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»

Выполнил:

студент гр. 153503

Филипеня А.Д.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 5](#_Toc146631500)

[Выводы 6](#_Toc146631501)

[Список использованных источников 7](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 8](#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является изучение интерфейса сокетов и основы сетевого программирования (*Windows*), программирование взаимодействия через сеть с использованием интерфейса сокетов, реализация стандартных сетевых протоколов, а также разработка клиент-серверного приложения для обмена текстовыми сообщениями с использованием *TCP* сокетов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.[1]

Средства обмена данными и совместного доступа в Windows представляют собой мощные инструменты для обмена информацией между различными процессами и устройствами в операционной системе Windows. Эти средства обеспечивают эффективную передачу данных, обмен сообщениями и разделяемый доступ к ресурсам.

Одним из наиболее распространенных средств обмена данными в Windows является Windows Sockets (Winsock). Winsock предоставляет стандартный интерфейс для разработки сетевых приложений, позволяя программам обмениваться данными по протоколам TCP/IP. Это средство позволяет приложениям работать с сетевыми ресурсами, такими как веб-серверы, базы данных и другие сетевые службы.

Для сокета необходимо указать три параметра. Это *IP* адрес, связанный с сокетом, номер порта, для которого будут выполняться операции передачи данных, а также тип сокета.

Что касается последнего параметра (тип сокета), то существуют сокеты двух типов. Первый тип предназначен для передачи данных в виде датаграмм, второй – с использованием каналов связи.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

1. Сокеты – это конечные точки для установления соединения между узлами в сети. Создание сокета происходит с помощью socket(). В данном случае используется сокет типа SOCK\_STREAM, что означает, что будет установлено надежное TCP-соединение.

2. Привязка к адресу и порту: сервер привязывает свой сокет к определенному IP-адресу и порту с помощью bind(). Это позволяет серверу слушать входящие соединения на указанном порту. Клиентский сокет привязан к серверному адресу и порту через вызов функции connect().

3. Прослушивание соединений: функция listen() используется для настройки серверного сокета на прослушивание входящих соединений. Это позволяет серверу принимать соединения от клиентов.

4. Прием клиентов: сервер ожидает входящие соединения с помощью accept().

5. Для получения данных от сервера используется recv(), для отправки данных на сервер – send().

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было разработано приложение сетевого локального чата между клиентом и сервером.

При запуске открывается два консольных окна. Сервер и клиент могут обмениваться сообщениями. На рисунке 3.1 показано, что клиент отправил сообщение серверу, который его получил, что показано на рисунке 3.2.

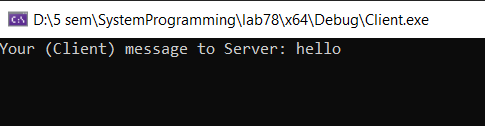


Рисунок 3.1 – Отправка сообщения от клиента к серверу

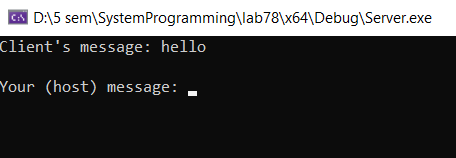


Рисунок 3.2 – Получение сообщения сервером от клиента

На рисунке 3.3 показано, что сервер отправил сообщение клиенту, который его получил, что показано на рисунке 3.4.

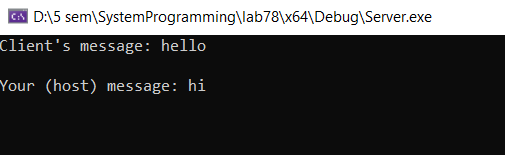


Рисунок 3.3 – Отправка сообщения от сервера клиенту

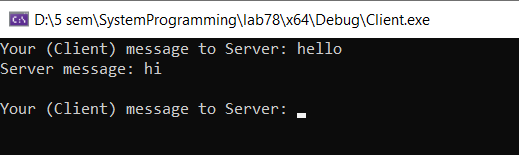


Рисунок 3.4 – Получение сообщения клиентом от сервера

## ВЫВОДЫ

В результате выполнения лабораторной работы были изучены интерфейс сокетов и основы сетевого программирования (*Windows*), программирование взаимодействия через сеть с использованием интерфейса сокетов, реализация стандартных сетевых протоколов, а также разработано клиент-серверное приложение для обмена текстовыми сообщениями с использованием *TCP* сокетов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.

**ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

**Server**

**main.cpp**

#include <iostream>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <stdio.h>

#include <vector>

#pragma comment(lib, "Ws2\_32.lib")

using namespace std;

int main(void) {

const short BUFF\_SIZE = 1024;

WSADATA wsData;

int error\_status = WSAStartup(MAKEWORD(2, 2), &wsData);

if (error\_status != 0) {

cout << "Error WinSock version initializaion " << WSAGetLastError();

return 1;

}

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

if (serverSocket == INVALID\_SOCKET) {

cout << "Invalid socket" << WSAGetLastError();

closesocket(serverSocket);

WSACleanup();

return 1;

}

//Привязка сокета к паре IP-адрес/Порт

in\_addr ipToNum;

error\_status = inet\_pton(AF\_INET, "127.0.0.1", &ipToNum);

if (error\_status <= 0) {

cout << "Error in IP translation to special numeric format" << endl;

return 1;

}

sockaddr\_in servSockInfo;

ZeroMemory(&servSockInfo, sizeof(servSockInfo));

servSockInfo.sin\_family = AF\_INET;

servSockInfo.sin\_addr = ipToNum;

servSockInfo.sin\_port = htons(1234);

error\_status = bind(serverSocket, (sockaddr\*)&servSockInfo, sizeof(servSockInfo));

if (error\_status != 0) {

cout << "Error Socket binding to server info. Error # " << WSAGetLastError() << endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

error\_status = listen(serverSocket, SOMAXCONN);

if (error\_status != 0) {

cout << "Error listen() " << WSAGetLastError() << endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

sockaddr\_in clientInfo;

ZeroMemory(&clientInfo, sizeof(clientInfo));

int clientInfo\_size = sizeof(clientInfo);

SOCKET ClientConn = accept(serverSocket, (sockaddr\*)&clientInfo, &clientInfo\_size);

if (ClientConn == INVALID\_SOCKET) {

cout << "Client detected, but can't connect to a client. Error # " << WSAGetLastError() << endl;

closesocket(serverSocket);

closesocket(ClientConn);

WSACleanup();

return 1;

}

vector<char> servBuffer(BUFF\_SIZE), clBuffer(BUFF\_SIZE);

short packet\_size = 0;

while (true) {

packet\_size = recv(ClientConn, servBuffer.data(), servBuffer.size(), 0);

cout << "Client's message: " << servBuffer.data() << endl;

cout << "Your (host) message: ";

fgets(clBuffer.data(), clBuffer.size(), stdin);

if (clBuffer[0] == 's' && clBuffer[1] == 't' && clBuffer[2] == 'o' && clBuffer[3] == 'p') {

shutdown(ClientConn, SD\_BOTH);

closesocket(serverSocket);

closesocket(ClientConn);

WSACleanup();

return 0;

}

packet\_size = send(ClientConn, clBuffer.data(), clBuffer.size(), 0);

if (packet\_size == SOCKET\_ERROR) {

cout << "Can't send message to Client. Error # " << WSAGetLastError() << endl;

closesocket(serverSocket);

closesocket(ClientConn);

WSACleanup();

return 1;

}

}

closesocket(serverSocket);

closesocket(ClientConn);

WSACleanup();

return 0;

}

**Client**

**main.cpp**

#include <iostream>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <stdio.h>

#include <vector>

#pragma comment(lib, "Ws2\_32.lib")

using namespace std;

int main(void) {

const short BUFF\_SIZE = 1024;

WSADATA wsData;

int error\_status = WSAStartup(MAKEWORD(2, 2), &wsData);

if (error\_status != 0) {

cout << "Error WinSock version initializaion " << WSAGetLastError();

return 1;

}

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

if (clientSocket == INVALID\_SOCKET) {

cout << "Invalid socket" << WSAGetLastError();

closesocket(clientSocket);

WSACleanup();

return 1;

}

//Привязка сокета к паре IP-адрес/Порт

in\_addr ipToNum;

error\_status = inet\_pton(AF\_INET, "127.0.0.1", &ipToNum);

if (error\_status <= 0) {

cout << "Error in IP translation to special numeric format" << endl;

return 1;

}

sockaddr\_in clientSockInfo;

ZeroMemory(&clientSockInfo, sizeof(clientSockInfo));

clientSockInfo.sin\_family = AF\_INET;

clientSockInfo.sin\_addr = ipToNum;

clientSockInfo.sin\_port = htons(1234);

error\_status = connect(clientSocket, (sockaddr\*)&clientSockInfo, sizeof(clientSockInfo));

if (error\_status != 0) {

cout << "Connection to server error # " << WSAGetLastError() << endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

vector<char> servBuffer(BUFF\_SIZE), clBuffer(BUFF\_SIZE);

short packet\_size = 0;

while (true) {

cout << "Your (Client) message to Server: ";

fgets(clBuffer.data(), clBuffer.size(), stdin);

if (clBuffer[0] == 's' && clBuffer[1] == 't' && clBuffer[2] == 'o' && clBuffer[3] == 'p') {

shutdown(clientSocket, SD\_BOTH);

closesocket(clientSocket);

WSACleanup();

return 0;

}

packet\_size = send(clientSocket, clBuffer.data(), clBuffer.size(), 0);

if (packet\_size == SOCKET\_ERROR) {

cout << "Can't send message to Server. Error # " << WSAGetLastError() << endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

packet\_size = recv(clientSocket, servBuffer.data(), servBuffer.size(), 0);

if (packet\_size == SOCKET\_ERROR) {

cout << "Can't receive message from Server. Error # " << WSAGetLastError() << endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

else

cout << "Server message: " << servBuffer.data() << endl;

}

closesocket(clientSocket);

WSACleanup();

return 0;

}