Санкт-Петербургский государственный политехнический
университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе
Дисциплина: Сети ЭВМ и телекоммуникации
Тема: Программирование сокетов протоколов TCP и UDP

Выполнил
студент гр. 43501/3 _____ Зобков Д.А.
(подпись)

Преподаватель _____ Зозуля А.В.
(подпись)

“___”_____ 2016 г.

Санкт-Петербург
2016 г.

СОДЕРЖАНИЕ

1 Цель работы

Ознакомиться с принципами программирования собственных протоколов, созданных на основе TCP и UDP.

2 Программа работы

TCP:

1. Реализация простейшего TCP клиента и сервера на ОС Linux и Windows соответственно;
2. Реализация многопоточного обслуживания клиентов на сервере;
3. Реализация собственного протокола на основе TCP для индивидуального задания;
4. Реализация синхронизации с помощью мьютексов.

UDP:

1. Модификация клиента и сервера для протокола UDP на ОС Windows и Linux соответственно;
2. Обеспечение надежности протокола UDP посредством нумерации пакетов и посылки ответов.

3 Индивидуальное задание

Разработать клиент-серверную систему терминального доступа, позволяющую клиентам подсоединяться к серверу и выполнять элементарные команды операционной системы.

4 Протокол команд

4.1 Форматы команд

Для получения или изменения информации на сервере клиент посылает серверу текстовые команды. Набор команд для TCP клиента должен удовлетворять следующим требованиям:

- Все команды пишутся в нижнем регистре;
- Соответствие шаблону сообщения.

Протокол TCP имеет следующий шаблон сообщения:

<команда> <атрибут> <атрибут>

В начале сообщения, всегда присутствует требуемая для выполнения команда, далее в зависимости от нее могут идти атрибуты, которые отделены друг от друга пробелом. В случае, если пересылаемое клиентом сообщение не соотвествует шаблону, сервер сообщает о неверно введенной команде.

Команды оперируют некоторыми сущностями:

Таблица 4.1. Команды администратора сервера

Мнемоника	Описание
<LOGIN>	Имя пользователя
<PASSWORD>	Пароль пользователя
<PATH>	Относительный путь к новой директории
<PERMISSIONS>	Привилегии на использование некоторых команд: 1. c — chmod; 2. k — kill; 3. w — who.
<ID>	ID пользователя, используется для команды k <ID> на сервере

Список команд, которыми оперирует клиент:

Таблица 4.2. Команды пользователя до авторизации

Команда	Атрибуты	Действие	Ответ сервера
login	<LOGIN> <PASSWORD>	Попытка авторизации, используя введенный логин и пароль	1. Вывод текущей директории; 2. Login/password not match! 3. This user already online!
addusr	<LOGIN> <PASSWORD>	Попытка зарегистрироваться, используя введенный логин и пароль	1. Вывод текущей директории; 2. Matching login/password are not allowed! 3. This user already exists!

Таблица 4.3. Команды пользователя после успешной авторизации

Команда	Атрибуты	Действие	Ответ сервера
ls	Отсутствуют	Вывод содержимого в текущей директории	Список файлов и директорий
pwd	Отсутствуют	Вывод абсолютного пути текущей директории	Текущая директория
cd	<PATH>	Перемещение в заданную директорию	1. Вывод новой текущей директории; 2. Directory “%DirectoryName%” is not exist.
who	Отсутствуют	Вывод списка пользователей, их текущей директории и наличия в сети	1. Вывод информации; 2. You doesn’t have permissions for this command!
chmod	<LOGIN> <PERMISSIONS>	Изменение привилегий другого пользователя	1. Вывод текущей директории; 2. You doesn’t have permissions for this command! 3. You can’t change root and yours permissions! 4. This user doesn’t exists!
kill	<LOGIN>	Завершение сеанса другого пользователя	1. Вывод текущей директории; 2. You doesn’t have permissions for this command! 3. You can’t kill root and yourself! 4. This user is offline or not exists!
logout	Отсутствуют	Смена пользователя	Login or register new user (login/addusr LOGIN PASSWORD)

Список команд, которыми оперирует администратор сервера:

Таблица 4.4. Команды администратора сервера

Команда	Атрибуты	Действие
l	Отсутствуют	Вывод всех подключенных пользователей в виде: <ID>|<IP>:<PORT>|<LOGIN> (если авторизован, если нет, то <LOGIN> отсутствует)
k	<ID>	Отключение пользователя с введенным <ID>
q	Отсутствуют	Отключение сервера

Для протокола UDP индивидуальное задание не реализовывалось, разработанное приложение представляет собой эхо-сервер, возвращающий клиенту посланное им сообщение с припиской “Echo: ”.

4.2 Подключение к серверу

Клиент пытается подключиться к предварительно записанным ip-адресу и порту.
Сервер TCP: 192.168.222.1:8080
Сервер UDP: 192.168.222.15:8080
ip-адрес и порт, к которому подключается клиент, можно изменить, используя при его запуске ключи -i и -p соответственно, также возможно изменить порт сервера ключом -p при его запуске.

4.3 Формат хранимых файлов на сервере

Серверная программа TCP позволяет сохранять, изменять и загружать информацию о пользователях. На сервере обязательно присутствует файл users.txt, в котором хранятся существующие пользователи в следующем виде:

<USER>||<PASSWORD>||<PERMISSIONS>||<PATH>

Серверная программа UDP не хранит никакой информации в файлах во время своей работы.

5 Реализация программы

5.0.1 Структура проекта

При разработке приложения для операционной системы семейства Linux использовались текстовые редакторы Sublime Text 3 и Vim и компилятор g++.
При разработке приложения для операционной системы семейства Windows использовался текстовый редактор Sublime Text 3 и компилятор MinGW.
Язык программирования — C++.

5.0.2 Сетевая часть TCP

Клиентское приложение в TCP только отсылает команды на сервер, поэтому оно ничем не отличается от telnet клиента. Сервер обрабатывает команды, работает с файлом, сохраняет и загружает свое состояние, присылает уведомления и т.д.
В первую очередь, на сервере происходит инициализация WinSock (на Windows), создание сокета (функция socket), привязка сокета к конкретному адресу (функция bind), подготовка сокета к принятию сообщений (функция listen). Реализация инициализации сервера представлена в разд. А.1 на с. 12.

После этого ожидаем подключения клиентов в бесконечном цикле с помощью функции accept. Если функция возвращает положительное значение, которое является клиентским сокетом, то создаем новый поток, в котором обрабатываем клиентские сообщения. Реализация подключения клиентов представлена в разд. А.2 на с. 12. Структуры, используемые для работы с клиентами, представлены в разд. А.3 на с. 14.

Клиентский поток вызывает функцию считывания символов в бесконечном цикле, как только при считывается знак перевода строки, функция возращает прочитанные символы. Если функция не вернула исключение, то посылаем команду на обработку, в противном случае это обозначает отключение клиента. Также отключение клиента может быть произведено извне обработчика клиентского потока, посредством закрытия клиентского сокета (функция считывания в этом случае сразу же вернет исключение) командой k <ID> или q. Реализация клиентского потока представлена в разд. А.4 на с. 14.

Функция считывания до символа перевода строки — это оболочка для функции recv. Считывание происходит по одному символу и записывается в результирующую строку. Если соединение с клиентом разорвано, то функция возвращает исключение. Реализация функции считывания до символа перевода строки представлена в разд. А.5 на с. 18. Реализация закрытия сокета представлена в разд. А.6 на с. 18.

5.0.3 Сетевая часть UDP

Клиентское приложение в UDP не может быть заменено сторонним приложением по типу telnet, потому что используется нумерация пакетов, контроль команд и посылка ответов.

Реализация сетевой части UDP сервера похожа на TCP, за исключением следующих отличий:

- Функция создания сокета выполняется с параметрами SOCK_DGRAM, IPPROTO_UDP;
- Отсутствует функция подготовки к принятию сообщений listen;
- Отсутствует функция установления соединения accept, вместо нее используется функция recvfrom и создание клиентского сокета функцией socket;
- Вместо функций send и recv используются функции sendto и recvfrom с явным указанием адресной структуры.

Реализации инициализации сервера и подключения новых клиентов представлены в раздел. Б.1–Б.2 на с. 20. Структура, используемая для работы с клиентами, представлены в разд. Б.3 на с. 21.

Так как протокол UDP ненадежный и без установления соединения, некоторые пакеты могут затеряться в сети. Во избежание этого была реализована нумерация пакетов и посылка ответных пакетов. Для клиента каждое отправляемое сообщение соответствует следующей структуре:

<PACKET_ID> <MESSAGE>

После отправки данного сообщения, клиент в течение 2-х секунд ожидает ответа от сервера, после его получения отсылает сообщение о подтверждении получения данного ответа следующего формата:

<PACKET_ID> #CHECK

Далее возможны следующии ситуации:

- В случае если подтверждение не пришло, то будет выведено сообщение о том, что пакет <PACKET_ID> был потерян;
- В случае если подтверждение пришло, но номер принятого пакета подтверждения не соответствует ожидаемому, будет выведено соответствующее сообщение.

Как только сервер получил от клиента сообщение, происходит проверка на дубликат или перемешивание <PACKET_ID>, в случае если данная проверка подтверждается, то сервер пересылает клиенту сообщение о дубликате или перемешивании.

В случае если сообщение не является дубликатом или не тем пакетом, то сервер возвращает эхо-сообщение, с такой же структурой, которое было получено от клиента.

6 Тестирование

Тестирование проводилось на операционных системах Ubuntu Server 16.04 и Windows 10. Были проверены все команды, многопоточность, уведомления, UDP-пакеты, правильное завершение всех потоков.

6.1 TCP

6.1.1 Запуск и выполнение команд

```
Server settings
Threads: 2
Port:    8080
Buffer:  8

Connection request received.
New socket was created at address 192.168.222.1:16115
l
Client list:
0|192.168.222.1:16115|
192.168.222.1:16115 login dan0n pass
l
Client list:
0|192.168.222.1:16115|dan0n
```

Листинг 6.1. Лог сервера

```
Socket created
Connect success
Login or register new user (login/addusr LOGIN PASSWORD)
login dan0n pass
dan0n @ D:\4 course $
```

Листинг 6.2. Лог клиента

После запуска на стороне клиента вывелись сообщения об успешной инициализации. Далее с сервера было автоматически получено приветственное сообщение, с предложение авторизации или регистрации. Пользователем была введена команда авторизации, а также уже существующий логин и пароль. После успешной проверки введенного сообщения, сервер отправил путь текущей директории для данного пользователя.

На стороне сервера, были выведены сообщения об инициализации, затем после подключения клиента было выведено соответствующее сообщение. Также показано, как данный клиент отображается в списке до и после авторизации.

6.1.2 Многопоточность

```
...
Connection request received.
New socket was created at address 192.168.222.1:18637
Connection request received.
New socket was created at address 192.168.222.1:21297
```

```
Connection request received.
New socket was created at address 192.168.222.1:22978
```

Листинг 6.3. Лог сервера

К серверу были подключены три клиента, два из них были подключены через написанный клиент, а третий через telnet. Сервер и клиенты успешно функционировали.

6.1.3 Отключение клиента

```
Server settings
Threads: 2
Port:    8080
Buffer:  8

Connection request received.
New socket was created at address 192.168.222.1:16115
l
Client list:
0|192.168.222.1:16115|
192.168.222.1:16115 login dan0n pass
l
Client list:
0|192.168.222.1:16115|dan0n
192.168.222.1:16115 was disconnected
```

Листинг 6.4. Лог сервера

```
Socket created
Connect success
Login or register new user (login/addusr LOGIN PASSWORD)
login dan0n pass
dan0n @ D:\4 course $
quit
You are disconnected!
```

Листинг 6.5. Лог клиента

Клиент успешно подключился к серверу, затем процесс клиента был завершен. На сервере сразу же вывелось соответствующее сообщение об отключении, так как при очередной попытке считать данные из сокета возникла ошибка, и процесс закончился.

6.1.4 Отключение клиента сервером

```
Server settings
Threads: 2
Port:    8080
Buffer:  8

Connection request received.
New socket was created at address 192.168.222.1:54521
k 0
192.168.222.1:54521 was disconnected
```

Листинг 6.6. Лог сервера

```
Socket created
Connect success
Login or register new user (login/addusr LOGIN PASSWORD)
You are disconnected!
```

Листинг 6.7. Лог клиента

Клиент успешно подключился к серверу, на сервере этот клиент был отключен командой k 0. В клиенте было выведено сообщение о том, что он отключен.

6.2 UDP

6.2.1 Запуск и выполнение команд

```
Server port: 8080

Add new client 127.0.0.1:14554
ID:0 checked
ID:0 checked
```

Листинг 6.8. Лог сервера

```
Use phrase "quit" for exit
qwerty
Echo: qwerty

echo
Echo: echo
```

Листинг 6.9. Лог клиента

После отправления сообщения клиентом приходят эхо-ответы от сервера. На стороне сервера видно подключение клиента и сообщения о проверенных пакетах, дошедших до клиента.

6.2.2 Многопоточность

```
Server port: 8080

Add new client 127.0.0.1:14554
ID:0 checked
ID:0 checked
Add new client 127.0.0.1:23532
ID:1 checked
ID:1 checked
l
Client list:
0|127.0.0.1:14554
1|127.0.0.1:23532
```

Листинг 6.10. Лог сервера

К серверу были подключены два клиента, команда l подтвердила их подключение и вывела некоторую информацию о них.

6.2.3 Перемешивание/повторение пакетов

Проведем эксперимент. Заставим клиента постоянно повторять 3 и 4 пакеты и посмотрим на результат.

```
Server port: 8080

Add new client 127.0.0.1:41436
ID:0 checked
ID:0 checked
ID:0 checked
ID:0 checked
ID:0 checked
ID:0 checked
ID:0 checked
ID:0 checked
```

Листинг 6.11. Лог сервера

```
Use phrase "quit" for exit
1
Echo: 1

2
Echo: 2

3
Echo: 3

4
Echo: 4

3
PACKAGE 3 WAS LOST
RECEIVED: #MIXING
4
PACKAGE 4 WAS LOST
RECEIVED: #SAME
3
PACKAGE 3 WAS LOST
RECEIVED: #MIXING
4
PACKAGE 4 WAS LOST
RECEIVED: #SAME
```

Листинг 6.12. Лог клиента

Как можно заметить, при получении 3-го пакета после 4-го от сервера приходит сообщение о перемешивании пакетов, при повторной отправке 4-го пакета приходит сообщение о дублировании пакетов, так как ожидается 5-ый пакет.

7 Выводы

В данной лабораторной работе были реализованы клиент-серверная программа терминального доступа и эхо-сервер с разработкой собственного протокола на основе TCP и UDP. Протокол был реализован на языке C++ для операционных систем Windows и Linux.

На примере данной разработки были изучены основные приемы использования протокола транспортного уровня TCP — транспортного механизма, предоставляющего поток данных с предварительной установкой соединения. Его преимуществом является достоверность получаемых данных за счет осуществления повторного запроса данных в случае их потери, устранения дублирования при получении копий одного пакета.

Однако TCP может не подойти в некоторых ситуациях обмена по сети вследствие медленной (по сравнению с UDP) работы. Например, передавая по сети данные, требующие быстрого отклика в реальном времени, необходимо соблюдать жесткие временные рамки, с которыми протокол TCP может не справиться.

В ходе данной работы было проведено ознакомление с протоколом UDP и реализовано простейшее клиент-серверное приложение эхо-сервера. По сравнению с TCP, UDP — более простой, основанный на сообщениях, протокол без установления соединения, однако требует дополнительного контроля доставки сообщений ввиду следующих особенностей:

- Неупорядоченности — если два сообщения отправлены последовательно, порядок их получения не может быть предугадан;
- Ненадежности — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути.

Поэтому, UDP наиболее часто используется требовательными ко времени приложениями, когда небольшие потери не играют большой роли. Если же требуется надежность доставки, то предпочтительнее использовать TCP.

Приложение А Часть TCP

А.1 Создание сервера

```
int main(int argc, char *argv[]){

    /* ... */
    /* Обработка ключей при запуске и открытие файла users.txt */

    startWSA();

    WaitForSingleObject(hMutex, INFINITE);
    for (int i = 0; i < maxThreads; i++)
        clientDesc[i].sock = INVALID_SOCKET;
    ReleaseMutex(hMutex);

    sockaddr_in server;
    listenSocket = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port        = htons(serverPort);
    server.sin_family      = AF_INET;

    if (listenSocket < 0){
        puts("Socket failed with error");
        WSACleanup();
        exit(2);
    }

    if(bind(listenSocket, (struct sockaddr *) &server, sizeof(server)) < 0){
        puts("Bind failed. Error");
        exit(3);
    }

    if(listen(listenSocket, SOCKET_ERROR) == SOCKET_ERROR){
        puts("Listen call failed. Error");
        exit(4);
    }

    HANDLE hAccept = CreateThread(NULL, 0, acceptConnections, (void *)
   listenSocket, 0, NULL);

    serverProcess();

    WaitForSingleObject(hAccept, INFINITE);

    WSACleanup();
    return 0;
}
```

А.2 Подключение клиентов

```
DWORD WINAPI acceptConnections(void *listenSocket){
    SOCKET acceptSocket;
```

```
    clientDescriptor desc;
    sockaddr_in clientInfo;
    int clientInfoSize = sizeof(clientInfo);

    while(true){
        int ind = -1;

        acceptSocket = accept((SOCKET)listenSocket, (struct sockaddr*)&
clientInfo, &clientInfoSize);

        if(acceptSocket == INVALID_SOCKET) break;
        else if(acceptSocket < 0){
            puts("Accept failed");
            continue;
        }

        WaitForSingleObject(hMutex, INFINITE);
        for (int i = maxThreads - 1; i >= 0; i--)
        if(clientDesc[i].sock == INVALID_SOCKET){
            ind = i;
        }
        ReleaseMutex(hMutex);

        if(ind < 0){
            char *ip  = inet_ntoa(clientInfo.sin_addr);

            sendMSG(acceptSocket, MSG_FULL);
            shutdown(acceptSocket, SD_BOTH);
            closesocket(acceptSocket);

            cout << ip << ":" << clientInfo.sin_port << " was disconnected
because of server overload" << endl;
            continue;
        }

        desc.sock = acceptSocket;
        desc.ip   = inet_ntoa(clientInfo.sin_addr);
        desc.port = clientInfo.sin_port;

        cout << "Connection request received." << endl;
        cout << "New socket was created at address " << desc.ip << ":" <<
desc.port << endl;

        clientDesc[ind] = desc;
        clientDesc[ind].handle = CreateThread(NULL, 0, (
LPTHREAD_START_ROUTINE)clientProcess, (void *)desc.sock, 0, NULL);
    }

    WaitForSingleObject(hMutex, INFINITE);
    vector<HANDLE> hClients;
    for(int i = 0; i < maxThreads; i++)
        if(clientDesc[i].sock != INVALID_SOCKET){
            hClients.push_back(clientDesc[i].handle);
            closeSocket(i);
        }
```

```
54      ReleaseMutex(hMutex);

56      if (!hClients.empty()){
            WaitForMultipleObjects(hClients.size(), hClients.data(), TRUE,
        INFINITE);
58      }
        return 0;
60  }
```

A.3 Структуры, используемые для работы с клиентами

```
1   struct clientDescriptor{
2       SOCKET sock;
        HANDLE handle;
4       char *ip;
        int port;
6       string login;
    };
8
    struct userData{
10      string login;
        string password;
12      string permissions;
        string path;
14      bool online;
    };
16
    clientDescriptor clientDesc[MAX_THREADS_POSSIBLE];
18  vector<userData> users;
```

A.4 Поток обработки клиентских сообщений

```
1   DWORD WINAPI clientProcess(void* socket){
2       int ind;
        bool exitFlag = false;
4       state state = CONNECT;
        char buffer[packetSize];
6       string line;

8       for (int i = 0; i < maxThreads; i++){
            if (clientDesc[i].sock == (SOCKET)socket) ind = i;
10      }

12      while(!exitFlag){
            string login;
14          string password;
            string cmd;
16          int logInd;
            size_t pos = 0;
18          line = "";

20          switch(state){
                case CONNECT:
```

```
22                  sendMSG(clientDesc[ind].sock, MSG_WELCOME);

24                  if(recvS(clientDesc[ind].sock, buffer, line) != 1){
                        exitFlag = true;
26                      break;
                    }else cout << clientDesc[ind].ip << ":" << clientDesc[ind].
    port << " " << line << endl;
28
                    pos = line.find(" ");
30                  cmd = line.substr(0, pos);

32                  if(cmd == "login" || cmd == "addusr"){
                        if(line.find_first_of(" ") != string::npos)
34                          line = line.substr(line.find_first_of(" "), string::
    npos);
                        if(line.find_first_not_of(" ") != string::npos)
36                          line = line.substr(line.find_first_not_of(" "),
    string::npos);

38                      pos = line.find(" ");
                        login = line.substr(0, pos);
40
                        if(line.find_first_of(" ") != string::npos)
42                          line = line.substr(line.find_first_of(" "), string::
    npos);
                        if(line.find_first_not_of(" ") != string::npos)
44                          line = line.substr(line.find_first_not_of(" "),
    string::npos);

46                      pos = line.find(" ");
                        password = line.substr(0, pos);
48                  }
                    else sendMSG(clientDesc[ind].sock, MSG_INVALID_CMD);
50
                    if(cmd == "login"){
52                      if(loginCommand(login, password) == 0){
                            logInd = getUserIndex(login);
54
                            WaitForSingleObject(hMutex,INFINITE);
56                          clientDesc[ind].login = login;
                            ReleaseMutex(hMutex);
58
                            state = WORK;
60                      }
                        else if(loginCommand(login, password) == 1){
62                          sendMSG(clientDesc[ind].sock, MSG_LOGPASS_NOT_MATCH);
                            break;
64                      }
                        else{
66                          sendMSG(clientDesc[ind].sock, MSG_USER_ONLINE);
                            break;
68                      }
                    }
70
                    if(cmd == "addusr"){
```

```
72                      if(addusrCommand(login, password) == 0){
                            logInd = getUserIndex(login);
74
                            WaitForSingleObject(hMutex,INFINITE);
76                          clientDesc[ind].login = login;
                            users[logInd].online = true;
78                          ReleaseMutex(hMutex);

80                          state = WORK;
                        }
82                      else if(addusrCommand(login, password) == 1){
                            sendMSG(clientDesc[ind].sock, MSG_USER_EXISTS);
84                          break;
                        }
86                      else if(addusrCommand(login, password) == 2){
                            cout << "ERROR: Can't open users.txt" << endl;
88                          return 2;
                        }
90                      else{
                            sendMSG(clientDesc[ind].sock, MSG_LOGPASS_MATCHING);
92                      }
                    }
94
                    break;
96              case WORK:
                    while(true){
98                      string cmd;
                        vector<string> names;
100
                        string prompt = users[logInd].login + " @ " + users[
    logInd].path + " $ \n";
102                     sendMSG(clientDesc[ind].sock, prompt);

104                     cmd = "";
                        if(recvS(clientDesc[ind].sock, buffer, cmd) != 1){
106                         exitFlag = true;
                            break;
108                     }else cout << clientDesc[ind].ip << ":" << clientDesc[ind
    ].port
                                   << "|" << clientDesc[ind].login << "| " << cmd
     << endl;
110
                        if(cmd == "ls"){
112                         names = lsCommand(users[logInd].path);

114                         for(int i = 0; i < names.size(); i++)
                                sendMSG(clientDesc[ind].sock, names[i] + "\n");
116                         names.clear();
                        }
118
                        else if(cmd.find("cd ") != string::npos){
120                         cmd = cmd.substr(3, string::npos);
                            cmd = cmd.substr(0, cmd.find_last_not_of(" ") + 1);
122
```

```
                        users[logInd].path = cdCommand(clientDesc[ind].sock,
    cmd, users[logInd].path);
124                     rewriteUserFile();
                    }
126
                    else if(cmd == "pwd"){
128                     sendMSG(clientDesc[ind].sock, users[logInd].path + "\
    n");
                    }
130
                    else if(cmd == "who"){
132                     if(users[logInd].permissions.find("w") != string::
    npos){
                            for(int i = 0; i < users.size(); i++){
134                             string online, send;
                                if(users[i].online) online = "ONLINE";
136                             send = users[i].login + "\t" + online + "\t" +
     users[i].path + "\n";
                                sendMSG(clientDesc[ind].sock, send);
138                         }
                        }else sendMSG(clientDesc[ind].sock,
    MSG_NO_PERMISSIONS);
140                 }

142                 else if(cmd.find("chmod ") != string::npos){
                        if(users[logInd].permissions.find("c") != string::
    npos){
144                         cmd = cmd.substr(6, string::npos);
                            cmd = cmd.substr(0, cmd.find_last_not_of(" ") + 1)
    ;
146
                            pos = cmd.find(" ");
148                         string login = cmd.substr(0, pos);

150                         if(login == "root" || login == users[logInd].
    login){
                                sendMSG(clientDesc[ind].sock,
    MSG_LOCK_PERMISSIONS);
152                             continue;
                            }
154
                            if(chmodCommand(cmd) == 0) rewriteUserFile();
156                         else sendMSG(clientDesc[ind].sock,
    MSG_USER_NOT_EXISTS);
                        }else sendMSG(clientDesc[ind].sock,
    MSG_NO_PERMISSIONS);
158                 }

160                 else if(cmd.find("kill ") != string::npos){
                        if(users[logInd].permissions.find("k") != string::
    npos){
162                         cmd = cmd.substr(5, string::npos);
                            cmd = cmd.substr(0, cmd.find_last_not_of(" ") + 1)
    ;
164
```

```
                            if(cmd == "root" || cmd == users[logInd].login){
                                sendMSG(clientDesc[ind].sock, MSG_LOCK_KILL);
                                continue;
                            }

                            if(killCommand(cmd) != 0) sendMSG(clientDesc[ind].sock, MSG_KILL_FAILED);
                        }else sendMSG(clientDesc[ind].sock, MSG_NO_PERMISSIONS);
                    }

                    else if(cmd == "logout"){
                        break;
                    }
                    else sendMSG(clientDesc[ind].sock, MSG_INVALID_CMD);
                }

                WaitForSingleObject(hMutex,INFINITE);
                users[logInd].online = false;
                clientDesc[ind].login = "";
                ReleaseMutex(hMutex);

                state = CONNECT;

                break;
            default:
                break;
        }
    }

    closeSocket(ind);
    return 0;
}
```

A.5 Функция чтения до перевода строки

```
int recvS(SOCKET socket, char *buf, string &line){
    int rc = 1;
    while(rc != 0){
        rc = recv(socket, buf, 1, 0);
        if(rc <= 0) return 0;
        if(buf[0] == '\n') return 1;
        line = line + buf[0];
    }
}
```

A.6 Функция закрытия сокета

```
void closeSocket(int ind){
    WaitForSingleObject(hMutex, INFINITE);
    if(ind >= 0 && ind < maxThreads && clientDesc[ind].sock != INVALID_SOCKET)
    {
            if(closesocket(clientDesc[ind].sock))
```

```
                puts("Closesocket failed");
            else{
                cout << clientDesc[ind].ip << ":" << clientDesc[ind].port <<
    " was disconnected" << endl;
                clientDesc[ind].sock = INVALID_SOCKET;
            }
     }
     ReleaseMutex(hMutex);
}
```

Приложение Б Часть UDP

Б.1 Инициализация сервера

```
int main(int argc, char *argv[]){

    /* ... */
    /* Обработка ключей при запуске */

    for(int i = 0; i < MAX_THREADS; i++) clientsData[i].id = -1;

    pthread_mutex_init(&mutex, NULL);

    mainSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    if(mainSocket < 0){
        puts("Socket failed with error");
        return 1;
    }

    memset((char *) &serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
    serverAddress.sin_port = htons(serverPort);

    if(bind(mainSocket, (struct sockaddr*) &serverAddress, sizeof(
   serverAddress)) < 0){
        puts("Bind failed. Error");
        return 2;
    }

    processClients();

    int status_addr;

    pthread_join(serverThread, (void**) &status_addr);
    for(int i = 0; i < MAX_THREADS; i++) {
        pthread_join(clients[i], (void**) &status_addr);  // ЧТО-ТО СТРАННОЕ
    }

    close(mainSocket);
    mainSocket = -1;

    puts("Server is stopped.\n");
    return 0;
}
```

Б.2 Подключение клиентов

```
void processClients(){
    pthread_create(&serverThread, NULL, serverProcess, NULL);

    while(serverStart){
        struct sockaddr_in clientAddress;
```

```
        int size = sizeof(struct sockaddr_in);
        char buff[BUFLEN] = "";
        bool newThread = true;
        bool exist = false;

        recvfrom(mainSocket, buff, BUFLEN, 0, (sockaddr *) &clientAddress, (socklen_t *) &size);
        int pos = findExistingClientsData(clientAddress);
        if(pos >= 0) newThread = false;
        else pos = findFreeClientsData();

        if(pos >= 0){
            clientStruct client;
            client.id = pos;
            memcpy(client.buffer, buff, sizeof(client.buffer));
            client.cl_sock = clientAddress;

                pthread_mutex_lock(&mutex);
                clientsData[pos] = client;
                pthread_mutex_unlock(&mutex);

            if(newThread && serverStart){
                cout << "Add new client " << inet_ntoa(clientAddress.sin_addr)

                     << ":" << clientAddress.sin_port << endl;

                pthread_create(&clients[pos], NULL, clientProcess, (void*) &pos);
            }
        }else{
            cout << "SERVER OVERLOAD, KILL TO FREE THREAD" << endl;
        }
    }
}
```

Б.3 Структура, используемая для работы с клиентами

```
struct clientStruct{
    int id;
    char buffer[BUFLEN];
    sockaddr_in cl_sock;
};

struct clientStruct clientsData[MAX_THREADS];
```

Б.4 Обработка действий клиента

```
void* clientProcess(void *args){
    int id = *((int*) args);
    string temp = clientsData[id].buffer;

    int numSpace = temp.find(' ', 0);
    int idPackageInt = atoi(temp.substr(0, numSpace).data()) - 1;
```

```
    string buffer = "";
    clock_t t1, t2;
    bool needCheck = false;

    while(clientsData[id].id >= 0){
        if(buffer != clientsData[id].buffer){
            buffer = clientsData[id].buffer;
            string answer;

            if(count(buffer.begin(), buffer.end(), ' ') > 0){
                //поиск ID пакета
                int numSpace = buffer.find(' ', 0);
                string idPackage = buffer.substr(0, numSpace);
                buffer.erase(0, numSpace + 1);

                if(needCheck){
                    t2 = clock();
                    if(buffer == "#CHECK" && ((double)(t2 - t1) /
CLOCKS_PER_SEC) < 3){
                        cout << "ID:" << id << " checked" << endl;
                    }
                    else
                        cout << "ID:" << id << " has not received a message"
<< endl;
                    needCheck = false;
                }
                else if(atoi(idPackage.data()) - idPackageInt == 1){
                    answer = "Echo: " + buffer + "\n";

                    idPackageInt = atoi(idPackage.data());
                    send(clientsData[id].cl_sock, idPackage + " " + answer);
                    needCheck = true;
                    t1 = clock();
                }
                else{
                    if(idPackageInt - atoi(idPackage.data()) == 0) answer =
SAME_ERROR;
                    else answer = MIXING_ERROR;
                    send(clientsData[id].cl_sock, answer);
                    needCheck = true;
                    t1 = clock();
                }
            }
            else{
                answer = FAILED_ERROR;
                send(clientsData[id].cl_sock, answer);
                needCheck = true;
                t1 = clock();
            }
        buffer = clientsData[id].buffer;
        }
    }
    pthread_mutex_lock(&mutex);
    clientsData[id].id = -1;
```

```
    cout << "ID:" << id << " removed from list" << endl;
    pthread_mutex_unlock(&mutex);
}
```

Б.5 Функция отправки сообщений на клиенте

```
int sendData(sockaddr_in server, string msg, int id_package) {
    memset(buf, 0, BUFFER_SIZE);

    string res = "";

    char buffer[8];
    sprintf(buffer, "%d", id_package);
    string id = buffer;

    res = id + " " + msg;

    int len = sizeof(server);
    sendto(clientSocket, res.data(), res.size(), 0, (struct sockaddr *) &
   server, len);
    string getAnswer = receiveData(server);

    if(getAnswer.empty()){
        cout << "PACKAGE " + id + " WAS LOST" << endl;
        cout << "NO CONNECTION" << endl;
    }
    else{
        int numSpace = getAnswer.find(" ", 0);
        if(numSpace < 0){
            cout << "PACKAGE " + id + " WAS LOST" << endl;
            cout << "RECEIVED: " + getAnswer << endl;
        }
        else{
            int id_getPackage = atoi(getAnswer.substr(0, numSpace).data());
            if(id_getPackage != id_package){
                cout << "PACKAGE " + id + " WAS LOST" << endl;
                cout << "RECEIVED: " + getAnswer << endl;
            }
            else{
                getAnswer.erase(0, numSpace + 1);
                cout << getAnswer << endl;
            }
        }
        res = id + " #CHECK";
        sendto(clientSocket, res.data(), strlen(res.data()), 0, (struct
   sockaddr *) &server, len);
    }
    return 0;
}
```

Б.6 Функция приема сообщений на клиенте

```
string receiveData(sockaddr_in &server) {
    memset(buf, 0, BUFFER_SIZE);

    int len = sizeof(server);

    struct timeval tv;
    tv.tv_sec = 2;
    tv.tv_usec = 0;

    fd_set rfds;
    FD_ZERO(&rfds);
    FD_SET(clientSocket, &rfds);

    int i = select(clientSocket + 1, &rfds, NULL, NULL, &tv);

    if(i > 0){
        recvfrom(clientSocket, buf, BUFFER_SIZE, 0, (struct sockaddr *) &
   server, &len);
        string res(buf);
        return res;
    }else{
        cout << "TIMEOUT" << endl;
        return "";
    }
    return "";
}
```