МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»

(БГТУ им. В.Г.Шухова)

**Лабораторная работа №4**

дисциплина «Сети ЭВМ и телекоммуникации»

тема: «**Программирование протоколов TCP/UDP с использованием библиотеки WinSock**»

Выполнил: студент группы ВТ-31

Ковалёв И. Д.

Проверил: Федотов Е. А.

Белгород 2020

**Цель работы:** Изучить протоколы TCP/UDP, основные функции библиотеки Winsock и составить программу для приема/передачи пакетов.

**Теоретические сведения**

**Протокол TCP**

Transmission Control Protocol (TCP) (протокол управления передачей) - один из основных сетевых протоколов Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Выполняет функции протокола транспортного уровня модели OSI.

TCP - это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи. Реализация TCP, как правило, встроена в ядро ОС, хотя есть и реализации TCP в контексте приложения.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. Также TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере. Программы для электронной почты и обмена файлами используют TCP. TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик.

Протокол TCP предназначен для обмена данными — это «надежный» протокол, потому что он, во-первых, обеспечивает надежную доставку данных, так как предусматривает установления логического соединения; во-вторых, нумерует пакеты и подтверждает их прием квитанцией, а в случае потери организует повторную передачу; в-третьих, делит передаваемый поток байтов на части — сегменты - и передает их нижнему уровню, на приемной стороне снова собирает их в непрерывный поток байтов.

TCP соединение начинается с т.н. “рукопожатия”: узел A посылает узлу B специальный пакет SYN — приглашение к соединению; B отвечает пакетом SYN-ACK — согласием об установлении соединения; A посылает пакет ACK — подтверждение, что согласие получено.

После этого TCP соединение считается установленным, и приложения, работающие в этих узлах, могут посылать друг другу пакеты с данными.

«Соединение» означает, что узлы помнят друг о друге, нумеруют все пакеты, идущие в обе стороны, посылают подтверждения о получении каждого пакета и перепосылают потерявшиеся по дороге пакеты. Для узла A это соединение называется исходящим, а для узла B - входящим. Любое установленное TCP соединение симметрично, и пакеты с данными по нему всегда идут в обе стороны.

В отличие от традиционной альтернативы - UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:

1. Установка соединения.

2. Передача данных.

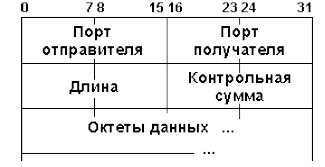
3. Завершение соединения.

**Протокол UDP**

Протокол UDP (User Datagram Protocol) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP. Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения. Протокол UDP не требует соединения с удаленным модулем UDP ("бессвязный" протокол). К заголовку IP-пакета UDP добавляет поля порт отправителя и порт получателя, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также поля длина UDP-дейтограммы и контрольная сумма, позволяющие поддерживать целостность данных. Таким образом, если на уровне IP для определения места доставки пакета используется адрес, на уровне UDP - номер порта.

UDP обеспечивает многоканальную передачу (с помощью номеров портов) и проверку целостности (с помощью контрольных сумм) заголовка и существенных данных. Надежная передача в случае необходимости должна реализовываться пользовательским приложением.

Заголовок UDP состоит из четырех полей, каждое по 2 байта (16 бит). Два из них необязательны к использованию в IPv4, в то время как в IPv6 необязателен только порт отправителя. В поле Порт отправителя указывается номер порта отправителя. Предполагается, что это значение задает порт, на который при необходимости будет посылаться ответ. В противном же случае, значение должно быть равным 0. Поле Порт получателя обязательно и содержит порт получателя. Поле Длина датаграммы задает длину всей датаграммы (заголовка и данных) в байтах. Минимальная длина равна длине заголовка – 8 байт. Теоретически, максимальный размер поля – 65535 байт для UDP-датаграммы (8 байт на заголовок и 65527 на данные). Фактический предел для длины данных при использовании IPv4 – 65507 (помимо 8 байт на UDP-заголовок требуется еще 20 на IP-заголовок).



Из-за недостатка надежности, приложения UDP должны быть готовыми к некоторым потерям, ошибкам и дублированиям. Некоторые из них (например, TFTP) могут при необходимости добавить элементарные механизмы обеспечения надежности на прикладном уровне.

Более серьезной потенциальной проблемой является то, что в отличие от TCP, основанные на UDP приложения не обязательно имеют хорошие механизмы контроля и избежания перегрузок. Чувствительные к перегрузкам UDP-приложения, которые потребляют значительную часть доступной пропускной способности, могут поставить под угрозу стабильность в Интернете.

**Задание к работе**

1. Разработать программу “Сервер” на языке программирования Pascal или C), которая принимает запросы от клиентов и посылает им в качестве ответа некоторое сообщение.

2. Разработать программу “Клиент” на языке программирования Pascal или C), которая посылает запрос серверу и “ждет” от него ответного сообщения.

3. Провести анализ функционирования разработанных программ (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

4. Провести сравнительный анализ протоколов TCP и UDP. Сделать выводы.

**Основные функции API, использованные в данной работе:**

В целом, структура программ и использованные функции не отличаются многим от таковых, использованных в 3 лабораторной работе, поскольку в процессе выполнения обеих работ использовалась библиотека winsock.h

**int WSAStartup(WORD wVersionRequired, LPWSADATA lpWSAData);**

**Назначение:** инициализирует Winsock dll. Возвращает 0, если инициализация прошла успешно, иначе – код ошибки.

**Void WSACleanup();**

**Назначение:** завершает работу с Winsock dll. Возвращает 0, если закрытие прошло успешно, иначе – код ошибки.

**SOCKET WSAAPI socket(int af, int type, int protocol);**

**Назначение:** создает сокет.

Af – семейство адресов. Был использован AF\_IPX.

Type – тип создаваемого сокета.

Protocol – используемый протокол.

**bind(s, (sockaddr\*)& clientAdress, sizeof(SOCKADDR\_IPX));**

**Назначение:** связывает локальный адрес с сокетом. В случае успеха возвращает 0, иначе – код ошибки.

S – дескриптор несвязанного сокета.

Name – указатель на структуру sockaddr.

Namelen – размер структуры sockaddr.

**int WSAAPI sendto(SOCKET s, const char \*buf, int len, int flags, const sockaddr \*to, int tolen);**

**Назначение:** отправляет данные data по определенному адресу to\*.

**Анализ протоколов**

В процессе выполнения работы с сервера отправлялся файл размером 59034 байт.

**UDP:**

|  |  |  |
| --- | --- | --- |
|  | **Время, сек** | **Принято байт** |
| **1 – 1** | **1,285** | 59034 |
| **2 -2** | **2,399** | 58991 |
| **3 -3** | **6,549** | 58123 |

**TCP:**

|  |  |  |
| --- | --- | --- |
|  | **Время, сек** | **Принято байт** |
| **1 – 1** | **1,311** | 59034 |
| **2 -2** | **2,542** | 59034 |
| **3 -3** | **7,375** | 59034 |

**Вывод:** в процессе выполнения лабораторной работы были изучены протоколы TCP/UDP, основные функции библиотеки winsock и составлены программы для приема-передачи пакетов.

Проанализировав данные, полученные в результате выполнения лабораторной работы, можно прийти к выводу, что протокол TCP гарантирует отправку пакетов без потерь, поскольку существует понятие установки соединения, что гарантирует надежную доставку. Но из за этого передача длится немного дольше, поскольку установка соединения требует времени. Передача по протоколу UDP немного быстрее, но не гарантирует доставку пакетов без потерь.

**Контрольные вопросы:**

1. Что представляет собой протокол TCP? Как он работает?

Протокол транспортного уровня OSI. Предназначен для управления передачи данных. Перед передачей устанавливает соединение между сервером и клиентом. В случае, если были потеряны данные, осуществляет повторный запрос данных, так же устраняет дублирование при получении двух одинаковых пакетов. Гарантирует целостность передаваемых данных.

1. Порядок установления TCP-соединения.

Состоит из трех этапов.

1. Клиент посылает серверу фрагмент с номером последовательности и флагом синхронизации. Сервер получает эти данные и создает сокет для обслуживания клиента. В результате передает клиенту сообщение об успехе или провале данной операции.
2. Если клиент получает сообщение о том, что операция была проведена успешно, переходит в состояние, которое сигнализирует о том, что соединение было установлено, иначе клиент прекращает попытки соединения (если был получен сегмент с установленным флагом reset) либо после ожидания в 10 секунд повторяет подключение (если ответ не был получен).
3. Сервер, получая от клиента сообщение о том, что тот перешел в состояние установленного соединения, сам сервер тоже переходит в это состояние.
4. В чём состоит отличие протокола UDP от IP?

Протокол UDP мультиплексирует соединение, используя порты. Работает он поверх IP, добавляя в кадр IP поля, которые несут информацию о порте получателя, порте отправителя, длине сообщения и контрольную сумму.

1. Формат заголовка пакета UDP.

Выделяется по 16 бит на:

номер порта отправителя,

номер порта получателя,

длина,

контрольная сумма.

После этого заголовка располагаются данные.

1. Опишите работу функций sendto и send библиотеки Winsocket.

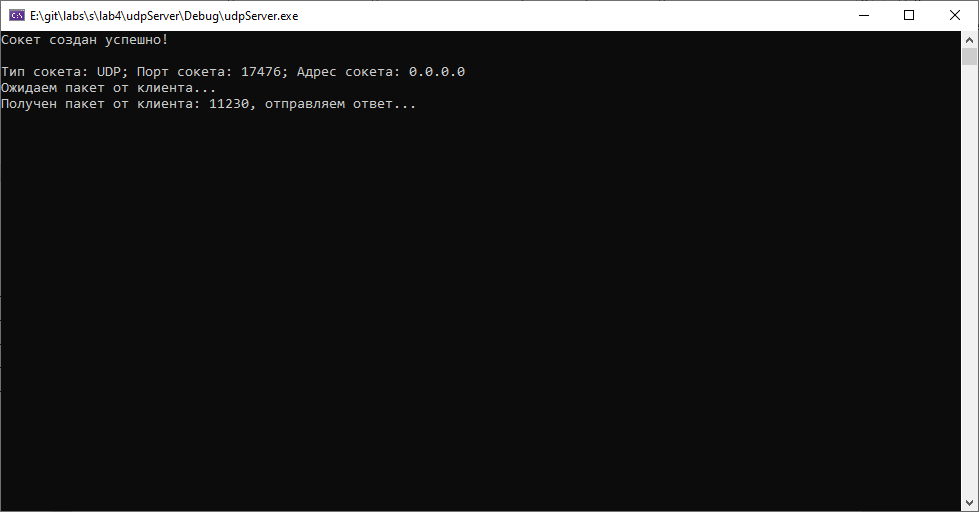
Функция Send применятся, когда существует уже открытый сокет с установленным соединением. sendto используется для отправки данных как через сокет с установлением соединения, так и с сокетом без установления соединения. При этом если sendto используется с

сокетом без установления соединения, то пакет отправляется на адрес назначения, указанный в аргументе to. Если sendto используется с сокетом, ориентированным на установление соединения, адрес получателя, указанный в аргументе, игнорируется, и функция sendto становится эквивалентной функции send.

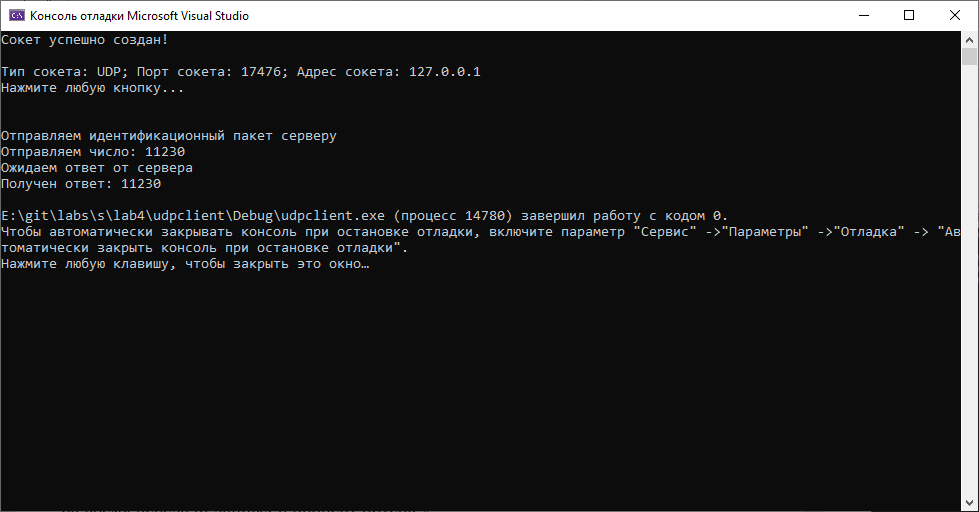
1. В каких случаях предпочтительней использовать протокол UDP?Этот протокол желательно использовать в приложениях, в которых нет необходимости в длительной установке соединений и отправки подтверждений, а так же приложения, для работы которых необходима широковещательная рассылка

**Скриншоты разработанных программ**

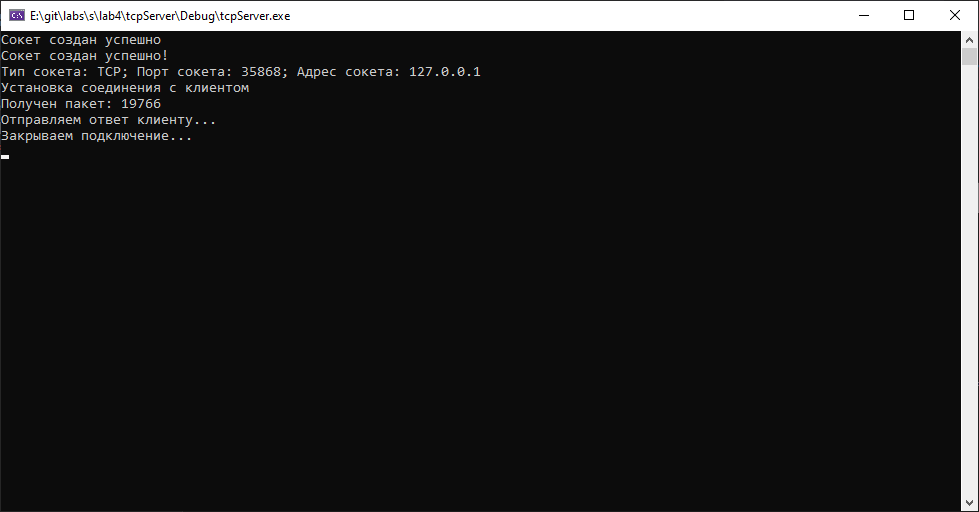
**UdpServer:**



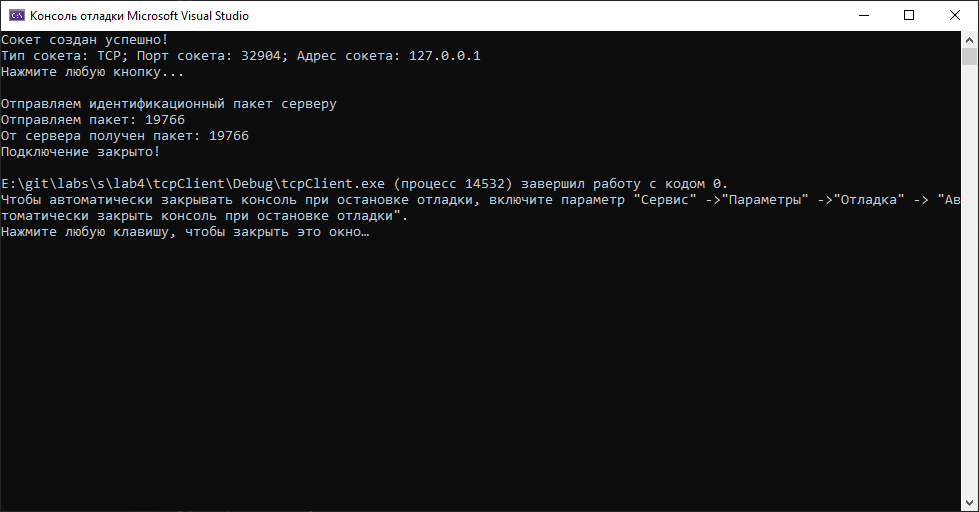
**UdpClient:**



**TcpServer:**



**TcpClient:**



**Листинг программ, реализующих получение ответа клиенту сервером:**

1. **UdpClient.cpp:**

// udpclient.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <windows.h>

#include <winsock.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <ctime>

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

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

void intHandler(int dummy) {

exit(0);

}

int main() {

setlocale(LC\_ALL, "Russian");

signal(SIGINT, intHandler);

time\_t t;

srand((unsigned)time(&t));

int data = rand();

WSADATA wsaData = { 0 };

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

int sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (sock == SOCKET\_ERROR) {

perror("Ошибка создания сокета!\n");

return 1;

};

printf("Сокет успешно создан!\n");

struct sockaddr\_in address;

address.sin\_family = AF\_INET;

unsigned short port = 0x4444;

address.sin\_port = htons(port);

char addressStr[] = "127.0.0.1";

address.sin\_addr.s\_addr = inet\_addr(addressStr);

printf("\nТип сокета: UDP; Порт сокета: %hu; Адрес сокета: %s\n", port, addressStr);

printf("Нажмите любую кнопку...\n");

getchar();

getchar();

printf("Отправляем идентификационный пакет серверу\n");

printf("Отправляем число: %d\n", data);

int result = sendto(sock, (char\*)(&data), sizeof(int), 0, (SOCKADDR\*)&addressStr, sizeof(addressStr));

if (result == SOCKET\_ERROR) {

printf("\nПроизошла ошибка при отправке. Код ошибки: %d\n", WSAGetLastError());

closesocket(sock);

WSACleanup();

return 1;

}

printf("Ожидаем ответ от сервера\n");

int echo;

struct sockaddr\_in senderAdress;

int senderAdress\_size = sizeof(senderAdress);

result = recvfrom(sock, (char\*)(&echo), sizeof(int), 0, (SOCKADDR\*)&senderAdress, &senderAdress\_size);

if (result == SOCKET\_ERROR) {

printf("Ошибка при получении ответа. Код ошибки: %d\n", WSAGetLastError());

closesocket(sock);

WSACleanup();

return 1;

};

printf("Получен ответ: %d\n", echo);

return 0;

}

1. **UdpServer.cpp:**

// udpServer.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <windows.h>

#include <winsock.h>

#include <stdlib.h>

#include <stdio.h>

#include <sys/types.h>

#include <signal.h>

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

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

typedef struct \_thread\_arg {

struct sockaddr\_in adr;

int data;

} thread\_arg;

DWORD WINAPI sendEcho(LPVOID arg) {

printf("Получен пакет от клиента: %d, отправляем ответ...\n", ((thread\_arg\*)arg)->data);

int sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (sock == SOCKET\_ERROR) {

perror("Ошибка открытия сокета\n");

return 1;

};

int result = sendto(sock, (char\*)(&((thread\_arg\*)arg)->data), sizeof(int), 0, (SOCKADDR\*)&((thread\_arg\*)arg)->adr, sizeof(((thread\_arg\*)arg)->adr));

if (result == SOCKET\_ERROR) {

printf("Ошибка при отправке ответа. Код ошибки: %d\n", WSAGetLastError());

closesocket(sock);

WSACleanup();

return 1;

}

closesocket(sock);

return 0;

}

void intHandler(int dummy) {

exit(0);

}

int main() {

setlocale(LC\_ALL, "Russian");

signal(SIGINT, intHandler);

WSADATA wsaData = { 0 };

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

int sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (sock == SOCKET\_ERROR) {

perror("Ошибка создания сокета\n");

return 1;

};

printf("Сокет создан успешно!\n");

struct sockaddr\_in address;

address.sin\_family = AF\_INET;

unsigned short port = 0x4444;

address.sin\_port = htons(port);

address.sin\_addr.s\_addr = htonl(INADDR\_ANY);

printf("\nТип сокета: UDP; Порт сокета: %hu; Адрес сокета: %s\n", port, inet\_ntoa(address.sin\_addr));

if (bind(sock, (SOCKADDR\*)&address, sizeof(address)) < 0) {

printf("Ошибка при назначении сокету адрес! Код ошибки: %d\n", WSAGetLastError());

return 2;

};

printf("Ожидаем пакет от клиента...\n");

int data;

while (1) {

struct sockaddr\_in senderAdress;

int senderAdressSize = sizeof(senderAdress);

int result = recvfrom(sock, (char\*)(&data), sizeof(int), 0, (SOCKADDR\*)&senderAdress, &senderAdressSize );

if (result == SOCKET\_ERROR) {

closesocket(sock);

WSACleanup();

return 1;

}

else {

thread\_arg arg;

arg.data = data;

arg.adr = senderAdress;

HANDLE thread = CreateThread(NULL, 0, sendEcho, &arg, 0, NULL);

};

};

return 0;

}

1. **TcpClient.cpp:**

// tcpClient.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

#define WIN32\_LEAN\_AND\_MEAN

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "27015"

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <ctime>

#include <iostream>

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

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

void intHandler(int dummy) {

exit(0);

}

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

setlocale(LC\_ALL, "Russian");

signal(SIGINT, intHandler);

WSADATA wsaData;

SOCKET ConnectSocket = INVALID\_SOCKET;

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

time\_t t;

srand((unsigned)time(&t));

int data = rand();

char sendbuf[DEFAULT\_BUFLEN];

((int\*)sendbuf)[0] = data;

char recvbuf[DEFAULT\_BUFLEN];

int iResult;

int recvbuflen = DEFAULT\_BUFLEN;

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

if (iResult != 0) {

printf("Ошибка инициализации WSA! Код ошибки: %d\n", iResult);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

char address[] = "127.0.0.1";

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

if (iResult != 0) {

printf("Ошибка получении информации об адресе! Код ошибки: %d\n", iResult);

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) {

printf("Ошибка создания сокета! Код ошибки: %ld\n", WSAGetLastError());

WSACleanup();

return 1;

}

printf("Сокет создан успешно!\n");

printf("Тип сокета: TCP; Порт сокета: %hu; Адрес сокета: %s\n", DEFAULT\_PORT, address);

printf("Нажмите любую кнопку...\n");

getchar();

printf("Отправляем идентификационный пакет серверу\n");

printf("Отправляем пакет: %d\n", ((int\*)sendbuf)[0]);

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

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

printf("Не удалось подключится к серверу!!\n");

WSACleanup();

return 1;

}

iResult = send(ConnectSocket, sendbuf, sizeof(int), 0);

if (iResult == SOCKET\_ERROR) {

printf("Ошибка при отправке! Код ошибки: %d\n", WSAGetLastError());

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

iResult = shutdown(ConnectSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

do {

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

if (iResult > 0)

printf("От сервера получен пакет: %d\n", ((int\*)recvbuf)[0]);

else if (iResult == 0)

printf("Подключение закрыто!\n");

else

printf("Ошибка при получении ответа! Код ошики: %d\n", WSAGetLastError());

} while (iResult > 0);

closesocket(ConnectSocket);

WSACleanup();

return 0;

}

1. **TcpServer.cpp:**

// tcpServer.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <iostream>

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

#pragma comment (lib, "Mswsock.lib")

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "27015"

typedef struct \_threadArg {

struct sockaddr\_in adr;

int data;

} threadArg;

DWORD WINAPI sendEcho(LPVOID arg) {

SOCKET ClientSocket = (\*(SOCKET\*)arg);

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

int iResult;

int iSendResult;

printf("Установка соединения с клиентом\n");

do {

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

if (iResult > 0) {

printf("Получен пакет: %d\n", ((int\*)recvbuf)[0]);

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

if (iSendResult == SOCKET\_ERROR) {

printf("Ошибка отправки! Код ошибки: %d\n", WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

return 1;

}

printf("Отправляем ответ клиенту...\n");

}

else if (iResult == 0)

printf("Закрываем подключение...\n");

else {

printf("Ошибка отправки! Код ошибки: %d\n", WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

return 1;

}

} while (iResult > 0);

iResult = shutdown(ClientSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

closesocket(ClientSocket);

WSACleanup();

return 1;

}

closesocket(ClientSocket);

free(arg);

}

void intHandler(int dummy) {

exit(0);

}

int \_\_cdecl main(void){

setlocale(LC\_ALL, "Russian");

signal(SIGINT, intHandler);

WSADATA wsaData;

int iResult;

SOCKET listenSocket = INVALID\_SOCKET;

struct addrinfo\* result = NULL;

struct addrinfo hints;

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

if (iResult != 0) {

printf("Ошибка инициализации WSA! Код ошибки: %d\n", iResult);

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) {

printf("Ошибка при получении адреса и порта! Код ошибки: %d\n", iResult);

WSACleanup();

return 1;

}

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

if (listenSocket == INVALID\_SOCKET) {

printf("Ошибка создания сокета! Код ошибки: %ld\n", WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

printf("Сокет создан успешно\n");

iResult = bind(listenSocket, result->ai\_addr, (int)result->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

freeaddrinfo(result);

closesocket(listenSocket);

WSACleanup();

return 1;

}

printf("Сокет создан успешно!\n");

printf("Тип сокета: TCP; Порт сокета: %hu; Адрес сокета: 127.0.0.1\n", DEFAULT\_PORT);

freeaddrinfo(result);

while (1) {

iResult = listen(listenSocket, SOMAXCONN);

if (iResult == SOCKET\_ERROR) {

closesocket(listenSocket);

WSACleanup();

return 1;

}

SOCKET\* ClientSocket = new SOCKET;

(\*ClientSocket) = accept(listenSocket, NULL, NULL);

if (\*ClientSocket == INVALID\_SOCKET) {

closesocket(listenSocket);

WSACleanup();

return 1;

}

HANDLE thread = CreateThread(NULL, 0, sendEcho, ClientSocket, 0, NULL);

}

closesocket(listenSocket);

WSACleanup();

return 0;

}