Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра вычислительных систем

Курсовая работа по дисциплине
Сетевое программирование

Разработка сетевого приложения. Клиент на базе протокола FTP в активном режиме.

Выполнил:

студент гр.ИП-012 _____/ Николаев А.Д. /
ФИО студента

«__» _____ 2023 г.

Проверил:

Ассистент кафедры ВС _____/ Ревун А.Л. /
ФИО преподавателя

«__» _____ 2023 г. Оценка_____

Новосибирск 2023 г.

Оглавление

Задание на курсовую работу

Разработать клиент-серверное консольное сетевое приложение, работающее на базе протокола прикладного уровня FTP, на языке программирования C/C++ в операционной системе на базе ядра Linux.

Теоретический материал

Протокол FTP (File Transfer Protocol) – протокол пересылки файлов поверх протокола транспортного уровня TCP. Информация о данном протоколе и принципах его работы хранится в документе RFC (Request for Comments) 959.

Протокол FTP использует два порта: один порт используется для передачи команд (обычно порт с номером 21), а второй используется для передачи данных (обычно порт с номером 20) (см. рис. 1). [2]

Рис. 1 Схема работы протокола FTP

Пересылаемые по управляющему соединению команды имеют следующий формат: большими латинскими буквами пишется 3-буквенное или 4-буквенное название команды и аргумент команды, если необходим. В курсовом проекте будет реализована только часть из этих команд (см. табл. 1).

В ходе общения по FTP клиент получает от сервера сообщения, содержащие комментарии и коды-отклики, информация о которых также содержится в документе RFR 959. Код отклика представляется 3-значным числом. Каждая из цифр имеет определённую значимость. Первая цифра отвечает за один из трёх исходов: успех, отказ или указание на ошибку либо неполный ответ. Вторая цифра определяет тип ошибки. Третья цифра окончательно специфицируют ошибку. [2]

Классификация откликов по 1-ой цифре:

1yz – указывает на выполнение операции, необходимо подождать её завершения;

2yz – сообщает об успешном выполнении команды;

3yz – указывает на успешное достижение промежуточной точки, но для продолжения требуется дополнительная информация;

4yz – извещает о временной ошибке, но можно повторно отправить команду;

5yz – извещает о постоянной ошибке;

Таблица 1. Команды FTP-сервера

Команда	**Описание**
ABOR	Прервать передачу файла
CDUP	Сменить директорию на вышестоящую
CWD	Сменить директорию (CWD <dirname>)
DELE	Удалить файл (DELE <filename>)
HELP	Выводит список команд, принимаемых сервером
LIST	Возвращает список файлов директории. Список передаётся через соединение данных
MKDT	Возвращает время модификации файлов (MDTM <filename>)
MKD	Создать директорию (MKD <dirname>)
NLST	Возвращает список файлов директории в более кратком формате. Список передаётся через соединение данных
NOOP	Пустая операция
PASS	Ввести пароль (PASS <password>)
PORT	Войти в активный режим (PORT <IP-address 4 Byte>,<port 2Byte>),
PWD	Возвращает текущую директорию
QUIT	Отключиться
RETR	Скачать файл. Файл передаётся через соединение данных (RETR <filename>)
RMD	Удалить директорию (RMD <dirname>)
RNFR и FNTO	Переименовать файл. RNFR – что переименовывать, RNTO – во что переименовывать (RNFR <filename> RNTO <new filename>)
SIZE	Возвращает размер файла (SIZE <filename>)
STOR	Закачать файл. Файл передаётся через соединение данных (STOR <filename>)
SYST	Возвращает тип системы
TYPE	Установить тип передачи файла (TYPE I(A))
USER	Имя пользователя для входа на сервер (USER <name>)

Классификация откликов по 2-ой цифре:

x0z – синтаксическая ошибка;
x1z – информационное сообщение;
x2z – ошибка в соединении (управляющем или данных);
x3z – ошибка при авторизации пользователя;
x4z – не определено;
x5z – ошибка файловой системы.

Для каждой команды FTP-сервера есть свой набор возможных кодов-откликов, которые прописаны в документе RFC 959. Для примера указаны коды отклики для некоторых команд (см. рис. 2) [1].

```
Transfer parameters
   PORT
      200
      500, 501, 421, 530
   PASV
      227
      500, 501, 502, 421, 530
   MODE
      200
      500, 501, 504, 421, 530
   TYPE
      200
      500, 501, 504, 421, 530
   STRU
      200
      500, 501, 504, 421, 530
File action commands
   ALLO
      200
      202
      500, 501, 504, 421, 530
   REST
      500, 501, 502, 421, 530
      350
   STOR
      125, 150
         (110)
         226, 250
         425, 426, 451, 551, 552
      532, 450, 452, 553
      500, 501, 421, 530
```

Рис. 2 Коды-отклики для некоторых команд

Описание работы

Для создания серверного приложения на основе протокола FTP был создан отдельный файл serverCommand.cpp, в котором хранится реализация каждой команды сервера, перечисленной в таблице 1. Для реализации путешествия по рабочему пространству, на котором был установлен FTP-сервер, были использованы bash-скрипты для перемещения по директориям, их удалением или созданием, вывода списка содержимого текущей директории, проверки существования файла и т.п с их вызовом с помощью команды system(). Результат сохраняется в специальный текстовый файл в папке temp, после чего в скором времени будет удалён.

Вся основная информация о текущем состоянии сервера: её текущем местоположении, дескрипторах сокетов для приёма команд и передачи данных, проверке на авторизацию и т.п. – хранится в специальной переменной структуры типа FTP_info.

Все отклики, которые могут использованы при выполнении команд FTP-сервера для удобства хранятся в перечисляемом типе enum server_response, где их названия соответствуют их назначению, насколько это возможно.

Для просмотра информации об отдельных файлах при вызовах команд MDTM и SIZE (см. табл. 1) была использована библиотека <sys/stat.h>, которая позволяет просматривать информацию о файле: его тип, права доступа, время изменения и создания, размер и т.д.

Получая на вход команду, сервер вызывает функцию instructionProcessor(), которая считывает переданную на вход команду и вызывает соответствующую функцию. Для команд требующих параллельной работы по каналу передачи данных: STOR, RETR – создаётся отдельный поток, который по завершению отправляет отклик по каналу команд и который может быть прерван командой ABOR (команды LIST и NLST выполняются последовательно и не могут быть прерваны) (см. табл. 1).

FTP-серверное приложение, программа которой хранится в файле server.cpp, представляет представляет собой псевдопараллельный сервер на базе протокола TCP, использующий функции файла serverCommand.cpp и библиеотеку <sys/select.h>. На вход при запуске сервера подаётся максимальное количество одновременно работающих клиентов FTP-сервера. В результате создаётся массив переменных типа FTP_info, которые обнуляются и им присваивается и идентификационный номер, с помощью которого файлы в папке temp будут отличаться по названию.

Псевдопараллельный режим работы сервера был выбран с намерением уменьшением нагрузки программы. Так как команды на сервер подаются достаточно медленно, создание параллельно работающих в режиме ожидания процессов на каждого клиента было бы лишней тратой ресурсов компьютера.

Для установления соединения между клиентом и сервером была использована библиотека <sys/socket.h>, которая была подробно изучена на протяжении курса, так что детальная информация о ней будет упущена в целях экономии времени и места.

Клиентское приложение реализовано в файле client.cpp. В нём на вход программе подаётся хост-имя компьютера, на котором работает FTP-сервер, и его порт, который не равняется порту 21 по умолчанию, так как он может быть занят другим FTP-сервером. Клиент отправляет запрос на общение с сервером и после получения отклика “220 Hello new user” начинается общение с передачей серверу команд.

С целью удобства для работы клиента в активном режиме PORT, в котором надо вводить свой IP-адрес и байты значения порта для канала передачи данных, была создана команда PORTMY, при вводе которой клиент вычисляет IP-адрес и ищет свободный порт, после чего отправляет серверу команду PORT с нужными данными.

Данное клиентское приложение реализовано с возможностью установления соединения для передачи данных между 2 серверами. Т.к. PORTMY предназначена для удобного подключения к самому клиентскому приложению, то PORT стал предназначаться для подключения активного порта к пассивному порту на другом сервере (см. рис. 3,4).

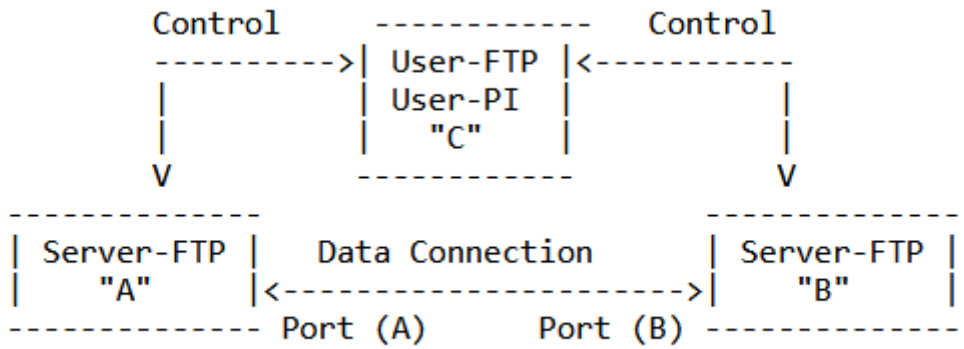

Рис. 3 Создание канала передачи данных между 2 серверами

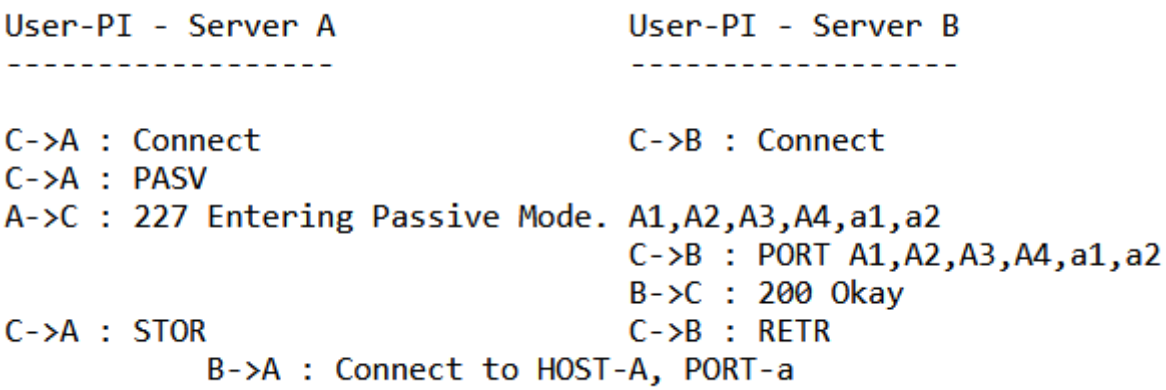

Рис. 4 Последовательность команд для создания канала данных между активным и пассивным серверами

Список используемой литературы и интернет-источников

1. Павский К. В., Ефимов А. В. Разработка сетевых приложений (протоколы TCP/IP, клиент-сервер, PCAP, Boost.ASIO): Учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск, 2018. – 80 с.
2. Request for Comments: 959 [Электронный ресурс]. URL: https://www.rfc-editor.org/rfc/rfc959 (дата обращения: 02.05.2023).

Результат работы программы

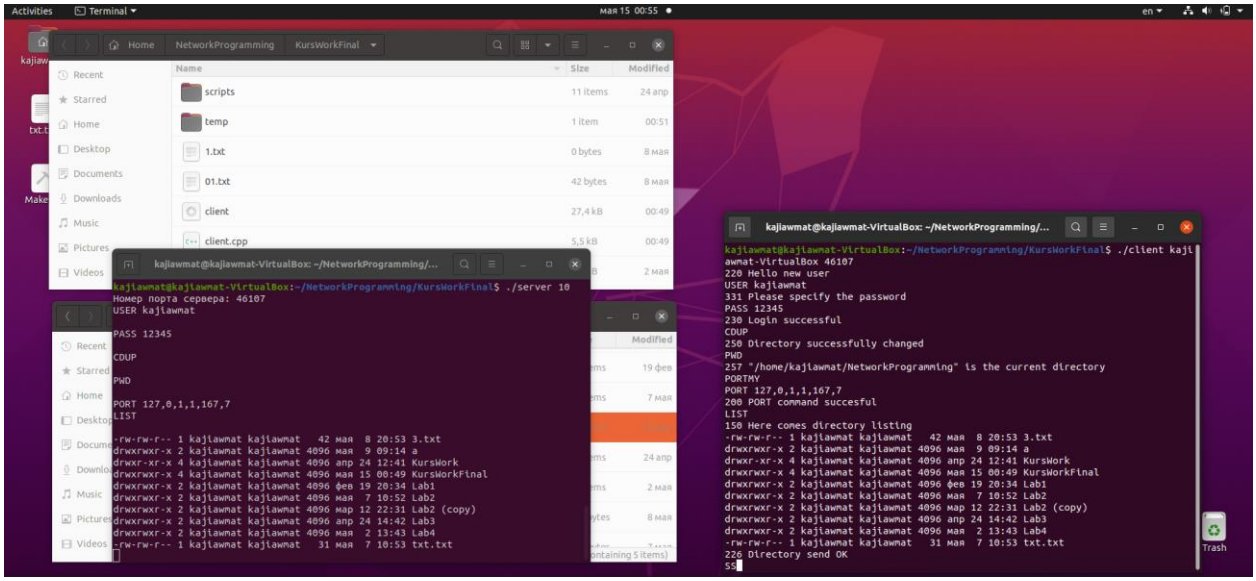

Рис. 5 Пример работы клиент-серверного приложения с передачей списка директории

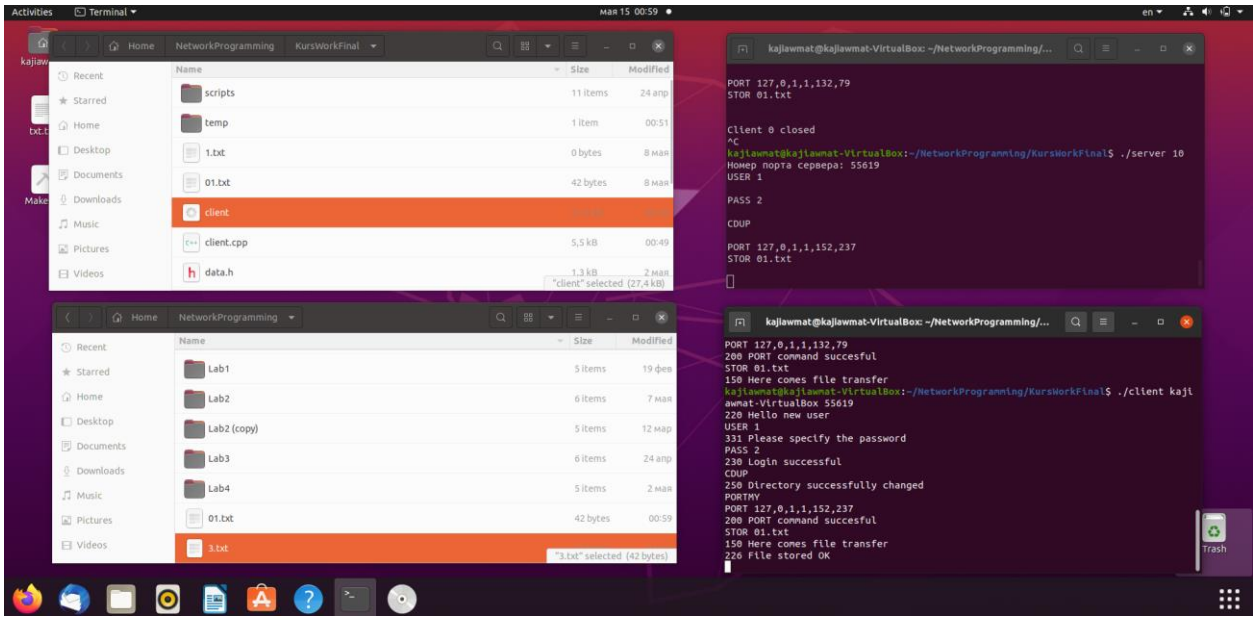

Рис. 6 Пример работы клиент-серверного приложения с передачей файла на сервер

Листинг программы

data.h:

```
#ifndef DATA_H
#define DATA_H

#include<stdio.h>
#include<stdlib.h>
#define BUFFER_SIZE 2048

enum server_response
{
	COMMENT = 110,
	TRANSFER_DELAY = 120,
	TRANSFER_START = 125,
	FILE_READY = 150,
	COMMAND_CORRECT = 200,
	SYST_INFO = 211,
	DIR_STATUS = 212,
	FILE_STATUS = 213,
	HELP_MESSAGE = 214,
	SYSTEM_TYPE = 215,
	FTP_OVERLOAD = 220,
	QUIT_SUCCESS = 221,
	CHANNEL_CREATED = 225,
	CHANNEL_CLOSED = 226,
	USER_AUTHORIZED = 230,
	REQUEST_SUCCESS = 250,
	PATH_CREATED = 257,
	PASSWORD_NEED = 331,
	AUTHORIZE_NEED = 332,
	CHANNEL_NO_USED = 421,
	CHANNEL_NO_CREATED = 425,
	TRANSFER_AUTOSTOPPED = 426,
	FILE_IS_USED = 450,
	LOCAL_ERROR = 451,
	OUT_OF_MEMORY = 452,
	COMMAND_INCORRECT = 500,
	COMMAND_ARG_INCORRECT = 501,
	COMMAND_IMPOSSIBLE = 502,
	COMMAND_ORDER_INCORRECT = 503,
	USER_NO_AUTHORIZED = 530,
	AUTHORIZED_IS_RECQUIRED = 532,
	FILE_NOT_EXISTS = 550,
	CRIT_OUT_OF_MEMORY = 552
};

enum transfer_type
{
	TYPE_A = 256,
	TYPE_I = 2
};

struct FTP_info
{
	int ID;
	int sockCommand;
	int sockData;
	bool isPasswordNeed;
	bool isAuthorized;
	bool isConnected;
	bool isAborted;
	bool isPasv;
	bool isPort;
```

```
        bool isTransfer;
        enum transfer_type typeTransfer;
        char* nameClient;
        char* IPClient;
        char* currentPath;
        char* filePath;
        char* buffPath;
};

#endif
```

serverCommand.cpp (только instructionProcessor):

```
#include "serverCommand.h"

void ZeroFTP(FTP_info &ftp)
{
	ftp.sockCommand=-1;
	ftp.sockData=-1;
	ftp.isPasswordNeed=false;
	ftp.isAuthorized=false;
	ftp.isConnected=false;
	ftp.isTransfer=false;
	ftp.isAborted=false;
	ftp.isPasv=false;
	ftp.isPort=false;
	ftp.typeTransfer=TYPE_I;
	ftp.nameClient=new char[BUFFER_SIZE];
	ftp.IPClient=new char[BUFFER_SIZE];
	ftp.currentPath=new char[BUFFER_SIZE];
	ftp.filePath=new char[BUFFER_SIZE];
	ftp.buffPath=new char[BUFFER_SIZE];
}

void* FileStore(void* _ftp)
{
	struct FTP_info* ftp=((struct FTP_info*) _ftp);
	int length;
	FILE *fp;
	char *txt=new char[BUFFER_SIZE];
	char *message=new char[BUFFER_SIZE];
	if(ftp->typeTransfer==TYPE_A)
	{
		fp=fopen(ftp->filePath,"w");
		while(true)
		{
			if(ftp->isAborted)
			{
				SendMessage(ftp->sockCommand,TRANSFER_AU-
TOSTOPPED,"Connection closed; transfer aborted");
				ftp->isAborted=false;
				ftp->isConnected=false;
				close(ftp->sockData);
				fclose(fp);
				return NULL;
			}
			bzero(message,BUFFER_SIZE);
			bzero(txt,BUFFER_SIZE);
			length=RecvData(ftp->sockData,message);
			if(length==0) break;
			strncpy(txt,message,length);
			fprintf(fp,"%s",txt);
		}
	}
	else
	{
		fp=fopen(ftp->filePath,"wb");
		while(true)
		{
			if(ftp->isAborted)
			{
				SendMessage(ftp->sockCommand,TRANSFER_AU-
TOSTOPPED,"Connection closed; transfer aborted");
				ftp->isAborted=false;
				ftp->isConnected=false;
```

```
                        close(ftp->sockData);
                        return NULL;
                }
                bzero(message,BUFFER_SIZE);
                length=RecvData(ftp->sockData,message);
                if(length==0) break;
                strncpy(txt,message,length);
                fwrite(txt,sizeof(char),length,fp);
            }
        }
        fclose(fp);
        ftp->isConnected=false;
        close(ftp->sockData);
        SendMessage(ftp->sockCommand,CHANNEL_CLOSED,"File stored OK");
        //delete message;
        //delete txt;
        return NULL;
}

void* FileRetrieve(void* _ftp)
{
        struct FTP_info* ftp=((struct FTP_info*) _ftp);
        int length;
        FILE *fp;
        char message[BUFFER_SIZE];
        if(ftp->typeTransfer==TYPE_A)
        {
            fp=fopen(ftp->filePath,"r");
            while(true)
            {
                if(ftp->isAborted)
                {
                    SendMessage(ftp->sockCommand,TRANSFER_AU-
TOSTOPPED,"Connection closed; transfer aborted");
                    ftp->isAborted=false;
                    ftp->isConnected=false;
                    close(ftp->sockData);
                    fclose(fp);
                    return NULL;
                }
                bzero(message,BUFFER_SIZE);
                if(fgets(message,BUFFER_SIZE,fp)==NULL) break;
                SendData(ftp->sockData,message);
            }
        }
        else
        {
            fp=fopen(ftp->filePath,"rb");
            while(true)
            {
                if(ftp->isAborted)
                {
                    SendMessage(ftp->sockCommand,TRANSFER_AU-
TOSTOPPED,"Connection closed; transfer aborted");
                    ftp->isAborted=false;
                    ftp->isConnected=false;
                    close(ftp->sockData);
                    return NULL;
                }
                if(feof(fp)) break;
                bzero(message,BUFFER_SIZE);
                fread(message,sizeof(char),BUFFER_SIZE,fp);
                SendData(ftp->sockData,message);
            }
```

```
	}
	fclose(fp);
	ftp->isConnected=false;
	close(ftp->sockData);
	SendMessage(ftp->sockCommand,CHANNEL_CLOSED,"File retrieved OK");
	return NULL;
}

int SendMessage(int sockClient,enum server_response num, const char txt[])
{
	char message[BUFFER_SIZE];
	snprintf(message,BUFFER_SIZE,"%i %s\n",num,txt);
	return send(sockClient,message,strlen(message),0);
}

int SendData(int sockClient, const char txt[])
{
	return send(sockClient,txt,strlen(txt),0);
}

int RecvData(int sockClient, char* &txt)
{
	return recv(sockClient,txt,BUFFER_SIZE,0);
}

void ReadFilePath(FILE* fp, char* &txt)
{
	txt=new char[BUFFER_SIZE];
	fscanf(fp,"%s",txt);
}
void ReadFileList(FILE* fp, FTP_info &ftp)
{
	char *txt=new char[BUFFER_SIZE];
	fgets(txt,BUFFER_SIZE,fp);
	while(true)
	{
		if(ftp.isAborted)
		{
			SendMessage(ftp.sockCommand,TRANSFER_AUTOSTOPPED,"Connection
closed; transfer aborted");
			ftp.isAborted=false;
			ftp.isConnected=false;
			close(ftp.sockData);
			return;
		}
		fgets(txt,BUFFER_SIZE,fp);
		if(feof(fp)) break;
		printf("%s",txt);
		SendData(ftp.sockData,txt);
	}
	ftp.isConnected=false;
	ftp.isTransfer=false;
	close(ftp.sockData);
	SendMessage(ftp.sockCommand,CHANNEL_CLOSED,"Directory send OK");
}

void ReadFileHelp(FILE* fp,char* &txt)
{
	txt=new char[BUFFER_SIZE];
	char temp[BUFFER_SIZE];
	while(true)
	{
		if(fgets(temp,BUFFER_SIZE,fp)==NULL) break;
		strcat(txt,temp);
```

```
	}
}

void Abort(FTP_info &ftp)
{
	if(ftp.isConnected)
	{
		ftp.isAborted=true;
		while(ftp.isConnected);
		SendMessage(ftp.sockCommand,CHANNEL_CLOSED,"Channel data success-
fully aborted");
	}
	else
	{
		SendMessage(ftp.sockCommand,COMMAND_IMPOSSIBLE,"Channel data was-
n't created");
	}
}
void CurrDirUp(FTP_info &ftp)
{
	char com[BUFFER_SIZE];
	snprintf(com,BUFFER_SIZE,"./scripts/Currdirup.sh %i %s",ftp.ID,ftp.cur-
rentPath);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	ReadFilePath(fp,txt);
	strcpy(ftp.currentPath,txt);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	//delete txt;
	SendMessage(ftp.sockCommand,REQUEST_SUCCESS,"Directory successfully
changed");
}
void CurrWorkDir(FTP_info &ftp, char command[])
{
	char com[BUFFER_SIZE];
	char *savePtr;
	char *path=strtok_r(command," ",&savePtr);
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
new directory");
		return;
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Existsdir.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	FILE *fp=fopen(ftp.buffPath,"r");
	if(fp==NULL)
	{
		SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Failed to change di-
rectory");
		return;
	}
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"./scripts/Currworkdir.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	char *txt;
	fp=fopen(ftp.buffPath,"r");
	ReadFilePath(fp,txt);
```

```
	strcpy(ftp.currentPath,txt);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	SendMessage(ftp.sockCommand,REQUEST_SUCCESS,"Directory successfully
changed");
}
void DelFile(FTP_info &ftp, char command[])
{
	char com[BUFFER_SIZE];
	char *savePtr;
	char *path=strtok_r(command," ",&savePtr);
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
deleting file");
		return;
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Existsfile.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	if(fp==NULL)
	{
		SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Failed to //delete
file");
		return;
	}
	ReadFilePath(fp,txt);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"./scripts/Delfile.sh %s",txt);
	system(com);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	//delete txt;
	SendMessage(ftp.sockCommand,REQUEST_SUCCESS,"File successfully //de-
leted");
}
void Help(FTP_info &ftp)
{
	char* txt;
	FILE *fp=fopen("Help.txt","r");
	if(fp==NULL)
	{
		SendMessage(ftp.sockCommand,LOCAL_ERROR,"Help.txt not found");
		return;
	}
	ReadFileHelp(fp,txt);
	SendData(ftp.sockCommand,txt);
	SendMessage(ftp.sockCommand,HELP_MESSAGE,"Help OK");
	fclose(fp);
	//delete txt;
}
void List(FTP_info &ftp)
{
	if(ftp.isConnected==false)
	{
		SendMessage(ftp.sockCommand,CHANNEL_NO_CREATED,"Channel data not
created");
		return;
	}
	if(ftp.isTransfer)
```

```
	{
		SendMessage(ftp.sockCommand,TRANSFER_START,"Data connection al-
ready open; transfer starting");
		return;
	}
	char com[BUFFER_SIZE];
	snprintf(com,BUFFER_SIZE,"./scripts/List.sh %i %s",ftp.ID,ftp.cur-
rentPath);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	SendMessage(ftp.sockCommand,FILE_READY,"Here comes directory listing");
	ReadFileList(fp,ftp);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	////delete txt;
}
void ModDateTime(FTP_info &ftp, char command[])
{
	char com[BUFFER_SIZE];
	char *savePtr;
	char *path=strtok_r(command," ",&savePtr);
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
existing filepath");
		return;
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Existsfile.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	if(fp==NULL)
	{
		SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Could not get file
modification time");
		return;
	}
	ReadFilePath(fp,txt);
	fclose(fp);
	struct stat buf;
	stat(txt,&buf);
	char message[BUFFER_SIZE];
	snprintf(message,BUFFER_SIZE,"%s",ctime(&buf.st_mtime));
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	SendMessage(ftp.sockCommand,FILE_STATUS,message);
	//delete txt;
}
void MakeDir(FTP_info &ftp, char command[])
{
	char com[BUFFER_SIZE];
	char *savePtr;
	char *path=strtok_r(command," ",&savePtr);
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
filepath to new directory");
		return;
	}
```

```
	snprintf(com,BUFFER_SIZE,"./scripts/Existsdir.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	FILE *fp=fopen(ftp.buffPath,"r");
	if(fp!=NULL)
	{
		SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Directory already ex-
ists");
		fclose(fp);
		snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
		system(com);
		return;
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Makedir.sh %s %s",ftp.cur-
rentPath,path);
	system(com);
	SendMessage(ftp.sockCommand,PATH_CREATED,"Directory created");
}
void NameList(FTP_info &ftp)
{
	if(ftp.isConnected==false)
	{
		SendMessage(ftp.sockCommand,CHANNEL_NO_CREATED,"Channel data not
created");
		return;
	}
	if(ftp.isTransfer)
	{
		SendMessage(ftp.sockCommand,TRANSFER_START,"Data connection al-
ready open; transfer starting");
		return;
	}
	char com[BUFFER_SIZE];
	snprintf(com,BUFFER_SIZE,"./scripts/Namelist.sh %i %s",ftp.ID,ftp.cur-
rentPath);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	SendMessage(ftp.sockCommand,FILE_READY,"Here comes directory listing");
	ReadFileList(fp,ftp);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	////delete txt;
}

void Noop(FTP_info &ftp)
{
	SendMessage(ftp.sockCommand,COMMAND_CORRECT,"Noop successful");
}
void InPass(FTP_info &ftp, char command[])
{
	if(ftp.isAuthorized)
	{
		SendMessage(ftp.sockCommand,COMMAND_ORDER_INCORRECT,"You already
logged in");
		return;
	}
	char *savePtr;
	char *pass=strtok_r(command," ",&savePtr);
	pass=strtok_r(NULL," ",&savePtr);
	if(pass==NULL)
	{
```

```
            SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"Please input
password");
            return;
      }
      if(ftp.isPasswordNeed==false)
      {
            SendMessage(ftp.sockCommand,COMMAND_ORDER_INCORRECT,"You should
use command USER first");
            return;
      }
      ftp.isAuthorized=true;
      ftp.isPasswordNeed=false;
      SendMessage(ftp.sockCommand,USER_AUTHORIZED,"Login successful");
}
void ActMode(FTP_info &ftp, char command[])
{
      char *savePtr,*savePtr1;
      char *adr=strtok_r(command," ",&savePtr);
      char *temp;
      struct sockaddr_in dataAddr;
      int numbers[6];
      adr=strtok_r(NULL," ",&savePtr);
      if(adr==NULL)
      {
            SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"Please input
IP-adress and port");
            return;
      }
      //printf("%s\n",adr);
      temp=strtok_r(adr,",",&savePtr1);
      try
      {
            for(int i=0;i<6;i++)
            {
                  sscanf(temp,"%i",&(numbers[i]));
                  temp=strtok_r(NULL,",",&savePtr1);
            }
      }
      catch(...)
      {
            SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
corrrect IP-adress and port");
            return;
      }
      for(int i=0;i<6;i++) if(numbers[i]<0 || numbers[i]>255)
      {
            SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
corrrect IP-adress and port");
            return;
      }
      bzero((char*) &dataAddr, sizeof(dataAddr));
      dataAddr.sin_family=AF_INET;
      snprintf(ftp.IPClient,BUFFER_SIZE,"%i.%i.%i.%i",numbers[0],num-
bers[1],numbers[2],numbers[3]);
      inet_aton(ftp.IPClient,&(dataAddr.sin_addr));
      dataAddr.sin_port=htons(numbers[4]*256+numbers[5]);
      ftp.sockData=socket(AF_INET, SOCK_STREAM, 0);
      if(connect(ftp.sockData,(sockaddr*)&dataAddr,sizeof(dataAddr))<0)
      {
            printf("Connect Error");
            exit(1);
      }
      ftp.isConnected=true;
      SendMessage(ftp.sockCommand,COMMAND_CORRECT,"PORT command succesful");
```

```
}
void PresWorkDir(FTP_info &ftp)
{
      char message[BUFFER_SIZE];
      snprintf(message,BUFFER_SIZE,"\"%s\" is the current directory",ftp.cur-
rentPath);
      SendMessage(ftp.sockCommand,PATH_CREATED,message);
}
void Quit(FTP_info &ftp)
{
      if(ftp.isTransfer) Abort(ftp);
      SendMessage(ftp.sockCommand,QUIT_SUCCESS,"Goodbye Client.");
      ZeroFTP(ftp);
}

void Retrieve(FTP_info &ftp, char command[])
{
      if(ftp.isConnected==false)
      {
            SendMessage(ftp.sockCommand,CHANNEL_NO_CREATED,"Channel data not
created");
            return;
      }
      if(ftp.isTransfer)
      {
            SendMessage(ftp.sockCommand,TRANSFER_START,"Data connection al-
ready open; transfer starting");
            return;
      }
      char *savePtr;
      char *name=strtok_r(command," \n",&savePtr);
      name=strtok_r(NULL," \n",&savePtr);
      if(name==NULL)
      {
            SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
file for store on server");
            return;
      }
      snprintf(ftp.filePath,BUFFER_SIZE,"%s/%s",ftp.currentPath,name);
      FILE *fp=fopen(ftp.filePath,"r");
      if(fp==NULL)
      {
            SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Failed to retrieve
file (doesn't exist)");
            return;
      }
      fclose(fp);
      SendMessage(ftp.sockCommand,FILE_READY,"Here comes file transfer");
      pthread_t pth_retrieve;
      pthread_create(&pth_retrieve,NULL,&FileRetrieve,(void*) &ftp);
      pthread_detach(pth_retrieve);
}

void RemoveDir(FTP_info &ftp, char command[])
{
      char com[BUFFER_SIZE];
      char *savePtr;
      char *path=strtok_r(command," ",&savePtr);
      path=strtok_r(NULL," ",&savePtr);
      if(path==NULL)
      {
            SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
filepath to new directory");
            return;
```

```
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Existsdir.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	if(fp==NULL)
	{
		SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Directory not ex-
ists");
		return;
	}
	ReadFilePath(fp,txt);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"./scripts/Removedir.sh %s",txt);
	system(com);
	//delete txt;
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
}
void ReName(FTP_info &ftp, char command[])
{
	char com[BUFFER_SIZE];
	char *savePtr;
	char *path=strtok_r(command," ",&savePtr);
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
old filename");
		return;
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Existsfile.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	char *txt1;
	FILE *fp=fopen(ftp.buffPath,"r");
	if(fp==NULL)
	{
		SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Failed to rename file
(doesn't exist)");
		return;
	}
	ReadFilePath(fp,txt1);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL || strncmp(path,"RNTO",4)!=0)
	{
		SendMessage(ftp.sockCommand,COMMAND_INCORRECT,"Command RNTO ex-
pected");
		return;
	}
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
new filename");
		return;
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Rename.sh %s %s %s",ftp.cur-
rentPath,txt1,path);
	system(com);
```

```
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	SendMessage(ftp.sockCommand,REQUEST_SUCCESS,"File successfully re-
named");
}
void FileSize(FTP_info &ftp, char command[])
{
	char com[BUFFER_SIZE];
	char *savePtr;
	char *path=strtok_r(command," ",&savePtr);
	path=strtok_r(NULL," ",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
existing filepath");
		return;
	}
	snprintf(com,BUFFER_SIZE,"./scripts/Existsfile.sh %i %s
%s",ftp.ID,ftp.currentPath,path);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	if(fp==NULL)
	{
		SendMessage(ftp.sockCommand,FILE_NOT_EXISTS,"Could not get file
size in bytes");
		return;
	}
	ReadFilePath(fp,txt);
	fclose(fp);
	struct stat buf;
	stat(txt,&buf);
	char message[BUFFER_SIZE];
	snprintf(message,BUFFER_SIZE,"%li B",buf.st_size);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	SendMessage(ftp.sockCommand,FILE_STATUS,message);
	//delete txt;
}
void Store(FTP_info &ftp, char command[])
{
	if(ftp.isConnected==false)
	{
		SendMessage(ftp.sockCommand,CHANNEL_NO_CREATED,"Channel data not
created");
		return;
	}
	if(ftp.isTransfer)
	{
		SendMessage(ftp.sockCommand,TRANSFER_START,"Data connection al-
ready open; transfer starting");
		return;
	}
	char *savePtr,*savePtr1;
	char *path=strtok_r(command," \n",&savePtr);
	path=strtok_r(NULL," \n",&savePtr);
	if(path==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
file for store on server");
		return;
	}
	char *name, *next_token;
	name=strtok_r(path,"/",&savePtr1);
```

```
	next_token=strtok_r(NULL,"/",&savePtr1);
	while(next_token!=NULL)
	{
		name=next_token;
		next_token=strtok_r(NULL,"/",&savePtr1);
	}
	snprintf(ftp.filePath,BUFFER_SIZE,"%s/%s",ftp.currentPath,name);
	FILE *fp;
	int ind=0;
	do
	{
		fp=fopen(ftp.filePath,"r");
		if(fp==NULL) break;
		fclose(fp);
		snprintf(ftp.filePath,BUFFER_SIZE,"%s/%i%s",ftp.cur-
rentPath,ind,name);
		ind++;
	}while(true);
	SendMessage(ftp.sockCommand,FILE_READY,"Here comes file transfer");
	pthread_t pth_store;
	pthread_create(&pth_store,NULL,&FileStore,(void*) &ftp);
	pthread_detach(pth_store);
}
void SystemInfo(FTP_info &ftp)
{
	char com[BUFFER_SIZE];
	snprintf(com,BUFFER_SIZE,"./scripts/Systeminfo.sh %i",ftp.ID);
	system(com);
	char *txt;
	FILE *fp=fopen(ftp.buffPath,"r");
	ReadFilePath(fp,txt);
	SendMessage(ftp.sockCommand,SYSTEM_TYPE,txt);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",ftp.ID);
	system(com);
	//delete txt;
}
void TypeTransfer(FTP_info &ftp, char command[])
{
	char *savePtr;
	char *type=strtok_r(command," ",&savePtr);
	type=strtok_r(NULL," ",&savePtr);
	if(type==NULL)
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"PLease input
type transfer");
		return;
	}
	if(strncmp(type,"I",1)==0) ftp.typeTransfer=TYPE_I;
	else if(strncmp(type,"A",1)==0) ftp.typeTransfer=TYPE_A;
	else
	{
		SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"This type not
exists");
		return;
	}
	SendMessage(ftp.sockCommand,COMMAND_CORRECT,"Type transfer change suc-
cessful");
}
void InUser(FTP_info &ftp, char command[])
{
	if(ftp.isAuthorized)
	{
```

```
            SendMessage(ftp.sockCommand,USER_NO_AUTHORIZED,"Can't change to
another user");
            return;
      }
      char *savePtr;
      char *name=strtok_r(command," ",&savePtr);
      name=strtok_r(NULL," ",&savePtr);
      if(name==NULL)
      {
            SendMessage(ftp.sockCommand,COMMAND_ARG_INCORRECT,"Syntax error in
parameters or arguments");
            return;
      }
      strcpy(ftp.nameClient,name);
      ftp.isPasswordNeed=true;
      ftp.isAuthorized=false;
      SendMessage(ftp.sockCommand,PASSWORD_NEED,"Please specify the pass-
word");

}
void UnNamedFunction(FTP_info &ftp)
{
      SendMessage(ftp.sockCommand,COMMAND_INCORRECT,"Syntax error, command
inrecognized\nPlease use HELP");
}

void instructionProcessor(FTP_info &ftp, char command[])
{
      if(strncmp(command,"HELP",4)==0)
      {
            Help(ftp);
            return;
      }
      if(ftp.isPasswordNeed)
      {
            if(strncmp(command,"PASS",4)!=0) SendMessage(ftp.sockCommand,COM-
MAND_ORDER_INCORRECT,"Please, input password to log in");
            else InPass(ftp,command);
            return;
      }
      if(ftp.isAuthorized==false && strncmp(command,"USER",4)!=0)
      {
            SendMessage(ftp.sockCommand,USER_NO_AUTHORIZED,"Please log in with
USER and PASS");
            return;
      }
      if(strncmp(command,"ABOR",4)==0)          Abort(ftp);
      else if(strncmp(command,"CDUP",4)==0)     CurrDirUp(ftp);
      else if(strncmp(command,"CWD",3)==0)      CurrWorkDir(ftp,command);
      else if(strncmp(command,"DELE",4)==0)     DelFile(ftp,command);
      else if(strncmp(command,"LIST",4)==0)     List(ftp);
      else if(strncmp(command,"MDTM",4)==0)     ModDateTime(ftp,command);
      else if(strncmp(command,"MKD",3)==0)      MakeDir(ftp,command);
      else if(strncmp(command,"NLST",4)==0)     NameList(ftp);
      else if(strncmp(command,"NOOP",4)==0)     Noop(ftp);
      else if(strncmp(command,"PORT",4)==0)     ActMode(ftp,command);
      else if(strncmp(command,"PWD",3)==0)      PresWorkDir(ftp);
      else if(strncmp(command,"QUIT",4)==0)     Quit(ftp);
      else if(strncmp(command,"RETR",4)==0)     Retrieve(ftp,command);
      else if(strncmp(command,"RMD",3)==0)      RemoveDir(ftp,command);
      else if(strncmp(command,"RNFR",4)==0)     ReName(ftp,command);
      else if(strncmp(command,"SIZE",4)==0)     FileSize(ftp,command);
      else if(strncmp(command,"STOR",4)==0)     Store(ftp,command);
      else if(strncmp(command,"SYST",4)==0)     SystemInfo(ftp);
```

```
        else if(strncmp(command,"TYPE",4)==0)     TypeTransfer(ftp,command);
        else if(strncmp(command,"USER",4)==0)     InUser(ftp,command);
        else                                      UnNamedFunction(ftp);
}
```

server.cpp:

```
#include "serverCommand.h"

struct FTP_info *ftpA;
char* path;

using namespace std;

int main(int argc, char* argv[])
{
	if(argc<2)
	{
		printf("Too few arguments");
		exit(1);
	}
	fd_set rds,ads;
	int ind=-1;
	int MainsockFTP;
	struct sockaddr_in serverAddr, clientAddr;
	unsigned int Length,N;
	char message[BUFFER_SIZE];
	int msgLength;

	int Size=atoi(argv[1]);
	char com[BUFFER_SIZE];
	snprintf(com,BUFFER_SIZE,"./scripts/Existsdir.sh %i . .",Size);
	system(com);
	snprintf(com,BUFFER_SIZE,"temp/%i.txt",Size);
	FILE *fp=fopen(com,"r");
	ReadFilePath(fp,path);
	fclose(fp);
	snprintf(com,BUFFER_SIZE,"rm temp/%i.txt",Size);
	system(com);

	ftpA=new struct FTP_info[Size];
	for(int i=0;i<Size;i++)
	{
		ftpA[i].ID=i;
		ZeroFTP(ftpA[i]);
		snprintf(ftpA[i].buffPath,BUFFER_SIZE,"temp/%i.txt",ftpA[i].ID);
		strcpy(ftpA[i].currentPath,path);
	}

	MainsockFTP=socket(AF_INET, SOCK_STREAM, 0);
	bzero((char*) &serverAddr, sizeof(serverAddr));
	serverAddr.sin_family=AF_INET;
	serverAddr.sin_addr.s_addr=htonl(INADDR_ANY);
	serverAddr.sin_port=0;
	if(bind(MainsockFTP,(sockaddr*) &serverAddr,sizeof(serverAddr)))
	{
		printf("Bind error");
		exit(1);
	}
	Length=sizeof(serverAddr);
	if(getsockname(MainsockFTP,(sockaddr*) &serverAddr,&Length))
	{
		printf("getsockname() error");
		exit(1);
	}
	listen(MainsockFTP,5);
	cout<<"Номер порта сервера: "<<ntohs(serverAddr.sin_port)<<endl;
	N=getdtablesize();
	FD_ZERO(&ads);
	FD_SET(MainsockFTP,&ads);
```

```
	while(1)
	{
		memcpy(&rds,&ads,sizeof(rds));
		if(select(N,&rds,(fd_set*)0,(fd_set*)0,(struct timeval*)0)<0)
		{
			printf("select error");
			exit(1);
		}
		if(FD_ISSET(MainsockFTP,&rds))
		{
			ind=-1;
			for(int i=0;i<Size;i++) if(ftpA[i].sockCommand==-1)
			{
				ind=i;
				break;
			}
			if(ind!=-1)
			{
				if((ftpA[ind].sockCommand=accept(MainsockFTP,(sock-
addr*) &clientAddr,&Length))<0)
				{
					printf("accept() error");
					exit(1);
				}
				SendMessage(ftpA[ind].sockCommand,FTP_OVERLOAD,"Hello
new user");
				FD_SET(ftpA[ind].sockCommand,&ads);
			}
		}
		for(int i=0;i<N;i++) if(i!=MainsockFTP && FD_ISSET(i,&rds))
		{
			bzero(message,BUFFER_SIZE);
			msgLength=recv(i,message,BUFFER_SIZE,0);
			message[msgLength-2]='\0';
			printf("%s\n",message);
			if(msgLength<0)
			{
				printf("recvfrom() error");
				exit(1);
			}
			else if(msgLength==0)
			{
				for(int k=0;k<Size;k++)
				{
					if(ftpA[k].sockCommand==i)
					{
						ZeroFTP(ftpA[k]);
						snprintf(ftpA[k].buff-
Path,BUFFER_SIZE,"temp/%i.txt",ftpA[k].ID);
						strcpy(ftpA[k].currentPath,path);
						printf("Client %i closed\n", ftpA[k].ID);
					}
				}
				close(i);
				FD_CLR(i,&ads);
				break;
			}
			for(int k=0;k<Size;k++) if(ftpA[k].sockCommand==i)
			{
				instructionProcessor(ftpA[k], message);
				break;
			}
		}
```

```
        }
        close(MainsockFTP);
}
```

client.cpp:

```
#include "data.h"
#include<stdio.h>
#include<iostream>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<unistd.h>
#include<string.h>
#include<pthread.h>

using namespace std;

char hostname[BUFFER_SIZE];
char filename[BUFFER_SIZE];
int sockCommand;
int sockData=-1;
int tempSockData=-1;
bool requestPORT=true;
bool localPORT=false;

void* RecvFile(void* file)
{
	char data[BUFFER_SIZE];
	if(localPORT)
	{
		int len;
		FILE* fp=fopen(filename,"w");
		char txt[BUFFER_SIZE];
		while(true)
		{
			bzero(data,BUFFER_SIZE);
			len=recv(sockData,data,BUFFER_SIZE,0);
			if(len==0) break;
			strncpy(txt,data,len);
			fprintf(fp,"%s",txt);
		}
		fclose(fp);
		close(sockData);
	}
	sockData=-1;
	bzero(data,BUFFER_SIZE);
	recv(sockCommand,data,BUFFER_SIZE,0);
	cout<<data;
	return NULL;
}

void* SendFile(void* file)
{
	char data[BUFFER_SIZE];
	if(localPORT)
	{
		FILE* fp=fopen(filename,"r");
		while(true)
		{
			bzero(data,BUFFER_SIZE);
			if(fgets(data,BUFFER_SIZE,fp)==NULL) break;
			if(send(sockData,data,strlen(data),0)<=0) break;

		}
		fclose(fp);
```

```
            close(sockData);
      }
      sockData=-1;
      bzero(data,BUFFER_SIZE);
      recv(sockCommand,data,BUFFER_SIZE,0);
      cout<<data;
      return NULL;
}

void* RecvList(void* unused)
{
      int len;
      char response[BUFFER_SIZE];
      if(localPORT)
      {
            while(true)
            {
                  bzero(response,BUFFER_SIZE);
                  len=recv(sockData,response,BUFFER_SIZE,0);
                  if(len==0) break;
                  cout<<response;
            }
            close(sockData);
      }
      sockData=-1;
      bzero(response,BUFFER_SIZE);
      len=recv(sockCommand,response,BUFFER_SIZE,0);
      cout<<response;
      return NULL;
}

void* ListenSock(void* unused)
{
      struct sockaddr_in clientAddr;
      unsigned int Length=sizeof(clientAddr);
      int passSock=tempSockData;
      tempSockData=accept(passSock,(sockaddr*) &clientAddr,&Length);
      close(passSock);
      return NULL;
}

int ResponseActions(char message[], char response[])
{
      int len;
      pthread_t pth;
      char *typeCommand;
      char *name;
      char *savePtr1;
      char *savePtr2;
      typeCommand=strtok_r(message," ",&savePtr1);
      if(strncmp(typeCommand,"HELP",4)==0)
      {
            bzero(response,BUFFER_SIZE);
            len=recv(sockCommand,response,BUFFER_SIZE,0);
            cout<<response;
            if(strncmp(strtok_r(response," ",&savePtr2),"451",3)==0) return
len;
            bzero(response,BUFFER_SIZE);
            len=recv(sockCommand,response,BUFFER_SIZE,0);
            cout<<response;
            return len;
      }
      if(strncmp(typeCommand,"PORT",4)==0)
      {
```

```
            len=recv(sockCommand,response,BUFFER_SIZE,0);
            cout<<response;
            if(strncmp(strtok_r(response," ",&savePtr2),"200",3)!=0) re-
questPORT=false;
            else sockData=tempSockData;
            return len;
      }
      name=strtok_r(NULL," \n",&savePtr1);
      len=recv(sockCommand,response,BUFFER_SIZE,0);
      cout<<response;
      if(((strncmp(typeCommand,"LIST",4)==0) || (strncmp(typeCom-
mand,"NLST",4)==0)) && (strncmp(response,"150",3)==0))
      {
            pthread_create(&pth,NULL,RecvList,NULL);
            pthread_detach(pth);
      }
      if((strncmp(typeCommand,"STOR",4)==0) && (strncmp(response,"150",3)==0))
      {
            bzero(filename,BUFFER_SIZE);
            strcpy(filename,name);
            pthread_create(&pth,NULL,SendFile,NULL);
            pthread_detach(pth);
      }
      if((strncmp(typeCommand,"RETR",4)==0) && (strncmp(response,"150",3)==0))
      {
            bzero(filename,BUFFER_SIZE);
            strcpy(filename,name);
            pthread_create(&pth,NULL,RecvFile,NULL);
            pthread_detach(pth);
      }
      if(strncmp(typeCommand,"QUIT",4)==0) return 0;
      return len;
}

int main(int argc, char* argv[])
{
      int msgLength;
      pthread_t pth;
      struct sockaddr_in serverAddr,clientAddr;
      struct hostent *hp;
      unsigned int Length;
      char IPData[20];
      int portData;
      char message[BUFFER_SIZE];
      char response[BUFFER_SIZE];
      if(argc<3)
      {
            printf("Too few arguments");
            exit(1);
      }
      system("hostname > Host.txt");
      FILE* fp=fopen("Host.txt","r");
      fscanf(fp,"%s",hostname);
      fclose(fp);
      system("rm Host.txt");
      sockCommand=socket(AF_INET, SOCK_STREAM, 0);
      if(sockCommand<0)
      {
            printf("Open UDPsocket error");
            exit(1);
      }

      bzero((char*) &serverAddr, sizeof(serverAddr));
      serverAddr.sin_family=AF_INET;
```

```
	hp=gethostbyname(argv[1]);
	bcopy(hp->h_addr, &serverAddr.sin_addr,hp->h_length);
	serverAddr.sin_port=htons(atoi(argv[2]));
	if(connect(sockCommand,(sockaddr*)&serverAddr,sizeof(serverAddr))<0)
	{
		printf("Connect Error");
		exit(1);
	}
	bzero(response,BUFFER_SIZE);
	recv(sockCommand,response,BUFFER_SIZE,0);
	cout<<response;
	while(true)
	{
		bzero(message,BUFFER_SIZE);
		bzero(response,BUFFER_SIZE);
		fgets(message,BUFFER_SIZE,stdin);
		if(strncmp(message,"PORTMY",6)==0)
		{
			tempSockData=socket(AF_INET, SOCK_STREAM, 0);
			bzero((char*) &clientAddr, sizeof(serverAddr));
			clientAddr.sin_family=AF_INET;
			hp=gethostbyname(hostname);
			bcopy(hp->h_addr, &clientAddr.sin_addr,hp->h_length);
			clientAddr.sin_port=0;
			if(bind(tempSockData,(sockaddr*) &clientAddr,sizeof(clien-
tAddr)))
			{
				printf("Bind error\n");
				break;
			}
			Length=sizeof(clientAddr);
			if(getsockname(tempSockData,(sockaddr*) &clien-
tAddr,&Length))
			{
				printf("getsockname() error");
				exit(1);
			}
			snprintf(IPData,20,"%s",inet_ntoa(serverAddr.sin_addr));
			for(int i=0;i<20;i++) if(IPData[i]=='.') IPData[i]=',';
			portData=ntohs(clientAddr.sin_port);
			snprintf(message,BUFFER_SIZE,"PORT
%s,%i,%i",IPData,portData/256,portData%256);
			cout<<message<<endl;
			listen(tempSockData,2);
			pthread_create(&pth,NULL,&ListenSock,NULL);
			localPORT=true;
		}
		else if(strncmp(message,"PORT",4)==0) localPORT=false;
		strcat(message," \n");
		if(send(sockCommand,message,strlen(message),0)<0)
		{
			printf("send() error");
			exit(1);
		}
		msgLength=ResponseActions(message,response);
		if(msgLength==0)
		{
			printf("Client closed\n");
			close(sockCommand);
			break;
		}
		if(!requestPORT)
		{
			pthread_cancel(pth);
```

```
                requestPORT=true;
            }
        }
        close(sockCommand);
        return 0;

}
```