**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)

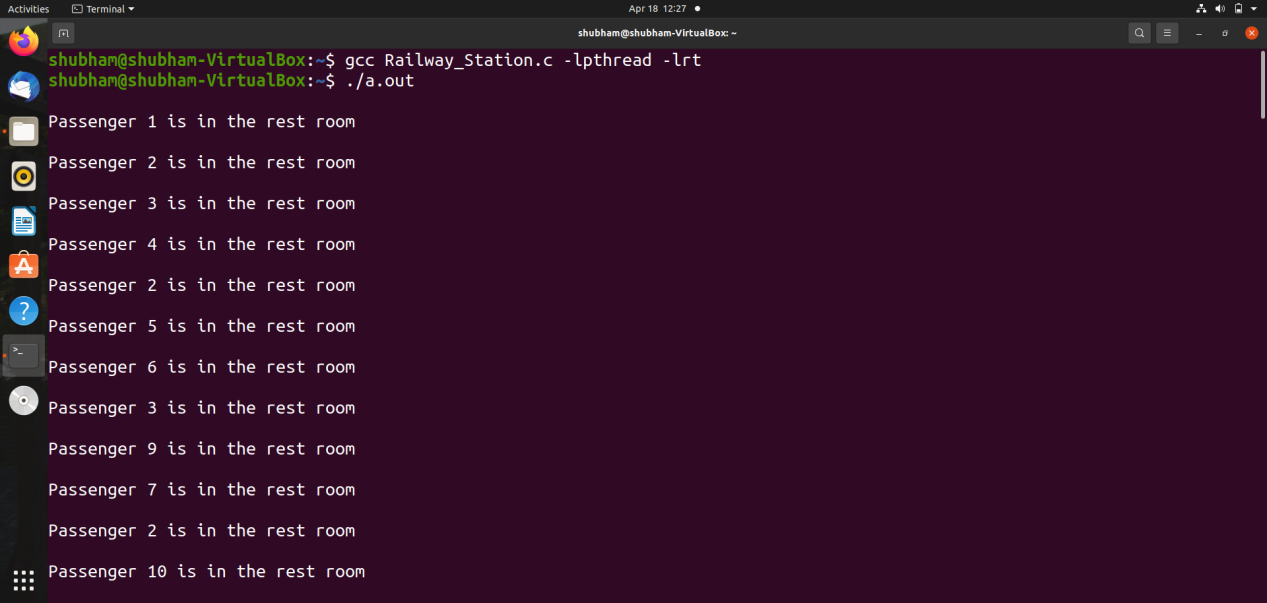
{

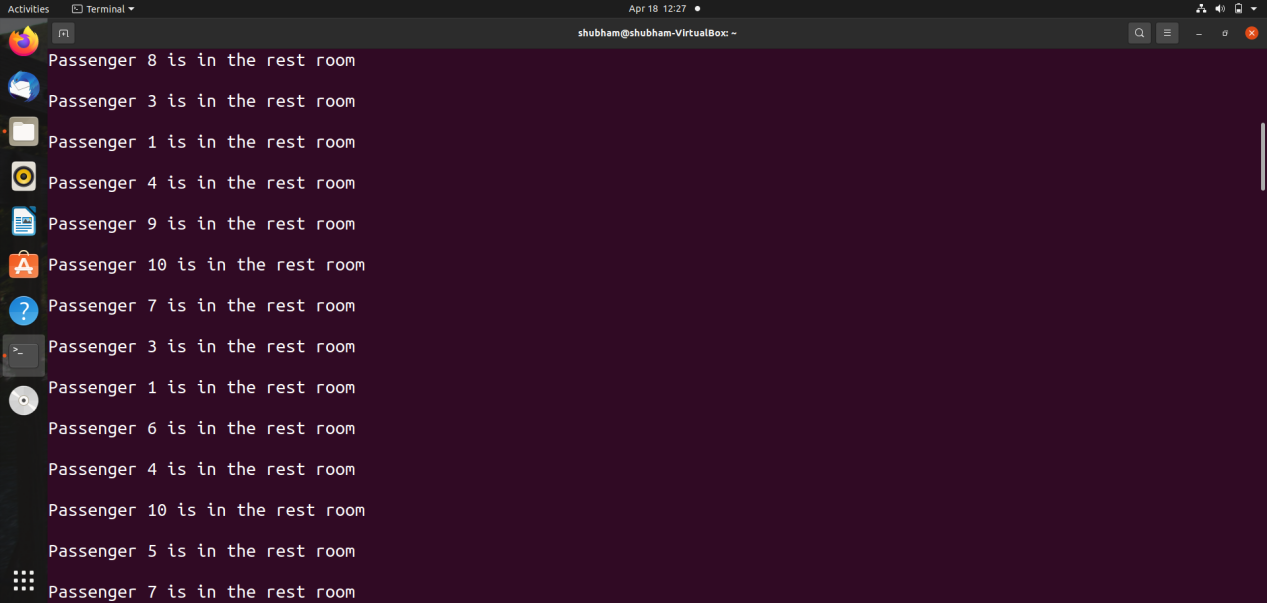
printf("\nPassenger %d is in the rest room\n",(i+1));

sleep(1);

}

**OUTPUT** : -





1. 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)

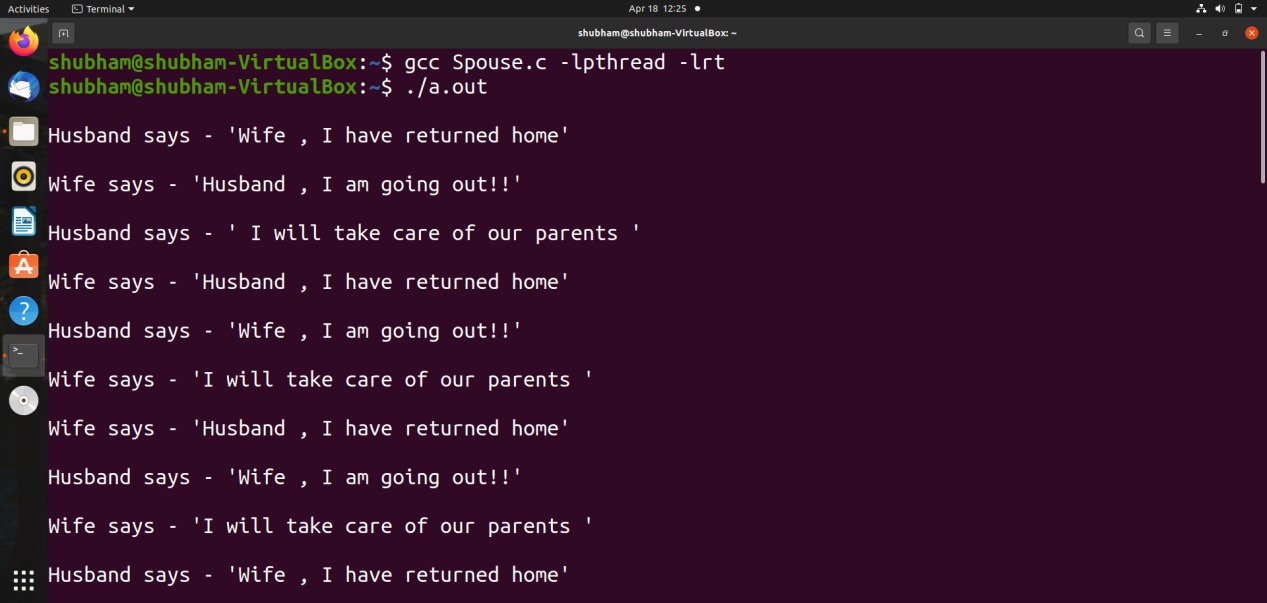
printf("\nHusband says - ' I will take care of our parents '\n");

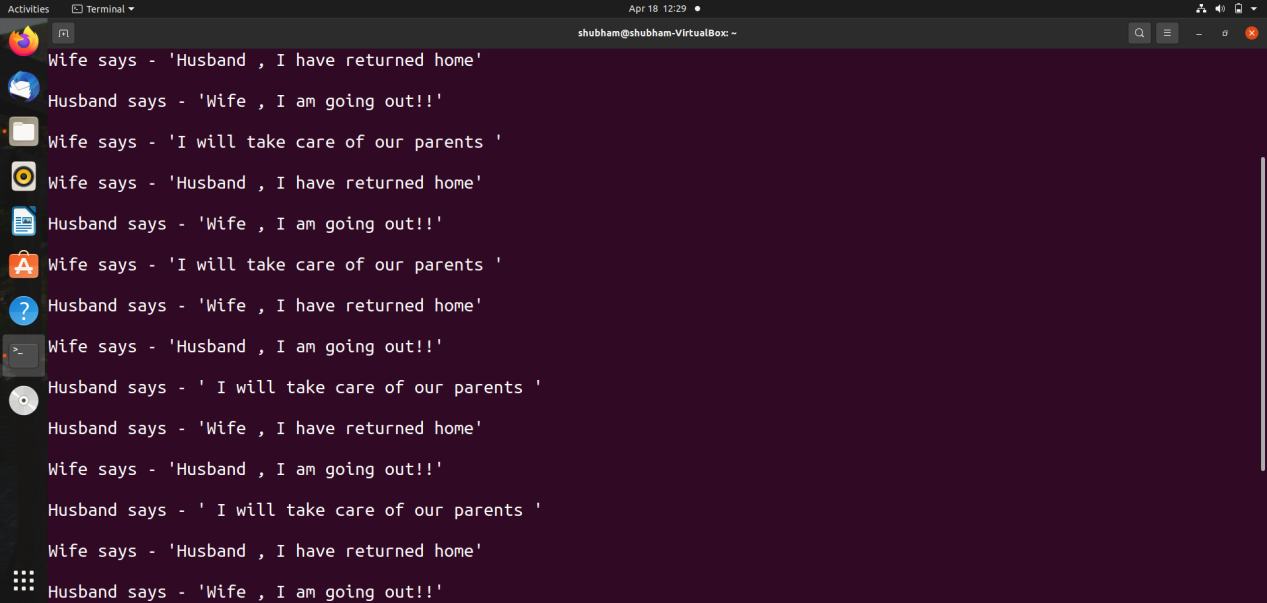
else

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

}

**OUTPUT** : -





1. 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 :** -

