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

ОТЧЁТ

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 группы 153503

Киселёва Елизавета Андреевна

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc151682695)

[2 Краткие теоретические сведения 4](#_Toc151682696)

[3 Результаты выполнения лабораторной работы 6](#_Toc151682697)

[Выводы 8](#_Toc151682698)

[Список использованных источников 9](#_Toc151682699)

[Приложение А (обязательное) Листинг кода 10](#_Toc151682700)

## 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 создание клиент-серверного приложения для обмена текстовыми сообщениями с использованием *TCP* сокетов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В локальных и глобальных сетях существует два принципиально разных способа передачи данных.

Первый из них предполагает посылку пакетов данных от одного узла другому (или сразу нескольким узлам) без получения подтверждения о доставке и даже без гарантии того, что передаваемые пакеты будут получены в правильной последовательности. Примером такого протокола может служить протокол *UDP* (*User Datagram Protocol*), который используется в сетях *TCP*/*IP*, или протокол *IPX*, который является базовым в сетях *Novell* *NetWare*.

Основные преимущества датаграмных протоколов заключаются в высоком быстродействии и возможности широковещательной передачи данных, когда один узел отправляет сообщения, а другие их получают, причем все одновременно.

Второй способ передачи данных предполагает создание канала передачи данных между двумя различными узлами сети. При этом канал создаётся средствами датаграммных протоколов, однако доставка пакетов в канале является гарантированной. Пакеты всегда доходят в целостности и сохранности, причём в правильном порядке, хотя быстродействие получается в среднем ниже за счёт посылки подтверждений. Примерами протоколов, использующих каналы связи, могут служить протоколы *TCP* и *SPX* (протокол *NETBIOS* допускает передачу данных с использованием как датаграмм, так и каналов связи). [1]

Для передачи данных с использованием любого из перечисленных выше способов каждое приложение должно создать объект, который называется сокетом.

По своему назначению сокет больше всего похож на идентификатор файла (*file handle*), который нужен для выполнения над файлом операций чтения или записи. Прежде чем приложение, запущенное на узле сети сможет выполнять передачу или приём данных, оно должно создать сокет и проинициализировать его, указав некоторые параметры.

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.

В лабораторной работе использовались следующие функции работы с сокетами:

1 *socket*: Создание сокета. В данном случае, создаётся сокет семейства *AF\_INET* (*IPv*4), типа *SOCK\_STREAM* (для *TCP*), и протокола 0 (стандартный протокол для выбранного типа и семейства).

2 *inet\_addr*: Преобразование строки в сетевой адрес. В данном случае, строка «127.0.0.1» преобразуется в *IPv*4 адрес и устанавливается в поле *sin\_addr* структуры *addr*.

3 *connect*: Установка соединения с сервером. Клиентское приложение пытается установить соединение с сервером, используя созданный сокет и структуру *addr*, содержащую адрес и порт сервера.

4 *recv*: Получение данных от сервера. В цикле клиент принимает данные от сервера с использованием функции *recv*. В случае ошибки, выводится сообщение об ошибке.

5 *send*: Отправка данных на сервер. Клиент отправляет данные на сервер с использованием функции *send*. В случае ошибки, выводится сообщение об ошибке. [2]

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ    РАБОТЫ

В ходе выполнения лабораторной работы было разработано клиент-серверного приложения для обмена текстовыми сообщениями с использованием *TCP* сокетов.

При запуске приложения открывается два консольных окна. Первое из них – сервер (рисунок 3.1), второе – клиент (рисунок 3.2).

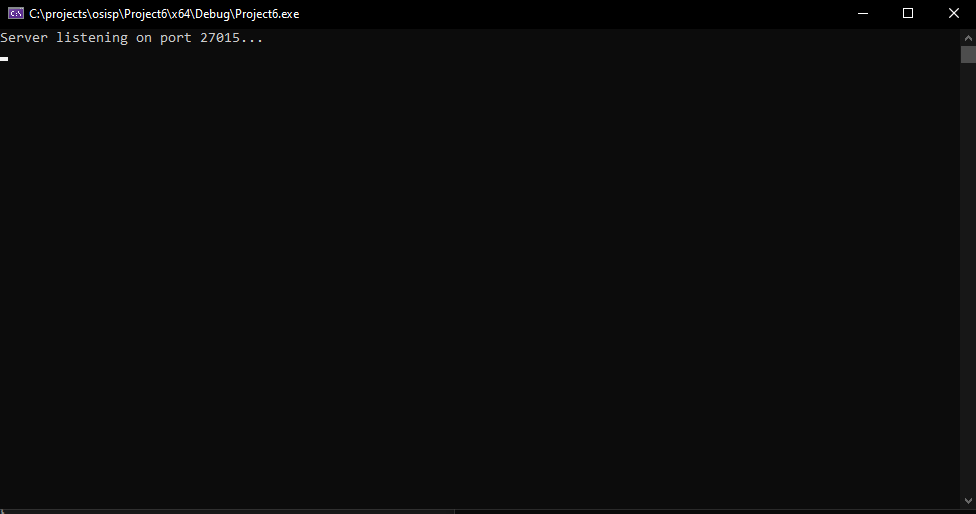
****

Рисунок 3.1 – Окно сервера

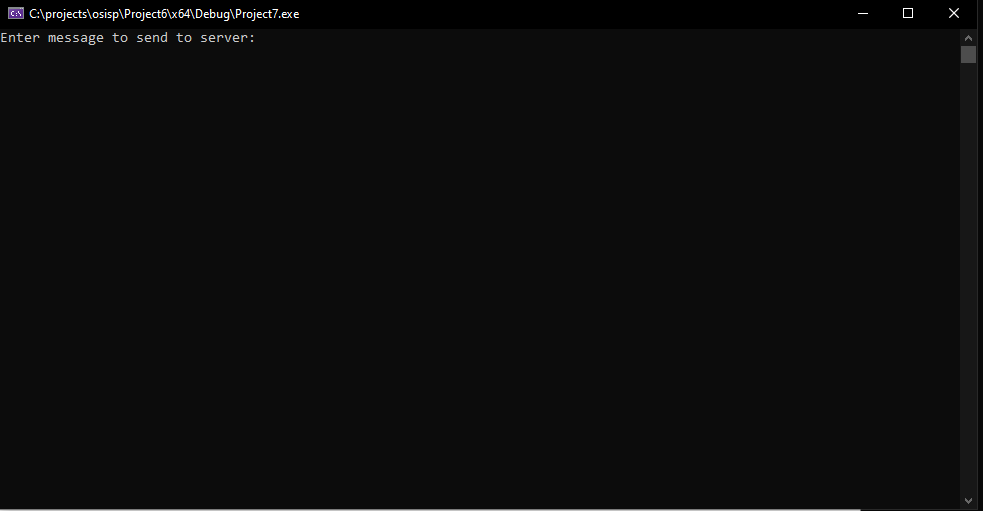


Рисунок 3.2 – Окно клиента

Сервер и клиент имеют возможность обмена сообщениями. Сервер получил сообщение от клиента и отправил ответ (рисунок 3.3).

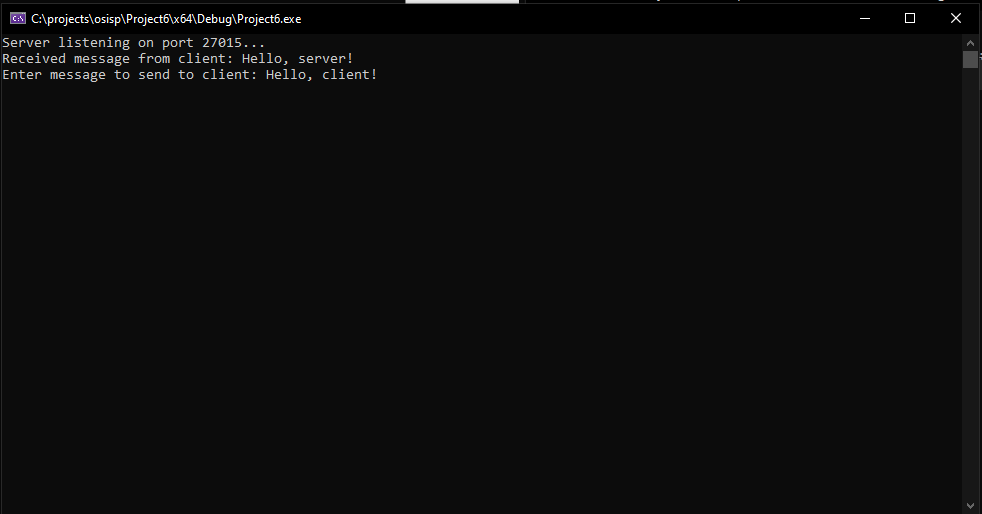
****

Рисунок 3.3 – Работа сервера

Клиент отправил сообщение серверу и получил ответ (рисунок 3.4).

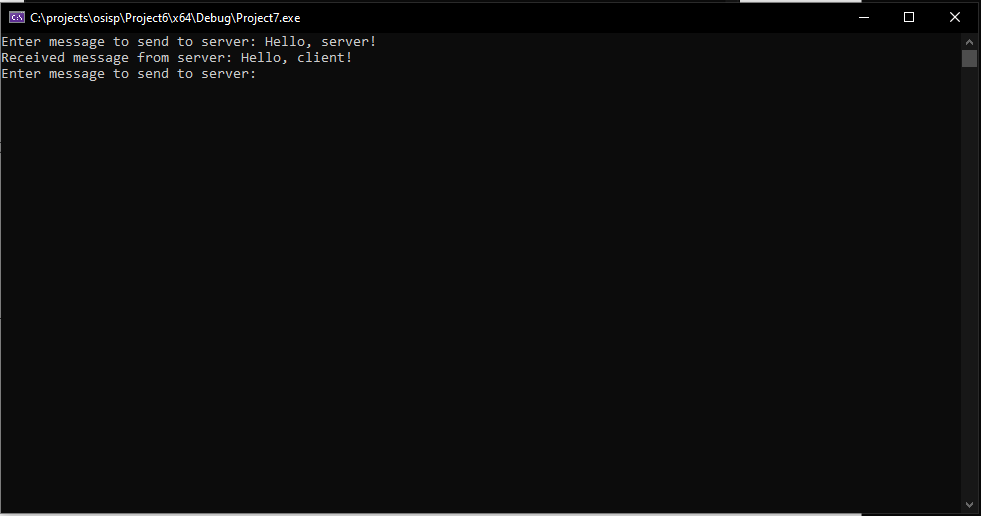


Рисунок 3.4 – Работа клиента

## ВЫВОДЫ

В ходе выполнения данной 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] Библиотека системного программиста : в 34 т. / А. В. Фролов. – Москва : Диалог-МИФИ, 1991-1997. – Т. 23 : Глобальные сети компьютеров. Практическое введение в Internet, E-Mail, FTP, WWW и HTML, программирование для Windows Sockets / Г. В. Фролов, 1996. – 283 с.

[2] Заголовок winsock2.h [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/winsock2/

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Server.cpp**

#include <iostream>

#include <string>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

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

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "27015"

int main() {

WSADATA wsaData;

int iResult;

SOCKET ListenSocket = INVALID\_SOCKET;

SOCKET ClientSocket = INVALID\_SOCKET;

struct addrinfo\* result = NULL;

struct addrinfo hints;

int recvbuflen = DEFAULT\_BUFLEN;

char recvbuf[DEFAULT\_BUFLEN];

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

std::cerr << "WSAStartup failed with error: " << iResult << std::endl;

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

hints.ai\_flags = AI\_PASSIVE;

iResult = getaddrinfo(NULL, DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

std::cerr << "getaddrinfo failed with error: " << iResult << std::endl;

WSACleanup();

return 1;

}

ListenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (ListenSocket == INVALID\_SOCKET) {

std::cerr << "socket failed with error: " << WSAGetLastError() << std::endl;

freeaddrinfo(result);

WSACleanup();

return 1;

}

iResult = bind(ListenSocket, result->ai\_addr, static\_cast<int>(result->ai\_addrlen));

if (iResult == SOCKET\_ERROR) {

std::cerr << "bind failed with error: " << WSAGetLastError() << std::endl;

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

iResult = listen(ListenSocket, SOMAXCONN);

if (iResult == SOCKET\_ERROR) {

std::cerr << "listen failed with error: " << WSAGetLastError() << std::endl;

closesocket(ListenSocket);

WSACleanup();

return 1;

}

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

ClientSocket = accept(ListenSocket, NULL, NULL);

if (ClientSocket == INVALID\_SOCKET) {

std::cerr << "accept failed with error: " << WSAGetLastError() << std::endl;

closesocket(ListenSocket);

WSACleanup();

return 1;

}

closesocket(ListenSocket);

do {

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);

if (iResult > 0) {

recvbuf[iResult] = '\0'; // Null-terminate the received data

std::cout << "Received message from client: " << recvbuf << std::endl;

// Get user input and send to the client

std::cout << "Enter message to send to client: ";

std::cin.getline(recvbuf, sizeof(recvbuf));

iResult = send(ClientSocket, recvbuf, strlen(recvbuf), 0);

if (iResult == SOCKET\_ERROR) {

std::cerr << "send failed with error: " << WSAGetLastError() << std::endl;

closesocket(ClientSocket);

WSACleanup();

return 1;

}

}

else if (iResult == 0) {

std::cout << "Connection closed by client\n";

break;

}

else {

std::cerr << "recv failed with error: " << WSAGetLastError() << std::endl;

closesocket(ClientSocket);

WSACleanup();

return 1;

}

} while (true);

iResult = shutdown(ClientSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

std::cerr << "shutdown failed with error: " << WSAGetLastError() << std::endl;

closesocket(ClientSocket);

WSACleanup();

return 1;

}

closesocket(ClientSocket);

WSACleanup();

return 0;

}

**Client.cpp**

#include <iostream>

#include <string>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

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

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "27015"

int main(int argc, char\*\* argv) {

WSADATA wsaData;

int iResult;

SOCKET ConnectSocket = INVALID\_SOCKET;

struct addrinfo\* result = NULL, \* ptr = NULL, hints;

char sendbuf[DEFAULT\_BUFLEN];

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

if (argc != 2) {

std::cerr << "usage: " << argv[0] << " <server IP address>\n";

return 1;

}

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

std::cerr << "WSAStartup failed with error: " << iResult << std::endl;

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

iResult = getaddrinfo(argv[1], DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

std::cerr << "getaddrinfo failed with error: " << iResult << std::endl;

WSACleanup();

return 1;

}

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype, ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

std::cerr << "socket failed with error: " << WSAGetLastError() << std::endl;

WSACleanup();

return 1;

}

iResult = connect(ConnectSocket, ptr->ai\_addr, static\_cast<int>(ptr->ai\_addrlen));

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

std::cerr << "Unable to connect to server! Error code: " << WSAGetLastError() << std::endl;

WSACleanup();

return 1;

}

do {

// Get user input and send to the server

std::cout << "Enter message to send to server: ";

std::cin.getline(sendbuf, sizeof(sendbuf));

iResult = send(ConnectSocket, sendbuf, strlen(sendbuf), 0);

if (iResult == SOCKET\_ERROR) {

std::cerr << "send failed with error: " << WSAGetLastError() << std::endl;

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

// Receive message from the server

iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);

if (iResult > 0) {

recvbuf[iResult] = '\0'; // Null-terminate the received data

std::cout << "Received message from server: " << recvbuf << std::endl;

}

else if (iResult == 0) {

std::cout << "Connection closed by server\n";

break;

}

else {

std::cerr << "recv failed with error: " << WSAGetLastError() << std::endl;

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

} while (true);

closesocket(ConnectSocket);

WSACleanup();

return 0;

}