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 М. А. Шкарубский
Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной 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.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Сведения о серверах и клиентах

Архитектура «Клиент-Сервер» (также используются термины «сеть Клиент-Сервер» или «модель Клиент-Сервер») предусматривает разделение процессов предоставление услуг (серверная часть) и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других (клиентская часть) [1]. Обычно процессы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине.

Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

2.2 Понятие сокета. Принципы сокетов

Сокеты – название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет в данном контексте суть абстрактный объект, представляющий конечную точку соединения [1].

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 (например, как на рисунке 2.1.1) – сокет отправителя и сокет получателя. 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.

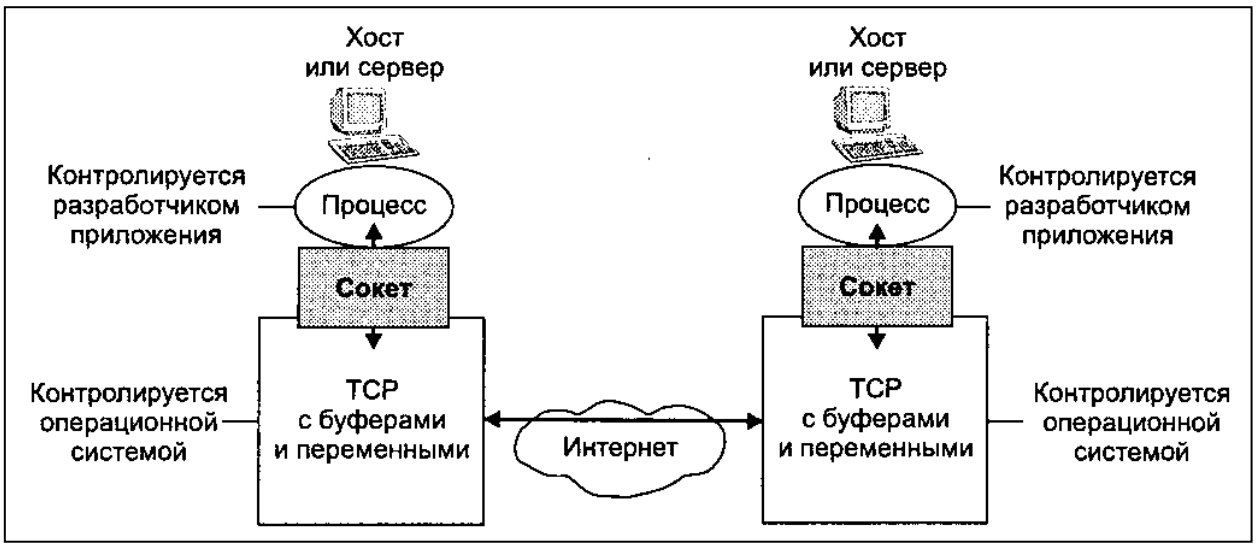

Рисунок 2.1.1 – Взаимодействие процессов при помощи TCP-сокетов

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.

2.2 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 [2]. 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

В рамках выполнения лабораторной работы было разработано многопользовательское приложение-мессенджер для обмена текстовыми сообщениями между несколькими клиентами в локальной сети. На рисунке 3.1 изображено окно такого приложения.

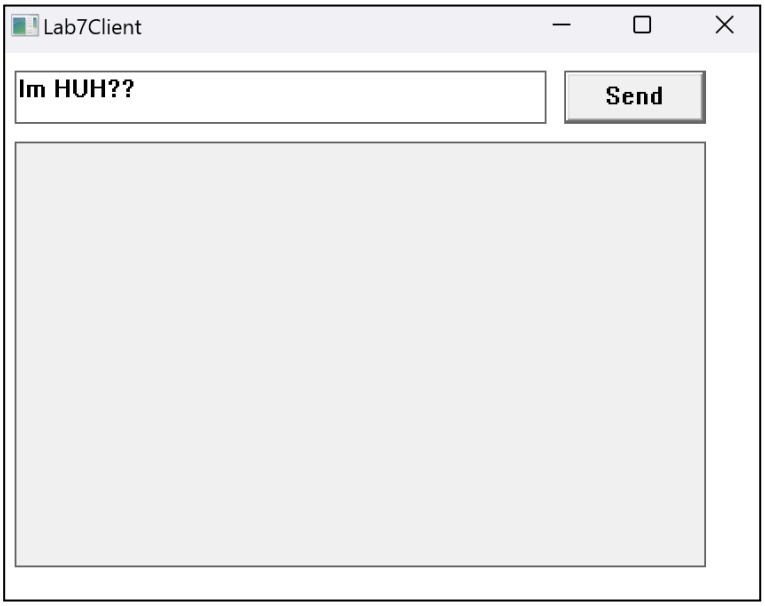

Рисунок 3.1 – Окно приложения

В процессе использования приложения каждый клиент может отправлять сообщения, которые будут отображаться в окнах других клиентов, подключенных к одному и тому же локальному серверу. Например, рисунок 3.2 иллюстрирует сценарий использования, где три клиента общаются между собой о поп-культуре.

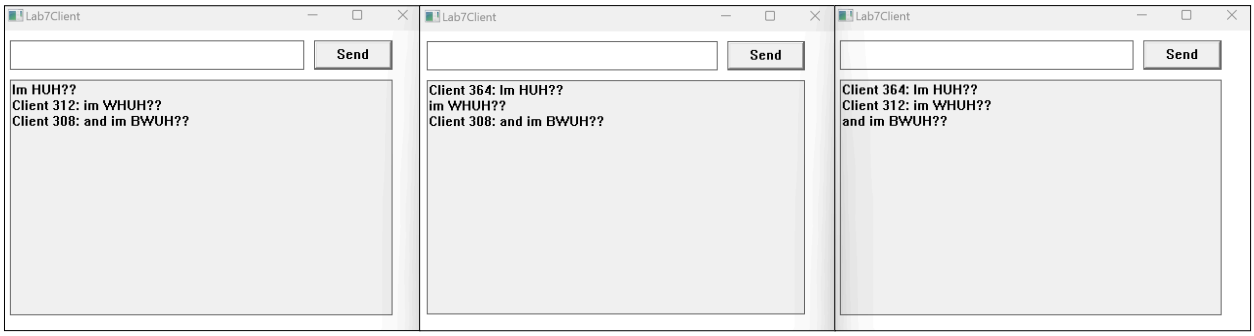

Рисунок 3.2 – Сценарий использования приложения

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были изучены и освоены основы сетевого программирования с использованием Win32 C++ API, сокетов и библиотеки для их реализации – WinSock2. Помимо этого, для закрепления изученных навыков было разработано многопользовательское приложение, использующее вышеперечисленные технологии для обмена сообщениями в локальной сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Архитектура «Клиент-Сервер» [Электронный ресурс]. – Режим доступа: https://itelon.ru/blog/arkhitektura-klient-server/.

[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();

bool IsRunning();
void SetIsRunning(bool isRunning);
SOCKET GetClientSocket();
void BroadcastMessage(const char* message);

std::vector<SOCKET>* connectedClients;

private:
SOCKET clientSocket;
HANDLE threadHandle;
bool running;
};
```

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()) {
        std::string clientName = "Client " + std::to_string(serverThread-
>GetClientSocket());

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

        if (bytesRead <= 0)
```

```
            break;

        buffer[bytesRead] = '\0';

        std::string prefixedMsg = clientName + ": " + buffer;

        std::cout << "Message received by server" << std::endl;
        std::cout << buffer << std::endl;

        serverThread->BroadcastMessage(prefixedMsg.c_str());
    }

    return 0;
}

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

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

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

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

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

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 Lab7Server.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;

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

    server.StartListening();

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

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

        connectedClients.push_back(clientSocket);
    }

    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, int bufferSize);
    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, int bufferSize) {
```

```
    if (!connected)
        return false;

    int bytesRead = recv(clientSocket, buffer, bufferSize - 1, 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 Lab7Client.cpp

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


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


// Global variables for GUI controls
HWND hwndInput;     // Text input field
HWND hwndSendButton; // Send button
HWND hwndMessageLog; // Message display area
ClientNetwork clientNetwork;


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

        size_t convertedChars = 0;
        size_t 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, ClientNetwork& clientNetwork) {
    wchar_t wMessage[256];
    char message[256];

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

    // 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);
        SendMessage(hwndMessageLog, EM_REPLACESEL, 0, (LPARAM)L"\r\n");

        // Clear the input field
        SetWindowText(hwndInput, L"");
    }
    else {
        MessageBox(hWnd, L"FAILED", L"FAILED", NULL);
    };
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam) {
    switch (message) {
        case WM_CREATE:
            // Start the message reception thread when the window is created
            CreateThread(NULL, 0, ClientReceiveThread, hWnd, 0, NULL);
            break;

        case WM_COMMAND:
            if (lParam == (LPARAM)hwndSendButton && HIWORD(wParam) ==
BN_CLICKED)
                OnSendButtonClick(hWnd, clientNetwork);

            break;

            // Handle other window messages here

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

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) {
    // Main application logic and clientNetwork initialization
    if (!clientNetwork.Initialize("127.0.0.1", 12345)) {
        // Handle initialization error
        return 1;
```

```
    }

    if (!clientNetwork.ConnectToServer()) {
        // Handle connection error
        return 1;
    }

    // Create the main window
    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, 0, 450, 350, NULL, NULL, hInstance,
NULL);

    // Create GUI controls
    hwndInput = CreateWindow(L"EDIT", L"", WS_CHILD | WS_VISIBLE | WS_BORDER
| ES_AUTOHSCROLL, 10, 10, 300, 30, hWnd, NULL, hInstance, NULL);
    hwndSendButton = CreateWindow(L"BUTTON", L"Send", WS_CHILD | WS_VISIBLE |
BS_DEFPUSHBUTTON, 320, 10, 80, 30, hWnd, NULL, hInstance, NULL);
    hwndMessageLog = CreateWindow(L"EDIT", L"", WS_CHILD | WS_VISIBLE |
WS_BORDER | ES_AUTOVSCROLL | ES_MULTILINE | ES_READONLY, 10, 50, 390, 240,
hWnd, NULL, hInstance, NULL);

    // Show the main window
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    // Run the message loop
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```