**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc167373128)

[1 ОБЗОР ЛИТЕРАТУРЫ 6](#_Toc167373129)

[1.1 Постановка задачи 7](#_Toc167373130)

[1.2 Обзор методов и алгоритмов решения поставленной задачи 7](#_Toc167373131)

[2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 9](#_Toc167373132)

[2.1 Структура программы 10](#_Toc167373133)

[2.2 Описание библиотек 12](#_Toc167373134)

[3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 14](#_Toc167373135)

[3.1 Разработка алгоритма проверки файлов 14](#_Toc167373136)

[4 РЕЗУЛЬТАТЫ РАБОТЫ 17](#_Toc167373137)

[ЗАКЛЮЧЕНИЕ 21](#_Toc167373138)

[ЛИТЕРАТУРА 23](#_Toc167373139)

[ПРИЛОЖЕНИЕ А 24](#_Toc167373140)

[ПРИЛОЖЕНИЕ Б 25](#_Toc167373141)

# **ВВЕДЕНИЕ**

Linux, разработанное Линусом Торвальдсом в 1990-х годах, представляет собой мощное ядро операционной системы, обеспечивающее взаимодействие между аппаратным и программным обеспечением. Одним из ключевых преимуществ Linux является открытый исходный код, который позволяет пользователям свободно изучать, модифицировать и распространять его, обеспечивая гибкость и настраиваемость системы под индивидуальные потребности.

Дистрибутивы Linux представляют собой готовые пакеты операционной системы, основанные на ядре Linux. Они включают не только ядро, но и дополнительные компоненты, такие как программы, утилиты GNU, менеджеры пакетов и инструменты для настройки рабочего стола. Это упрощает установку и использование Linux, предоставляя различные настройки и функции для различных пользователей.

Linux также славится своей стабильностью и надежностью, что делает его популярным выбором для серверов и критически важных систем. Благодаря активному сообществу разработчиков, которые постоянно работают над исправлением ошибок и улучшением ядра, Linux обеспечивает высокий уровень стабильности.

В целом, Linux представляет собой гибкую, стабильную и надежную операционную систему с открытым исходным кодом, которая предлагает множество возможностей для настройки и адаптации под индивидуальные потребности пользователей.

## **1 ОБЗОР ЛИТЕРАТУРЫ**

В процессе разработки данной программы мы обратились к важным книгам, которые оказали значительное влияние на наш проект. Книга "Основы программирования в Linux" Нэйла Мэтью и "Программирование на языке С++ в Linux" Николая Секунова предоставили нам необходимые знания и руководства для программирования в Linux.

Книга "Linux Kernel Development" Роберта Лава была также очень полезной, так как она подробно рассматривает внутренние механизмы ядра Linux и разработку драйверов устройств, системных вызовов и управления памятью. Это помогло нам лучше понять работу операционной системы Linux и эффективно использовать ее возможности.

"Advanced Linux Programming" Марка Митчелла и Джеффа Олдхэма является ценным ресурсом для программистов, которые хотят расширить свои знания о программировании на платформе Linux. Она предлагает подробные объяснения, множество примеров кода и практические руководства, которые помогают развивать навыки программирования и создавать качественные программы на платформе Linux.

"Linux System Programming" Роберта Лава и Роджера Лайва охватывает широкий спектр тем, связанных с системным программированием в Linux, включая работу с процессами, потоками, сигналами, сокетами и другими системными ресурсами. Эта книга позволила нам углубить знания о системном программировании в Linux и применить их для реализации сложных функций в нашем проекте.

Чтение данных книг позволило нам освоить более продвинутые техники программирования в Linux, повысить эффективность и создать надежный и мощный проект.

* 1. **Постановка задачи**

Разрабатываемый проект представляет собой эмулятор арифметических инструкций с фиксированной запятой (формат Q16.16), предназначенный для моделирования и анализа работы основных арифметических операций над числами с ограниченной точностью. Цель проекта – создание надёжного и удобного инструмента, который позволит:

Отрабатывать выполнение арифметических операций (сложение, вычитание, умножение, деление, отрицание, вычисление модуля) для чисел в формате фиксированной запятой. • Исследовать особенности работы с ограниченной точностью, выявлять ошибки округления и переполнения. • Обеспечивать интерактивное тестирование алгоритмов через командный интерфейс, способный преобразовывать значения в различные числовые представления (float, int, двоичный и шестнадцатеричный форматы).

Таким образом, эмулятор служит как инструмент для обучения, отладки и экспериментальной проверки алгоритмов арифметики, применяемых в системах с вычислительными ограничениями (например, встраиваемые системы и цифровые процессоры).

* 1. **Обзор методов и алгоритмов решения поставленной задачи**

**Алгоритмы, реализованные в эмуляторе:**

**1) Алгоритм разбора входных инструкций:**

Программа принимает строковое представление арифметической команды от пользователя.

Выполняется синтаксический и семантический анализ ввода для определения типа операции и операндов.

**Если данные корректны, операнды преобразуются в формат фиксированной запятой для дальнейшей обработки.**

**2) Алгоритм выполнения операций сложения и вычитания:**

**Программа выравнивает операнды с учетом фиксированного масштаба.**

Выполняется операция сложения или вычитания, при этом осуществляется проверка переполнения.

В случае обнаружения потенциального переполнения выводится предупреждение.

3) Алгоритм умножения:

* Программа запрашивает у пользователя имя файла для скачивания.
* Она отправляет команду RETR на сервер с именем файла.
* Программа ожидает ответа от сервера.
* Сервер отправляет содержимое файла.
* Программа сохраняет содержимое файла на диск.
* Программа ожидает ответа от сервера.
* Если сервер отвечает кодом 226, то файл успешно скачан.
  + - * 1. **Алгоритм удаления файла:**
* Программа запрашивает у пользователя имя файла для удаления.
* Она отправляет команду DELE на сервер с именем файла.
* Программа ожидает ответа от сервера.
* Сервер отвечает кодом 250, если файл успешно удален.
  + - * 1. **Алгоритм смены каталога:**
* Программа запрашивает у пользователя имя каталога для смены.
* Она отправляет команду CD на сервер с именем каталога.
* Программа ожидает ответа от сервера.
* Сервер отвечает кодом 250, если каталог успешно изменен.
  + - * 1. **Алгоритм просмотра локальных файлов и каталогов:**
* Программа использует функцию opendir для открытия текущего каталога.
* Она использует функцию readdir для чтения имен файлов и каталогов в текущем каталоге.
* Программа выводит имена файлов и каталогов на экран.
  + - * 1. **Алгоритм выхода из системы:**
* Программа отправляет команду QUIT на сервер.
* Программа закрывает соединение с сервером.

## **2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

Функциональное проектирование является одним из ключевых этапов в разработке систем и приложений. Оно позволяет определить функции, которые должна выполнять система, и установить взаимосвязи между ними. Вот более подробное описание этапов функционального проектирования:

* + - * 1. Анализ требований: В начале процесса проводится тщательный анализ требований к системе. Это включает изучение целей и задач, которые система должна решать, а также функциональных и нефункциональных требований, предъявляемых к системе. Анализ требований помогает определить общий контекст системы и ее основные цели.
        2. Определение функций: На основе анализа требований определяются отдельные функции, которые система должна реализовывать. Функции представляют собой конкретные действия или операции, которые система выполняет для достижения своих целей. Например, если разрабатывается онлайн-магазин, функциями могут быть добавление товаров в корзину, оформление заказа и обработка платежей.
        3. Определение взаимодействия: После определения функций необходимо установить связи и взаимодействия между ними. Это позволяет определить последовательность выполнения функций, передачу данных между функциями и другие взаимодействия. Например, функция "оформление заказа" может зависеть от функции "добавление товаров в корзину" и передавать данные о заказе функции "обработка платежей".
        4. Создание диаграмм и моделей: Для визуализации функционального проектирования используются различные диаграммы и модели. Например, диаграмма потоков данных (DFD) позволяет показать поток данных между функциями, а диаграмма последовательности или диаграмма вариантов использования (Use Case) помогают описать последовательность выполнения функций и взаимодействие с акторами.

Качественное функциональное проектирование играет важную роль в успешной реализации системы. Оно обеспечивает согласованность между функциями, что позволяет избежать конфликтов и несоответствий. Кроме того, функциональное проектирование обеспечивает масштабируемость системы, позволяя легко добавлять новые функции при необходимости. Оптимизированные функции, разработанные в процессе функционального проектирования, позволяют достичь максимальной производительности и эффективного использования ресурсов системы.

## **2.1 Структура программы**

Структура программы выглядит следующим образом:

* + - * 1. **Подключение библиотек:**
* <iostream>;
* <sys/types.h>;
* <sys/socket.h>;
* <arpa/inet.h>;
* <unistd.h>;
* <cstring>;
* <dirent.h>;
* <sys/stat.h>;
* <cstdlib>;
* <cerrno>;
* <cstdio>.
  + - * 1. **Определение константы:**
* SIZE\_BUF: размер буфера для чтения/записи данных.
  + - * 1. Определение глобальных переменных:
* initial\_dir и current\_dir для хранения начальной и текущей директории проекта.
  + - * 1. **Функции:**
* send\_command: отправляет команду на сервер;
* recv\_response: получает ответ от сервера;
* login: осуществляет вход в систему;
* show\_menu: выводит меню доступных опций;
* handle\_list: обрабатывает команду LIST;
* handle\_stor: обрабатывает команду STOR;
* handle\_retr: обрабатывает команду RETR;
* handle\_dele: обрабатывает команду DELE;
* handle\_cd: обрабатывает команду CD;
* handle\_cd\_local: обрабатывает команду CD\_LOCAL;
* list\_directory\_tree\_local: рекурсивно выводит содержимое каталога;
* handle\_list\_local: обрабатывает команду LIST\_LOCAL;
* handle\_sendarch: обрабатывает команду SENDARCH;
* handle\_getarch: обрабатывает команду GETARCH.
  + - * 1. **Основная функция:**
* Создает сокет;
* Устанавливает соединение с сервером;
* Выполняет вход в систему;
* В цикле ожидает ввода пользователя и выполняет соответствующие действия:
  + - * 1. LIST: выводит список файлов и каталогов;
        2. STOR: отправляет файл на сервер;
        3. RETR: скачивает файл с сервера;
        4. DELE: удаляет файл на сервере;
        5. SENDARCH: отправляет архив на сервер;
        6. GETARCH: скачивает архив с сервера;
        7. CD: меняет текущий каталог на сервере;
        8. CD\_LOCAL: меняет текущий каталог локально;
        9. LIST\_LOCAL: выводит список файлов и каталогов локально;
        10. QUIT: закрывает соединение и завершает программу.

## **2.2 Описание библиотек**

Описание библиотек:

* <iostream>: Это библиотека ввода-вывода в языке программирования C++. Она предоставляет функции и объекты для работы с потоками ввода и вывода, такими как стандартные потоки (std::cin, std::cout) и функции для чтения и записи данных;
* <sys/types.h>: Эта библиотека содержит определения основных типов данных, используемых в системных вызовах. Она предоставляет типы данных, такие как pid\_t (идентификатор процесса), uid\_t (идентификатор пользователя) и другие, которые используются при работе с системными вызовами операционной системы;
* <sys/socket.h>: Это библиотека, которая предоставляет функции и структуры для работы с сокетами в языке программирования C. С помощью этой библиотеки можно создавать, устанавливать соединение, отправлять и принимать данные через сетевые сокеты;
* <arpa/inet.h>: Эта библиотека содержит функции для преобразования IP-адресов и портов между текстовым и двоичным представлением. Она также предоставляет функции для работы с сетевыми адресами и структурами данных, такими как sockaddr\_in;
* <unistd.h>: Эта библиотека содержит набор функций для работы с операционной системой Unix-like. Она предоставляет функции для работы с процессами, файлами, директориями, управления доступом к ресурсам и другими системными вызовами.
* <cstring>: Это библиотека, которая предоставляет функции для работы со строками в языке программирования C. Она содержит функции для копирования строк, сравнения строк, поиска символов и другие операции со строками;
* <dirent.h>: Эта библиотека содержит функции и структуры для работы с директориями в языке программирования C. Она позволяет получать информацию о файлах и поддиректориях в заданной директории, а также выполнять различные операции с ними;
* <sys/stat.h>: Эта библиотека предоставляет функции и структуры для получения информации о файле, такой как его размер, права доступа и другие атрибуты. Она также позволяет изменять атрибуты файлов;
* <cstdlib>: Это библиотека, которая содержит набор общих функций и макросов в языке программирования C++. Она предоставляет функции для управления памятью, генерации случайных чисел, выполнения арифметических операций и другие утилиты;
* <cerrno>: Эта библиотека содержит определения и макросы для работы с ошибками в языке программирования C++. Она предоставляет доступ к глобальной переменной errno, которая содержит код последней ошибки, произошедшей в программе, и функции для ее обработки;
* <cstdio>: Это библиотека, которая предоставляет функции для работы с файлами в языке программирования C. Она позволяет открывать, закрывать, читать и записывать данные в файлы, а также выполнять другие операции с файловой системой.

## **3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

Разработка программных модулей включает создание отдельных частей программного обеспечения с определенными функциями в рамках системы. Каждый модуль может быть написан на разных языках программирования и выполнять разнообразные задачи, такие как обработка данных, взаимодействие с пользователем и специфические вычисления. Процесс включает проектирование архитектуры, написание кода, тестирование и интеграцию с другими компонентами системы.

## **3.1 Разработка алгоритма проверки файлов**

Схема алгоритма функции обработки команды DELE приведена в приложении А.

Ниже представлены алгоритмы функций handle\_list и handle\_stor.

Алгоритм handle\_list:

* + - * 1. Отправка команды "LIST" на удаленный сервер с помощью функции send\_command, используя сокет sock;
        2. Создание буфера Buff определенного размера для хранения ответа от удаленного сервера;
        3. Запуск бесконечного цикла while;
        4. Получение длины ответа от удаленного сервера с помощью функции recv\_response и сохранение результата в переменную len;
        5. Если длина len меньше или равна нулю, выход из цикла;
        6. Если первые три символа строки Buff равны "END" (с помощью функции strncmp), выход из цикла;
        7. Вывод содержимого буфера Buff на экран с помощью функции printf;
        8. Повторение шагов 4-7 до тех пор, пока не будет достигнут выход из цикла;
        9. Вывод строки "226 Transfer complete." на экран.

Алгоритм handle\_stor:

* + - * 1. Создание буфера Buff определенного размера для хранения вводимого пользователем имени файла;
        2. Вывод на экран сообщения "Enter filename to store: ", запрашивающего у пользователя ввести имя файла для сохранения;
        3. Считывание строки из ввода пользователя с помощью функции fgets, которая сохраняет введенное имя файла в буфер Buff; Ограничение размера вводимой строки задается константой SIZE\_BUF;
        4. Удаление символа новой строки (\n) из строки Buff с помощью функции strcspn. Это выполняется для удаления символа новой строки, который автоматически добавляется функцией fgets;
        5. Формирование команды для сохранения файла с использованием строки "STOR " и введенного пользователем имени файла. Строка формируется с помощью оператора сложения (+) и функций std::string и c\_str();
        6. Отправка сформированной команды на сохранение файла с помощью функции send\_command, которая принимает сокет sock и передает команду для выполнения на удаленном сервере;
        7. Открытие файла с именем из буфера Buff в режиме чтения в формате двоичных данных с помощью функции fopen. Если файл не найден, вывод сообщения "File not found." и возврат из функции;
        8. Чтение данных из файла в буфер Buff с помощью функции fread и отправка данных через сокет sock с помощью функции send;
        9. Закрытие файла с помощью функции fclose;
        10. Отправка строки "END" через сокет sock с помощью функции send, чтобы указать конец передачи данных;
        11. Создание буфера recv\_buff определенного размера для хранения ответа от удаленного сервера;
        12. Цикл while для получения ответа от удаленного сервера и сохранения его в буфер recv\_buff;
        13. Вывод содержимого буфера recv\_buff на экран с помощью функции printf;
        14. Вывод строки "File stored successfully.\n" на экран.

## **4 РЕЗУЛЬТАТЫ РАБОТЫ**

Запуск и подключение клиента представлен на рисунке 4.1.

Initial working directory: /home/admin/Рабочий стол/tar\_working\_dir/FTPclient (копия)

Socket created

Connected to server

Username: 1

Password: 1

Login successful.

1. LIST

2. STOR <filename>

3. RETR <filename>

4. DELE <filename>

5. SENDARCH <archive\_name>

6. GETARCH <archive\_name>

7. CD <directory>

8. CD\_LOCAL <directory>

9. LIST\_LOCAL

10. QUIT

Рисунок 4.1 – Запуск клиента

При вводе команды «STOR» клиент получит файл от сервера рисунок 4.2.

Enter option: 2

Enter filename to store: 5.txt

150 Ready to receive data.

226 Transfer complete.

File stored successfully.

1. LIST

2. STOR <filename>

3. RETR <filename>

4. DELE <filename>

5. SENDARCH <archive\_name>

6. GETARCH <archive\_name>

7. CD <directory>

8. CD\_LOCAL <directory>

9. LIST\_LOCAL

10. QUIT

Enter option: 1

Root:

- 1.txt

- 2.txt

- 3.txt

- 5.txt

226 Transfer complete.

Рисунок 4.2 – Отправка файла

При вводе команды «LIST» клиент получит директорию сервера рисунок 4.3.

Enter option: 1

Root:

- 1.txt

- 2.txt

- 3.txt

226 Transfer complete.

Рисунок 4.3 – Вывод директории

При вводе команды «RETR» клиент получит указанный файл с сервера рисунок 4.4.

Enter option: 3

Enter filename to retrieve: 1.txt

150 Ready to send data.

END

File retrieved successfully.

1. LIST

2. STOR <filename>

3. RETR <filename>

4. DELE <filename>

5. SENDARCH <archive\_name>

6. GETARCH <archive\_name>

7. CD <directory>

8. CD\_LOCAL <directory>

9. LIST\_LOCAL

10. QUIT

Enter option: 9

Root:

- 4.txt

- 5.txt

- 1.txt

END

Рисунок 4.4 – Скачивание файла

При вводе команды «DELE» сервер удалит указанный файл рисунок 4.5.

Enter option: 4

Enter filename to delete: 2.txt

Delete response: 250 File deleted.

1. LIST

2. STOR <filename>

3. RETR <filename>

4. DELE <filename>

5. SENDARCH <archive\_name>

6. GETARCH <archive\_name>

7. CD <directory>

8. CD\_LOCAL <directory>

9. LIST\_LOCAL

10. QUIT

Enter option: 1

Root:

- 1.txt

- 3.txt

- 5.txt

226 Transfer complete.

Рисунок 4.5 – Удаление файла

При вводе команды «CD» сервер сменит директорию на указанную представлено на рисунке 4.6.

Enter option: 7

Enter directory to change to (leave empty to return to root): ..

250 Directory changed successfully.

1. LIST

2. STOR <filename>

3. RETR <filename>

4. DELE <filename>

5. SENDARCH <archive\_name>

6. GETARCH <archive\_name>

7. CD <directory>

8. CD\_LOCAL <directory>

9. LIST\_LOCAL

10. QUIT

Enter option: 7

Enter directory to change to (leave empty to return to root): RootDirectory

Рисунок 4.6 – Смена директории

При вводе команды «CD\_LOCAL» клиент сменит директорию на указанную представлено на рисунке 4.7.

Enter option: 8

Enter local directory to change to (leave empty to return to project root): Catal

Attempting to change to directory: /home/admin/Рабочий стол/tar\_working\_dir/FTPclient/ Catal

250 Local directory changed successfully to /home/admin/Рабочий стол/tar\_working\_dir/ FTPclient/Catal.

Рисунок 4.7 – Смена директории клиента

При вводе команды «LIST\_LOCAL» клиент выведет свою директорию рисунок 4.8.

Enter option: 9

Root:

- Catal

- 4.txt

- 5.txt

- Makefile

- 1.txt

- 2.txt

- client.cpp

- 3.txt

- server

- client

END

Рисунок 4.8 – Вывод директории клиента

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной курсовой работы на операционной системе Linux был разработан FTP-клиент с возможностью отправки и получения архивированных каталогов. Linux, как мощная и гибкая операционная система, предоставляет разработчикам широкий спектр свободы и контроля над системой, что сыграло важную роль в успешной реализации проекта.

Благодаря обширному набору инструментов и библиотек для программирования на языке C++ на платформе Linux, разработчики имели возможность создать высококачественное программное обеспечение. Они смогли полностью использовать все преимущества, предоставляемые этими средствами, что привело к разработке надежного и эффективного FTP-клиента.

Таким образом, выполнение данной курсовой работы подтверждает, что Linux является оптимальной операционной системой для разработки программного обеспечения, обеспечивая разработчикам широкие возможности и инструменты для достижения высокого качества и функциональности.

# **ЛИТЕРАТУРА**

[1] Н. Мэтью Основы программирования в Linux. 2009 г.

[2] Н. Секунов Программирование на языке С++ в Linux. БХВ-Петербург, 2004 г.

[3] М. Митчелл и Д. Олдхэм «Advanced Linux Programming», 2001 г.

[4] Р. Лава «Linux Kernel Development», 2006 г.

[5] Электронный ресурс: https://github.com.

# **ПРИЛОЖЕНИЕ А**

Схема алгоритма

# **ПРИЛОЖЕНИЕ Б**

Ведомость документов