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 Е. А. Сироткин

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 7](#_Toc146752071)

[Список использованных источников 8](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 9](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цель данной лабораторной работы заключается в изучении технологий средств обмена данными и сетевых протоколов, приобретении практических навыков разработки программ для обмена данными по сети, а также в закреплении полученных знаний в результате выполнения лабораторной работы.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сетевое программирование – это область программирования, связанная с разработкой приложений, способных взаимодействовать между собой через компьютерные сети. Эта область включает в себя создание программ, которые могут обмениваться данными и ресурсами с удаленными системами через различные сетевые протоколы.

*UDP protocol* – протокол, обеспечивающий передачу данных (датаграмм) без предварительного создания соединения между хостами. При отправке датаграмм нет уверенности в существовании получателя и его готовности к обмену. Сетевой протокол *UDP* не обеспечивает также упорядочивание датаграмм при получении. Он используется приложениями, для которых существенное значение имеет время доставки. Датаграммы могут доставляться не в заданном порядке, дублироваться или вовсе не доставляться.

Протокол передачи данных *TCP –* протокол, обеспечивающий надежную доставку пакетов данных, он обеспечивает установку соединения между двумя хостами методом «рукопожатия», после которого может осуществляться обмен данными.

Перед началом передачи пакетов через *TCP* соединение устанавливается сессия с получателем, в рамках которой затем производится передача данных. Это позволяет убедиться в том, что получатель существует и готов принимать данные. После завершения передачи сессия закрывается, получатель извещается о том, что данных больше не будет, а отправитель извещается о том, что получатель извещён [1].

Локальная сеть (*Local Area Network*) представляет собой сеть компьютеров, устройств и ресурсов, ограниченную определенной географической областью, такой как офис, здание или кампус. Цель локальной сети – обеспечить обмен данными и ресурсами между подключенными устройствами.

Основные типы локальных сетей: проводные и беспроводные. Они различаются, например, по способу взаимодействия устройств в сетевых структурах или по скорости реагирования.

Беспроводная локальная сеть (*WLAN*) – не требует кабелей или других физических соединений для работы устройств. В этом случае чаще всего используется *Wi-Fi*.

Проводные локальные сети – для того чтобы устройства в них работали, они должны быть соединены проводом, обычно в виде кабеля или оптоволоконного кабеля [2].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения данной лабораторной работы были разработаны клиентское и серверное приложения с использованием сокетов, которые позволяют пользователям обмениваться сообщениями в анонимной форме (рисунок 1).

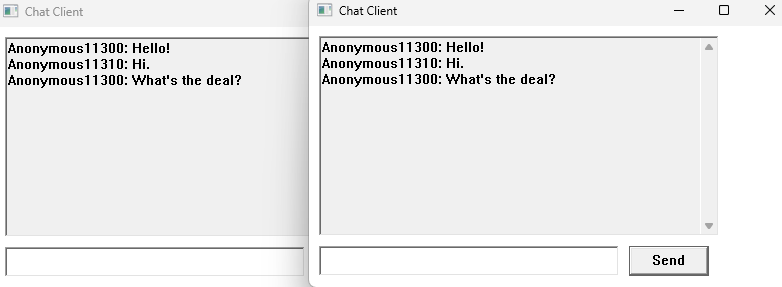


Рисунок 1– Окна клиентов

Был создан серверный сокет, привязанный к определенному порту. После получения сообщений от пользователей, сервер использует сохраненные сокеты клиентов для последующего их отправления. Вся полученная информация в виде сообщений клиентов отображается в консоли (рисунок 2).

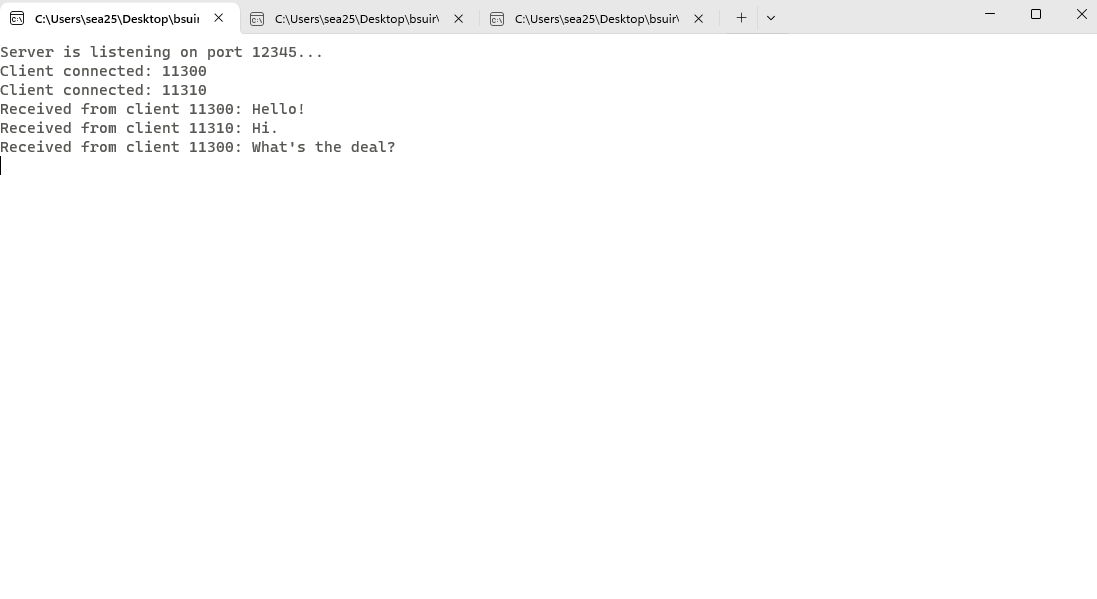


Рисунок 2 – Окно сервера

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были освоены аспекты работы с сетевыми протоколами и технологиями обмена данных. В рамках работы над приложением были реализованы возможности передачи сообщений между клиентами через сервер, а также получение информации о подключенных клиентах.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Протоколы TCP и UDP: в чем разница [Электронный ресурс]. – Режим доступа: https://whoer.net/blog/ru/protokoly-tcp-i-udp/.
2. Все о локальной компьютерной сети [Электронный ресурс]. – Режим доступа: https://vashtvmir.ru/lokalnaya-set-chto-eto-vse-o-lokalnoj-kompjuternoj-seti/.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Server.cpp*

#include <Winsock2.h>

#include <iostream>

#include <thread>

#include <vector>

#include <string>

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

constexpr int DEFAULT\_PORT = 12345;

struct ClientInfo {

SOCKET socket;

std::string clientId;

};

std::vector<ClientInfo> clients;

std::string GenerateClientId() {

auto now = std::time(nullptr);

// Используем текущее время для создания уникального идентификатора

std::srand(static\_cast<unsigned>(now));

int randomNum = std::rand();

return std::to\_string(randomNum);

}

void HandleClient(ClientInfo clientInfo) {

char buffer[2056];

int bytesReceived;

while (true) {

bytesReceived = recv(clientInfo.socket, buffer, sizeof(buffer) - 1, 0);

if (bytesReceived == SOCKET\_ERROR || bytesReceived == 0) {

std::cerr << "Client " << clientInfo.clientId << " disconnected" << std::endl;

closesocket(clientInfo.socket);

return;

}

buffer[bytesReceived] = '\0'; // Добавим нулевой символ для корректного вывода строки

std::cout << "Received from client " << clientInfo.clientId << ": " << buffer << std::endl;

for (const auto& client : clients) {

std::string messageWithId = clientInfo.clientId + ": " + buffer;

int bytesSent = send(client.socket, messageWithId.c\_str(), messageWithId.size(), 0);

if (bytesSent == SOCKET\_ERROR)

std::cout << "Error sending message to client" << ".\n";

}

}

}

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock" << std::endl;

return 1;

}

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

if (serverSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create server socket" << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

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

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

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

std::cerr << "Failed to bind server socket" << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

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

std::cerr << "Error in listen function" << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "Server is listening on port " << DEFAULT\_PORT << "..." << std::endl;

std::vector<std::thread> clientThreads;

while (true) {

SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Failed to accept client connection" << std::endl;

continue;

}

std::string clientId = GenerateClientId();

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

clients.push\_back({ clientSocket, clientId});

std::thread(HandleClient, clients.back()).detach();

}

// Дожидаемся завершения всех потоков

for (auto& thread : clientThreads) {

thread.join();

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

Листинг 2 – Файл *Client.cpp*

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <iostream>

#include <string>

#include <Windows.h>

#include <thread>

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

constexpr int DEFAULT\_PORT = 12345;

HWND hwndOutput, hwndInput;

SOCKET clientSocket;

LRESULT CALLBACK KeyboardHook(int nCode, WPARAM wParam, LPARAM lParam);

HHOOK hHook = NULL;

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

void AddMessageToOutput(const std::string& message) {

int len = GetWindowTextLengthA(hwndOutput);

SendMessageA(hwndOutput, EM\_SETSEL, len, len);

SendMessageA(hwndOutput, EM\_REPLACESEL, FALSE, reinterpret\_cast<LPARAM>(message.c\_str()));

}

void PostMessageToOutput(const std::string& message) {

PostMessage(hwndOutput, EM\_REPLACESEL, FALSE, reinterpret\_cast<LPARAM>(message.c\_str()));

}

void SendMessageToServer() {

char buffer[1024];

GetWindowTextA(hwndInput, buffer, sizeof(buffer));

if (send(clientSocket, buffer, static\_cast<int>(strlen(buffer)), 0) == SOCKET\_ERROR) {

std::cerr << "Failed to send message to the server" << std::endl;

}

SetWindowTextA(hwndInput, "");

}

void ReceiveMessagesFromServer() {

char buffer[1024];

int bytesReceived;

while (true) {

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

if (bytesReceived == SOCKET\_ERROR || bytesReceived == 0) {

std::cerr << "Connection closed by the server" << std::endl;

break;

}

buffer[bytesReceived] = '\0'; // Добавим нулевой символ для корректного вывода строки

std::size\_t pos = std::string(buffer).find(":");

if (pos != std::string::npos) {

std::string clientId = std::string(buffer).substr(0, pos);

std::string message = "Anonymous" + clientId + ": " + std::string(buffer).substr(pos + 2) + "\n"; // +2 для пропуска ": "

// Обновляем окно вывода сообщений

AddMessageToOutput(message);

}

}

}

int main() {

// Инициализация Winsock

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock" << std::endl;

return 1;

}

// Создание сокета

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

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create client socket" << std::endl;

WSACleanup();

return 1;

}

std::thread receiveThread(ReceiveMessagesFromServer);

// Установка адреса сервера

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

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

if (inet\_pton(AF\_INET, "127.0.0.1", &serverAddr.sin\_addr) != 1) {

std::cerr << "Invalid IP address" << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

// Подключение к серверу

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

std::cerr << "Failed to connect to the server" << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

// Создание окна

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = GetModuleHandle(NULL);

wc.lpszClassName = L"ChatClient";

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0,

L"ChatClient",

L"Chat Client",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 500, 300,

NULL,

NULL,

GetModuleHandle(NULL),

NULL

);

if (hwnd == NULL) {

std::cerr << "Failed to create window" << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

hwndOutput = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

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

10, 10, 400, 200,

hwnd,

NULL,

GetModuleHandle(NULL),

NULL

);

hwndInput = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

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

10, 220, 300, 30,

hwnd,

NULL,

GetModuleHandle(NULL),

NULL

);

CreateWindow(

L"BUTTON",

L"Send",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

320, 220, 80, 30,

hwnd,

(HMENU)1,

GetModuleHandle(NULL),

NULL

);

ShowWindow(hwnd, SW\_SHOW);

UpdateWindow(hwnd);

// Основной цикл обработки сообщений

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

receiveThread.join();

closesocket(clientSocket);

WSACleanup();

return 0;

}

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

switch (uMsg) {

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

case WM\_CREATE:

hHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardHook, GetModuleHandle(NULL), 0);

break;

case WM\_COMMAND:

if (LOWORD(wParam) == 1) {

OutputDebugString(L"ctuvybhunjk key pressed\n");

SendMessageToServer();

}

break;

case WM\_KEYDOWN:

OutputDebugString(L"Enter key pressed\n");

if (wParam == VK\_RETURN) {

SendMessageToServer();

}

break;

}

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

}

LRESULT CALLBACK KeyboardHook(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode >= 0) {

if (wParam == WM\_KEYDOWN) {

KBDLLHOOKSTRUCT\* keyInfo = (KBDLLHOOKSTRUCT\*)lParam;

if (keyInfo->vkCode == VK\_RETURN) {

// Simulate a button click when Enter is pressed

SendMessageToServer();

}

}

}

return CallNextHookEx(hHook, nCode, wParam, lParam);

}