***PROJECT REPORT***

***Implementing Airline Reservation system using reader writers problem***

***Muhammad Talha 21K-3288***

***Zarah Hassan 20K-1889***

***Sanjna Karera 21K-3409***

**Project Objective:**

The reader-writer problem is a famous problem in which there is content, a file, or any shared data between processes. The readers will read the data from the content and the writer will write the content.

In our scenario, the book and cancel functions are the writers and the reader function displays all the customer data.

Therefore, the objective of the problem is to avoid the race condition and simultaneously occurrences of the readers and writers.

**Project Description:**

In the reader-writer, the problem occurs when the reader starts to read the content of and at the same time the writer starts to write some content, so it will create problems in the content.

This problem is used to manage the synchronization of the readers and writer’s process so that the same phenomenon does not happen and in this scenario, we cannot book or cancel at the same time. Similarly, when a reader starts to read the content so no writer would be able to write the content, however the other reader could read the content.

Therefore, the cases become:

W - W: False

W - R: False

R - W: False

R - R: True

Flight data is stored in a txt file with customer details and read accordingly when required. The desired data is then pushed/stored using queue data structure where one queue handles the non-priority customers where as the other queue handles the priority customers. The data is then merged and displayed using a reader function.

Synchronization between threads is maintained using semaphores/locks and different threads for book,cancel and reader is called as per requirement.

Two system call is made for wait and signal semaphores in which the value of semaphores is passed as argument and the value is incremented/decremented accordingly.

**Conclusion:**

Hence, by using the semaphores synchronization is successfully done between the reader and writer threads so that the readers and writers don't process simultaneously and the shared data is now free from race conditions. Hence, all the requirements have been met as asked and the program is running as it was supposed to.

**Project code:**

#include<iostream>

#include<stdio.h>

#include<pthread.h>

#include<fstream>

#include<unistd.h>

#include<time.h>

#include<semaphore.h>

#include<stdlib.h>

#include<string.h>

#include <queue>

#include<linux/kernel.h>

#include<sys/syscall.h>

using namespace std;

int read\_mutex;

int write\_mutex;

int resource\_mutex;

int read\_count=0;

int avail\_seats=10;

struct FlightData {

string name;

string flight\_no;

string destinations;

string special\_needs;

int flight\_count;

string passport\_no;

};

struct ReservationData {

queue<FlightData> request;

}priority,temp2,normal;

void ReadFromFile(FlightData d[5]) {

int i=0;

ifstream inputfile("flight\_data.txt");

if (!inputfile) {

cout<<"Error opening file!"<<endl;

}

else {

//file.ignore(5,'\n');

while(1) {

inputfile>> d[i].name >> d[i].passport\_no >> d[i].flight\_no >>d[i].destinations>> d[i].special\_needs >> d[i].flight\_count;

i++;

if(inputfile.eof())

break;

}

}

inputfile.close();

}

class RWLocks {

public:

void lock\_read() {

syscall(336,read\_mutex);

read\_count++;

if (read\_count == 1) {

syscall(336,resource\_mutex);

}

syscall(337,read\_mutex);

}

void unlock\_read() {

syscall(336,read\_mutex);

read\_count--;

if (read\_count == 0) {

syscall(337,resource\_mutex);

}

syscall(337,read\_mutex);

}

void lock\_write() {

syscall(336,write\_mutex);

syscall(336,resource\_mutex);

}

void unlock\_write() {

syscall(337,resource\_mutex);

syscall(337,write\_mutex);

}

};

void\*bookSeat(void \*arg) {

int id=\*((int\*)arg);

FlightData d[5];

RWLocks lockobj;

int flag=1;

int i=0;

int count=0;

lockobj.lock\_write();

ReadFromFile(d);

cout<<"\nwriter id "<<id<<endl;

while(i<4) {

/\*if (avail\_seats <0 ){

cout<<"sorry all seats are reserved";

break;

}\*/

if(avail\_seats > 0 && (d[i].special\_needs=="true") || d[i].flight\_count>=5) {

// cout<<"\nEnter your desired destinations for "<<d[i].name<<endl;

// while(count <5 && flag !=0){

// cin>>d[i].destinations[count];

// count++;

// cout<<"Press 1 to add more and press 0 to stop\n";

// cin>>flag;

// }

priority.request.push(d[i]);

avail\_seats--;

cout<<"Reserved seat for " <<d[i].name<<" on flight no "<<d[i].flight\_no<<" to "<<d[i].destinations<<endl;

cout<<"Available seats are "<<endl<<avail\_seats<<endl;

//break;

}

else{

count=0;

//cout<<"\nNot a priority client will reserve later!"<<endl;

// cout<<"\nEnter your desired destinations for "<<d[i].name<<endl;

// while(count <5 && flag !=0){

// cin>>d[i].destinations[count];

// count++;

// cout<<"Press 1 to add more and press 0 to stop\n";

// cin>>flag;

// }

normal.request.push(d[i]);

}

flag=1;

i++;

}

while(!normal.request.empty()){

priority.request.push(normal.request.front());

cout<<"Reserved seat for " <<normal.request.front().name<<" on flight no "<<normal.request.front().flight\_no<<"to"<<normal.request.front().destinations<<endl;

avail\_seats--;

cout<<"Available seats are "<<endl<<avail\_seats<<endl;

normal.request.pop();

}

lockobj.unlock\_write();

}

/\*void booknorm(){

while(!normal.request.empty()){

cout<<"Reserved seat for " <<normal.request.front().name<<" on flight no "<<normal.request.front().flight\_no;

avail\_seats--;

cout<<"Available seats are "<<endl<<avail\_seats<<endl;

priority.request.push(normal.request.front());

normal.request.pop();

}

}\*/

void\*cancel\_seat(void\*arg){

string pas=\*((string\*)arg);

RWLocks lockobj;

lockobj.lock\_write();

int i=0;

int flag=0;

while(!priority.request.empty()){

if(pas==priority.request.front().passport\_no){

cout<<"Cancelled seat for "<<priority.request.front().name<<" on flight number "<<priority.request.front().flight\_no<<endl;

priority.request.pop();

avail\_seats++;

cout<<"Available seats are "<<endl<<avail\_seats<<endl;

flag=1;

break;

}

else{

temp2.request.push(priority.request.front());

priority.request.pop();

continue;

}

}

while(!temp2.request.empty()){

priority.request.push(temp2.request.front());

temp2.request.pop();

}

lockobj.unlock\_write();

if(flag==0){

cout<<"No passenger found!";

}

}

void \*reader(void\*arg) {

int id = \*((int\*)arg);

FlightData d[5];

RWLocks lockobj;

lockobj.lock\_read();

ReadFromFile(d);

printf("\nreader id:%d\nreading flight data:\n",id);

cout<<d[id].name<<" "<<d[id].passport\_no<<" "<<d[id].flight\_no<<" "<<d[id].destinations<<" "<<d[id].special\_needs<<endl;

cout<<"Read count is "<<read\_count<<endl;

read\_count++;

lockobj.unlock\_read();

}

int main(){

//sem\_init(&read\_mutex,0,1);

//sem\_init(&write\_mutex,0,1);

//sem\_init(&resource\_mutex,0,1);

read\_mutex=1,write\_mutex=1,resource\_mutex=1;

pthread\_t readers[5],reserver[2], cancelc;

int readerid[5]={0,1,2,3,4};

int reserverid[2]={0,1};

string pas;

for(int i=0;i<2;i++){

pthread\_create(&reserver[i],NULL,bookSeat,&reserverid[i]);

}

for(int i=0;i<4;i++){

pthread\_create(&readers[i],NULL,reader,&readerid[i]);

}

for(int i=0;i<2;i++){

pthread\_join(reserver[i],NULL);

}

for(int i=0;i<4;i++){

pthread\_join(readers[i],NULL);

}

cout<<"\nenter your passport number to cancel your reservation:";

cin>>pas;

pthread\_create(&cancelc,NULL,cancel\_seat,&pas);

pthread\_join(cancelc,NULL);

//booknorm();

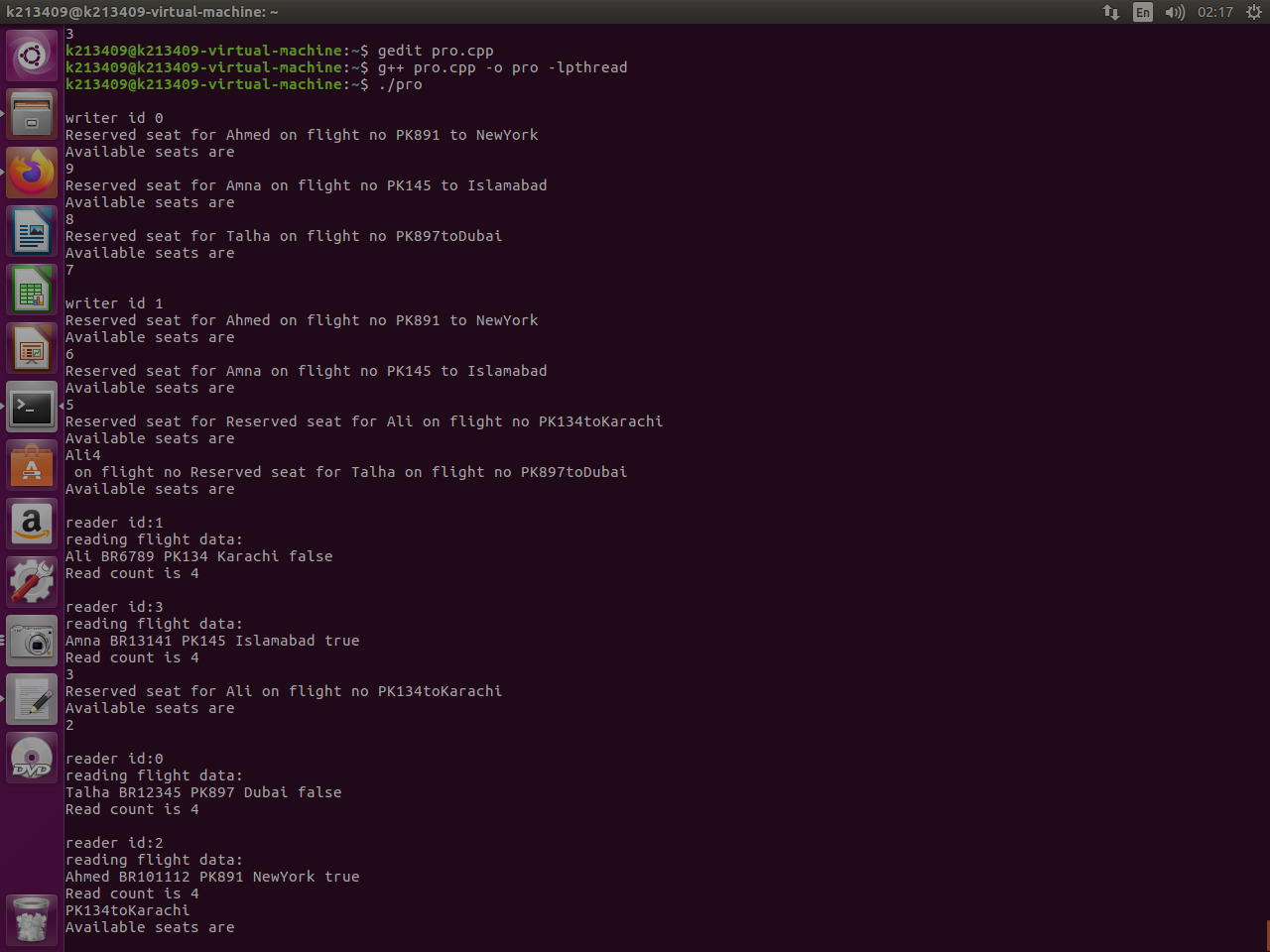
//sem\_destroy(&read\_mutex);

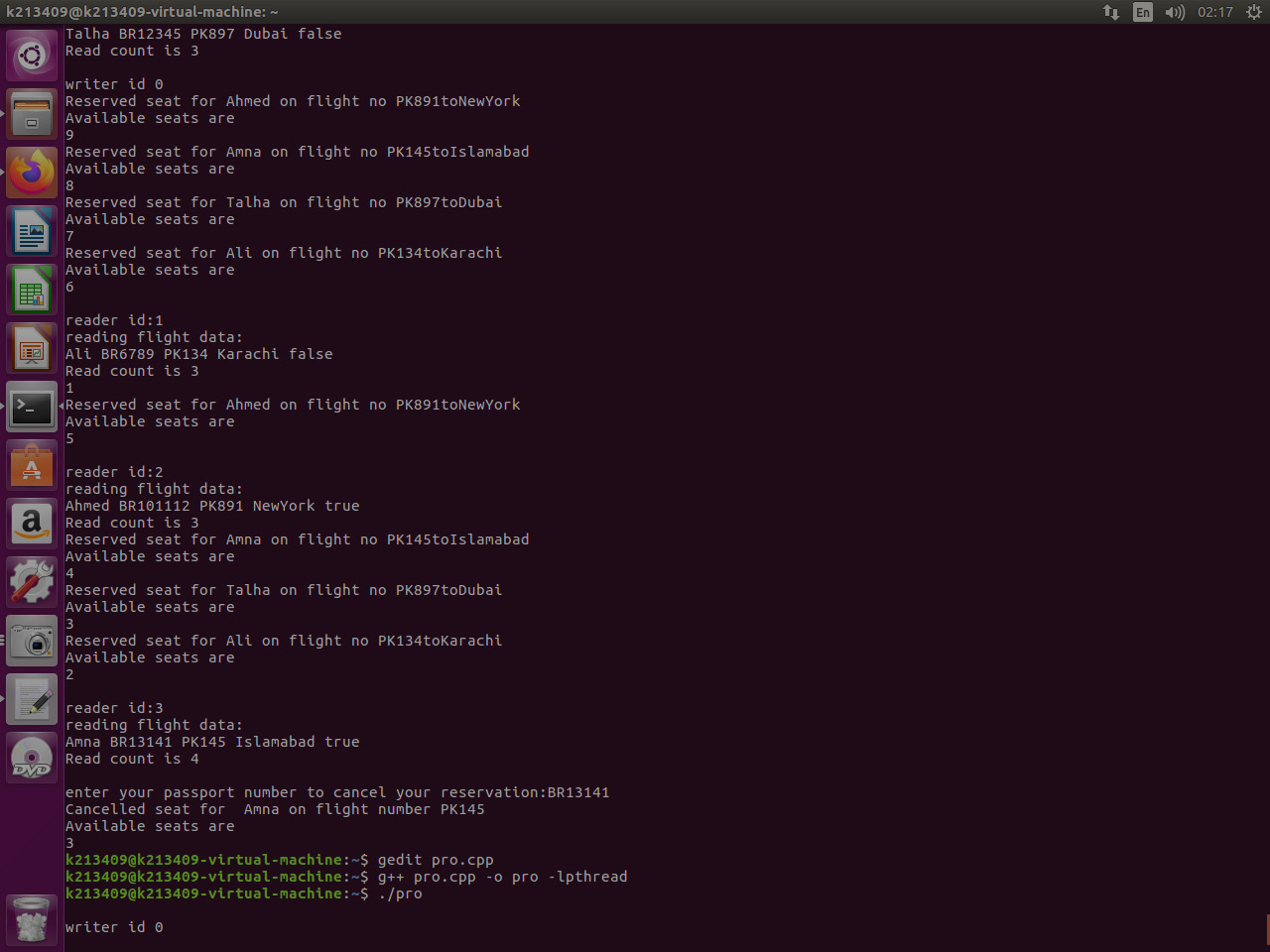
//sem\_destroy(&write\_mutex);

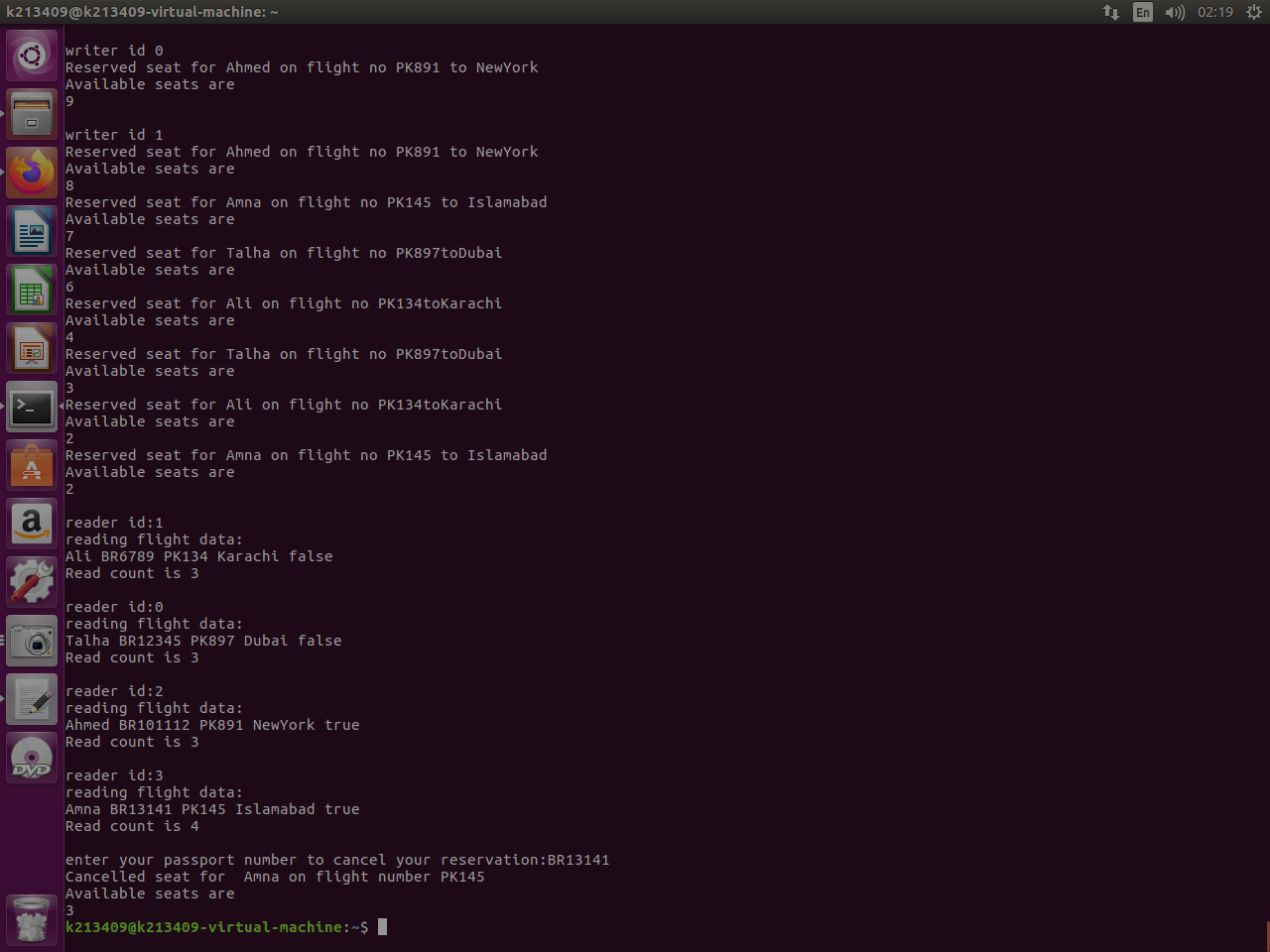
//sem\_destroy(&resource\_mutex);

}

**Screenshots:**

****

****

****