# Министерство образования и науки РФ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Омский государственный технический университет»

|  |  |
| --- | --- |
| Факультет (институт) | Факультет информационных технологий и компьютерных систем |
|  |  |
| Кафедра | Информатики и вычислительной техники |
|  |  |

КУРСОВОЙ ПРОЕКТ (РАБОТА)

|  |  |
| --- | --- |
| по дисциплине | Операционные системы |
|  |  |
| на тему | Многопоточная графическая Linux модель имитации рабочих поездок пассажиров по круговому чартерному международному авиамаршруту между четырьмя столицами |

Пояснительная записка

|  |  |
| --- | --- |
| Шифр проекта (работы) | КП-02068999-20-97 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Студента (ки) | | | | Соколков Глеб Сергеевич | | | | | | |
|  |  |  |  | | | | фамилия, имя, отчество полностью | | | | | | |
|  |  |  | Курс | 2 |  | | | Группа | | ИВТ-172 | | |  |
|  |  |  |  |  | |  | | |  |  | | |  |
|  |  |  | Направление (специальность) | | | | | | |  | | | |
|  |  |  | ***09.03.01 (правильный номер)*** | | | | | | | ***Информатика и вычислительная техника*** | | | |
|  |  |  | код, наименование | | | | | | | | | | |
|  |  |  | Руководитель | | | | доцент, к.т.н | | | | | | |
|  |  |  |  | | | | ученая степень, звание | | | | | | |
|  |  |  | Флоренсов Александр Николаевич | | | | | | | | | | |
|  |  |  | фамилия, инициалы | | | | | | | | | | |
|  |  |  | Выполнил (а) | | | |  | | | | | | |
|  |  |  |  | | | | дата, подпись студента (ки) | | | | | | |
|  |  |  | К защите | | | |  | | | | | | |
|  |  |  |  | | | | дата, подпись руководителя | | | | | | |
|  |  |  | Выполнение и подготовка к защите КП (КР) | | | | | Защита КП (КР) | | | | Итоговый рейтинг | | |
|  |  |  |  | | | | |  | | | |  | | |
|  |  |  | Проект (работа) защищен (а) с оценкой | | | | | | | |  | | |

Омск 2018

Министерство образования и науки РФ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Омский государственный технический университет»

**ОТЗЫВ**

**на курсовой проект (работу)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Факультет (институт) | | | | | | | Факультет информационных технологий и компьютерных систем | | | | | | | | | |
|  | | | | | | |  | | | | | | | | | |
| Кафедра | | | Информатики и вычислительной техники | | | | | | | | | | | | | |
|  | | |  | | | | | | | | | | | | | |
| Дисциплина | | | | Операционные системы | | | | | | | | | | | | |
|  | | | |  | | | | | | | | | | | | |
| Тема | Многопоточная графическая Linux модель имитации рабочих поездок пассажиров по круговому чартерному международному авиамаршруту между четырьмя столицами. | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | |
| Студент (ка) | | | | Соколков Глеб Сергеевич | | | | | | | | | | | | |
|  | | | | фамилия, имя, отчество полностью | | | | | | | | | | | | |
| Курс | 2 | | | | Группа | | | ИВТ-171 |  | | | | | | | |
|  | |  | | |  | | |  |  | | | | | | | |
| Руководитель | | | | | Флоренсов Александр Николаевич | | | | | | | | | | | |
|  | | | | | ученая степень, звание, ФИО | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| **Содержание отзыва** | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| **Рейтинговые баллы за выполнение и подготовку к защите курсового проекта (работы)** | | | | | | | | | | | | |  | | | | |
| **Заключение о допуске к защите** | | | | | | | | | |  | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| **Руководитель** | | | | | |  | | | | | Дата |  | | 20 |  | г. |

Реферат

Пояснительная записка по курсовому проекту 25 с., 4 ч., 7 рис., 3 источ, 1 прил.

C, LINUX, GCC, XLIB, МНОГОПОТОЧНОЕ ПРИЛОЖЕНИЕ.

Объектом исследования является взаимодействия множества потоков в операционной системе Linux в графическом окне.

Цель работы – разработка многопоточной программной модели рабочих перевозок пассажиров по круговому чартерному авиамаршруту между четырьмя столицами.

В ходе работы был разработан алгоритм взаимодействия множества потоков при работе с графическим окном в операционной системе Linux.

В результате была разработана программа, которая демонстрирует модель рабочих перевозок пассажиров по круговому авиамаршруту между четырьмя столицами.

Содержание

[Введение 4](#_Toc512456690)

[1 Введение в проблематику разработки многопоточных приложений 5](#_Toc512456691)

[2 Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения 6](#_Toc512456692)

[3 Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения](#_Toc512456693) 7

[4 Детальное текстовое описание на основе сочетания естественного языка и программных конструкций алгоритмов всех процедур](#_Toc512456694) 8

[Заключение](#_Toc512456695) 14

[Список использованных источников](#_Toc512456696) 15

[Приложение](#_Toc512456697) 16

Введение

Курсовой проект по дисциплине «Операционные системы», 2 курс. В проекте использовался язык программирования C.

Задача: Разработать для Linux многопоточную программу, которая имитирует рабочие поездки пассажиров по круговому чартерному междунардному авиамаршруту Нью-Йорк – Лондон – Москва – Токио. На маршруте летает один самолёт, вмещающий до 5 пассажиров. При запуске программы, по запросу к пользователю, задаётся значение параметра М – суммарное число пассажиров, использующих указанные маршруты. Для каждого пассажира, который должен моделироваться отдельной нитью, случайным образом задаётся время пребывания в городе, после итечения которого он появляется в аэропорту и ожидает нужный ему самолет. Выбор направления чартерного маршрута осуществляется случайным образом в момент появления пассажиров в аэропорту. Первоначально все М пассажиров появляются в Москве. Поведение каждого пассажира должно имитироваться с помощью отдельной нити. Отображение динамики модели должно осуществляться в графическом окне с помощью условных схем, значков и текстов, размер окна не менее 600 на 800 пикселов. Изображение городов и дорого должны быть представлены с помощью специально подобранных схем или символов. Рекомендуется разметить изображения городов в различных углах окна. Для правильного взаимодействия использовать семафоры и мьютексы.

Проект состоит из пяти разделов:

* Введение в проблематику разработки многопоточных приложений
* Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения
* Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения
* Детальное текстовое описание на основе сочетания естественного языка и программных конструкций алгоритмов всех процедур

1 Введение в проблематику разработки многопоточных приложений

При разработке многопоточных программ следует иметь в виду необходимость явного указания для системы разработки, что данная программа будет использовать более одной нити. Для таких приложений в процессе построения исполняемого файла подключаются специальные библиотеки подпрограмм.

При разработке многопоточных программ для Linux следует указывать соответствующую библиотеку поддержки. Такое указание может задаваться в одной из двух основных форм. Первая из них явно задает библиотеку и имеет вид:

gcc prog.c /usr/lib/libpthread

а вторая задает эту же библиотеку неявно и записывается в виде:

gcc prog.c -lpthread

В современных операционных системах широко используются нити (thread), называемые несколько неточно в русском переводе также потоками. Обычно нить своей работой реализует действия одной из процедур программы. Теоретически любой нити процесса доступны все части программы процесса, в частности, все его процедуры, но реально работа организуется так, чтобы нити отвечала отдельная процедура. Учитывая, что процедуре для нормальной работы необходимы локальные переменные, становится понятным закрепление области этих переменных за нитью. Объект хранения локальных переменных (вместе со служебной информацией при вызове подпрограмм) называют стеком. (Более точное понятие стека программы формируется только с помощью архитектуры процессора.) Этот стек в действительности является частью оперативной памяти, он используется не только программно, но и аппаратно, в частности, при реализации прерываний. Стек процедуры является неотъемлемой частью ресурсов, принадлежащих процедуре. (Более точным термином является кадр стека для процедуры или фрейм стека)

В операционной системе Unix многопоточное программирование появилось достаточно поздно. К настоящему времени эта возможность входит в стандарт POSIX для Unix и поддерживается во всех современных ОС. Использование нитей при этом требует подключения заголовочного файла pthread.h.

Важной задачей в многопоточном программировании является правильная реализации потоков. Если в программе требуется определённая последовательность работы потоков, то в таком случае следует использовать

семафоры или мьютексы, потому что последовательность работы потоков зависит от скорости выполнения других процессов, соответственно, если второй поток выполнится быстрее первого, то он и быстрее обработает данные, что собственно не должно произойти, поэтому не стоит забывать о правильной работе семафоров и мьютексов, если таковое имеется в программе. Потому что бывают такие ситуации, когда один поток заблокировал другой поток и после выполнения должен разблокировать, а этого не происходит, потому что программист просто не реализовал освобождение заблокированного потока.

2 Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения

Основным элементом функционирования являются графические примитивы, которые собственно и позволяют пользователю отслеживать действия объектов модели. Все происходящие действия объектов происходят в графическом окне 1300x800 пикселов. Графические примитивы рисуются, путём получения координат объекта по x и y. Объект имеет состояния, которые присутствуют в структуре данных этого объекта, к примеру флаги состояний обьекта при изменении которых он будет по разному реагировать на различные ситуации.

При помощи процедур изменения координат объекта, его состояний, а также очистки графического окна, рисуются графические примитивы. Сначала очищается окно, после же рисуются все графические примитивы и окно вновь очищается, что собственно представляет собой покадровое рисование объектов модели.

В процедуре main реализовано выделение памяти для потоков и создание самих потоков, с передачей в них соответсвующего элемента массива структур, описывающие логику соответствующих потоков. Также в main происходит создание графического окна, а в конце main вызывается бесконечный цикл для рисования графических примитивов, которые определены в функции перерисовки.

Процедура calcPosOfPlane вычисляет текущею позицию для каждого самолёта в зависимости от его положения и направления.

Процедура SetDefaultProps устанавливает стандартные свойства для отдельной нити.

Процедура get\_rand\_range\_int выдает случайное целое число из диапазона.

Процедура MovePasses отвечает за передвижение пассажиров.

Процедура PassesLogic отвечает за всю логику взаимодействия нитей с городами и самолётами.

Процедура redraw отвечает за отрисовку объектов в графическом окне, как статических, так и динамических. В main процедуре реализован бесконечный цикл, который собственно и содержит вызов функции рисования.

3 Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения

В программе используются следующие глобальные переменные:

1. Display \*dspl – это внутренняя структура библиотеки Xlib, на которую ссылается программа пользователя при вызове функций, требующих обмена данных с X-сервером или функций, которым необходимы сведения о X-сервере. Поля этой структуры в документации не описываются и прямые обращения из программы к полям данных этой структуры запрещены.
2. int screen – номер экрана, используемый по умолчанию. Этот номер указывается при подключении к X-серверу.
3. Window hwnd – идентификатор окна.
4. GC gc – дескриптор контекста вывода графики. Дескриптор представляет из себя указатель на структуру данных, в которой хранятся специфичные для библиотеки Xlib данные и идентификатор графического контекста на X-сервере.
5. char m[], n[], t[], l[] – названия городов.
6. struct plane – структура данных самолёта, int: dirForCit – мировое направление города, dir – направление движения самолёта (по часовой или против часовой стрелки), x, y – позиция самолёта в окне программы, tx – направления самолета между некоторыми городами, passengers – кол-во пассажиров в самолёте, speed – скорость самолёта, waitTime – время ожидания в городах, waitFlag – флаг ожидания.
7. int plCount - кол-во самолётов.
8. struct planes[2] – массив структур самолётов.
9. int pasCount – количество всех созданных пассажиров.
10. int sizeH, sizeV – отступы для правильной прорисовки городов.
11. int vDistance, hDistance – дистанции между городами по вертикали и горизонали.
12. struct city – структура данных для городов.
13. struct cities[4] – массив структур городов.
14. pas – структура данных пассажиров.
15. struct passes[100] – массив структур всех пассажиров.
16. Pthread\_mutex\_t mtx – мутекс.

4 Детальное текстовое описание на основе сочетания естественного языка и программных конструкций алгоритмов всех процедур

Программа отрисовывает изменения данных из глобальных массивов струкур с некоторой задержкой. Задержка устанавливается при помощи функции usleep(), благодаря чему можно увидеть всю динамику перемещения пассажиров по городам.

Графическое сопровождение программы реализовано при помощи графического окна и графических примитивов, которые программа рисует в данном графическом окне. В программе происходит покадровое рисование графических примитивов при помощи бесконечного цикла while, который каждый последующий шаг рисует графические примитивы путём вызова фунции redraw(). В данном цикле обрабатываются данные пассажиров и самолётов, которые в последствии используются для отображения перемещения пассажира, отображение изменения позиции каждого пассажира, а также самолётов. Данный цикл расположен в функции main. В данном цикле также происходит отрисовка статичных обьектов, которые никак не двигаются.

В главной функции main создается графическое окно, задаётся количество пассажиров. Также создаются структуры городов, самолётов и пассажиров которые передаются в массив структур для каждой структуры соответственно. В конце main происходит бесконечный вызов функции redraw() перерисовывающей окно.

За поведение пассажиров отвечает функция PassesLogic, в которой обрабатываются все состояния для каждого отдельного пассажира и действия этих пассажиров на определённые события, к примеру действием пассажира на флаг isPlane будет перемещение пассажира к самолёту и если в самолёте больше 5 пассижров то, они возвращаются в аэропорт. Эта функция имеет бесконечный цикл while для того чтобы в реальном времени изменять поведение пассажиров на определённые события.

За поведение самолёта отвечает функция calcPosOfPlane. В данной функции реализовано перемещение самого самолёта между городами, а также приостановка в городах для загрузки или выгрузки пассажиров. При приостановке в городах самолёт посредством установки флага waitFlag = 1 говорит пассажирам, что можно заходить в самолёт, а если waitFlag = 0, то значит что самолёт покидает город и заходить в самолёт нельзя.

Схема алгоритма главной процедуры представлена на рисунке 1.

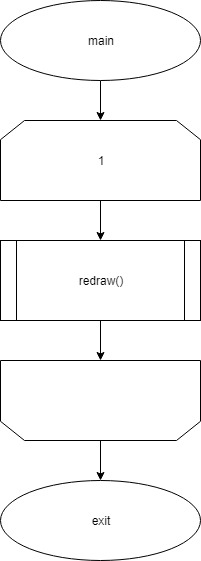


Рисунок 1 – схема алгоритма основной процедуры

Скриншот работы программы представлен на рисунках 2-6.

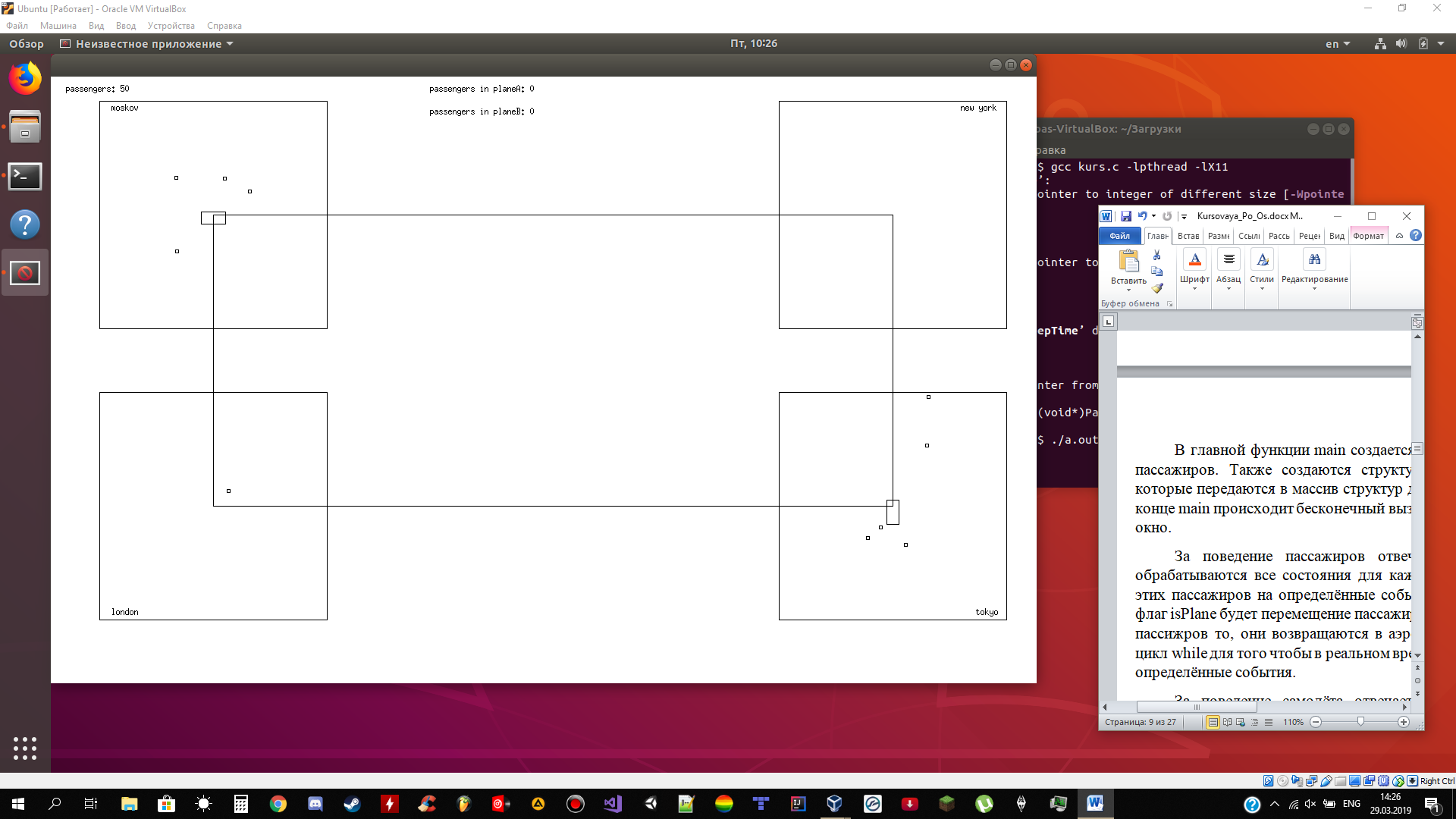


Рисунок 2 – пример работы программы

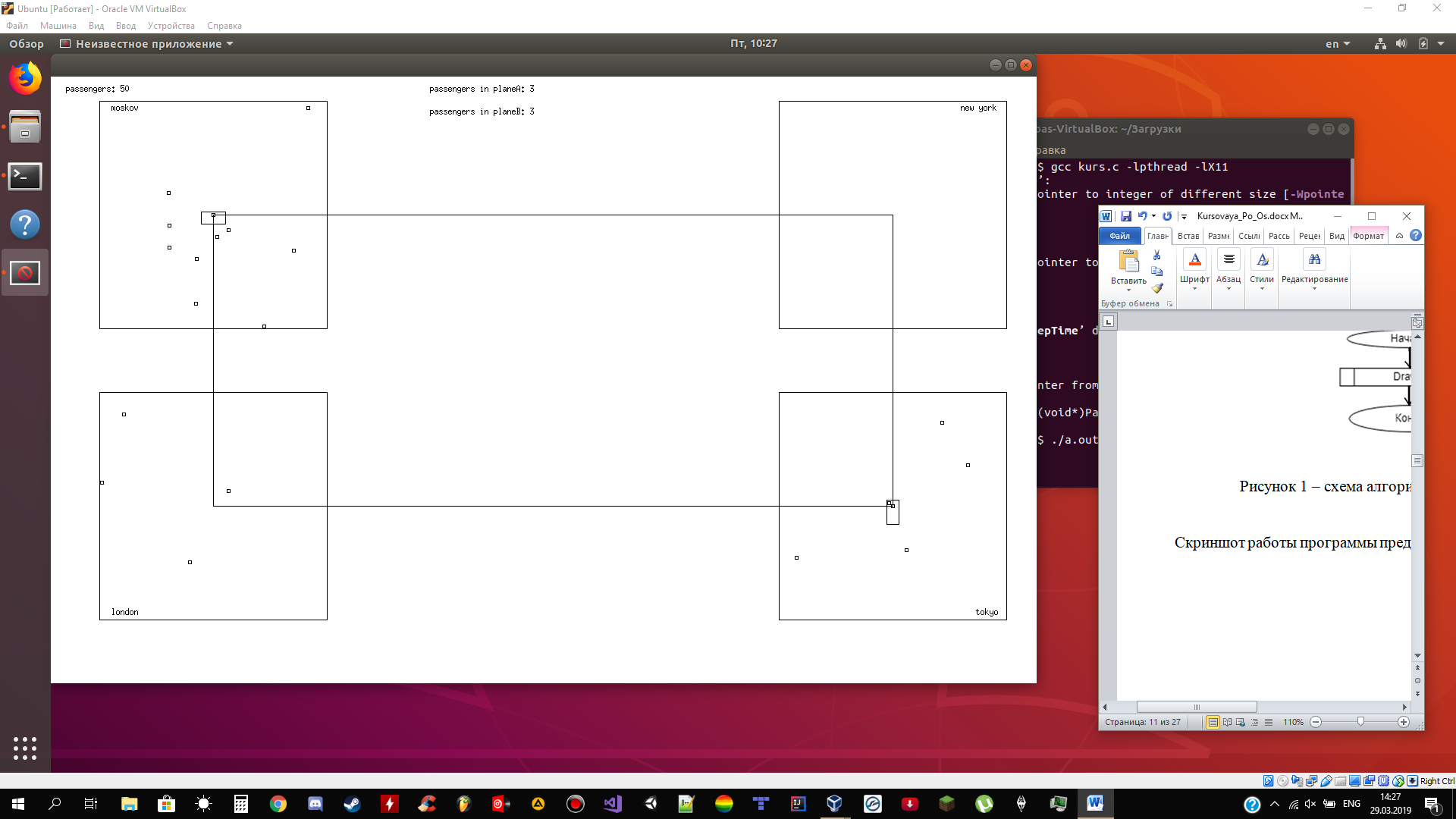


Рисунок 3 – пример работы программы

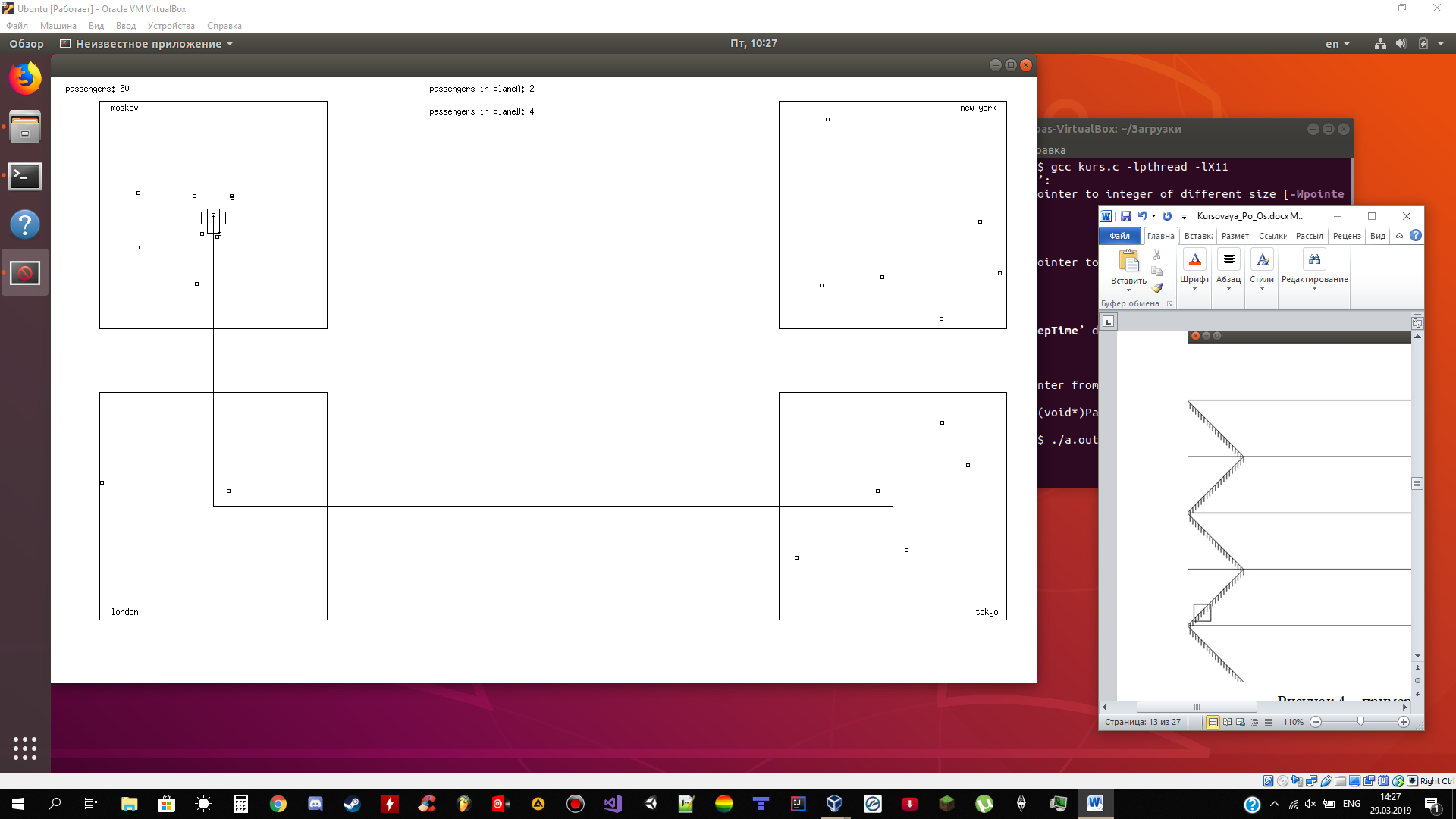


Рисунок 4 – пример работы программы

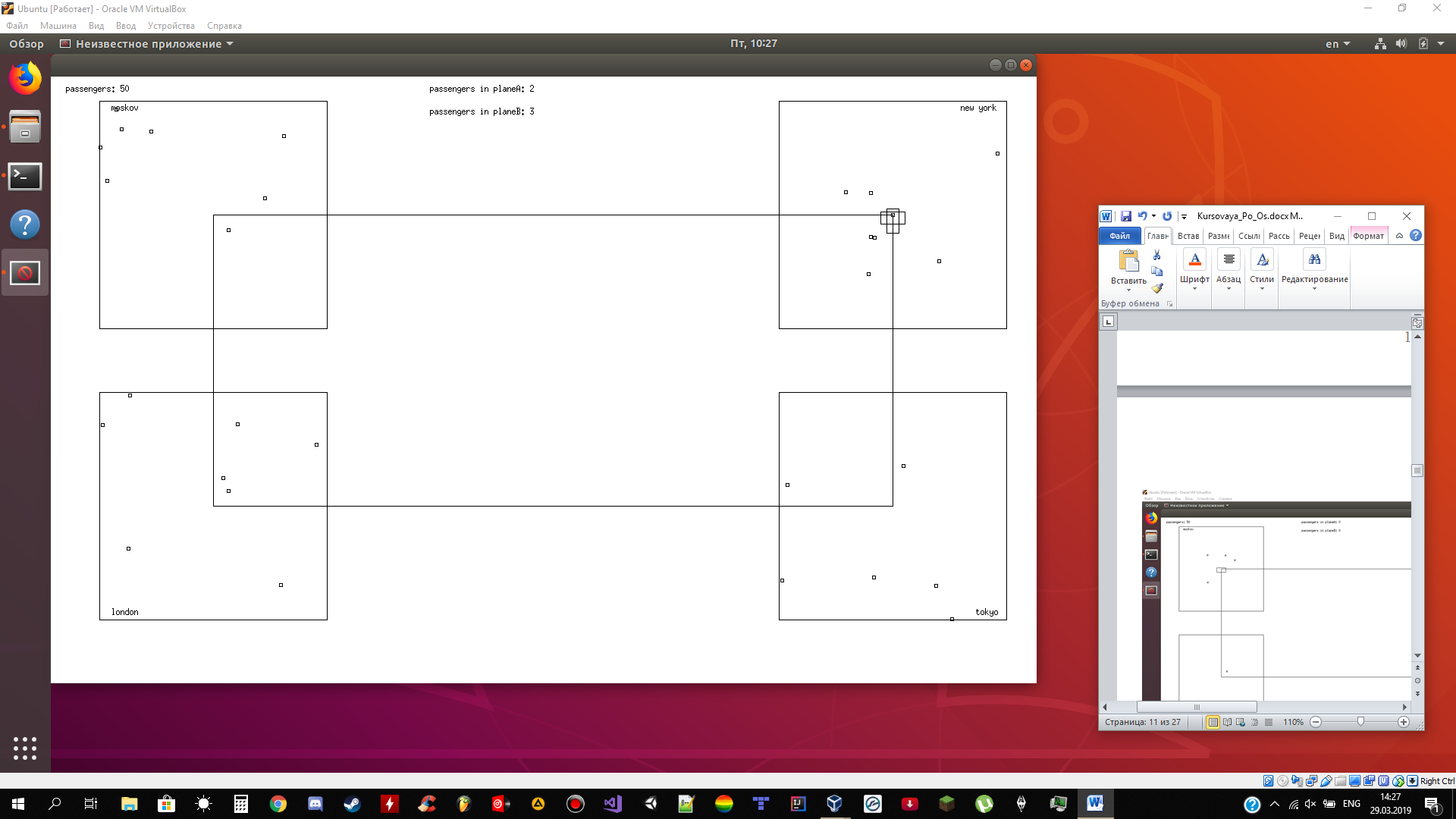


Рисунок 5 – пример работы программы

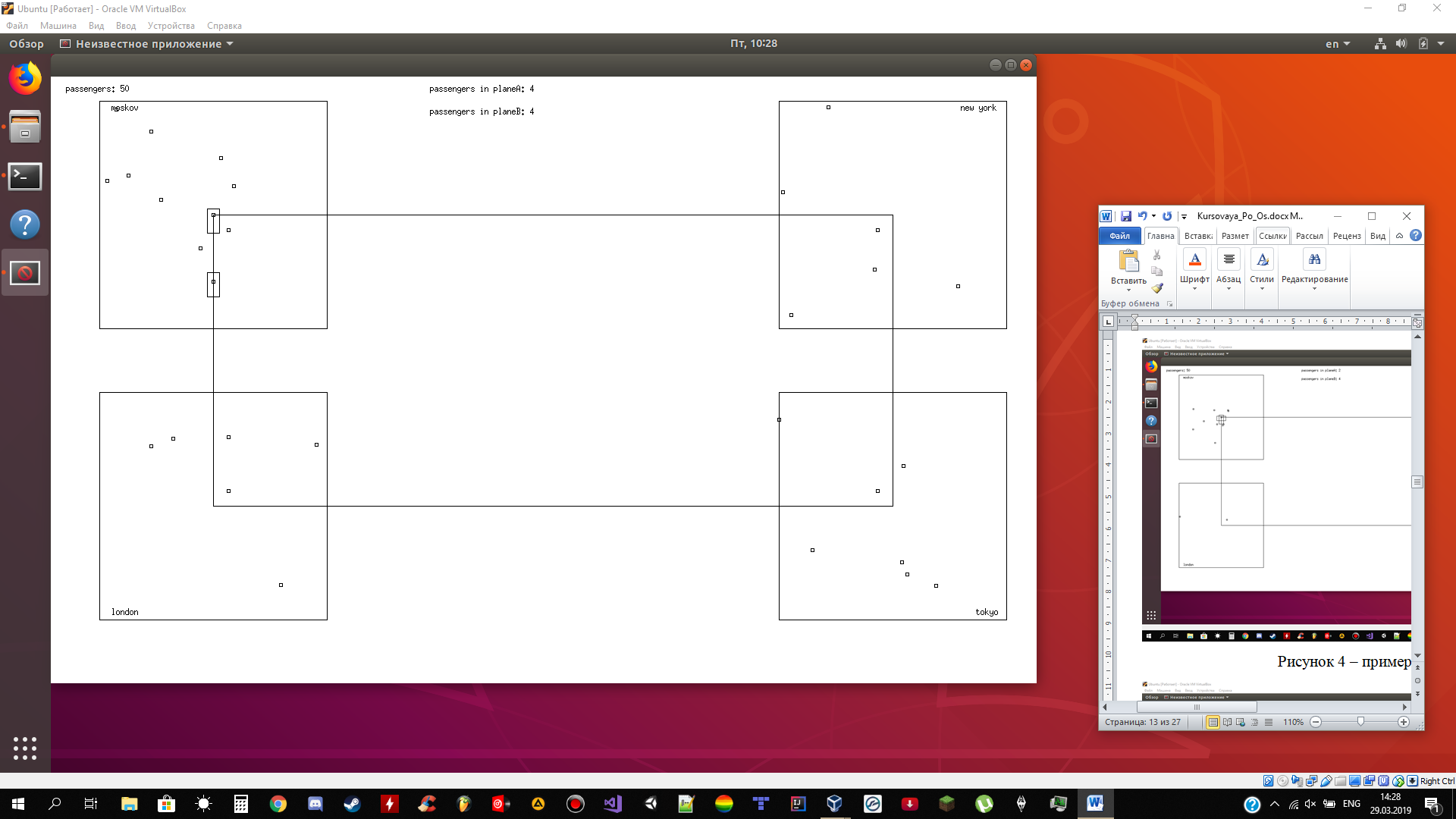


Рисунок 6 – пример работы программы

Скриншот файла для трансляции программы представлен на рисунке 7.

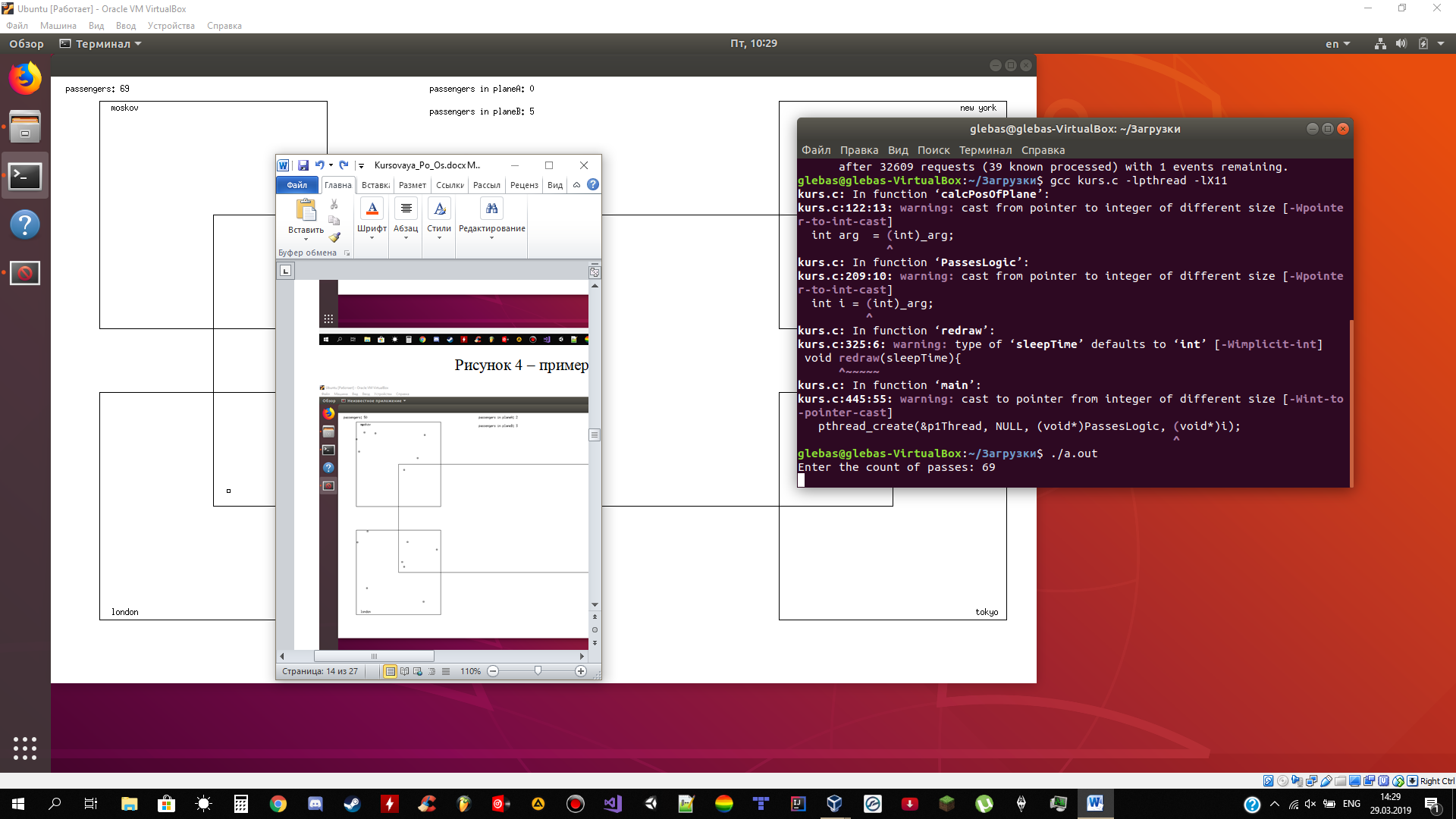


Рисунок 7 – cкриншот файла для трансляции программы

Заключение

В результате работы над проектом был реализован алгоритм взаимодействия нескольких потоков при работе с графическим окном в операционной системе Linux в программе, написанной на языке программирования C. Программа демонстрирует модели рабочих перевозок пассажиров по круговому чартерному авиамаршруту между четырьмя столицами.

Список использованных источников

1. Флоренсов, А.Н. Операционные системы для программиста. Омск. ОмГТУ, 2005.
2. Гордеев, А.В. Операционные системы / А.В. Гордеев. – 2-е изд. – СПб.: Питер, 2007
3. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

Приложение

**Исходный код программы**

#include <stdio.h>

#include <X11/Xlib.h>

#include <string.h>

#include "stdlib.h"

#include <unistd.h>

#include <pthread.h>

#include <math.h>

#include <time.h>

#define HEIGHT 1200

#define WIDTH 800

char m[] = "moskov";

char n[] = "new york";

char t[] = "tokyo";

char l[] = "london";

Display \*dspl;

int screen;

Window hwnd;

XEvent event;

GC gc;

pthread\_mutex\_t mtx;

struct plane{

int dirForCit;

int dir;

int x, y;

int tx, ty;

int passengers;

int speed;

int waitTime;

int waitFlag;

};

int plCount = 2;

struct plane planes[2];

int pascount = 0;

int sizeV = 32;

int sizeH = 64;

int vDistance = 384;

int hDistance = 896;

struct city{

int x, y;

int stopX, stopY;

};

struct city cities[4];

struct pas{

int x;

int y;

int tx;

int ty;

int sleep;

int maxSleepTime;

int sleepTime;

int wait;

int drive;

int isStop;

int isCity;

int isPlane;

int city;

int dir;

int canLeave;

};

struct pas passes[100];

int get\_rand\_range\_int(const int min, const int max) {

return rand() % (max - min + 1) + min;

}

void SetDefaultProps(int i){

passes[i].city = 3;

passes[i].x = cities[3].x - 150;

passes[i].y = cities[3].y - 150;

passes[i].dir = rand()%2;

passes[i].tx = cities[passes[i].city].stopX;

passes[i].ty = cities[passes[i].city].stopY;

passes[i].sleepTime = passes[i].maxSleepTime/2 + rand()%passes[i].maxSleepTime/2;

passes[i].sleepTime = passes[i].sleepTime \* 1000;

}

void MovePasses(int i){

if(passes[i].x < passes[i].tx) passes[i].x++;

if(passes[i].x > passes[i].tx) passes[i].x--;

if(passes[i].y < passes[i].ty) passes[i].y++;

if(passes[i].y > passes[i].ty) passes[i].y--;

}

/\*void GetCoords(int city, int c){

if(city == 0){

passes[c].x = 75 + get\_rand\_range\_int(10, 50);

passes[c].y = 75 + get\_rand\_range\_int(10, 50);

}

if(city == 1){

passes[c].x = 1000 - get\_rand\_range\_int(10, 50);

passes[c].y = 75 + get\_rand\_range\_int(10, 50);

}

if(city == 2){

passes[c].x = 1000 - get\_rand\_range\_int(10, 50);

passes[c].y = 600 + get\_rand\_range\_int(10, 50);

}

if(city == 3){

passes[c].x = 75 + get\_rand\_range\_int(10, 50);

passes[c].y = 600 - get\_rand\_range\_int(10, 50);

}

}\*/

void calcPosOfPlane(void \*\_arg) {

int arg = (int)\_arg;

while(1){

if(planes[arg].dir == 0){

if(planes[arg].dirForCit == 0){

planes[arg].tx = cities[1].x;

planes[arg].ty = cities[1].y;

planes[arg].x += planes[arg].speed;

}

if(planes[arg].dirForCit == 1){

planes[arg].tx = cities[2].x;

planes[arg].ty = cities[2].y;

planes[arg].y += planes[arg].speed;

}

if(planes[arg].dirForCit == 2){

planes[arg].tx = cities[3].x;

planes[arg].ty = cities[3].y;

planes[arg].x -= planes[arg].speed;

}

if(planes[arg].dirForCit == 3){

planes[arg].tx = cities[0].x;

planes[arg].ty = cities[0].y;

planes[arg].y -= planes[arg].speed;

}

if(planes[arg].x == planes[arg].tx && planes[arg].y == planes[arg].ty){

if(planes[arg].dirForCit != 3) {

planes[arg].dirForCit++;

}

else planes[arg].dirForCit = 0;

planes[arg].waitFlag = 1;

usleep(planes[arg].waitTime);

planes[arg].waitFlag = 0;

}

}

if(planes[arg].dir == 1){

if(planes[arg].dirForCit == 0){

planes[arg].tx = cities[3].x;

planes[arg].ty = cities[3].y;

planes[arg].y += planes[arg].speed;

}

if(planes[arg].dirForCit == 1){

planes[arg].tx = cities[0].x;

planes[arg].ty = cities[0].y;

planes[arg].x -= planes[arg].speed;

}

if(planes[arg].dirForCit == 2){

planes[arg].tx = cities[1].x;

planes[arg].ty = cities[1].y;

planes[arg].y -= planes[arg].speed;

}

if(planes[arg].dirForCit == 3){

planes[arg].tx = cities[2].x;

planes[arg].ty = cities[2].y;

planes[arg].x += planes[arg].speed;

}

if(planes[arg].x == planes[arg].tx && planes[arg].y == planes[arg].ty){

if(planes[arg].dirForCit != 0)

planes[arg].dirForCit--;

else

planes[arg].dirForCit = 3;

planes[arg].waitFlag = 1;

usleep(planes[arg].waitTime);

planes[arg].waitFlag = 0;

}

}

usleep(20000);

}

}

void PassesLogic(void \*\_arg){

int i = (int)\_arg;

SetDefaultProps(i);

while(1){

if(passes[i].isStop == 1){

MovePasses(i);

if(passes[i].x == passes[i].tx && passes[i].y == passes[i].ty){

passes[i].isStop = 0;

passes[i].wait = 1;

}

}

if(passes[i].wait){

if(abs(passes[i].x - planes[passes[i].dir].x) < 64 && abs(passes[i].y - planes[passes[i].dir].y) < 64 && planes[passes[i].dir].waitFlag == 1 && planes[passes[i].dir].passengers < 5)

{

passes[i].tx = planes[passes[i].dir].x;

passes[i].ty = planes[passes[i].dir].y;

passes[i].wait = 0;

passes[i].isPlane = 1;

}

}

if(passes[i].isPlane == 1){

if(planes[passes[i].dir].waitFlag == 1)

{

if(planes[passes[i].dir].passengers < 5)

MovePasses(i);

else{

passes[i].tx = cities[passes[i].city].stopX;

passes[i].ty = cities[passes[i].city].stopY;

passes[i].isStop = 1;

passes[i].isPlane = 0;

}

if(passes[i].x == passes[i].tx && passes[i].y == passes[i].ty){

if(planes[passes[i].dir].passengers < 5)

{

passes[i].drive = 1;

pthread\_mutex\_lock(&mtx);

planes[passes[i].dir].passengers++;

pthread\_mutex\_unlock(&mtx);

passes[i].canLeave = 0;

passes[i].isPlane = 0;

}

else{

passes[i].tx = cities[passes[i].city].stopX;

passes[i].ty = cities[passes[i].city].stopY;

passes[i].isStop = 1;

passes[i].isPlane = 0;

}

}

}else{

passes[i].tx = cities[passes[i].city].stopX;

passes[i].ty = cities[passes[i].city].stopY;

passes[i].isStop = 1;

passes[i].isPlane = 0;

}

}

if(passes[i].drive == 1)

{

passes[i].x = planes[passes[i].dir].x;

passes[i].y = planes[passes[i].dir].y;

if(planes[passes[i].dir].waitFlag == 0) passes[i].canLeave = 1;

if(planes[passes[i].dir].waitFlag == 1 && passes[i].canLeave ==1)

{

passes[i].drive = 0;

passes[i].isCity = 1;

if(planes[passes[i].dir].passengers > 0)

{

pthread\_mutex\_lock(&mtx);

planes[passes[i].dir].passengers--;

pthread\_mutex\_unlock(&mtx);

}

if(passes[i].dir == 0)

{

if(passes[i].city != 3) passes[i].city++;

else passes[i].city = 0;

}else{

if(passes[i].city != 0) passes[i].city--;

else passes[i].city = 3;

}

passes[i].tx = cities[passes[i].city].x - 150 +rand()%300;

passes[i].ty = cities[passes[i].city].y - 150 +rand()%300;

}

}

if(passes[i].isCity)

{

MovePasses(i);

if(passes[i].x == passes[i].tx && passes[i].y == passes[i].ty){

passes[i].sleep = 1;

usleep(passes[i].sleepTime);

passes[i].sleep = 0;

passes[i].isCity = 0;

passes[i].isStop = 1;

passes[i].wait = 0;

passes[i].dir = rand()%2;

passes[i].tx = cities[passes[i].city].stopX;

passes[i].ty = cities[passes[i].city].stopY;

}

}

usleep(20000);

}

}

/\*void DrawPlanes(){

// GC gc;

while(1) {

gc = XCreateGC(dspl, hwnd, 0, NULL);

XDrawRectangle(dspl, hwnd, gc, planes[0].x, planes[0].y, 25, 15);

XFreeGC(dspl, gc);

XFlush(dspl);

}

}\*/

int countM;

void redraw(sleepTime){

XClearWindow(dspl, hwnd);

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

XDrawRectangle(dspl, hwnd, gc, cities[i].x - 150, cities[i].y - 150, 300, 300);

XDrawRectangle(dspl, hwnd, gc, sizeH + 150, sizeV + 150, 896, 0);

XDrawRectangle(dspl, hwnd, gc, sizeH + 150, sizeV + 150, 0, 384);

XDrawRectangle(dspl, hwnd, gc, sizeH + 150, sizeV + 150 + 384, 896, 0);

XDrawRectangle(dspl, hwnd, gc, sizeH + 150 + 896, sizeV + 150, 0, 384);

for(int i = 0; i < pascount; i++)

XDrawRectangle(dspl, hwnd, gc, passes[i].x-2, passes[i].y-2, 4,4);

char pc[14];

char cpop[24];

sprintf(pc, "passengers: %d", pascount);

XDrawString(dspl,hwnd, gc, 20, 20, pc, (pascount < 10) ? 13 : (pascount > 10 && pascount < 100) ? 14 : 15);

sprintf(cpop, "passengers in planeA: %d", planes[0].passengers);

XDrawString(dspl,hwnd, gc, 500, 20, cpop, 23);

sprintf(cpop, "passengers in planeB: %d", planes[1].passengers);

XDrawString(dspl,hwnd, gc, 500, 50, cpop, 23);

XDrawString(dspl,hwnd, gc, 80, 45, m, sizeof(m)-1);

XDrawString(dspl,hwnd, gc, 1200, 45, n, sizeof(n)-1);

XDrawString(dspl,hwnd, gc, 1220, 710, t, sizeof(t)-1);

XDrawString(dspl,hwnd, gc, 80, 710, l, sizeof(l)-1);

if(planes[0].dirForCit == 0 || planes[0].dirForCit == 2)

XDrawRectangle(dspl, hwnd, gc, planes[0].x - 16, planes[0].y - 4, 32, 16);

else

XDrawRectangle(dspl, hwnd, gc, planes[0].x - 8, planes[0].y - 8, 16, 32);

if(planes[1].dirForCit == 1 || planes[1].dirForCit == 3)

XDrawRectangle(dspl, hwnd, gc, planes[1].x - 16, planes[1].y - 4, 32, 16);

else

XDrawRectangle(dspl, hwnd, gc, planes[1].x - 8, planes[1].y - 8, 16, 32);

XFlush(dspl);

usleep(sleepTime);

}

void main(){

int M;

printf("Enter the count of passes: ");

scanf("%d", &M);

pascount = M;

struct city c[4];

c[0].x = sizeH + 150; c[0].y = sizeV + 150; c[0].stopX = c[0].x + 20; c[0].stopY = c[0].y + 20;

c[1].x = sizeH + 896 + 150; c[1].y = sizeV + 150; c[1].stopX = c[1].x - 20; c[1].stopY = c[1].y + 20;

c[2].x = sizeH + 896 + 150; c[2].y = sizeV + 384 + 150; c[2].stopX = c[2].x - 20; c[2].stopY = c[2].y - 20;

c[3].x = sizeH + 150; c[3].y = sizeV + 384 + 150; c[3].stopX = c[3].x + 20; c[3].stopY = c[3].y - 20;

for(int k = 0; k < 4; k++)

cities[k] = c[k];

dspl = XOpenDisplay(NULL);

gc = XDefaultGC(dspl,0);

if(dspl == 0) {printf("Error XOpenDisplay\n"); exit(1);}

screen = XDefaultScreen(dspl);

hwnd = XCreateSimpleWindow(dspl, RootWindow(dspl, screen), 100,50,1300,800,3, BlackPixel(dspl,screen), WhitePixel(dspl,screen));

if(hwnd == 0) {printf("Error XCreateSimpleWindow\n"); exit(1);}

XSelectInput(dspl, hwnd, ExposureMask | KeyPressMask );

XMapWindow(dspl, hwnd);

KeySym ks;

struct plane planeA;

planeA.x = sizeH + 150;

planeA.y = sizeV + 150;

planeA.tx = sizeH + 150 + 896;

planeA.ty = sizeV + 150;

planeA.speed = 4;

planeA.dir = 0;

planeA.passengers = 0;

planeA.dirForCit = 0;

planeA.waitTime = 2000000;

planeA.waitFlag = 0;

int rc;

planes[0] = planeA;

pthread\_t busFthread;

pthread\_create(&busFthread, NULL, (void\*)calcPosOfPlane, (void\*)0);

struct plane planeB;

planeB.x = sizeH + 150;

planeB.y = sizeV + 150;

planeB.tx = sizeH + 150;

planeB.ty = sizeV + 150 + 384;

planeB.dir = 1;

planeB.speed = 2;

planeB.passengers = 0;

planeB.dirForCit = 0;

planeB.waitTime = 4000000;

planeB.waitFlag = 0;

planes[1] = planeB;

pthread\_t busBthread;

pthread\_create(&busBthread, NULL, (void\*)calcPosOfPlane, (void\*)1);

for(int i = 0; i < pascount; i++)

{

struct pas pass;

pass.x=0;

pass.y =0;

pass.tx = 0;

pass.ty = 0;

pass.city = 0;

pass.drive = 0;

pass.sleep = 0;

pass.maxSleepTime = 32000;

pass.sleepTime = 0;

pass.isCity = 0;

pass.isStop = 1;

pass.wait = 0;

passes[i] = pass;

pthread\_t p1Thread;

pthread\_create(&p1Thread, NULL, (void\*)PassesLogic, (void\*)i);

}

while(1)

{

redraw(50000);

}

}