# 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 Linux графическая модель работы двух лифтов в различных концах здания |

Пояснительная записка

|  |  |
| --- | --- |
| Шифр проекта (работы) | 020-КП-09.03.01-10-ПЗ |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Студента (ки) | | | | Зайцев Владимир Александрович | | | | | | | | | | | | |  |
|  |  |  |  | | | | 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 Linux графическая модель работы двух лифтов в различных концах здания | | | | | | | | | | | | | | | | | |  | |
|  | |  | | | | | | | | | | | | | | | |  | | |
| Студент (ка) | | | | Зайцев Владимир Александрович | | | | | | | | | | | | | |  | | |
|  | | | | фамилия, имя, отчество полностью | | | | | | | | | | | | | |  | | |
| Курс | 2 | | | | | Группа | | | ИВТ-172 |  | | | | | | | | |  | |
|  | |  | | | |  | | |  |  | | | | | | | | |  | |
| 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 29 с., 4 ч., 7 рис., 3 источ, 1 прил.

C, LINUX, XLIB, GCC, МНОГОПОТОЧНОЕ ПРИЛОЖЕНИЕ.

Объектом исследования является алгоритм взаимодействия нескольких потоков в операционной системе Linux при работе в графическом окне.

Цель работы – разработка многопоточной графической модели раобты двух лифтов и пассажиров пользующихся ими.

В ходе работы реализован алгоритм взаимодействия нескольких потоков при работе с графическим окном в операционной системе Linux.

В результате была получена программа, которая демонстрирует модель работы лифта для шести этажей и автономного поведения пассажиров.

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 ОС Linux многопоточную програмную имитацию работы лифтов. Два лифта в различных концах здания обслуживают шесть этажей. Фиксированное число N потенциальных пассажиров двигаются независимо друг от друга по этажам или временно уходят из здания на первом этаже. Через случайное время пассажир подходит к одному из лифтов на том этаже, где они раньше вышли из него, и даожидается появыления лифта, ожидая, если оказывается необходимым, но без образования правильно организованной очрееди, и воходит в лфт для подъема или спуска на случайно вычилсяемый этаж. Грузоподьемность лифта 3 человека. После выхода из лфита пассажиры с разлчной скоростью движутся по этажу. Програма должна пред началом работы вводить значение числа N. Поведение пассажиров и лифта должно имитироваться с помощью отдельных нитей. Для правильного взаимодейтсвия использовать семафоры или мьютексы. Отображение динамики модели одлжно осущетсвляться в графическом окне с помощью условных схем, значков и текстов, размер окна не менее 600 на 800 пикселов. Изображение должно занимать большую часть окна.

Проект состоит из четырех 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 приложений следует иметь в виду необходимость явного указания для системы разработки, что данное приложение 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

Естественно, что при этом могут быть использованы и другие опции вызова компилятора, в частности, явное именование результирующего исполняемого файла и добавление отладочной информации.

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 600x800 пикселов. Графические примитивы рисуются, путём получения координат объекта по x и y. Графический объект также имеет состояния, которые присутствуют в структуре данных этого объекта, например состояние бега, которое говорит о том, что объект в данный момент времени бежит или нет.

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 соответствующий структур в функции, 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 этом цикле.

Процедура ElevThreadFunction отвечает за логику передвижения лифтов между шестью этажами.

Процедура peopleThreadFunction отвечает за логику поведения пассажиров, за их перемещение, выбор лифтов

Процедура showContent отвечает за отрисовку объектов в графическом окне, как статических, так и динамических. В данной процедуре реализован бесконечный цикл, который собственно и содержит вызовы 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. Xevent event -переменная — хранящая данные об обрабатываемых событих графического окна.
5. KeySym ks — переменная необходимая для перевода кода нажатой клавиши в симво char.
6. 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.
7. level — структура необходимая для хранения координат этажей, а так же координат принадлежащих им остановок (места перед лифтами).
8. elev — структура необходимая для хранения информации о лифтах
9. people — структура необходимая для хранения данных о пассажирах.
10. pthread\_mutex\_t hmtx — мьютекс необходимый для блокирования критических участков многопоточной программы.

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

Программа работает за счёт изменения координат x и y с заданной задержкой руководствуясь условиями, определяющими текущее состояние объекта. В результате этих действий координаты графических примитивов изменяются с заданной частотой. Изменение координат происходит за счёт простого инкремента или декремента переменных, отвечающих за соответствующие координаты у объекта графического примитива.

Графическое сопровождение программы реализовано при помощи графического окна и графических примитивов, которые программа рисует в данном графическом окне. В программе реализовано покадровое рисование графических примитивов при помощи бесконечного цикла while, который каждый последующий шаг рисует графические примитивы, а после, в конце цикла, очищает экран. В данном цикле обрабатываются данные как пассажиров, так и лифтов, которые в последствии используются для отображения перемещения пассажира, отображение количества людей, которые стоят возле лифта, отображение самих лифтов по их текущим координатам. Данный цикл расположен в функции showContent а эта процудра в свою очередь находится в бесконечном цикле внутри main. Также в данном цикле уже заранее написаны функции, отвечающие за статические элементы, которые всегда заданного размера и расположены на определённых координатах графического окна, они отвечают за рисование количества пассажиров, а также за рисование статичных этажей.

В главной функции main мы инициализируем графическое окно, создаём указали на наши будущие потоки, после инициализируем начальные данные для пассажиров и лифтов, а затем запускаем их потоки. После чего, запущенные потоки начинают исполнять свои операции, которые определены в их потоковых функциях, которые собственно и отвечают за логику работы пассажиров и перемещения пассажиров между этажами и лифтами.

За поведение пассажиров отвечает функция chelThread. В данной функции описано перемещение пассажиров между рандомными координатами в пределах этажа и лифта, а также взаимодействия пассажира с этими лифтами. Если пассажир не едет лифте, то соответственно он либо бежит по этажу или к лифту, либо ожидает лифт. В таком случае, с помощью циклов описывается движение пассажира от текущего этажа, до следующего, куда ему и нужно. По достижению следующего этажа, пассажир покидает лифт и отправляется в глубь этажа, после чего засыпает на рандомное количество времени. После пробуждения пассажир выдвигается к случайно выбранному лифту.

За поведение лифтов отвечает функция liftThread. В данной функции реализовано перемещение самого лифта между этажами и своевременная его остановка. Для того чтобы количество пассажиров в лифте не превышало максимально-допустимого, был исползован семафор hmtx блокирующий в необходимый момент запись в структуру elev в которой хранится информация о текущем количестве пассажиров. При перемещени лифта, координаты пассажиров также изменяются, при достижении следующего, лифт оповещает об этом пассажира и после чего пассажир присваивает текущий этаж, как свой текущий этаж, в котором он находится. После чего пассажир начинает движение в глубь этажа, а лифт продолжает движение в заданном направлении

Схема алгоритма главной процедуры представлена на рисунке 1.

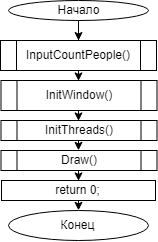


Рисунок 1 – схема алгоритма основной процедуры

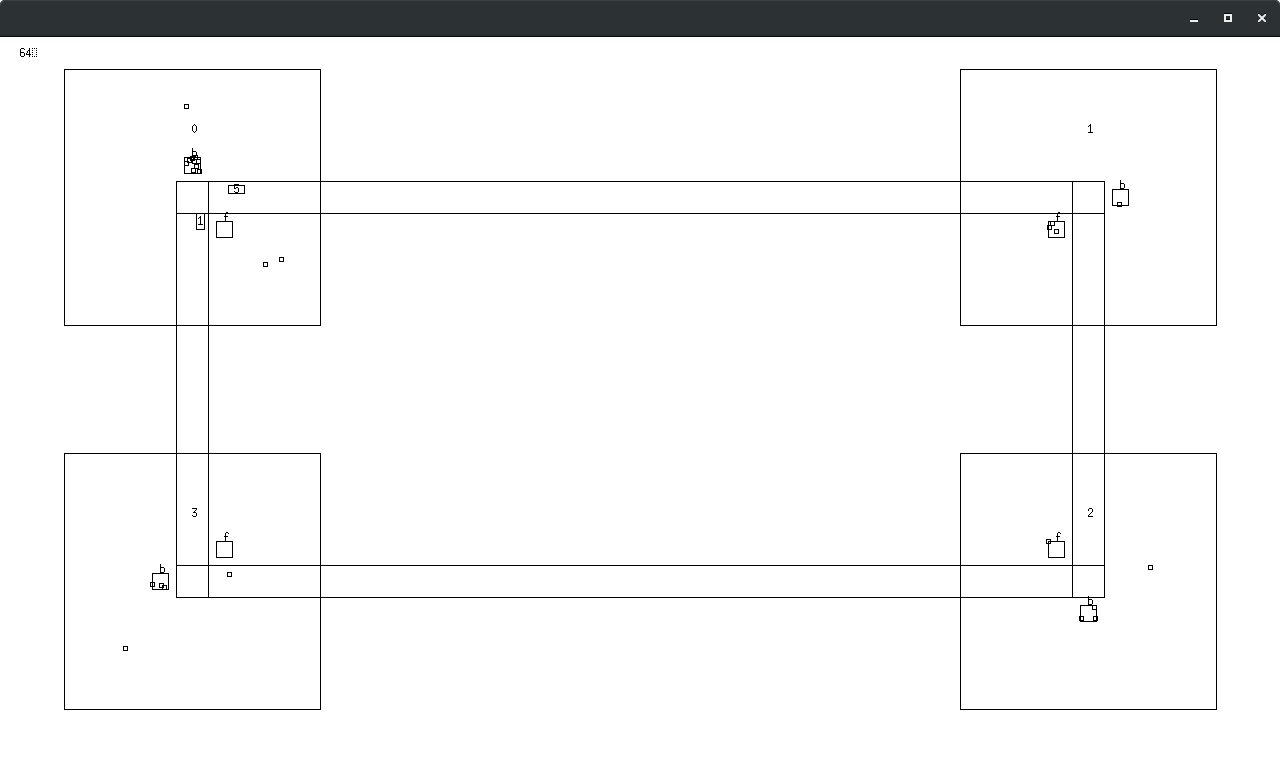
Скриншот работы программы представлен на рисунках 2-6.

Рисунок 2 – пример работы программы

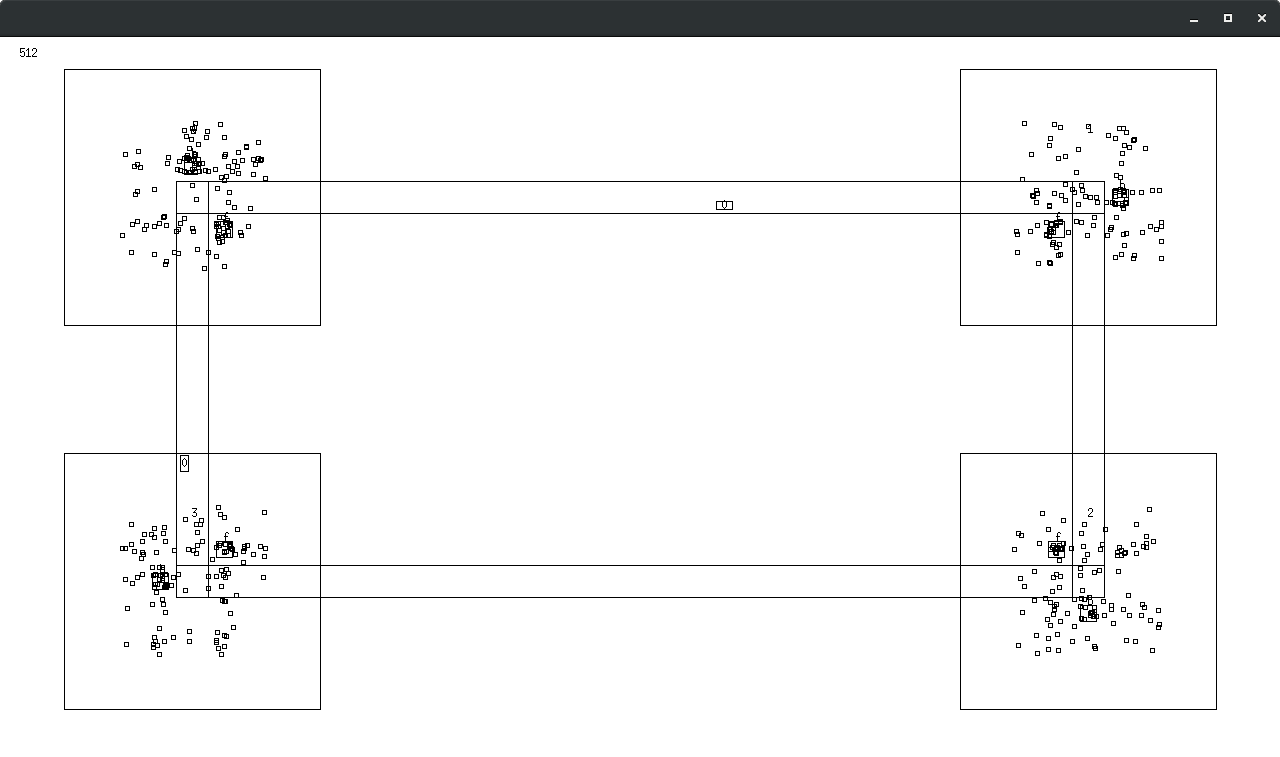


Рисунок 3 – пример работы программы

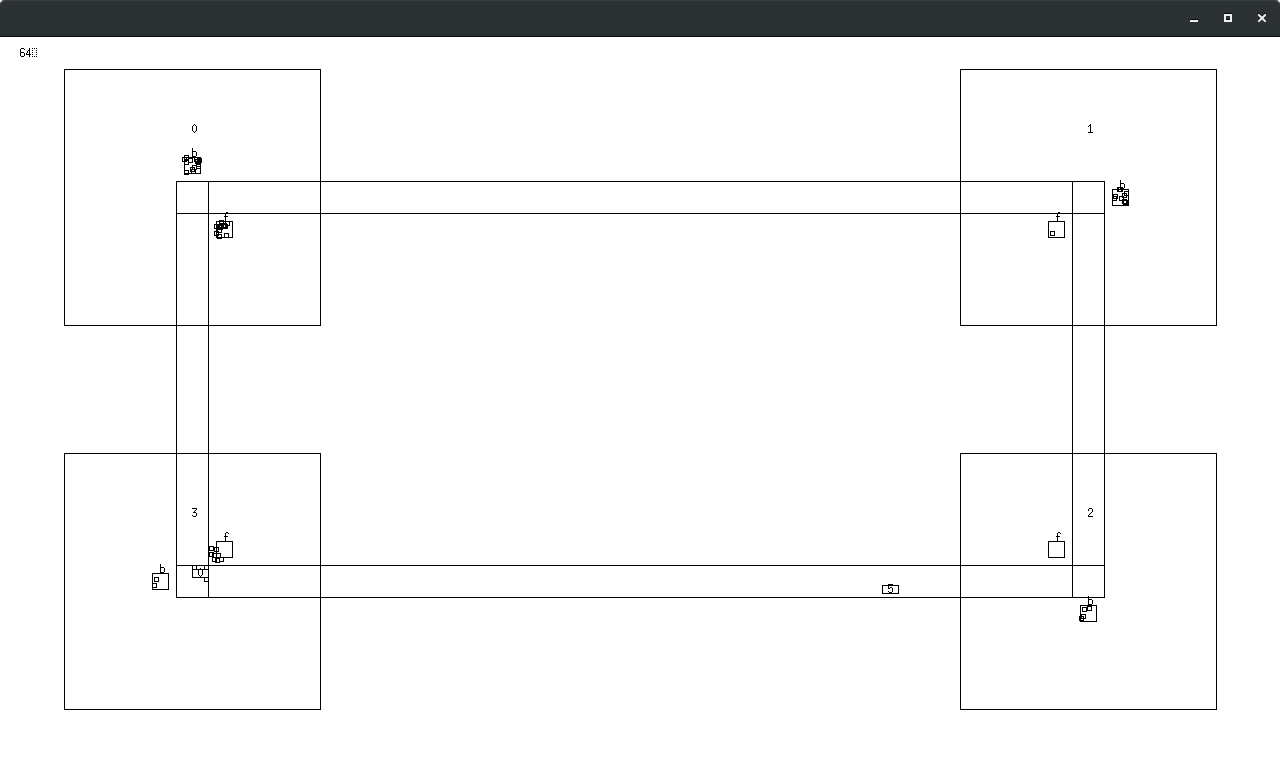


Рисунок 4 – пример работы программы

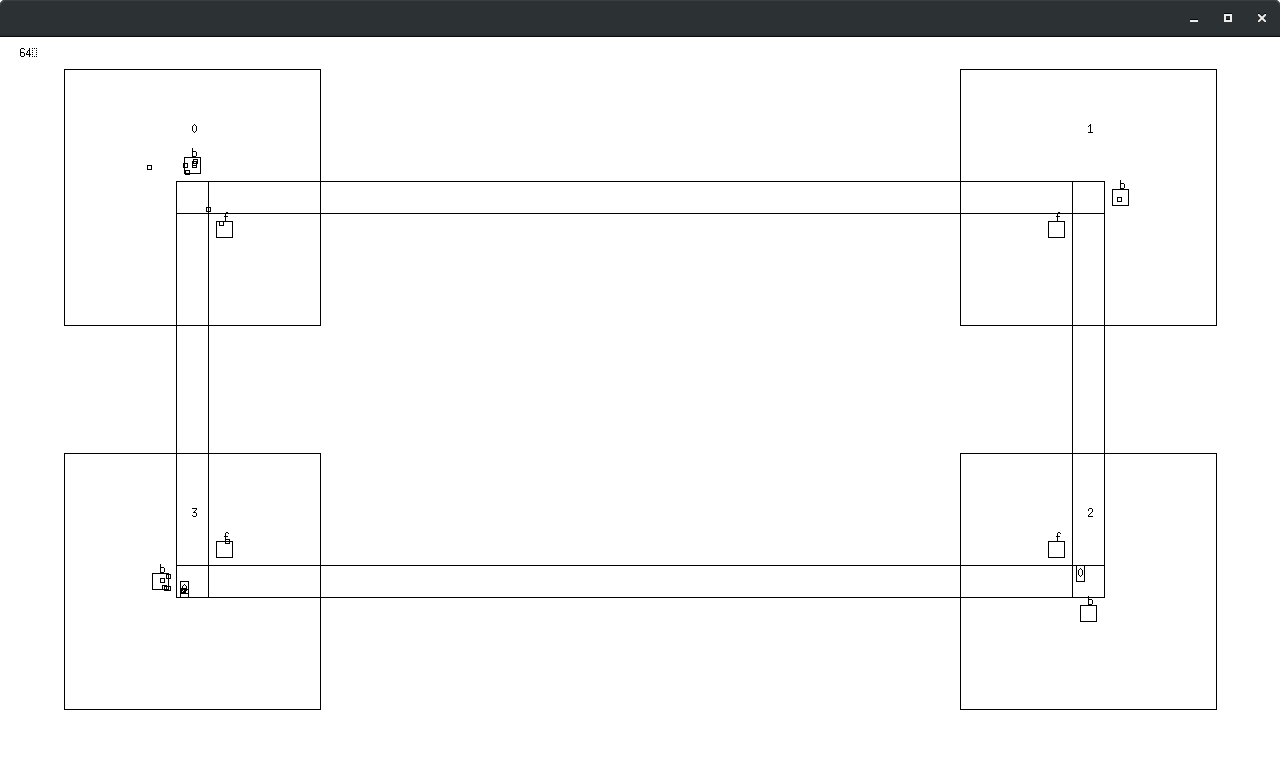
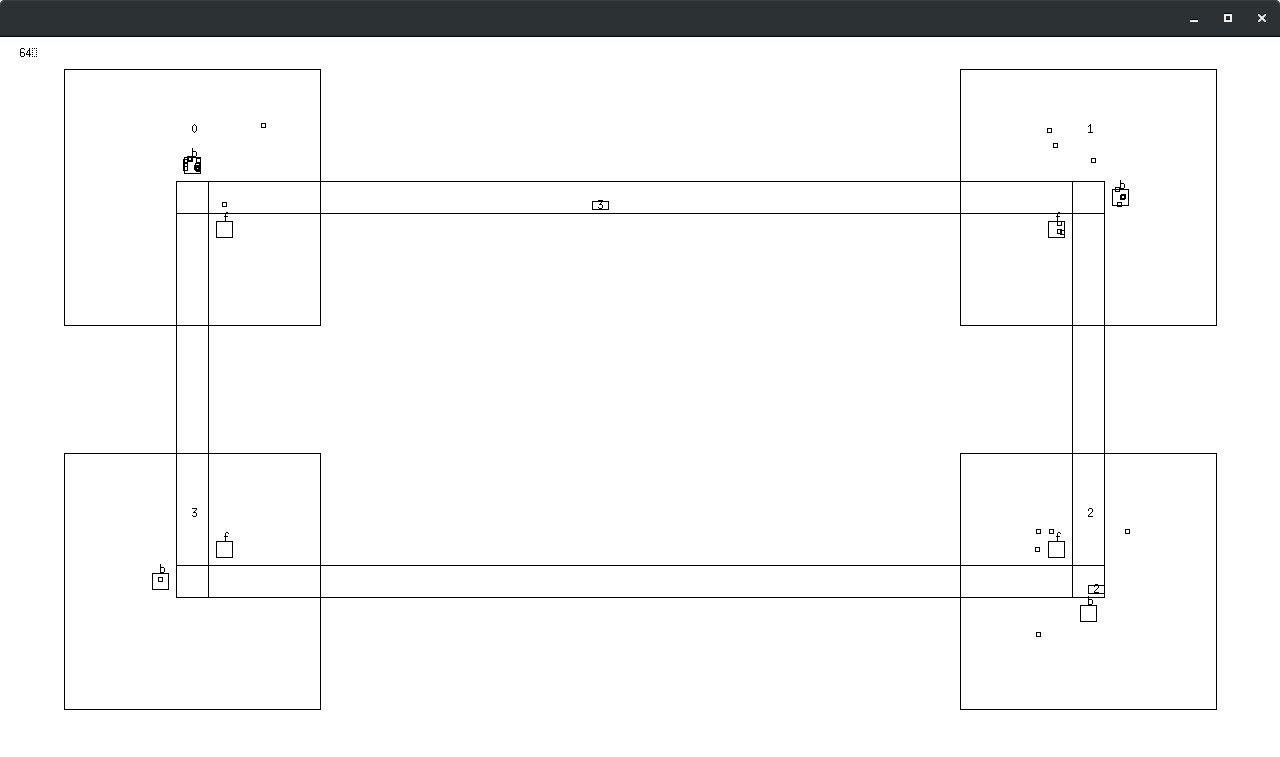
Рисунок 5 – пример работы программы

Рисунок 6 – пример работы программы

Скриншот файла для трансляции программы представлен на рисунке 7.

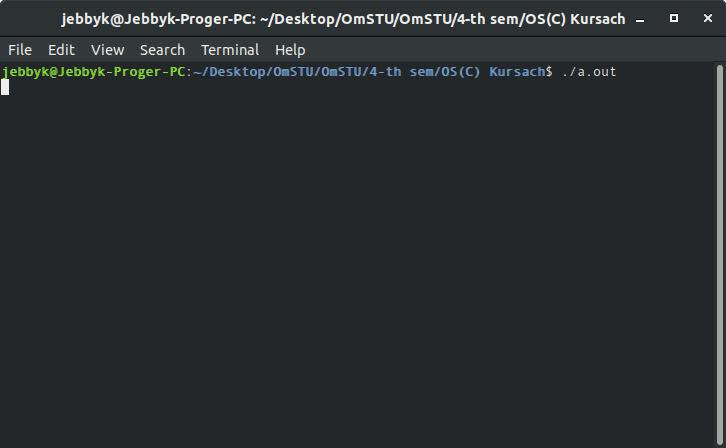


Рисунок 7 – cкриншот файла для трансляции программы

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 графическую модель работы двух лифтов и автономных пассажиров.

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 <pthread.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdlib.h>

#include <X11/Xlib.h>//библиотека для графики

#include <math.h>//библиотека для математических функций

unsigned short target\_framerate = 60;//плавность и скорость работы

Display \*dspl;//ссылка на дисплей для рисования

int screen;

Window hwnd;//ссылка на окно в котором происходит рисование

XEvent event;//события окна

KeySym key\_sym;//переменная для ковертирования клавиш в цифрыы

GC gc;//ссылк на обьект для рисования

pthread\_mutex\_t hmtx;//мьютекс для блокировки доступа к лифту

XColor SvZelen, tZeleny, white, gray, sery, jolty, krasny, siny;//используемые цвета

unsigned int level\_size = 32;//размер этажа

typedef struct{//структура для координат

float x;

float y;

}coord;

typedef struct {//структура для описсания этажа

coord position;//координаты этажа

coord level\_first;//координаты остановочного места для одного лифта

coord level\_second;//координаты остановочного места для второго

}level;

level etajMassiv[6];

void Setlevels()//расположение всех этажей и остановочных мест для каждого этажа

{

etajMassiv[0].position.x = 390;//координаты самих этажей

etajMassiv[0].position.y = 96 \* 6;

etajMassiv[1].position.x = 390;

etajMassiv[1].position.y = 96 \* 5;

etajMassiv[2].position.x = 390;

etajMassiv[2].position.y = 96 \* 4;

etajMassiv[3].position.x = 390;

etajMassiv[3].position.y = 96 \* 3;

etajMassiv[4].position.x = 390;

etajMassiv[4].position.y = 96 \* 2;

etajMassiv[5].position.x = 390;

etajMassiv[5].position.y = 96 \* 1;

for(int i = 0; i < 6; i++)

{

etajMassiv[i].level\_first.x = etajMassiv[i].position.x - 128;//координаты остановочных мест для каждого этажа (для движения в двух лифтах)

etajMassiv[i].level\_first.y = etajMassiv[i].position.y ;

etajMassiv[i].level\_second.x = etajMassiv[i].position.x + 128;

etajMassiv[i].level\_second.y = etajMassiv[i].position.y;

}

}

typedef struct{//структура описывающая персонажей

coord position;//текущий позиция персонажа

coord MoveTo;//кординаты точки, куда нужно идти человеку

short drive;//флаг говорящий о том что человвек едет

short sleep;//спит

short walk;//идет к остановке

short vLift;//идет к лифту

short fromLift;//идет от этажа

short wait;//ждет

short direction;//направление в котором хочет ехать человек

short etaj;//текущий этаж, на котором находится человек

short current\_elev;

float speed;//текущий скорость перемещения человека

float max\_speed;

}people;

typedef struct{//структура описывающая лифт

coord position;//текущий позиция

coord MoveTo;//точка в которую едет лифт

float speed;//его скорость

short drive;//флаг, который указывает, едет ли сейчас лифт

short wait;//ждет на этаже

short chelCount;//количество лудй внутри

short etaj;//текущий этаж, на котором находится лифт

short direction;//направление движения лифту

}elev;

elev elev1;//сам лифт

elev elev2;

elev liftMassiv[2];

people chelovekMassiv[1024];//массив содержащий множество структур со всеми людьми

short N;//переменная хранящая количесттво людей котрое ввел пользователь

void chleThread(void \*\_i)//функция с алгоритмом поведения персонажей

{

int i = \_i;//сохраняемый переданный функции аргумент, это будет номер элемента массива с людьми, к оторому у функции будте доступ

chelovekMassiv[i].etaj = rand()%6;//задаем рандомную остановку при старрте

chelovekMassiv[i].max\_speed = 2 + rand()%9;

chelovekMassiv[i].max\_speed/=10;

chelovekMassiv[i].direction = rand()%2;//задаем ранодомное направление

chelovekMassiv[i].current\_elev = rand()%2;

if(chelovekMassiv[i].etaj == 0) chelovekMassiv[i].direction = 0;//если этаж - нижний, то направление - вперед

if(chelovekMassiv[i].etaj == 5) chelovekMassiv[i].direction = 1;//если этаж - верхний, то направление - обратное

if(chelovekMassiv[i].current\_elev == 0) chelovekMassiv[i].MoveTo = etajMassiv[chelovekMassiv[i].etaj].level\_first;//указываем точку движения персонажа соответствующие

if(chelovekMassiv[i].current\_elev == 1) chelovekMassiv[i].MoveTo = etajMassiv[chelovekMassiv[i].etaj].level\_second;//координаты остановочных мест

chelovekMassiv[i].MoveTo.x += -8 + rand()%16;//добавляем рандомный разброс, чтоб толпились возле лифта

chelovekMassiv[i].position = etajMassiv[chelovekMassiv[i].etaj].position;//указываем начальные координаты самих персонажей

chelovekMassiv[i].position.x += -128 + rand()%256;

chelovekMassiv[i].walk = 1;//ставим флаг, что персоонаж движется к лифту

while(1)//бесконечный цикл для непрерыной обработки персонажей

{

if(chelovekMassiv[i].walk)//елси текущий флаг - движение к лифту

{

coord dir\_vector = {chelovekMassiv[i].MoveTo.x-chelovekMassiv[i].position.x, chelovekMassiv[i].MoveTo.y-chelovekMassiv[i].position.y};//рассчитываем вектор для движеня персонажа

float l = sqrt(pow(dir\_vector.x,2)+pow(dir\_vector.y, 2));//рассчитываем длину этого вектора

coord normal\_v = dir\_vector;//нормализованный вектор, который должен иметь то же направление, но длинну 1

normal\_v.x /= l;//для этого делим его на длину первого вектора

normal\_v.y /= l;

if(chelovekMassiv[i].speed < chelovekMassiv[i].max\_speed && chelovekMassiv[i].speed < l/20)//если скорость персонажа меньше достаточно млаа

{

chelovekMassiv[i].speed += 0.01;//то постеменно ее увеличиваем

}

if(chelovekMassiv[i].speed > l/20)//если скорость слишком велика по отношению к оставшемуся расстоянию до цели,

{

chelovekMassiv[i].speed = l/20;//то снижаем ее

}

chelovekMassiv[i].position.x += normal\_v.x \* chelovekMassiv[i].speed;//изменяем координаты персонажа на основании его скорости и нормализованного вектора длвижения

chelovekMassiv[i].position.y += normal\_v.y \* chelovekMassiv[i].speed;

if(chelovekMassiv[i].speed < 0.001)//если персонаж остановился

{

chelovekMassiv[i].walk = 0;//сбрасываем флаг, что он движется к остановке

chelovekMassiv[i].wait = 1;//устанавливаем флаг ожидания транспорта

}

}

if(chelovekMassiv[i].wait > 0)//если активен флаг ожидания

{

//смотрим, не переполнен ли лифт, остановился ли он, и находится ли лифт на той же остановке, что и персонаж

if(chelovekMassiv[i].direction == liftMassiv[chelovekMassiv[i].current\_elev].direction && chelovekMassiv[i].etaj == liftMassiv[chelovekMassiv[i].current\_elev].etaj && liftMassiv[chelovekMassiv[i].current\_elev].chelCount < 3)

{

chelovekMassiv[i].vLift = 1;//елси это так, то устанавливаем флаг, что персонаж движется к лифту

chelovekMassiv[i].wait = 0;//сбрасываем флаг ожидания транспорта

chelovekMassiv[i].MoveTo = liftMassiv[chelovekMassiv[i].current\_elev].position;//утсанавливаем точку перемещения персонжа - координаты лифта

}

}

if(chelovekMassiv[i].vLift == 1)//если активен флаг передвижения к лифту

{

if(liftMassiv[chelovekMassiv[i].current\_elev].chelCount < 4 && liftMassiv[chelovekMassiv[i].current\_elev].wait > 0)//если лифт все еще ждет и там есть свободные места

{

coord dir\_vector = {chelovekMassiv[i].MoveTo.x-chelovekMassiv[i].position.x, chelovekMassiv[i].MoveTo.y-chelovekMassiv[i].position.y};//активируем алгоритм перемещения

float l = sqrt(pow(dir\_vector.x,2)+pow(dir\_vector.y, 2));

coord normal\_v = dir\_vector;

normal\_v.x /= l;

normal\_v.y /= l;

if(chelovekMassiv[i].speed < chelovekMassiv[i].max\_speed && chelovekMassiv[i].speed < l/20)

{

chelovekMassiv[i].speed += 0.01;

}

if(chelovekMassiv[i].speed > l/20)

{

chelovekMassiv[i].speed = l/20;

}

chelovekMassiv[i].position.x += normal\_v.x \* chelovekMassiv[i].speed;

chelovekMassiv[i].position.y += normal\_v.y \* chelovekMassiv[i].speed;

if(chelovekMassiv[i].speed < 0.01)//и если человек почти дошел

{

pthread\_mutex\_lock(&hmtx);//захватываем мьютексб на доступ к лифту

chelovekMassiv[i].drive = 1;//устанавливаем флаг езды

liftMassiv[chelovekMassiv[i].current\_elev].chelCount++;//увеличиваем у лифта информацию о количестве пассажирова

chelovekMassiv[i].vLift = 0;//сбрасываем флаг движения к лифту

pthread\_mutex\_unlock(&hmtx);//отпускаем мьютекс

}

}else{//елси лифт уже поехал, или там уже закончились места

chelovekMassiv[i].walk = 1;//идем обратно на туже остановку

chelovekMassiv[i].vLift = 0;//сбарсываем флаг движения к лифту

if(chelovekMassiv[i].current\_elev == 0)//в зависимости от текущего направления, в котром решил ехать персонаж

{

chelovekMassiv[i].MoveTo = etajMassiv[chelovekMassiv[i].etaj].level\_first;//выбираем координаты нужного остановочного места как цлеевые/

chelovekMassiv[i].MoveTo.x += -8 + rand()%16;//добавляем немного рандома для создания толпы

}

if(chelovekMassiv[i].current\_elev == 1)//в зависимости от текущего направления, в котром решил ехать персонаж

{

chelovekMassiv[i].MoveTo = etajMassiv[chelovekMassiv[i].etaj].level\_second;//выбираем координаты нужного остановочного места как цлеевые/

chelovekMassiv[i].MoveTo.x += -8 + rand()%16;//рандом для толпы

}

}

}

if(chelovekMassiv[i].drive == 1)//есди установлен флаг движени в лифту

{

chelovekMassiv[i].position = liftMassiv[chelovekMassiv[i].current\_elev].position;//приравнивем координаты пассажира к координатам лифта

if(liftMassiv[chelovekMassiv[i].current\_elev].wait && chelovekMassiv[i].etaj != liftMassiv[chelovekMassiv[i].current\_elev].etaj)//еслли лифт остановился, и этаж не равна предыдущей остановке, на котором былл человек

{

chelovekMassiv[i].etaj = liftMassiv[chelovekMassiv[i].current\_elev].etaj;//обновляем информацию о текущей остановке персонажа

int r = rand()%2;//делаем рандом который вернет 1 или 0

if(r == 1)//елс 1 то

{

pthread\_mutex\_lock(&hmtx);//захватываем мьютекс

chelovekMassiv[i].drive = 0;//сбрасываем флаг езда на лифту

chelovekMassiv[i].fromLift = 1;//устанавливаем флаг движения от этажа

liftMassiv[chelovekMassiv[i].current\_elev].chelCount--;//уменьшаем информациию о количестве пассажиров в лифте

chelovekMassiv[i].MoveTo = etajMassiv[chelovekMassiv[i].etaj].position;//указываем целевую точку движения - координаты текущего этажа

chelovekMassiv[i].MoveTo.x += -64 + rand()%128;//и добавляем большой разброс в координтаах

if(chelovekMassiv[i].etaj == 0) chelovekMassiv[i].MoveTo.x += -512 + rand()%1024;

pthread\_mutex\_unlock(&hmtx);//отпускаем мьютекс

}

}

}

if(chelovekMassiv[i].fromLift == 1)//если установлен флаг движения в направлении от становки

{

coord dir\_vector = {chelovekMassiv[i].MoveTo.x-chelovekMassiv[i].position.x, chelovekMassiv[i].MoveTo.y-chelovekMassiv[i].position.y};//запускается алгорим перемещения

float l = sqrt(pow(dir\_vector.x,2)+pow(dir\_vector.y, 2));

coord normal\_v = dir\_vector;

normal\_v.x /= l;

normal\_v.y /= l;

if(chelovekMassiv[i].speed < chelovekMassiv[i].max\_speed && chelovekMassiv[i].speed < l/20)

{

chelovekMassiv[i].speed += 0.01;

}

if(chelovekMassiv[i].speed > l/20)

{

chelovekMassiv[i].speed = l/20;

}

chelovekMassiv[i].position.x += normal\_v.x \* chelovekMassiv[i].speed;

chelovekMassiv[i].position.y += normal\_v.y \* chelovekMassiv[i].speed;

if(chelovekMassiv[i].speed < 0.01)//елси дошел до цлеи

{

chelovekMassiv[i].fromLift = 0;//сбрасыаем флаг движения от этажа

chelovekMassiv[i].sleep = 1;//устанавливаем флаг сна

usleep(100000\*(rand()%100 + 100));//засыпаем на неопределенное количество времени

chelovekMassiv[i].sleep = 0;//после сна сбрасываем флаг сна

chelovekMassiv[i].walk = 1;//устанавливаем флаг движения к остановке снова

chelovekMassiv[i].direction = rand()%2;//делаем рандомное направление в котором хочет ехть человек

chelovekMassiv[i].current\_elev = rand()%2;

if(chelovekMassiv[i].etaj == 0) chelovekMassiv[i].direction = 0;//елси текущий этаж - крайняя то точно указываем направление в котором вообще можно ехать

if(chelovekMassiv[i].etaj == 5) chelovekMassiv[i].direction = 1;

if(chelovekMassiv[i].current\_elev == 0)//в зависимости от выбранного направления

{

chelovekMassiv[i].MoveTo = etajMassiv[chelovekMassiv[i].etaj].level\_first;//выбираем нужное остановочное место

chelovekMassiv[i].MoveTo.x += -8 + rand()%16;//и добавляем рандом для создания толп

}

if(chelovekMassiv[i].current\_elev == 1)

{

chelovekMassiv[i].MoveTo = etajMassiv[chelovekMassiv[i].etaj].level\_second;

chelovekMassiv[i].MoveTo.x += -8 + rand()%16;

}

}

}

usleep(1000000/target\_framerate);//отправляем поток в сон на 1 шестидесятую секунды.

}

}

void showContent()//фнукция для отображения всего происходящего через окно

{

XClearWindow(dspl, hwnd);//сначал очищщаем экран

XSetForeground(dspl, gc, jolty.pixel);//устанавливаем цвет рисования - желтый

XFillRectangle(dspl, hwnd, gc, liftMassiv[0].position.x-8, liftMassiv[0].position.y - 16, 16, 32);//заполняем прямоугльник лифту желтым

XFillRectangle(dspl, hwnd, gc, liftMassiv[1].position.x-8, liftMassiv[1].position.y - 16, 16, 32);

coord op;//вспомогательная координата предыдущей этажа для рисования дороги

for(int i = 0; i < 6; i++)//для всех четырех остановок

{

XSetForeground(dspl, gc, krasny.pixel);//устанавливаем красный цвет рисования

coord p1 = etajMassiv[i].level\_second;//берем координты остановчного места для обратного движения

XDrawRectangle(dspl, hwnd, gc, p1.x-level\_size/2, p1.y-level\_size/2, level\_size, level\_size);//и рисуем эту осттановку

XSetForeground(dspl, gc, SvZelen.pixel);//для остновочного место с обычным направлением движения берем зеленую кисть

coord p2 = etajMassiv[i].level\_first;//и выбираем координаты нужного остановочного метса

XDrawRectangle(dspl, hwnd, gc, p2.x-level\_size/2, p2.y-level\_size/2, level\_size, level\_size);

coord p = etajMassiv[i].position;//вспомогательная координата текущей этажа

XSetForeground(dspl, gc, sery.pixel);//устанавливаем ссерый цвет рисования

XDrawLine(dspl, hwnd, gc, p.x-256, p1.y+level\_size/2, p.x+256, p1.y+level\_size/2);//и рисуем эту осттановку

char buf[1];

sprintf(buf, "%d", i+1);

XDrawString(dspl, hwnd, gc, p.x-256, p1.y, buf,1);

}

/\* char pc[4] = "";

sprintf(pc, "%d", N);//преобразуем количество людей в строку\*/

for(int i = 0; i < N; i++)//проходим циклом по всему ссписку людей в масиве

{

if(chelovekMassiv[i].sleep == 1) XSetForeground(dspl, gc, sery.pixel);//елси человек спит, то рисуем его серым

else XSetForeground(dspl, gc, BlackPixel(dspl,screen));//иначи рисуем белым

XFillArc(dspl, hwnd, gc, chelovekMassiv[i].position.x - 4, chelovekMassiv[i].position.y - 12+4, 8, 24, 0,360\*64);//куржок символизирующий персонажа

XFillArc(dspl, hwnd, gc, chelovekMassiv[i].position.x - 4, chelovekMassiv[i].position.y - 20+4, 8, 8, 0,360\*64);//куржок символизирующий персонажа

}

XFlush(dspl);// отправление графической информации в само окно на вывод

}

void liftThread(void \*arg)

{

while(1)//запускаем безконечный цикл

{

for(int j = 0; j < 2; j++)

{

if(liftMassiv[j].drive == 1)//в нем проверяем флаг езды, и еси он 1, то

{

coord dir\_vector = {liftMassiv[j].MoveTo.x-liftMassiv[j].position.x, liftMassiv[j].MoveTo.y-liftMassiv[j].position.y};//выполняем перемещение так же как и у пассажиров, только с более высокой скоростью

float l = sqrt(pow(dir\_vector.x,2)+pow(dir\_vector.y, 2));

coord normal\_v = dir\_vector;

normal\_v.x /= l;

normal\_v.y /= l;

if(liftMassiv[j].speed < 10 && liftMassiv[j].speed < l/20)//сдесь ограничение скорости выставленно в 2 раза выше чем у человеков

{

liftMassiv[j].speed += 0.05;

}

if(liftMassiv[j].speed > l/20)

{

liftMassiv[j].speed = l/20;

}

liftMassiv[j].position.x += normal\_v.x \* liftMassiv[j].speed;

liftMassiv[j].position.y += normal\_v.y \* liftMassiv[j].speed;

if(liftMassiv[j].speed < 0.001)//елси лифту почти полностью осттановилось, то

{

liftMassiv[j].drive = 0;//останавливаем его полностью, сбрасывае флаг езды

liftMassiv[j].wait = 360;//устанавливая вемя ожиданя

if(liftMassiv[j].direction == 0)//дальше проверяем, направление движени

{

liftMassiv[j].etaj++;//прибавляем 1 к информации о ткущей остановке лифту

if(liftMassiv[j].etaj == 5) //еси она оказалась на крайней остановке

{

liftMassiv[j].direction = 1;//изменяем направление движения на обратное

}

}else{//тут вспе тоже самое для только для обратного направления движения

liftMassiv[j].etaj--;

if(liftMassiv[j].etaj == 0)

{

liftMassiv[j].direction = 0;

}

}

}

}

if(liftMassiv[j].wait > 0)//елси еще не закончислоь время ожидания

{

liftMassiv[j].wait--;//то ждем,и немного это времяуменьшаем

if(liftMassiv[j].wait == 0) //елси время истекло

{

liftMassiv[j].drive = 1;//устанавливаем флаг езды

int c;

if(liftMassiv[j].direction == 0)//в зависимости от направления движения

{

c = liftMassiv[j].etaj + 1;////выбираем слудующую останвоку

}

if(liftMassiv[j].direction == 1)

{

c = liftMassiv[j].etaj - 1;

}

liftMassiv[j].MoveTo.y = etajMassiv[c].position.y;//передаем ее координаты в целевую точку для движения

if(j == 0) liftMassiv[j].MoveTo.x = etajMassiv[c].position.x - 192;

else liftMassiv[j].MoveTo.x = etajMassiv[c].position.x + 192;

}

}

}

usleep(1000000/target\_framerate);//отправляем основной поток вв сон на одну шестидесятую секунды

}

}

void main()

{

dspl = XOpenDisplay(NULL);//создание окна

gc = XDefaultGC(dspl,0);//дескриптор для рисования в графическом окне

if(dspl == 0) {printf("Error XOpenDisplay\n"); exit(1);}//выводим ошибку если не получиолсь соткрыть дисплей

screen = XDefaultScreen(dspl);//сохранаяем инфо о экране

hwnd = XCreateSimpleWindow(dspl, RootWindow(dspl, screen), 100,50,800,600,3, BlackPixel(dspl,screen), WhitePixel(dspl,screen));//созаем простое графическое окно с размерами 1200 на 600

Colormap screen\_colormap;//созадем новую палитру цветов

screen\_colormap = DefaultColormap(dspl, DefaultScreen(dspl));//копируем то что есть по умолчанию

XAllocNamedColor(dspl, screen\_colormap, "green", &SvZelen, &SvZelen);//у добавляем туда свои цвета

XAllocNamedColor(dspl, screen\_colormap, "gold", &jolty, &jolty);

XAllocNamedColor(dspl, screen\_colormap, "web gray", &sery, &sery);

XAllocNamedColor(dspl, screen\_colormap, "krasny", &krasny, &krasny);

XAllocNamedColor(dspl, screen\_colormap, "white", &white, &white);

XAllocNamedColor(dspl, screen\_colormap, "dark green", &tZeleny, &tZeleny);

XAllocNamedColor(dspl, screen\_colormap, "sky siny", &siny, &siny);

XSetForeground(dspl, gc, BlackPixel(dspl, screen));//утсанавливаем желлтый цвет рисования

if(hwnd == 0) {printf("Error XCreateSimpleWindow\n"); exit(1);}//если произошли ошибки с окном то выводим предупреждение и выходим

XSelectInput(dspl, hwnd, ExposureMask | KeyPressMask );//здес указываем какой ввод с мыши и клавиатуры мы будем считывать во время работы программы

XMapWindow(dspl, hwnd);

int n = 0;//флаг для ввода цифр

char Nstr[4] = "";//строка для хранения вводимого числа

while(n != -1)//пока не сброшен флаг

{

XNextEvent(dspl, &event);//считыавем события от окна Х сревера

if(event.type == KeyPress)//если событие - это нажатие клавиши на клавиатурпе

{

char kb[1];

XLookupString(&event.xkey, kb,1, &key\_sym, 0);//преобразуем его в сивол

char c = kb[0];

int i = c - '0';//потом из символа в цифру

if(i >= 0 && i <=9)//если она в пределах от 0 до 9

{

XDrawString(dspl, hwnd, gc, 10 + n\*10, 10, kb, 1);//выводим на дисплей его

Nstr[n] = kb[0];//записываем в сстроку

n++;// увеличиваем флаг чисел

}else{

n = -1;//елси ввели не знам между 0 и 9 то сбрасываем флаг ввода чисел

sscanf(Nstr, "%d", &N);//и сохраняем количество персонажей

}

}

}

Setlevels();//выполняем расположение остановк и остановочных мест в соответствии с координатами

for(int i = 0; i < N; i++)//запускаем цикл на количество итераций равному числу, которое ввел польователь

{

people p;//создаем переменную для человека

pthread\_t thread;//перменную для потока

pthread\_create(&thread, NULL, (void\*)chleThread, (void\*)i);//запускаем поток

}

pthread\_t thread;//перменную для потока

pthread\_create(&thread, NULL, (void\*)liftThread, (void\*)0);//запускаем поток

//дальше идет алгоритм для маршрутного лифту

elev1.drive = 1;//усатнавливаем, что при старте программы лифту едет

elev1.direction = 1;//указываем направление (на первой же остановке оно поменяется)

elev1.etaj = 1;//и сотановка

elev1.MoveTo.y = etajMassiv[0].position.y;//укаываем начальную координату

elev1.MoveTo.x = etajMassiv[0].position.x - 192;

elev1.position.y = etajMassiv[0].position.y+1;//укаываем начальную координату

elev1.position.x = etajMassiv[0].position.x - 192;

elev2.drive = 1;//усатнавливаем, что при старте программы лифту едет

elev2.direction = 1;//указываем направление (на первой же остановке оно поменяется)

elev2.etaj = 2;//и сотановка

elev2.MoveTo.y = etajMassiv[1].position.y;//укаываем начальную координату

elev2.MoveTo.x = etajMassiv[1].position.x + 192;

elev2.position.y = etajMassiv[1].position.y+1;//укаываем начальную координату

elev2.position.x = etajMassiv[1].position.x + 192;

liftMassiv[0] = elev1;

liftMassiv[1] = elev2;

while(1)

{

showContent();//выполняем отображене всех людей, лифту и остановоок в графическом окне

usleep(1000000/target\_framerate);//отправляем основной поток вв сон на одну шестидесятую секунды

}

getchar(); printf("\033[0m");

}