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 гр. 153503

Звягинцева В.А.

Проверил:

Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Постановка задачи3

2 Краткие теоретические сведения4

3 Результаты выполнения лабораторной работы7

Вывод9

Список использованных источников10

Приложение А11

**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**

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 приложение сетевого локального чата между клиентом и сервером.

При запуске открывается два консольных окна. Сервер (рисунок 3.1) и клиент (рисунок 3.2).

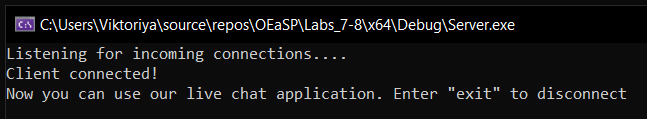


Рисунок 3.1. – Окно сервера

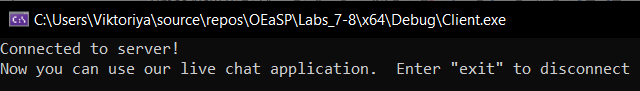


Рисунок 3.2. – Окно клиента

Сервер и клиент могут обмениваться сообщениями. На рисунке 3.3 показано, что сервер отправил сообщение клиенту и получил ответ.

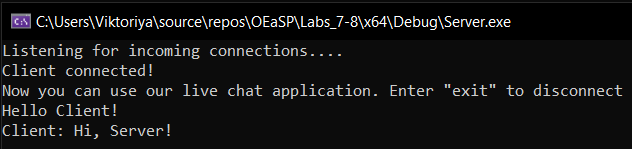


Рисунок 3.3 – Работа сервера

На рисунке 3.4 показано, что клиент получил сообщение от сервера и отправил ответ.

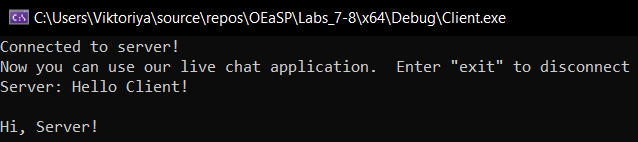


Рисунок 3.4 – Работа клиента

**ВЫВОД**

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 ИСПОЛЬЗУЕМЫХ 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**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <cstdio>

#include <cstring>

#include <winsock2.h>

#pragma comment(lib, "WS2\_32.lib")

using namespace std;

DWORD WINAPI serverReceive(LPVOID lpParam) { //Получение данных от клиента

char buffer[1024] = { 0 }; //Буфер для данных

SOCKET client = \*(SOCKET\*)lpParam; //Сокет для клиента

while (true) { //Цикл работы сервера

if (recv(client, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

//Если не удалось получить данные буфера, сообщить об ошибке и выйти

cout << "recv function failed with error " << WSAGetLastError() << endl;

return -1;

}

if (strcmp(buffer, "exit\n") == 0) { //Если клиент отсоединился

cout << "Client Disconnected." << endl;

break;

}

cout << "Client: " << buffer << endl; //Иначе вывести сообщение от клиента из буфера

memset(buffer, 0, sizeof(buffer)); //Очистить буфер

}

return 1;

}

DWORD WINAPI serverSend(LPVOID lpParam) { //Отправка данных клиенту

char buffer[1024] = { 0 };

SOCKET client = \*(SOCKET\*)lpParam;

while (true) {

fgets(buffer, 1024, stdin);

if (send(client, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

cout << "send failed with error " << WSAGetLastError() << endl;

return -1;

}

if (strcmp(buffer, "exit\n") == 0) {

cout << "Thank you for using the application" << endl;

break;

}

}

return 1;

}

int main() {

WSADATA WSAData; //Данные

SOCKET server, client; //Сокеты сервера и клиента

SOCKADDR\_IN serverAddr, clientAddr; //Адреса сокетов

WSAStartup(MAKEWORD(2, 0), &WSAData);

server = socket(AF\_INET, SOCK\_STREAM, 0); //Создали сервер

if (server == INVALID\_SOCKET) {

cout << "Socket creation failed with error:" << WSAGetLastError() << endl;

return -1;

}

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

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(5555);

if (bind(server, (SOCKADDR\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

cout << "Bind function failed with error: " << WSAGetLastError() << endl;

return -1;

}

if (listen(server, 0) == SOCKET\_ERROR) { //Если не удалось получить запрос

cout << "Listen function failed with error:" << WSAGetLastError() << endl;

return -1;

}

cout << "Listening for incoming connections...." << endl;

char buffer[1024]; //Создать буфер для данных

int clientAddrSize = sizeof(clientAddr); //Инициализировать адерс клиента

if ((client = accept(server, (SOCKADDR\*)&clientAddr, &clientAddrSize)) != INVALID\_SOCKET) {

//Если соединение установлено

cout << "Client connected!" << endl;

cout << "Now you can use our live chat application. " << "Enter \"exit\" to disconnect" << endl;

DWORD tid; //Идентификатор

HANDLE t1 = CreateThread(NULL, 0, serverReceive, &client, 0, &tid); //Создание потока для получения данных

if (t1 == NULL) //Ошибка создания потока

cout << "Thread Creation Error: " << WSAGetLastError() << endl;

HANDLE t2 = CreateThread(NULL, 0, serverSend, &client, 0, &tid); //Создание потока для отправки данных

if (t2 == NULL)

cout << "Thread Creation Error: " << WSAGetLastError() << endl;

WaitForSingleObject(t1, INFINITE);

WaitForSingleObject(t2, INFINITE);

closesocket(client); //Закрыть сокет

if (closesocket(server) == SOCKET\_ERROR) { //Ошибка закрытия сокета

cout << "Close socket failed with error: " << WSAGetLastError() << endl;

return -1;

}

WSACleanup();

}

}

**client.cpp**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <cstdio>

#include <cstring>

#include <winsock2.h>

#pragma comment(lib, "WS2\_32.lib")

using namespace std;

DWORD WINAPI clientReceive(LPVOID lpParam) { //Получение данных от сервера

char buffer[1024] = { 0 };

SOCKET server = \*(SOCKET\*)lpParam;

while (true) {

if (recv(server, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

cout << "recv function failed with error: " << WSAGetLastError() << endl;

return -1;

}

if (strcmp(buffer, "exit\n") == 0) {

cout << "Server disconnected." << endl;

return 1;

}

cout << "Server: " << buffer << endl;

memset(buffer, 0, sizeof(buffer));

}

return 1;

}

DWORD WINAPI clientSend(LPVOID lpParam) { //Отправка данных на сервер

char buffer[1024] = { 0 };

SOCKET server = \*(SOCKET\*)lpParam;

while (true) {

fgets(buffer, 1024, stdin);

if (send(server, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

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

return -1;

}

if (strcmp(buffer, "exit") == 0) {

cout << "Thank you for using the application" << endl;

break;

}

}

return 1;

}

int main() {

WSADATA WSAData;

SOCKET server;

SOCKADDR\_IN addr;

WSAStartup(MAKEWORD(2, 0), &WSAData);

if ((server = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET) {

cout << "Socket creation failed with error: " << WSAGetLastError() << endl;

return -1;

}

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //коннект к серверу

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(5555); //порт

if (connect(server, (SOCKADDR\*)&addr, sizeof(addr)) == SOCKET\_ERROR) {

cout << "Server connection failed with error: " << WSAGetLastError() << endl;

return -1;

}

cout << "Connected to server!" << endl;

cout << "Now you can use our live chat application. " << " Enter \"exit\" to disconnect" << endl;

DWORD tid;

HANDLE t1 = CreateThread(NULL, 0, clientReceive, &server, 0, &tid);

if (t1 == NULL) cout << "Thread creation error: " << GetLastError();

HANDLE t2 = CreateThread(NULL, 0, clientSend, &server, 0, &tid);

if (t2 == NULL) cout << "Thread creation error: " << GetLastError();

WaitForSingleObject(t1, INFINITE);

WaitForSingleObject(t2, INFINITE);

closesocket(server);

WSACleanup();

}