**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 №2

дисциплина: Сети ЭВМ и телекоммуникации

тема: «Программирование протоколов IPX/SPX с использованием

библиотеки Winsock»

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Федотов Е.А.

Белгород 2021

**Цель работы:** изучить протоколы IPX/SPX, основные функции библиотеки Winsock и разработать программу для приема/передачи пакетов.

**Теоритические сведения**

IPX (Internetwork Packet Exchange) является оригинальным протоколом сетевого уровня стека Novell, разработанным в начале 80-х годов на основе протокола Internetwork Datagram Protocol (IDM) компании Xerox [1, 3, 7].

Протокол IPX соответствует сетевому уровню модели OSI и поддерживает только дейтаграммный (без установления соединений) способ обмена сообщениями. В сети NetWare самая быстрая передача данных при наиболее экономном расходовании памяти реализуется именно протоколом IPX. Для надежной передачи пакетов используется протокол транспортного уровня SPX (Sequenced Packet Exchange), который работает с установлением соединения и восстанавливает пакеты при их потере или повреждении. Если по каким-то причинам пакет не дошел до получателя, выполняется его повторная передача. Следовательно, последовательность отправления совпадает с последовательностью получения пакетов. Обмен пакетами на уровне сеанса связи реализован с помощью протокола SPX, который построен на базе IPX.

SPX – протокол последовательного обмена пакетами (Sequenced Packet Exchange Protocol), разработанный Novell [6]. Система адресов протокола SPX аналогична системе адресов протокола IPX и также состоит из 3 частей: номера сети, адреса станции и сокета. Протокол SPX использует такой же блок ECB для передачи и приёма пакетов, что и протокол IPX. Однако, пакет, передаваемый при помощи протокола SPX, имеет более длинный заголовок. Дополнительно к 30 байтам стандартного заголовка пакета IPX добавляется еще 12 байт.



Поле **DestConnID** содержит номер канала связи принимающей стороны. Так как все пакеты приходят на один номер сокета и могут принадлежать разным каналам связи (на одном сокете можно открыть несколько каналов связи), необходимо классифицировать приходящие

пакеты по номеру канала связи.

Поле **SeqNumber** содержит счетчик пакетов, переданных по каналу в одном направлении. На каждой стороне канала используется свой счетчик. После достижения значения FFFFh счетчик сбрасывается в нуль, после чего процесс счета продолжается. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

Поле **AckNumber** содержит номер следующего пакета, который должен быть принят драйвером SPX. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его

значение.

Поле **AllocNumber** содержит количество буферов, распределенных программой для приема пакетов. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение [4].

**Библиотека Winsock**

**Windows Sockets API (WSA)** (сокр. **Winsock)** – техническая спецификация, которая определяет, как сетевое программное обеспечение Windows будет получать доступ к сетевым сервисам [3].

**Winsock** – это интерфейс сетевого программирования для Microsoft Windows. Winsock основывается на сокетной парадигме, изложенной в документах под названием Berkley System Distribution от University of California в Berkley.

Winsock включает в себя несколько стилей программирования. Первый – это стандартный однопотоковый стиль с блокированием потока определенными командами, второй – с использованием оконных процедур и третий – с использованием асинхронных процедур. Стандартная модель программирования от Berkley является de facto для сетей TCP/IP, но под Windows можно использовать эту библиотеку для программирования протоколов IPX/SPX.

**Описание функций**

Функция **WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData)** инициализирует библиотеку Winsock. Вслучае успеха возвращает 0. Дальше можно использовать любые

остальные функции этой библиотеки, иначе возвращает код возникшей ошибки. WwVersionRequested – это необходимая минимальная версия библиотеки, при присутствии которой приложение будет корректно работать. Младший байт содержит номер версии, а старший – номер ревизии. LpWSAData – структура, в которую возвращается информация по инициализированной библиотеке (статус, версия и т.д.).

Функция **WSAGetLastError (void)** возвращает код ошибки возникшей при выполнении последней операции. После работы с библиотекой, её необходимо выгрузить из памяти.

Функция **WSACleanup (void)** осуществляет очистку памяти, занимаемой библиотекой Winsock. Функция деинициализирует библиотеку Winsock и возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR. Расширенный код ошибки можно получить при помощи функции **WSAGetLastError.**

Функция **SOCKET (int af, int type, int protocol)** возвращает либо дескриптор созданного сокета, либо ошибку INVALID\_SOCKET. Расширенный код ошибки можно получить при помощи функции WSAGetLastError. Параметр af содержит сведения о семействе протоколов (AF\_INET, AF\_IPX). В данной лабораторной работе необходимо использовать константу AF\_IPX. Параметр type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе для IPX необходимо использовать константу SOCK\_DGRAM, а для SPX – константу SOCK\_SEQPACKET, которая означает, что пакеты будут отсылаться последовательно и в порядке очереди. Параметр protocol – протокол передачи данных. Для протокола IPX используется константа NSPROTO\_IPX, для SPX – NSPROTO\_SPX. Чтобы работать дальше с созданным сокетом его нужно привязать к какому-нибудь локальному адресу и порту.

Функция **bind (SOCKET s, const struct sockaddr FAR\* name, int namelen**). Привязывает к локальному адресу и порту. Здесь s – дескриптор сокета, который данная функция именует; name – указатель на структуру имени сокета; namelen – размер, в байтах, структуры name.

Функция **listen (SOCKET s, int backlog)** переводит сокет в

состояние “прослушивания” (для протокола SPX). Здесь s –

дескриптор сокета; backlog – это максимальный размер очереди

входящих сообщений на соединение.

Эта функция используется сервером, чтобы информировать ОС,

что он ожидает запросы связи на данном сокете. Без такой функции

всякое требование связи с этим сокетом будет отвергнуто [6].

Функция **connect (SOCKET s, const struct sockaddr FAR\* name, int namelen)** используется процессом-клиентом для установлениясвязи с сервером по протоколу SPX. В случае успешного установлениясоединения connect возвращает 0, иначе SOCKET\_ERROR и номерошибки можно получить при помощи функции WSAGetLastError.

Функция **accept (SOCKET s, struct sockaddr FAR\* addr, int FAR\* addrlen)** используется для принятия связи на сокет. Здесь s –дескриптор сокета; addr – указатель на структуру sockaddr; addrlen –размер структуры addr. Сокет должен быть уже слушающим в моментвызова функции. Если сервер устанавливает связь с клиентом, тоданная функция возвращает новый сокет дескриптор, через который ипроизводит общение клиента с сервером. Пока устанавливается связьклиента с сервером, функция блокирует другие запросы связи сданным сервером, а после установления связи “прослушивание”запросов возобновляется [8].

Функция **getsockname (SOCKET s, struct sockaddr FAR\* name, int FAR\***

**namelen).** Узнаётузнатькакой адрес и порт присвоен сокету Здесь s — дескриптор сокета; name — структура sockaddr, в которую система поместит данные; namelen — размер, в байтах, 22 структуры name. Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Передача данных по протоколу IPX осуществляется с помощью функции **sendto ( SOCKET s, const char FAR \* buf, int len, int flags,** **const struct sockaddr FAR \* to, int tolen).** Здесь 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; to - указатель на структуру sockaddr, которая содержит адрес сокета-приёмника; tolen - размер структуры to. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Передача данных по протоколу SPX осуществляется с помощью функции **send (SOCKET s, const char FAR \* buf, int len, int flags).** Здесь 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.

Прием данных по протоколу IPX осуществляется с помощью функции **recvfrom (SOCKET s, char FAR\* buf, int len, int flags, struct** **sockaddr FAR\* from, int FAR\* fromlen)**. Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Прием данных по протоколу SPX осуществляется с помощью функции **recv (SOCKET s, char FAR\* buf, int len, int flags).** Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно

получить при помощи функции WSAGetLastError.

Функция **closesocket(SOCKET s)** служит для закрытия сокета. Возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

**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.

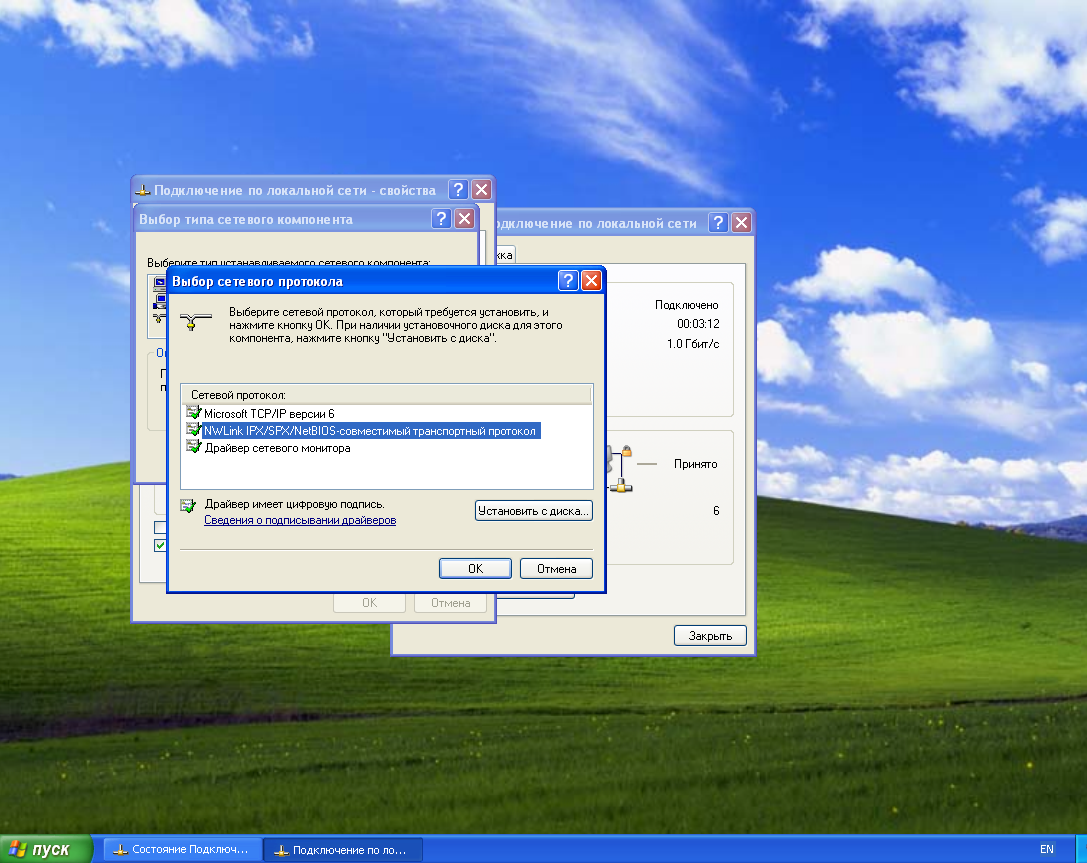
3. Провести анализ функционирования разработанных программ при передаче файла в формате \*.jpg размером не менее 20 Мб (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

4. Провести сравнительный анализ протоколов IPX и SPX. Сделать выводы.

**Выполнение работы**

Для начала установим widows xp и подключим реализацию протоколов IPX и SPX, потому что иначе наша программа будет выдавать ошибке и не будет работать.

Для этого мы в меню «Пуск» находим «Панель управления», а в ней «Сетевые подключения». После чего заходим в «Подключения по локальной сети». Далее устанавливаем новый компонент (Протокол).



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

**Код программ**

Server.c

#include <WinSock2.h>

#include <WSipx.h>

#include <sys/stat.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

const WORD ServerSocket = 0x5649;

const int MaxDataLength = 512;

int AddressLength = sizeof(SOCKADDR\_IPX);

void PrintIpxAddress(const char\* NetworkNumber, const char\* NodeNumber) {

    for (int i = 0; i < 4 ; ++i) {

        printf("%02X", (UCHAR)NetworkNumber[i]);

    }

    putchar('/');

    for (int i = 0; i < 6; ++i) {

        printf("%02X", (UCHAR)NodeNumber[i]);

    }

    putchar('\n');

}

int GetFileSize(const char\* FileName) {

    struct stat FileStat;

    stat(FileName, &FileStat);

    return FileStat.st\_size;

}

void IPXServer(int Socket) {

    /\*задаем параметры для сокета (сервера) для этого нам необходимо объявить структуру SOCKADDR\_IPX\*/

    SOCKADDR\_IPX SocketIPX, SocketIPXClient;

    memset(&SocketIPX, 0, sizeof(SOCKADDR\_IPX));

    /\*определяет используемый формат адреса (набор протоколов)\*/

    SocketIPX.sa\_family = AF\_IPX;

    SocketIPX.sa\_socket = htons(ServerSocket);

    if (bind(Socket, (PSOCKADDR)&SocketIPX, sizeof(SOCKADDR\_IPX))) {

        printf("bind() failed with error code %d\n", WSAGetLastError());

        return;

    }

    /\*Узнать адрес и порт сокета\*/

    getsockname(Socket, (PSOCKADDR)&SocketIPX, &AddressLength);

    puts("Waiting connect to client");

    char FileName[128];

    /\*Принять имя файла\*/

    int Result = recvfrom(Socket, FileName, 128, 0, (PSOCKADDR)&SocketIPXClient, &AddressLength);

    if (Result == SOCKET\_ERROR) {

        printf("recvfrom() failed with error code %d\n", WSAGetLastError());

        return;

    }

    FileName[Result] = '\0';

    int FileSize;

    /\*Принять размер файла\*/

    /\*recvfrom - получает дейтаграмму и сохраняет адрес источника.\*/

    Result = recvfrom(Socket, (char\*)&FileSize, sizeof(FileSize), 0, (PSOCKADDR)&SocketIPXClient, &AddressLength);

    if (Result == SOCKET\_ERROR) {

        printf("recvfrom() failed with error code %d\n", WSAGetLastError());

        return;

    }

    printf("Message Received from Client IPX Address: ");

    PrintIpxAddress(SocketIPXClient.sa\_netnum, SocketIPXClient.sa\_nodenum);

    printf("File name: %s\n", FileName);

    printf("File size: %d bytes\n", FileSize);

    int PacketCount = (FileSize + MaxDataLength - 1) / MaxDataLength;

    printf("Packet count: %d\n", PacketCount);

    /\*Установить максимальную задержку между приёмами 2-х пакетов в 3 секунды\*/

    int MaxDelay = 3000;

    setsockopt(Socket, SOL\_SOCKET, SO\_RCVTIMEO, (char\*)&MaxDelay, sizeof(MaxDelay));

    char Buffer[MaxDataLength];

    int ReceivedPacketCount = 0;

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

    if (File == NULL) {

        puts("Could not open/create file");

        return;

    }

    int begin = GetTickCount();

    for (int i = 0; i < PacketCount; ++i) {

        /\*Синхронизировать передачу (отправить данные)\*/

        Result = sendto(Socket, Buffer, 1, 0, (PSOCKADDR)&SocketIPXClient, sizeof(SOCKADDR\_IPX));

        if (Result == SOCKET\_ERROR) {

            printf("sendto() failed with error code %d\n", WSAGetLastError());

            break;

        }

        /\*Получить очередную порцию данных\*/

        Result = recvfrom(Socket, Buffer, MaxDataLength, 0, (PSOCKADDR)&SocketIPXClient, &AddressLength);

        if (Result == SOCKET\_ERROR) {

            printf("recvfrom() failed with error code %d\n", WSAGetLastError());

            break;

        }

        ReceivedPacketCount += 1;

        /\*То что в буфере записываем в файл\*/

        fwrite(Buffer, 1, Result, File);

    }

    int ending = GetTickCount();

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

    fclose(File);

    printf("Received packet count: %d\n", ReceivedPacketCount);

}

int main() {

    /\*Прежде чем воспользоваться функцией socket необходимо проинициализировать процесс библиотеки wsock32.dll вызвав функцию WSAStartup\*/

    WSADATA wsadata;

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

        printf("WSAStartup() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    /\*SOCK\_DGRAM - без установления соединени\*/

    int Socket = socket(AF\_IPX, SOCK\_DGRAM, NSPROTO\_IPX);

    if (Socket == INVALID\_SOCKET) {

        printf("socket() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    IPXServer(Socket);

    closesocket(Socket);

    if (WSACleanup()) {

        printf("WSACleanup() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    system("pause");

return 0;

}

SPXserver.cpp

#include <WinSock2.h>

#include <sys/stat.h>

#include <wsipx.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

const WORD ServerSocket = 0x5649;

const int MaxDataLength = 512;

int AddressLength = sizeof(SOCKADDR\_IPX);

int GetFileSize(const char\* FileName) {

    struct stat FileStat;

    stat(FileName, &FileStat);

    return FileStat.st\_size;

}

void SPXServer(SOCKET Socket) {

    SOCKADDR\_IPX SocketIPX, SocketIPXClient;

    memset(&SocketIPX, 0, AddressLength);

    SocketIPX.sa\_family = AF\_IPX;

    SocketIPX.sa\_socket = htons(ServerSocket);

    /\*Связать сокет с адресом сервера\*/

    if (bind(Socket, (PSOCKADDR)&SocketIPX, AddressLength) == SOCKET\_ERROR) {

        printf("bind() failed with error code %d\n", WSAGetLastError());

        return;

    }

    /\*Узнать адрес и порт сокета\*/

    getsockname(Socket, (PSOCKADDR)&SocketIPX, &AddressLength);

    puts("Waiting connect to client");

    char FileName[128];

    /\*Получить имя файла\*/

    int Result = recvfrom(Socket, (char\*)&FileName, sizeof(FileName), 0, (PSOCKADDR)&SocketIPXClient, &AddressLength);

    if (Result == SOCKET\_ERROR) {

        printf("recvfrom() failed with error code %d\n", WSAGetLastError());

        return;

    }

    FileName[Result] = '\0';

    char Buffer[MaxDataLength];

    /\*Отправить фиктивное сообщение, чтобы клиент мог узнать адрес сервера\*/

    Result = sendto(Socket, Buffer, 1, 0, (PSOCKADDR)&SocketIPXClient, sizeof(SOCKADDR\_IPX));

    if (Result == SOCKET\_ERROR) {

        printf("sendto() failed with error code %d\n", WSAGetLastError());

        return;

    }

    /\*Пересоздать сокет для передачи данных с помощью протокола SPX\*/

    Socket = socket(AF\_IPX, SOCK\_SEQPACKET, NSPROTO\_SPX);

    memset(&SocketIPX, 0, AddressLength);

    SocketIPX.sa\_family = AF\_IPX;

    SocketIPX.sa\_socket = htons(ServerSocket);

    if (bind(Socket, (PSOCKADDR)&SocketIPX, AddressLength) == SOCKET\_ERROR) {

        printf("bind() failed with error code %d\n", WSAGetLastError());

        return;

    }

    /\*Ожидать пока клиент не попробует подключиться к серверу\*/

    if (listen(Socket, 3)) {

        printf("listen() failed with error code %d\n", WSAGetLastError());

        return;

    }

    /\*Создать сокет для общения с клиентом\*/

    SOCKET ClientSocket = accept(Socket, (PSOCKADDR)&SocketIPXClient, &AddressLength);

    if (ClientSocket == INVALID\_SOCKET) {

        printf("accept() failed with error code %d\n", WSAGetLastError());

        return;

    }

    /\*Получить размер файла\*/

    int FileSize;

    Result = recv(ClientSocket, (char\*)&FileSize, sizeof(FileSize), 0);

    if (Result == SOCKET\_ERROR) {

        printf("recv() failed with error code %d\n", WSAGetLastError());

        return;

    }

    printf("File size: %d bytes\n", FileSize);

    int PacketCount = (FileSize + MaxDataLength - 1) / MaxDataLength;

    printf("Packet count: %d\n", PacketCount);

    int ReceivedPacketCount = 0;

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

    if (File == NULL) {

        puts("Could not open/create file");

        return;

    }

    /\*Установить максимальную задержку, между приёмами 2-х пакетов, в 3 секунды\*/

    int MaxDelay = 3000;

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

    int begin = GetTickCount();

    for (int i = 0; i < PacketCount; ++i) {

        /\*Синхронизировать приём\*/

        int Result = send(ClientSocket, Buffer, 1, 0);

        if (Result == SOCKET\_ERROR) {

            printf("send() failed with error code %d\n", WSAGetLastError());

            break;

        }

        /\*Получить очередную порцию даных\*/

        Result = recv(ClientSocket, Buffer, MaxDataLength, 0);

        if (Result == SOCKET\_ERROR) {

            printf("recv() failed with error code %d\n", WSAGetLastError());

            break;

        }

        ReceivedPacketCount++;

        fwrite(Buffer, 1, Result, File);

    }

    int ending = GetTickCount();

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

    printf("Received packet count: %d\n", ReceivedPacketCount);

    fclose(File);

    closesocket(ClientSocket);

}

int main() {

    WSADATA wsadata;

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

        printf("WSAStartup() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    int Socket = socket(AF\_IPX, SOCK\_DGRAM, NSPROTO\_IPX);

    if (Socket == INVALID\_SOCKET) {

        printf("socket() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    SPXServer(Socket);

    closesocket(Socket);

    if (WSACleanup()) {

        printf("WSACleanup() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

return 0;

}

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

Client.c

#include <WinSock2.h>

#include <WSipx.h>

#include <sys/stat.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

const WORD ServerSocket = 0x5649;

const int MaxDataLength = 512;

int AddressLength = sizeof(SOCKADDR\_IPX);

int GetFileSize(const char\* FileName) {

    struct stat FileStat;

    stat(FileName, &FileStat);

    return FileStat.st\_size;

}

void IPXClient(int Socket) {

    SOCKADDR\_IPX SocketIPX;

    /\*Указываем семейство протоколов \*/

    SocketIPX.sa\_family = AF\_IPX;

    /\*Преобразует байты, чтобы библиотека Winsock работа корректно\*/

    SocketIPX.sa\_socket = htons(ServerSocket);

    /\*Широковещательная рассылка\*/

    memset(SocketIPX.sa\_netnum, 0, sizeof(SocketIPX.sa\_netnum)); // номер сети

    memset(SocketIPX.sa\_nodenum, 0xFF, sizeof(SocketIPX.sa\_nodenum)); // номер узла

    int Broadcast = 1;

    /\*Установка флагов\*/

    setsockopt(Socket, SOL\_SOCKET, SO\_BROADCAST, (char\*)&Broadcast, sizeof(Broadcast));

    char FileName[128];

    printf("Enter file name: "); scanf("%s", FileName);

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

    if (File == NULL) {

        puts("Could not open file");

        return;

    }

    int Result;

    /\*Отправить имя файла\*/

    Result = sendto(Socket, FileName, strlen(FileName), 0, (PSOCKADDR)&SocketIPX, sizeof(SocketIPX));

    if (Result == SOCKET\_ERROR) {

        printf("sendto() failed with error code %d\n", WSAGetLastError());

        fclose(File);

        return;

    }

    int FileSize = GetFileSize(FileName);

    printf("File size: %d bytes\n", FileSize);

    /\*Отправить размер файла\*/

    Result = sendto(

        Socket, (char\*)&FileSize, sizeof(FileSize), 0, (PSOCKADDR)&SocketIPX, sizeof(SocketIPX)

    );

    if (Result == SOCKET\_ERROR) {

        printf("sendto() failed with error code %d\n", WSAGetLastError());

        fclose(File);

        return;

    }

    SOCKADDR\_IPX SocketIPXServer;

    char Buffer[MaxDataLength];

    int PacketCount = (FileSize + MaxDataLength - 1) / MaxDataLength;

    printf("Packet count: %d\n", PacketCount);

    for (int i = 0; i < PacketCount; ++i) {

        /\*Синхронизировать передачу

        /recvfrom - приём данных по протоколу IPX\*/

        Result = recvfrom(Socket, Buffer, MaxDataLength, 0, (PSOCKADDR)&SocketIPXServer, &AddressLength);

        if (Result == SOCKET\_ERROR) {

            printf("recvfrom() failed with error code %d\n", WSAGetLastError());

            fclose(File);

            return;

        }

        fread(Buffer, 1, MaxDataLength, File);

        /\*Отправить очередную порцию данных\*/

        Result = sendto(Socket, Buffer, MaxDataLength, 0, (PSOCKADDR)&SocketIPXServer, sizeof(SocketIPXServer));

        if (Result == SOCKET\_ERROR) {

            printf("sendto() failed with error code %d\n", WSAGetLastError());

            fclose(File);

            return;

        }

    }

    fclose(File);

    printf("File %s sended\n", FileName);

}

int main() {

    WSADATA wsadata;

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

        printf("WSAStartup() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    /\* Возвращает дескриптор созданного сокета \*/

    int Socket = socket(AF\_IPX, SOCK\_DGRAM, NSPROTO\_IPX);

    if (Socket == INVALID\_SOCKET) {

        printf("socket() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    IPXClient(Socket);

    closesocket(Socket);

    if (WSACleanup()) {

        printf("WSACleanup() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    system("pause");

    return 0;

}

SPXclient.cpp

#include <WinSock2.h>

#include <sys/stat.h>

#include <wsipx.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

const WORD ServerSocket = 0x5649;

const int MaxDataLength = 512;

int AddressLength = sizeof(SOCKADDR\_IPX);

int GetFileSize(const char\* FileName) {

    struct stat FileStat;

    stat(FileName, &FileStat);

    return FileStat.st\_size;

}

void SPXClient(SOCKET Socket) {

    SOCKADDR\_IPX SocketIPX;

    SocketIPX.sa\_family = AF\_IPX;

    SocketIPX.sa\_socket = htons(ServerSocket);

    /\*Широковещательная рассылка\*/

    memset(SocketIPX.sa\_netnum, 0, sizeof(SocketIPX.sa\_netnum));

    memset(SocketIPX.sa\_nodenum, 0xFF, sizeof(SocketIPX.sa\_nodenum));

    int Broadcast = 1;

    setsockopt(Socket, SOL\_SOCKET, SO\_BROADCAST, (char\*)&Broadcast, sizeof(Broadcast));

    char FileName[128];

    printf("Enter file name: "); gets(FileName);

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

    if (File == NULL) {

        puts("Could not open file");

        return;

    }

    /\*Отправить имя файла\*/

    sendto(Socket, FileName, strlen(FileName), 0, (PSOCKADDR)&SocketIPX, sizeof(SocketIPX));

    SOCKADDR\_IPX SocketIPXServer;

    char Buffer[MaxDataLength];

    /\*Принять фиктивное сообщени, чтобы узнать адрес сервера\*/

    recvfrom(Socket, Buffer, MaxDataLength, 0, (PSOCKADDR)&SocketIPXServer, &AddressLength);

    /\*Пересоздать сокет для передачи данных с помощью протокола SPX\*/

    Socket = socket(AF\_IPX, SOCK\_SEQPACKET, NSPROTO\_SPX);

    if (Socket == INVALID\_SOCKET) {

       printf("socket() failed with error code %d\n", WSAGetLastError());

        fclose(File);

        return;

    }

    /\*Попытаться соединиться с сервером\*/

    if (connect(Socket, (PSOCKADDR)&SocketIPXServer, sizeof(SocketIPXServer))) {

        printf("connect() failed with error code %d\n", WSAGetLastError());

       fclose(File);

       return;

    }

    int FileSize = GetFileSize(FileName);

    printf("File size: %d bytes\n", FileSize);

    /\*Отправить размер файла\*/

    send(Socket, (char\*)&FileSize, sizeof(FileSize), 0);

    int PacketCount = (FileSize + MaxDataLength - 1) / MaxDataLength;

    printf("Packet count: %d\n", PacketCount);

    for (int i = 0; i < PacketCount; ++i) {

        /\*Синхронизировать передачу\*/

        int Result = recv(Socket, Buffer, MaxDataLength, 0);

        if (Result == SOCKET\_ERROR) {

            printf("recv() failed with error code %d\n", WSAGetLastError());

            fclose(File);

            return;

        }

        int ByteCount = fread((void \*)Buffer, 1, MaxDataLength, File);

        /\*Передачать очередную порцию данных\*/

        Result = send(Socket, Buffer, ByteCount, 0);

        if (Result == SOCKET\_ERROR) {

            printf("send() failed with error code %d\n", WSAGetLastError());

            fclose(File);

            return;

        }

    }

    fclose(File);

    printf("File %s sended\n", FileName);

}

int main() {

    WSADATA wsadata;

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

        printf("WSAStartup() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    int Socket = socket(AF\_IPX, SOCK\_DGRAM, NSPROTO\_IPX);

    if (Socket == INVALID\_SOCKET) {

        printf("socket() failed with error code %d\n", WSAGetLastError());

        return 1;

    }

    SPXClient(Socket);

    closesocket(Socket);

    if (WSACleanup()) {

        printf("WSACleanup() failed with error code %d\n", WSAGetLastError());

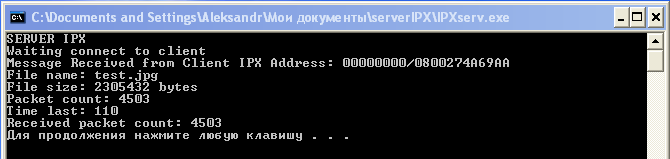
        return 1;

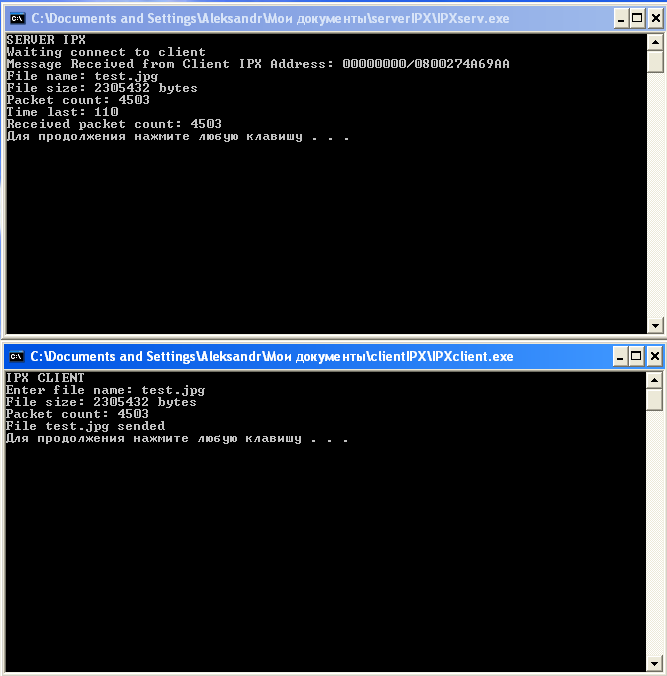
    }

}

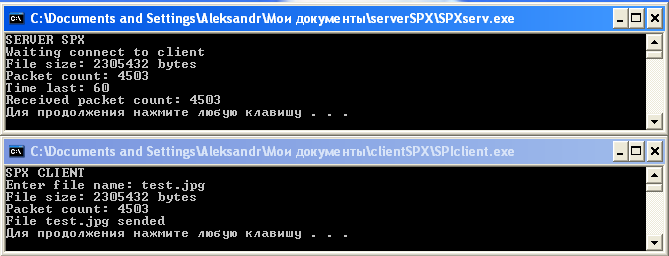
**Результат работы программы**

Сервер и клиент IPX





Сервер и клиент SPX





**Задание 3**. Провести анализ функционирования разработанных программ при передаче файла в формате \*.jpg размером не менее 20 Мб (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

**Анализ функционирования разработанных программ**

|  |  |
| --- | --- |
| 6.2 мб | C:\Users\500a5\Desktop\starry-sky-night-glitter-stars-river-1200080.jpg |

Размер файла: 6479120 байт

Количество пакетов: 13048

IPX:

|  |  |  |  |
| --- | --- | --- | --- |
| Получатели | Время передачи, мс | Число утерянных пакетов, шт | Процент потерь |
| 1 | 1103 | 2 | 0.016% |
| 2 | 5359  5472 | 3  2 | 0.025%  0.016% |
| 3 | 4678  3621  3634 | 7  8  7 | 0.058%  0.067%  0.058% |

SPX:

|  |  |  |  |
| --- | --- | --- | --- |
| Получатели | Время передачи, мс | Число утерянных пакетов, шт | Процент потерь |
| 1 | 1636 | 0 | 0.0% |
| 2 | 3428  3460 | 0  0 | 0.0%  0.0% |
| 3 | 4664  4670  4643 | 0  0  0 | 0.0%  0.0%  0.0% |

Можно заметить, как с увеличением кол-ва получателей растет процент потерь при передаче данных через протокол IPX

При этом SPX обеспечивает 100%-ую передачу пакетов в независимости от того, сколько клиентов пытается передать файл.

**Задание 4**. Провести сравнительный анализ протоколов IPX и SPX. Сделать выводы.

1) IPX – это протокол сетевого уровня, а SPX – транспортного

2) IPX не устанавливает соединение между узлами, а SPX тратит время на установление соединения

3) IPX не гарантирует доставки пакетов получателю, а SPX отправляет пакеты до тех пор, пока они не будут приняты получателем

4) IPX принимает пакеты в случайном порядке, а SPX – строго последовательно

5) Заголовок SPX длиннее заголовка IPX, однако оба протокола используют для приема и передачи пакетов одинаковый блок ECB

**Выводы**

В процессе выполнения лаб. работы были получены навыки работы с библиотекой Winsock, протоколом SPX. Закреплены навыки работы с виртуальными машинами и протоколом IPX.

Протокол IPX может работать быстрее протокола SPX за счет того, что он не подразумевает установления соединения между узлами и повторной передачи пакетов в случае их повреждения или потери. Однако протокол SPX обеспечивает большую надежность передачи данных.

**Ответы на вопросы**

**1. Назовите отличия протокола SPX от IPX.**

IPX – это протокол сетевого уровня, который не гарантирует доставку пакета и не использует постоянное соединение между программами.

SPX – это протокол транспортного уровня, что означает, что данный протокол обеспечивает гарантированную доставку пакета.

**2. Что представляет собой библиотека Winsock?**

**Windows Sockets API (WSA) (сокр. Winsock)** – техническая спецификация, которая определяет, как сетевое программное обеспечение Windows будет получать доступ к сетевым сервисам.

**Winsock** – это интерфейс сетевого программирования для Microsoft Windows. Winsock основывается на сокетной парадигме, изложенной в документах под названием Berkley System Distribution от University of California в Berkley.

**3. Какие действия необходимо выполнить для корректного создания сокета, настроенного на приём сообщений?**

После создания сокета (socket) с необходимыми настройками нужно привязать сокет к конкретному адресу и порту (bind). К сокету может подключаться сразу несколько устройств, поэтому нужно выставить размер очереди (listen). После этого сокет готов пассивн ожидать подключения (accept).

**4. Назовите функции библиотеки WInsocket, используемые для отправки и приёма сообщений через протокол IPX.**

Передача данных по протоколу IPX осуществляется с помощью функции **sendto**

Прием данных по протоколу IPX осуществляется с помощью функции **recvfrom**

**5. Что представляет собой структура sockaddr?**

В сетях IPX/SPX структура sockaddr имеет вид:

typedef struct sockaddr\_ipx {

short sa\_family;

char sa\_netnum[4];

char sa\_nodenum[6];

unsigned short sa\_socket;

} SOCKADDR\_IPX, \*PSOCKADDR\_IPX,FAR \*LPSOCKADDR\_IPX;

Параметры структуры:

sin\_family — семейство протоколов.

sin\_netnum — номер сети.

sin\_nodenum — номер узла

sa\_socket – номер сокета

**6. Принцип построения программы «клиент-сервер» с использованием протокола SPX**

Система «клиент – сервер» с использованием протокола SPX должна работать по следующему принципу: сервер при запуске демонстрирует пользователю полный сетевой адрес и создаёт поток, вызывающий функцию ожидания подключений. В свою очередь, данный поток для каждого подключившегося клиента создаёт отдельный поток, работающий с ним. Таким образом, сервер способен обслуживать несколько клиентов в параллельном режиме. Процесс-клиент должен требовать ввода сетевого адреса сервера и имени требуемого файла. Процесс-клиент отправляет имя файла серверу и ждёт ответа. Сервер передаёт файл клиенту, распределяя его по пакетам. Следует обратить внимание на правильность указания адреса сервера в функции connect при использовании протокола SPX. Тот адрес, который возвращает функция getsockname подходит только для работы приложений на одном компьютере. Для того чтобы сервер и клиент могли обмениваться сообщениями непосредственно по сети с использованием протокола SPX, необходим другой адрес. Чтобы получить этот адрес можно использовать протокол IPX: клиент отправляет широковещательное сообщение; сервер посылает ответ, из которого на стороне клиента можно извлечь адрес, пригодный для последующего сетевого обмена сообщениями по протоколу SPX.