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 Д. С. Кончик

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цель данной лабораторной работы заключается в следующем:

1. Изучение теоретических основ сетевого программирования.
2. Изучение стандартных сетевых протоколов и их особенностей. Понимание принципов их работы.
3. Ознакомление с основными функциями и структурами, предоставляемыми *Windows* для работы с сокетами. Освоение интерфейса сокетов в операционной системе *Windows*:
4. Получение практических навыков путем разработки клиент-серверного приложения для обмена текстовыми сообщениями с использованием TCP сокетов.
5. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сетевое программирование – это область программирования, связанная с разработкой приложений, способных обмениваться данными через компьютерные сети. Один из ключевых инструментов для реализации сетевого взаимодействия – это сокеты.

Сокет (*socket*) – это конечная точка для обмена данными между процессами в компьютерной сети. Сокеты предоставляют абстракцию для взаимодействия между процессами, работающими на разных узлах сети, используя протоколы транспортного уровня, такие как *TCP* или *UDP)* [1]*.*

*TCP* и *UDP* – это два основных протокола транспортного уровня, используемых в сетевом программировании.

*TCP (Transmission Control Protocol)* – обеспечивает устойчивое, упорядоченное и надежное соединение между двумя точками в сети. Применяется в ситуациях, где важна доставка данных без потерь и в правильном порядке.

*UDP (User Datagram Protocol) -* обеспечивает простую и неупорядоченную передачу данных. Применяется в сценариях, где допускаются потери данных, но критически важна минимальная задержка [2].

*Windows Sockets API (Winsock)* – это стандартный интерфейс для разработки сетевых приложений под операционной системой Windows. Он предоставляет функции и структуры данных, необходимые для работы с сокетами.

Инициализация *Winsock* выполняется с использованием функции *WSAStartup*, а завершение - с использованием функции *WSACleanup*. Эти функции обеспечивают правильную инициализацию и очистку ресурсов, связанных с работой сокетов.

Сокет создается с использованием функции *socket.* После создания сокета, необходимо задать параметры, такие как адрес и порт, с которыми он будет ассоциирован. Эти параметры настраиваются с использованием структуры *sockaddr\_in*.

Для установки соединения с удаленным сервером используется функция *connect.* Она требуется для клиентских приложений и выполняет установку соединения с сервером по указанному адресу и порту.

Функции *send* и *recv* используются для отправки и приема данных через сокеты. Эти функции позволяют передавать байты данных между клиентом и сервером [3].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы было создано два приложения: сервер и клиент.

Реализация серверного приложения (рисунок 1):

1. Создан серверный сокет, привязанный к определенному порту, ожидающий подключений от клиентов.
2. Сокеты клиентов сохраняются для последующей рассылки сообщений. Также сообщения всех клиентов выводятся в консоль.
3. Реализован многопоточный механизм обработки сообщений клиентов.

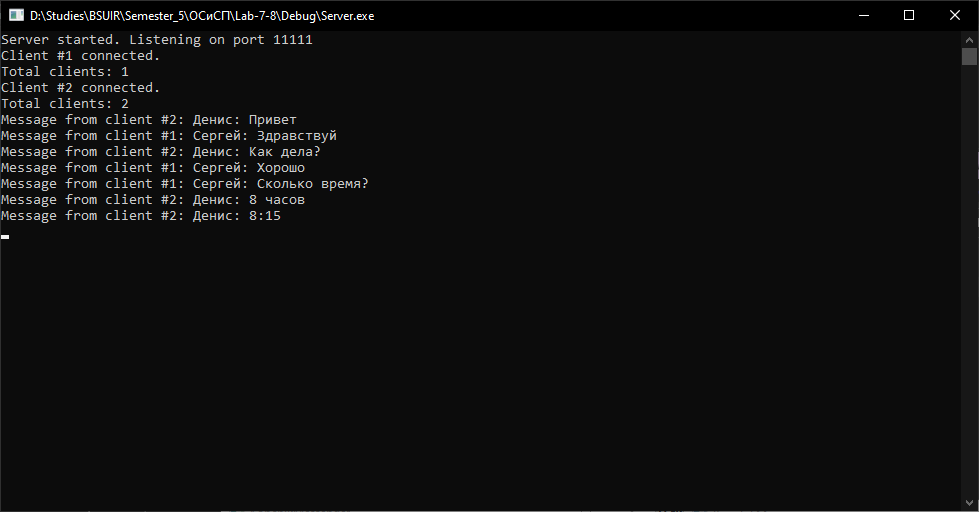


Рисунок 1 – Окно сервера

Реализация клиентского приложения (рисунок 2):

1. Создано графическое окно с элементами управления для ввода имени и сообщения, отправки сообщения, и окна для отображения сообщений.
2. Выполнено подключение к серверу по указанному *IP*-адресу и порту.
3. Реализована отправка сообщений на сервер.
4. Реализован многопоточный механизм для асинхронного приема сообщений от сервера.

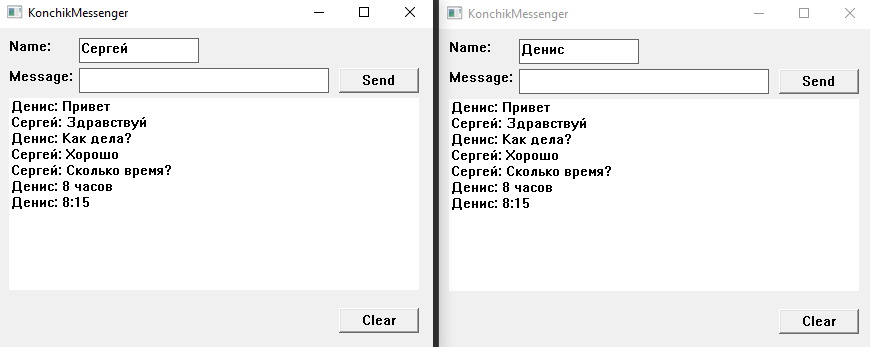


Рисунок 2 – Два запущенных клиента

# ЗАКЛЮЧЕНИЕ

Лабораторная работа была выполнена с целью изучения теоретических основ сетевого программирования, стандартных сетевых протоколов (*TCP* и *UDP*), а также основных функций и структур, предоставляемых *Windows* для работы с сокетами. В результате работы были разработаны клиентское и серверное приложения, обеспечивающие взаимодействие через сетевые сокеты. Лабораторная работа успешно достигла поставленных целей, предоставив практические навыки создания, настройки и использования сокетов в среде *Windows*. Полученные знания служат основой для дальнейшего изучения и разработки сетевых приложений, а также обеспечивают понимание принципов работы сетевых протоколов и сокетов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование сетевых задач [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Программирование_сетевых_задач>.
2. Протоколы TCP и UDP [Электронный ресурс]. – Режим доступа: https://professorweb.ru/my/csharp/web/level1/1\_4.php.
3. Использование Winsock [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/winsock/using-winsock>.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Серверное приложение

#include <WinSock2.h>

#include <iostream>

#include <vector>

#include <thread>

#include <string>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

#define DEFAULT\_PORT 11111

vector<SOCKET> clients;

int getUniqueNumber() {

static int number = 1;

return number++;

}

// Обработка сообщений от клиента

void HandleClient(const SOCKET& clientSocket) {

int clientNumber = getUniqueNumber();

cout << "Client #" << clientNumber << " connected." << endl;

cout << "Total clients: " << clients.size() << endl;

while (true) {

char buffer[1000];

int bytesRead = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);

if (bytesRead <= 0) {

cout << "Client #" << clientNumber << " disconnected.\n";

// Удалить сокет из вектора

auto it = find(clients.begin(), clients.end(), clientSocket);

if (it != clients.end())

clients.erase(it);

// Закрыть сокет

closesocket(clientSocket);

return;

}

buffer[bytesRead] = '\0';

cout << "Message from client #" << clientNumber << ": " << buffer << endl;

// Рассылка сообщения всем клиентам

for (size\_t i = 0; i < clients.size(); i++) {

const SOCKET& otherClient = clients[i];

int bytesSent = send(otherClient, buffer, bytesRead, 0);

if (bytesSent == SOCKET\_ERROR)

cout << "Error sending message to client #" << i << ".\n";

}

}

}

int main() {

setlocale(LC\_ALL, "Ru");

WSADATA wsaData;

// Инициализация библиотеки Winsock

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

cout << "Failed to initialize Winsock." << endl;

return 1;

}

// Создание сокета

SOCKET listenSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (listenSocket == INVALID\_SOCKET) {

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

WSACleanup();

return 1;

}

// Настройка параметров сервера

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(DEFAULT\_PORT);

// Привязка сокета

if (bind(listenSocket, reinterpret\_cast<sockaddr\*>(&serverAddr), sizeof(serverAddr)) == SOCKET\_ERROR) {

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

closesocket(listenSocket);

WSACleanup();

return 1;

}

// Ожидание подключений

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

cout << "Error while listening for connections." << endl;

closesocket(listenSocket);

WSACleanup();

return 1;

}

cout << "Server started. Listening on port " << DEFAULT\_PORT << endl;

while (true) {

// Принятие подключения

SOCKET clientSocket = accept(listenSocket, NULL, NULL);

if (clientSocket == INVALID\_SOCKET) {

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

continue;

}

// Добавление клиента в вектор

clients.push\_back(clientSocket);

// Создание нового потока для обработки клиента

thread(HandleClient, clientSocket).detach();

}

// Закрытие сокета и очистка Winsock

closesocket(listenSocket);

WSACleanup();

return 0;

}

Листинг 2 – Клиентское приложение

#include <Windows.h>

#include <string>

#include <thread>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

#define DEFAULT\_PORT 11111

#define ID\_BUTTON\_SEND 1

#define ID\_BUTTON\_CLEAR 2

HWND hNameEdit, hMessageEdit, hListBox;

SOCKET clientSocket;

void HandleMessages() {

while (true) {

char buffer[1000];

int bytesRead = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);

if (bytesRead <= 0) {

MessageBox(NULL, L"Error reading messages or server disconnected.", L"Error", MB\_OK | MB\_ICONERROR);

closesocket(clientSocket);

exit(1);

}

buffer[bytesRead] = '\0';

SendMessageA(hListBox, LB\_ADDSTRING, 0, reinterpret\_cast<LPARAM>(buffer));

}

}

void ConnectToServer(const char\* serverIP) {

WSADATA wsaData;

// Инициализация библиотеки Winsock

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

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

exit(1);

}

// Создание сокета

clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket == INVALID\_SOCKET) {

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

WSACleanup();

exit(1);

}

// Настройка параметров сервера

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr(serverIP); // Преобразование IP-адреса из строки

serverAddr.sin\_port = htons(DEFAULT\_PORT);

if (serverAddr.sin\_addr.s\_addr == INADDR\_NONE) {

MessageBox(NULL, L"Invalid server IP address.", L"Error", MB\_OK | MB\_ICONERROR);

closesocket(clientSocket);

WSACleanup();

exit(1);

}

// Подключение к серверу

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

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

closesocket(clientSocket);

WSACleanup();

exit(1);

}

// Запуск потока для приема сообщений от сервера

thread(HandleMessages).detach();

}

void SendButtonClicked() {

char name[100], message[500];

GetWindowTextA(hNameEdit, name, sizeof(name) / sizeof(name[0]));

GetWindowTextA(hMessageEdit, message, sizeof(message) / sizeof(message[0]));

string fullMessage = name;

fullMessage += ": ";

fullMessage += message;

// Отправка сообщения на сервер

if (send(clientSocket, fullMessage.c\_str(), fullMessage.size(), 0) == SOCKET\_ERROR)

MessageBox(NULL, L"Failed to send message to the server.", L"Error", MB\_OK | MB\_ICONERROR);

}

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

switch (uMsg) {

case WM\_CREATE:

CreateWindowEx(

0, L"STATIC", L"Name:",

WS\_CHILD | WS\_VISIBLE,

10, 10, 70, 25,

hwnd, NULL, NULL, NULL

);

hNameEdit = CreateWindowEx(

0, L"EDIT", NULL,

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL,

80, 10, 120, 25,

hwnd, NULL, NULL, NULL

);

SendMessage(hNameEdit, EM\_LIMITTEXT, 10, 0);

CreateWindowEx(

0, L"STATIC", L"Message:",

WS\_CHILD | WS\_VISIBLE,

10, 40, 70, 25,

hwnd, NULL, NULL, NULL

);

hMessageEdit = CreateWindowEx(

0, L"EDIT", NULL,

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL,

80, 40, 250, 25,

hwnd, NULL, NULL, NULL

);

SendMessage(hMessageEdit, EM\_LIMITTEXT, 40, 0);

CreateWindowEx(

0, L"BUTTON", L"Send",

WS\_CHILD | WS\_VISIBLE,

340, 40, 80, 25,

hwnd, reinterpret\_cast<HMENU>(ID\_BUTTON\_SEND), NULL, NULL);

hListBox = CreateWindowEx(

0, L"LISTBOX", NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL,

10, 70, 410, 205,

hwnd, NULL, NULL, NULL

);

CreateWindowEx(

0, L"BUTTON", L"Clear",

WS\_CHILD | WS\_VISIBLE,

340, 280, 80, 25,

hwnd, reinterpret\_cast<HMENU>(ID\_BUTTON\_CLEAR), NULL, NULL

);

// Подключение к серверу

ConnectToServer("127.0.0.1");

break;

case WM\_COMMAND:

if (LOWORD(wParam) == ID\_BUTTON\_SEND && HIWORD(wParam) == BN\_CLICKED) {

SendButtonClicked();

SetWindowTextA(hMessageEdit, "");

}

else if (LOWORD(wParam) == ID\_BUTTON\_CLEAR && HIWORD(wParam) == BN\_CLICKED) {

SendMessage(hListBox, LB\_RESETCONTENT, 0, 0);

}

break;

case WM\_DESTROY:

// Завершение работы с сокетом и Winsock

closesocket(clientSocket);

WSACleanup();

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR cmdline, int ss) {

// Создание класса главного окна

WNDCLASS wc;

wc.style = NULL;

wc.lpfnWndProc = WindowProc;

wc.cbClsExtra = NULL;

wc.cbWndExtra = NULL;

wc.hInstance = hInst;

wc.hIcon = NULL;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW);

wc.lpszMenuName = NULL;

wc.lpszClassName = L"MainWindowClass";

if (!RegisterClass(&wc))

return EXIT\_FAILURE;

// Создание главного окна

HWND hWnd = CreateWindow(

L"MainWindowClass", L"KonchikMessenger",

WS\_OVERLAPPEDWINDOW,

0, 0, 450, 360,

NULL, NULL, hInst, NULL

);

if (!hWnd)

return EXIT\_FAILURE;

ShowWindow(hWnd, ss);

// Цикл обработки сообщений

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}