Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе №7

на тему

**СРЕДСТВА ОБМЕНА ДАННЫМИ (WINDOWS). ИЗУЧЕНИЕ И ИСПОЛЬЗОВАНИЕМ СРЕДСТВ ОБМЕНА ДАННЫМИ И СОВМЕСТНОГО ДОСТУПА**

Студент О. Л. Дайнович

Преподаватель Н. Ю. Гриценко

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc151690323)

[2 Теоретические сведения 4](#_Toc151690324)

[3 Описание функций программы 6](#_Toc151690325)

[Заключение 9](#_Toc151690326)

[Список использованных источников 10](#_Toc151690327)

[Приложение А (обязательное) Листинг кода 11](#_Toc151690328)

# **1 ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является создание оконного приложения для обмена текстовыми сообщениями по локальной сети с использованием сокетов.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сокет (socket) - это программный интерфейс (API) для обмена данными между процессами, выполняющимися на разных узлах в компьютерной сети. Он предоставляет абстракцию для создания сетевых соединений и передачи данных по сети.

Сокеты работают на основе протоколов передачи данных, таких как TCP (Transmission Control Protocol) и UDP (User Datagram Protocol), и позволяют приложениям обмениваться информацией через сетевое соединение.

API для работы с сокетами предоставляется операционной системой или сетевой библиотекой. Программист может использовать функции этого API для создания, привязки, прослушивания и управления сокетами, а также для передачи данных через сетевое соединение. [1]

Программа, использующая сокеты, обычно включает в себя создание сокета, настройку его параметров, установку соединения с другим узлом, передачу данных и закрытие сокета после завершения обмена данными.

Следует различать [клиентские](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и [серверные](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) сокеты. Клиентские сокеты грубо можно сравнить с конечными аппаратами [телефонной сети](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C_%D0%BE%D0%B1%D1%89%D0%B5%D0%B3%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), а серверные — с [коммутаторами](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BC%D1%83%D1%82%D0%B0%D1%82%D0%BE%D1%80). Клиентское приложение (например, [браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80)) использует только клиентские сокеты, а серверное (например, [веб-сервер](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), которому браузер посылает запросы) — как клиентские, так и серверные сокеты.

[Интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) сокетов впервые появился в [BSD Unix](https://ru.wikipedia.org/wiki/BSD). [Программный интерфейс](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) сокетов описан в стандарте [POSIX](https://ru.wikipedia.org/wiki/POSIX).1 и в той или иной мере поддерживается всеми современными [операционными системами](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0).

Для взаимодействия между машинами с помощью стека протоколов [TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP) используются адреса и порты. Адрес представляет собой 32-битную структуру для протокола [IPv4](https://ru.wikipedia.org/wiki/IPv4), 128-битную для [IPv6](https://ru.wikipedia.org/wiki/IPv6). Номер порта — целое число в диапазоне от 0 до 65535 (для протокола [TCP](https://ru.wikipedia.org/wiki/TCP)).

Эта пара определяет сокет («гнездо», соответствующее [адресу](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%B0%D0%B4%D1%80%D0%B5%D1%81) и [порту](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D1%81%D0%B5%D1%82%D0%B8))).

В процессе обмена, как правило, используется два сокета — сокет отправителя и сокет получателя. Например, при обращении к серверу на [HTTP](https://ru.wikipedia.org/wiki/HTTP)-порт сокет будет выглядеть так: 194.106.118.30:80, а ответ будет поступать на mmm.nnn.ppp.qqq:xxxxx. [2]

Каждый [процесс](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) может создать «слушающий» сокет (серверный сокет) и привязать его к какому-нибудь [порту](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(TCP/UDP)) операционной системы (в [UNIX](https://ru.wikipedia.org/wiki/UNIX) непривилегированные процессы не могут использовать порты меньше 1024).

Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т. д.

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются [INET-адрес](https://ru.wikipedia.org/wiki/INET-%D0%B0%D0%B4%D1%80%D0%B5%D1%81) и [UNIX-адрес](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82_%D0%B4%D0%BE%D0%BC%D0%B5%D0%BD%D0%B0_Unix). Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут сообщаться любые локальные процессы путём чтения/записи из него. Сокеты типа [INET](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82%D1%8B_%D0%91%D0%B5%D1%80%D0%BA%D0%BB%D0%B8) доступны из сети и требуют выделения номера порта.

Обычно клиент явно «подсоединяется» к слушателю, после чего любое чтение или запись через его [файловый дескриптор](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B4%D0%B5%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%80) будут передавать данные между ним и сервером. [3]

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

В ходе выполнения лабораторной работы было разработано оконное приложение для обмена текстовыми сообщениями по локальной сети с использованием сокетов.

Были созданы приложения с реализацией сервера и клиента. После запуска сервера он принимает подключение от клиента (рисунок 1).

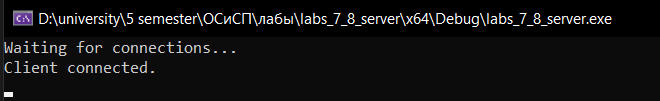


Рисунок 1 – Подключение клиента к серверу

В случае неудачного подключения к серверу выводится сообщение об ошибке (рисунок 2).

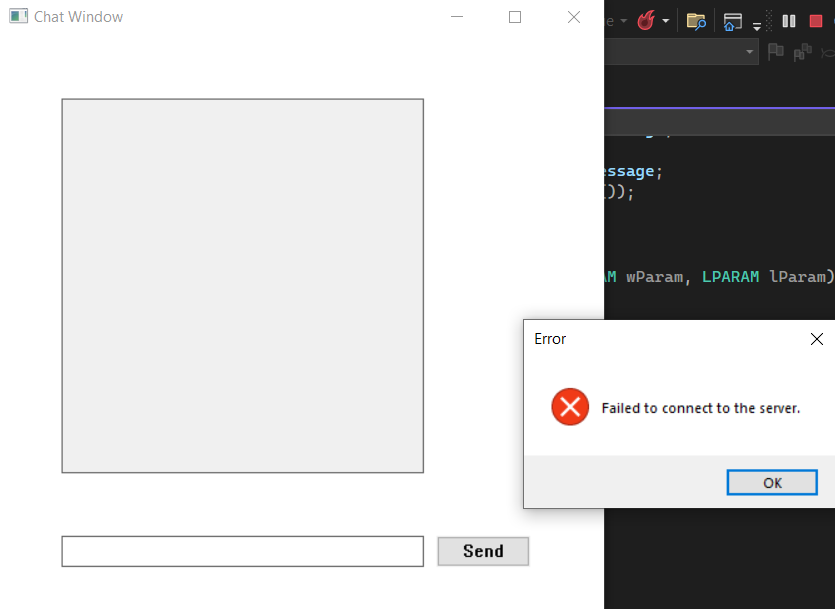


Рисунок 2 – Сообщение об ошибке подключения

После подключения клиента к серверу клиент и сервер могут обмениваться сообщениями, которые выводятся в окне приложения клиента (рисунок 3).

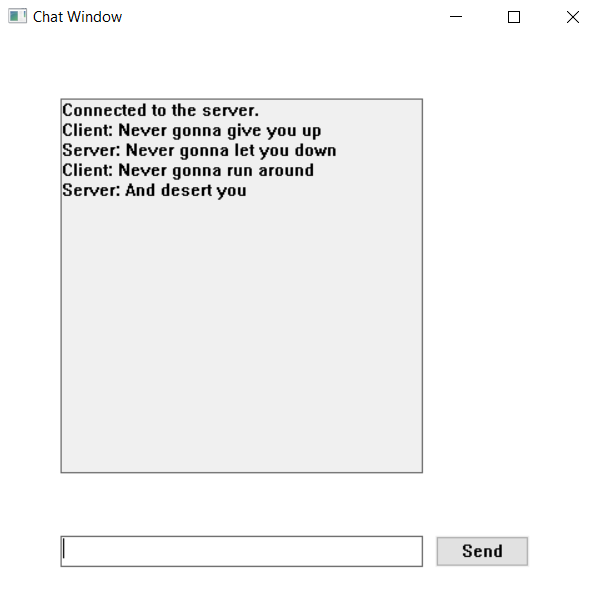


Рисунок 3 – Вывод сообщений на стороне клиента

Сообщения выводятся также и на стороне сервера (рисунок 4).

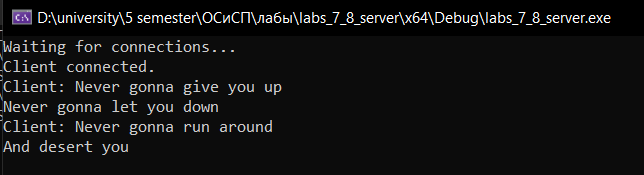


Рисунок 4 – Вывод сообщений на стороне сервера

После отключения клиента от сервера сервер получает соответствующее сообщение (рисунок 5).

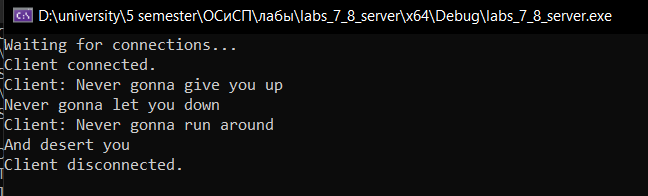


Рисунок 5 – Сообщение об отключении клиента от сервера

# **ЗАКЛЮЧЕНИЕ**

В результате лабораторной работы были изучены принципы работы с сокетами в Win32 API. Были созданы приложения клиента и сервера, позволяющие обмениваться текстовыми сообщениями по локальной сети с использованием сокетов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Microsoft Learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/.

[2] Начало работы с классическими приложениями для Windows, которые используют API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming.

[3] Что такое сокет и зачем он нужен [Электронный ресурс]. – Режим доступа: https://thecode.media/socket/

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 – Файл lab\_7\_8\_server.cpp

#include <iostream>

#include <winsock2.h>

#include <thread>

#include <string>

#pragma comment(lib, "WS2\_32.lib")

// Буфер для чтения и записи данных

const int BUFFER\_SIZE = 4096;

char buffer[BUFFER\_SIZE];

void ReceiveMessages(SOCKET clientSocket) {

while (true) {

int bytesRead = recv(clientSocket, buffer, BUFFER\_SIZE, 0);

if (bytesRead <= 0) {

std::cout << "Client disconnected." << std::endl;

break;

}

std::cout << "Client: " << buffer << std::endl;

// Очистка буфера

memset(buffer, 0, BUFFER\_SIZE);

}

}

void SendMessages(SOCKET clientSocket) {

while (true) {

std::string message;

std::getline(std::cin, message);

if (send(clientSocket, message.c\_str(), message.length(), 0) == SOCKET\_ERROR) {

std::cout << "Failed to send data to the client." << std::endl;

break;

}

}

}

int main() {

WSADATA wsaData;

SOCKET serverSocket, clientSocket;

struct sockaddr\_in serverAddress, clientAddress;

// Инициализация Winsock

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cout << "Failed to initialize socket." << std::endl;

return 1;

}

// Создание сокета

if ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET) {

std::cout << "Failed to create socket." << std::endl;

WSACleanup();

return 1;

}

// Настройка адреса сервера

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(12345); // Порт сервера

serverAddress.sin\_addr.s\_addr = INADDR\_ANY; // Принимать подключения на все IP-адреса

// Привязка сокета к адресу сервера

if (bind(serverSocket, (struct sockaddr\*)&serverAddress, sizeof(serverAddress)) == SOCKET\_ERROR) {

std::cout << "Failed to bind socket." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

// Прослушивание входящих подключений

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) {

std::cout << "Failed to listen on socket." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "Waiting for connections..." << std::endl;

// Принятие входящих подключений

int clientAddressSize = sizeof(clientAddress);

clientSocket = accept(serverSocket, (struct sockaddr\*)&clientAddress, &clientAddressSize);

if (clientSocket == INVALID\_SOCKET) {

std::cout << "Failed to accept client connection." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "Client connected." << std::endl;

std::thread receiveThread(ReceiveMessages, clientSocket);

std::thread sendThread(SendMessages, clientSocket);

receiveThread.join();

sendThread.join();

closesocket(clientSocket);

closesocket(serverSocket);

WSACleanup();

return 0;

}

Листинг 2 – Файл lab\_7\_8\_client.cpp

#include <iostream>

#include <winsock2.h>

#include <thread>

#include <string>

#include <Windows.h>

#include <CommCtrl.h>

#pragma comment(lib, "Comctl32.lib")

#pragma comment(linker, "\"/manifestdependency:type='win32' \ name = 'Microsoft.Windows.Common-Controls' version = '6.0.0.0' \

processorArchitecture = '\*' publicKeyToken = '6595b64144ccf1df' language = '\*'\"")

#pragma once

#pragma warning(disable: 4996)

#pragma comment(lib, "WS2\_32.lib")

#define ID\_BUTTON\_SEND 1000

#define EDIT\_MESSAGE 1001

const int BUFFER\_SIZE = 4096;

char buffer[BUFFER\_SIZE];

HWND hwnd; // Handle to the main window

HWND hShowMessage;

HWND hWriteMessage;

HWND hSendButton;

SOCKET clientSocket;

const UINT WM\_SOCKET\_MESSAGE = WM\_USER + 1; // Custom message for socket events

void ReceiveMessages(SOCKET clientSocket) {

while (true) {

int bytesRead = recv(clientSocket, buffer, BUFFER\_SIZE, 0);

if (bytesRead <= 0) {

PostMessage(hwnd, WM\_SOCKET\_MESSAGE, 0, 0);

break;

}

std::string message = "\nServer: ";

message += buffer;

int bufferSize = GetWindowTextLength(hShowMessage) + 1;

std::wstring history(bufferSize, L'\0');

GetWindowText(hShowMessage, &history[0], bufferSize);

size\_t buffer\_Size = history.length() + 1;

char\* nbuffer = new char[buffer\_Size];

size\_t convertedChars = 0;

wcstombs\_s(&convertedChars, nbuffer, buffer\_Size, history.c\_str(), \_TRUNCATE);

std::string str\_history(nbuffer);

delete[] nbuffer;

std::string new\_history = str\_history + message;

SetWindowTextA(hShowMessage, new\_history.c\_str());

memset(buffer, 0, BUFFER\_SIZE);

}

}

void SendMessages(SOCKET clientSocket, HWND hwnd) {

WCHAR message[256];

GetDlgItemText(hwnd, EDIT\_MESSAGE, message, 256);

int size = WideCharToMultiByte(CP\_UTF8, 0, message, -1, nullptr, 0, nullptr, nullptr);

char\* c\_message = new char[size];

WideCharToMultiByte(CP\_UTF8, 0, message, -1, c\_message, size, nullptr, nullptr);

if (send(clientSocket, c\_message, size, 0) == SOCKET\_ERROR) {

MessageBox(hwnd, L"Failed to send data to the server", L"Error", MB\_OK | MB\_ICONERROR);

}

else {

int bufferSize = GetWindowTextLength(hShowMessage) + 1;

std::wstring history(bufferSize, L'\0');

GetWindowText(hShowMessage, &history[0], bufferSize);

size\_t buffer\_Size = history.length() + 1;

char\* buffer = new char[buffer\_Size];

size\_t convertedChars = 0;

wcstombs\_s(&convertedChars, buffer, buffer\_Size, history.c\_str(), \_TRUNCATE);

std::string str\_history(buffer);

delete[] buffer;

std::string str\_message;

str\_message.assign(c\_message);

std::string full\_message = "\nClient: " + str\_message;

std::string new\_history = str\_history + full\_message;

SetWindowTextA(hShowMessage, new\_history.c\_str());

}

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_SOCKET\_MESSAGE:

MessageBox(hwnd, L"Connection closed by the server.", L"Message", MB\_OK | MB\_ICONERROR);

break;

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case ID\_BUTTON\_SEND:

SendMessages(clientSocket, hwnd);

break;

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow) {

WSADATA wsaData;

struct sockaddr\_in serverAddress;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

MessageBox(NULL, L"Failed to initialize socket.", L"Error", MB\_OK | MB\_ICONERROR);

return 1;

}

const wchar\_t className[] = L"MyWindowClass";

const wchar\_t windowTitle[] = L"Chat Window";

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = className;

RegisterClass(&wc);

hwnd = CreateWindowEx(

0,

className,

windowTitle,

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 500, 500,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

MessageBox(NULL, L"Failed to create the window.", L"Error", MB\_OK | MB\_ICONERROR);

WSACleanup();

return 1;

}

hShowMessage = CreateWindow(

L"STATIC", L"",

WS\_VISIBLE | WS\_CHILD | WS\_BORDER,

50, 50, 290, 300,

hwnd, NULL, hInstance, NULL);

hWriteMessage = CreateWindow(

L"EDIT", L"",

WS\_VISIBLE | WS\_CHILD | WS\_BORDER,

50, 400, 290, 25,

hwnd, reinterpret\_cast<HMENU>(EDIT\_MESSAGE), hInstance, NULL);

hSendButton = CreateWindow(L"BUTTON", L"Send", WS\_VISIBLE | WS\_CHILD, 350, 400, 75, 25, hwnd, (HMENU)ID\_BUTTON\_SEND, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);

if ((clientSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET) {

MessageBox(hwnd, L"Failed to create socket.", L"Error", MB\_OK | MB\_ICONERROR);

WSACleanup();

return 1;

}

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(12345);

serverAddress.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (connect(clientSocket, (struct sockaddr\*)&serverAddress, sizeof(serverAddress)) == SOCKET\_ERROR) {

MessageBox(hwnd, L"Failed to connect to the server.", L"Error", MB\_OK | MB\_ICONERROR);

closesocket(clientSocket);

WSACleanup();

return 1;

}

SetWindowTextA(hShowMessage, "Connected to the server.");

std::thread receiveThread(ReceiveMessages, clientSocket);

//std::thread sendThread(SendMessages, clientSocket);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

receiveThread.join();

//sendThread.join();

closesocket(clientSocket);

WSACleanup();

return 0;

}