Учреждение образования

«Белорусский Государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

«Многопоточный FTP-Клиент»

по дисциплине

«Системное программное обеспечение вычислительных машин»

Выполнил: Руководитель:

студент гр. 850505

Григарович Алексей Андреевич Глоба А. А.

Минск, 2020

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc41505934)

[1. ОБЗОР ИСТОЧНИКОВ 4](#_Toc41505935)

1.1. Основные термины …………………………………………….. 4

1.2. Выбор технологий для создания проекта…………………….. 4

[2. СТРУКТУРНОЕ ПРОЕКТИРОВАНИE 5](#_Toc41505936)

2.1. Общее структурное описание состава программного обеспечения…………………………………………………………. 5

2.2. Описание деления проекта на отдельные программы, модули или другие составные части………………………………………... 5

[3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 6](#_Toc41505937)

3.1. Класс Commands………………………………………………... 6

3.2. Поток……………………………………………………………. 7

[4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 10](#_Toc41505938)

[5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 11](#_Toc41505939)

[6. ТЕСТИРОВАНИЕ 13](#_Toc41505940)

[ЗАКЛЮЧЕНИЕ 16](#_Toc41505941)

[СПИСОК ИСТОЧНИКОВ 17](#_Toc41505942)

# **ВВЕДЕНИЕ**

В данной работе планируется реализация FTP-Клиента для упрощенного доступа к FTP-Серверу. В зависимости от назначения может либо предоставлять пользователю простой доступ к удалённому FTP-серверу в режиме текстовой консоли, беря на себя только работу по пересылке команд пользователя и файлов, либо отображать файлы на удалённом сервере, как если бы они являлись частью файловой системы компьютера пользователя, либо и то и другое. В этой работе будет использована текстовая консоль, так как в пользовательском интерфейсе нет необходимости.

В простейшем для пользователя случае FTP-клиент представляет собой эмулятор файловой системы, которая просто находится на другом компьютере. С этой файловой системой можно совершать все привычные пользователю действия: копировать файлы с сервера и на сервер, удалять файлы, создавать новые файлы. В отдельных случаях возможно также открытие файлов — для просмотра, запуска программ, редактирования. Необходимо учитывать лишь, что открытие файла подразумевает его предварительное скачивание на компьютер пользователя.

В работе будет использован чужой FTP-Сервер для тестирования собственного FTP-Клиента, так как цель данной работы написать FTP-Клиент, а не сервер.

# **ОБЗОР ИСТОЧНИКОВ**

* 1. **Основные термины.**

Сокет – название программного интерфейса для обеспечения обмена данными между процессами.

Сокет данных – сокет по которому между клиентом и сервером будут передаваться строковые данные.

* 1. **Выбор технологий для создания проекта.**

Для разработки была выбрана IDE CLion, так как это по моему мнению лучшее IDE для дистрибутива Manjaro Linux и не только. Разработка будет идти на языке C++.

FileZilla — один из бесплатных FTP клиентов. Он поддерживает FTP, SFTP, и FTPS (FTP через SSL/TLS) и имеет настраиваемый интерфейс с поддержкой смены тем оформления. Оснащён возможностью перетаскивания объектов, синхронизацией каталогов и поиском на удалённом сервере. Поддерживает многопоточную загрузку файлов, а также докачку при обрыве (если поддерживается сервером) интернет-соединения. В этой же работе поддерживается многопоточность, а интерфейс и вовсе отсутствует.

# **СТРУКТУРНОЕ ПРОЕКТИРОВАНИE**

* 1. **Общее структурное описание состава программного обеспечения**

Структурная схема представлена в Приложении А.

Данная программа состоит из четырех блоков: блок пользовательского ввода, управляющий блок, блок отправки команд, блок передачи/принятия файлов.

Блок пользовательского ввода отвечает за ввод информации непосредственно пользователем.

Управляющий блок осуществляет обработку ввода и последующее управление оставшимися модулями.

Блок отправки команд осуществляет отправку команд на сервер и получение ответа с сервера.

Блок передачи/принятия файлов отвечает за загрузку и скачивание файлов с сервера.

* 1. **Описание деления проекта на отдельные программы, модули или другие составные части**

Блок пользовательского ввода ждет пока пользователь введет команду в консоль, затем он ее отправляет в управляющий блок.

Управляющий блок получает введенную пользователем команду от соответствующего блока и анализирует ее. В зависимости от того какая команда была введена, он посылает команду блоку отправки команд. Если введенная команда не реализована, то он снова ожидает ввод.

Блок отправки команд отправляет команду по сокету команд серверу, после чего принимает ответ от сервера и выводит его в консоль.

Блок передачи/принятия файлов использует второй поток, в котором по сокету данных отправляет или получает файлы и записывает их на диск. Еще этот блок используется для отображения файлов и папок на сервере, так как сервер отправляет их имена по сокету данных.

# **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

* 1. **Класс Commands:**

Диаграмма классов представлена в Приложении Б.

public:

static string getReply(int sock);

static int sendCommand(int sock, char buff[BUFF\_SIZE]);

static int pasv(int sock, char buff[BUFF\_SIZE]);

static int list(int sock, int dataSock, char buff[BUFF\_SIZE]);

static int retr(int sock, int dataSock, char buff[BUFF\_SIZE], char typeData, string path);

static int stor(int sock, int dataSock, char buff[BUFF\_SIZE], char typeData, string path);

static char type(int sock, char buff[BUFF\_SIZE]);

static int quit(int sock, char buff[BUFF\_SIZE]);

Класс Commands хранит в себе все необходимые методы связанные с отправкой на сервер команд либо файловых данных.

Метод getReply(int sock) нужен для получения сообщения от сервера. Он принимает в аргументах сокет. Дальше при помощи команды recv(), по сокету принимается сообщение от сервера, которое записывается в буфер. Выводит этот самый буфер в консоль и возвращает его. Если во время приема сообщения функцией recv() разрывается соединение, то функция возвращает “-1”.

Метод sendCommand(int sock, char buff[BUFF\_SIZE]) отправляет команду серверу. Он принимает в себя сокет и буфер (сообщение), который нужно передать серверу. Передача осуществляется командой send(). Дальше идет считывание ответа сервера и возвращается 0 при успешных действиях, иначе -1.

Метод pasv(int sock, char buff[BUFF\_SIZE]) предназначен для входа в пассивный режим. Он принимает в аргументах сокет и сообщение серверу. Для начала это сообщение отправляется серверу и считывается его ответ. В ответе сервер должен прислать строку типа “277 … (…)”. В скобках будут 6 чисел, первые 4 из них это IP адрес, а 2 последние это составляющие порта. Чтобы из них получить порт, нужно первое число умножить на 256 и прибавить к произведению второе число. Дальше создается новое соединение с новым ip и портом для отправки по нему уже данных, а не команд. При успешной инициализации сокета, функция его возвращает, иначе вернет 0.

Метод list(int sock, int dataSock, char buff[BUFF\_SIZE]) выводит список папок и файлов отправляемых сервером. Он принимает в себя сокет, сокет для отправки данных и сообщение серверу. Первым делом отправляется запрос серверу на получение списка папок и файлов. Дальше функция ждет пока по сокету отправки данных будет отправлен список. После получения, список выводится в консоль. Сокет для отправки данных закрывается и считывается ответ с сервера.

Метод retr(int sock, int dataSock, char buff[BUFF\_SIZE], char typeData, string path) нужен для скачивания файлов с сервера. В аргументах он получет сокет для отправки команд и сокет отправки данных, сообщение серверу, тип открытия файла и директорию файла. Сперва отправляется сообщение серверу. Дальше функция создает файл с указанным названием в аргументах команды и указанной директорией бинарным либо текстовым способом. В файл начинается записывается полученными по сокету данных буферами, которые отправляет сервер. После скачивания считывается ответ сервера и возвращается 0.

Метод stor(int sock, int dataSock, char buff[BUFF\_SIZE], char typeData, string path) нужен для загрузки файла на сервер. Подробнее об этой функции написано в главе Разработки программных модулей.

Метод type(int sock, char buff[BUFF\_SIZE]) нужен для установки типа передачи файла по сокету передачи данных. Функция принимает в аргументах сокет и сообщение для отправки серверу. Функция отправляет сообщение, в аргументах которого указан тип “I” (бинарный) или “A” (текстовый). Если сервер отправил ответ с кодом 200, то функция возвращает вводимый тип, иначе возвращает предыдущий.

Метод quit(int sock, char buff[BUFF\_SIZE]) нужен для отсоединения от сервера. Функция принимает сокет и сообщение, которое отправляет серверу по тому же сокету. После сокет закрывается и возвращается 1. Программа закрывается.

* 1. **Поток:**

Функция, исполняемая потоком, выглядит следующим образом:

void\* threadFun(void\* arg) {

char buff[BUFF\_SIZE];

int pipeRead = ((data\*)arg)->filedes[0], pipeWrite = ((data\*)arg)->filedes[1];

int sock = ((data\*)arg)->sock, dataSock = ((data\*)arg)->dataSock;

read(pipeRead, &buff, BUFF\_SIZE);

isWorking = true;

cout << "\r< Выполнение задачи в фоновом потоке..." << endl;

//LIST - список файлов

if (strstr(buff, "LIST") != nullptr && dataSock != 0) authFlag = Commands::list(sock, dataSock, buff);

//RETR - скачать файл с сервера

if (strstr(buff, "RETR") != nullptr && dataSock != 0) authFlag = Commands::retr(sock, dataSock, buff, typeData, path);

//STOR - загрузить файл на сервер

if (strstr(buff, "STOR") != nullptr && dataSock != 0) authFlag = Commands::stor(sock, dataSock, buff, typeData, path);

cout << "< Завершение выполнения задачи в фоновом потоке..." << endl;

memset(buff, 0, BUFF\_SIZE);

isWorking = false;

pthread\_exit(0);

}

Функция принимает в аргументах структуру:

struct data {

int filedes[2]; //0 - чтение, 1 - запись

int sock = 0, dataSock = 0;

};

В filedes записываются дескрипторы чтения и записи для канала, в sock записывается сокет для отправки команд и в dataSock сокет для отправки или принятия данных.

Функцией read() поток ждет пока основной поток запишет в канал сообщение. После получения поток выполняет полученную ему команду и завершается.

Сам поток создается после выполнения функции pasv(), функцией pthread\_create(&pthread, nullptr, threadFun, args), аргументами которой является функция threadFun(void\* arg) принимающая структуру, указанную четвертым аргументом.

# **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

Для примера рассмотрим алгоритм действий, при вводе пользователем команды “STOR”.

В аргументах функции передается sock (сокет передачи команд серверу), dataSock (сокет для передачи данных), buff (буфер с командой), typeData (тип данных, используется для открытия файла в текстовом режиме, либо в бинарном), path (месторасположение файла).

Функция сразу проверяет: если введена команда STOR без аргументов, то выводится сообщение “недостаточно аргументов” и возвращает 0.

Дальше идет оправка buff серверу, чтобы запросить начало загрузки файла. И сразу запомним имя файла в filename, указанного в аргументах STOR.

Получим ответ от сервера. Если пришел код ответа 125, то начнем передавать файл, иначе выйдем.

Перед передачей файла откроем его с режимом указанном в typeData (askii или binary) и отправим его с помощью функции sendfile().

Выводится надпись об успешной отправке с количеством переданных байт, которое нам вернула функция sendfile(). Дальше считывается ответ от сервера и возвращается 0, если не было ошибки считывания ответа.

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

При запуске программы пользователь увидит консоль. Первым делом программа попросит нас ввести ip адрес сервера, а затем порт для подключения к нему. После подключения к нему мы уже можем вводить команды, но для начала нужно “залогиниться” на сервере с помощью команд USER и PASS:

|  |  |
| --- | --- |
| USER имя\_пользователя | После USER нам нужно ввести свой логин на сервере |
| PASS пароль | Пароль нам нужен чтобы войти под логином и уже дальше производить нужные нам операции. |

После логина мы уже можем делать все доступные нам операции. Есть много команд для отправки на FTP-Сервер, но не все могут быть реализованы. В нашем же случае, на FTP-Сервере реализованы следующие команды:

|  |  |
| --- | --- |
| USER “аргумент” | Отправка логина серверу. Аргумент команды — логин |
| PASS “аргумент” | Отправка пароля серверу. Аргумент команды — пароль |
| PASV | Вход в пассивный режим. С ним мы можем пользоваться такими командами как: LIST, STOR, RETR. Эта команда создает отдельный сокет для передачи файлов по ip и порту, который нам отправит сервер. |
| LIST | После отправки, сервер нам по сокету данных будет отправлять все содержимое текущей директории (файлы, папки). |
| RETR “аргумент” | Скачать файл с сервера. Аргумент команды — имя файла. Скачивание и запись происходит в отдельном потоке. |
| STOR “аргумент” | Загрузить файл на сервер. Аргумент команды — имя файла. Загрузка происходит в отдельном потоке. |
| ABOR | Прервать передачу файла. |
| CWD “аргумент” | Перейти в директорию. Аргумент команды — имя директории. |
| MKD “аргумент” | Создать директорию в текущей директории. Аргумент команды — имя создаваемой директории. |
| PWD | Вернуть имя текущей директории в данный момент. |
| RMD “аргумент” | Удалить директорию. Аргумент команды — имя удаляемой директории. |
| TYPE “аргумент” | Установка режима записи файлов. Аргумент команды — режим(‘I’, ‘A’). ‘A’ — текстовый режим записи. ‘I’ — бинарный режим записи. |
| HELP | Показать реализованные сервером команды. |
| SYST | Получить информацию о системе. |
| STAT | Получить информацию о соединении. |
| NOOP | Ничего не делать. |
| DELE “аргумент” | Удалить файл. Аргумент команды — имя файла. |
| SIZE “аргумент” | Вернуть размер файла. Аргумент команды — имя файла. |
| QUIT | Отключиться от сервера и завершить работу. |

# **ТЕСТИРОВАНИЕ**

Для тестирования подключимся к FTP-Серверу, который загружен на сервисы Амазона. IP сервера на данный момент 13.56.207.108 и порт для подключения 2000. Попробуем подключиться (см. рисунок 6.1).

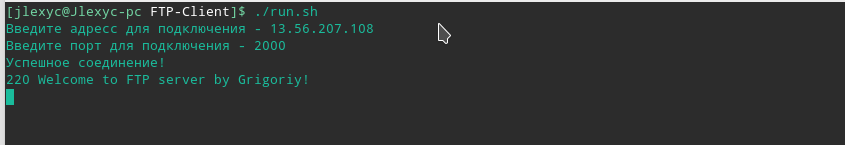


Рисунок 6.1 – Подключение к серверу.

Успешное подключение. Сервер нас приветствует. Дальше нужно войти в свою учетную запись. Используем логин Lexys и пароль Lexys для входа (см. рисунок 6.2).



Рисунок 6.2 – Логин.

Вход в учетную запись выполнен. Теперь попробуем загрузить на сервер картинку, а потом ее скачать. А перед этим создадим папку, куда ее и загрузим. Для начала возьмем следующую картинку (см. рисунок 6.3).

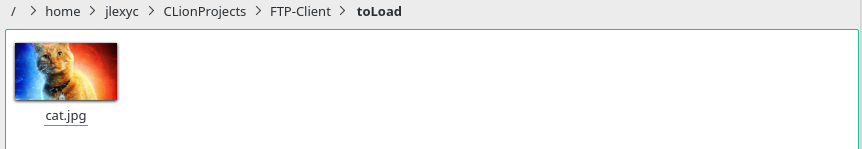


Рисунок 6.3 – Файл для загрузки.

Теперь создадим директорию командой MKD, куда будем загружать картинку (см. рисунок 6.4).



Рисунок 6.4 – Создание папки files.

Чтобы загрузить картинку, нужно перейти в эту директорию при помощи команды CWD и дальше использовать команду STOR, предварительно войдя в пассивный режим (см. рисунок 6.5).

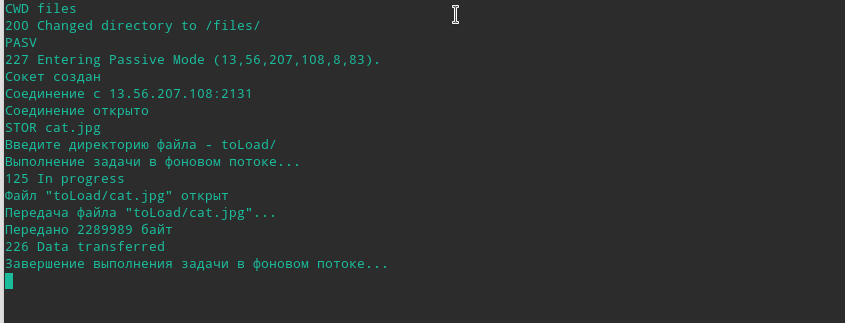


Рисунок 6.5 – Загрузка картинки.

Картинка успешно загрузилась. Теперь скачаем эту картинку с этой же папки с помощью команды RETR (см. рисунок 6.6).

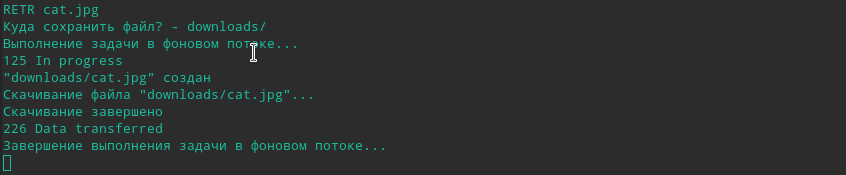


Рисунок 6.6 – Скачивание картинки.

Картинка успешно скачана без потери данных в папку downloads/ (см. рисунок 6.7).

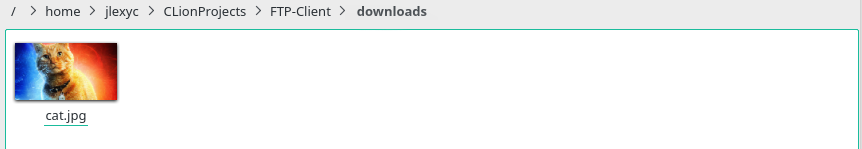


Рисунок 6.7 – Скаченная картинка.

Теперь закроем соединение и выйдем командой QUIT (см. рисунок 6.8).

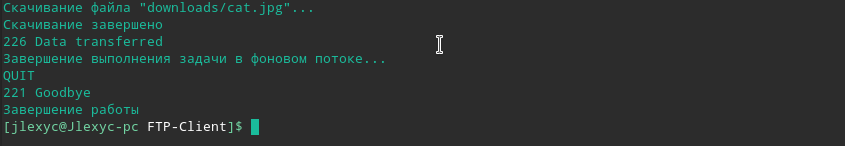


Рисунок 6.8 – Закрытие соединения.

Задача выполнена и тестирование завершено успешно.

# **ЗАКЛЮЧЕНИЕ**

В результате работы над этим проектом был разработан FTP-Клиент. Был реализован некоторый список команд для его работы с сервером. Были выполнены все условия, поставленные темой курсового проекта. Клиент был протестирован на сервере. В результате тестирования проблем выявлено не было. Был получен опыт в конструировании ПО.

# **СПИСОК ИСТОЧНИКОВ**

1. FTP-Клиент [Электронный ресурс].– электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/FTP-клиент>
2. Протокол FTP [Электронный ресурс].– электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/FTP>
3. Список кодов ответов FTP-Сервера [Электронный ресурс].– электронные данные.– Режим доступа: [https://ru.wikipedia.org/wiki/Список\_кодов\_ответов\_FTP](https://ru.wikipedia.org/wiki/Список_кодов_ответов_FTP%20)
4. Описание команд серверу [Электронный ресурс].– электронные данные. – Режим доступа: <https://support.solarwinds.com>