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

Курсовая работа на тему:

# Разработка сетевого приложения «Чат». Мультипроцессная реализация сервера, на базе протокола TCP; fork.

Выполнил: студент гр. ИП-013

Копытина Т.А.

Проверил:

ассистент кафедры ВС

Ревун А.Л.

Новосибирск, 2023 г.

# Оглавление

Постановка задачи ..... 3
Описание протокола ..... 4
Описание реализации ..... 5
Клиент ..... 5
Сервер ..... 7
Скан экрана работы программы ..... 10
Подключение клиентов ..... 10
Переписка ..... 11
Отключение клиентов ..... 12
Текст программы ..... 13
Client.cpp ..... 13
Server.cpp ..... 17
Список источников ..... 24

# Постановка задачи

Реализовать сетевое приложение «Чат», сервер должен быть мультипроцессным – реализован через функцию fork. Приложение должно быть на базе протокола TCP.

# Описание протокола

**Протокол TCP** (Transmission Control Protocol – протокол управления передачей) является ориентированным на соединение транспортным протоколом с надежной доставкой данных. Поэтому он имеет жесткие алгоритмы обнаружения ошибок, разработанные для обеспечения целостности передаваемых данных.

Для обеспечения надежной доставки применяется последовательная нумерация и подтверждение. С помощью последовательной нумерации определяется порядок следования данных в пакетах и выявляются пропущенные пакеты. Последовательная нумерация с подтверждением позволяет организовать надежную связь, которая называется полным дуплексом (full duplex). Каждая сторона соединения обеспечивает собственную нумерацию для другой стороны.

TCP – является байтовым последовательным протоколом. В отличие от пакетных последовательных протоколов, он присваивает последовательный номер каждому передаваемому байту пакета, а не каждому пакету в отдельности.

Формат заголовка TCP-пакета:

<table>
<tr><td>Offset</td><td colspan="10">0 7 8 15 16 31</td></tr>
<tr><td>0</td><td colspan="5">Source Port</td><td colspan="5">Destination Port</td></tr>
<tr><td>2</td><td colspan="10">Sequence Number</td></tr>
<tr><td>8</td><td colspan="10">Acknowledgement Number</td></tr>
<tr><td>1</td><td>Data Offset</td><td colspan="2">Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window</td></tr>
<tr><td>1</td><td colspan="5">Checksum</td><td colspan="5">Urgent Pointer</td></tr>
<tr><td>2</td><td colspan="8">Options</td><td colspan="2">Padding</td></tr>
</table>

# Описание реализации

## Клиент

При запуске клиент указывает *ip* и *порт сервера*, а также *никнейм*, отображаемый в чате, после чего происходит связывание и подключение к серверу.

```
#pragma region Connecting
if(send(sock, name.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
if(recv(sock, buff, BUFFSIZE, 0) == -1) std::cout << "Recieve error" << std::endl;

std::string onNameAnswer(buff);
std::cout << onNameAnswer << std::endl;
if(onNameAnswer.find("exist") != std::string::npos){
    exit(1);
}

if(recv(sock, buff, BUFFSIZE, 0) == -1) std::cout << "Recieve error" << std::endl;
std::string usersList(buff);
std::cout << usersList << std::endl;

std::thread asyncReciever(AsyncSend, std::ref(sock));
asyncReciever.detach();
#pragma endregion Connecting
```

Здесь серверу отправляется имя клиента и ожидается ответ об успешном подключении. Если такое имя уже есть у другого пользователя, то клиенту нужно будет указать новое.
Когда все клиенты подключатся, то каждому приходит список со всеми участниками чата и выводится на экран, после чего у клиента запускается в поток функция **AsyncSend**, считывающая его сообщения.

```
void AsyncSend(int& sock){
    while(1){
        std::string message = "";
        std::cin >> message;

        if(needThreadTerminate){
            threadTerminated = true;
            return;
        }

        if(!message.empty()){
            if(send(sock, message.c_str(), BUFFSIZE, 0) == -1){
                std::cout << "Send error" << std::endl;
            }
        }
        else{
            std::cout << "Message can't be empty, try again.\n";
        }
    }
}
```

Функция AsyncSend принимает дескриптор гнезда, куда нужно отправлять сообщения и в бесконечном цикле считывает ввод пользователя, и отправляет его на сервер, если оно не пустое. Поток завершается, когда главный поток запросит его завершение – при отключении клиента от чата.

```
    while(1){
        int msgLen = recv(sock, buff, BUFFSIZE, 0);
        if(msgLen == -1) std::cout << "Recieve error" << std::endl;


        std::string answer(buff);
        std::cout << answer << std::endl;
        if(answer.find("closed") != std::string::npos){
            needThreadTerminate = true;
            break;
        }

        if(answer.find("left chat.") != std::string::npos){

            std::string message("EraseClient " + answer.substr(0, answer.find(" ")));
            if(send(sock, message.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
        }
    }
    while(!threadTerminated){}

    close(sock);
    return 0;
}
```

В главном потоке клиент ожидает сообщения сервера, если сервер говорит об успешном отключении от чата, клиент завершает свою работу. Когда кто-то покидает чат, клиент получает уведомление, выводит его на экран и посылает своему сервер-процессу запрос на удаление такого клиента из своего списка. После окончания работы, клиент ждёт завершения потока ввода, закрывает гнездо и завершает работу полностью.

# Сервер

```
#pragma region Connecting
for(int i = 0; i < clientCount; i++){
    clientSocket = accept(listener, 0, 0);

    messageLength = recv(clientSocket, buff, BUFFSIZE, 0);
    if(messageLength == -1) std::cout << "Recieve error" << std::endl;

    buff[messageLength] = '\0';
    std::string message(buff);
    std::cout << "Request message: " << message << std::endl;

    auto sameName = std::find_if(Clients.begin(), Clients.end(), [&](const Client& client){return client.Name == message;});
    if(sameName != Clients.end()){
        std::string request("Server: This username already exist.\n");
        if(send(clientSocket, request.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
        close(clientSocket);
        i--;
    }

    Clients.push_back(Client{message, clientSocket});
    message = "Server: Connected.";
    if(send(clientSocket, message.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
}
```

После инициализации и создания своего гнезда сервер ожидает подключения указанного количества клиентов. При подключении он получает имя клиента и, если оно не занято, добавляет его в свой список и сообщает клиенту об успешном подключении.

```
for(const auto& user : Clients){
    std::string availableClients = "";
    int availableClientsCount = Clients.size() - 1;
    availableClients += "Server: " + std::to_string(availableClientsCount) + " available users:\n";

    int clientNum = 1;
    for(const auto& client : Clients){
        if(client.Name  != user.Name){
            availableClients += std::to_string(clientNum) + ". " + client.Name + "\n";
            clientNum++;
        }
    }

    availableClients += "Server: Chat Started.\n";
    if(send(user.Descriptor, availableClients.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
}

UpdateClientsFile();
#pragma endregion Connecting
```

После подключения всех клиентов сервер отправляет каждому из них список активных пользователей в чате.

```
    for(int i = 0; i < clientCount; i++) {
        clientSocket = Clients[i].Descriptor;
        std::string clientName = Clients[i].Name;
        switch(fork()){
            case -1:{
                printf("Error in fork\n");
                exit(1);
            }
            case 0:{
            default:{

                while(waitpid(-1, NULL, WNOHANG) > 0);
            }
        }
    }
    wait(NULL);
    close(listener);

    return 0;
}
```

После чего на каждого клиента создается копия процесса через fork, для копии происходит переход к бесконечному циклу обработки, а родительский процесс закрывает за собой сокеты и просто ожидает завершения дочерних процессов.

```
void reaper(int sig) {
    int status;
    while(wait3(&status, WNOHANG, (struct rusage*)0) >= 0);
}
```

После завершения работы сервера с клиентом вызывается функция reaper, которая полностью завершает процесс, чтобы он не стал «зомби».

```cpp
while (1)
{
    messageLength = recv(clientSocket, buff, BUFFSIZE, 0);
    if(messageLength == -1) std::cout << "Recieve error" << std::endl;

    buff[messageLength] = '\0';
    std::string message(buff);
    std::cout << "Request message: " << message << std::endl;

    if(1){
        if(message == "disconnect"){
            std::string info("Server: Chat closed.\n");
            if(send(clientSocket, info.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;

            info = std::string(clientName + " left chat.");
            for(const auto& client : Clients){
                if(client.Name != clientName){
                    std::cout << "Reciever: " << client.Name << " Answer: " << info << std::endl << std::endl;
                    if(send(client.Descriptor, info.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
                }
            }
            break;
        }

        if(message.find("EraseClient") != std::string::npos){
            std::string deleteName(message.substr(message.find(" ") + 1));
            std::cout << deleteName << std::endl;
            auto deleteClient = std::find_if(Clients.begin(), Clients.end(), [&](const Client& client){return client.Name == deleteName;});
            Clients.erase(deleteClient);
            continue;
        }

        std::string messageForReciever = clientName + ": " + message;

        for(const auto& client : Clients){
            if(client.Name != clientName){
                std::cout << "Reciever: " << client.Name << " Answer: " << messageForReciever << std::endl << std::endl;
                if(send(client.Descriptor, messageForReciever.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
            }
        }
    }
```

В бесконечном цикле сервер принимает сообщения от клиента и пересылает их остальным участникам чата. Если клиент покидает чат, то сервер-процесс завершает свою работу, уведомив всех остальных участников чата об отключении клиента. При получении запроса на удаление покинувшего чат клиента, сервер удаляет его из своего списка. Если какой-либо из клиентов указывает «EraseClient *никнейм другого клиента*», то указанный пользователь не будет получать сообщения от пользователя, который вызвал данную команду.

# Скан экрана работы программы

## Подключение клиентов

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Mable
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Sara
2. Garry
Server: Chat Started.
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Sara
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Mable
2. Garry
Server: Chat Started.
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Garry
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Mable
2. Sara
Server: Chat Started.
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Server
Server: Port Number - 42083
SERVER: IP 127.0.0.1
Request message: Mable
Request message: Sara
Request message: Garry
```

# Переписка

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Mable
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Sara
2. Garry
Server: Chat Started.

Garry: Hi
Sara: Hello
Yo
Sara: loving
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Sara
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Mable
2. Garry
Server: Chat Started.

Garry: Hi
Hello
Mable: Yo
loving
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Garry
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Mable
2. Sara
Server: Chat Started.

Hi
Sara: Hello
Mable: Yo
Sara: loving
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Server
Server: Port Number - 42083
SERVER: IP 127.0.0.1
Request message: Mable
Request message: Sara
Request message: Garry
Request message: Hi
Reciever: Mable Answer: Garry: Hi

Reciever: Sara Answer: Garry: Hi

Request message: Hello
Reciever: Mable Answer: Sara: Hello

Reciever: Garry Answer: Sara: Hello

Request message: Yo
Reciever: Sara Answer: Mable: Yo

Reciever: Garry Answer: Mable: Yo

Request message: loving
Reciever: Mable Answer: Sara: loving

Reciever: Garry Answer: Sara: loving
```

## Отключение клиентов

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Mable
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Sara
2. Garry
Server: Chat Started.

Garry: Hi
Sara: Hello
Yo
Sara: loving
Garry: desconnect
Garry left chat.
disconnect
Server: Chat closed.
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Sara
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Mable
2. Garry
Server: Chat Started.

Garry: Hi
Hello
Mable: Yo
loving
Garry: desconnect
Garry left chat.
Mable left chat.
disconnect
Server: Chat closed.
```

```
firefox@localhost:/mnt/c/network/kurs1$ ./Client 127.0.0.1 42083 Garry
CLIENT: Port number - 42083
Server: Connected.
Server: 2 available users:
1. Mable
2. Sara
Server: Chat Started.

Hi
Sara: Hello
Mable: Yo
Sara: loving
desconnect
disconnect
Server: Chat closed.
```

```
Request message: desconnect
Reciever: Mable Answer: Garry: desconnect

Reciever: Sara Answer: Garry: desconnect

Request message: disconnect
Reciever: Mable Answer: Garry left chat.

Reciever: Sara Answer: Garry left chat.

Request message: EraseClient Garry
Garry
Request message: EraseClient Garry
Garry
Request message: disconnect
Reciever: Sara Answer: Mable left chat.

Request message: EraseClient Mable
Mable
Request message: disconnect
```

# Текст программы

## Client.cpp

```
#include <netdb.h>

#include <netinet/in.h>

#include <stdio.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <errno.h>

#include <stdlib.h>

#include <strings.h>

#include <string.h>


#include <arpa/inet.h>

#include <unistd.h>


#include <string>

#include <thread>

#include <iostream>


#define BUFFSIZE 1024


bool threadTerminated = false;

bool needThreadTerminate = false;


void AsyncSend(int& sock){

    while(1){

        std::string message = "";

        std::cin >> message;


        if(needThreadTerminate){

            threadTerminated = true;

            return;

        }
```

```
        if(!message.empty()){
            if(send(sock, message.c_str(), BUFFSIZE, 0) == -1){
                std::cout << "Send error" << std::endl;
            }
        }
        else{
            std::cout << "Message can't be empty, try again.\n";
        }
    }
}

int main(int argc, char* argv[]) {
    #pragma region Init
    int sock;
    char buff[BUFFSIZE];

    if (argc < 4) {
        printf("Wrong input: should be 'ip address, port, name.'\n");
        exit(1);
    }

    std::string name(argv[3]);
    if(name.empty()){
        std::cout << "Name is empty.";
        exit(1);
    }

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Can't get socket\n");
        exit(1);
    }

    struct sockaddr_in addres;
```

```
addres.sin_family = AF_INET;
addres.sin_port = htons(atoi(argv[2]));
addres.sin_addr.s_addr = inet_addr(argv[1]);

if(connect(sock, (struct sockaddr*)&addres, sizeof(addres)) < 0){
    printf("Error in connect\n");
    exit(1);
}

printf("CLIENT: Port number - %d\n", ntohs(addres.sin_port));
#pragma endregion Init

#pragma region Connecting
if(send(sock, name.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" << std::endl;
if(recv(sock, buff, BUFFSIZE, 0) == -1) std::cout << "Recieve error" << std::endl;

std::string onNameAnswer(buff);
std::cout << onNameAnswer << std::endl;
if(onNameAnswer.find("exist") != std::string::npos){
    exit(1);
}

if(recv(sock, buff, BUFFSIZE, 0) == -1) std::cout << "Recieve error" << std::endl;
std::string usersList(buff);
std::cout << usersList << std::endl;

std::thread asyncReciever(AsyncSend, std::ref(sock));
asyncReciever.detach();
#pragma endregion Connecting

while(1){
    int msgLen = recv(sock, buff, BUFFSIZE, 0);
    if(msgLen == -1) std::cout << "Recieve error" << std::endl;
```

```
        std::string answer(buff);
        std::cout << answer << std::endl;
        if(answer.find("closed") != std::string::npos){
            needThreadTerminate = true;
            break;
        }
        if(answer.find("left chat.") != std::string::npos){

            std::string message("EraseClient " + answer.substr(0, answer.find(" ")));
            if(send(sock, message.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" <<
std::endl;
        }
    }
    while(!threadTerminated){}

    close(sock);
    return 0;
}
```

# Server.cpp

```
#include <netdb.h>

#include <netinet/in.h>

#include <stdio.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <errno.h>

#include <stdlib.h>

#include <strings.h>

#include <arpa/inet.h>

#include <signal.h>

#include <unistd.h>

#include <sys/wait.h>


#include <string>

#include <vector>

#include <iostream>

#include <fstream>

#include <algorithm>


#define BUFFSIZE 1024


struct Client{

    std::string Name = "";

    int Descriptor;

};


std::vector<Client> Clients;

std::string filename = "Clients.txt";

std::fstream ClientsFile;


void reaper(int sig) {

    int status;

    while(wait3(&status, WNOHANG, (struct rusage*)0) >= 0);
```

```
}


void RefreshClients(){
    std::string name;
    int descriptor;
    auto clientsCopy = Clients;
    Clients.clear();

    ClientsFile.clear();
    ClientsFile.seekg(0);

    while(ClientsFile >> name >> descriptor){
        Clients.push_back(Client{name, descriptor});
    }

    for(const auto& oldClient : clientsCopy){
        if(std::find_if(Clients.begin(), Clients.end(), [&](const Client& client){return
client.Name == oldClient.Name;}) == Clients.end()){
            close(oldClient.Descriptor);
        }
    }
}

void UpdateClientsFile(){
    ClientsFile.close();
    ClientsFile.open("Clients.txt", std::fstream::out | std::fstream::in);

    for(const auto& client : Clients){
        ClientsFile << client.Name << " " << client.Descriptor << std::endl;
    }
}

int main(int argc, char* argv[])
{
    #pragma region Init
```

```
ClientsFile.open(filename.c_str(), std::fstream::out | std::fstream::in | std::fstream::trunc);
int clientSocket;
int listener;
int messageLength;
char buff[BUFFSIZE];
socklen_t length = sizeof(struct sockaddr);
struct sockaddr_in serverAddr;

listener = socket(AF_INET, SOCK_STREAM, 0);
serverAddr.sin_family = AF_INET;
inet_aton("127.0.0.1",&serverAddr.sin_addr);
serverAddr.sin_port = 0;

if (bind(listener, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
    printf("Failed binding\n");
    exit(1);
}

listen(listener, 5);
if (getsockname(listener, (struct sockaddr*)&serverAddr, &length)) {
    printf("Error when calling getsockname\n");
    exit(1);
}

printf("Server: Port Number - %d\n", ntohs(serverAddr.sin_port));
printf("SERVER: IP %s\n", inet_ntoa(serverAddr.sin_addr));

signal(SIGCHLD, reaper);
#pragma endregion Init

int clientCount = 3;

#pragma region Connecting
for(int i = 0; i < clientCount; i++){
```

```
        clientSocket = accept(listener, 0, 0);


        messageLength = recv(clientSocket, buff, BUFFSIZE, 0);

        if(messageLength == -1) std::cout << "Recieve error" << std::endl;


        buff[messageLength] = '\0';

        std::string message(buff);

        std::cout << "Request message: " << message << std::endl;


        auto sameName = std::find_if(Clients.begin(), Clients.end(), [&](const Client&
client){return client.Name == message;});

        if(sameName != Clients.end()){

            std::string request("Server: This username already exist.\n");

            if(send(clientSocket, request.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" <<
std::endl;

            close(clientSocket);

            i--;

        }


        Clients.push_back(Client{message, clientSocket});

        message = "Server: Connected.";

        if(send(clientSocket, message.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send error" <<
std::endl;

    }


    for(const auto& user : Clients){

        std::string availableClients = "";

        int availableClientsCount = Clients.size() - 1;

        availableClients += "Server: " + std::to_string(availableClientsCount) + " available
users:\n";


        int clientNum = 1;

        for(const auto& client : Clients){

            if(client.Name  != user.Name){

                availableClients += std::to_string(clientNum) + ". " + client.Name + "\n";

                clientNum++;
```

```
            }

        }


        availableClients += "Server: Chat Started.\n";

        if(send(user.Descriptor, availableClients.c_str(), BUFFSIZE, 0) == -1) std::cout << "Send
error" << std::endl;

    }


    UpdateClientsFile();

    #pragma endregion Connecting


    for(int i = 0; i < clientCount; i++) {

        clientSocket = Clients[i].Descriptor;

        std::string clientName = Clients[i].Name;

        switch(fork()){

            case -1:{

                printf("Error in fork\n");

                exit(1);

            }

            case 0:{

                close(listener);


                while (1)

                {

                    messageLength = recv(clientSocket, buff, BUFFSIZE, 0);

                    if(messageLength == -1) std::cout << "Recieve error" << std::endl;


                    buff[messageLength] = '\0';

                    std::string message(buff);

                    std::cout << "Request message: " << message << std::endl;


                    if(1){

                        if(message == "disconnect"){

                            std::string info("Server: Chat closed.\n");
```

```
                            if(send(clientSocket, info.c_str(), BUFFSIZE, 0) == -1) std::cout <<
"Send error" << std::endl;



                            info = std::string(clientName + " left chat.");

                            for(const auto& client : Clients){

                                if(client.Name != clientName){

                                    std::cout << "Reciever: " << client.Name << " Answer: " << info
<< std::endl << std::endl;

                                    if(send(client.Descriptor, info.c_str(), BUFFSIZE, 0) == -1)
std::cout << "Send error" << std::endl;

                                }

                            }

                            break;

                        }



                        if(message.find("EraseClient") != std::string::npos){

                            std::string deleteName(message.substr(message.find(" ") + 1));

                            std::cout << deleteName << std::endl;

                            auto deleteClient = std::find_if(Clients.begin(), Clients.end(),
[&](const Client& client){return client.Name == deleteName;});

                            Clients.erase(deleteClient);

                            continue;

                        }



                        std::string messageForReciever = clientName + ": " + message;



                        for(const auto& client : Clients){

                            if(client.Name != clientName){

                                std::cout << "Reciever: " << client.Name << " Answer: " <<
messageForReciever << std::endl << std::endl;

                                if(send(client.Descriptor, messageForReciever.c_str(), BUFFSIZE, 0)
== -1) std::cout << "Send error" << std::endl;

                            }

                        }

                    }

                }
```

```
                auto sender = std::find_if(Clients.begin(), Clients.end(), [&](const Client&
client){return client.Name == clientName;});

                Clients.erase(sender);

                UpdateClientsFile();



                close(clientSocket);

                exit(0);

            }

            default:{

                //close(clientSocket);

                while(waitpid(-1, NULL, WNOHANG) > 0);

            }

        }

    }

    wait(NULL);

    close(listener);



    return 0;

}
```

# Список источников


1. Руководство по C++ [Электронный ресурс]. – URL: https://metanit.com/cpp/
2. Проект OpenNET [Электронный ресурс]. – URL: https://www.opennet.ru/
3. The Linux Programming Interface [Электронный ресурс]. – URL: https://man7.org/index.html
4. Протоколы TCP/IP и разработка сетевых приложений: учеб. пособие / К.В. Павский ; Сиб. гос. ун-т телекоммуникаций и информатики. - Новосибирск: СибГУТИ, 2013. – 130с.
5. https://studfile.net/preview/2947071/page:2/