# 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 для шетси этажей и автономнго поведения пассажиров. |

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 для шести этажей и автономного поведения пассажиров | | | | | | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | | | | |  | |
| 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 25 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 peopleFunctionThread. 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 elevThreadFucntion. 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.

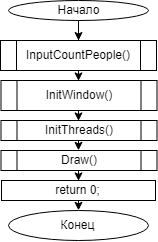


Рисунок 1 – схема алгоритма основной процедуры

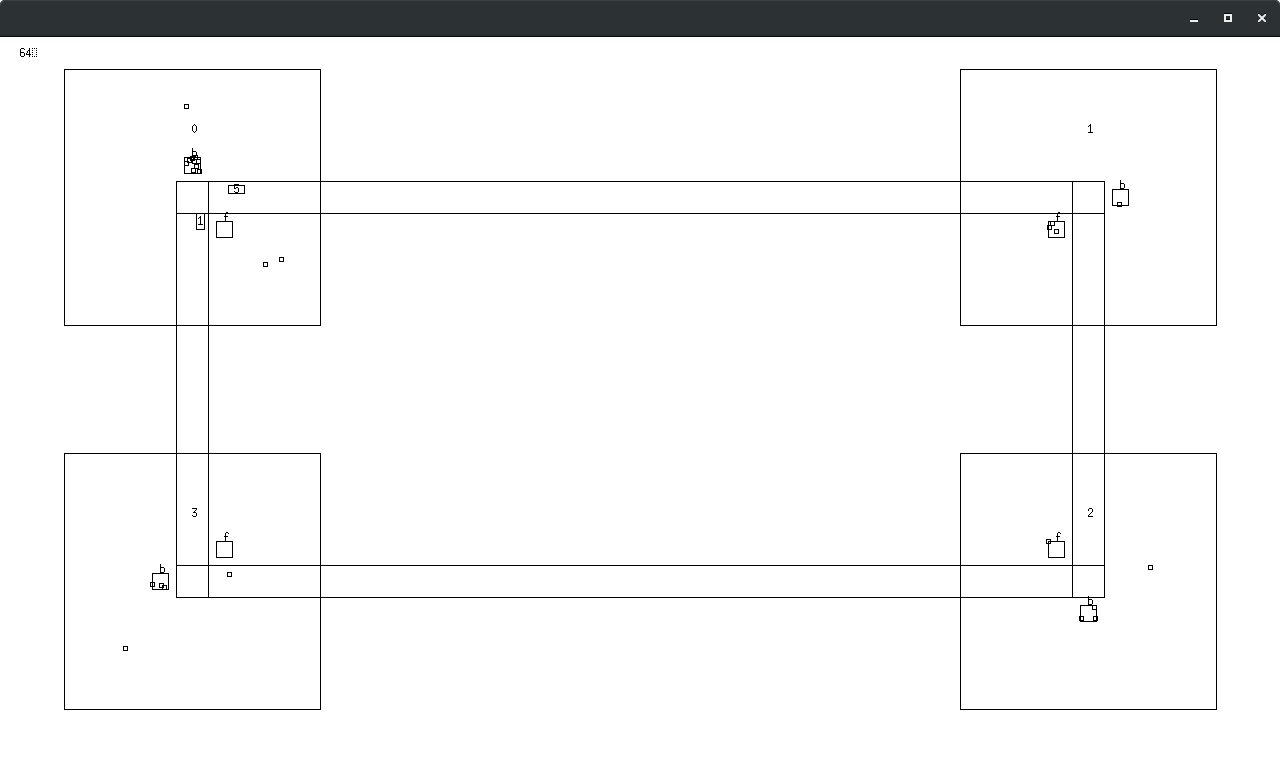
Скриншот работы программы представлен на рисунках 2-6.

Рисунок 2 – пример работы программы

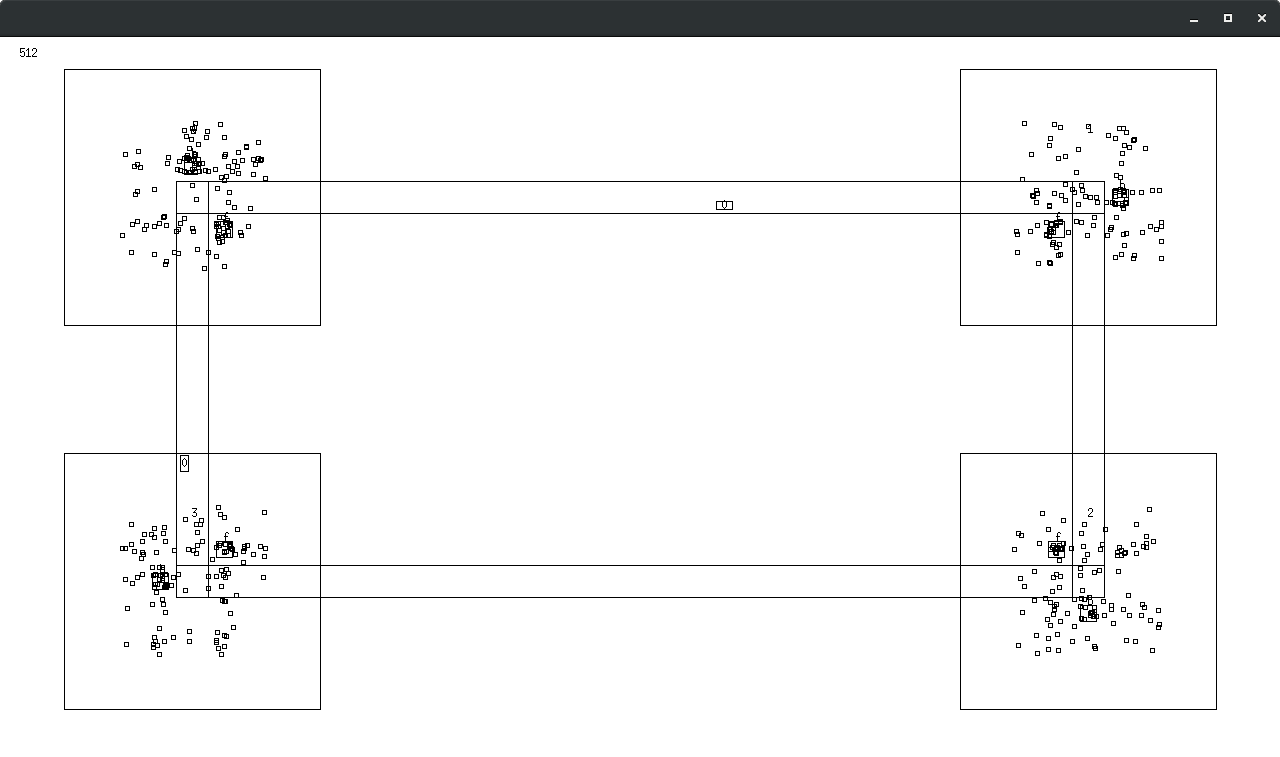


Рисунок 3 – пример работы программы

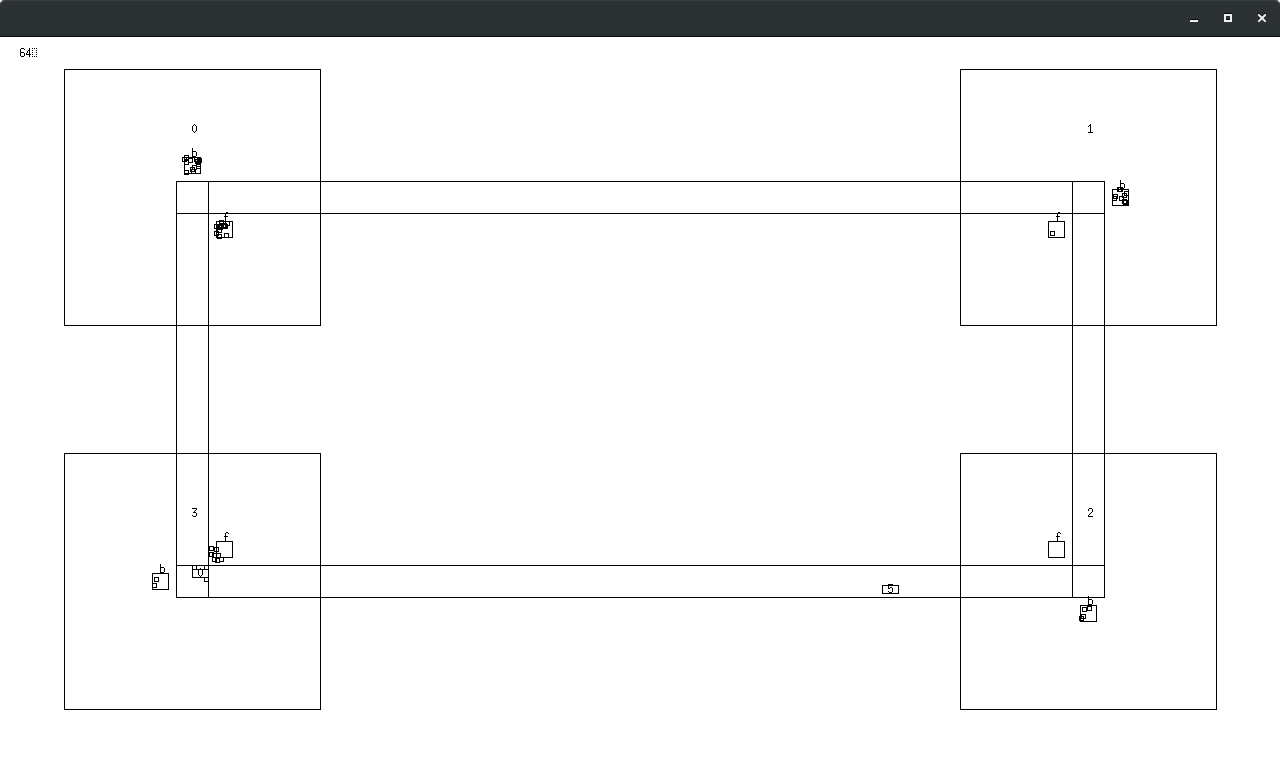


Рисунок 4 – пример работы программы

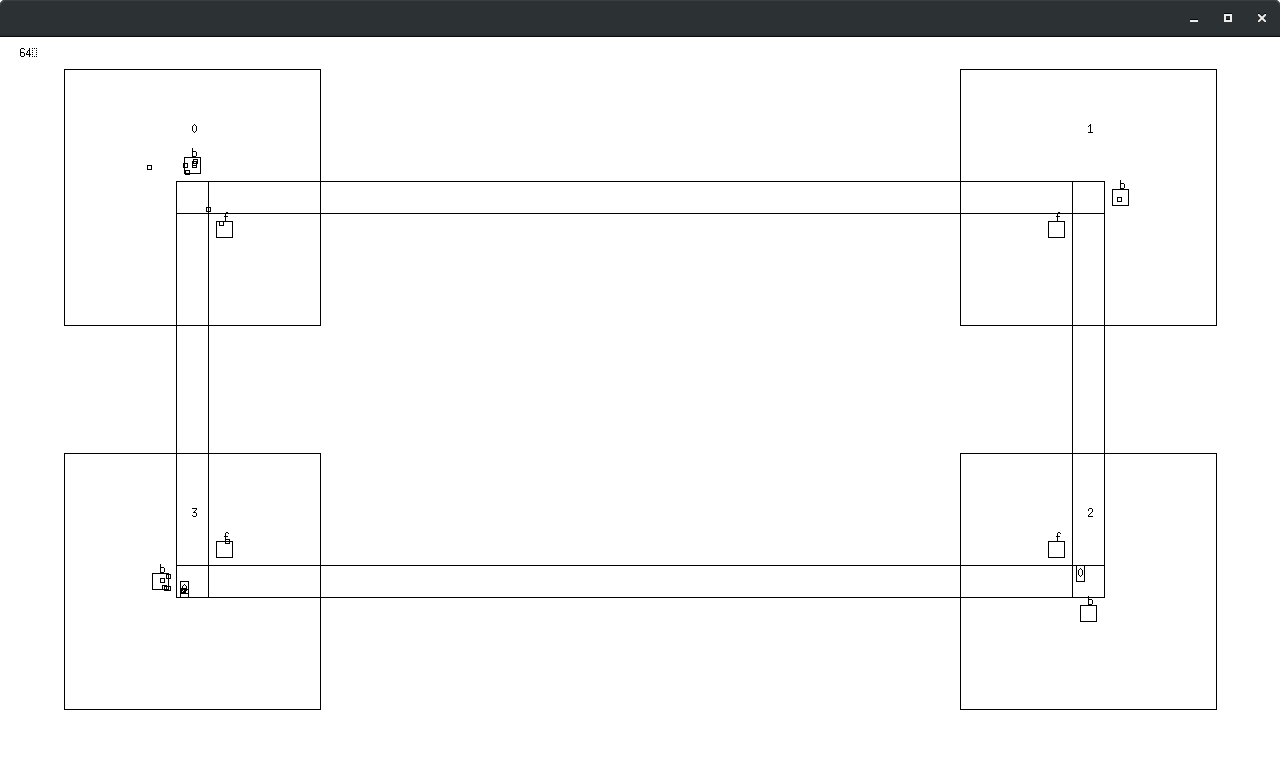
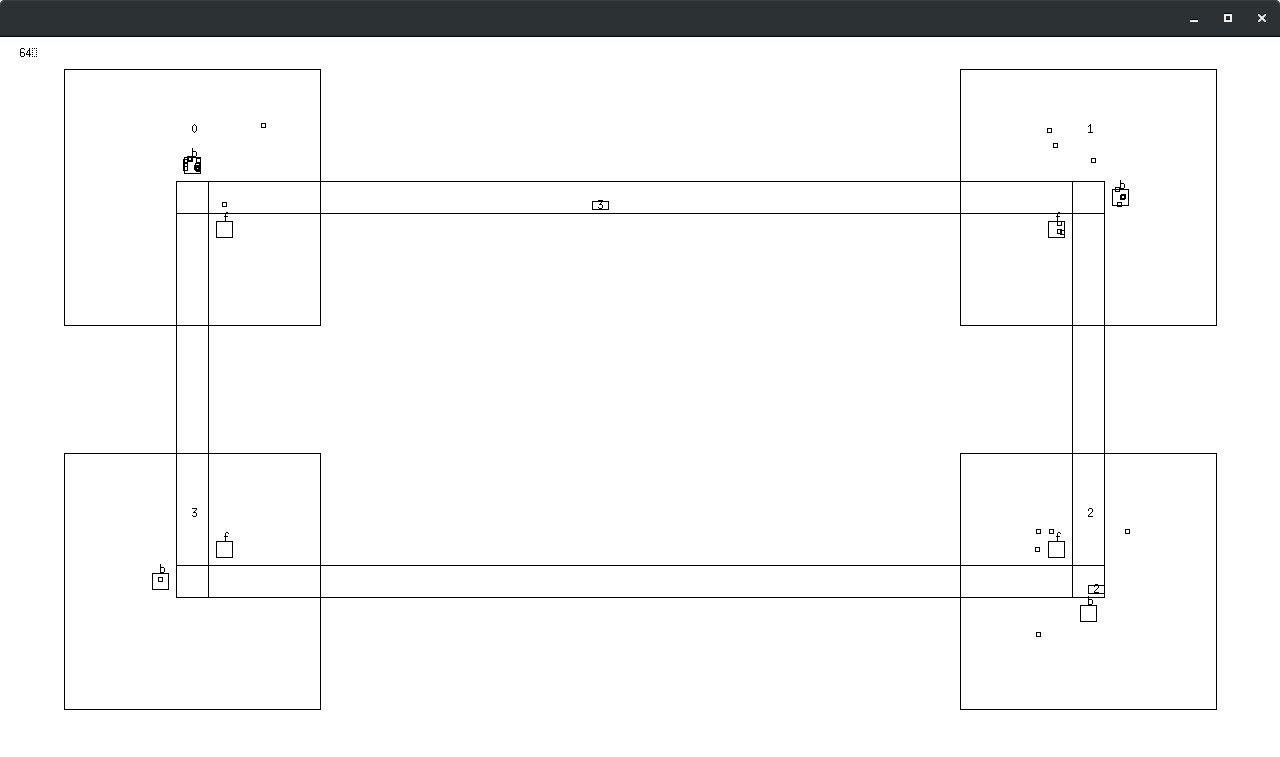
Рисунок 5 – пример работы программы

Рисунок 6 – пример работы программы

Скриншот файла для трансляции программы представлен на рисунке 7.

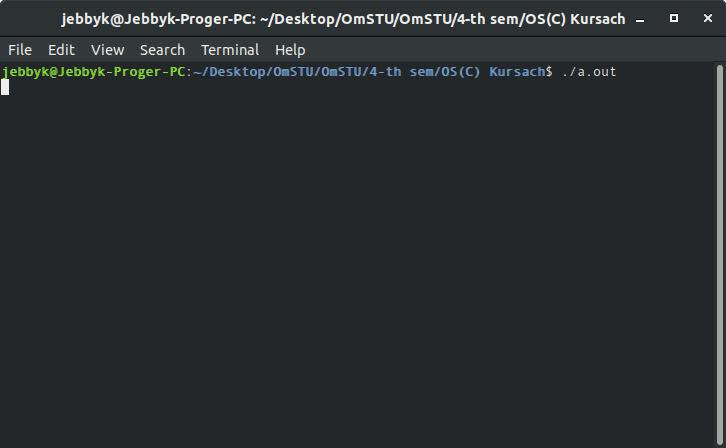


Рисунок 7 – cкриншот файла для трансляции программы

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 <pthread.h> |
| #include <unistd.h> |
| #include <fcntl.h> |
| #include <stdlib.h> |
| #include <X11/Xlib.h> |
|  |
| #include <math.h> |
|  |
|  |
| Display \*dspl; |
| int screen; |
| Window hwnd; |
| XEvent event; |
| KeySym ks; |
| GC gc; |
|  |
| int roadTickness = 16; |
|  |
| pthread\_mutex\_t hmtx; |
|  |
| int passengersCount = 0; |
|  |
|  |
|  |
| int citySize = 256; |
| int vPadding = 32;// distances from top of the window to the city edge |
| int hPadding = 64;// |
| int vDistance = 256+128;//distances between cities |
| int hDistance = 512+256 + 128; |
| int rThickness = 32;//thickness of roads |
|  |
| struct city{ |
| int x, y; |
| int stopFx, stopFy; |
| int stopBx, stopBy; |
| char name[5]; |
| }; |
|  |
| struct city cityList[4]; |
|  |
| struct bus{ |
| int dir4;//world dirrection |
| int dir2;//dirrection on route |
| int x, y; |
| int tx, ty;//target coordinates |
| int passengers;//count |
| int speed; |
| int waitingTime; |
| int waitingFlag; |
| } b\_bus; |
|  |
|  |
| int bussesCount = 2;//must bee the same |
| struct bus bussesList[2];//SAAAAAAMEEEEE!!!! |
|  |
| void ProcBus(void \*\_arg) |
| { |
| int arg = (int)\_arg; |
| struct bus b = bussesList[arg]; |
| while(1) |
| { |
| if(b.x < b.tx) b.x += b.speed; //movement |
| if(b.y < b.ty) b.y += b.speed; |
| if(b.x > b.tx) b.x -= b.speed; |
| if(b.y > b.ty) b.y -= b.speed; |
| if(b.y == b.ty && b.x == b.tx) |
| { |
| b.waitingFlag = 1; |
| pthread\_mutex\_lock(&hmtx); |
| bussesList[arg] = b; |
| pthread\_mutex\_unlock(&hmtx); |
|  |
| usleep(b.waitingTime);//waits near the busStop |
|  |
| pthread\_mutex\_lock(&hmtx); |
| b = bussesList[arg]; |
| pthread\_mutex\_unlock(&hmtx); |
| b.waitingFlag = 0; |
| if(b.dir2 == 0) //forward or backwrd direction |
| { |
| if(b.dir4 != 3) b.dir4++;//clockwise |
| else b.dir4 = 0; |
| switch(b.dir4)//world dirrection |
| { |
| case 0: |
| { |
| b.tx = cityList[1].x - rThickness/4;//setting target points depending on direction |
| b.ty = cityList[1].y + rThickness/4; |
| break; |
| } |
| case 1: |
| { |
| b.tx = cityList[2].x - rThickness/4; |
| b.ty = cityList[2].y - rThickness/4; |
| break; |
| } |
| case 2: |
| { |
| b.tx = cityList[3].x + rThickness/4; |
| b.ty = cityList[3].y - rThickness/4; |
| break; |
| } |
| case 3: |
| { |
| b.tx = cityList[0].x + rThickness/4; |
| b.ty = cityList[0].y + rThickness/4; |
| break; |
| } |
| } |
| }else{ |
| if(b.dir4 != 0) b.dir4--;//counter-clockwise |
| else b.dir4 = 3; |
| switch(b.dir4) |
| { |
| case 0: |
| { |
| b.tx = cityList[2].x + rThickness/4; |
| b.ty = cityList[2].y + rThickness/4; |
| break; |
| } |
| case 1: |
| { |
| b.tx = cityList[3].x - rThickness/4; |
| b.ty = cityList[3].y + rThickness/4; |
| break; |
| } |
| case 2: |
| { |
| b.tx = cityList[0].x - rThickness/4; |
| b.ty = cityList[0].y - rThickness/4; |
| break; |
| } |
| case 3: |
| { |
| b.tx = cityList[1].x + rThickness/4; |
| b.ty = cityList[1].y - rThickness/4; |
| break; |
| } |
| } |
| } |
| } |
| pthread\_mutex\_lock(&hmtx); |
| bussesList[arg] = b;//update bus info |
| pthread\_mutex\_unlock(&hmtx); |
| usleep(1000000/50);// do it 50 times in second |
| } |
| } |
|  |
|  |
|  |
| struct passenger{ |
| //coordinates of passengers and some flags about passenger satate (what they doing) |
| int x, y, tx, ty, sleep, maxSleepTime, sleepTime, wait, drive, toStop, toCity, toBus, curCity, curDir, canLeave; |
| float fx,fy,fSpeed; |
| }; |
|  |
| struct passenger passengersList[512]; |
|  |
| void ProcPassenger(void \*\_arg) |
| { |
| int i = (int)\_arg; |
| struct passenger p; |
| p = passengersList[i]; |
|  |
| /\*int iSpeed = 40 + rand()%60; |
| p.fSpeed = (float)iSpeed/100.0f;\*/ |
| p.fSpeed = 1; |
|  |
| int rS = rand()%4;//selecting random city |
| p.curCity = rS; |
| p.x = cityList[rS].x - citySize/2 + rand()%citySize;//setting random start position |
| p.y = cityList[rS].y - citySize/2 + rand()%citySize; |
| p.fx = p.x; |
| p.fy = p.y; |
|  |
|  |
| int rD = rand()%2;//selecting random route direction |
| p.curDir = rD; |
| if(p.curDir == 0) |
| { |
| p.tx = cityList[p.curCity].stopFx - 8 + rand()%16;//setting random point on needed stop |
| p.ty = cityList[p.curCity].stopFy - 8 + rand()%16;//depending on selected dirrection |
| }else{ |
| p.tx = cityList[p.curCity].stopBx - 8 + rand()%16; |
| p.ty = cityList[p.curCity].stopBy - 8 + rand()%16; |
| } |
|  |
|  |
| p.sleepTime = p.maxSleepTime/2 + rand()%p.maxSleepTime/2;//setting random sleepInCity time |
| p.sleepTime = p.sleepTime \* 1000;// rand generates not bigger then ~32000, so its just a mult to get bigger number |
| while(1) |
| { |
| struct bus b; |
| b = bussesList[p.curDir]; |
|  |
|  |
| if(p.toStop == 1) |
| { |
| float vx = p.tx - p.fx;//setting moveing vector |
| float vy = p.ty - p.fy; |
| float vl = sqrt(vx\*vx + vy\*vy); |
| if(vl > 1) |
| { |
| vx /= vl; |
| vy /= vl; |
| } |
|  |
| p.fx += vx \* p.fSpeed;//replacing passenger; |
| p.fy += vy \* p.fSpeed; |
| p.x = p.fx; |
| p.y = p.fy; |
|  |
| if(p.x == p.tx && p.y == p.ty){// if comes to stop then wait |
| p.toStop = 0; |
| p.wait = 1; |
| } |
| } |
|  |
| if(p.wait) |
| { |
| if(abs(p.x - b.x) < 64 && abs(p.y - b.y) < 64 && b.waitingFlag == 1 && bussesList[p.curDir].passengers < 5)//if bus is close and it's waits |
| { |
| p.tx = b.x; p.ty = b.y;//start moving to bus |
| p.wait = 0; |
| p.toBus = 1; |
| } |
| } |
|  |
| if(p.toBus == 1) |
| { |
| if(b.waitingFlag == 1)//if buss waits |
| { |
| if(bussesList[p.curDir].passengers < 5) |
| { |
| float vx = p.tx - p.fx;//setting moveing vector |
| float vy = p.ty - p.fy; |
| float vl = sqrt(vx\*vx + vy\*vy); |
| if(vl > 1) |
| { |
| vx /= vl; |
| vy /= vl; |
| } |
| p.fx += vx \* p.fSpeed;//replacing passenger; |
| p.fy += vy \* p.fSpeed; |
| p.x = p.fx; |
| p.y = p.fy; |
| }else{ |
| if(p.curDir == 0)//returns to needed stop |
| { |
| p.tx = cityList[p.curCity].stopFx - 8 + rand()%16;//random position on busstop |
| p.ty = cityList[p.curCity].stopFy - 8 + rand()%16; |
| }else{ |
| p.tx = cityList[p.curCity].stopBx - 8 + rand()%16; |
| p.ty = cityList[p.curCity].stopBy - 8 + rand()%16; |
| } |
| p.toStop = 1; |
| p.toBus = 0; |
| } |
|  |
| if(p.x == p.tx && p.y == p.ty){//if comes to bus |
| if(bussesList[p.curDir].passengers < 5) |
| { |
| p.drive = 1;//then drive and can't leave in the same city |
| pthread\_mutex\_lock(&hmtx); |
| bussesList[p.curDir].passengers++; |
| pthread\_mutex\_unlock(&hmtx); |
| p.canLeave = 0; |
| p.toBus = 0; |
| }else{ |
| if(p.curDir == 0)//returns to needed stop |
| { |
| p.tx = cityList[p.curCity].stopFx - 8 + rand()%16;//random position on busstop |
| p.ty = cityList[p.curCity].stopFy - 8 + rand()%16; |
| }else{ |
| p.tx = cityList[p.curCity].stopBx - 8 + rand()%16; |
| p.ty = cityList[p.curCity].stopBy - 8 + rand()%16; |
| } |
| p.toStop = 1; |
| p.toBus = 0; |
| } |
| } |
| }else{//if bus not waiting |
| if(p.curDir == 0)//returns to needed stop |
| { |
| p.tx = cityList[p.curCity].stopFx - 8 + rand()%16;//random position on busstop |
| p.ty = cityList[p.curCity].stopFy - 8 + rand()%16; |
| }else{ |
| p.tx = cityList[p.curCity].stopBx - 8 + rand()%16; |
| p.ty = cityList[p.curCity].stopBy - 8 + rand()%16; |
| } |
| p.toStop = 1; |
| p.toBus = 0; |
| } |
| } |
|  |
| if(p.drive == 1) |
| { |
| p.x = b.x;//pos = bus pos |
| p.y = b.y; |
| p.fx = p.x; |
| p.fy = p.y; |
| if(b.waitingFlag == 0) p.canLeave = 1;//if bus not waits then can leave |
| if(b.waitingFlag == 1 && p.canLeave ==1)//(will leave only when bus stops |
| { |
| p.drive = 0; |
| p.toCity = 1; |
| if(bussesList[p.curDir].passengers > 0) |
| { |
| pthread\_mutex\_lock(&hmtx); |
| bussesList[p.curDir].passengers--; |
| pthread\_mutex\_unlock(&hmtx); |
| } |
| if(p.curDir == 0)//update info about curent city depending on curentDirrection |
| { |
| if(p.curCity != 3) p.curCity++; |
| else p.curCity = 0; |
| }else{ |
| if(p.curCity != 0) p.curCity--; |
| else p.curCity = 3; |
| } |
| p.tx = cityList[p.curCity].x - citySize/2 + rand()%citySize;//selecting random target point in curent city |
| p.ty = cityList[p.curCity].y - citySize/2 + rand()%citySize; |
| } |
| } |
|  |
| if(p.toCity) |
| { |
| float vx = p.tx - p.fx;//setting moveing vector |
| float vy = p.ty - p.fy; |
| float vl = sqrt(vx\*vx + vy\*vy); |
| if(vl > 1) |
| { |
| vx /= vl; |
| vy /= vl; |
| } |
|  |
| p.fx += vx \* p.fSpeed;//replacing passenger; |
| p.fy += vy \* p.fSpeed; |
| p.x = p.fx; |
| p.y = p.fy; |
|  |
| if(p.x == p.tx && p.y == p.ty){//if comes to target point then sleep some seconds |
| p.sleep = 1; |
| passengersList[i] = p; |
|  |
| usleep(p.sleepTime); |
|  |
| p.sleep = 0; |
|  |
| p.toCity = 0; |
| p.toStop = 1; |
| p.wait = 0; |
|  |
| rD = rand()%2;//select random direction (backwards or forwards) |
| p.curDir = rD; |
| if(p.curDir == 0) |
| { |
| p.tx = cityList[p.curCity].stopFx - 8 + rand()%16;//and selcting target point on stop |
| p.ty = cityList[p.curCity].stopFy - 8 + rand()%16; |
| }else{ |
| p.tx = cityList[p.curCity].stopBx - 8 + rand()%16; |
| p.ty = cityList[p.curCity].stopBy - 8 + rand()%16; |
| } |
| } |
| } |
|  |
|  |
|  |
| passengersList[i] = p; |
| usleep(1000000/50); |
|  |
| /\*int calcHardness = 1024;// //////////////////////////////MEGACALCULATION\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ |
| int res = 0; |
| for(int i = 1; i < calcHardness; i++) |
| { |
| for(int j = 1; j < calcHardness; j++) |
| { |
| res = i + j - i%j + j%i + i/j - j/i \* 7; |
| } |
| }\*/// \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\whill make your processor very very hot!!/////////////////////////////////// |
| } |
|  |
|  |
| } |
|  |
|  |
|  |
| void Draw(int sleepTime) |
| { |
| XClearWindow(dspl, hwnd); |
|  |
|  |
| char pc[3]; |
| sprintf(pc, "%d", passengersCount);//draws passenger coutn |
| XDrawString(dspl,hwnd, gc, 20, 20, pc, 3); |
|  |
| for(int i = 0; i < 4; i++)//draws cities |
| { |
| struct city c = cityList[i]; |
| XDrawRectangle(dspl, hwnd, gc, c.x - citySize/2, c.y - citySize/2, citySize, citySize); |
|  |
| char d[1]; |
| d[0] = 'f'; |
| XDrawRectangle(dspl, hwnd, gc, c.stopFx - 8, c.stopFy - 8, 16, 16); |
| XDrawString(dspl,hwnd, gc, c.stopFx, c.stopFy-8, d, 1); |
| d[0] = 'b'; |
| XDrawRectangle(dspl, hwnd, gc, c.stopBx - 8, c.stopBy - 8, 16, 16); |
| XDrawString(dspl,hwnd, gc, c.stopBx, c.stopBy-8, d, 1); |
|  |
|  |
| XDrawString(dspl,hwnd, gc, c.x+citySize/4, c.y - citySize/4, c.name, sizeof(c.name)); |
| } |
| //draw roads |
| XDrawRectangle(dspl, hwnd, gc, hPadding + citySize/2 - rThickness/2, vPadding + citySize/2 - rThickness/2, hDistance + rThickness, rThickness); |
| XDrawRectangle(dspl, hwnd, gc, hPadding + citySize/2 - rThickness/2, vPadding + citySize/2 - rThickness/2, rThickness, vDistance + rThickness); |
| XDrawRectangle(dspl, hwnd, gc, hPadding + citySize/2 - rThickness/2, vPadding + citySize/2 + vDistance - rThickness/2, hDistance + rThickness, rThickness); |
| XDrawRectangle(dspl, hwnd, gc, hPadding + citySize/2 - rThickness/2 + hDistance, vPadding + citySize/2 - rThickness/2, rThickness, vDistance + rThickness); |
|  |
|  |
| for(int i = 0; i < bussesCount; i++)//draw each bus |
| { |
| struct bus b = bussesList[i]; |
| if(b.dir4 == 0 || b.dir4 == 2)//depending on it's direction |
| { |
| XDrawRectangle(dspl, hwnd, gc, b.x - 8, b.y - 4, 16, 8);//draw vertycal or horizontal |
| }else |
| { |
| XDrawRectangle(dspl, hwnd, gc, b.x - 4, b.y - 8, 8, 16); |
| } |
| char cn[1]; |
| sprintf(cn, "%d", b.passengers); |
| XDrawString(dspl,hwnd, gc, b.x-2, b.y+4, cn, 1); |
| } |
|  |
| for(int i = 0; i < passengersCount; i++)//draw each passanger |
| { |
| struct passenger p = passengersList[i]; |
| if(p.sleep == 0 && p.drive == 0) XDrawRectangle(dspl, hwnd, gc, p.x-2, p.y-2, 4,4); |
| } |
| XFlush(dspl); |
| usleep(1000000/50); |
| } |
|  |
|  |
| void main() |
| { |
| struct city c1;//setting cityes positions |
| c1.x = hPadding + citySize/2; |
| c1.y = vPadding + citySize/2; |
| c1.stopFx = c1.x + rThickness; |
| c1.stopFy = c1.y + rThickness; |
| c1.stopBx = c1.x; |
| c1.stopBy = c1.y - rThickness; |
| strcpy(c1.name, "Dijon"); |
|  |
| struct city c2; |
| c2.x = hPadding + hDistance + citySize/2; |
| c2.y = vPadding + citySize/2; |
| c2.stopFx = c2.x - rThickness; |
| c2.stopFy = c2.y + rThickness; |
| c2.stopBx = c2.x + rThickness ; |
| c2.stopBy = c2.y; |
| strcpy(c2.name, "Ruan"); |
|  |
| struct city c3; |
| c3.x = hPadding + hDistance + citySize/2; |
| c3.y = vPadding + vDistance + citySize/2; |
| c3.stopFx = c3.x - rThickness; |
| c3.stopFy = c3.y - rThickness; |
| c3.stopBx = c3.x; |
| c3.stopBy = c3.y + rThickness; |
| strcpy(c3.name, "Paris"); |
|  |
| struct city c4; |
| c4.x = hPadding + citySize/2; |
| c4.y = vPadding + vDistance + citySize/2; |
| c4.stopFx = c4.x + rThickness; |
| c4.stopFy = c4.y - rThickness; |
| c4.stopBx = c4.x - rThickness; |
| c4.stopBy = c4.y; |
| strcpy(c4.name, "Lion"); |
|  |
| cityList[0] = c1; |
| cityList[1] = c2; |
| cityList[2] = c3; |
| cityList[3] = c4; |
|  |
|  |
|  |
| dspl = XOpenDisplay(NULL);//setting up new window |
| gc = XDefaultGC(dspl,0); |
| if(dspl == 0) {printf("Error XOpenDisplay\n"); exit(1);} |
| screen = XDefaultScreen(dspl); |
| hwnd = XCreateSimpleWindow(dspl, RootWindow(dspl, screen), 100,50,1280,720,3, BlackPixel(dspl,screen), WhitePixel(dspl,screen)); |
| if(hwnd == 0) {printf("Error XCreateSimpleWindow\n"); exit(1);} |
| XSelectInput(dspl, hwnd, ExposureMask | KeyPressMask );//input mask |
| XMapWindow(dspl, hwnd); |
|  |
|  |
| while(passengersCount == 0)//inputs char from keyboard while it more then -1 and less then 10 |
| { |
| char buf[]= "Enter number of passangers!"; |
| XDrawString(dspl, hwnd, gc, 10, 10, buf, 27); |
| XNextEvent(dspl, &event); |
| if(event.type == KeyPress) |
| { |
| XClearWindow(dspl,hwnd); |
| char kb[1]; |
| XLookupString(&event.xkey, kb,1, &ks, 0);//converts keyKode to char symbol |
| XDrawString(dspl, hwnd, gc, 10, 10, kb, 1); |
| char c = kb[0]; |
| int i = c - '0'; |
| if(i >= 0 && i < 10) |
| { |
| int r = 1; |
| for(int c = 0; c < i; c++ ) |
| { |
| r\*=2;//makes 2^inputedNumber to create more passangers |
| } |
| passengersCount = r; |
| } |
|  |
| } |
| } |
|  |
|  |
| struct bus busF;//creating two busses |
| busF.x = hPadding + citySize/2; |
| busF.y = vPadding + citySize/2 + rThickness/4; |
| busF.tx = hPadding + citySize/2 + hDistance - rThickness/4; |
| busF.ty = vPadding + citySize/2 + rThickness/4; |
| busF.speed = 4; |
| busF.dir2 = 0; |
| busF.passengers = 0; |
| busF.dir4 = 0; |
| busF.waitingTime = 2000000; |
| busF.waitingFlag = 0; |
| int rc; |
| bussesList[0] = busF; |
| pthread\_t busFthread; |
| pthread\_create(&busFthread, NULL, (void\*)ProcBus, (void\*)0);//starts thread with selected bus |
|  |
|  |
| struct bus busB; |
| busB.x = hPadding + citySize/2 - rThickness/4; |
| busB.y = vPadding + citySize/2; |
| busB.tx = hPadding + citySize/2 - rThickness/4; |
| busB.ty = vPadding + citySize/2 + vDistance + rThickness/4; |
| busB.dir2 = 1; |
| busB.speed = 2; |
| busB.passengers = 0; |
| busB.dir4 = 1; |
| busB.waitingTime = 4000000; |
| busB.waitingFlag = 0; |
| bussesList[1] = busB; |
| pthread\_t busBthread; |
| pthread\_create(&busBthread, NULL, (void\*)ProcBus, (void\*)1); |
|  |
|  |
| for(int i = 0; i < passengersCount; i++)//creats needed count of passengers |
| { |
| struct passenger p; |
| p.x=0; |
| p.y =0; |
| p.tx = 0; |
| p.ty = 0; |
| p.curCity = 0; |
| p.drive = 0; |
| p.sleep = 0; |
| p.maxSleepTime = 64000; |
| p.sleepTime = 0; |
| p.toCity = 0; |
| p.toStop = 1; |
| p.wait = 0; |
| passengersList[i] = p; |
| pthread\_t p1Thread; |
| pthread\_create(&p1Thread, NULL, (void\*)ProcPassenger, (void\*)i);//starts thread with passenger |
| } |
|  |
| while(1) |
| { |
| Draw(1000000/50);//bigger number - lower FPS |
| } |
|  |
| getchar(); printf("\033[0m"); |
|  |