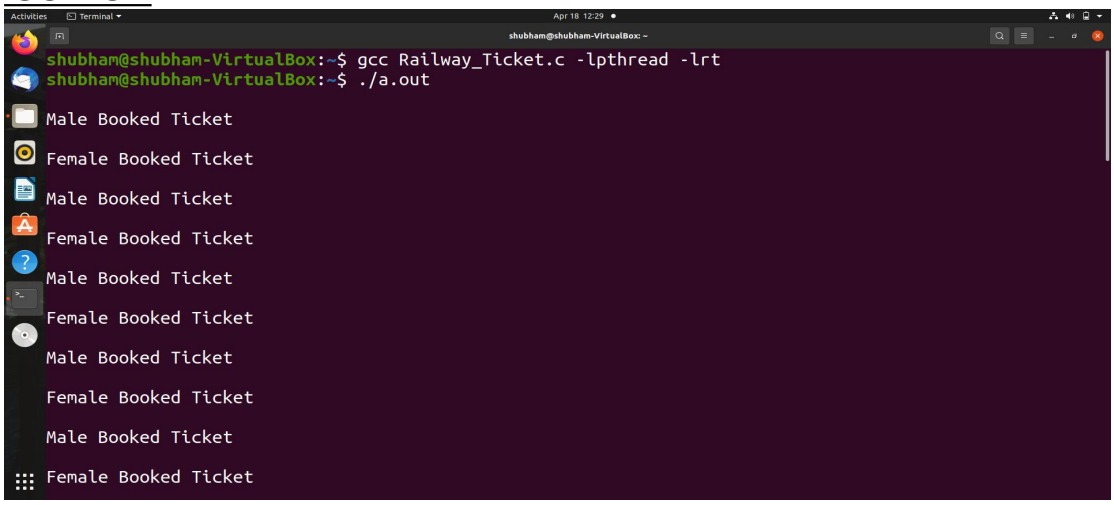
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>
#include<pthread.h>
#include<semaphore.h>


//Semaphore
sem_t person_cnt , female_cnt , male_cnt , counter;


//Global Variable
int count = 0;


//USER-DEFINED FUNCTIONS
void *Book_Ticket(void *args);
void Male(void);
void Female(void);
void Male_book_ticket(void);
void Female_book_ticket(void);


//Main
int main()
{
 int i;

 pthread_t Thread[2];

 //Initializing semaphore
 sem_init(&person_cnt,0,10);
 sem_init(&female_cnt,0,2);
 sem_init(&male_cnt,0,2);
 sem_init(&counter,0,3);
```

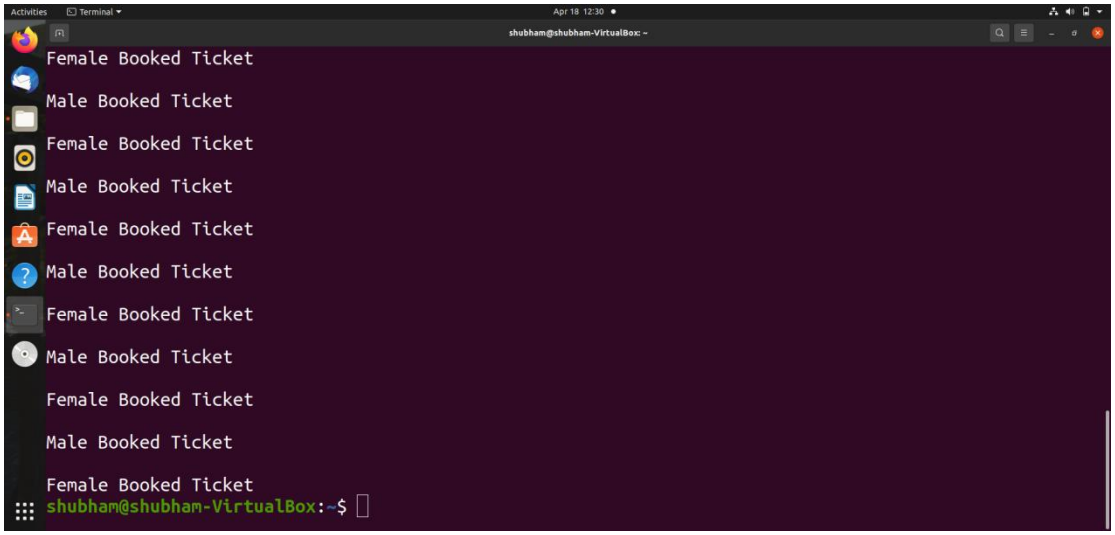
```
//Creating Threads
for(i=0;i<2;i++)
{
 if(i==0)
 {
  if(pthread_create(&Thread[i],NULL,&Book_Ticket,NULL)!=0)
  {
    perror("Failed to create thread\n");
  }

 }
 else
 {

  if(pthread_create(&Thread[i],NULL,&Book_Ticket,NULL)!=0)
  {
    perror("Failed to create thread\n");
  }

 }

}

//Joining the threads
for(i=0;i<2;i++)
{
 if(pthread_join(Thread[i],NULL)!=0)
 {
   perror("Failed to join the threads\n");
 }

}


//Destroying the semaphore
sem_destroy(&person_cnt);
sem_destroy(&female_cnt);
sem_destroy(&male_cnt);
sem_destroy(&counter);
```

```
 return 0;
}


void *Book_Ticket(void *args)
{

 //20 number of people are arriving to book ticket but 10 persons are
//allowed at a time
  for(;count<20;count++)
 {
   Male();      //Male Ticket Book Function

   Female(); //Female Ticket Book Function
 }

}

//Male Ticket Book Function
void Male(void)
{
 sem_wait(&person_cnt);           //p(person_cnt)

    sem_wait(&male_cnt);           //p(male_cnt)

        sem_wait(&counter);         //p(counter)

            Male_book_ticket();

        sem_post(&person_cnt);   //v(person_cnt)

     sem_post(&male_cnt);         //v(male_cnt)

 sem_post(&counter);     //v(counter)

}

//Female Ticket Book Function
void Female(void)
```

```
{
 sem_wait(&person_cnt);                    //p(person_cnt)

   sem_wait(&female_cnt);       //p(female_cnt)

     sem_wait(&counter);      //p(counter)

       Female_book_ticket();

     sem_post(&person_cnt);     //v(person_cnt)

   sem_post(&female_cnt);       //v(female_cnt)

 sem_post(&counter);     //v(counter)

}

//Male_book_ticket Function
void Male_book_ticket()
{
 printf("\nMale Booked Ticket\n");
}

//Female_book_ticket Function
void Female_book_ticket()
{
 printf("\nFemale Booked Ticket\n");
}
```

OUTPUT : -




Activities
Terminal
Apr 18 12:30
shubham@shubham-VirtualBox: ~
Female Booked Ticket
Male Booked Ticket
Female Booked Ticket
Male Booked Ticket
Female Booked Ticket
Male Booked Ticket
Female Booked Ticket
Male Booked Ticket
Female Booked Ticket
Male Booked Ticket
Female Booked Ticket
shubham@shubham-VirtualBox:~$