Задание

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

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.

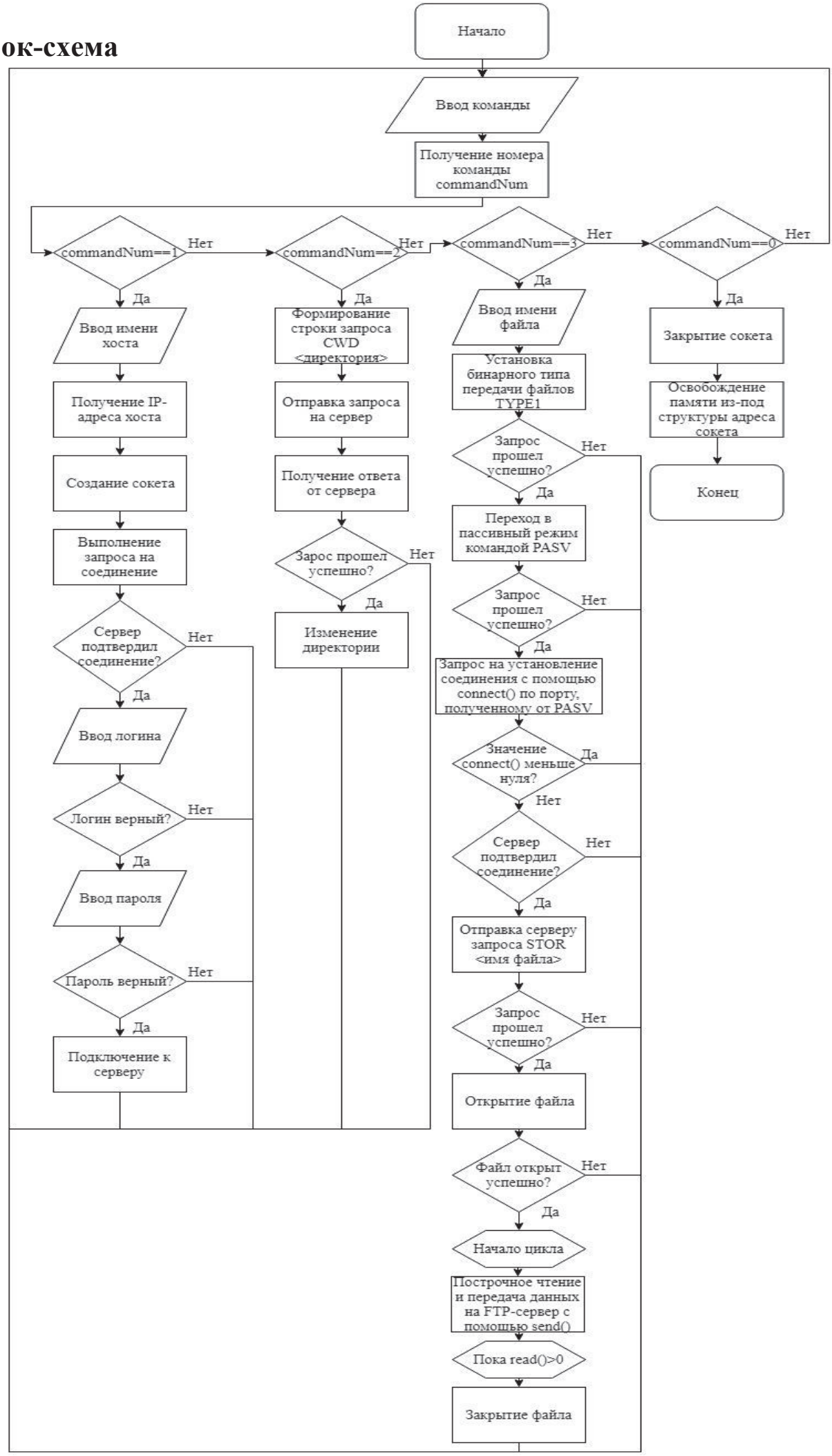
	masOfCommands[]			
Индекс	0	1	2	3
Команда	quit	open	cd	put

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. Для обмена данными используется полученный от сервера по команде PASV порт.
- Как только достигнут конец файла, цикл завершается.
- Закрытие файла
- Закрытие сокета
- Освобождение памяти, выделенной под структуру Connection

Блок-схема


Начало
Ввод команды
Получение номера команды commandNum
commandNum==1
Нет
commandNum==2
Нет
commandNum==3
Нет
commandNum==0
Нет
Да
Ввод имени хоста
Получение IP-адреса хоста
Создание сокета
Выполнение запроса на соединение
Сервер подтвердил соединение?
Нет
Да
Ввод логина
Логин верный?
Нет
Да
Ввод пароля
Пароль верный?
Нет
Да
Подключение к серверу
Формирование строки запроса CWD <директория>
Отправка запроса на сервер
Получение ответа от сервера
Зарос прошел успешно?
Нет
Да
Изменение директории
Ввод имени файла
Установка бинарного типа передачи файлов TYPE1
Запрос прошел успешно?
Нет
Да
Переход в пассивный режим командой PASV
Запрос прошел успешно?
Нет
Да
Запрос на установление соединения с помощью connect() по порту, полученному от PASV
Значение connect() меньше нуля?
Да
Нет
Сервер подтвердил соединение?
Нет
Да
Отправка серверу запроса STOR <имя файла>
Запрос прошел успешно?
Нет
Да
Открытие файла
Файл открыт успешно?
Нет
Да
Начало цикла
Построчное чтение и передача данных на FTP-сервер с помощью send()
Пока read()>0
Закрытие файла
Закрытие сокета
Освобождение памяти из-под структуры адреса сокета
Конец

Результаты работы

```
yuiop@yuiop-VirtualBox:~/labs/lab3v12$ ./a.out
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
yuiop@yuiop-VirtualBox:~/labs/lab3v12$
```

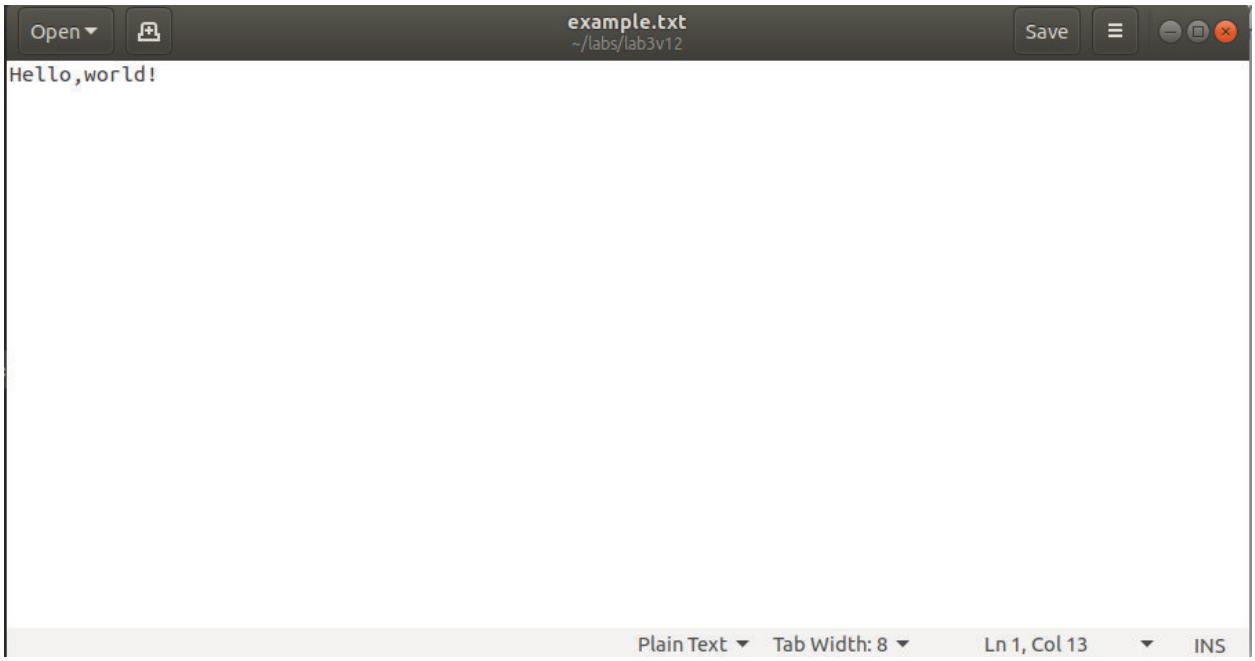


Рис. Содержимое файла

```
-rwxrwxrwx 1 1001 1001    206 Apr 24 12:10 controller.php
-rwxrwxrwx 1 1001 1001   5656 Apr 24 10:06 cop.php
-rw-r----- 1 1001 1001    329 Apr 24 17:53 dbconnect.php
drwxr-x--- 2 1001 1001   4096 Jun  6 10:04 dh
-rw-r----- 1 1001 1001   2506 May 15 09:51 dump.sql
-rwxrwxrwx 1 1001 1001     35 Apr 17 15:00 eba.php
-rw-r----- 1 1001 1001     13 Aug 20 11:59 example.txt
-rw-r----- 1 1001 1001 105004 Apr 24 09:21 fedoruk.png
-rw-r----- 1 1001 1001      0 May 15 08:57 femdb.sql
-rwxrwxrwx 1 1001 1001  12276 Apr 24 08:28 femdbDIMAS.php
-rwxrwxrwx 1 1001 1001  12344 May 15 13:53 femdb_show.php
-rwxrwxrwx 1 1001 1001  13290 May 22 09:18 femdb_show_good.php
-rwxrwxrwx 1 1001 1001   1782 Jun  6 08:53 form.html
-rwxrwxrwx 1 1001 1001   4162 Jun  6 11:09 form.php
-rwxrwxrwx 1 1001 1001   1622 Jun  6 11:09 functions.php
-rw-r----- 1 1001 1001   2971 May 15 10:30 gause.inc.php
-rw-r----- 1 1001 1001   1847 May 15 10:33 gause.php
-rwxrwxrwx 1 1001 1001   2248 May 15 10:47 gbuild.php
drwxrwxrwx 2 1001 1001   4096 May 15 06:58 glo
-rwxrwxrwx 1 1001 1001   5650 Apr 24 10:51 gphp19.php
-rwxrwxrwx 1 1001 1001   6016 Apr 24 09:10 gphp1999.php
-rw-r----- 1 1001 1001   5653 Apr 17 17:29 gphp4.php
-rwxrwxrwx 1 1001 1001   5653 Apr 24 11:02 gphpg9.php
-rwxrwxrwx 1 1001 1001   5654 Apr 24 09:42 gphpp.php
```

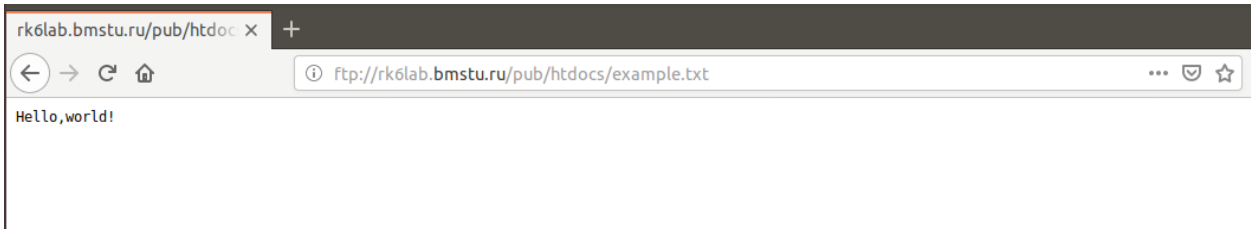


Рис. Файл на сервере

Текст программы

```
1.  #include <fcntl.h>
2.  #include <arpa/inet.h>
3.  #include <signal.h>
4.  #include <sys/stat.h>
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.  #include <unistd.h>
8.  #include <string.h>
9.  #include <netdb.h>
10. #include <ifaddrs.h>
11. #include <libgen.h>
12. #include <ctype.h>
13.
14. #define SERVER_PORT 8081
15. #define MAX_COMMAND_SIZE 4          // Кол-во команд
16. #define USER "USER %s\r\n"          // Запрос на логин
17. #define PASS "PASS %s\r\n"          // Запрос на ввод пароля
18. #define STOR "STOR %s\r\n"          // Запрос на загрузку файла
19. #define PRODUCTION
20. char buf[1024];                              // Буфер
21. int bytes_read;                              // Кол-во прочитанных байт
22. char *myIP;                                        // Строка IP-адреса
23. typedef void (*sighandler) (int);
24.
25. /*Структура "Подключение"*/
26. struct Connection
27. {
28.    int sock;
29.    struct addrinfo *info;
30. } *toListen = NULL, *toRecive=NULL;
31.
32. /*Инициализация структуры Connection по умолчанию*/
33. void toDefault(struct Connection *connection)
34. {
35.    if (connection != NULL)
36.    {
37.           connection->sock = -1;
38.           connection->info = NULL;
39.    }
40. }
41. typedef void (*Commands) (struct Connection *);
42. /*Проверка ответа от сервера*/
43. int isOneAnswer(char *buf, int len)
44. {
45.    int i = 0;
46.    while(i < len && !(buf[i - 1] == '\r' && buf[i] == '\n'))
47.           ++i;
48.    if (i + 2 == len)
49.           return 0;
50.    if (i + 1 == len)
51.           return 0;
52.    return i + 1;
53. }
54. /* Закрытие программы при прерывании по сигналу (^C)*/
55. void extremeClose (int c)
56. {
57.    if (toListen != NULL && toListen->sock >= 0)
58.           close(toListen->sock);
59.    printf("Interupting programm. Closing connection.\n");
```

```
60.    exit(c);
61. }
62. /*Смена точек на запятые в строке*/
63. void changePointsToCommas(char *str)
64. {
65.    int len = strlen(str);
66.    int i;
67.    for (i = 0; i < len; ++i)
68.          if (str[i] == '.')
69.                str[i] = ',';
70. }
71. /*Смена запятых на точки в строке*/
72. void changeCommasToPoints(char *str)
73. {
74.    int len = strlen(str);
75.    int i;
76.    for (i = 0; i < len; ++i)
77.          if (str[i] == ',')
78.                str[i] = '.';
79. }
80. /*Создание сокета для соединения по порту и ip*/
81. struct Connection *createConnection(const char *port, const char *ipAdress)
82. {
83.    struct Connection *NewConnection = (struct Connection*)malloc(sizeof(struct
      Connection));
84.    toDefault(NewConnection);
85.    int status;
86.    struct addrinfo hints;
87.    memset(&hints, 0, sizeof hints);
88.    hints.ai_family = AF_INET;
89.    if (ipAdress == NULL)
90.          hints.ai_flags = AI_PASSIVE;
91.    printf("connection port:%s\n", port);
92.    status = getaddrinfo(ipAdress, port, &hints, &(NewConnection->info));
93.    NewConnection->sock = socket(NewConnection->info->ai_family, NewConnection-
      >info->ai_socktype, NewConnection->info->ai_protocol);
94.    if(NewConnection->sock < 0)
95.    {
96.          perror("socket");
97.          extremeClose (1);
98.    }
99.    return NewConnection;
100.   }
101.   /* Побайтная отправка файла FTP-серверу по соединению данных*/
102.   int putFile(struct Connection *connection, char* fileName)
103.   {
104.         int getFile = 0;
105.         char buf[1024];
106.         int bytes_read;
107.         getFile = open(fileName, O_RDONLY);
108.           if (getFile < 0){printf("Mistake\n");}
109.
110.         while(1)
111.         {
112.               bytes_read = read(getFile, buf, 1024);
113.               if(bytes_read <= 0)
114.                     break;
115.               send(connection->sock, buf, bytes_read, 0);
116.         }
117.         close(getFile);
118.         close(connection->sock);
```

```
119.        freeaddrinfo(connection->info);
120.        free(connection);
121.        return 0;
122.  }
123.  /* Получение команды*/
124.  int getCommand()
125.  {
126.        char* command = (char*) malloc (sizeof(char) * MAX_COMMAND_SIZE);
127.        int commandNum = -1;
128.        printf("ftp> ");
129.        scanf("%s", command);
130.        if (strcmp(command, "open") == 0)
131.              commandNum = 1;
132.        if (strcmp(command, "cd") == 0)
133.              commandNum = 2;
134.        if (strcmp(command, "put") == 0)
135.              commandNum = 3;
136.        if (strcmp(command, "quit") == 0)
137.              commandNum = 0;
138.        free(command);
139.        return commandNum;
140.  }
141.
142.  int length(char* str)
143.  {
144.        int count = 0;
145.        while(str[count++] != '\n');
146.        return count;
147.  }
148.  /* Получение ответа от sock*/
149.  void getAnswer(char *buf, int *bytes_read, int sock, int fileDescriptor)
150.  {
151.        *bytes_read = recv(sock, buf, 1024, 0);
152.        buf[(*bytes_read)++] = '\n';
153.        if(fileDescriptor > -1)
154.              write(fileDescriptor, buf, *bytes_read);
155.  }
156.  /* Формирование управляющего соединения с FTP-сервером*/
157.  void openConnection(struct Connection *toSend)
158.  {
159.        char* ipAdress = (char*) malloc (sizeof(char) * 15);
160.        do
161.              scanf("%s", ipAdress);
162.        while (strlen(ipAdress) == 0);
163.        *toSend = *(createConnection("21", ipAdress));
164.        if(connect(toSend->sock, toSend->info->ai_addr, toSend->info->ai_addrlen)
      < 0)
165.        {
166.              perror("connect");
167.              extremeClose (2);
168.        }
169.        free(ipAdress);
170.        getAnswer(buf, &bytes_read, toSend->sock, 1);
171.        if (strncmp(buf, "220", 3) != 0)
172.              toDefault(toSend);
173.        else
174.        {
175.              printf("Write your login: ");
176.              char* username = (char*) malloc (sizeof(char) * 256);
177.              scanf("%s", username);
```

```
178.                char* user = (char*) malloc (sizeof(char) * (4 + 1 +
    strlen(username) + 2));
179.                sprintf(user, USER, username);
180.                free(username);
181.                send(toSend->sock, user, length(user)* sizeof(char), 0);
182.                getAnswer(buf, &bytes_read, toSend->sock, 1);
183.                free(user);
184.                if (strncmp(buf, "331", 3) != 0)
185.                      toDefault(toSend);
186.                else
187.                {
188.                      printf("Write your password: ");
189.                      char* password = (char*) malloc (sizeof(char) * 256);
190.                      scanf("%s", password);
191.                      char* pass = (char*) malloc (sizeof(char) * (4 + 1 +
    strlen(password) + 2));
192.                      sprintf(pass, PASS, password);
193.                      free(password);
194.                      send(toSend->sock, pass, length(pass) * sizeof(char), 0);
195.                      free(pass);
196.                      getAnswer(buf, &bytes_read, toSend->sock, 1);
197.                      if (strncmp(buf, "230", 3) != 0)
198.                            toDefault(toSend);
199.                }
200.          }
201.    }
202.    /* Отправка запроса на закачку файла*/
203.    void put(struct Connection *toSend)
204.    {
205.          int i=0;
206.          int flag = 0;
207.          int tempNumber = 0;
208.          int mas[6] = {0, 0, 0, 0, 0, 0};
209.          int j = 0;
210.          char* ipAdress_toRecive = (char*) malloc (sizeof(char) * 15);
211.          char* port_toRecive = (char*) malloc (sizeof(char) * 5);
212.          if (toSend == NULL || toSend->info == NULL)
213.                return;
214.          char* fileName = (char*) malloc (sizeof(char) * 512);
215.          char* nameFile = (char*) malloc (sizeof(char) * 512);
216.          scanf("%s", fileName);
217.          char *stor = (char*) malloc (sizeof(char) * (4 + 1 +
    strlen(basename(fileName)) + 2));
218.          sprintf(stor, STOR, basename(fileName));
219.          char type[] = "TYPE I\r\n";
220.          send(toSend->sock, type, length(type) * sizeof(char), 0);
221.          getAnswer(buf, &bytes_read, toSend->sock, 1);
222.          if (strncmp(buf, "200", 3) == 0)
223.          {
224.                char pasv[] ="PASV\r\n";
225.                send(toSend->sock, pasv, length(pasv) * sizeof(char), 0);
226.                getAnswer(buf, &bytes_read, toSend->sock, 1);
227.                if (strncmp(buf, "227", 3) == 0)
228.                {
229.                      for (; i < length(buf); ++i)
230.                      {
231.                            if (buf[i] == '(')
232.                                  flag = 1;
233.                            if (isdigit(buf[i]) && flag == 1)
234.                                  tempNumber = tempNumber * 10 + buf[i] - '0';
235.                            if (buf[i] == ',' || buf[i] == ')')
```

```
236.                            {
237.                                    mas[j++] = tempNumber;
238.                                    tempNumber = 0;
239.                            }
240.                    }
241.
        sprintf(ipAdress_toRecive,"%d.%d.%d.%d",mas[0],mas[1],mas[2],mas[3]);
242.                    sprintf(port_toRecive,"%d",mas[4]*256+mas[5]);
243.                    toRecive = createConnection(port_toRecive,
    ipAdress_toRecive);
244.                    if(connect(toRecive->sock, toRecive->info->ai_addr,
    toRecive->info->ai_addrlen) < 0)
245.                    {
246.                            perror("connect");
247.                            extremeClose (2);
248.                    }
249.                    send(toSend->sock, stor, length(stor) * sizeof(char), 0);
250.                    getAnswer(buf, &bytes_read, toSend->sock, 1);
251.                    int wasSend = 0;
252.                    if (strncmp(buf, "150", 3) == 0)
253.                    {
254.                            sprintf(nameFile,"%s", basename(fileName));
    putFile(toRecive, fileName);
255.                            if((wasSend = isOneAnswer(buf, bytes_read /
    (sizeof(char)))) == 0)
256.                                    getAnswer(buf + wasSend, &bytes_read, toSend-
    >sock, 1);
257.                    }
258.            }
259.        }
260.        free(port_toRecive);
261.        free(ipAdress_toRecive);
262.        free(fileName);
263.        free(nameFile);
264.        free(stor);
265.    }
266.    /* Выход из клиента*/
267.    void quit(struct Connection *toSend)
268.    {
269.        printf("Goodbye\n");
270.        close(toSend->sock);
271.        freeaddrinfo(toSend->info);
272.    }
273.    /* Переход в нужную директорию на сервере*/
274.    void cd(struct Connection *toSend)
275.    {
276.        char* pwd = (char*) malloc(sizeof(char) * 3 + 2);
277.        sprintf(pwd, "CWD ");
278.        strcat(pwd, "/pub/htdocs\n\0");
279.        write(1, pwd, strlen(pwd));
280.        send(toSend->sock, pwd, length(pwd) * sizeof(char), 0);
281.        getAnswer(buf, &bytes_read, toSend->sock, 1);
282.        free(pwd);
283.    }
284.    /* Цикл чтения команд и вызова соотвествующих функций*/
285.    void cycle(char *myIP)
286.    {
287.        struct Connection toSend;
288.        Commands masOfCommands[] = {quit, openConnection, cd, put};
289.        int commandNum;
290.        do
```

```
291.         {
292.                 commandNum = getCommand();
293.                 if(commandNum > -1)
294.                         masOfCommands[commandNum](&toSend);
295.         }
296.         while(commandNum != 0);
297.   }
298.
299.   void setMyIP(char *myIP)
300.   {
301.         struct ifaddrs *ifaddr, *ifa;
302.         int family, s;
303.         char host[NI_MAXHOST];
304.         if (getifaddrs(&ifaddr) == -1)
305.         {
306.                 perror("getifaddrs");
307.                 exit(EXIT_FAILURE);
308.         }
309.         for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next)
310.         {
311.                 family = ifa->ifa_addr->sa_family;
312.                 if (family == AF_INET)
313.                 {
314.                         s = getnameinfo(ifa->ifa_addr, sizeof(struct sockaddr_in),
315.                                 host, NI_MAXHOST, NULL, 0, NI_NUMERICHOST);
316.                         if (s != 0)
317.                         {
318.                                 extremeClose (-4);
319.                                 printf("getnameinfo() failed: %s\n",
      gai_strerror(s));
320.                         }
321.                         if (strcmp(ifa->ifa_name, "lo") != 0)
322.                         {
323.                                 sprintf(myIP, "%s", host);
324.                                 return;
325.                         }
326.                 }
327.         }
328.   }
329.   /* main*/
330.   int main(int argc, char* argv[])
331.   {
332.         myIP = (char*) malloc(sizeof(char) * 23);
333.         setMyIP(myIP);
334.         signal (SIGINT, (sighandler) extremeClose);
335.         cycle(myIP);
336.         return 0;
337.   }
```