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 №8
на тему

ИНТЕРФЕЙС СОКЕТОВ И ОСНОВЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ (WINDOWS). ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ЧЕРЕЗ СЕТЬ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА СОКЕТОВ. РЕАЛИЗАЦИЯ СЕТЕВЫХ ПРОТОКОЛОВ: СОБСТВЕННЫХ ИЛИ СТАНДАРТНЫХ.

Студент М. А. Шкарубский
Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является проектирование и написание многопользовательского приложения для обмена текстовыми данными в локальной сети с использованием сокетов. Приложение позволяет как отправлять сообщения нескольким клиентам, так и просматривать полученные от них сообщения, предусматривает отключение и повторное соединение с сервером, а также смену установку имени пользователя.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Понятие протокола. Протоколы TCP/IP

Протоколом называется набор правил, задающих форматы сообщений и процедуры, которые позволяют компьютерам и прикладным программам обмениваться информацией. Эти правила соблюдаются каждым компьютером в сети, в результате чего любой хост-получатель может понять отправленное ему сообщение. Набор протоколов TCP/IP можно рассматривать как многоуровневую структуру. Такая структура для протоколов TCP/IP показана на рисунке 2.1.1.

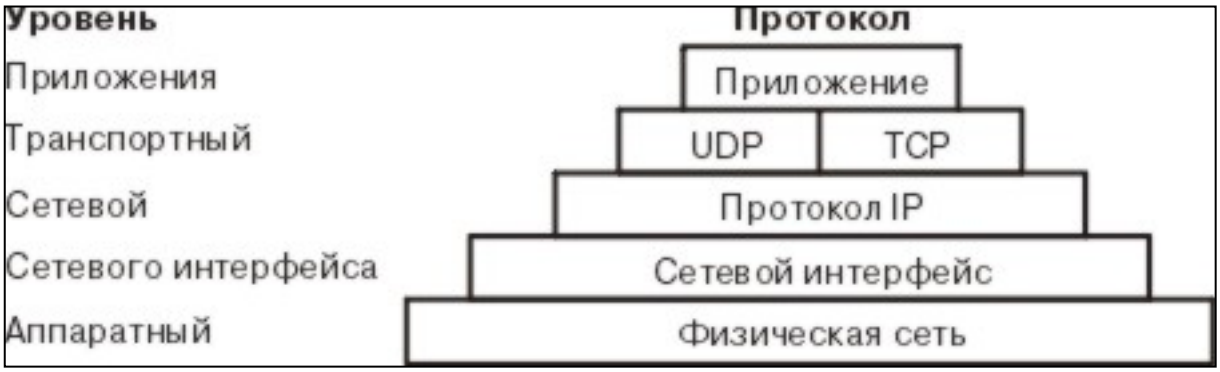

Рисунок 2.1.1 – Набор протоколов TCP/IP

В протоколе TCP/IP строго зафиксированы правила передачи информации от отправителя к получателю. Сообщение или поток данных приложения отправляется протоколу Internet транспортного уровня, то есть Протоколу пользовательских дейтаграмм (UDP) или Протоколу управления передачей (TCP). Получив данные от приложения, эти протоколы разделяют всю информацию на блоки, называемые пакетами. К каждому пакету добавляется адрес назначения, после чего пакет передается на следующий уровень протоколов Internet, то есть сетевой уровень.

На сетевом уровне пакет помещается в дейтаграмму протокола Internet (IP), к которой добавляется заголовок и концевик. Протокол сетевого уровня определяет адрес следующего пункта назначения IP-дейтаграммы и отправляют ее на уровень сетевого интерфейса [1].

2.2 Понятие сокета. Принципы сокетов

Для взаимодействия между машинами с помощью протоколов TCP/IP используются адреса и порты. Адрес представляет собой 32-битную или

128-битную структуру в зависимости от протокола (IPv4 и IPv6 соответственно). Номер порта — целое число в диапазоне от 0 до 65535 (в частности для протокола TCP).

Эта пара определяет сокет («гнездо», соответствующее адресу и порту).

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

Каждый процесс может создать «слушающий» сокет (серверный сокет) и привязать его к определенному порту операционной системы. 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 так далее.

2.3 Сокеты Windows 2

Windows Sockets 2 (Winsock) позволяет программистам создавать расширенные приложения Интернета, интрасети и других сетевых приложений для передачи данных приложений по сети независимо от используемого сетевого протокола. Благодаря Winsock программистам предоставляется доступ к расширенным сетевым возможностям Microsoft® Windows®, таким как многоадресная рассылка и качество обслуживания (QoS).

Winsock следует модели Windows Open System Architecture (WOSA); он определяет стандартный интерфейс поставщика услуг (SPI) между программным интерфейсом приложения (API) с экспортируемыми функциями и стеками протоколов [3]. Программирование Winsock ранее было сосредоточено на TCP/IP. Некоторые методики программирования, которые работали с TCP/IP, работают не с каждым протоколом. В результате API сокетов Windows 2 добавляет функции при необходимости для обработки нескольких протоколов.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

3.1 Общая характеристика и интерфейс приложения

В рамках выполнения лабораторной работы было разработано многопользовательское приложение-мессенджер для обмена текстовыми сообщениями между несколькими клиентами в локальной сети с возможностью повторного подключения к серверу и смены имени пользователя. На рисунке 3.1 изображено окно приложения.

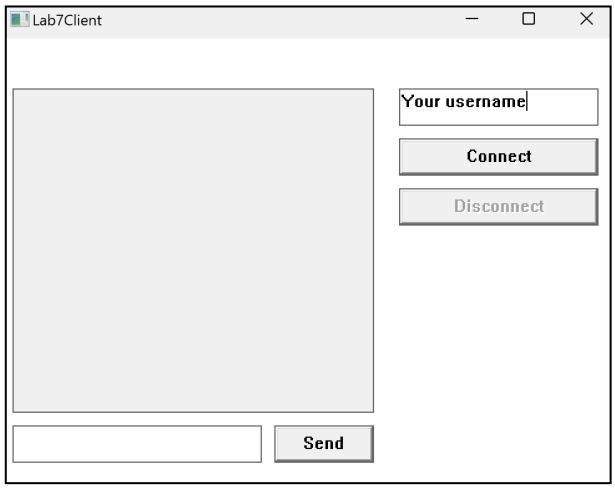

Рисунок 3.1.1 – Окно приложения

3.2 Функциональность приложения

Так для полноты функциональности приложения требуется наличие запущенного локального сервера, в приложении предусмотрен ряд необходимых проверок и защитных механизмов на случай потенциальных ошибок. Результаты работы некоторых проверок продемонстрированы на рисунке 3.2.1.

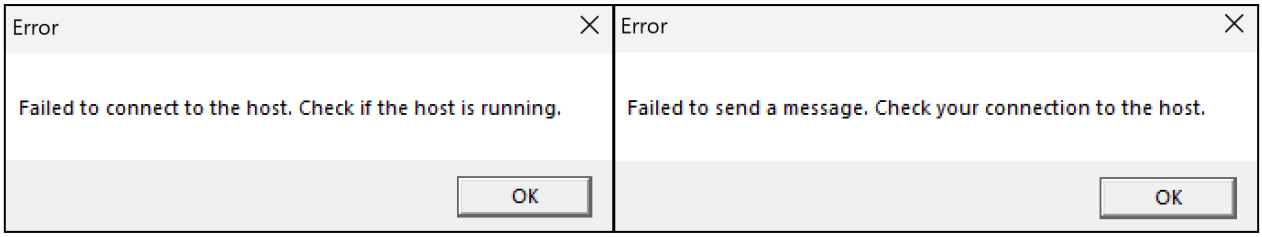

Рисунок 3.2.1 – Обработка ошибок при использованиии приложения

При подключении на сервер пользователю присваивается имя, выбранное им самим, или же соответствующее номеру сокета его клиента. Для смены имени требуется отключиться от сервера, изменить имя пользователя, и заново установить подключение, нажав на кнопку «Connect». Диалог между двумя пользователями представлен на рисунке 3.2.2.

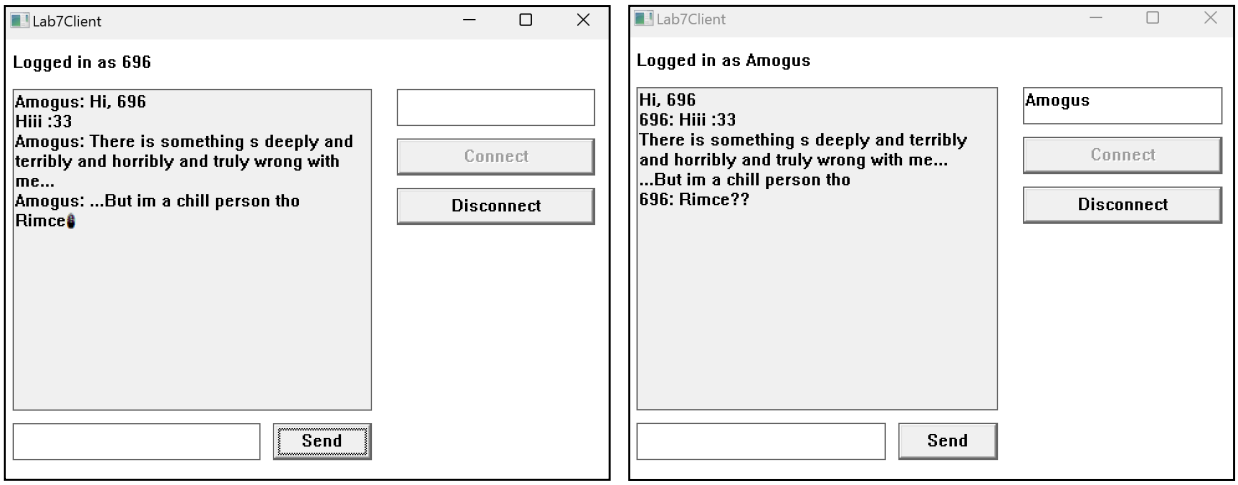

Рисунок 3.2.2 – Сценарий использования приложения

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были изучен и освоен интерфейс сокетов и основы сетевого программирования Windows с использованием Win32 C++ API. Помимо этого, для закрепления изученных навыков было разработано многопользовательское приложение, использующее вышеперечисленные технологии для обмена сообщениями в локальной сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Протоколы TCP/IP [Электронный ресурс]. – Режим доступа: https://www.ibm.com/docs/ru/aix/7.2?topic=protocol-tcpip-protocols.

[2] Сокеты [Электронный ресурс]. – Режим доступа: https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/intro.html.

[3] Сокеты Windows 2 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/winsock/windows-sockets-start-page-2.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл ServerNetwork.h

```
#pragma once

#include <WinSock2.h>

class ServerNetwork {
public:
    ~ServerNetwork();

    bool Initialize(int port);
    void StartListening();
    SOCKET GetServerSocket();

private:
    SOCKET serverSocket;
};
```

Листинг 2 – Файл ServerNetwork.cpp

```
#pragma comment(lib, "Ws2_32.lib")

#include "ServerNetwork.h"

ServerNetwork::~ServerNetwork() {
    closesocket(serverSocket);
    WSACleanup();
}

bool ServerNetwork::Initialize(int port) {
    WSAData wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
        return false;

    serverSocket = socket(AF_INET, SOCK_STREAM, 0);

    if (serverSocket == INVALID_SOCKET) {
        WSACleanup();
        return false;
    }

    SOCKADDR_IN serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(port);

    if (bind(serverSocket, (SOCKADDR*)&serverAddress, sizeof(serverAddress))
== SOCKET_ERROR) {
        closesocket(serverSocket);
        WSACleanup();
        return false;
```

```
    }

    return true;
}

void ServerNetwork::StartListening() {
    listen(serverSocket, SOMAXCONN);
}

SOCKET ServerNetwork::GetServerSocket() {
    return serverSocket;
}
```

Листинг 3 – Файл ServerThread.h

```
#pragma once

#include <WinSock2.h>
#include <vector>
#include <string>

class ServerThread {
public:
    ServerThread(SOCKET clientSocket);
    ServerThread(SOCKET clientSocket, std::vector<SOCKET>* connectedClients);
    ~ServerThread();

    SOCKET GetClientSocket();
    std::vector<SOCKET>* GetConnectedClients();
    void SetConnectedClients(std::vector<SOCKET>* connectedClients);
    bool IsRunning();
    void SetIsRunning(bool isRunning);
    bool IsFirstMessage();
    void SetIsFirstMessage(bool isFirstMessage);
    std::string GetClientUsername();
    void SetClientUsername(std::string newClientUsername);

    void BroadcastMessage(const char* message);

private:
    HANDLE threadHandle;
    SOCKET clientSocket;
    std::vector<SOCKET>* connectedClients;
    bool running;
    bool firstMessage;
    std::string clientUsername;
};
```

Листинг 4 – Файл ServerThread.cpp

```
#pragma comment(lib, "Ws2_32.lib")

#include "ServerThread.h"
#include <iostream>

DWORD WINAPI ClientThread(LPVOID param) {
ServerThread* serverThread = static_cast<ServerThread*>(param);
```

```
while (serverThread->IsRunning()) {
char buffer[1024];
int bytesRead = recv(serverThread->GetClientSocket(), buffer, 1024, 0);

if (bytesRead <= 0) {
serverThread->SetIsRunning(false);
closesocket(serverThread->GetClientSocket());

auto it = std::find(serverThread->GetConnectedClients()->begin(),
serverThread->GetConnectedClients()->end(), serverThread->GetClientSocket());
if (it != serverThread->GetConnectedClients()->end())
serverThread->GetConnectedClients()->erase(it);

break;
}

buffer[bytesRead] = '\0';
std::cout << "Message received by server" << std::endl;
std::cout << buffer << std::endl;

if (serverThread->IsFirstMessage()) {
serverThread->SetClientUsername(buffer);
serverThread->SetIsFirstMessage(false);

continue;
}

std::string prefixedMsg = serverThread->GetClientUsername() + ": " + buffer;

if (serverThread->GetConnectedClients()->size() > 0)
serverThread->BroadcastMessage(prefixedMsg.c_str());
}

return 0;
}

ServerThread::ServerThread(SOCKET clientSocket) : clientSocket(clientSocket),
running(true), firstMessage(true) {
    threadHandle = CreateThread(NULL, 0, ClientThread, this, 0, NULL);
}

ServerThread::ServerThread(SOCKET clientSocket, std::vector<SOCKET>*
connectedClients) : clientSocket(clientSocket), running(true),
firstMessage(true), connectedClients(connectedClients) {
    threadHandle = CreateThread(NULL, 0, ClientThread, this, 0, NULL);
}

ServerThread::~ServerThread() {
    CloseHandle(threadHandle);
}

SOCKET ServerThread::GetClientSocket() {
    return clientSocket;
}

std::vector<SOCKET>* ServerThread::GetConnectedClients()
{
    return connectedClients;
```

```
}

void ServerThread::SetConnectedClients(std::vector<SOCKET>* connectedClients)
{
    connectedClients = connectedClients;
}

bool ServerThread::IsRunning() {
    return running;
}

void ServerThread::SetIsRunning(bool isRunning) {
    running = isRunning;
}

bool ServerThread::IsFirstMessage() {
    return firstMessage;
}

void ServerThread::SetIsFirstMessage(bool isFirstMessage) {
    firstMessage = isFirstMessage;
}

std::string ServerThread::GetClientUsername()
{
    return clientUsername;
}

void ServerThread::SetClientUsername(std::string newClientUsername)
{
    clientUsername = newClientUsername;
}

void ServerThread::BroadcastMessage(const char* message)
{
    for (SOCKET client : *connectedClients)
        if (client != clientSocket) {
            send(client, message, strlen(message), 0);
            std::cout << "Broadcasted message to client " << client <<
std::endl;
        }

    std::cout << std::endl;
}
```

Листинг 5 – Файл Lab8Server.cpp

```
#pragma comment(lib, "Ws2_32.lib")

#include "ServerNetwork.h"
#include "ServerThread.h"
#include <windows.h>
#include <iostream>
#include <vector>

int main() {
    ServerNetwork server;
    std::vector<SOCKET>* connectedClients = new std::vector<SOCKET>;
```

```
    if (!server.Initialize(12345))
        return 1;

    server.StartListening();

    while (true) {
        SOCKET clientSocket = accept(server.GetServerSocket(), NULL, NULL);

        if (clientSocket != INVALID_SOCKET) {
            connectedClients->push_back(clientSocket);
            ServerThread* thread = new ServerThread(clientSocket,
connectedClients);
        }
        else {
            std::cout << "Invalid socket" << std::endl;
        }
    }

    return 0;
}
```

Листинг 6 – Файл ClientNetwork.h

```
#pragma once

#include <WinSock2.h>
#include <ws2tcpip.h>


class ClientNetwork {
public:
    ClientNetwork();
    ~ClientNetwork();

    bool Initialize(const char* serverIP, int serverPort);
    bool ConnectToServer();
    void DisconnectFromServer();
    bool Send(const char* message);
    bool Receive(char* buffer);
    bool IsConnected() const;

    SOCKET GetClientSocket();

private:
    SOCKET clientSocket;
    sockaddr_in serverAddress;
    bool connected;
};
```

Листинг 7 – Файл ClientNetwork.cpp

```
#pragma comment(lib, "Ws2_32.lib")

#include "ClientNetwork.h"

ClientNetwork::ClientNetwork() : clientSocket(INVALID_SOCKET),
connected(false) {}
```

```
ClientNetwork::~ClientNetwork() {
    DisconnectFromServer();
}

bool ClientNetwork::Initialize(const char* serverIP, int serverPort) {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
        return false;

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == INVALID_SOCKET) {
        WSACleanup();
        return false;
    }

    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(serverPort);
    inet_pton(AF_INET, serverIP, &(serverAddress.sin_addr));

    return true;
}

bool ClientNetwork::ConnectToServer() {
    if (connected)
        return true;

    int result = connect(clientSocket, (sockaddr*)&serverAddress,
sizeof(serverAddress));
    if (result == SOCKET_ERROR)
        return false;

    connected = true;

    return true;
}

void ClientNetwork::DisconnectFromServer() {
    if (connected) {
        closesocket(clientSocket);
        WSACleanup();
        connected = false;
    }
}

bool ClientNetwork::Send(const char* message) {
    if (!connected)
        return false;

    int result = send(clientSocket, message, strlen(message), 0);
    if (result == SOCKET_ERROR)
        return false;

    return true;
}

bool ClientNetwork::Receive(char* buffer) {
    if (!connected)
```

```
        return false;

    int bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);

    if (bytesRead <= 0)
        return false;

    buffer[bytesRead] = '\0';

    return true;
}

bool ClientNetwork::IsConnected() const {
    return connected;
}

SOCKET ClientNetwork::GetClientSocket()
{
    return clientSocket;
}
```

Листинг 8 – Файл Lab8Client.cpp

```
#pragma comment(lib, "Ws2_32.lib")


#include "Lab7Client.h"
#include "ClientNetwork.h"
#include <windows.h>
#include <string>

// Global variables for GUI controls
HWND hwndInput;     // Text input field
HWND hwndSendButton; // Send button
HWND hwndMessageLog; // Message display area
HWND hwndUsernameDisplay; // Username display area
HWND hwndUsernameInput;  // Username input field
HWND hwndConnectButton; // Connect to the server button
HWND hwndDisconnectButton; // Disconnect from the server button
ClientNetwork* clientNetwork;


DWORD WINAPI ClientReceiveThread(LPVOID param) {
    while (true) {
        char buffer[1024];
        wchar_t wBuffer[1024];

        size_t convertedChars = 0;
        int bytesRead = recv(clientNetwork->GetClientSocket(), buffer,
sizeof(buffer), 0);

        if (bytesRead > 0) {
            buffer[bytesRead] = '\0';
            mbstowcs_s(&convertedChars, wBuffer, strlen(buffer) + 1, buffer,
_TRUNCATE);
            wBuffer[bytesRead] = '\0';

            // Append the received message to the message log
```

```
            SendMessage(hwndMessageLog, EM_SETSEL, -1, -1);
            SendMessage(hwndMessageLog, EM_REPLACESEL, 0, (LPARAM)wBuffer);
            SendMessage(hwndMessageLog, EM_REPLACESEL, 0, (LPARAM)L"\r\n");
        }

        Sleep(1000);
    }

    return 0;
}


void OnSendButtonClick(HWND hWnd) {
    wchar_t wMessage[256];
    char message[256];

    GetWindowTextA(hwndInput, message, sizeof(message));
    GetWindowText(hwndInput, wMessage, sizeof(wMessage));

    // Send the message to the server using clientNetwork
    if (clientNetwork->Send(message)) {
        // Append the sent message to the message log
        SendMessage(hwndMessageLog, EM_SETSEL, -1, -1);
        SendMessage(hwndMessageLog, EM_REPLACESEL, 0, (LPARAM)wMessage);

        // Clear the input field
        SetWindowText(hwndInput, L"");
        SendMessage(hwndMessageLog, EM_REPLACESEL, 0, (LPARAM)L"\r\n");
    }
    else {
        MessageBox(hWnd, L"Failed to send a message. Check your connection to
the host.", L"Error", NULL);
    };
}

int OnConnectButtonClick(HWND hWnd) {
    if (!clientNetwork->Initialize("127.0.0.1", 12345)) {
        MessageBox(hWnd, L"Failed to initialize client network.", L"Error",
NULL);
        return 1;
    }

    if (!clientNetwork->ConnectToServer()) {
        MessageBox(hWnd, L"Failed to connect to the host. Check if the host
is running.", L"Error", NULL);
        return 1;
    }

    wchar_t wUsername[256];
    char username[256];

    GetWindowTextA(hwndUsernameInput, username, sizeof(username));
    GetWindowText(hwndUsernameInput, wUsername, sizeof(username));

    std::wstring usernameDisplay = L"Logged in as ";
    usernameDisplay += !(std::wstring(wUsername).empty()) ? wUsername :
std::to_wstring(clientNetwork->GetClientSocket());
```

```
    SetWindowText(hwndUsernameDisplay, usernameDisplay.c_str());

    clientNetwork->Send(std::string(username).empty() ?
std::to_string(clientNetwork->GetClientSocket()).c_str() : username);

    if (clientNetwork->IsConnected()) {
        EnableWindow(hwndConnectButton, FALSE);
        EnableWindow(hwndDisconnectButton, TRUE);
    }
    else {
        MessageBox(hWnd, L"Unable to connect to server.", L"Error", NULL);
    }

    return 0;
}

void OnDisconnectButtonClick(HWND hWnd) {
    clientNetwork->DisconnectFromServer();

    if (!clientNetwork->IsConnected()) {
        EnableWindow(hwndConnectButton, TRUE);
        EnableWindow(hwndDisconnectButton, FALSE);
    }
    else {
        MessageBox(hWnd, L"Unable to disconnect from server.", L"Error",
NULL);
    }
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam) {
    switch (message) {
        case WM_COMMAND:
        {
            if (lParam == (LPARAM)hwndSendButton && HIWORD(wParam) ==
BN_CLICKED)
                OnSendButtonClick(hWnd);

            if (lParam == (LPARAM)hwndDisconnectButton && HIWORD(wParam) ==
BN_CLICKED)
                OnDisconnectButtonClick(hWnd);

            if (lParam == (LPARAM)hwndConnectButton && HIWORD(wParam) ==
BN_CLICKED) {
                int result = OnConnectButtonClick(hWnd);
                if (result == 1)
                    return 1;

                CreateThread(NULL, 0, ClientReceiveThread, hWnd, 0, NULL);
            }
        }

            break;

        case WM_DESTROY:
            clientNetwork->DisconnectFromServer();

        default:
```

```
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) {
    clientNetwork = new ClientNetwork();

    WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };
    wcex.lpfnWndProc = WndProc;
    wcex.hInstance = hInstance;
    wcex.lpszClassName = L"Lab7ClientWindowClass";
    RegisterClassEx(&wcex);

    HWND hWnd = CreateWindow(
        L"Lab7ClientWindowClass",
        L"Lab7Client",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        510, 400, NULL, NULL,
        hInstance, NULL
    );

    hwndUsernameDisplay = CreateWindow(L"EDIT", L"", WS_CHILD | WS_VISIBLE,
10, 10, 290, 20, hWnd, NULL, NULL, NULL);
    hwndUsernameInput = CreateWindow(L"EDIT", L"", WS_CHILD | WS_VISIBLE |
WS_BORDER, 320, 40, 160, 30, hWnd, NULL, NULL, NULL);
    hwndConnectButton = CreateWindow(L"BUTTON", L"Connect", WS_CHILD |
WS_VISIBLE | BS_DEFPUSHBUTTON, 320, 80, 160, 30, hWnd, NULL, NULL, NULL);
    hwndDisconnectButton = CreateWindow(L"BUTTON", L"Disconnect", WS_CHILD |
WS_VISIBLE | BS_DEFPUSHBUTTON, 320, 120, 160, 30, hWnd, NULL, NULL, NULL);
    hwndInput = CreateWindow(L"EDIT", L"", WS_CHILD | WS_VISIBLE | WS_BORDER
| ES_AUTOHSCROLL, 10, 310, 200, 30, hWnd, NULL, NULL, NULL);
    hwndSendButton = CreateWindow(L"BUTTON", L"Send", WS_CHILD | WS_VISIBLE |
BS_DEFPUSHBUTTON, 220, 310, 80, 30, hWnd, NULL, NULL, NULL);
    hwndMessageLog = CreateWindow(L"EDIT", L"", WS_CHILD | WS_VISIBLE |
WS_BORDER | ES_AUTOVSCROLL | ES_MULTILINE | ES_READONLY, 10, 40, 290, 260,
hWnd, NULL, NULL, NULL);

    EnableWindow(hwndDisconnectButton, FALSE);

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```