



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине: «Разработка программных систем»

Студент Гусаров Аркадий Андреевич

Группа РК6-63Б

Тип задания Лабораторная работа №3

Вариант лабораторной работы 12

Студент _____ **Гусаров А.А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Федорук В.Г.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2022 г.

ОГЛАВЛЕНИЕ

Текст задания	3
Описание прикладного протокола сетевого взаимодействия	3
Описание структуры программы.....	3
Блок-схема.....	6
Пример работы программы.....	8
Текст программы.....	9

Текст задания

Разработать программу "мини-ftp-клиент", обеспечивающую передачу файла (команда *put*) на удаленный ftp-сервер в пассивном режиме.

Описание прикладного протокола сетевого взаимодействия

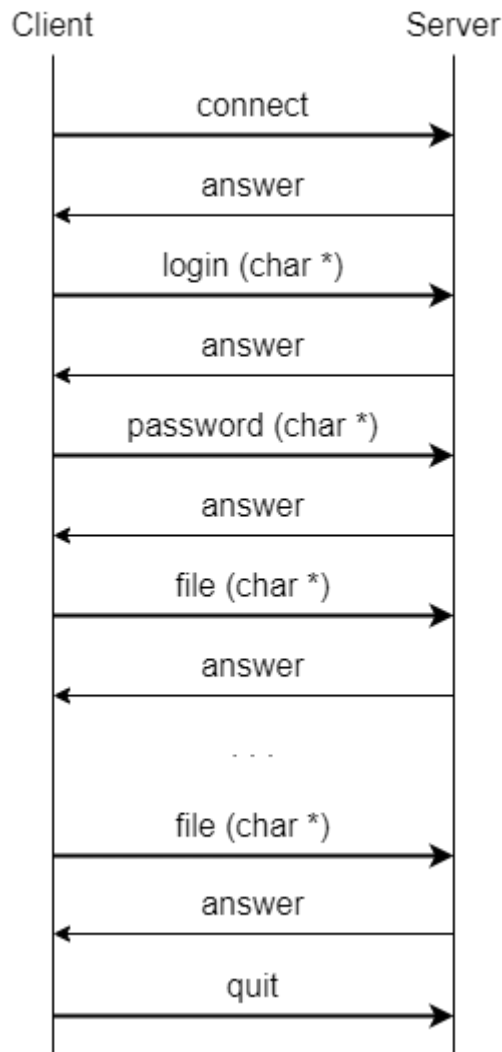


Рис. 1. Описание прикладного протокола сетевого взаимодействия

Описание структуры программы

В программе используются следующие структуры данных:

- `struct Connection {int sock; struct addrinfo *info;}` – структура "подключение". Содержит поля сокета и информации о подключении.
- `typedef void (*Commands) (struct Connection *)`

Commands masOfCommands[] – массив команд.

- *char buf[1024]* – буфер ввода / вывода.

С помощью функции *cycle()*, реализованной в программе, в цикле происходит чтение команд пользователя:

- *open <host>* – создание соединения с FTP-сервером.
- *cd* – переход в нужную директорию на сервере (/pub/htdocs).
- *put <filename>* – загрузка файла по указанному пути на FTP-сервер.
- *quit* – выход из программы.

Всем этим командам присваивается определенный индекс при занесении в массив *masOfCommands*. В зависимости от введенной пользователем команды программа определяет какую соответствующую функцию вызвать.

Таблица 1. Массив команд

	<i>masOfCommands[]</i>			
Индекс	0	1	2	3
Команда	<i>quit</i>	<i>open</i>	<i>cd</i>	<i>put</i>

Загрузка файла на FTP-сервер начинается с открытия файла в исходном формате. Передача файла производится в потоковом режиме, структура файла воспринимается как непрерывный поток байтов, а тип файла устанавливается в двоичный (TYPE I). Если при открытии файла возникла ошибка, программа завершает текущую команду и ожидает ввода следующей, иначе – начинается передача файла на сервер по следующей схеме:

- В цикле осуществляется чтение очередной строки файла в буфер.
- Содержимое буфера с помощью функции *send()* передается на сокет соединения передачи данных FTP-сервера.

- Как только достигнут конец файла, цикл завершается.
- Закрытие файла.
- Закрытие сокета.
- Освобождение памяти, выделенной под структуру *Connection*.

Блок-схема

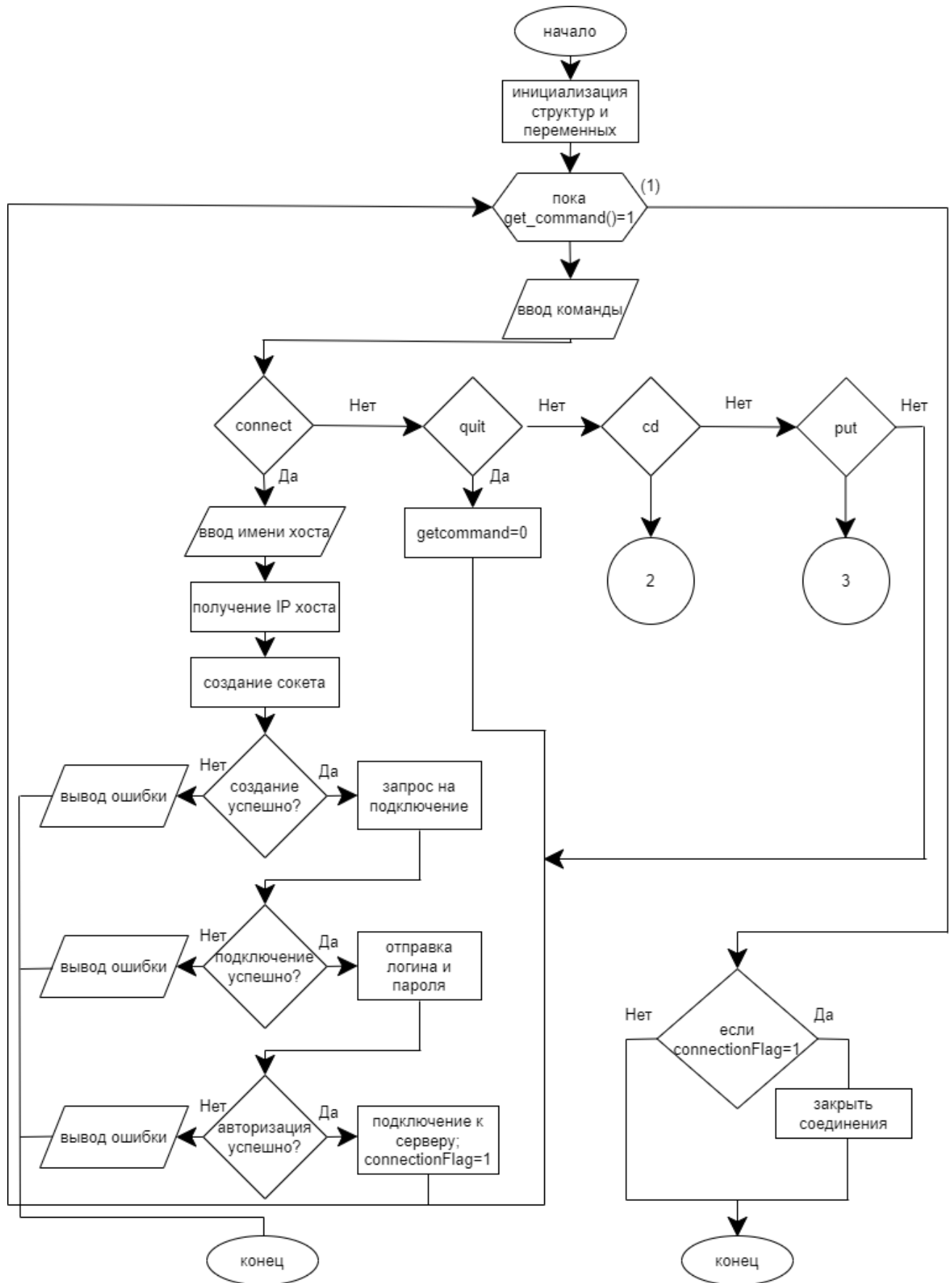


Рис. 2. Блок-схема основной программы

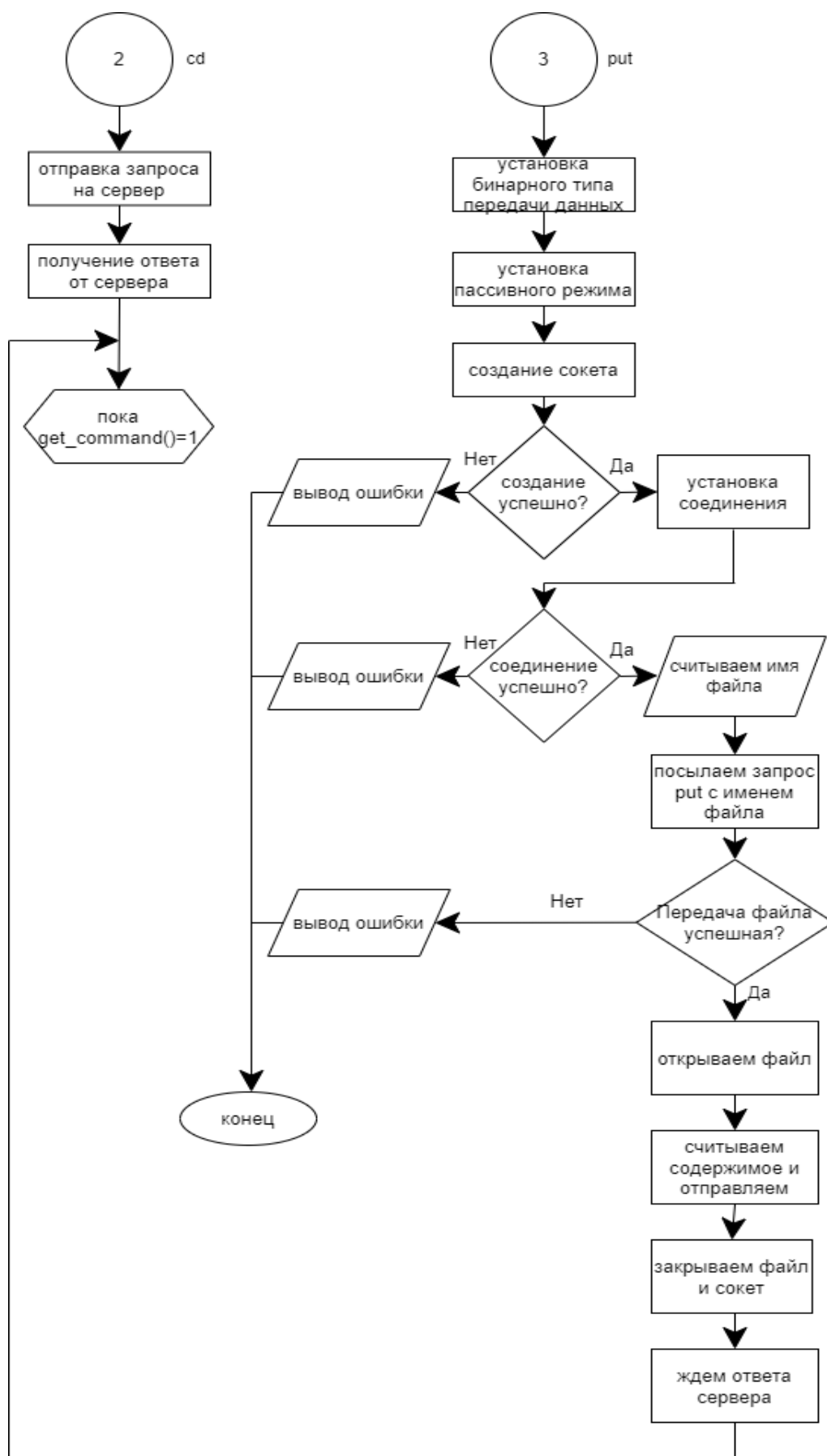


Рис. 3. Блок-схема функций *cd* и *put*

Пример работы программы

```
ftp> open rk6lab.bmstu.ru
connection port:21
220 bigor.bmstu.ru FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.

Write your login: rk6stud
331 Password required for rk6stud.

Write your password: rk6stud
230 User rk6stud logged in.

ftp> cd
CWD /pub/htdocs
250 CWD command successful.

ftp> put example.txt
200 Type set to I.

227 Entering Passive Mode (195,19,40,252,228,18)

connection port:58386
150 Opening BINARY mode data connection for 'example.txt'.

226 Transfer complete.

ftp> quit
Goodbye
```

Рис. 4. Примеры работы программы



Рис. 5. Содержимое файла example.txt

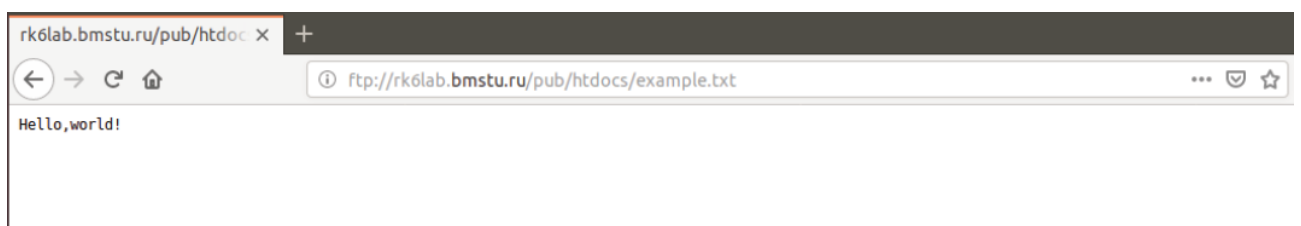


Рис. 6. Файл на сервере

Текст программы

```
#include <fcntl.h>
#include <arpa/inet.h>
#include <signal.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <ifaddrs.h>
#include <libgen.h>
#include <ctype.h>
#define SERVER_PORT "21"
#define MAX_COMMAND_SIZE 4 // Кол-во команд
#define USER "USER %s\r\n" // Запрос на логин
#define PASS "PASS %s\r\n" // Запрос на ввод пароля
#define STOR "STOR %s\r\n" // Запрос на загрузку файлов
#define PRODUCTION
char buf[1024]; // Буфер
int bytes_read; // Кол-во прочитанных байт
typedef void (*sighandler)(int);

// Структура "Подключение"
struct Connection
{
    int sock;
    struct addrinfo *info;
} *toListen = NULL, *toReceive = NULL;

// Инициализация структуры Connection по умолчанию
void toDefault(struct Connection *connection)
{
    if (connection != NULL)
    {
        connection->sock = -1;
        connection->info = NULL;
    }
}

typedef void (*Commands)(struct Connection *);

// Проверка ответа от сервера
int isOneAnswer(char *buf, int len)
{
    int i = 0;
```

```

while (i < len && !(buf[i - 1] == '\r' && buf[i] == '\n'))
    ++i;
if (i + 2 == len)
    return 0;
if (i + 1 == len)
    return 0;
return i + 1;
}

// Закрытие программы при прерывании по сигналу (^C)
void extremeClose(int c)
{
    if (toListen != NULL && toListen->sock >= 0)
        close(toListen->sock);
    printf("Interrupting program. Closing connection.\n");
    exit(c);
}

// Создание сокета для соединения по порту и ip
struct Connection *createConnection(const char *port, const char *ipAddress)
{
    struct Connection *NewConnection = (struct Connection *)malloc(sizeof(struct
Connection));
    toDefault(NewConnection);
    int status;
    struct addrinfo hints;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET;

    if (ipAddress == NULL)
        hints.ai_flags = AI_PASSIVE;

    printf("Connection on PORT: %s\n", port);

    status = getaddrinfo(ipAddress, port, &hints, &(NewConnection->info));
    NewConnection->sock = socket(NewConnection->info->ai_family, NewConnection-
>info->ai_socktype, NewConnection->info->ai_protocol);

    if (NewConnection->sock < 0)
    {
        perror("socket");
        extremeClose(1);
    }

    return NewConnection;
}

// Побайтная отправка файла FTP-серверу по соединению данных

```

```

int putFile(struct Connection *connection, char *fileName)
{
    int getFile = 0;
    char buf[1024];
    int bytes_read;

    getFile = open(fileName, O_RDONLY);

    if (getFile < 0)
        printf("Mistake\n");

    while (1)
    {
        bytes_read = read(getFile, buf, 1024);
        if (bytes_read <= 0)
            break;
        send(connection->sock, buf, bytes_read, 0);
    }

    close(getFile);
    close(connection->sock);
    freeaddrinfo(connection->info);
    free(connection);

    return 0;
}

// Получение команды
int getCommand()
{
    char *command = (char *)malloc(sizeof(char) * MAX_COMMAND_SIZE);
    int commandNum = -1;

    printf("ftp> ");
    scanf("%s", command);

    if (strcmp(command, "open") == 0)
        commandNum = 1;
    if (strcmp(command, "cd") == 0)
        commandNum = 2;
    if (strcmp(command, "put") == 0)
        commandNum = 3;
    if (strcmp(command, "quit") == 0)
        commandNum = 0;
    free(command);

    return commandNum;
}

```

```

int length(char *str)
{
    int count = 0;

    while (str[count++] != '\n')
        ;

    return count;
}

// Получение ответа от sock
void getAnswer(char *buf, int *bytes_read, int sock, int fileDescriptor)
{
    *bytes_read = recv(sock, buf, 1024, 0);
    buf[(*bytes_read)++] = '\n';

    if (fileDescriptor > -1)
        write(fileDescriptor, buf, *bytes_read);
}

// Формирование управляющего соединения с FTP-сервером
void openConnection(struct Connection *toSend)
{
    char *ipAddress = (char *)malloc(sizeof(char) * 15);
    do
    {
        scanf("%s", ipAddress);
        while (strlen(ipAddress) == 0);

        *toSend = *(createConnection(SERVER_PORT, ipAddress));

        if (connect(toSend->sock, toSend->info->ai_addr, toSend->info->ai_addrlen) <
0)
        {
            perror("connect");
            extremeClose(2);
        }

        free(ipAddress);
        getAnswer(buf, &bytes_read, toSend->sock, 1);

        if (strncmp(buf, "220", 3) != 0)
            toDefault(toSend);
        else
        {
            printf("Write your login: ");
            char *username = (char *)malloc(sizeof(char) * 256);
            scanf("%s", username);

```

```

        char *user = (char *)malloc(sizeof(char) * (4 + 1 + strlen(username) +
2));

        sprintf(user, USER, username);

        free(username);
        send(toSend->sock, user, length(user) * sizeof(char), 0);

        getAnswer(buf, &bytes_read, toSend->sock, 1);
        free(user);

        if (strncmp(buf, "331", 3) != 0)
            toDefault(toSend);
        else
        {
            printf("Write your password: ");
            char *password = (char *)malloc(sizeof(char) * 256);
            scanf("%s", password);

            char *pass = (char *)malloc(sizeof(char) * (4 + 1 + strlen(password)
+ 2));

            sprintf(pass, PASS, password);
            free(password);

            send(toSend->sock, pass, length(pass) * sizeof(char), 0);
            free(pass);

            getAnswer(buf, &bytes_read, toSend->sock, 1);

            if (strncmp(buf, "230", 3) != 0)
                toDefault(toSend);
        }
    }
}

// Отправка запроса на загрузку файла
void put(struct Connection *toSend)
{
    int i = 0;
    int flag = 0;
    int tempNumber = 0;
    int mas[6] = {0, 0, 0, 0, 0, 0};
    int j = 0;
    char *ipAdress_toRecive = (char *)malloc(sizeof(char) * 15);
    char *port_toRecive = (char *)malloc(sizeof(char) * 5);

    if (toSend == NULL || toSend->info == NULL)
        return;

```

```

char *fileName = (char *)malloc(sizeof(char) * 512);
char *nameFile = (char *)malloc(sizeof(char) * 512);

scanf("%s", fileName);

char *stor = (char *)malloc(sizeof(char) * (4 + 1 +
strlen(basename(fileName)) + 2));
sprintf(stor, STOR, basename(fileName));

char type[] = "TYPE I\r\n";
send(toSend->sock, type, length(type) * sizeof(char), 0);

getAnswer(buf, &bytes_read, toSend->sock, 1);

if (strncmp(buf, "200", 3) == 0)
{
    char pasv[] = "PASV\r\n";

    send(toSend->sock, pasv, length(pasv) * sizeof(char), 0);
    getAnswer(buf, &bytes_read, toSend->sock, 1);

    if (strncmp(buf, "227", 3) == 0)
    {
        for (; i < length(buf); ++i)
        {
            if (buf[i] == '(')
                flag = 1;
            if (isdigit(buf[i]) && flag == 1)
                tempNumber = tempNumber * 10 + buf[i] - '0';
            if (buf[i] == ',' || buf[i] == ')')
            {
                mas[j++] = tempNumber;
                tempNumber = 0;
            }
        }

        sprintf(ipAdress_toRecive, "%d.%d.%d.%d", mas[0], mas[1], mas[2],
mas[3]);

        sprintf(port_toRecive, "%d", mas[4] * 256 + mas[5]);

        toReceive = createConnection(port_toRecive, ipAdress_toRecive);

        if (connect(toReceive->sock, toReceive->info->ai_addr, toReceive-
>info->ai_addrlen) < 0)
        {
            perror("connect");
            extremeClose(2);

```

```

    }
}

send(toSend->sock, stor, length(stor) * sizeof(char), 0);
getAnswer(buf, &bytes_read, toSend->sock, 1);

int wasSend = 0;

if (strncmp(buf, "150", 3) == 0)
{
    sprintf(nameFile, "%s", basename(fileName));
    putFile(toReceive, fileName);

    if ((wasSend = isOneAnswer(buf, bytes_read / (sizeof(char)))) == 0)
        getAnswer(buf + wasSend, &bytes_read, toSend->sock, 1);
}
}

free(port_toRecive);
free(ipAdress_toRecive);
free(fileName);
free(nameFile);
free(stor);
}

// Выход из клиента
void quit(struct Connection *toSend)
{
    printf("Exit\n");
    close(toSend->sock);
    freeaddrinfo(toSend->info);
}

// Переход в нужную директорию на сервере
void cd(struct Connection *toSend)
{
    char *pwd = (char *)malloc(sizeof(char) * 3 + 2);

    sprintf(pwd, "CWD ");
    strcat(pwd, "/pub/htdocs\n\0");
    write(1, pwd, strlen(pwd));
    send(toSend->sock, pwd, length(pwd) * sizeof(char), 0);
    getAnswer(buf, &bytes_read, toSend->sock, 1);
    free(pwd);
}

// Цикл чтения команд и вызова соответствующих функций
void cycle()

```

```

{
    struct Connection toSend;
    Commands massOfCommands[] = {quit, openConnection, cd, put};
    int commandNum;

    do
    {
        commandNum = getCommand();

        if (commandNum > -1)
            massOfCommands[commandNum](&toSend);
    } while (commandNum != 0);
}

int main(int argc, char *argv[])
{
    cycle();
    return 0;
}

```