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 №3
дисциплина «Сети ЭВМ и телекомуникации»
по теме «Программирование протокола IP с использованием библиотеки Winsock»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Федотов Е.А.

Белгород 2020

Лабораторная работа №3

«Программирование протокола IP с использованием библиотеки Winsock»

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

Вариант 6

Содержание отчета

1. Краткие теоретические сведения.
2. Основные функции API, использованные в данной работе.
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Ход работы

1. **Краткие теоретические сведения.**

Transmission Control Protocol (TCP, протокол управления передачей) — один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных.

В стеке протоколов TCP/IP выполняет функции транспортного уровня модели OSI.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от UDP, целостность передаваемых данных и уведомление отправителя о результатах передачи.

Реализации TCP обычно встроены в ядра ОС. Существуют реализации TCP, работающие в пространстве пользователя.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. TCP осуществляет надёжную передачу потока байтов от одного процесса к другому. TCP реализует управление потоком, управление перегрузкой, рукопожатие, надёжную передачу.

UDP (англ. User Datagram Protocol — протокол пользовательских датаграмм) — один из ключевых элементов набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае

называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных. Протокол был разработан Дэвидом П. Ридом в 1980 году и официально определён в RFC 768.

UDP использует простую модель передачи, без неявных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, UDP предоставляет ненадёжный сервис, и датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в системах реального времени. При необходимости исправления ошибок на сетевом уровне интерфейса приложение может задействовать TCP или SCTP, разработанные для этой цели.

Природа UDP как протокола без сохранения состояния также полезна для серверов, отвечающих на небольшие запросы от огромного числа клиентов, например DNS и потоковые мультимедийные приложения вроде IPTV, Voice over IP, протоколы туннелирования IP и многие онлайн-игры.

1. **Основные функции API, использованные в данной работе.**

- `WSAStartup(WORD wVersionRequested,LPWSADATA lpWSAData)`
- `WSACleanup (void)`
- `WSAGetLastError (void)`
- `u_short htons (u_short hostshort)`
- `socket (int af, int type, int protocol)`
- `bind (SOCKET s, const struct sockaddr FAR* name, int namelen)`
- `recvfrom (SOCKET s, char FAR* buf, int len, int flags,struct sockaddr FAR* from, int FAR* fromlen)`
- `sendto (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen)`

3. **Разработка программы. Блок-схемы программы.**

   Для UDP клиент и сервер имеют структуру как в предыдущей лабораторной работе.

   В сервере TCP добавлены этапы установки соединения между клиентом и сервером.

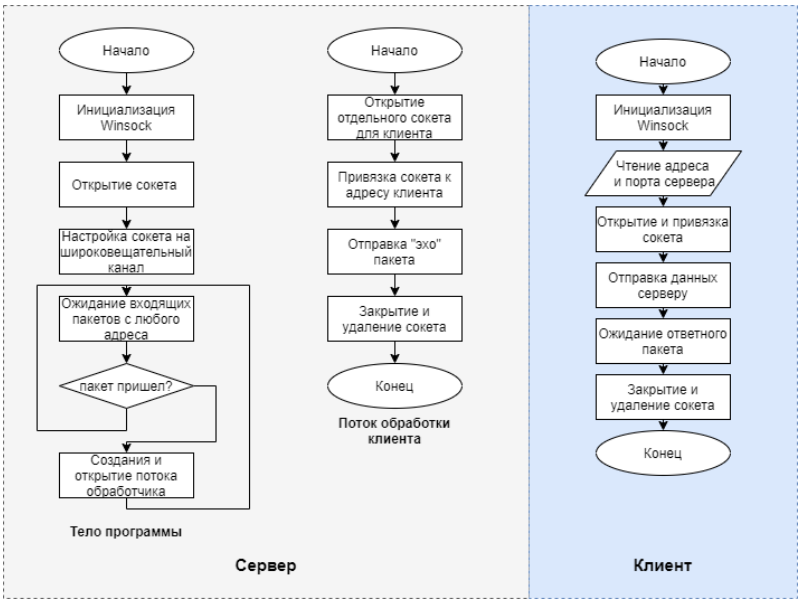

Рис. 1: Блок схемы программы сервер/клиент UDP

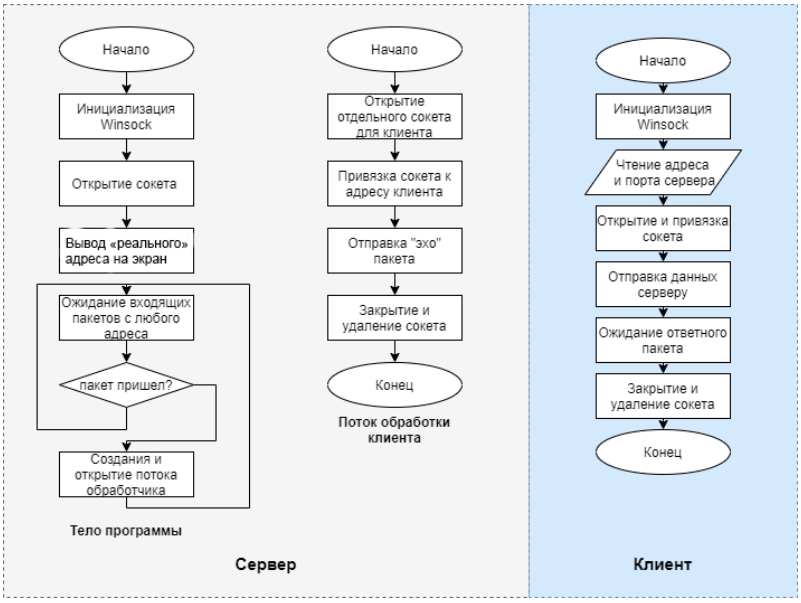

Рис. 2: Блок схемы программы сервер/клиент TCP

4. **Анализ функционирования разработанных программ.**

Анализ функционирования программ проводился в пределах одного ПК (за неимением возможности протестировать передачу данных между несколькими машинами). Для одновременного запуска нескольких клиентов был написан скрипт, запускающий несколько экземпляров программы-клиент. Текст скрипта приведен ниже:

```
@echo off
set loopcount=1000
```

```
:loop
start /b client < input.txt
set /a loopcount=loopcount-1
if %loopcount%==0 goto exitloop
goto loop
:exitloop
pause
```

Ошибок в передаче 1000 пакетов что в TCP что в UDP не было обнаружено. Все клиенты получили обратно “эхо” пакеты.

5. **Выводы.**

Библиотека Winsock имеет умеет удобно API для написание многопоточного сетевого программного обеспечения для операционной системы Windows, с использованием протоколов TCP и UDP. Протокол TCP гарантирует порядок и доставку пакетов ценой скорости. Протокол UDP похож в использовании на чистый IP и не гарантирует доставку и порядок пакетов.

6. **Тексты программ. Скриншоты программ.**

Тексты программ см. в приложении.

Рис. 3: Программа-сервер

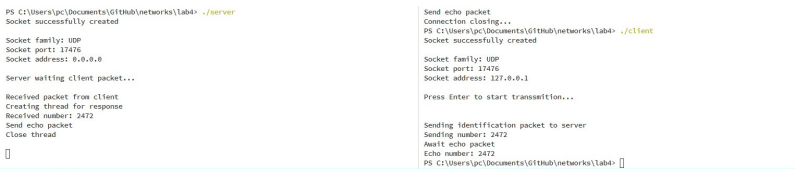

Рис. 4: Программа-клинет

7. **Контрольные вопросы**

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

Протокол TCP - один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных. Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от UDP, целостность передаваемых данных и уведомление отправителя о результатах передачи.

2. Порядок установления TCP-соединения.

- Клиент, который намеревается установить соединение, посылает серверу сегмент с номером последовательности и флагом SYN.
- Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK.
- Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.
- В противном случае после тайм-аута он закрывает сокет и переходит в состояние CLOSED.

3. В чём состоит отличие протокола UDP от IP?

Протокол IP - протокол сетевого уровня и обеспечивает обьединение сегментов сетей в одну обеспечивая доставку между узлами. Протокол UDP - протокол транспортного уровня, простейшая надстройка над IP добавляющая в пакет 4 поля (порт приемника, получателя, длина, контрольная сумма). Не гарантирует доставку и порядок отправленных пакетов.

4. Формат заголовка пакета UDP.

порт приемника, получателя, длина, контрольная сумма

5. Опишите работу функций sendto и send библиотеки Winsocket.

Функция **sendto** предназначена для отправки данных, без установки соединения (UDP).

Аргументы:

```
* socket - сокет, через который будут отправлены данные.
* message - указатель на буфер, содержащий данные для отправки.
* length - определяет длину сообщения в байтах.
* flags - определяет параметры передачи сообщения.
* dest_addr - указатель на структуру, содержащую адрес получателя.
* dest_len - определяет длину структуры, на которую указывает dest_add
```

Функция **send** предназначена для отправки данных, при установленном соединении (TCP).

Аргументы:

```
* socket - сокет, через который будут отправлены данные.
* message - указатель на буфер, содержащий данные для отправки.
* length - определяет длину сообщения в байтах.
* flags - определяет параметры передачи сообщения.
```

6. В каких случаях предпочтительней использовать протокол UDP?

Для систем реального времени, передачи звуковой и видео информации, компьютерные игры, когда важна скорость получения пакетов нежели их целостность и порядок.

Приложение

Содержимое файла makefile

```
all: build-all

build-all: build-tcp

build-tcp: tcp-build-server tcp-build-client

build-udp: udp-build-server udp-build-client

tcp-build-client:
	gcc tcp_client.cpp -o client.exe -lws2_32

tcp-build-server:
	gcc tcp_server.cpp -o server.exe -lws2_32

udp-build-client:
	gcc udp_client.cpp -o client.exe -lws2_32

udp-build-server:
	gcc udp_server.cpp -o server.exe -lws2_32
```

Содержимое файла tcp-client.cpp

```
#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <ctime>


// Need to link with Ws2_32.lib, Mswsock.lib, and Advapi32.lib
#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")

void intHandler(int dummy) {
	printf("Exit by keyboard interrupt\n");
	exit(0);
}

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "60000"

int __cdecl main(int argc, char **argv)
{
	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;

// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
        printf("WSAStartup failed with error: %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";

// Resolve the server address and port
iResult = getaddrinfo(address, DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
        printf("getaddrinfo failed with error: %d\n", iResult);
        WSACleanup();
        return 1;
}

// Attempt to connect to an address until one succeeds
for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {

        // Create a SOCKET for connecting to server
        ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
                ptr->ai_protocol);
        if (ConnectSocket == INVALID_SOCKET) {
                printf("socket failed with error: %ld\n", WSAGetLastError());
                WSACleanup();
                return 1;
        }
        printf("Socket succesfully created\n");

        printf("\nSocket family: TCP\n");
        printf("Socket port: 60000\n");
        printf("Socket address: %s\n\n", address);

        printf("Press Enter to start transsmition...\n");
        getchar();
        getchar();
        printf("Sending identification packet to server\n");
        printf("Sending number: %d\n", ((int *)sendbuf)[0]);

        // Connect to server.
        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("Unable to connect to server!\n");
                WSACleanup();
                return 1;
        }

        // Send an initial buffer
        iResult = send(ConnectSocket, sendbuf, sizeof(int), 0);
        if (iResult == SOCKET_ERROR) {
                printf("send failed with error: %d\n", WSAGetLastError());
                closesocket(ConnectSocket);
                WSACleanup();
                return 1;
        }

        // shutdown the connection since no more data will be sent
        iResult = shutdown(ConnectSocket, SD_SEND);
        if (iResult == SOCKET_ERROR) {
                printf("shutdown failed with error: %d\n", WSAGetLastError());
                closesocket(ConnectSocket);
                WSACleanup();
                return 1;
        }

        // Receive until the peer closes the connection
        do {

                iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
                if (iResult > 0)
                        printf("Received number: %d\n", ((int *)recvbuf)[0]);
                else if (iResult == 0)
                        printf("Connection closed\n");
                else
                        printf("recv failed with error: %d\n", WSAGetLastError());

        } while (iResult > 0);

        // cleanup
        closesocket(ConnectSocket);
        WSACleanup();

        return 0;
}
```

Содержимое файла tcp-server.cpp

```
#undef UNICODE

#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
```

```c
#include <signal.h>

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

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "60000"

typedef struct _thread_arg {
        struct sockaddr_in adr;
        int data;
} thread_arg;

DWORD WINAPI send_echo_adr(LPVOID arg) {
        SOCKET ClientSocket = (*(SOCKET *)arg);
        char recvbuf[DEFAULT_BUFLEN];
        int recvbuflen = DEFAULT_BUFLEN;
        int iResult;
        int iSendResult;
        printf("\Establishing connection with client\n");
        // Получаем, пока клиент не разорвет соединение
        do {

                iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
                if (iResult > 0) {
                        printf("Received number: %d\n", ((int *)recvbuf)[0]);

                        // Echo the buffer back to the sender
                        iSendResult = send(ClientSocket, recvbuf, iResult, 0);
                        if (iSendResult == SOCKET_ERROR) {
                                printf("send failed with error: %d\n", WSAGetLastError());
                                closesocket(ClientSocket);
                                WSACleanup();
                                return 1;
                        }
                        printf("Send echo packet\n");
                }
                else if (iResult == 0)
                        printf("Connection closing...\n");
                else {
                        printf("recv failed with error: %d\n", WSAGetLastError());
                        closesocket(ClientSocket);
                        WSACleanup();
                        return 1;
                }

        } while (iResult > 0);

        // shutdown the connection since we're done
        iResult = shutdown(ClientSocket, SD_SEND);
        if (iResult == SOCKET_ERROR) {
                printf("shutdown failed with error: %d\n", WSAGetLastError());
                closesocket(ClientSocket);
                WSACleanup();
                return 1;
        }

        // cleanup
        closesocket(ClientSocket);
        free(arg);
```

```
}

void intHandler(int dummy) {
        printf("Exit by keyboard interrupt\n");
        exit(0);
}


int __cdecl main(void)
{
        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("WSAStartup failed with error: %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;

        // Получить адрес, порт
        iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
        if (iResult != 0) {
                printf("getaddrinfo failed with error: %d\n", iResult);
                WSACleanup();
                return 1;
        }

        // Создать сокет
        ListenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
        if (ListenSocket == INVALID_SOCKET) {
                printf("socket failed with error: %ld\n", WSAGetLastError());
                freeaddrinfo(result);
                WSACleanup();
                return 1;
        }
        printf("Socket successfully created\n");

        // Привязать сокет к интерфейсу
        iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen);
        if (iResult == SOCKET_ERROR) {
                printf("bind failed with error: %d\n", WSAGetLastError());
                freeaddrinfo(result);
                closesocket(ListenSocket);
                WSACleanup();
                return 1;
        }
```

```
        printf("\nSocket family: TCP\n");
        printf("Socket port: 60000\n");
        printf("Socket address: %s\n\n", "127.0.0.1");

        freeaddrinfo(result);
        while (1) {
                // Слушаем входящие соединения
                iResult = listen(ListenSocket, SOMAXCONN);
                if (iResult == SOCKET_ERROR) {
                        printf("listen failed with error: %d\n", WSAGetLastError());
                        closesocket(ListenSocket);
                        WSACleanup();
                        return 1;
                }

                // Разрешаем соединение
                SOCKET ClientSocket;
                (ClientSocket) = accept(ListenSocket, NULL, NULL);
                if (ClientSocket == INVALID_SOCKET) {
                        printf("accept failed with error: %d\n", WSAGetLastError());
                        closesocket(ListenSocket);
                        WSACleanup();
                        return 1;
                }
                HANDLE thread = CreateThread(NULL, 0, send_echo_adr, &ClientSocket, 0, NULL);
        }

        // Закрываем сокет
        closesocket(ListenSocket);

        WSACleanup();

        return 0;
}
```

Содержимое файла test.bat

```
@echo off
set loopcount=10000
:loop
start /b client < input.txt
set /a loopcount=loopcount-1
if %loopcount%==0 goto exitloop
goto loop
:exitloop
pause
```

Содержимое файла udp-client.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#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")
```

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

void intHandler(int dummy) {
        printf("Exit by keyboard interrupt\n");
        exit(0);
}

int main() {
        signal(SIGINT, intHandler);

        time_t t;
        srand((unsigned)time(&t));
        int data = rand();

        WSADATA wsaData = { 0 };
        int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

        // Создаём сокет
        int s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if (s == SOCKET_ERROR) {
                perror("Socket creating error\n");
                return 1;
        };
        printf("Socket successfully created\n");

        struct sockaddr_in addr;
        addr.sin_family = AF_INET;
        // Задаем порт
        unsigned short port = 0x4444;
        addr.sin_port = htons(port);
        // Задаем адрес сервера
        char address[] = "127.0.0.1";

        addr.sin_addr.s_addr = inet_addr(address);

        printf("\nSocket family: UDP\n");
        printf("Socket port: %hu\n", port);
        printf("Socket address: %s\n\n", address);

        printf("Press Enter to start transsmition...\n");
        getchar();
        getchar();
        printf("Sending identification packet to server\n");
        printf("Sending number: %d\n", data);
        int ret_val = sendto(
                s,
                (char *)(&data),
                sizeof(int), 0,
                (SOCKADDR *)& addr,
                sizeof(addr));
        if (ret_val == SOCKET_ERROR) {
                printf("\nsendto failed with error: %d\n", WSAGetLastError());
                closesocket(s);
                WSACleanup();
                return 1;
        }
        printf("Await echo packet\n");
        int echo_data;
        struct sockaddr_in sender_addr;
        int sender_addr_size = sizeof(sender_addr);
```

```
        ret_val = recvfrom(
                s,
                (char *)(&echo_data),
                sizeof(int), 0,
                (SOCKADDR*)&sender_addr,
                &sender_addr_size
        );
        if (ret_val == SOCKET_ERROR) {
                printf("\nrecvfrom failed with error: %d\n", WSAGetLastError());
                closesocket(s);
                WSACleanup();
                return 1;
        };
        printf("Echo number: %d\n", echo_data);
        return 0;
}
```

Содержимое файла udp-server.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#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 send_echo_adr(LPVOID arg) {
        printf("\nReceived packet from client\n");
        printf("Creating thread for response\n");
        printf("Received number: %d\n", ((thread_arg *)arg)->data);
        printf("Send echo packet\n");
        int s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if (s == SOCKET_ERROR) {
                perror("Socket creating error\n");
                return 1;
        };
        int ret_val = sendto(
                s,
                (char *)(&((thread_arg *)arg)->data),
                sizeof(int), 0,
                (SOCKADDR *)&((thread_arg *)arg)->adr,
                sizeof(((thread_arg *)arg)->adr)
        );
        if (ret_val == SOCKET_ERROR) {
                printf("sendto failed with error: %d\n", WSAGetLastError());
                closesocket(s);
                WSACleanup();
                return 1;
```

```
        }
        printf("Close thread\n\n");
        closesocket(s);
        return 0;
}

void intHandler(int dummy) {
        printf("Exit by keyboard interrupt\n");
        exit(0);
}

int main() {
        signal(SIGINT, intHandler);
        WSADATA wsaData = { 0 };
        int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

        // Создаём сокет
        int s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if (s == SOCKET_ERROR) {
                perror("Socket creating error\n");
                return 1;
        };
        printf("Socket successfully created\n");

        struct sockaddr_in addr;
        addr.sin_family = AF_INET;
        // Устанавливаем порт
        unsigned short port = 0x4444;
        addr.sin_port = htons(port);
        // Слушаем всю сеть
        addr.sin_addr.s_addr = htonl(INADDR_ANY);
        printf("\nSocket family: UDP\n");
        printf("Socket port: %hu\n", port);
        printf("Socket address: %s\n\n", inet_ntoa(addr.sin_addr));

        // Назначаем сокету адрес
        if (bind(s, (SOCKADDR *)&addr, sizeof(addr)) < 0) {
                printf("bind failed with error %d\n", WSAGetLastError());
                return 2;
        };

        printf("Server waiting client packet...\n");
        int rec_data;
        while (1) {
                struct sockaddr_in sender_addr;
                int sender_addr_size = sizeof(sender_addr);
                int ret_val = recvfrom(
                        s,
                        (char *)(&rec_data),
                        sizeof(int), 0,
                        (SOCKADDR*)&sender_addr,
                        &sender_addr_size
                );
                if (ret_val == SOCKET_ERROR) {
                        wprintf(L"recvfrom failed with error: %d\n", WSAGetLastError());
                        closesocket(s);
                        WSACleanup();
                        return 1;
                }
                else {
```

```
                        thread_arg arg;
                        arg.data = rec_data;
                        arg.adr = sender_addr;
                        HANDLE thread = CreateThread(NULL, 0, send_echo_adr, &arg, 0, NULL);
                };
        };

        return 0;
}
```