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 программная модель посещения отделения Сбербанка |

Пояснительная записка

|  |  |
| --- | --- |
| Шифр проекта (работы) | КП-02…. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Студента (ки) | | | | Бабанов Артем Леонидович | | | | | | |
|  |  |  |  | | | | фамилия, имя, отчество полностью | | | | | | |
|  |  |  | Курс | 2 |  | | | Группа | | ИВТ-183 | | |  |
|  |  |  |  |  | |  | | |  |  | | |  |
|  |  |  | 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 2020

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 программная модель посещения отделения Сбербанка | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | |
| Студент (ка) | | | | Бабанов Артем Леонидович | | | | | | | | | | | | |
|  | | | | фамилия, имя, отчество полностью | | | | | | | | | | | | |
| Курс | 2 | | | | Группа | | | ИВТ-183 |  | | | | | | | |
|  | |  | | |  | | |  |  | | | | | | | |
| 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 29с., 4 ч., 8 рис., 3 источ.,

1 прил.

C, LINUX, GCC, КОНСОЛЬ, МНОГОПОТОЧНОЕ ПРИЛОЖЕНИЕ.

Объектом исследования является алгоритм взаимодействия нескольких потоков в операционной системе Linux.

Цель работы – разработка многопоточной программной модели посещения отделения Сбербанка в операционной системе Linux.

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

В результате была получена программа, которая демонстрирует модель посещения отделения сбербанка множеством людей одновременно.

Содержание

[Введение 1](#_Toc39866417)

[1 Введение в проблематику разработки многопоточных приложений 2](#_Toc39866418)

[2 Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения 4](#_Toc39866419)

[3 Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения 6](#_Toc39866420)

[4 Детальное текстовое описание на основе сочетания естественного языка и программных конструкций алгоритмов всех процедур 7](#_Toc39866421)

[Заключение 17](#_Toc39866422)

[Список использованных источников 18](#_Toc39866423)

[Приложение А 19](#_Toc39866424)

# Введение

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

Задача: Разработать — в визуальном представлении на консоли Linux — многопоточную модель поведения посетителей отделения Сбербанка. Моделируется поведение 20 посетителей, каждый из которых независимо от других входит в отделение, подходит к автомату выдачи талонов на обслуживание, в модели считается, что очередь к этому автомату фактически не формируется, он сразу же получает талон с номером. Затем ждет, пока не появиться объявление о его талоне и окне, к которому нужно подойти. Таких окон в модели должно быть 5. Подойдя к окну он обслуживается некоторой время, определяемое типом операции, заданной в талоне. Если получение наличных, то 3 сек., если другая операция, то 5 сек. После чего отходит и выходит из помещения. Появление посетителей моделируется как случайное в интервале времени модели от 1 до 6 секунд. Поведение модели должно отображаться в консольном окне с помощью условных схем, значков и текстов, размер окна не менее 40 строк на 100 столбцов, занимая в ходе моделирующих движений все пространство окна. Отдельный посетитель обозначается одной латинской буквой. Для правильного взаимодействия использовать семафоры или мьютексы.

Проект состоит из четырех разделов:

1. Введение в проблематику разработки многопоточных приложений
2. Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения
3. Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения
4. 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.

В современных операционных системах широко используются нити (threads), 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.

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. Именно эта задача и составляет основную сложность работы с многопоточными приложениями.

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

Основой любого консольного приложения являются символы. Так как работать с символами придется на протяжении всего процесса создания приложения, я сразу же описал простую функцию для вывода строки по координатам (PrintInPoint()). После чего решил, что весь интерфейс будет состоять из квадратов, рамки которых нужно рисовать повсеместно, для чего была написана простая функция (DrawBorder()). Соответственно, имея уже эти две функции можно нарисовать весь интерфейс в консоли. Но, чтобы изображение было более красивое, нужно менять у некоторых элементов цвет, для чего была описана функция ChangeColor() и три директивы с кодами цветов.

Далее была описана функция DrawField(), которая содержит в себе вызовы других функций – для отрисовки «здания», окон, автомата, сидений и таблицы посетителей. Все они, в сущности, содержат три шага. 1 (опциальный) – подготовка данных для вывода. 2 – определение цвета. И 3 – вывод в консоль. Естественно, данные разные и используются функции, описанные в самом начале. В сущности, отрисовка поля выполняется при запуске программы в точке входа и при смене координат в функции для перемещения посетителя.

Собственно, функция перемещения посетителя. Это тоже одно из основополагающих действий в программе. В конечном итоге все сводится к тому, что, зная 2 точки, «имя» посетителя и режим (как сначала идти – по вертикали или по горизонтали), программа просто стирает «имя» в предыдущем месте, и пишет его в новом. Сначала делает это для одной оси, затем для другой. Алгоритм простой, но не идеальный, поэтому, чтобы восстановить стертые части поля, оно отрисовывается каждый раз при смене оси движения.

Соответственно, когда все основные функции отрисовки готовы, можно приступить к функционалу самого посетителя. Посетителем является поток, функция которого описана в visitor(). Для его инициализации необходимо имя посетителя, которое, вместе с запуском потоков в количестве 20-ти штук, создается по очереди, от A до T. Собственно, посетитель может ходить в определенные точки, но нужно чтобы делал он это с умом. Для этого сделаны 2 однотипных блока функций и переменных – windows и seats – окна и сиденья. Подразумевается, что, если все окна заняты, то посетитель идет садится на не занятое сиденье, и ждет там, пока окно не освободится. Иначе, если хоть одно окно свободно, то он идет в это окно. Как уже было сказано, сиденья и окна – однотипные модули, и для них обоих были написаны две функции – для чтения и записи, используя мьютексы, чтобы не допустить одновременного чтения одних и тех же данных у двух и более посетителей. Более подробно алгоритм посетителя будет описан далее.

# 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:

int numberOfTalons – количество талонов – для определения и выдачи нового талона посетителю.

int windows[5] – массив состояний окон. Если 0 – свободно, если 1 – занято.

int seats[8] – массив состояний сидений. Если 0 – свободно, если 1 – занято.

int fieldWidth – ширина поля, «здания»

int fieldHeight – высота поля, длина «здания»

int freeWindow – пустое окно. Буфер, нужный для передачи данных между функцией и алгоритмом, определяющим пустое окно (читатель).

int freeSeat - пустое сидение. Буфер, нужный для передачи данных между функцией и алгоритмом, определяющим пустое окно (читатель).

pthread\_mutex\_t mtxWinWrite,mtxWinRead – мьютексы для работы с чтением и записью в массив окон.

pthread\_mutex\_t mtxSeatWrite, mtxSeatRead – мьютексы для работы с чтением и записью в массив сидений.

pthread\_t tid[20] - идентификаторы нитей посетителей.

Для синхронизации потоков используются мьютексы. Для инициализации мьютексов применяется функция pthread\_mutex\_init, для блокировки и разблокировки функции pthread\_mutex\_lock и pthread\_mutex\_unlock соответственно. В первой функции атрибутами являются указатель на мьютекс и атрибуты мьютекса. В последних двух функциях в атрибутах используется только указатель на мьютекс.

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

Отталкиваясь от описания, данного в 2 пункте, нужно описать более подробно большую часть функций.

Начнем с малого. Функция PrintInPoint. Это упрощение, сделанное с целью упростить процесс прочтения кода. Это функция, которая принимает координаты в консоли в виде двух int-ов и строку в виде указателя на char. И выводит в консоль управляющую последовательность \033 с символами, которые принимает функция – координаты и данные, которые выводятся после управляющей последовательности. Таким же образом работает и ChangeColor(), но принимает значение цвета текста (фон по умолчанию прозрачный).

Немаловажный алгоритм для отрисовки границ прямоугольников устроен следующим образом. Функция DrawBorder() принимает 4 целых значения – координаты первой и второй, диагональной первой, точек. После чего, используя символы, совокупность которых принято называть псевдографикой, выводятся углы прямоугольника, и потом, оставшиеся стороны рисуются уже другими символами – вертикальных и горизонтальных черт, которые выводятся в циклах. Схема алгоритма представлена на рисунке 1.



Рисунок 1 – схема алгоритма функции DrawBorder()

Вся дальнейшая отрисовка будет использовать эти функции. «Центром» всей отрисовки поля является функция DrawField(), которая запускает последовательно 5 других: DrawFieldBorder() – отрисовка границ поля, DrawWindowsState() - окон, DrawTalonMachine() – автомата с талонами, DrawSeats() - сидений, DrawTable() – таблицы. В сущности, все эти функции просты. Первая – использует DrawBorder() с координатами левого верхнего угла консоли и точки, содержащей длину и ширину границ здания. Вторая – делает небольшие расчеты ширины окна, чтобы в любой размер границы здания, который кратен 1 + 5n, можно было вписать 5 окон. Далее, в цикле для 5-ти окон, координаты которых высчитываются исходя из ширины окна и номера окна, выводит текстовую информацию об окне и границу окна с помощью DrawBorder(). После чего, выводя два других символа из псевдографики, границы между окнами сглаживаются (если это не первое окно).

Третья функция просто выводит посередине, с некоторым сдвигом от центра по вертикали, прямоугольник с помощью DrawBorder() и текст «Талоны». Четвертая – выводит, опять же, используя символы псевдографики, 8 сидений – 4 справа и 4 слева. И пятая – с помощью DrawBorder(), сразу после границы здания, выводит шапку и основу таблицы. И текстовое содержимое (номер талона, имя, операцию и окно) следом.

Чтобы в дальнейшем не отвлекаться на простые функции вывода, стоит описать еще две функции, используемые для работы с таблицей. Для записи строки в таблицу используется функция WriteInTable(), которая принимает «имя» в виде строки, талон, операцию и окно в целочисленном виде, которые потом преобразуются в вид строки (указателя на char) через выделение памяти malloc(), и выводит значения в строку с номером талона. А обратную операцию производит функция ClearRow(), которая попросту пишет 20 пробелов в строку с номером талона, который она принимает в качестве аргумента.

Так же, одна из основных функций отрисовки – GoFromTo(). Задача ее – перемещать посетителя из точки А в точку Б. Собственно, функция принимает имя пользователя в виде строки, координаты двух точек в виде целых чисел, и способ – как перемещаться – сначала по оси х или сначала по оси у. Переменная how, которая это определяет, либо кратная двум, либо нет. Соответственно, если кратное – то сначала нужно идти по оси х. Это происходит следующим образом: определяется в какую сторону идти (сугубо, добавлять к переменной или вычитать из нее) и что является конечной точкой (это определяется уже исходя из того, где пользователь находится – прошел ли он уже ось у или нет). После чего, просто перемещаем букву имени пользователя, меняя координаты, и затирая пробелом предыдущую точку. Аналогично и для оси у. После перемещения отрисовывается поле заново, чтобы скрыть дефекты, которые нанесло перемещение. Схема алгоритма представлена на рисунке 2.



Рисунок 2 – схема алгоритма GoFromTo

Пожалуй, главный алгоритм в логической части программы – это алгоритм посетителя – функция visitor(). При запуске ее из точки входа приложения, в нее передается имя, которое будет потом передаваться в другие функции для отрисовки. В общем случае, алгоритм можно поделить на 3 части: 1 – прийти к автомату и записать данные о талоне, операции и окне, 2 – если окна заняты – то идти на сиденья и ждать, пока освободится, 3 – если есть пустое окно, то идти в него, постоять там, и уйти из отделения. Схема алгоритма представлена на рисунке 3.



Рисунок 3 – схема алгоритма функции visitor()

Рассмотрим более подробно этот алгоритм. После преобразования данных о имени и ожидания некоторого времени, перед тем как войти, пользователь перемещается с помощью функции GoFromTo() к автомату с талонами, определяет операцию через рандомайзер(остаток от деления на 2 дает либо 0 – снятие наличных, либо 1 – другие операции) ,«берет» талон с помощью функции GetTalon(), которая возвращает номер незанятого талона и прибавляет к количестве талонов единицу. Дальше происходит чтение из массива окон информации о свободном окне используя средства синхронизации потоков. На этом стоит остановится, и описать процесс подробно.

Так как, процессы чтения и записи для окон и сидений однотипные, то, чтобы не создавать несколько одинаковых функций, были описаны две – одна для чтения, принимающая все параметры, требуемые для чтения (мьютекс для чтения, мьютекс для записи, функция – читатель, и «режим»), и вторая для записи, которая требует мьютекс для записи и функцию писатель. Все функции для работы с массивами windows[] и seats[] однотипны и отличаются лишь названием и данными, с которыми они работают. Так функции GetWindow() и GetSeat() – читатели (перебор элементов и сравнение с 0) с массивов windows[] и seats[]. SetWindow() и SetSeat() – писатели(пишет, в ячейку с номером буфера, 1) для них. Так же есть еще FreeAWindow() и FreeASeat(), которые нужны для «освобождения» окна или сиденья – записи 0 в нужную ячейку, которая является аргументом функции. Все это, в купе с двумя абстрактными функциями для записи и чтения, позволяет создать, как бы, два однотипных модуля. Соответственно, при вызове этих абстрактных функций будут меняться только указатели на переменные, связанные с конкретным модулем – окон или сидений. Дальше функции будем рассматривать на примере работы с модулем окон, т.к. разницы почти нет, но в «окнах» есть использование «режима» работы.

Итак, функция читатель. Если несколько потоков начнут одновременно читать одни и те же данные с массива, а другие потоки еще и в добавок будут записывать в этот массив информацию, то мы никогда не сможем сказать точно, что за информация находится в массиве, следовательно, и не сможем с ней работать. Для этого, для синхронизации работы нескольких потоков, используются мьютексы (mutexs), которые позволяют разграничить доступ к данным для нескольких потоков. В случае с конкретной задачей, необходимо еще не допустить, чтобы не только писатели и читатели не работали одновременно, но, и чтобы не читались одни и те же данные (хоть это и не меняет их). Потому что, в противном случае, два посетителя увидят, что освободилось какое-то окно и пойдут в него вместе. Соответственно, в цикле функция пытается заблокировать мьютекс по чтению. Если не получается – пытается еще раз. Если вышло заблокировать, то мы убеждаемся в том, что читать далее сможет только один поток. Что дальше и происходит. Блокируется мьютекс по записи (чтобы не допустить изменения данных перед тем, как читать), собственно чтение функцией читателем (функция передается как указатель на нее, и в случае с окнами – это GetWindow()). После чтения, использованные мьютексы разблокируются в виду того, что данные прочитаны, и, если не нужно ожидать, пока освободится окно, выходит из цикла, и возвращается в основную программу. Схема алгоритма этой функции представлена на рисунке 4.



Рис. 4 – схема алгоритма AbstractReader() Рис. 5 – схема алгоритма AbstractWriter()

После того, как информация о свободном окне была записана в буфер, функция посетитель эту информацию записывает в локальную переменную window, и после функция писатель должна изменить информацию в массиве окон для этого окна (информация об свободном окне берется из буфера).

Соответственно, функция писатель. По аналогии с абстрактным читателем, абстрактный писатель так же через указатель на конкретную функцию (в случае с окнами – SetWindow()) получает функцию для записи данных. После, аналогично, начинает цикл, в котором пытается заблокировать мьютекс на запись в этот массив. В случае, если успешно заблокировался мьютекс (то есть записывает только этот поток), используется функция писатель, меняется флаг для выхода из цикла и разблокируется мютекс по записи. Схема алгоритма представлена на рисунке 5.

В итоге, определив окно, записав его в локальную переменную, и определив его как занятое, посетитель некоторое время еще стоит у автомата с талонами, и если нашлось хоть одно свободное окно (window не равен 0), то, пересчитав координаты окна, идем к этому окну. Далее запускается другая подпрограмма – ChangeWindowTime(), которая приостанавливает поток на некоторое количество секунд, которое определяется операцией. Передав в нее численные значения окна, операции и талона, функция меняет номер талона у окна, и раз в секунду обновляет время, которое прошло с тех пор, как посетитель подошел к этому окну. Далее, освобождаем окно с помощью функции FreeAWindow(), которая была упомянута выше, очищаем строку с помощью ClearRow(), передав в нее номер талона, и идем к выходу в два шага. Первый – до линии дверей, чуть не доходя до автомата, и второй – до самих дверей. Дальше, из-за особенностей функции GoFromTo(), посетитель 200 миллисекунд ожидает в дверях и потом окончательно стирается пробелом.

А в случае, если свободного окна не было, тогда, с помощью абстрактного читателя, и передачи ему переменных, связанных с блоком сидений, и операцией чтения (мьютексы и читатель для сидений), мы читаем информацию о свободном сидении в буфер, из буфера информацию забираем в локальную переменную seat, и записываем с помощью абстрактного писателя это сидение как занятое. И, просчитав координаты определенного сиденья (учитывая и его сторону, справа те, чьи номера больше 4), идем к нему. Сев на это сиденье, посетитель, передав в абстрактный читатель указатели на мьютексы чтения, записи, и функцию чтения, а также режим, равный единице, ожидает, пока освободится окно. После того, как оно освободилось, информация об освободившемся окне записывается из буфера в локальную переменную window, и через функцию абстрактного писателя, записывается, что данное окно теперь занято. Далее, освобождается сиденье, используя функцию FreeASeat(), и записывается в таблицу обновленная информация о имени, талоне, операции и окне. И функция возвращается в точку, если бы окно свободное было и изначально.

Описав почти все функции, стоит вернуться к тому, откуда все начинается – к точке входа приложения – функции main(). В ней сначала очищается консоль, с помощью вывода нужной управляющей последовательности. После отрисовывается изначальное поле – DrawField(). А так же происходит инициализация мютексов в функции MutexsInit().

Далее, используя особенности языка С, в цикле for для 20-ти букв от А до T, (особенность заключается в том, что к char можно добавить численное значение и получить определенный символ. В этом случае – следующий, если добавить единицу к предыдущему) создается поток посетителя с именем в качестве аргумента, на котором и построен цикл. Далее приостанавливается поток.

После того, как все потоки посетителей выполнятся, пользователь может нажать на Ctrl+C или на любую клавишу, подтвердив ввод кнопкой «Enter», мютексы записи и чтения для окон и сидений уничтожаются, что сгруппировано в функции MutexsDestroy().

Скриншоты выполнения программы представлено на рисунках 6-8.

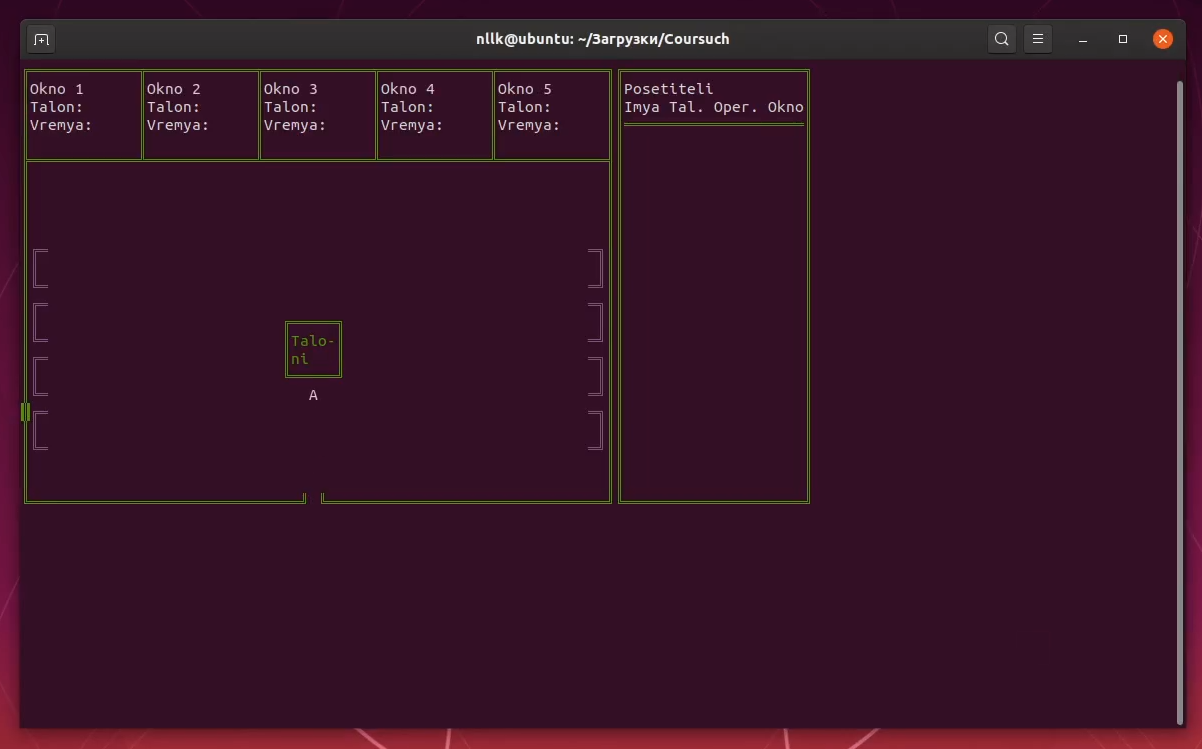


Рисунок 6 – выполнение программы

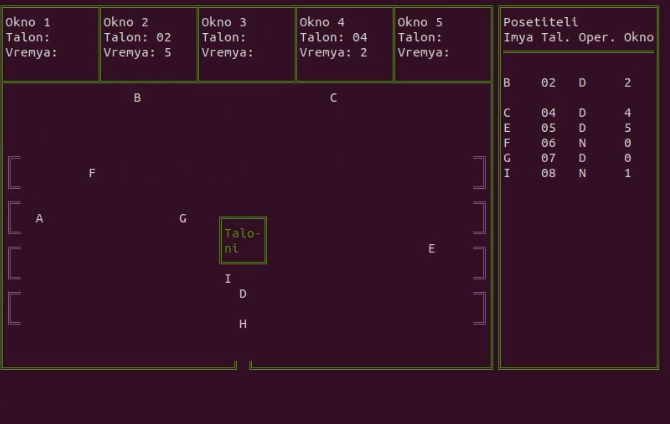


Рисунок 7 – выполнение программы

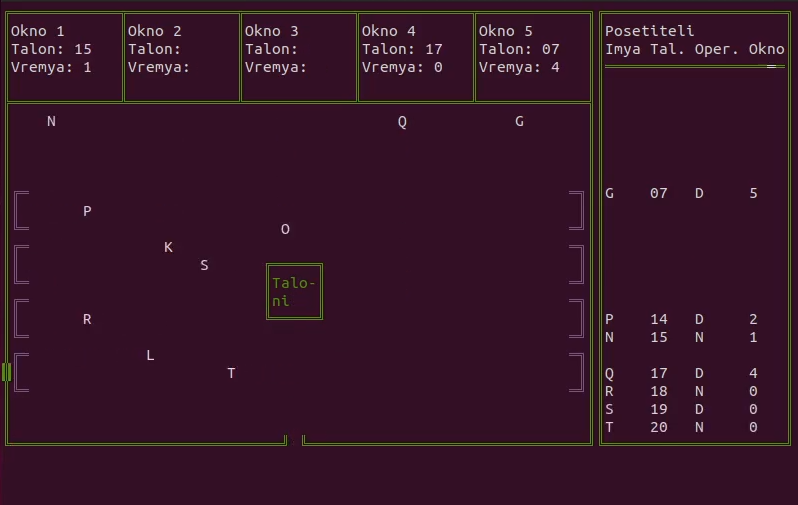


Рисунок 8 – выполнение программы

# Заключение

В результате работы над проектом был реализован алгоритм взаимодействия нескольких потоков при работе с консолью в операционной системе Linux в программе, написанной на языке программирования C. Программа демонстрирует модель посещения отделения Сбербанка.

# Список использованных источников

1. Флоренсов А. Н. Операционные системы для программиста; Учеб. пособие. Омск: Изд-во ОмГТУ, 2005. -240 с

2. Флоренсов А. Н. Методические указания по лабораторным работам дисциплины «Операционные системы», Омск: Изд-во ОмГТУ, 2019. -45 с

3. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

# Приложение А

Исходный код программы

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <unistd.h>

#include <semaphore.h>

#include <stdbool.h>

#define COLOR\_Green 32

#define COLOR\_White 37

#define COLOR\_Crimson 35

int numberOfTalons=0;//кол-во талонов в системе

int windows[5]={0,0,0,0,0};//состояние окон

int seats[8]={0,0,0,0,0,0,0,0};//состояние сидений

int fieldWidth=66;//ширина, 36+5\*width =>36,41,46...

int fieldHeight=25;//высота

int freeWindow;//пустое окно

pthread\_mutex\_t mtxWinWrite,mtxWinRead;//мютексы для записи и чтения с windows[]

int freeSeat;//пустое кресло

pthread\_mutex\_t mtxSeatWrite, mtxSeatRead;//мютексы для записи и чтения с seats[]

pthread\_t tid[20];//нити посетителей

void PrintInPoint(int x, int y, char\* string)

{//печать в строку. требует координаты х,у, и строку

printf("\033[%d;%dH%s\n",y,x, string);

}

void ChangeColor(int color)

{//смена цвета, требует числовое значение цвета

printf("\033[0;%d;48m\n",color);

}

void DrawBorder(int x1, int y1, int x2, int y2)

{//печать рамки. Требует координаты двух диоганальных точек

//x,y ╔══╗ X,y

// ║ ║

//x,Y ╚══╝ X,Y

ChangeColor(COLOR\_Green); PrintInPoint(x1,y1,"╔");//угол 1

ChangeColor(COLOR\_Green); PrintInPoint(x2,y1,"╗");//угол 2

ChangeColor(COLOR\_Green); PrintInPoint(x1,y2,"╚");//угол 3

ChangeColor(COLOR\_Green); PrintInPoint(x2,y2,"╝");//угол 4

if ((x2-x1 != 1) && (y2 - y1 != 1))//в случае, если это квадрат 1\*1 то все уже было выведено

{

for (int x=x1+1;x<=x2-1;x++)//верхняя и нижняя границы

{ ChangeColor(COLOR\_Green); PrintInPoint(x,y1,"═");

ChangeColor(COLOR\_Green); PrintInPoint(x,y2,"═");

}

for(int y=y1+1; y<=y2-1;y++)//правая и левая границы

{ ChangeColor(COLOR\_Green); PrintInPoint(x1, y,"║");

ChangeColor(COLOR\_Green); PrintInPoint(x2, y,"║");

}

}

}

void DrawFieldBorder()

{//рисовать рамку поля - здания

ChangeColor(COLOR\_Green); DrawBorder(1,1,fieldWidth,fieldHeight);//рамка поля - здания

ChangeColor(COLOR\_Green); PrintInPoint(fieldWidth/2-1,fieldHeight,"╝ ╚");//двери

}

void DrawSeats()

{//рисовать сиденья

int x = 2;//х первого сиденья

int y = 11;//у первого сиденья

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

{//сиденья справа и слева

ChangeColor(COLOR\_Crimson); PrintInPoint(x,y+n\*3, "╔═");

ChangeColor(COLOR\_Crimson); PrintInPoint(x,y+1+n\*3,"║");

ChangeColor(COLOR\_Crimson); PrintInPoint(x,y+2+n\*3,"╚═");

ChangeColor(COLOR\_Crimson); PrintInPoint(x+fieldWidth-2\*x,y+n\*3, "═╗");

ChangeColor(COLOR\_Crimson); PrintInPoint(x+fieldWidth-2\*x+1,y+1+n\*3,"║");

ChangeColor(COLOR\_Crimson); PrintInPoint(x+fieldWidth-2\*x,y+2+n\*3, "═╝");

}

}

void DrawWindowsState()

{//рисовать окна

int width=2+(fieldWidth-6)/5;//ширина

int height=6;//высота

for (int count=0; count<=4;count++)//для 5-ти окон

{

char str[]="Okno 1";//заготовка под строку "окно"

str[5]=count+1+'0';//меняем цифру на нужную

ChangeColor(COLOR\_White); PrintInPoint(count\*(width-1)+2, 2, str);

ChangeColor(COLOR\_White); PrintInPoint(count\*(width-1)+2, 3, "Talon:");

ChangeColor(COLOR\_White); PrintInPoint(count\*(width-1)+2, 4, "Vremya:");

ChangeColor(COLOR\_Green); DrawBorder(1+count\*(width-1),1,width+count\*(width-1),height);//отрисовка окна

if (count>0)

{ChangeColor(COLOR\_Green);PrintInPoint(1+count\*(width-1),1,"╦");//сгладить линии между окнами сверху

ChangeColor(COLOR\_Green); PrintInPoint(1+count\*(width-1),height,"╩");//сгладить линии между окнами снизу

}

}

ChangeColor(COLOR\_Green); PrintInPoint(1,height,"╠");//сгладить линии между окном и рамкой слева

ChangeColor(COLOR\_Green); PrintInPoint(fieldWidth,height,"╣");//сгладить линии между окном и рамкой справа

}

void DrawTalonMachine()

{//рисовать автомат по выдаче талонов

int width=7;//ширина автомата

int height=4;//высота

int shiftH=5;//сдвиг от центра экрана

int x1 = fieldWidth/2 - width/2;//начальные координаты

int y1 = fieldHeight/2 - height/2;

int x2 = fieldWidth/2 + width/2;//конечные координаты

int y2 = fieldHeight/2 + height/2-1;

DrawBorder(x1,y1+shiftH,x2,y2+shiftH);//вывод коробки

ChangeColor(COLOR\_Green); PrintInPoint(x1+1,y1+1+shiftH,"Talo-");//и текста

ChangeColor(COLOR\_Green); PrintInPoint(x1+1,y1+2+shiftH,"ni");

}

void DrawTable()

{//отрисовка таблицы

int width = 21;//ширина таблицы

int stX = fieldWidth+1;//стартовая точка таблицы

DrawBorder(stX,1,stX+width,4);//основа таблицы

DrawBorder(stX,1,stX+width,fieldHeight);//шапка

ChangeColor(COLOR\_White); PrintInPoint(stX+1,2,"Posetiteli");//вывод информации в шапку

ChangeColor(COLOR\_White); PrintInPoint(stX+1,3,"Imya");

ChangeColor(COLOR\_White); PrintInPoint(stX+1+5,3,"Tal.");

ChangeColor(COLOR\_White); PrintInPoint(stX+1+5+5,3, "Oper.");

ChangeColor(COLOR\_White); PrintInPoint(stX+1+5+5+6,3, "Okno");

}

void WriteInTable(char\* name, int talon, int operation, int window)

{//писать в таблицу информацию. Требует имя, талон, операцию и окно

int stX = fieldWidth+1;//начальная точка таблицы

PrintInPoint(stX+1,4+talon, name);//вывод имени

char\* tal=malloc(sizeof(char)\*3);//подготовка строки с талоном

tal[0]=talon/10+'0'; tal[1]=talon%10+'0';tal[2]='\0';

char\* op=malloc(sizeof(char)\*2);//подготовка строки с операцией

op[0]= (operation==0)?'N':'D';//если 0 - снятие наличных, если 1 - другое

op[1]='\0';

char\* win=malloc(sizeof(char)\*2);//подготовка строки с окном

win[0]=window+'0'; win[1]='\0';

ChangeColor(COLOR\_White); PrintInPoint(stX+1+5,4+talon,tal);//вывод талона

ChangeColor(COLOR\_White); PrintInPoint(stX+1+5+5,4+talon, op);//операции

ChangeColor(COLOR\_White); PrintInPoint(stX+1+5+5+6,4+talon, win);//окна

}

void ClearRow(int talon)

{//очистка строки в таблице. Требует номер талона

int stX = fieldWidth+1;//стартовые координаты таблицы

for (int i=0;i<20;i++)//вывод двадцати пробелов в нужной строке

PrintInPoint(stX+1+i,4+talon," ");

}

void DrawField()

{//рисовать поле

DrawFieldBorder();//границы - здание

DrawWindowsState();//окна

DrawTalonMachine();//автомат по выдаче талонов

DrawSeats();//сиденья

DrawTable();//таблица

}

int GetTimeForOperation(int op)

{//возвращает время для операции - 3 секунды для снятия наличных и 5 для других операций

return (op==0)? 3: 5 ;

}

void ChangeWindowTime(int window, int operation, int talon)

{//меняет время и номер талона, когда посетитель подходит к окну

char\* tal=malloc(sizeof(char)\*3);//переводим номер талона в строку

tal[0]=talon/10+'0'; tal[1]=talon%10+'0';tal[2]='\0';

int width=2+(fieldWidth-6)/5;//ширина окна

int time = GetTimeForOperation(operation);//время для операции

PrintInPoint((window-1)\*(width-1)+9,3,tal);//вывод талона

for (char t='0'; t<='0'+time; t++)//для отсчета времени - от 0 до t

{ChangeColor(COLOR\_White);PrintInPoint((window-1)\*(width-1)+10,4,&t);//вывод секунды

usleep(1000000);//секунду спит

}

PrintInPoint((window-1)\*(width-1)+10,4," ");//очистка

PrintInPoint((window-1)\*(width-1)+9,3, " ");

}

int GetTalon()

{//получить талон

numberOfTalons++;//колво талонов всего

return numberOfTalons;//возвращает новый талон

}

void GetWindow()

{//получить свободное окно

freeWindow=0;//свободное окно

for(int i=0; i<5;i++)//перебираем все окна

{ if (windows[i]==0)//если окно свободное - вывод его в "свободное окно"

{ freeWindow=i+1;

break;

}

}

}

void SetWindow()

{//установитиьт окно как более не свободное

windows[freeWindow-1]=1;//занимаем окно единицей

}

void FreeAWindow(int window)

{//освободить окною Требует номер окна

windows[window-1]=0;//освобождаем окно нулем

}

void GetSeat()

{//получить свободное кресло

freeSeat = 0;//свободное кресло

for (int i = 0; i<8;i++)//для всех кресел

{ if (seats[i] == 0)//если кресло свободное - вывод его в "свободное кресло"

{ freeSeat = i+1;

return;

}

}

}

void SetSeat()

{//установить кресло как более не свободное

seats[freeSeat-1]=1;//занимаем кресло единицей

}

void FreeASeat(int seat)

{//освободить кресло. Требует номер кресла

seats[seat-1]=0;//освобождаем кресло нулем

}

void GoFromTo(char\* name, int x1, int y1, int x2, int y2, int how)

{//перемещение из точки в точку

//how - задает как идти к точке

//четное - по у потом по х. иначе - наоборот

PrintInPoint(x1,y1, " ");

int sign;//знак - в какую строну идем

int count=0;//вспомогательная переменная для how. определяет по какой координате идти и куда уже шли

for (int i=how; i<=how+1;i++)

{

int from=0;//откуда (по х и по у)

if (i%2==0)//если четное - то сначала по у идем

{ //идти по y

sign = (y2>=y1)?1:-1;//задаем в какую сторону

from = (count==0)?x1:x2;//и куда

for (int y=y1; y!=y2; y+=sign)

{

if(y!=y1)PrintInPoint(from, y-sign, " ");//затираем предыдущую точку

ChangeColor(COLOR\_White); PrintInPoint(from, y,name);//пишем в точке имя

usleep(200000);//ждем немного

}

if (y1!=y2) PrintInPoint(from, y2-sign, " ");//затираем предпоследнюю точку

}

else//иначе - по х

{ //идти по х

sign = (x2>=x1)?1:-1;//задаем в какую сторону

from = (count==0)?y1:y2;//и куда

for (int x = x1; x!=x2; x+=sign)

{

if (x!=x1)PrintInPoint(x-sign, from, " ");//затираем предыдущую точку

ChangeColor(COLOR\_White); PrintInPoint(x, from,name);//пишем в точке имя

usleep(200000);//ждем немного

}

if (x1!=x2)PrintInPoint(x2-sign, from, " ");//затираем предпоследнюю точку

}

count++;

DrawField();//отрисовка поля, чтобы восстановить окружение

}

ChangeColor(COLOR\_White); PrintInPoint(x2,y2,name);//вывод в конце концов имени

}

void AbstractReader(void\* mtxRead, void\* mtxWrite, void\* reader, int mode)

{//абстрактный читатель. Для окон и сидений. Требует мютексы чтения, записи, семафор для чтения, функцию читатель, и режим

//режим задает, нужно ли выходить из цикла, если получили любое значение - при 0, и при 1 - выйти, если не 0

void (\*program)() = reader;//получаем функцию

bool ex = false;//переменная для опеределения выхода

do{

int rc = pthread\_mutex\_trylock(mtxRead);//по возможности блокируем доступ по чтению

if (rc == 0)

{

pthread\_mutex\_lock(mtxWrite);//блокируем запись

program();//читаем

pthread\_mutex\_unlock(mtxWrite);//разблокируем запись

pthread\_mutex\_unlock(mtxRead);//разблокируем чтение

if ((mode == 1) && (freeWindow == 0)) ex=false;//на случай, если нужно ждать не нулевого значения

else ex=true;

}

}

while (ex == false);//пока не прочитаем

}

void AbstractWriter(void\* mtxWrite, void\* writer)

{//абстрактный писатель. Для окон и сидений. Требует мютекс для записи и функцию писатель

void (\*program)() = writer;//получаем функцию писатель

bool ex = false;//переменная для определения выхода

do

{

int rc = pthread\_mutex\_trylock(mtxWrite);//пытаемся заблокировать мютекс по записи

if (rc==0)//ждем, пока будет возможность записать

{

program();//записываем

ex=true;//собираемся выходить

pthread\_mutex\_unlock(mtxWrite);//разблокируем мютекс по записи

}

}while (ex==false);

}

void\* visitor(void \*arg)

{//посетитель. Требует имя пользователя в виде символа

char received = (char)arg;//получение имени из пришедшей информации

char\* name=malloc(sizeof(char));

name[0]=received;

usleep((random()%7)\*1000000);//спим некоторое время перед тем как войти

//идем

int hereX=33, hereY=25;//из дверей

GoFromTo(name,33,25,33,19,0);//к автомату

hereX = 33; hereY=19;//теперь мы у автомата

int operation = random()%2;//опеределяем операцию, ради которой пришли

int talon = GetTalon();//получили талон

int window = 0;

AbstractReader(&mtxWinRead, &mtxWinWrite, &GetWindow,0);//определили свободное окно

window=freeWindow;//запомнили

AbstractWriter(&mtxWinWrite, &SetWindow);//записали его как занятое

usleep(1000000);//немного постоим, подождем пока дадут талон

WriteInTable(name,talon,operation,window);//записали все это в таблицу посетителей

int seat;//сиденье

if (window==0) //если окон свободных нет - сесть посидеть и ждать пока освободится окно

{ //определить свободное место

AbstractReader(&mtxSeatRead,&mtxSeatWrite,&GetSeat,0);//прочитали свободное место

seat = freeSeat;//запомнили свободное место

AbstractWriter(&mtxSeatWrite, &SetSeat);//записали его как занятое

int x,y;//координаты сиденья

if (seat <=4)//высчитываем координаты в зависимости от стороны

{

x=3;

y=11+1+(seat-1)\*3;

}

else

{

x=fieldWidth-2;

y=11+1+(seat-5)\*3;

}

GoFromTo(name,hereX,hereY,x,y,0);//идем к сиденью

hereX=x; hereY=y;//теперь мы тут, на сиденьях

AbstractReader(&mtxWinRead, &mtxWinWrite, &GetWindow,1);//ждем, пока окно не освободится

window=freeWindow;//запомнили окно

AbstractWriter(&mtxWinWrite,&SetWindow);//записали его как занятое

FreeASeat(seat);//освободить сидушку

WriteInTable(name,talon,operation,window);//записали все это в таблицу посетителей

}

//если есть свободные окна - идти в него

int width = 1+(fieldWidth-6)/5;//высчитываем снова ширину окна

int x = (window-1)\*(width)+width/2;//высчитываем координаты нужного окна

int y = 7;

GoFromTo(name,hereX,hereY,x,y,1);//идем к окну

hereX=x; hereY=y;//теперь мы возле окна

ChangeWindowTime(window, operation, talon);

FreeAWindow(window);//освобождаем окно

ClearRow(talon);//удаляем себя из таблицы

GoFromTo(name,hereX,hereY,33,20,0);//идем на выход, до автомата(почти)

GoFromTo(name,33,20,33,25,1);//идем на выход окончательно

usleep(200000);//перед тем как выйти еще чуть-чуть существуем в дверях

PrintInPoint(33,25," ");//удаляемся окончательно

}

void MutexsInit()

{//инициализация мютексов и семафоров

pthread\_mutex\_init(&mtxWinRead, NULL);//мютекс для чтения окон

pthread\_mutex\_init(&mtxWinWrite, NULL);//мютекс для записи окон

pthread\_mutex\_init(&mtxSeatRead, NULL);//мютекс для чтения сидений

pthread\_mutex\_init(&mtxSeatWrite, NULL);//мютекс для записи сидений

}

void MutexsDestroy()

{

pthread\_mutex\_destroy(&mtxWinRead);

pthread\_mutex\_destroy(&mtxWinWrite);

pthread\_mutex\_destroy(&mtxSeatWrite);

pthread\_mutex\_destroy(&mtxSeatRead);

}

void main()

{//главная функция

printf("\033[H\033[2J");//очищаем окно

DrawField();//рисуем фон - поле

MutexsInit();//инициализируем мютексы и семафоры

for (char name='A';name<'A'+20; name++)//для имен от А до T

{

int rc= pthread\_create(&tid[name-'A'],NULL, (void\*)visitor, (void\*)name);//создаем поток

usleep(2000000);//дополнительно немного ждем

}

getchar();//ждем, пока будет нажата клавиша или ctrl+c

MutexsDestroy();

}