**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4

дисциплина: Сети ЭВМ и телекоммуникации

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

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Федотов Е.А.

Белгород 2021

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

**Задание к работе**

1. Разработать программу “Сервер”, которая принимает запросы от клиентов и посылает им в качестве ответа некоторое сообщение.

2. Разработать программу “Клиент”, 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.

**Краткие 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 избегания перегрузок. Чувствительные к перегрузкам UDP-приложения, которые потребляют значительную часть доступной пропускной способности, могут поставить под угрозу стабильность в Интернете.

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).

Поле контрольной суммы используется для проверки заголовка и данных на ошибки. Если сумма не сгенерирована передатчиком, то поле заполняется нулями. Поле является обязательным для IPv6.

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.

Но чаще такие механизмы не используются UDP-приложениями и даже мешают им. Потоковые медиа, многопользовательские игры в реальном времени и VoIP - примеры приложений, часто использующих протокол UDP. В этих конкретных приложениях потеря пакетов обычно не является большой проблемой. Если приложению необходим высокий уровень надежности, то можно использовать другой протокол.

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

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

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

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

**Параметры:** S – дескриптор несвязанного сокета.

Name – указатель на структуру sockaddr.

Namelen – размер структуры sockaddr.

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

**Назначение:** создает сокет.

**Параметры:** Af – семейство адресов. Был использован AF\_IPX.

Type – тип создаваемого сокета.

Protocol – используемый протокол.

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

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

**Функция:** Void WSACleanup();

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

**Функция**: 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\*.

**Выполнение работы**

**Задание 1**

Разработать программу “Сервер”, которая принимает запросы от клиентов и посылает им в качестве ответа некоторое сообщение.

**Содержание файла TCPserver:**

#include <WinSock2.h>

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

#include <sys/stat.h>

#include <iostream>

using namespace std;

#pragma warning(disable: 4996)

#define MAX\_DATA\_LEN 512

void TCPServer(){

char Buffer[MAX\_DATA\_LEN];

if (WSAStartup(0x0202, (WSADATA \*)Buffer)){

cout << "WSAStartup() error: " << WSAGetLastError() << endl;

return;

}

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

if (ServerSocket == INVALID\_SOCKET){

cout << "Socket() error: " << WSAGetLastError() << endl;

WSACleanup();

return;

}

struct hostent \*HostName = gethostbyname("");

SOCKADDR\_IN ServerAddress;

u\_short Port; // порт, который слушает сервер

cout << "Enter port: ";

cin >> Port;

memset(&ServerAddress, 0, sizeof(ServerAddress));

ServerAddress.sin\_family = AF\_INET;

ServerAddress.sin\_addr.s\_addr = \*(DWORD \*)HostName->h\_addr\_list[0];

ServerAddress.sin\_port = htons(Port);

cout << "Server IP: " << inet\_ntoa(ServerAddress.sin\_addr) << endl;

//привязывает адрес и порт к ранее созданному сокету

if (bind(ServerSocket, (PSOCKADDR)&ServerAddress, sizeof(ServerAddress)) == SOCKET\_ERROR){

cout << "Bind() error: " << WSAGetLastError() << endl;

closesocket(ServerSocket);

WSACleanup();

return;

}

if (listen(ServerSocket, 10)){

cout << "Listen() error: " << WSAGetLastError() << endl;

closesocket(ServerSocket);

WSACleanup();

return;

}

cout << "Waithing connect ..." << endl;

//извлекаем сообщения из очереди

int SizeFile = 0;

SOCKADDR\_IN ClientAddress;

SOCKET ClientSocket;

int AddressLength = sizeof(ClientAddress);

while (ClientSocket = accept(ServerSocket, (SOCKADDR \*)&ClientAddress, &AddressLength)){

recv(ClientSocket, (char \*)&SizeFile, sizeof(SizeFile), 0);

int CountPacket = SizeFile / MAX\_DATA\_LEN;

if (SizeFile % MAX\_DATA\_LEN){

CountPacket++;

}

char NameFile[MAX\_DATA\_LEN];

int ret = recv(ClientSocket, NameFile, MAX\_DATA\_LEN, 0);

NameFile[ret] = '\0';

int CountRecvPacket = 0;

char newName[50];

strcpy(newName, "Copy\_");

strcat(newName, NameFile);

FILE \*File = fopen(newName, "wb");

cout << "Count Packet: " << CountPacket << endl;

int begin = GetTickCount();

while (1 && (CountRecvPacket < CountPacket)){

//принимаем данные

int Result = recv(ClientSocket, Buffer, MAX\_DATA\_LEN, 0);

//считаем пакеты

CountRecvPacket++;

//cout << Result << endl;

fwrite(Buffer, 1, Result, File);

}

int ending = GetTickCount();

printf("Time: %d msec\n",ending-begin);

cout << "Count lost packets: " << CountPacket - CountRecvPacket << endl;

fclose(File);

closesocket(ClientSocket);

}

closesocket(ServerSocket);

if (WSACleanup())

cout << "WSACleanup() error: " << WSAGetLastError() << endl;

}

int main(){

TCPServer();

system("pause");

return 0;

}

**Содержание файла UDPserver:**

#include <WinSock2.h>

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

#include <wsipx.h>

#include <time.h>

#include <iostream>

using namespace std;

#pragma warning(disable: 4996)

#define MAX\_DATA\_LEN 512

void UDPServer(){

char Buffer[MAX\_DATA\_LEN];

WSADATA WSAData;

if (WSAStartup(0x0202, &WSAData)){

cout << "WSAStartup() error:" << WSAGetLastError() << endl;

return;

}

SOCKET ServerSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (ServerSocket == INVALID\_SOCKET){

cout << "Socket() error: " << WSAGetLastError() << endl;

}

struct hostent \*HostName = gethostbyname("");

SOCKADDR\_IN ServerAddress;

u\_short Port;

cout << "Enter port: "; cin >> Port;

memset(&ServerAddress, 0, sizeof(ServerAddress));

ServerAddress.sin\_family = AF\_INET;

ServerAddress.sin\_addr.S\_un.S\_addr = \*(DWORD \*)HostName->h\_addr\_list[0];

ServerAddress.sin\_port = htons(Port);

cout << "Server IP: " << inet\_ntoa(ServerAddress.sin\_addr) << endl;

if (bind(ServerSocket, (PSOCKADDR)&ServerAddress, sizeof(ServerAddress)) == SOCKET\_ERROR){

cout << "Bind() error: " << WSAGetLastError() << endl;

}

cout << "Waiting connect..." << endl;

int SizeFile = 0;

SOCKADDR\_IN ClientAddress;

int AddressSize = sizeof(ClientAddress);

int Value = 100000;

setsockopt(ServerSocket, SOL\_SOCKET, SO\_RCVTIMEO, (char \*)&Value, sizeof(Value));

int Result = recvfrom(ServerSocket, (char \*)&SizeFile, sizeof(SizeFile), 0,

(PSOCKADDR)&ClientAddress, &AddressSize);

if (Result == SOCKET\_ERROR){

cout << "recvfrom() error: " << WSAGetLastError() << endl;

return;

}

int CountPacket = SizeFile / MAX\_DATA\_LEN;

if (SizeFile % MAX\_DATA\_LEN)

CountPacket++;

char NameFile[MAX\_DATA\_LEN];

Result = sendto(ServerSocket, "test", strlen("test"), 0,

(PSOCKADDR)&ClientAddress, sizeof(ClientAddress));

if (Result == SOCKET\_ERROR){

cout << "Sendto() error: " << WSAGetLastError() << endl;

return;

}

Result = recvfrom(ServerSocket, NameFile, MAX\_DATA\_LEN, 0,

(PSOCKADDR)&ClientAddress, &AddressSize);

if (Result == SOCKET\_ERROR){

cout << "Recvfrom() error: " << WSAGetLastError() << endl;

return;

}

NameFile[Result] = '\0';

Result = sendto(ServerSocket, "test", strlen("test"), 0,

(PSOCKADDR)&ClientAddress, sizeof(ClientAddress));

if (Result == SOCKET\_ERROR){

cout << "Sendto() error: " << WSAGetLastError() << endl;

return;

}

int CountRecvPacket = 0;

Value = 3000;

setsockopt(ServerSocket, SOL\_SOCKET, SO\_RCVTIMEO, (char \*)&Value, sizeof(Value));

FILE\* File = fopen(NameFile, "wb");

cout << "Count Packet: " << CountPacket << endl;

clock\_t Time1 = clock();

while (true && (CountRecvPacket < CountPacket)){

//принимаем данные

Result = recvfrom(ServerSocket, Buffer, MAX\_DATA\_LEN, 0,

(PSOCKADDR)&ClientAddress, &AddressSize);

if (Result == SOCKET\_ERROR)

break;

CountRecvPacket++;

fwrite(Buffer, 1, Result, File);

}

cout << "Time: " << (double)(clock() - Time1) / CLOCKS\_PER\_SEC << endl;

cout << "Count lost packets: " << CountPacket - CountRecvPacket << endl;

fclose(File);

if (WSACleanup())

cout << "WSACleanup() error: " << WSAGetLastError() << endl;

}

int main(){

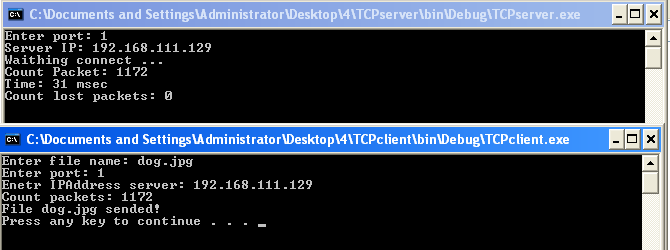
UDPServer();

system("pause");

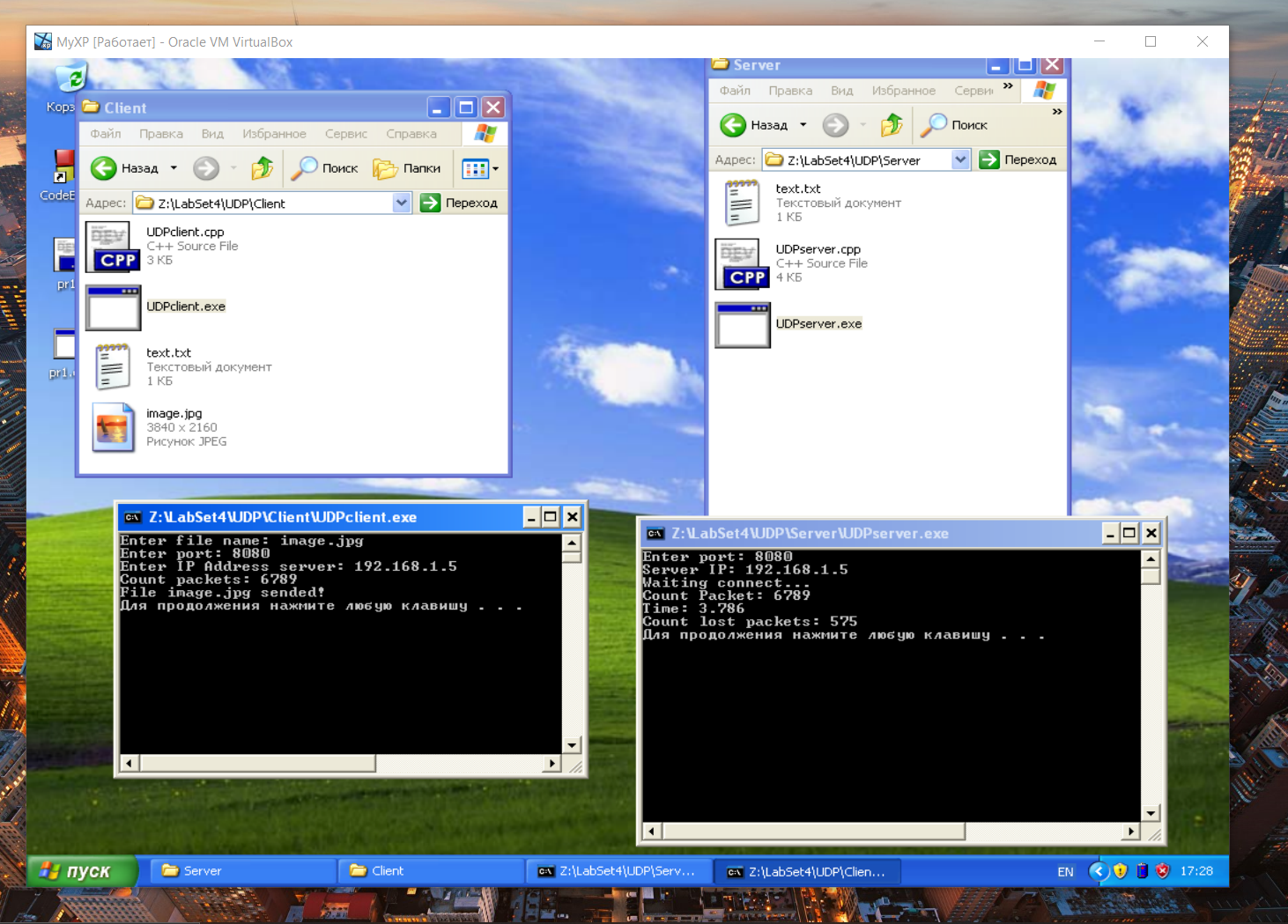
return 0;

}

Сервер TCP:



Сервер UDP:



**Задание 2**

Разработать программу “Клиент”, которая посылает запрос серверу и “ждет” от него ответного сообщения.

**Содержание файла TCPclient:**

#include <WinSock2.h>

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

#include <sys/stat.h>

#include <iostream>

using namespace std;

#pragma warning(disable: 4996)

#define MAX\_DATA\_LEN 512

int GetFileSize(const char \*InFileName){

struct stat FileStat;

stat(InFileName, &FileStat);

return FileStat.st\_size;

}

void TCPClient(){

char Buffer[MAX\_DATA\_LEN];

if (WSAStartup(0x0202, (WSADATA \*)Buffer)){

cout << "WSAtartup() error: " << WSAGetLastError() << endl;

return;

}

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

if (ClientSocket == INVALID\_SOCKET){

cout << "Socket() error: " << WSAGetLastError() << endl;

WSACleanup();

return;

}

char NameFile[MAX\_DATA\_LEN];

cout << "Enter file name: "; cin >> NameFile;

int SizeFile = 600000;

FILE \*File = fopen(NameFile, "rb");

if (File == NULL){

printf("open file error\n");

closesocket(ClientSocket);

WSACleanup();

return;

}

SOCKADDR\_IN ServerAddress;

u\_short Port = 0;

cout << "Enter port: "; cin >> Port;

char IPAddress[MAX\_DATA\_LEN];

cout << "Enetr IPAddress server: "; cin >> IPAddress;

memset(&ServerAddress, 0, sizeof(ServerAddress));

ServerAddress.sin\_family = AF\_INET;

ServerAddress.sin\_addr.s\_addr = inet\_addr(IPAddress);

ServerAddress.sin\_port = htons(Port);

if (connect(ClientSocket, (SOCKADDR \*)&ServerAddress, sizeof(ServerAddress))){

printf("connect() - %d\n", WSAGetLastError());

WSACleanup();

closesocket(ClientSocket);

}

//отправляем размер файла

send(ClientSocket, (char \*)&SizeFile, sizeof(SizeFile), 0);

recv(ClientSocket, Buffer, MAX\_DATA\_LEN, 0);

send(ClientSocket, NameFile, strlen(NameFile), 0);

recv(ClientSocket, Buffer, MAX\_DATA\_LEN, 0);

int Result;

int CountPacket = SizeFile / MAX\_DATA\_LEN;

if (SizeFile % MAX\_DATA\_LEN)

CountPacket++;

cout << "Count packets: " << CountPacket << endl;

while (!feof(File)){

//считываем с файла и отправляем

Result = fread((void \*)Buffer, 1, MAX\_DATA\_LEN, File);

//cout << result << endl;

send(ClientSocket, Buffer, Result, 0);

}

fclose(File);

cout << "File " << NameFile << " sended!" << endl;

closesocket(ClientSocket);

if (WSACleanup())

cout << "WSACleanup() error: " << WSAGetLastError() << endl;

}

int main(){

TCPClient();

system("pause");

return 0;

}

**Содержание файла UDPclient:**

#include <WinSock2.h>

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

#include <sys/stat.h>

#include <iostream>

using namespace std;

#pragma warning(disable: 4996)

#define MAX\_DATA\_LEN 512

int GetFileSize(const char \*InFileName){

struct stat FileStat;

stat(InFileName, &FileStat);

return FileStat.st\_size;

}

void UDPClient(){

WSADATA WSAData;

if (WSAStartup(0x0202, &WSAData)){

printf("WSAStartup() failed - %d\n", WSAGetLastError());

cout << "WSAStartup() error: " << WSAGetLastError() << endl;

return;

}

SOCKET ClientSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (ClientSocket == INVALID\_SOCKET){

cout << "Socket() error: " << WSAGetLastError() << endl;

return;

}

char NameFile[MAX\_DATA\_LEN];

cout << "Enter file name: "; cin >> NameFile;

FILE\* File = fopen(NameFile, "rb");

if (File == NULL){

cout << "Open file error" << endl;

return;

}

int SizeFile = GetFileSize(NameFile);

SOCKADDR\_IN ServerAddress;

u\_short Port = 0;

cout << "Enter port: "; cin >> Port;

char IPAddress[50];

cout << "Enter IP Address server: "; cin >> IPAddress;

memset(&ServerAddress, 0, sizeof(ServerAddress));

ServerAddress.sin\_family = AF\_INET;

ServerAddress.sin\_addr.S\_un.S\_addr = inet\_addr(IPAddress);

ServerAddress.sin\_port = htons(Port);

sendto(ClientSocket, (char \*)&SizeFile, sizeof(SizeFile), 0, (PSOCKADDR)&ServerAddress, sizeof(ServerAddress));

char chBuffer[MAX\_DATA\_LEN];

int val = 100000;

setsockopt(ClientSocket, SOL\_SOCKET, SO\_RCVTIMEO, (char \*)&val, sizeof(val));

recvfrom(ClientSocket, chBuffer, MAX\_DATA\_LEN, 0, (PSOCKADDR)&ServerAddress, nullptr);

sendto(ClientSocket, NameFile, strlen(NameFile), 0, (PSOCKADDR)&ServerAddress, sizeof(ServerAddress));

recvfrom(ClientSocket, chBuffer, MAX\_DATA\_LEN, 0, (PSOCKADDR)&ServerAddress, nullptr);

int Result;

int CountPacket = SizeFile / MAX\_DATA\_LEN;

if (SizeFile % MAX\_DATA\_LEN)

CountPacket++;

cout << "Count packets: " << CountPacket << endl;

while (!feof(File)){

Result = fread((void \*)chBuffer, 1, MAX\_DATA\_LEN, File);

sendto(ClientSocket, chBuffer, Result, 0, (PSOCKADDR)&ServerAddress, sizeof(ServerAddress));

}

fclose(File);

cout << "File " << NameFile << " sended!" << endl;

if (WSACleanup())

cout << "WSACleanup() error: " << WSAGetLastError() << endl;

}

int main(){

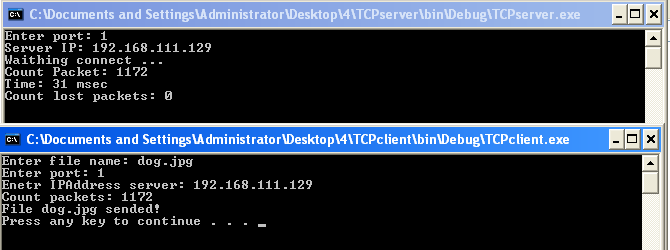
UDPClient();

system("pause");

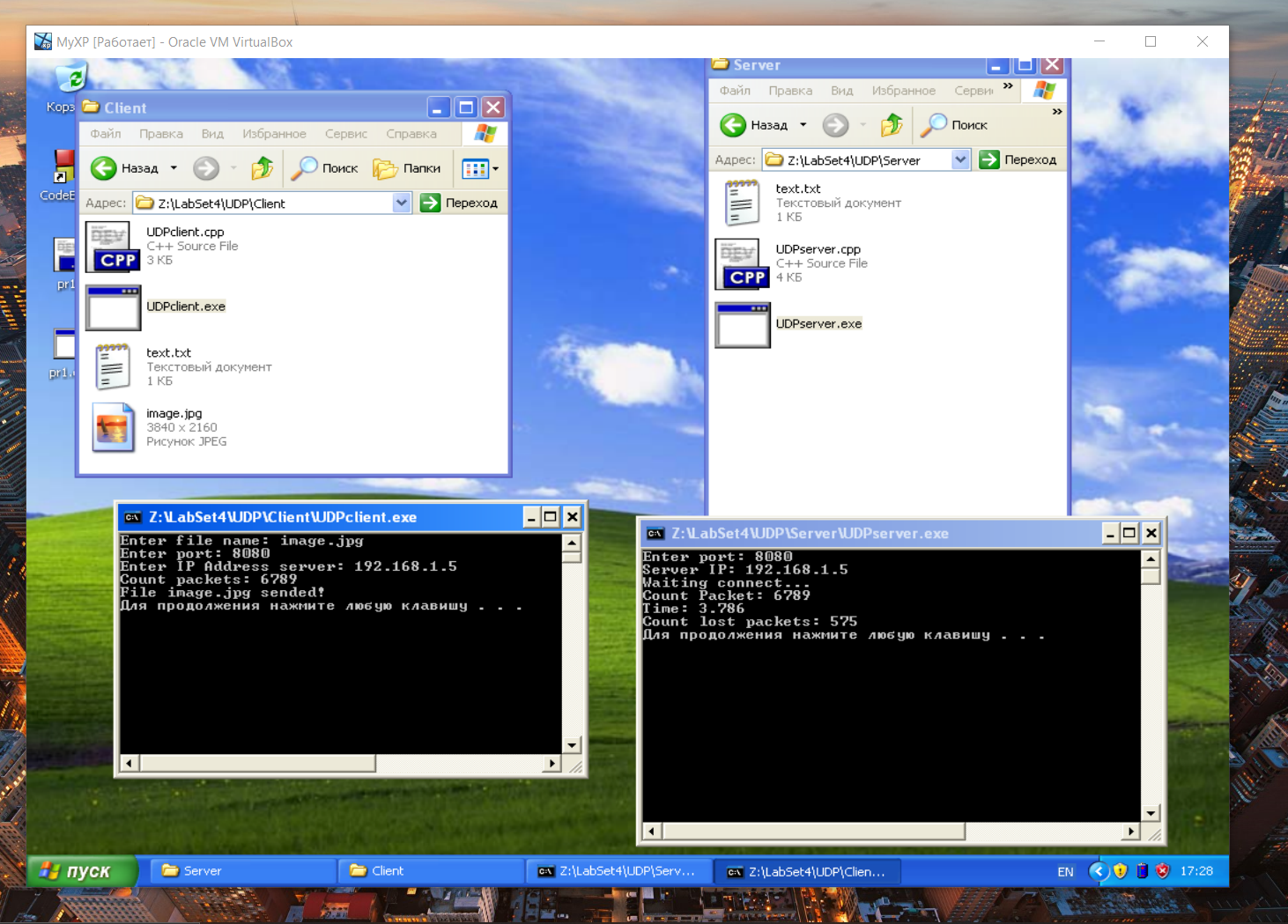
return 0;

}

Клиент TCP:



Клиент UDP:



**Задание 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.

|  |  |
| --- | --- |
| 6.2 мб | https://pinterest.ru.com/images/2018/11/24/MORE-KAMENISTYI-BEREG.jpg |

Размер файла: 6213147 байт

Количество пакетов: 12136

UDP:

|  |  |  |
| --- | --- | --- |
| Количество получателей | Время передачи, с | Число утерянных пакетов |
| 1 | 1,261 | 4660 |
| 2 | 2,353 | 2434 |
| 3 | 6,567 | 5656 |

TCP:

|  |  |  |
| --- | --- | --- |
| Количество получателей | Время передачи, с | Число утерянных пакетов |
| 1 | 1,338 | 0 |
| 2 | 2,572 | 0 |
| 3 | 7,409 | 0 |

**Задание 4**

Провести сравнительный анализ протоколов TCP и UDP. Сделать выводы.

**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.

**Ответы на контрольные вопросы**

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

TCP – это протокол транспортного уровня, который гарантирует доставку сообщения в правильном порядке. Работает поверх протокола сетевого уровня IP. TCP протокол для передачи данных использует соединение.

**2. Порядок установления TCP-соединения.**

Состоит из трех этапов.

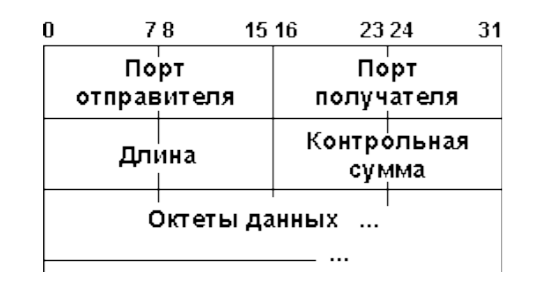
* Клиент посылает серверу фрагмент с номером последовательности и флагом синхронизации. Сервер получает эти данные и создает сокет для обслуживания клиента. В результате передает клиенту сообщение об успехе или провале данной операции.
* Если клиент получает сообщение о том, что операция была проведена 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).
* Сервер, получая от клиента сообщение о том, что тот перешел в состояние установленного соединения, сам сервер тоже переходит в это состояние.

**3. В чём состоит отличие протокола UDP от IP?**

Протокол UDP работает на транспортном уровне, тогда как IP работает на сетевом уровне. UDP обеспечивает многоканальную передачу (с помощью номеров портов) и проверку целостности (с помощью контрольных сумм) заголовка и существенных данных. на уровне IP для определения места доставки пакета используется адрес, на уровне UDP - номер порта.

**4. Формат заголовка пакета UDP.**

Выделяется по 16 бит на номер порта отправителя, номер порта получателя, длину и контрольную сумму. После заголовка располагаются данные.

****

**5. Опишите работу функций sendto и send библиотеки Winsocket.**

Для передачи данных по протоколу UDP используется функция **sendto** (SOCKET s, const char FAR \* buf, int len, int flags, const struct sockaddr FAR \* to, int tolen). Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Для передачи данных по протоколу TCP используется функция **send** (SOCKET s, const char FAR \* buf, int len, int flags), где s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError

**6. В каких случаях предпочтительней использовать протокол UDP?**

Протокол UDP гораздо быстрее протокола TCP, так как ему не нужно создавать соединение и разрывать его + не нужно ожидать подтверждения. Лучше всего протокол UDP использовать для передачи данных, в которых потеря пакетов не будет критичной, например, передача видеосигнала, передача информации в видео играх, так как важна скорость.