

Страничка тайтл

Страничка для ТЗ

Страничка для ТЗ

1 Реферат

Объектом разработки и исследования является протокол FTP для передачи данных посредством сети. Целью работы является разработка клиента для загрузки, передачи и управления файлами на удаленном сервере. В результате выполнения работы создается консольное приложение, пользователи которого могут передавать, загружать файлы. Данная расчетная записка содержит 41 страницу. Она включает в себя 10 рисунков, 1 листинг.

Содержание

1	Реферат	4
2	Введение	6
3	Аналитический раздел	8
3.0.1	Описание предметной области	8
3.0.2	Архитектура системы	8
3.1	Протокол прикладного уровня	10
3.2	Протокол транспортного уровня	12
3.3	Активный и пассивный режим	12
4	Конструкторский раздел	14
4.1	IDEF0	14
4.2	Описание структуры разрабатываемой программы	15
5	Технологический раздел	19
5.1	Язык программирования	19
5.2	Среды программирования	20
5.3	Листинг	20
5.4	Примеры работы программы	38
6	Заключение	40
	Список использованных источников	41

2 Введение

FTP-клиент FTP File Transfer Protocol — Компьютерная программа для упрощения доступа к FTP серверу. В зависимости от назначения может либо предоставлять пользователю простой доступ к удаленному FTP-серверу в режиме текстовой консоли, беря на себя только работу по пересылке команд пользователя и файлов, либо отображать файлы на удаленном сервере, как если бы они являлись частью файловой системы компьютера пользователя, либо и то и другое. В последних двух случаях FTP-клиент берёт на себя задачу интерпретации действий пользователя в команды протокола FTP, тем самым давая возможность использовать протокол передачи файлов без ознакомления со всеми его премудростями.

Частными примерами использования FTP-клиента могут быть:

- 1) публикация страниц на сайте Веб-разработчика;
- 2) скачивание музыки, программ и любых других файлов данных обычным пользователем интернета;

В простейшем для пользователя (но при этом наиболее комплексном) 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. Примерами таких программ могут служить:

- 1) интернет-браузеры (часто работают в режиме «только чтение», то есть не позволяют добавлять файлы на сервер);
- 2) онлайн клиенты, работа с которыми осуществляется посредством любого интернет-браузера;

В данной работе реализуется FTP клиент для Linux. Пользователь может подключиться к удаленному серверу используя ранее созданный профиль в удаленной системе. Анонимный пользователь не допускается.

- 1) получения файлов с сервера компании;
- 2) в курсе базового ознакомления с консолью;
- 3) обучение в рамках курса "Сети"

Целью проекта является получения программы, позволяющей пользователю подключаться к удаленному серверу и отправлять, загружать, манипулировать данными. Для достижения поставленной цели необходимо выполнить следующие поставленные задачи:

1) аналитические:

- а) формализовать предметную область;
- б) выбрать объект исследования;
- в) обосновать выбор;

2) конструкторские:

- а) описать алгоритмы;
- б) указать особенности практической реализации;
- в) формализовать описанные данные, ограничения и допущения, требования к программному обеспечению и способов взаимодействия программы с пользователем;

3) технологические:

- а) обосновать выбор программной реализации;
- б) описать требуемое программное обеспечение;
- в) описать входные и выходные данные разработанного программного продукта;
- г) тестирование;

3 Аналитический раздел

3.0.1 Описание предметной области

FTP — протокол передачи файлов по сети, является одним из старейших прикладных протоколов, появившихся задолго до HTTP, и даже до TCP/IP, в 1971 году; в первое время он работал поверх протокола NCP. Он и сегодня широко используется для распространения ПО и доступа к удалённым хостам. В отличие от TFTP, гарантирует передачу (либо выдачу ошибки) за счет применения квотируемого протокола.

Протокол построен на архитектуре «клиент-сервер» и использует разные сетевые соединения для передачи команд и данных между клиентом и сервером. Пользователи FTP могут пройти аутентификацию, передавая логин и пароль открытым текстом, или же, если это разрешено на сервере, они могут подключиться анонимно. Можно использовать протокол SSH для безопасной передачи, скрывающей (шифрующей) логин и пароль, а также шифрующей содержимое.

Первые клиентские FTP-приложения были интерактивными инструментами командной строки, реализующими стандартные команды и синтаксис. С тех пор были разработаны графические пользовательские интерфейсы для многих используемых по сей день операционных систем. Среди этих интерфейсов как компоненты программы общего веб-дизайна вроде Microsoft Expression Web, так и специализированные FTP-клиенты (например, FileZilla).

3.0.2 Архитектура системы

Клиент - программа для упрощения доступа к FTP серверу. В зависимости от назначения может либо предоставлять пользователю простой доступ к удаленному FTP-серверу в режиме текстовой консоли, беря на себя только работу по пересылке команд пользователя и файлов, либо отображать файлы на удаленном сервере как если бы они являлись частью файловой системы компьютера пользователя, либо и то и другое. В последних двух случаях FTP-клиент берет на себя задачу интерпретации действий пользователя в команды протокола FTP, тем самым давая возможность использовать

протокол передачи файлов без ознакомления со всеми его премудростями. В простейшем для пользователя (но при этом наиболее комплексном) случае FTP-клиент представляет из себя эмулятор файловой системы, которая просто находится на другом компьютере. С этой файловой системой можно совершать все привычные пользователю действия: копировать файлы с 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. Благодаря распространенности протокола FTP, простые (с точки зрения реализации) FTP-клиенты есть практически в каждой операционной системе. Однако использование этих клиентов требует навыков использования консоли, а также знания команд протокола для общения с сервером. Так в Windows такой утилитой является ftp.exe. Во многих сборках Linux так же есть утилита ftp. Файловая система на удаленном сервере как правило имеет настройки прав доступа для различных пользователей. Так, например, анонимным пользователям могут быть доступны лишь некоторые файлы, о существовании других пользователи знать не будут. Другой группе пользователей могут быть доступны другие файлы или, например, в дополнение к правам на чтение файлов, могут быть также даны права на запись новых или обновление имеющихся файлов. Диапазон вариантов прав доступа зависит от операционной системы и программного обеспечения каждого конкретного FTP-сервера. Как правило, разделяют права на просмотр содержимого папки (т.е. возможность получить список содержащихся в ней файлов), на чтение файла(ов), на запись (создание, удаление, обновление) файла(ов). Для авторизации FTP-сервер, при подключении к нему FTP-клиента, запрашивает у последнего имя пользователя и пароль. Большинство FTP-клиентов в свою очередь запрашивают эти данные у пользователя в интерактивном режиме. Есть так же и другой способ указать эти данные, включив их в URL FTP-сервера. Так, например, в строке: `//setevoy:pass@ftp.someserver.com://` - указание того, что мы используем протокол FTP, где:

1) setevoy - имя пользователя;

2) : - разделитель;

- 3) pass - пароль;
- 4) @ - разделитель аутентификационной информации и адреса сервера;
- 5) someserver.ru - адрес FTP - сервера;

Нередки случаи, когда такой метод указания имени пользователя и пароля является единственным, который поддерживает FTP-клиент.

3.1 Протокол прикладного уровня

Протокол пересылки файлов ftp - это протокол прикладного уровня стека TCP/IP . Работа этого протокола основана на архитектуре “клиент-сервер”. Команды и данные, в отличие от большинства других протоколов передаются по разным портам. Порт 20 используется для передачи данных, порт 21 для передачи команд. Основные команды:. Как правило, эта команда открывает сессию FTP между клиентом и сервером. Аргументом команды является имя (идентификатор) пользователя для работы с файловой системой. Эта команда может подаваться не только в начале, но и в середине сессии, если, например, пользователь желает изменить идентификатор, от имени которого будут проводиться действия. При этом все переменные, относящиеся к старому идентификатору, освобождаются. Если во время изменения идентификатора происходит обмен данными, обмен завершается со старым идентификатором пользователя. Данная команда подается после ввода идентификатора пользователя и, в качестве аргумента содержит пароль пользователя. Напомним, что данные аутентификации FTP передаются по сети открытым текстом, поэтому для обеспечения защищенности канала пользователю необходимо предпринимать дополнительные меры. Команда позволяет пользователям работать с различными каталогами удаленной файловой системы. Аргументом команды является строка, указывающая путь каталога удаленной файловой системы, в котором желает работать пользователь. Команда реинициализации. Эта команда очищает все переменные текущего пользователя, сбрасывает параметры соединения. Если в момент подачи команды происходит передача данных, передача продолжается и завершается с прежними параметрами. Команда закрывает управляющий канал. Если в момент подачи команды происходит передача данных, канал закрывается после окон-

чания передачи данных. Команды управления потоком устанавливают параметры передачи данных. Все параметры, описываемые этими командами имеют значение по умолчанию, поэтому команды управления потоком используются только тогда, когда необходимо изменить значение параметров передачи, используемых по умолчанию. Команды управления потоком могут подаваться в любом порядке, но все они должны предшествовать командам FTP-сервиса. Из команд управления потоком данных следует выделить следующие: Команда назначает адрес и порт хоста, который будет использоваться как активный участник соединения по каналу передачи данных. Аргументами команды являются 32-битный IP адрес и 16-битный номер порта соединения. Эти значения разбиты на шесть 8-битных полей и представлены в десятичном виде: h1, h2, h3, h4, p1, p2, где hN - байты адреса (от старшего к младшему), а pN - байты порта (от старшего к младшему). Эта команда отправляется модулю, который будет играть пассивную роль, в передаче данных («слушать» соединение). Ответом на данную команду должна быть строка, содержащая адрес и порт хоста, находящиеся в режиме ожидания соединения в формате команды PORT - «h1, h2, h3, h4, p1, p2». Команды TYPE, STRU, MODE определяют, соответственно, тип передаваемых данных (ASCII, Image и другие), структуру или формат передачи данных (File, Record, Page), способ передачи (Stream, Block и другие). Использование этих команд очень важно при построении взаимодействия в гетерогенных средах и весьма отличающихся операционных и файловых систем взаимодействующих хостов. Команды FTP-сервиса определяют действия, которые необходимо произвести с указанными файлами. Как правило, аргументом команд этой группы является путь к файлу. Синтаксис указанного пути должен удовлетворять требованиям формата файловой системы обработчика команды. Из команд FTP-сервиса можно выделить следующие: Эта команда указывает модулю «Программа передачи данных сервера» передать копию файла, заданного параметром этой команды, модулю передачи данных на другом конце соединения. Команда указывает модулю «Программа передачи данных сервера» принять данные по каналу передачи данных и сохранить их как файл, имя которого задано параметром этой команды. Если такой файл уже существует, он будет замещен новым, если нет, будет создан новый. Команды RNFR и RNT0 должны следовать одна за другой. Первая команда

содержит в качестве аргумента старое имя файла, вторая - новое. Последовательное применение этих команд переименовывает файл. Команда предписывает серверу прервать выполнение предшествующей сервисной команды (например, передачу файла) и закрыть канал передачи данных. Команда DELE удаляет указанный файл. Команды MKD и RMD, соответственно, создают и удаляют указанный в аргументе каталог. При помощи команд LIST и NLST можно получить список файлов в указанном каталоге.

3.2 Протокол транспортного уровня

Транспортный уровень (англ. Transport layer) - 4-й уровень сетевой модели OSI предназначен для доставки данных без ошибок, потерь и дублирования в той последовательности, как они были переданы. При этом не важно, какие данные передаются, откуда и куда, то есть он предоставляет сам механизм передачи. Transmission Control Protocol (TCP) (протокол управления передачей) - один из основных сетевых протоколов Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. TCP - это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета (см. также T/TCP). В отличие от UDP гарантирует, что приложение получит данные точно в такой же последовательности, в какой они были отправлены, и без потерь. Реализация TCP, как правило, встроена в ядро системы, хотя есть и реализации TCP в контексте приложения.

3.3 Активный и пассивный режим

В активном режиме FTP клиент соединяется с произвольного непривилегированного порта ($N > 1024$) к FTP серверному командному порту 21. Затем, клиент начинает слушать порт $N+1$ и посылать FTP команду PORT $N+1$ на FTP сервер. В ответ, сервер соединяется с указанным портом данных клиента из своего локального порта данных 20. (рисунок 1)

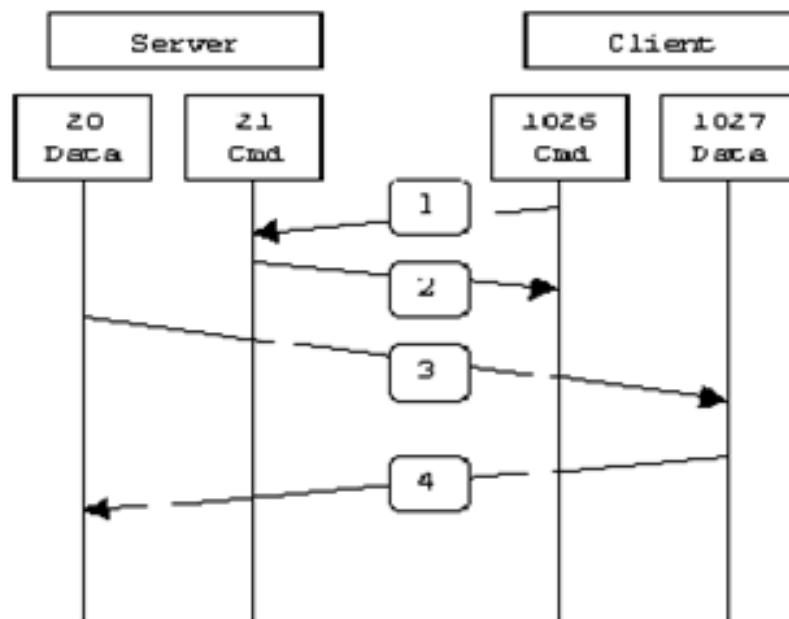


Рис. 1. Принцип работы ftp

В пассивном режиме FTP клиент инициирует оба соединения с сервером, решая проблему с фаерволами, которые фильтруют входящий порт данных клиента. При открытии FTP соединения, клиент локально открывает два непривилегированных порта ($N > 1024$ и $N+1$). Первый порт контактирует с сервером на порт 21, но вместо того, чтобы затем выдать команду PORT и позволить серверу в ответ соединиться с его портом данных, клиент выдает команду PASV. В результате сервер открывает произвольный непривилегированный порт ($P > 1024$) и посылает клиенту команду PORT P. Затем, для передачи данных, клиент инициирует соединение от порта $N+1$ к порту P на сервере.

4 Конструкторский раздел

Для создания системы необходимо определиться с её структурой и теми возможностями, которые она будет обеспечивать при накладываемых на неё ограничениях. В первую очередь следует определить формат данных в рамках предметной области, а также задать описание функционирования программы. Кроме того, необходимо задать перечень алгоритмов и подходов к ним, что позволит определиться с выбором конкретных из них, наиболее подходящих для реализации заданной цели.

4.1 IDEF0

Для описания функционирования программы можно использовать графическую нотацию IDEF0. IDEF0-диаграмма разрабатываемой системы приведена на рисунках 2, 3.

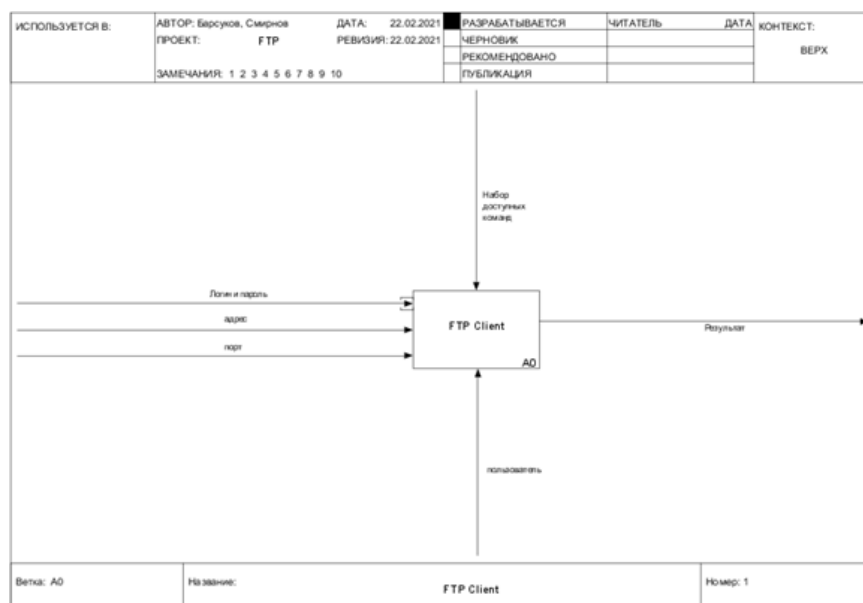


Рис. 2. A0

Общая концепция состоит в следующем: пользователь указывает необходимые данные для подключения и устанавливается соединение с ftp сервером через связанный сокет. После этого программа ожидает ввод команды и отправляет запрос на сервер после этого ждет ответ и выводит результат на экран до тех пока пользователь не завершит работу.

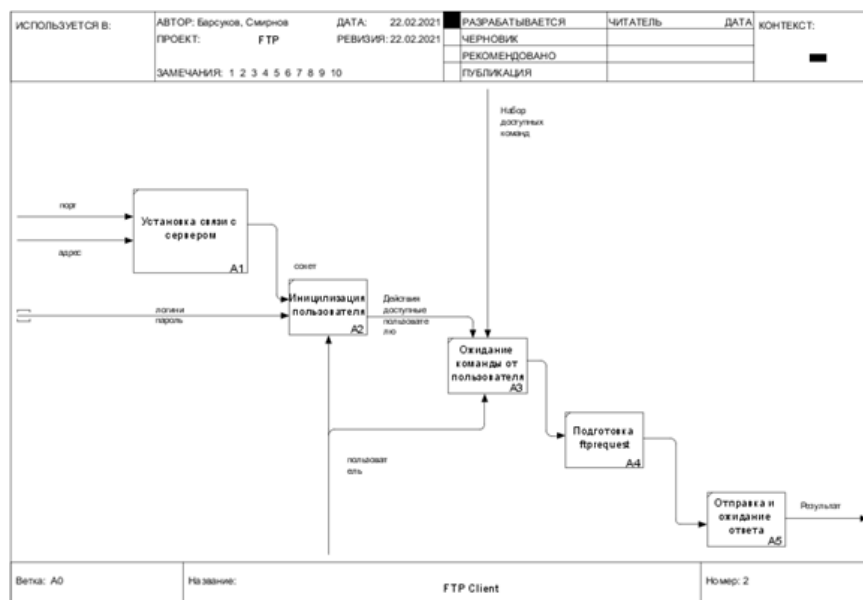


Рис. 3. A0 детально

4.2 Описание структуры разрабатываемой программы

Для разрабатываемой программы необходимо выделить хранимые данные, общий подход к структуре программного продукта, а также конкретизировать формат работы модулей.

На рисунках 4, 5 представлены прототипы используемых в работе классов. Рисунок 4 показывает прототип класса Socket, представляющий из себя обертку вокруг `<sys/socket>`. Предоставляет следующие атрибуты и методы:

1) Методы:

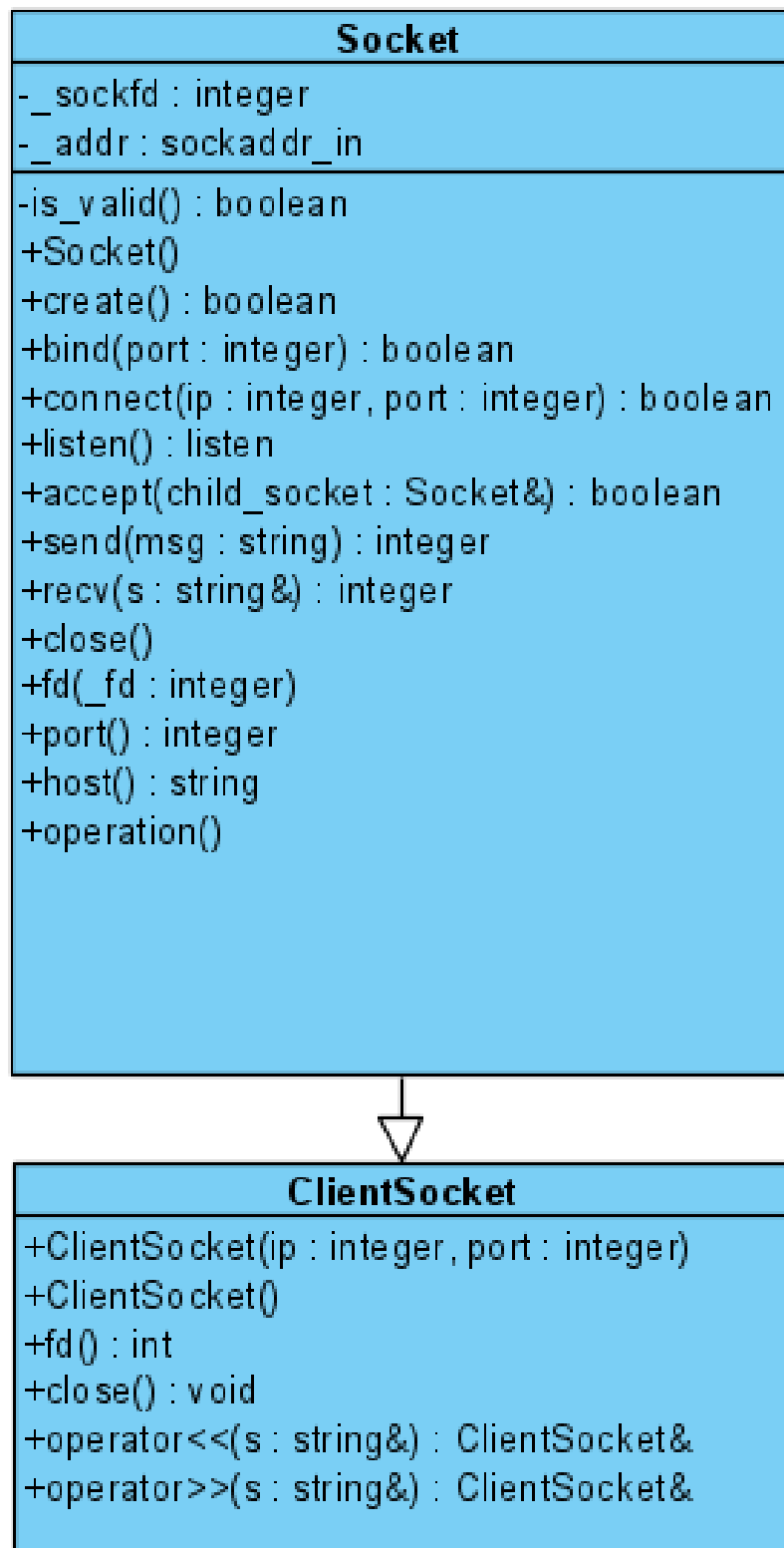
- а) `Socket()` - конструктор без аргументов. Инициализирует `_addr = 0` `_sockfd = -1`;
- б) `is_valid()` - проверяет файловый дескриптор;
- в) `fd()` - возвращает файловый дескриптор;
- г) `fd(int _fd)` - устанавливает значение `fd`;
- д) `port()` - возвращает порт ;
- е) `host()` - возвращает адрес хоста;
- ж) `create()` - создает файловый дескриптор сокета;

- з) `bind(int port)` - привязывает сокет к порту;
- и) `connect(int ip, int port)` - связывает сокет с удаленным сервером;
- к) `listen()` - ставит сокет на ожидание;
- л) `accept()` - Функция `accept` ожидает запрос на установку TCP-соединения от удаленного хоста. В качестве аргумента ей передается дескриптор слушающего сокета;
- м) `recv()` - получает ответ от связанного сокета;
- н) `close()` - освобождает ресурсы дескриптора сокета

2) Атрибуты:

- а) `_sockfd` - дескриптор сокета;
- б) `_addr` - представляет из себя структуру хранящую в себе адрес, порт и семейство

Также на рисунке 4 представлен сокет `clientSocket` класс который будет представлять из себя конкретную реализацию сокета используемого в нашем клиенте.



Powered By  Visual Paradigm Community Edition

Рис. 4. Диаграмма Socket и ClientSocket

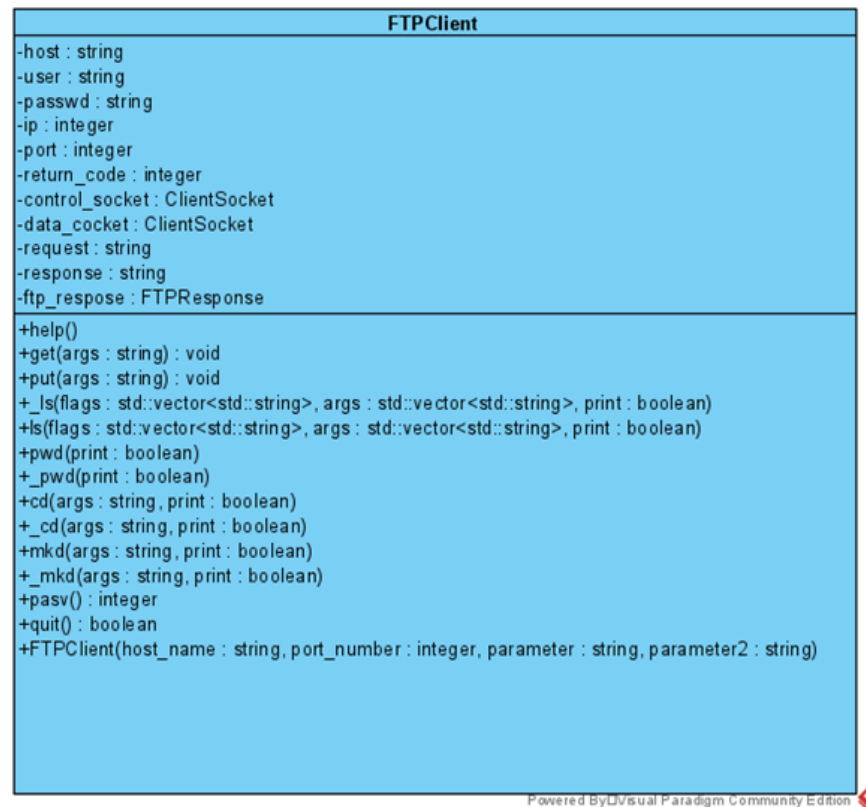


Рис. 5. Диаграмма FTPClient

На рисунке 5 изображен прототип FTPClient класса, который реализует основную логику взаимодействия пользователя с клиентом.

5 Технологический раздел

Для написания программы необходимо определить, какие средства программной реализации требуется использовать. Для этого нужно привести перечень технологий и описать достоинства и недостатки конкретного выбора.

5.1 Язык программирования

Программный продукт реализуется на языке C++. В научных вычислениях в последние годы все большее внимание обращается на язык c++ как основу для написания простых в использовании и при этом высокоэффективных программных комплексов

Некоторые отличительные особенности языка C++:

- 1) C++ содержит в себе все основные черты объектно-ориентированных языков программирования: наличие объектов и инкапсуляцию данных, наследование, полиморфизм и абстракцию типов;
- 2) C++ широко использует указатели на переменные и функции, кроме того, поддерживает арифметику указателей, и тем самым позволяет осуществлять непосредственный доступ и манипуляции с адресами памяти; удобным средством для передачи параметров являются ссылки;
- 3) Объем C++ невелик, так как практически все выполняемые функции оформлены в виде подключаемых библиотек, также C++ полностью поддерживает технологию структурного программирования и обеспечивает полный набор соответствующих операторов;
- 4) Базовые типы данных совпадают с типами данных Ассемблера, на преобразования типов налагаются незначительные ограничения;
- 5) C++ предлагает большой набор операций, многие из которых соответствуют машинным командам и допускают прямую трансляцию в машинный код, а их разнообразие позволяет выбирать различные наборы для минимизации результирующего кода;

5.2 Среды программирования

1) Clion

- a) Clion - это современная кроссплатформенная IDE для C и C++ разработанная компанией JetBrains. Предоставляет помощь при написании кода с помощью Smart Completion. Позволяет генерировать шаблоны для классов. Быстрый просмотр документации и безопасный рефакторинг

2) VS Code

- a) Visual Studio Code - редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией.

5.3 Листинг

```
// Class file of ftp client socket.
#include "headers/cp_ftp_client.h"

// This function set host name, port name, user name and
password of client.
FTPClient::FTPClient(std::string host_name, int port_number,
std::string user_name, std::string password){
    std::cout<<"\nSi.FTP-Client Started\n\n";
    host = host_name;
    user = user_name;
    passwd = password;
    port = port_number;
}
```

```

// Destructer function of ftp client class.
FTPClient::~FTPClient() {

}

// Initiate connection between Client and Server by sending
// UserName and Password.
void FTPClient::start(){
    std::cout<<"Connecting to Host : "<< host<< " Port : 
        "<<port<<std::endl;
    // try to connecting to server otherwise throws
        exceptions.
    try{
        control_socket = new ClientSocket(host,port);
        *control_socket>>response;
        std::cout<<FTPResponse(response).parseResponse
            ();

        request = FTPRequest("USER",user).getRequest();
        *control_socket<< request;
        *control_socket>>response;

        request = FTPRequest("PASS",passwd).getRequest
            ();
        *control_socket<<request;
        *control_socket>>response;

        std::cout<<FTPResponse(response).parseResponse(
            return_code);
        if(return_code != 230){
            std::cout<<"Re-enter User Name : ";
            std::cin>>user;
            std::cout<<"Re-enter Password : ";
            passwd = getPassword();
            start();
        }
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return ;
    }
}

```

```

    }
}

// Get a valid FTP command from user and send it to Server.
void FTPClient::communicate(){
    std::string command,cmd;
    std::vector<std::string> flags,args;
    // always ready to send.
    while(1){
        flags.clear();
        args.clear();
        std::cout<<"Cp.FTP > ";
        std::getline(std::cin,command);
        if(parseCommand(command,cmd,flags,args)){
            // get command to get file from
            serverside.
            if(cmd=="get" && (args.size() == 1 ||
                args.size()==2) && flags.size()==0){
                std::string curr_loc = _pwd(
                    false);
                std::string curr_loc_server =
                    pwd(false);

                if(args.size()==2){
                    if(_cd(args[1],false)!=
                        1){
                        std::cout<<"
                            Destination
                            doesn't
                            exist. File
                            Transfer
                            couldn't be
                            done."<<std
                                ::endl;
                        continue;
                    }
                }

                std::string filePath =
                    getFilePath(args[0]);
            }
        }
    }
}

```

```

        if(filePath!=""){
            if(cd(filePath,false)
                != 250){
                _cd(curr_loc,
                    false);
                std::cout<<"
                    Destination
                    doesn't
                    exist. File
                    Transfer
                    couldn't be
                    done."<<std
                    ::endl;
                continue;
            }
        }

        get(getFileName(args[0]));
        cd(curr_loc_server,false);
        _cd(curr_loc,false);
    }
    // put command to put file on the
    serverside .
    else if(cmd=="put" && (args.size() == 1
        || args.size()==2) && flags.size()
        ==0){
        std::string curr_loc = pwd(
            false);

        if(args.size()==2){
            if(cd(args[1],false)!=
                250){
                std::cout<<"
                    Destination
                    doesn't
                    exist. File
                    Transfer
                    couldn't be
                    done."<<std
                    ::endl;
                continue;
            }
        }
    }

```

```

        }
    }

    put(args[0]);
    cd(curr_loc, false);
}

// pwd command to see present working
// directory on sever side.
else if(cmd=="pwd" && args.size() == 0
        && flags.size() == 0){
    pwd();
}

// cd command to change directory on
// server side.
else if(cmd=="cd" && flags.size() == 0
        && args.size() == 1){
    cd(args[0]);
}

// ls command to list files and folders
// on server side.
else if(cmd=="ls"){

    if(pasv() != 227){
        std::cout << "Couldn't
                        get file listing." <<
                        std::endl;
        continue;
    }
    ls(flags, args);
}

// mkdir command to make directory on
// server side.
else if(cmd=="mkdir" && args.size() ==
        1 && flags.size() == 0){
    bool flag = true;
    std::string curr_loc = pwd(
        false);
    std::vector<string> dirs =
        tokenize(args[0], "/");

    for(int i=0; i<dirs.size(); i++){

```



```

        if(mkd(dirs[i],false)
            !=257 && cd(dirs[i],
            false) != 250){
            std::cout<<"
            Couldn't
            create the
            required
            directory
            structure
            ."<<std::
            endl;
            flag = false;
            break;
        }

    }

    cd(curr_loc,false);

    if(flag){
        std::cout<<"Directory
        structure "<<args
        [0]<< " successfully
        created."<<std::
        endl;
    }
}

// pwd command to see present working
// directory on clientside.
else if(cmd=="!pwd" && args.size() == 0
    && flags.size()==0){
    _pwd();
}

// cd command to change directory on
// clientside.
else if(cmd=="!cd" && flags.size() == 0
    && args.size() == 1){
    _cd(args[0]);
}

// ls command to list files and folders
// on clientside.

```

```

else if(cmd=="!ls"){
    _ls(flags, args);
}
// mkdir command to make directory on
clientside.
else if(cmd=="!mkdir" && args.size() ==
1 && flags.size() == 0){
    bool flag = true;
    std::string curr_loc = _pwd(
        false);

    std::vector<string> dirs =
        tokenize(args[0], "/");
    for(int i=0; i<dirs.size(); i++){
        int status = _mkd(dirs[
            i], false);
        status = status | _cd(
            dirs[i], false);
        if(_mkd(dirs[i], false)
            !=1 && _cd(dirs[i],
            false) != 1){
            std::cout<<"
                Couldn't
                create the
                required
                directory
                structure
                ."<<std::
                endl;
            flag = false;
            break;
        }
    }

    _cd(curr_loc, false);
    if(flag){
        std::cout<<"Directory
            structure "<<args
            [0]<< " successfully
            created."<<std::

```

```

                                endl;
                            }
                        }
                    }
// quit command to exit from Programme.
else if(cmd=="quit"){
    if(quit()){
        (*control_socket).close
        ();
        return;
    }else{
        std::cout<<"Couldn't
        terminate the
        session."<<std::endl
        ;
    }
}
// help command for any syntax related
help.
else if(cmd=="help"){
    help();
}
else{
    std::cout<<"Command improperly
    formatted. Type \"help\" for
    reference."<<std::endl;
}
}
}

// This function get file from server system.
void FTPClient::get(std::string args){
    std::ofstream out(getFileName(args).c_str(), std::ios::
    out| std::ios::binary);
    string data;
    double length;
    // check for availability of file.
    if(out){
        request = FTPRequest("TYPE","I").getRequest();
        // socket exceptions handling
        try{

```

```

        *control_socket<<request;
        *control_socket>>response;
        ftp_response.setResponse(response);
        std::cout<<ftp_response.parseResponse(
            return_code);
        if(return_code != 200){
            return;
        }
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return;
    }

    // transfer mode is not passive mode.
    if(pasv()!=227){
        std::cout<<"File Transfer couldn't be
            initiated."<<std::endl;
        return;
    }

    request = FTPRequest("RETR",getFileName(args))
        .getRequest();
    try{
        *control_socket<<request;
        *control_socket>>response;
        std::cout<<FTPResponse(response).
            parseResponse(return_code);
        if(return_code != 150){
            return;
        }
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return;
    }

    std::cout<<"Receiving File : "<<getFileName(
        args)<<" ...."<<std::endl;

    // store data in buffer named data.

```

```

while (1){
    data = "";
    *data_socket>>data;
    length = length + data.length();
    if(data.length()==0){
        break;
    }
    out<<data;
}

// close connection.
(*data_socket).close();
*control_socket>>response;
out.close();
int status_code,precision;
FTPResponse ftp_response(response);
std::cout<<ftp_response.parseResponse(
    status_code);

// get file size by status code.
if(status_code == 226){
    std::string size_msg = "bytes";
    precision = 0;

    if(length/1024 >= 1){
        size_msg = "KB";
        length /= 1024;
        precision = 2;

        if(length/1024 >= 1){
            size_msg="MB";
            length /= 1024;

            if(length/1024 >= 1){
                size_msg="GB";
                length /= 1024;
            }
        }
    }
}

```

```

        std::cout<<std::setprecision(precision)
        <<std::fixed<<"Succesfully
        transferred file : "<<getFileName(
        args)<< " ( " << length <<size_msg<<
        " )"<<std::endl;
    }
}
}
}

// This fucntion put file on server system.
void FTPClient::put(std::string args){
    std::ifstream in(args.c_str(), std::ios::in | std::ios
        ::binary | std::ios::ate);
    // check for existence of file name.
    if(in){
        long length = in.tellg();
        in.seekg (0, in.beg);
        request =  FTPRequest("TYPE","I").getRequest();
        // try to connct to server.
        try{
            *control_socket<<request;
            *control_socket>>response;
            ftp_response.setResponse(response);
            std::cout<<ftp_response.parseResponse(
                return_code);
            if(return_code != 200){
                return;
            }
        } catch(SocketException &e){
            std::cout<<"Exception occurred : "<<e.
                description()<<std::endl;
            return;
        }

        // transfer mode is not passive mode.
        if(pasv()!=227){
            std::cout<<"File Transfer couldn't be
                initiated."<<std::endl;

```

```

        return;
    }

    request =  FTPRequest("STOR",getFileName(args))
        .getRequest();
    try{
        *control_socket<<request;
        *control_socket>>response;
        ftp_response.setResponse(response);
        std::cout<<ftp_response.parseResponse(
            return_code);
        if(return_code != 150){
            return;
        }
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return;
    }

    std::cout<<"Sending File : "<<getFileName(args)
        <<" ...."<<std::endl;
    string data;

    double c_length=length;

    // send all data to server.
    while (length>0){
        int read_sz = MAXRECV<length ? MAXRECV
            : length;
        char buf[MAXRECV+1];
        in.read(buf,read_sz);
        data.assign(buf,read_sz);
        *data_socket<<data;
        length -= read_sz;
    }

    // close connection.
    (*data_socket).close();
    *control_socket>>response;
    in.close();

```

```

        int status_code,precision;
        FTPResponse ftp_response(response);
        std::cout<<ftp_response.parseResponse(
            status_code);

        // get file size by looking at status code.
        if(status_code == 226){
            std::string size_msg = "bytes";
            precision = 0;

            if(c_length/1024 >= 1){
                size_msg = "KB";
                c_length /= 1024;
                precision = 2;

                if(c_length/1024 >= 1){
                    size_msg="MB";
                    c_length /= 1024;

                    if(c_length/1024 >= 1){
                        size_msg="GB";
                        c_length /=
                            1024;
                    }
                }
            }
            std::cout<<std::setprecision(precision)
                <<std::fixed<<"Succefully
                transferred file : "<<getFileName(
                args)<< " ( " << c_length <<size_msg
                << " )" <<std::endl;
        }
    }else{
        std::cout<<"File : "<<getFileName(args)<<"
            doesn't exist. Please check the filename."<<
            std::endl;
    }
}

// This function return exit status.

```



```

bool FTPClient::quit(){
    request = FTPRequest("QUIT").getRequest();
    // socket exceptions handling
    try{
        *control_socket<<request;
        *control_socket>>response;
        std::cout<<FTPResponse(response).parseResponse
            ();
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return false;
    }
    return true;
}

// This function initiate passive mode.
int FTPClient::pasv(){
    request = FTPRequest("PASV").getRequest();
    // socket exceptions handling
    try{
        *control_socket<<request;
        *control_socket>>response;
        FTPResponse ftp_response(response);
        ftp_response.setResponse(response);
        std::cout<<ftp_response.parseResponse(
            return_code);
        if(return_code != 227){
            return return_code;
        }
        int port = ftp_response.getPort();
        data_socket = new ClientSocket(host,port);
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return -1;
    }
    return return_code;
}

```

```

// This function show present working directory on server
system.
std::string FTPClient::pwd( bool print){
    request = FTPRequest("PWD","").getRequest();
    // socket exceptions handling
    try{
        *control_socket<<request;
        *control_socket>>response;
        ftp_response.setResponse(response);

        std::string p_response = ftp_response.
            parseResponse(return_code);
        if(print){
            std::cout<<p_response;
        }
        return p_response.substr(1,p_response.length()
            -4);
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return "" ;
    }
}

// This function change directory on server system.
int FTPClient::cd(std::string args,bool print){
    request = FTPRequest("CWD",args).getRequest();
    // socket exceptions handling
    try{
        *control_socket<<request;
        *control_socket>>response;
        ftp_response.setResponse(response);
        if(print){
            std::cout<<ftp_response.parseResponse(
                return_code);
        }
        return ftp_response.returnValue();
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return -1;
    }
}

```

```

    }
}

// This function make directory on server system.
int FTPClient::mkd(std::string args, bool print){
    request = FTPRequest("MKD", args).getRequest();
    // socket exceptions handling
    try{
        *control_socket<<request;
        *control_socket>>response;
        ftp_response.setResponse(response);
        if(print){
            std::cout<<ftp_response.parseResponse(
                return_code);
        }
        return ftp_response.returnCode();
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return -1;
    }
}

// This function directory on Client System.
void FTPClient::ls(std::vector<std::string> flags, std::vector<
    std::string> args, bool print){
    request = FTPRequest("LIST", flags, args).getRequest();
    // socket exceptions handling
    try{
        *control_socket<<request;
        *control_socket>>response;
        ftp_response.setResponse(response);
        std::string p_response= ftp_response.
            parseResponse(return_code);
        if(print){
            std::cout<<p_response;
        }
        if(return_code != 150){
            response = "";
            return;
        }
    }
}

```

```

        // get response from data socket.
        while(1){
            response = "";
            *data_socket >> response;
            if(response.length()==0){
                break;
            }
            if(print){
                std::cout<<response;
            }

        }

        (*control_socket)>>response;
        ftp_response.setResponse(response);
        p_response= ftp_response.parseResponse(
            return_code);
        if(print){
            std::cout<<p_response;
        }
    } catch(SocketException &e){
        std::cout<<"Exception occurred : "<<e.
            description()<<std::endl;
        return ;
    }
}

// This function return present working directory on Client
System.
std::string FTPClient::_pwd(bool print){
    request = FTPRequest("pwd","").getRequest("\n");
    response = exec_cmd("pwd",request);
    if(print){
        std::cout<<response;
    }
    return response.substr(1,response.length()-3);
}

```

```

// This function change directory on Client System.
int FTPClient::_cd(std::string args, bool print){
    response = exec_cmd("cd",args,return_code);
    if(print){
        std::cout<<response;
    }
    return return_code;
}

// This function show listing of files and directories on
Client System.
void FTPClient::_ls(std::vector<std::string> flags, std::vector
<std::string> args, bool print){
    request = FTPRequest("ls",flags,args).getRequest("\n");
    response = exec_cmd("ls",request);
    if(print){
        std::cout<<response;
    }
}

// This function create directory on Client System.
int FTPClient::_mkd(std::string args,bool print){
    response = exec_cmd("mkdir",args,return_code);
    if(print){
        std::cout<<response;
    }
    return return_code;
}

// This function return solutions for syntax related queries.
void FTPClient::help(){
    string cmd_desc = "";

    cmd_desc += "put [sourcepath]filename [destination] :
    filename to upload the file(relative or absolute
    address) to the server [at the specified destination
    ].\n" ;
    cmd_desc += "get [sourcepath]filename [destination] :
    filename to download the file(relative or absolute
    address) from the server [at the specified
    destination].\n" ;
}

```

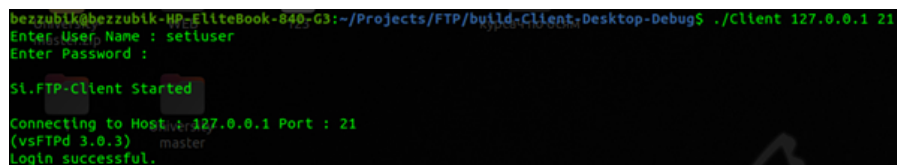
```

cmd_desc += "ls [flags(like -l, -a, etc)] [args1 args2
...] : to list the files under the present directory
or mentoned directories in args[] of the server.\n"
;
cmd_desc += "cd destiation : to change the present
working directory of the server to destination.\n" ;
cmd_desc += "pwd : to display the present working
directory of the server.\n" ;
cmd_desc += "!ls [flags(like -l, -a, etc)] [args1 args2
...] : to list the files under the present directory
or mentoned directories in args[] of your machine.\n
n" ;
cmd_desc += "!cd destiation : to change the present
working directory of your machine to destination.\n"
;
cmd_desc += "!pwd : to display the present working
directory of your machine.\n" ;
cmd_desc += "quit : to quit from ftp session and return
to Unix prompt.\n";

std::cout<<cmd_desc<<std::endl;
}

```

5.4 Примеры работы программы

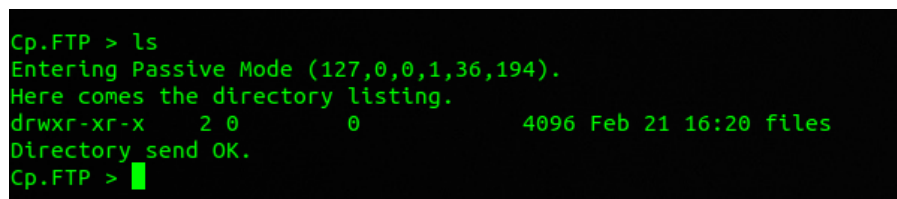


```

bezzubik@bezzubik-HP-EliteBook-840-G3:~/Projects/FTP/build-Client-Desktop-Debug$ ./Client 127.0.0.1 21
Enter User Name : setluser
Enter Password :
Sl.FTP-Client Started
Connecting to Host: 127.0.0.1 Port: 21
(vsFTPD 3.0.3) master
Login successful.

```

Рис. 6. Пример работы программы



```

Cp.FTP > ls
Entering Passive Mode (127,0,0,1,36,194).
Here comes the directory listing.
drwxr-xr-x  2 0          0          4096 Feb 21 16:20 files
Directory send OK.
Cp.FTP >

```

Рис. 7. Пример работы программы

```
Cp.FTP > ls
Entering Passive Mode (127,0,0,1,36,194).
Here comes the directory listing.
drwxr-xr-x  2 0      0      4096 Feb 21 16:20 files
Directory send OK.
Cp.FTP > █
```

Рис. 8. Пример работы программы

```
Cp.FTP > ls
Entering Passive Mode (127,0,0,1,36,194).
Here comes the directory listing.
drwxr-xr-x  2 0      0      4096 Feb 21 16:20 files
Directory send OK.
Cp.FTP > █
```

Рис. 9. Пример работы программы

```
Cp.FTP > ls
Entering Passive Mode (127,0,0,1,36,194).
Here comes the directory listing.
drwxr-xr-x  2 0      0      4096 Feb 21 16:20 files
Directory send OK.
Cp.FTP > █
```

Рис. 10. Пример работы программы

6 Заключение

В результате проведенной работы был реализован прототип FTP - клиента для операционной системы Linux. Были решены следующие задачи:

1) аналитические:

- а) формализована предметная область;
- б) выбран объект исследования;
- в) приведено обоснование выбора алгоритма.

2) конструкторские:

- а) приведено описание алгоритма;
- б) указаны особенности практической реализации;
- в) формализованы описания данных, ограничений и допущений, требований к программному обеспечению способов взаимодействия программы с пользователем.

3) технологические:

- а) обоснован выбор программной реализации;
- б) обоснован выбор языка программирования;
- в) приведен список использованного программного обеспечения;
- г) приведен листинг основных модулей программы.

Список использованных источников

1. Э. Таненбаум, Д. Уэзеролл Компьютерные сети, 2015
2. Visual Studio Code [Электронный ресурс] // URL: <https://code.visualstudio.com>
3. Clion [Электронный ресурс] // URL: <https://www.jetbrains.com/ru-ru/clion/>
4. В. Олифер, Н. Олифер Компьютерные сети, 2016-Питер
5. А. Сергеев Основы локальных компьютерных сетей, 2016
6. А. Робачевский Интернет изнутри. Экосистема глобальной сети, 2017