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 №2**

дисциплина «Сети ЭВМ и телекоммуникации»

тема: «Программирование протоколов IPX/SPX с использованием

библиотеки Winsock»

Выполнил: студент группы ВТ-31

Ковалёв И. Д.

Проверил: Федотов Е. А.

Белгород 2020

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

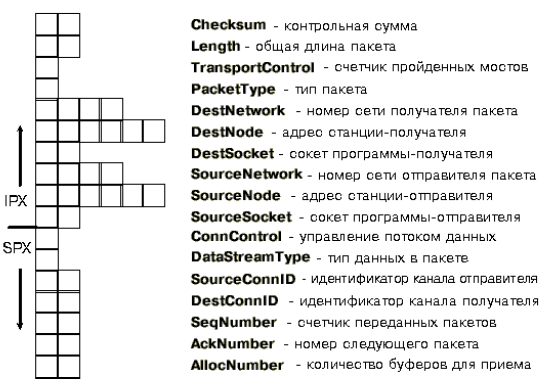
**Теоретические сведения**

Протокол 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.

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 ConnControl представляет собой как набор битовых флагов, управляющих передачей данных по каналу SPX.

Поле DataStreamType состоит из однобитовых флагов, которые используются для классификации данных, передаваемых или принимаемых при помощи протокола SPX.

Поле SourceConnID содержит номер канала связи передающей программы, присвоенный драйвером SPX при создании канала связи. Этот номер должен указываться функции передачи пакета средствами SPX.

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.

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.

**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.

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

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

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

**WSACleanup();**

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

**int WSAAPI connect(SOCKET s, const sockaddr \*name, int namelen);**

**Назначение:** устанавливает соединение с сокетом. В случае успеха возвращает 0, иначе – код ошибки.

S – дескриптор, который идентифицируется как неподключенный сокет.

Name – указатель на структуру sockaddr, которая содержит данные о подключаемом сокете.

Namelen – размер структуры sockaddr.

**int WSAAPI send(SOCKET s, const char \*buf, int len, int flags);**

**Назначение:** отправляет данные на подключенный сокет. В случае, если в процессе работы ошибок не возникло, вернет общее число отправленных байтов, иначе вернет SOCKET\_ERROR.

S – дескриптор, который идентифицируется как подключенный сокет.

Buf – указатель на буфер, который содержит передаваемые данные.

Len – размер буфера buf.

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.

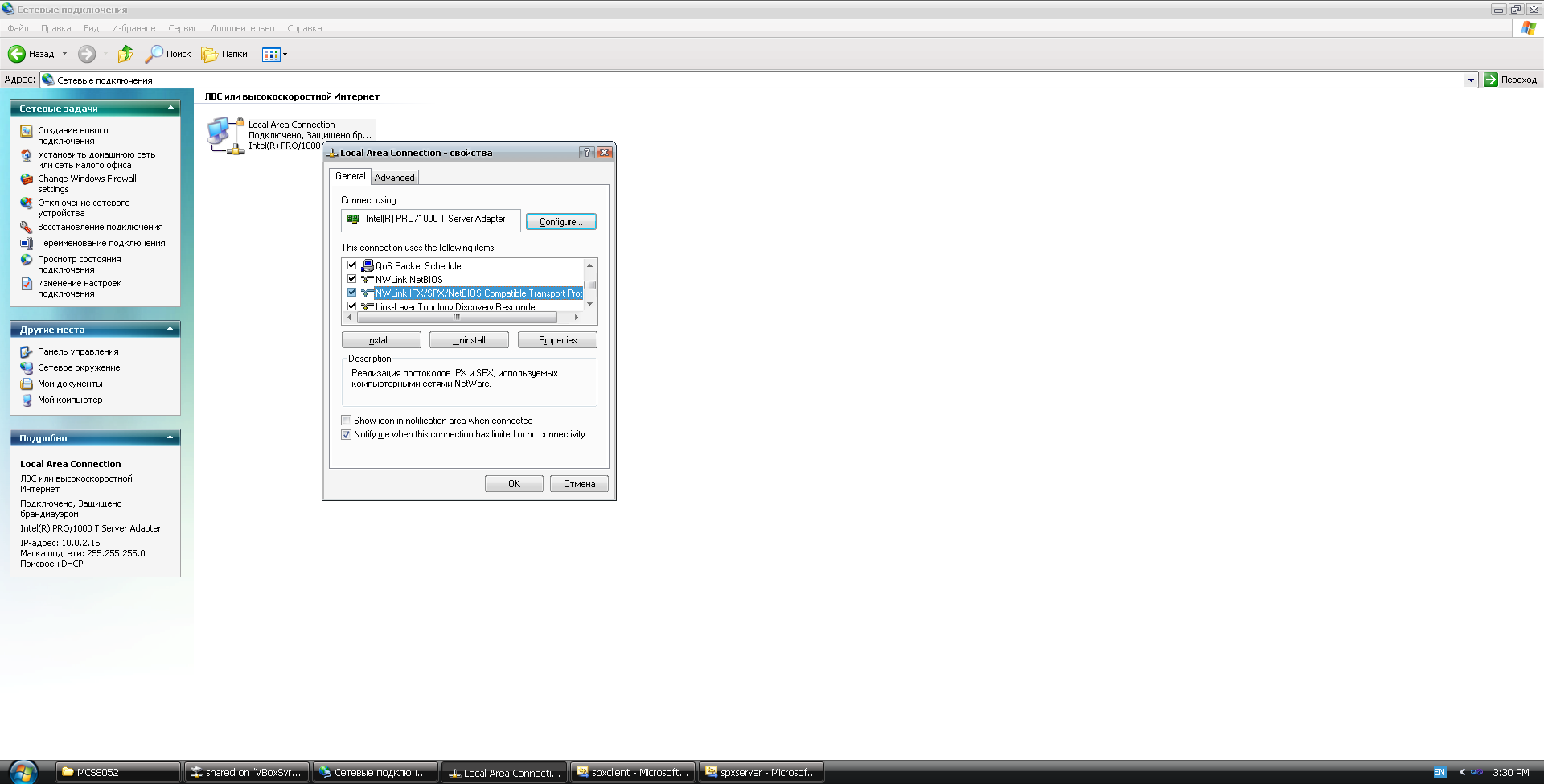
S – дескриптор несвязанного сокета.

Name – указатель на структуру sockaddr.

Namelen – размер структуры sockaddr.

**Выполнение работы**

В процессе выполнения работы была использована ОС Windows XP с установленной реализацией протоколов IPX и SPX.



В процессе функционирования разработанных программ были исследованы особенности передачи данных с помощью вышеуказанных протоколов.

Передавался файл размером 16091417 байт, который был разбит на 10969 пакетов (1 неполный).

**IPX:**

|  |  |
| --- | --- |
| Число получателей | Число утерянных пакетов |
| 1 | 0 |
| 2 | 7, 3 |
| 3 | 184, 183, 183 |

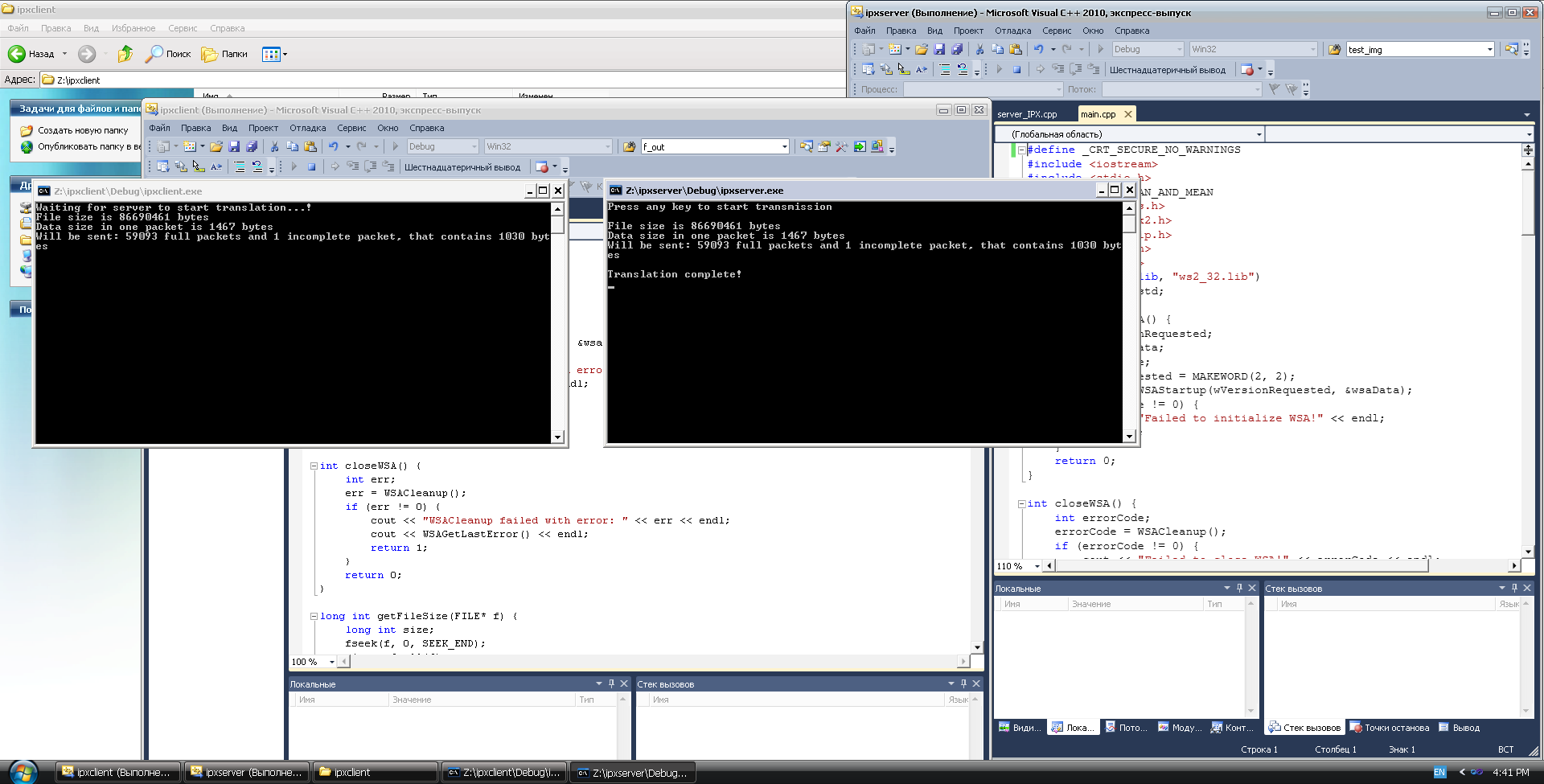
**SPX:**

|  |  |
| --- | --- |
| Число получателей | Число утерянных пакетов |
| 1 | 0 |
| 2 | 0, 0 |
| 3 | 0, 0, 0 |

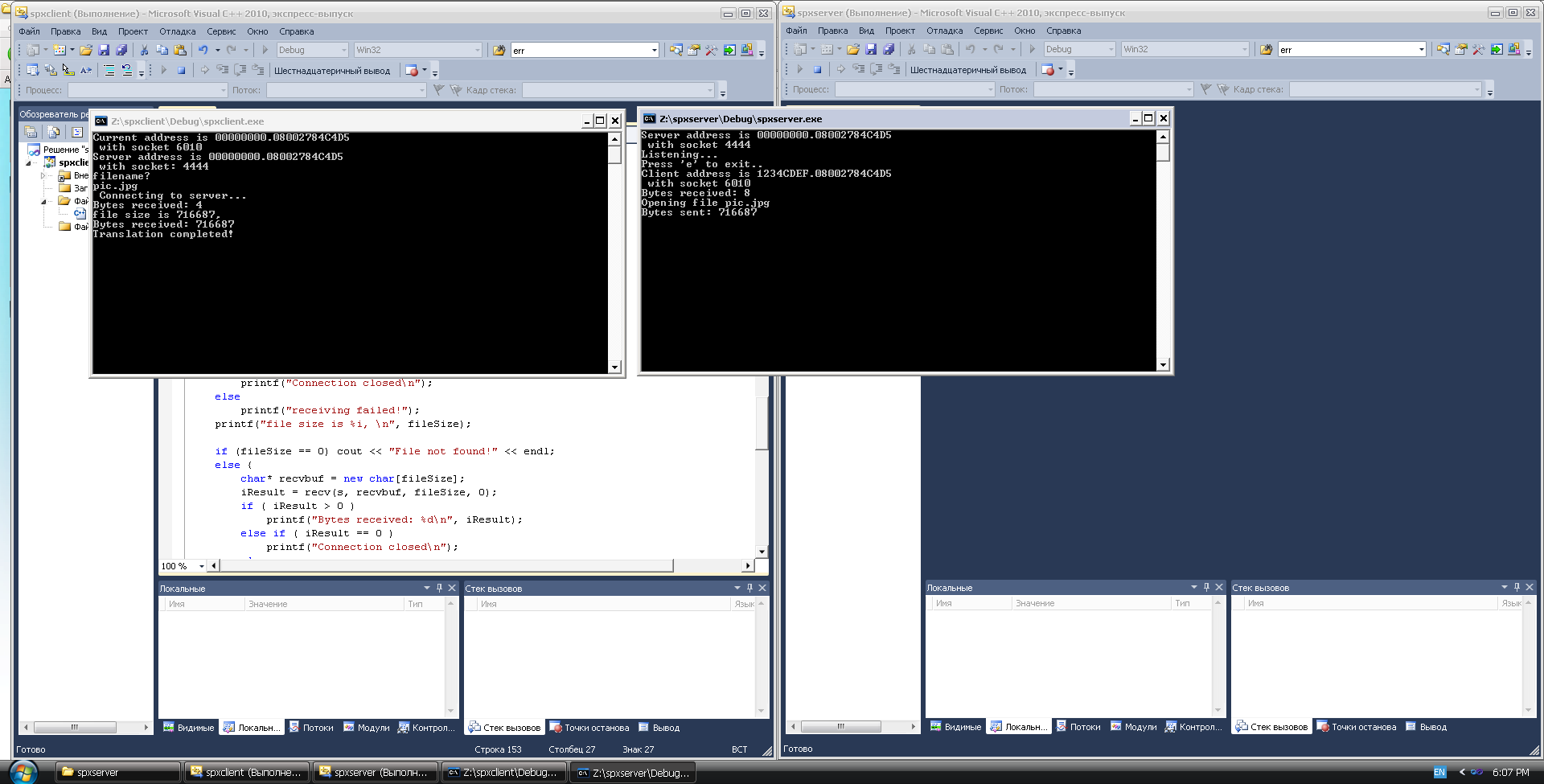
**Вывод:** согласно результатам проведенных экспериментов, можно прийти к выводу, что протокол SPX является более эффективным, так как гарантирует доставку пакетов без потерь. Следует заметить, что в процессе выполнения лабораторной работы было выявлено, что используя протокол SPX гарантируется большая скорость передачи данных, но только в случае, если передача осуществляется небольшому количеству клиентов (для 3 клиентов время отправки увеличилось вдвое относительно времени, затраченному на отправку 1 клиенту). Для протокола IPX разница во времени меньше (время передачи для трех клиентов больше всего лишь в полтора раза, чем для одного), но при этом общее время передачи относительно SPX в ~5 раз больше.

**Скриншоты разработанных программ**

IPX клиент / сервер:



SPX клиент / сервер:



**Листинг программ:**

1. **Spxclient:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <wsipx.h>

#include <string>

#include <stdlib.h>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

//

int initializeWSA() {

    WORD wVersionRequested;

    WSADATA wsaData;

    int errorCode;

    wVersionRequested = MAKEWORD(2, 2);

    errorCode = WSAStartup(wVersionRequested, &wsaData);

    if (errorCode != 0) {

        cout << "Failed to initialize WSA!" << endl;

        return 1;

    }

    return 0;

}

int closeWSA() {

    int errorCode;

    errorCode = WSACleanup();

    if (errorCode != 0) {

        cout << "Failed to close WSA!" << endl;

        return 1;

    }

    return 0;

}

void PrintIpxAddress(char \*lpsNetnum, char \*lpsNodenum){

    int i;

    for (i=0; i < 4 ;i++){

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

    }

    printf(".");

    for (i=0; i < 6 ;i++){

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

    }

}

void ReadIpxAddress(char \*lpsNetnum, char \*lpsNodenum){

    char buffer[3];

    int i;

    for (i=0; i < 4 ;i++){

        scanf("%2s", buffer);

        sscanf(buffer, "%X", (UCHAR \*)&(lpsNetnum[i]));

    }

    scanf("%\*c");

    for (i=0; i < 6 ;i++){

        scanf("%2s", buffer);

        sscanf(buffer, "%X", (UCHAR \*)&(lpsNodenum[i]));

    }

}

long int getFileSize(FILE\* f) {

    long int size;

    fseek(f, 0, SEEK\_END);

    size = ftell(f);

    fseek(f, 0, SEEK\_SET);

    return size;

};

void receiveFile(SOCKET s, struct sockaddr FAR\* saddr, FILE\* f) {

    char msg[] = "Hello world!\n";

    int errorCode = connect(s, saddr, sizeof(SOCKADDR\_IPX));

    if (errorCode){

        printf("Error while connecting! %X\n", WSAGetLastError());

        cin.get();

        return;

    }

    errorCode = send(s, msg, sizeof(msg), 0);

    if (errorCode){

        printf("Error while sending! %X\n", WSAGetLastError());

        cin.get();

        return;

    }

}

int main() {

    int errorCode;

    if (initializeWSA()) return 1;

    SOCKET s;

    unsigned short socketIDServer = 0x4444, socketIDClient = 0;

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

    if (s == INVALID\_SOCKET) {

        cout << "Socket creation failed!" << endl;

        if (closeWSA()) exit(10);

        exit(11);

    }

    SOCKADDR\_IPX serverAdress, clientAdress;

    clientAdress.sa\_family = AF\_IPX;

    clientAdress.sa\_socket = htons(socketIDClient);

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

    int size = sizeof(SOCKADDR\_IPX);

    getsockname(s, (sockaddr\*)& clientAdress, &size);

    printf("Current address is ");

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

    printf(" with socket %X\n", htons(clientAdress.sa\_socket));

    serverAdress.sa\_family = AF\_IPX;

    serverAdress.sa\_socket = htons(socketIDServer);

    memset(serverAdress.sa\_netnum, 0, 4);

    char \*c = serverAdress.sa\_nodenum;

    c[0] = 0x08; c[1] = 0; c[2] = 0x27; c[3] = 0x84; c[4] = 0xC4; c[5] = 0xD5;

    printf("Server address is ");

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

    printf(" with socket: %X\n", htons(serverAdress.sa\_socket));

    string filename;

    cout << "filename to receive?\n";

    cin >> filename;

    cin.get();

    cout << " Connecting to server..." << endl;

    errorCode = connect(s, (sockaddr\*)&serverAdress, sizeof(SOCKADDR\_IPX));

    if (errorCode){

        printf("Error while connecting! %X\n", WSAGetLastError());

        cin.get();

        exit(11);

    }

    errorCode = send(s, filename.c\_str(), filename.length()+1, 0);

    if (errorCode == SOCKET\_ERROR){

        printf("Error while sending! %X\n", WSAGetLastError());

        cin.get();

        exit(12);

    }

    int iResult;

    int fileSize;

    iResult = recv(s, (char\*)&fileSize, sizeof(int), 0);

    if (iResult > 0)

        printf(" %d received bytes\n", iResult);

    else if ( iResult == 0 )

        printf("Connection closed\n");

    else

        printf("receiving failed!");

    printf("file size is %i, \n", fileSize);

    if (fileSize == 0) cout << "File not found!" << endl;

    else {

        char\* recvbuf = new char[fileSize];

        iResult = recv(s, recvbuf, fileSize, 0);

        if ( iResult > 0 )

            printf("Bytes received: %d\n", iResult);

        else if ( iResult == 0 )

            printf("Connection closed\n");

        else

            printf("receiving failed with error: %d\n", WSAGetLastError());

        char tmp[4];

        string newFileName(itoa(htons(clientAdress.sa\_socket), tmp, 16));

        FILE\* f = fopen((newFileName + filename).c\_str(), "wb");

        fwrite(recvbuf, fileSize, 1, f);

        delete recvbuf;

        fclose(f);

    }

    errorCode = closesocket(s);

    if (errorCode == SOCKET\_ERROR) {

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

        if (closeWSA())

            return 12;

        return 2;

    }

    if (closeWSA())

        return 1;

    cout << "Translation completed!" << endl;

    getchar();

    return 0;

}

1. Spxserver:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <wsipx.h>

#include <string>

#include <process.h>

#define bufferLEN 256

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

int initializeWSA() {

    WORD wVersionRequested;

    WSADATA wsaData;

    int errorCode;

    wVersionRequested = MAKEWORD(2, 2);

    errorCode = WSAStartup(wVersionRequested, &wsaData);

    if (errorCode != 0) {

        cout << "Failed to initialize WSA!" << endl;

        return 1;

    }

    return 0;

}

int closeWSA() {

    int errorCode;

    errorCode = WSACleanup();

    if (errorCode != 0) {

        cout << "Failed to close WSA!" << endl;

        return 1;

    }

    return 0;

}

void PrintIpxAddress(char \*lpsNetnum, char \*lpsNodenum){

    int i;

    for (i=0; i < 4 ;i++){

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

    }

    printf(".");

    for (i=0; i < 6 ;i++){

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

    }

    printf("\n");

}

long int getFileSize(FILE\* f) {

    long int size;

    fseek(f, 0, SEEK\_END);

    size = ftell(f);

    fseek(f, 0, SEEK\_SET);

    return size;

};

DWORD WINAPI workWithClient(CONST LPVOID lpParam){

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

    char recvbuffer[bufferLEN];

    int iResult, iSendResult;

    int recvbufferlen = bufferLEN;

    iResult = recv(clientSocket, recvbuffer, recvbufferlen, 0);

    if (iResult > 0) {

        printf("Bytes received: %d\n", iResult);

    } else if (iResult == 0)

        printf("Connection closing...\n");

    else {

        printf("recv failed: %d\n", WSAGetLastError());

        closesocket(clientSocket);

        WSACleanup();

        ExitThread(10);

    }

    bool isFileFounded = true;

    cout << "Opening file " << recvbuffer << endl;

    FILE \*f = fopen(recvbuffer, "rb");

    if (f == NULL) {

        cout << "Can't find file \"" << recvbuffer << "\"" << endl;

        isFileFounded = false;

    }

    int fileSize = isFileFounded ? getFileSize(f) : 0;

    char \*f\_buffer = new char[fileSize];

    if (isFileFounded) fread(f\_buffer, fileSize, 1, f);

    iSendResult = send(clientSocket, (char\*)&fileSize, sizeof(int), 0);

    if (isFileFounded){

        if (iSendResult == SOCKET\_ERROR) {

            printf("send failed: %d\n", WSAGetLastError());

            closesocket(clientSocket);

            WSACleanup();

            ExitThread(10);

        }

        iSendResult = send(clientSocket, f\_buffer, fileSize, 0);

        fclose(f);

        if (iSendResult == SOCKET\_ERROR) {

            printf("send failed: %d\n", WSAGetLastError());

            closesocket(clientSocket);

            WSACleanup();

            ExitThread(10);

        }

        printf("Bytes sent: %d\n", iSendResult);

    }

    ExitThread(0);

}

DWORD WINAPI listenClients(CONST LPVOID lpParam){

    SOCKET s = \*((SOCKET\*)lpParam);

    SOCKET clientSocket;

    SOCKADDR\_IPX clientAdress;

    int size = sizeof(clientAdress);

    while(1){

        clientSocket = accept(s, (sockaddr\*)&clientAdress, &size);

        if (clientSocket == INVALID\_SOCKET) {

            printf("Failed to accept");

            closesocket(s);

            WSACleanup();

            ExitThread(10);

        }

        printf("New client accepted! \n");

        printf("Client address is ");

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

        printf(" with socket %X\n", htons(clientAdress.sa\_socket));

        CreateThread(NULL, 0, &workWithClient, &clientSocket, 0, NULL);

        Sleep(1);

    }

}

void sendFile(SOCKET s, const struct sockaddr FAR\* saddr, FILE \*f) {

    unsigned fileSize = getFileSize(f);

    char \*buffer = new char[fileSize];

    fread(buffer, 1, fileSize, f);

    int err = connect(s, saddr, sizeof(SOCKADDR\_IPX));

    if (err){

        printf("Error while connecting! %X\n", WSAGetLastError());

        cin.get();

        return;

    }

    err = send(s, buffer, fileSize, 0);

    if (err){

        printf("Error while sending! %X\n", WSAGetLastError());

        cin.get();

        return;

    }

}

int main() {

    int errorCode;

    if (initializeWSA()) return 1;

    SOCKET s;

    unsigned short socketIDServer = 0x4444, socketIDClient = 0x4445;

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

    if (s == INVALID\_SOCKET) {

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

        if (closeWSA())

            return 12;

        return 2;

    }

    SOCKADDR\_IPX serverAdress;

    serverAdress.sa\_family = AF\_IPX;

    serverAdress.sa\_socket = htons(socketIDServer);

    if (bind(s, (sockaddr\*)& serverAdress, sizeof(SOCKADDR\_IPX)) == SOCKET\_ERROR){

        printf("Error while binding server adress!\n");

        exit(10);

    }

    int size = sizeof(SOCKADDR\_IPX);

    getsockname(s, (sockaddr\*)& serverAdress, &size);

    printf("Server address is ");

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

    printf(" with socket %X\n", htons(serverAdress.sa\_socket));

    cout << "Listening...\n";

    if ( listen( s, SOMAXCONN ) == SOCKET\_ERROR ) {

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

        closesocket(s);

        WSACleanup();

        return 1;

    }

    CreateThread(NULL, 0, &listenClients, &s, 0, NULL);

    errorCode = closesocket(s);

    if (errorCode == SOCKET\_ERROR) {

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

        if (closeWSA())

            exit(11);

        exit(12);

    }

    if (closeWSA())

        return 1;

    cout << "Translation complete!";

    ExitProcess(0);

}

1. Ipxclient:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <wsipx.h>

#include <string>

#include <stdlib.h>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

int initializeWSA() {

    WORD wVersionRequested;

    WSADATA wsaData;

    int err;

    wVersionRequested = MAKEWORD(2, 2);

    err = WSAStartup(wVersionRequested, &wsaData);

    if (err != 0) {

        cout << "WSAStartup failed with error: " << err << endl;

        cout << WSAGetLastError() << endl;

        return 1;

    }

    return 0;

}

int closeWSA() {

    int err;

    err = WSACleanup();

    if (err != 0) {

        cout << "WSACleanup failed with error: " << err << endl;

        cout << WSAGetLastError() << endl;

        return 1;

    }

    return 0;

}

long int getFileSize(FILE\* f) {

    long int size;

    fseek(f, 0, SEEK\_END);

    size = ftell(f);

    fseek(f, 0, SEEK\_SET);

    return size;

};

struct firstPacket {

    unsigned fileSize;

    int maxbufferferSize;

    int fullPacketsCount;

    int lastPacketSize;

};

void receiveFile(SOCKET s, struct sockaddr FAR\* saddr, FILE\* f) {

    unsigned fileSize, bufferferSize;

    int fullPacketsCount, lastPacketSize, packet\_num;

    firstPacket fp;

    char tmp;

    int sz = sizeof(SOCKADDR\_IPX);

    recvfrom(s, (char\*) &fp, sizeof(fp), 0, saddr, &sz);

    fileSize= fp.fileSize;

    bufferferSize = fp.maxbufferferSize;

    fullPacketsCount = fp.fullPacketsCount;

    lastPacketSize = fp.lastPacketSize;

    packet\_num = fullPacketsCount + (lastPacketSize != 0 ? 1 : 0);

    cout << "File size is " << fileSize << " bytes" << endl;

    cout << "Data size in one packet is " << bufferferSize << " bytes" << endl;

    cout << "Will be sent: " << fullPacketsCount << " full packets";

    if (lastPacketSize != 0)

        cout << " and 1 incomplete packet, that contains " << lastPacketSize << " bytes";

    cout << endl;

    char\*\* buffer;

    buffer = new char\*[packet\_num];

    for (int i = 0; i < packet\_num; i++)

        buffer[i] = new char[bufferferSize];

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

        recvfrom(s, buffer[i], bufferferSize, 0, saddr, &sz);

    }

    if (lastPacketSize != 0){

        recvfrom(s, buffer[fullPacketsCount], bufferferSize, 0, saddr, &sz);

    }

    for (int i = 0; i < fullPacketsCount; i++)

        for (int k = 0; k < bufferferSize; k++)

            fputc(buffer[i][k], f);

    for (int k = 0; k < lastPacketSize; k++)

            fputc(buffer[fullPacketsCount][k], f);

    for (int i = 0; i < packet\_num; i++)

        delete buffer[i];

    delete buffer;

}

int main() {

    int errorCode;

    if (initializeWSA())

        return 1;

    SOCKET s;

    unsigned short socketID\_svr = 0x4444, socketID\_clt = 0x4445;

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

    if (s == INVALID\_SOCKET) {

        cout << "Failed to create socket! Error code is " << WSAGetLastError() << endl;

        closeWSA();

        exit(10);

    }

    SOCKADDR\_IPX serverAdress, clientAdress;

    clientAdress.sa\_family = AF\_IPX;

    clientAdress.sa\_socket = htons(socketID\_clt);

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

    int sz = sizeof(SOCKADDR\_IPX);

    getsockname(s, (sockaddr\*)& clientAdress, &sz);

    serverAdress.sa\_family = AF\_IPX;

    serverAdress.sa\_socket = htons(socketID\_svr);

    memset(serverAdress.sa\_netnum, 0, 4);

    memset(serverAdress.sa\_nodenum, 0xFF, 6);

    char tmp[4];

    string fileName = itoa(htons(clientAdress.sa\_socket), tmp, 16);

    fileName += "input.jpg";

    FILE\* result = fopen(fileName.c\_str(), "wb");

    if (result == NULL) {

        cout << "Failed to open file!";

        exit(20);

    }

    cout << "Waiting for server to start translation...!" << endl;

    receiveFile(s, (sockaddr\*)& serverAdress, result);

    cout << "Closing input file..." << endl;

    fclose(result);

    cout << "File closed!" << endl;

    closeWSA();

    errorCode = closesocket(s);

    if (errorCode == SOCKET\_ERROR) {

        cout << "Failed to close socket! Error code is " << WSAGetLastError() << endl;

        closeWSA();

        exit(11);

    }

    closeWSA();

    cout << "Translation complete!" << endl;

    getchar();

    return 0;

}

1. ipxServer:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <wsipx.h>

#include <string>

#include <process.h>

#define bufferLEN 256

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

int initializeWSA() {

    WORD wVersionRequested;

    WSADATA wsaData;

    int errorCode;

    wVersionRequested = MAKEWORD(2, 2);

    errorCode = WSAStartup(wVersionRequested, &wsaData);

    if (errorCode != 0) {

        cout << "Failed to initialize WSA!" << endl;

        return 1;

    }

    return 0;

}

int closeWSA() {

    int errorCode;

    errorCode = WSACleanup();

    if (errorCode != 0) {

        cout << "Failed to close WSA!" << endl;

        return 1;

    }

    return 0;

}

void PrintIpxAddress(char \*lpsNetnum, char \*lpsNodenum){

    int i;

    for (i=0; i < 4 ;i++){

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

    }

    printf(".");

    for (i=0; i < 6 ;i++){

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

    }

    printf("\n");

}

long int getFileSize(FILE\* f) {

    long int size;

    fseek(f, 0, SEEK\_END);

    size = ftell(f);

    fseek(f, 0, SEEK\_SET);

    return size;

};

DWORD WINAPI workWithClient(CONST LPVOID lpParam){

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

    char recvbuffer[bufferLEN];

    int iResult, iSendResult;

    int recvbufferlen = bufferLEN;

    iResult = recv(clientSocket, recvbuffer, recvbufferlen, 0);

    if (iResult > 0) {

        printf("Bytes received: %d\n", iResult);

    } else if (iResult == 0)

        printf("Connection closing...\n");

    else {

        printf("recv failed: %d\n", WSAGetLastError());

        closesocket(clientSocket);

        WSACleanup();

        ExitThread(10);

    }

    bool isFileFounded = true;

    cout << "Opening file " << recvbuffer << endl;

    FILE \*f = fopen(recvbuffer, "rb");

    if (f == NULL) {

        cout << "Can't find file \"" << recvbuffer << "\"" << endl;

        isFileFounded = false;

    }

    int fileSize = isFileFounded ? getFileSize(f) : 0;

    char \*f\_buffer = new char[fileSize];

    if (isFileFounded) fread(f\_buffer, fileSize, 1, f);

    iSendResult = send(clientSocket, (char\*)&fileSize, sizeof(int), 0);

    if (isFileFounded){

        if (iSendResult == SOCKET\_ERROR) {

            printf("send failed: %d\n", WSAGetLastError());

            closesocket(clientSocket);

            WSACleanup();

            ExitThread(10);

        }

        iSendResult = send(clientSocket, f\_buffer, fileSize, 0);

        fclose(f);

        if (iSendResult == SOCKET\_ERROR) {

            printf("send failed: %d\n", WSAGetLastError());

            closesocket(clientSocket);

            WSACleanup();

            ExitThread(10);

        }

        printf("Bytes sent: %d\n", iSendResult);

    }

    ExitThread(0);

}

DWORD WINAPI listenClients(CONST LPVOID lpParam){

    SOCKET s = \*((SOCKET\*)lpParam);

    SOCKET clientSocket;

    SOCKADDR\_IPX clientAdress;

    int size = sizeof(clientAdress);

    while(1){

        clientSocket = accept(s, (sockaddr\*)&clientAdress, &size);

        if (clientSocket == INVALID\_SOCKET) {

            printf("Failed to accept");

            closesocket(s);

            WSACleanup();

            ExitThread(10);

        }

        printf("New client accepted! \n");

        printf("Client address is ");

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

        printf(" with socket %X\n", htons(clientAdress.sa\_socket));

        CreateThread(NULL, 0, &workWithClient, &clientSocket, 0, NULL);

        Sleep(1);

    }

}

void sendFile(SOCKET s, const struct sockaddr FAR\* saddr, FILE \*f) {

    unsigned fileSize = getFileSize(f);

    char \*buffer = new char[fileSize];

    fread(buffer, 1, fileSize, f);

    int err = connect(s, saddr, sizeof(SOCKADDR\_IPX));

    if (err){

        printf("Error while connecting! %X\n", WSAGetLastError());

        cin.get();

        return;

    }

    err = send(s, buffer, fileSize, 0);

    if (err){

        printf("Error while sending! %X\n", WSAGetLastError());

        cin.get();

        return;

    }

}

int main() {

    int errorCode;

    if (initializeWSA()) return 1;

    SOCKET s;

    unsigned short socketIDServer = 0x4444, socketIDClient = 0x4445;

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

    if (s == INVALID\_SOCKET) {

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

        if (closeWSA())

            return 12;

        return 2;

    }

    SOCKADDR\_IPX serverAdress;

    serverAdress.sa\_family = AF\_IPX;

    serverAdress.sa\_socket = htons(socketIDServer);

    if (bind(s, (sockaddr\*)& serverAdress, sizeof(SOCKADDR\_IPX)) == SOCKET\_ERROR){

        printf("Error while binding server adress!\n");

        exit(10);

    }

    int size = sizeof(SOCKADDR\_IPX);

    getsockname(s, (sockaddr\*)& serverAdress, &size);

    printf("Server address is ");

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

    printf(" with socket %X\n", htons(serverAdress.sa\_socket));

    cout << "Listening...\n";

    if ( listen( s, SOMAXCONN ) == SOCKET\_ERROR ) {

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

        closesocket(s);

        WSACleanup();

        return 1;

    }

    CreateThread(NULL, 0, &listenClients, &s, 0, NULL);

    errorCode = closesocket(s);

    if (errorCode == SOCKET\_ERROR) {

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

        if (closeWSA())

            exit(11);

        exit(12);

    }

    if (closeWSA())

        return 1;

    cout << "Translation complete!";

    ExitProcess(0);

}