МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Брестский государственный технический университет»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №2

По дисциплине «Аппаратное и программное обеспечение сетей»

Тема: «Организация UDP – сервера и UDP – клиента»

Выполнил:

студент 3 курса

группы ИИ-23

Макаревич Н.Р.

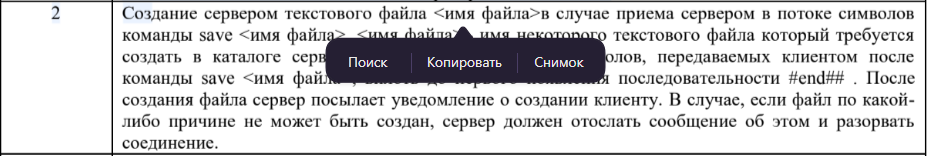
Проверил:

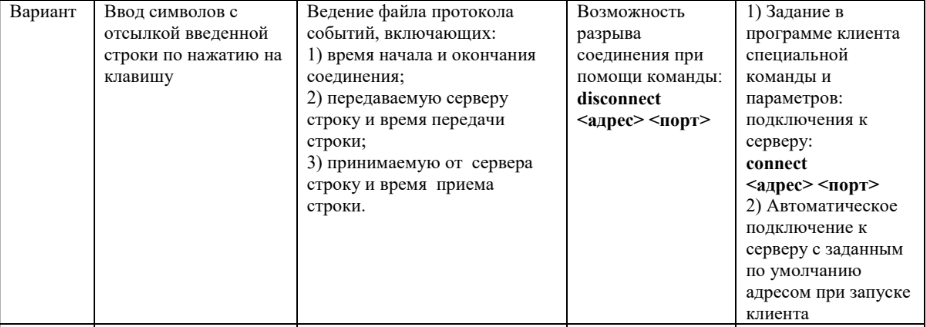
Степанчук В.И.

Брест 2024

Цель работы: 1) изучить основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрести навыки по практическому использованию библиотеки для реализации сетевых приложений в среде С++ на базе протоколов TCP.

Вариант 3.





**Реализация TCP-сервера:**

#include <iostream>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <stdio.h>

#include <vector>

#include <fstream>

#pragma comment(lib, "Ws2\_32.lib")

int main() {

WSADATA wsData;

int erStat = WSAStartup(MAKEWORD(2, 2), &wsData);

if (erStat != 0) {

std::cout << "Error WinSock version initializaion #";

std::cout << WSAGetLastError();

return 1;

}

else std::cout << "WinSock initialization is OK\n";

SOCKET ServSock = socket(AF\_INET, SOCK\_DGRAM, 0);

if (ServSock == INVALID\_SOCKET) {

std::cout << "Error initialization socket # " << WSAGetLastError() << "\n";

closesocket(ServSock);

WSACleanup();

return 1;

}

else std::cout << "Server socket initialization is OK\n";

in\_addr ip\_to\_num;

erStat = inet\_pton(AF\_INET, "127.0.0.1", &ip\_to\_num);

if (erStat <= 0) {

std::cout << "Error in IP translation to special numeric format\n";

return 1;

}

sockaddr\_in servInfo;

ZeroMemory(&servInfo, sizeof(servInfo));

servInfo.sin\_family = AF\_INET;

servInfo.sin\_addr = ip\_to\_num;

servInfo.sin\_port = htons(666);

erStat = bind(ServSock, (sockaddr\*)&servInfo, sizeof(servInfo));

if (erStat != 0) {

std::cout << "Error Socket binding to server info. Error # " << WSAGetLastError() << "\n";

closesocket(ServSock);

WSACleanup();

return 1;

}

else std::cout << "Binding socket to Server info is OK\n";

erStat = listen(ServSock, SOMAXCONN);

if (erStat != 0) {

std::cout << "Can't start to listen to. Error # " << WSAGetLastError() << "\n";

closesocket(ServSock);

WSACleanup();

return 1;

}

else std::cout << "Listening...\n";

sockaddr\_in clientInfo;

ZeroMemory(&clientInfo, sizeof(clientInfo));

int clientInfo\_size = sizeof(clientInfo);

SOCKET ClientConn = accept(ServSock, (sockaddr\*)&clientInfo, &clientInfo\_size);

if (ClientConn == INVALID\_SOCKET) {

std::cout << "Client detected, but can't connect to a client. Error # " << WSAGetLastError() << "\n";

closesocket(ServSock);

closesocket(ClientConn);

WSACleanup();

return 1;

}

else std::cout << "Connection to a client established successfully\n";

const short BUFF\_SIZE = 1024;

std::vector <char> servBuff(BUFF\_SIZE), clientBuff(BUFF\_SIZE);

short packet\_size = 0;

bool is\_saving = false;

std::ofstream fout("save.txt");

if (!fout.is\_open()) {

std::cout << "Can't open file. Ending session\n";

std::string message = "Error. Can't open file";

clientBuff.assign(message.begin(), message.end());

packet\_size = send(ClientConn, clientBuff.data(), clientBuff.size(), 0);

shutdown(ClientConn, SD\_BOTH);

closesocket(ServSock);

closesocket(ClientConn);

WSACleanup();

fout.close();

return 1;

}

fout.clear();

while (true) {

packet\_size = recv(ClientConn, servBuff.data(), servBuff.size(), 0);

std::cout << "Client's message: " << servBuff.data() << "\n";

if (servBuff[0] == 's' && servBuff[1] == 'a' && servBuff[2] == 'v' && servBuff[3] == 'e')

is\_saving = true;

if (is\_saving)

fout << servBuff.data() << "\n";

if (servBuff[0] == '#' && servBuff[1] == 'e' && servBuff[2] == 'n' && servBuff[3] == 'd' && servBuff[4] == '#' && servBuff[5] == '#') {

std::string message = "Succesfully saved";

clientBuff.assign(message.begin(), message.end());

is\_saving = false;

}

else {

std::cout << "Your (host) message: ";

fgets(clientBuff.data(), clientBuff.size(), stdin);

}

if (clientBuff[0] == 'x' && clientBuff[1] == 'x' && clientBuff[2] == 'x') {

shutdown(ClientConn, SD\_BOTH);

break;

}

packet\_size = send(ClientConn, clientBuff.data(), clientBuff.size(), 0);

if (packet\_size == SOCKET\_ERROR) {

std::cout << "Can't send message to Client. Error # " << WSAGetLastError() << "\n";

break;

}

}

fout.close();

closesocket(ServSock);

closesocket(ClientConn);

WSACleanup();

}

**Реализация TCP-клиента:**

#include <iostream>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <stdio.h>

#include <vector>

#include <fstream>

#include <ctime>

#include <iomanip>

#include <conio.h>

#pragma comment(lib, "Ws2\_32.lib")

using namespace std;

bool connect(std::string ip, int port, SOCKET& ClientSock);

void custom\_input(vector <char>& buff);

int main()

{

WSADATA wsData;

int erStat = WSAStartup(MAKEWORD(2, 2), &wsData);

if (erStat != 0) {

cout << "Error WinSock version initializaion #";

cout << WSAGetLastError();

return 1;

}

else cout << "WinSock initialization is OK" << endl;

SOCKET ClientSock = socket(AF\_INET, SOCK\_DGRAM, 0);

if (ClientSock == INVALID\_SOCKET) {

cout << "Error initialization socket # " << WSAGetLastError() << endl;

closesocket(ClientSock);

WSACleanup();

return 1;

}

else cout << "Client socket initialization is OK" << endl;

bool is\_connected = false; //writing ip and port

std::string ip;

int port;

while (!is\_connected) {

std::cout << "Enter ip: ";

std::cin >> ip;

std::cout << "Enter port: ";

std::cin >> port;

is\_connected = connect(ip, port, ClientSock);

}

if (erStat != 0) {

cout << "Connection to Server is FAILED. Error # " << WSAGetLastError() << endl;

closesocket(ClientSock);

WSACleanup();

return 1;

}

else cout << "Connection established SUCCESSFULLY. Ready to send a message to Server\n";

const int BUFF\_SIZE = 1024;

vector <char> servBuff(BUFF\_SIZE), clientBuff(BUFF\_SIZE);

short packet\_size = 0;

ofstream fout("log.txt");

while (true) {

cout << "Your (Client) message to Server: ";

custom\_input(clientBuff);

if (clientBuff[0] == 'x' && clientBuff[1] == 'x' && clientBuff[2] == 'x') {

shutdown(ClientSock, SD\_BOTH);

break;

}

packet\_size = send(ClientSock, clientBuff.data(), clientBuff.size(), 0);

std::time\_t now = std::time(nullptr);

std::tm localTime;

localtime\_s(&localTime, &now);

fout << std::put\_time(&localTime, "%Y-%m-%d %H:%M:%S") << " : " << clientBuff.data() << "\n";

if (packet\_size == SOCKET\_ERROR) {

cout << "Can't send message to Server. Error # " << WSAGetLastError() << endl;

break;

}

packet\_size = recv(ClientSock, servBuff.data(), servBuff.size(), 0);

now = std::time(nullptr);

localtime\_s(&localTime, &now);

if (packet\_size == SOCKET\_ERROR) {

cout << "Can't receive message from Server. Error # " << WSAGetLastError() << endl;

break;

}

else {

fout << std::put\_time(&localTime, "%Y-%m-%d %H:%M:%S") << " : " << servBuff.data() << "\n";

cout << "Server message: " << servBuff.data() << endl;

}

}

fout.close();

closesocket(ClientSock);

WSACleanup();

}

bool connect(std::string ip, int port, SOCKET& ClientSock) {

char cip[100];

strncpy\_s(cip, ip.c\_str(), sizeof(cip) - 1);

cip[sizeof(cip) - 1] = '\0';

in\_addr ip\_to\_num;

int erStat = inet\_pton(AF\_INET, cip, &ip\_to\_num);

if (erStat <= 0) {

cout << "Error in IP translation to special numeric format" << endl;

return false;

}

sockaddr\_in servInfo;

ZeroMemory(&servInfo, sizeof(servInfo));

servInfo.sin\_family = AF\_INET;

servInfo.sin\_addr = ip\_to\_num;

servInfo.sin\_port = htons(port);

erStat = connect(ClientSock, (sockaddr\*)&servInfo, sizeof(servInfo));

if (erStat == SOCKET\_ERROR) {

int error = WSAGetLastError();

cout << "Connection failed. Error #" << error << endl;

return false;

}

return true;

}

void custom\_input(vector <char>& buff) {

int index = 0;

while (1) {

char ch = \_getch();

if (ch == 109) {

printf("\n");

break;

}

if (ch == 8 && index > 0) {

buff[--index] = '\0';

printf("\b \b");

}

else if (ch >= 32 && ch <= 126 && index < buff.size() - 1) {

buff[index++] = ch;

printf("%c", ch);

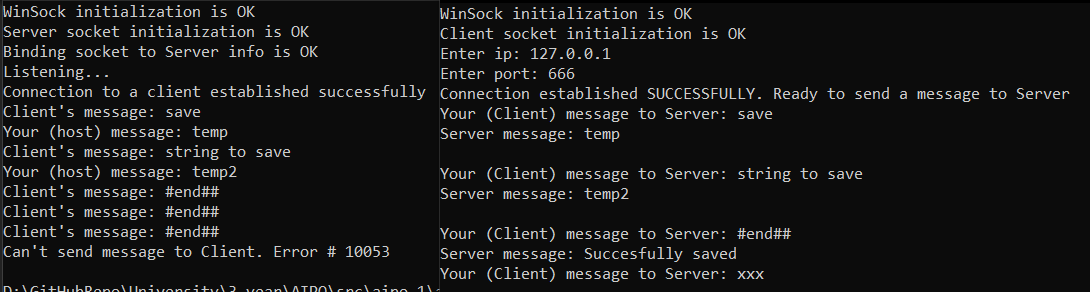
}

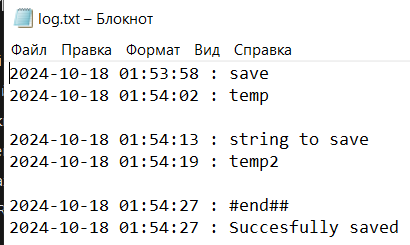
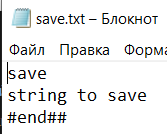
}

buff[index] = '\0';

}

**Результат работы сервера и клиента:**



**Вывод:** 1) изучил основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрёл навыки по практическому использованию библиотеки для реализации сетевых приложений в среде С++ на базе протоколов TCP.