

**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»  
по теме «Протоколы IPX/SPX в библиотеке Winsock»

Выполнил: студент группы ВТ-31  
Проверил:

Макаров Д.С.  
Федотов Е.А.

Белгород 2020

# Лабораторная работа №2

## «Протоколы IPX/SPX в библиотеке Winsock»

**Цель работы:**изучить протоколы IPX/SPX, основные функции библиотеки Winsock и разработать 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.

**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. 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 ЕСВ 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.

## 2. Основные функции API, использованные в данной работе.

- WSASStartup (WORD wVersionRequested, LPWSADATA lpWSADATA)
- SAGetLastError (void)
- WSACleanup (void)
- socket (int af, int type, int protocol)
- bind (SOCKET s, const struct sockaddr FAR\* name, int namelen)
- listen (SOCKET s, int backlog)
- connect (SOCKET s, const struct sockaddr FAR\* name, int namelen)
- accept (SOCKET s, struct sockaddr FAR\* addr, int FAR\* addrlen)
- getsockname (SOCKET s, struct sockaddr FAR\* name, int FAR\* namelen)
- sendto (SOCKET s, const char FAR \* buf, int len, int flags, const struct sockaddr FAR \* to, int tolen)
- send (SOCKET s, const char FAR \* buf, int len, int flags)
- recvfrom (SOCKET s, char FAR\* buf, int len, int flags, struct sockaddr FAR\* from, int FAR\* fromlen)
- recv (SOCKET s, char FAR\* buf, int len, int flags)
- closesocket(SOCKET s)

## 3. Разработка программы. Блок-схемы программы.

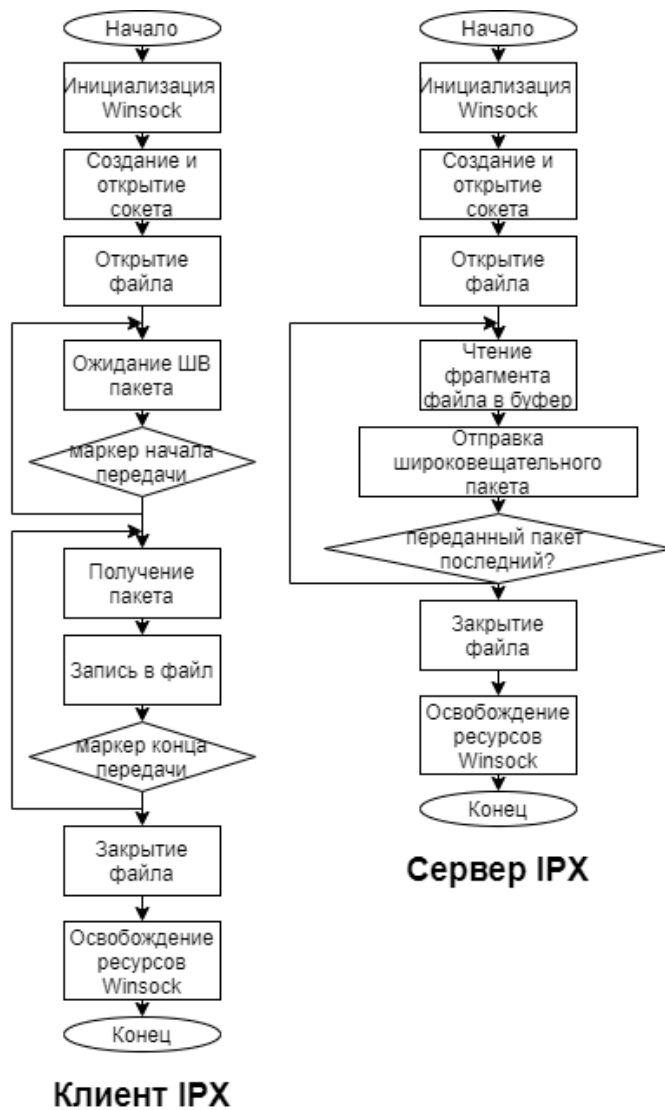


Рис. 1: Блок схемы программы сервер/клиент IPX



Рис. 2: Блок схемы программы сервер/клиент SPX

#### 4. Анализ функционирования разработанных программ.

##### IPX

Кол-во клиентов	Время передачи	Кол-во потерянных пакетов
1	204729 мс	3
2	239538 мс, 241053 мс	4, 1
3	310356 мс, 303564 мс, 309782 мс	215, 189, 204

##### SPX

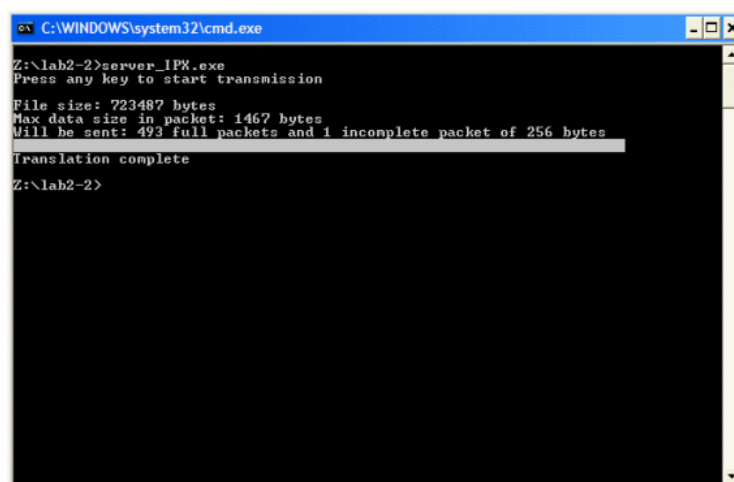
Кол-во клиентов	Время передачи	Кол-во потерянных пакетов
1	36294 мс	0
2	51954 мс	0, 0
3	75672 мс	0, 0, 0

#### 5. Выводы.

SPX в отличие от IPX гарантирует доставку пакета без потерь, а так же позволяет эффективно использовать канал передачи данных не занимая его широковещательными пакетами. API построенное по архитектуре сокетов Беркли а в частности Winsock предоставляет удобные абстракции для написания программ использующие сетевые соединения.

#### 6. Тексты программ. Скриншоты программ.

Тексты программ см. в приложении.



```
C:\WINDOWS\system32\cmd.exe
Z:\lab2-2>server_IPX.exe
Press any key to start transmission
File size: 723487 bytes
Max data size in packet: 1467 bytes
Will be sent: 493 full packets and 1 incomplete packet of 256 bytes
Translation complete
Z:\lab2-2>
```

Рис. 3: Программа-сервер IPX

```
C:\WINDOWS\system32\cmd.exe
Z:\lab2-2>client_IPX.exe
Waiting for start...
File size: 723487 bytes
Max data size in packet: 1467 bytes
Will be sent: 493 full packets and 1 incomplete packet of 256 bytes
Translation complete
Z:\lab2-2>_
```

Рис. 4: Программа-клиент IPX

```
C:\WINDOWS\system32\cmd.exe - server_SPX.exe
Z:\lab2-2>server_SPX.exe
Server address:
00000000.080027F7CE45
Socket: 4444
Listening...
Press 'e' to exit..

New client accept:
Client address: 1234CDEF.080027F7CE45
Socket: 6004
Bytes received: 13
Opening file test_img.jpg
Bytes sent: 723487
-
```

Рис. 5: Программа-сервер SPX

```
C:\WINDOWS\system32\cmd.exe - client_SPX.exe
Z:\lab2-2>client_SPX.exe
My address is: 00000000.080027F7CE45
Socket: 6004
Server address is: 00000000.080027F7CE45
Socket: 4444
Enter file name to load:

test_img.jpg
Connecting to server...
Bytes received: 4
File size: 723487
Bytes received: 723487
Translation complete
-
```

Рис. 6: Программа-клиент SPX

# Приложение

## Содержимое файла ipxClient.c

```
#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>

using namespace std;

int initWSA() {
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    wVersionRequested = MAKEWORD(2, 2);
    err = WSASStartup(wVersionRequested, &wsaData);
    if (err != 0) {
        cout << "WSASStartup 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 get_file_size(FILE* f) {
    long int size;
    fseek(f, 0, SEEK_END);
    size = ftell(f);
    fseek(f, 0, SEEK_SET);
    return size;
};

struct first_packet {
    unsigned file_sz;
    int max_buf_sz;
    int full_packet_num;
    int last_packet_size;
};

void receiveFile(SOCKET s, struct sockaddr FAR* saddr, FILE* f) {
    unsigned f_sz, buf_sz;
    int full_packet_num, last_packet_sz, packet_num;
    first_packet fp;
```

```

char tmp;
int sz = sizeof(SOCKADDR_IPX);
recvfrom(s, (char*) &fp, sizeof(fp), 0, saddr, &sz);
f_sz= fp.file_sz;
buf_sz = fp.max_buf_sz;
full_packet_num = fp.full_packet_num;
last_packet_sz = fp.last_packet_size;
packet_num = full_packet_num + (last_packet_sz != 0 ? 1 : 0);
cout << "File size: " << f_sz << " bytes" << endl;
cout << "Max data size in packet: " << buf_sz << " bytes" << endl;
cout << "Will be sent: " << full_packet_num << " full packets";
if (last_packet_sz != 0)
    cout << " and 1 incomplete packet of " << last_packet_sz << " bytes";
cout << endl;

char** buf;
buf = new char*[packet_num];
for (int i = 0; i < packet_num; i++)
    buf[i] = new char[buf_sz];
float progress_step = 70.0 / packet_num;
float step_count = 0;
for (int i = 0; i < full_packet_num; i++){
    recvfrom(s, buf[i], buf_sz, 0, saddr, &sz);
    step_count += progress_step;
    while(step_count > 1){
        cout << char(219);
        step_count--;
    }
}
if (last_packet_sz != 0){
    recvfrom(s, buf[full_packet_num], buf_sz, 0, saddr, &sz);
    step_count += progress_step;
    while(step_count > 1){
        cout << char(219);
        step_count--;
    }
}
cout << endl;
for (int i = 0; i < full_packet_num; i++)
    for (int k = 0; k < buf_sz; k++)
        fputc(buf[i][k], f);
for (int k = 0; k < last_packet_sz; k++)
    fputc(buf[full_packet_num][k], f);
for (int i = 0; i < packet_num; i++)
    delete buf[i];
delete buf;
}

int main() {
    int err;
    if (initWSA())
        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 << "Socket creation failed with error: " << WSAGetLastError() << endl;
        if (closeWSA())
            return 12;
    }
}

```



```

        return 2;
    }

    SOCKADDR_IPX svr_adr, clt_adr;
    clt_adr.sa_family = AF_IPX;
    clt_adr.sa_socket = htons(socketID_clt);
    bind(s, (sockaddr*)& clt_adr, sizeof(SOCKADDR_IPX));
    int sz = sizeof(SOCKADDR_IPX);
    getsockname(s, (sockaddr*)& clt_adr, &sz);

    svr_adr.sa_family = AF_IPX;
    svr_adr.sa_socket = htons(socketID_svr);
    memset(svr_adr.sa_netnum, 0, 4);
    memset(svr_adr.sa_nodenum, 0xFF, 6);

    char tmp[4];
    string f_name = itoa(htons(clt_adr.sa_socket), tmp, 16);
    f_name += "test_img.jpg";
    FILE* f_out = fopen(f_name.c_str(), "wb");
    if (f_out == NULL) {
        cout << "Unable to open file \"" << f_name << "\"" << endl;
        return 3;
    }
    cout << " Waiting for start..." << endl;
    receiveFile(s, (sockaddr*)& svr_adr, f_out);
    fclose(f_out);

    err = closesocket(s);
    if (err == SOCKET_ERROR) {
        cout << "Socket closure failed with error: " << WSAGetLastError() << endl;
        if (closeWSA())
            return 12;
        return 2;
    }

    if (closeWSA())
        return 1;
    cout << "Translation complete" << endl;
    getchar();
    return 0;
}

```

## Содержимое файла ipxServer.c

```

#include <iostream>
#include <stdio.h>
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <wsipx.h>
#include <string>

```

```
using namespace std;
```

```

int initWSA() {
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;

```

```

wVersionRequested = MAKEWORD(2, 2);
err = WSASStartup(wVersionRequested, &wsaData);
if (err != 0) {
    cout << "WSASStartup 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 get_file_size(FILE* f) {
    long int size;
    fseek(f, 0, SEEK_END);
    size = ftell(f);
    fseek(f, 0, SEEK_SET);
    return size;
};

struct first_packet {
    unsigned file_sz;
    unsigned max_buf_sz;
    unsigned full_packet_num;
    unsigned last_packet_size;
};

void sendFile(SOCKET s, const struct sockaddr FAR* saddr, FILE *f) {
    unsigned f_sz = get_file_size(f);
    unsigned optlen = sizeof(unsigned);
    unsigned optval;
    getsockopt(s, SOL_SOCKET, SO_MAX_MSG_SIZE, (char*)& optval, (int*)&optlen);
    unsigned buf_sz = optval;
    unsigned full_packet_num = f_sz / buf_sz;
    unsigned last_packet_sz = f_sz % buf_sz;
    unsigned packet_num = full_packet_num + (last_packet_sz != 0 ? 1 : 0);
    cout << "File size: " << f_sz << " bytes" << endl;
    cout << "Max data size in packet: " << buf_sz << " bytes" << endl;
    cout << "Will be sent: " << full_packet_num << " full packets";
    if (last_packet_sz != 0)
        cout << " and 1 incomplete packet of " << last_packet_sz << " bytes";
    cout << endl;

    first_packet fp;
    fp.file_sz = f_sz;
    fp.max_buf_sz = buf_sz;
    fp.full_packet_num = full_packet_num;
    fp.last_packet_size = last_packet_sz;
    sendto(s, (char*)& fp, sizeof(fp), 0, saddr, sizeof(SOCKADDR_IPX));
    char** buf;
    buf = new char*[packet_num];

```

```

for (int i = 0; i < packet_num; i++)
    buf[i] = new char[buf_sz];
for (int i = 0; i < full_packet_num; i++)
    fread(buf[i], buf_sz, 1, f);
if (last_packet_sz > 0)
    fread(buf[full_packet_num], last_packet_sz, 1, f);
Sleep(10);

float progress_step = 70.0 / packet_num;
float step_count = 0;
for (int i = 0; i < full_packet_num ; i++) {
    sendto(s, buf[i], buf_sz, 0, saddr, sizeof(SOCKADDR_IPX));
    step_count += progress_step;
    while(step_count > 1){
        cout << char(219);
        step_count--;
    }
    int x = 1000;
    while(x--);
}
if (last_packet_sz != 0){
    sendto(s, buf[full_packet_num], last_packet_sz, 0, saddr, sizeof(SOCKADDR_IPX));
    step_count += progress_step;
    while(step_count > 1){
        cout << char(219);
        step_count--;
    }
}
cout << endl;
for (int i = 0; i < packet_num; i++)
    delete buf[i];
delete buf;
}

int main() {
    int err;
    if (initWSA())
        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 << "Socket creation failed with error: " << WSAGetLastError() << endl;
        if (closeWSA())
            return 12;
        return 2;
    }
    SOCKADDR_IPX svr_adr, clt_adr;
    svr_adr.sa_family = AF_IPX;
    svr_adr.sa_socket = htons(socketID_svr);
    bind(s, (sockaddr*)& svr_adr, sizeof(SOCKADDR_IPX));
    clt_adr.sa_family = AF_IPX;
    clt_adr.sa_socket = htons(socketID_clt);
    memset(clt_adr.sa_netnum, 0, 4);
    memset(clt_adr.sa_nodenum, 0xFF, 6);
    int set_broadcast = 1;
    setsockopt(s, SOL_SOCKET, SO_BROADCAST, (char*)& set_broadcast, sizeof(set_broadcast));

    string f_name;
    f_name = "test_img.jpg";

```

```

FILE *f_in = fopen(f_name.c_str(), "rb");
if (f_in == NULL) {
    cout << "Unable to open file \"" << f_name << "\"" << endl;
    return 3;
}

cout << "Press any key to start transmission" << endl;
getchar();
sendFile(s, (sockaddr*)& clt_adr, f_in);
fclose(f_in);

err = closesocket(s);
if (err == SOCKET_ERROR) {
    cout << "Socket closure failed with error: " << WSAGetLastError() << endl;
    if (closeWSA())
        return 12;
    return 2;
}

if (closeWSA())
    return 1;
cout << "Translation complete";
getchar();
return 0;
}

```

## Содержимое файла spxClient.c

```

#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>

using namespace std;

int initWSA() {
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    wVersionRequested = MAKEWORD(2, 2);
    err = WSASStartup(wVersionRequested, &wsaData);
    if (err != 0) {
        cout << "WSASStartup 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;
}

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");
}

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 get_file_size(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!\n";
    int err = connect(s, saddr, sizeof(SOCKADDR_IPX));
    if (err){
        printf("Error while connecting! %X\n", WSAGetLastError());
        cin.get();
        return;
    }
    err = send(s, msg, sizeof(msg), 0);
    if (err){
        printf("Error while sending! %X\n", WSAGetLastError());
        cin.get();
        return;
    }
}

int main() {
    int err;
    if (initWSA())
        return 1;
}

```

```

SOCKET s;
unsigned short socketID_svr = 0x4444, socketID_clt = 0;
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 srv_adr, clt_adr;

clt_adr.sa_family = AF_IPX;
clt_adr.sa_socket = htons(socketID_clt);
bind(s, (sockaddr*)& clt_adr, sizeof(SOCKADDR_IPX));
int sz = sizeof(SOCKADDR_IPX);
getsockname(s, (sockaddr*)& clt_adr, &sz);

printf("My address is: ");
PrintIpAddress(clt_adr.sa_netnum, clt_adr.sa_nodenum);
printf("Socket: %X\n", htons(clt_adr.sa_socket));

srv_adr.sa_family = AF_IPX;
srv_adr.sa_socket = htons(socketID_svr);
memset(srv_adr.sa_netnum, 0, 4);
char *c = srv_adr.sa_nodenum;
c[0] = 0x08; c[1] = 0; c[2] = 0x27; c[3] = 0xF7; c[4] = 0xCE; c[5] = 0x45;

printf("Server address is: ");
PrintIpAddress(srv_adr.sa_netnum, srv_adr.sa_nodenum);
printf("Socket: %X\n", htons(srv_adr.sa_socket));

string f_name;
cout << "Enter file name to load: \n";
cin >> f_name;
cin.get();

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

err = connect(s, (sockaddr*)&srv_adr, sizeof(SOCKADDR_IPX));
if (err){
    printf("Error while connecting! %X\n", WSAGetLastError());
    cin.get();
    return 2;
}
err = send(s, f_name.c_str(), f_name.length()+1, 0);
if (err == SOCKET_ERROR){
    printf("Error while sending! %X\n", WSAGetLastError());
    cin.get();
    return 3;
}

int iResult;
int f_sz;

iResult = recv(s, (char*)&f_sz, sizeof(int), 0);
if ( iResult > 0 )
    printf("Bytes received: %d\n", iResult);
else if ( iResult == 0 )

```

```

        printf("Connection closed\n");
    else
        printf("recv failed with error: %d\n", WSAGetLastError());
    printf("file size: %i, \n", f_sz);

    if (f_sz == 0) cout << "File not found!" << endl;
    else {
        char* recvbuf = new char[f_sz];
        iResult = recv(s, recvbuf, f_sz, 0);
        if ( iResult > 0 )
            printf("Bytes received: %d\n", iResult);
        else if ( iResult == 0 )
            printf("Connection closed\n");
        else
            printf("recv failed with error: %d\n", WSAGetLastError());
        char tmp[4];
        string new_file_name(itoa(htons(clt_adr.sa_socket), tmp, 16));
        FILE* f = fopen((new_file_name + f_name).c_str(), "wb");
        fwrite(recvbuf, f_sz, 1, f);
        delete recvbuf;
        fclose(f);
    }
    err = closesocket(s);
    if (err == SOCKET_ERROR) {
        cout << "Socket closure failed with error: " << WSAGetLastError() << endl;
        if (closeWSA())
            return 12;
        return 2;
    }

    if (closeWSA())
        return 1;
    cout << "Translation complete" << endl;
    getchar();
    return 0;
}

```

## Содержимое файла spxServer.c

```

#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 BUFLen 256

using namespace std;

int initWSA() {
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    wVersionRequested = MAKEWORD(2, 2);
    err = WSASStartup(wVersionRequested, &wsaData);
    if (err != 0) {
        cout << "WSASStartup 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;
}

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 get_file_size(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 client_socket = *((SOCKET*)(lpParam));
    char recvbuf[BUFLen];
    int iResult, iSendResult;
    int recvbuflen = BUFLen;

    iResult = recv(client_socket, recvbuf, recvbuflen, 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(client_socket);
        WSACleanup();
        ExitThread(1);
    }
    bool file_found = true;

    cout << "Opening file " << recvbuf << endl;
    FILE *f = fopen(recvbuf, "rb");
    if (f == NULL) {
        cout << "Can't find file \"" << recvbuf << "\"" << endl;
    }
}

```



```

        file_found = false;
    }
    int f_sz = file_found ? get_file_size(f) : 0;
    char *f_buf = new char[f_sz];
    if (file_found) fread(f_buf, f_sz, 1, f);

    iSendResult = send(client_socket, (char*)&f_sz, sizeof(int), 0);
    if (file_found){
        if (iSendResult == SOCKET_ERROR) {
            printf("send failed: %d\n", WSAGetLastError());
            closesocket(client_socket);
            WSACleanup();
            ExitThread(1);
        }
        iSendResult = send(client_socket, f_buf, f_sz, 0);
        fclose(f);
        if (iSendResult == SOCKET_ERROR) {
            printf("send failed: %d\n", WSAGetLastError());
            closesocket(client_socket);
            WSACleanup();
            ExitThread(1);
        }
        printf("Bytes sent: %d\n", iSendResult);
    }
    ExitThread(0);
}

DWORD WINAPI listenClients(CONST LPVOID lpParam){
    SOCKET s = *((SOCKET*)lpParam);
    SOCKET client_socket;
    SOCKADDR_IPX clt_addr;
    int sz = sizeof(clt_addr);
    while(1){
        client_socket = accept(s, (sockaddr*)&clt_addr, &sz);
        if (client_socket == INVALID_SOCKET) {
            printf("accept failed: %d\n", WSAGetLastError());
            closesocket(s);
            WSACleanup();
            ExitThread(1);
        }
        printf("New client accept: \n");
        printf("Client address: ");
        PrintIpxAddress(clt_addr.sa_netnum, clt_addr.sa_nodenum);
        printf("Socket: %X\n", htons(clt_addr.sa_socket));
        CreateThread(NULL, 0, &workWithClient, &client_socket, 0, NULL);
        Sleep(1);
    }
}

void sendFile(SOCKET s, const struct sockaddr FAR* saddr, FILE *f) {
    unsigned f_sz = get_file_size(f);
    char *buf = new char[f_sz];
    fread(buf, 1, f_sz, f);
    int err = connect(s, saddr, sizeof(SOCKADDR_IPX));
    if (err){
        printf("Error while connecting! %X\n", WSAGetLastError());
        cin.get();
        return;
    }
    err = send(s, buf, f_sz, 0);
}

```

```

    if (err){
        printf("Error while sending! %X\n", WSAGetLastError());
        cin.get();
        return;
    }
}

int main() {
    int err;
    if (initWSA())
        return 1;
    SOCKET s;
    unsigned short socketID_svr = 0x4444, socketID_clt = 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 srv_adr;
    srv_adr.sa_family = AF_IPX;
    srv_adr.sa_socket = htons(socketID_svr);
    if (bind(s, (sockaddr*)& srv_adr, sizeof(SOCKADDR_IPX)) == SOCKET_ERROR){
        printf("Bind error %X\n", WSAGetLastError());
        return 10;
    }

    int sz = sizeof(SOCKADDR_IPX);
    getsockname(s, (sockaddr*)& srv_adr, &sz);

    printf("Server address: \n");
    PrintIpxAddress(srv_adr.sa_netnum, srv_adr.sa_nodenum);
    printf("Socket: %X\n", htons(srv_adr.sa_socket));

    cout << "Listening...\n";
    if ( listen( s, SOMAXCONN ) == SOCKET_ERROR ) {
        cout << "Listen failed with error: " << WSAGetLastError() << endl;
        closesocket(s);
        WSACleanup();
        return 1;
    }

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

    cout << "Press \'e\' to exit.." << endl;
    while (getchar() != 'e');

    err = closesocket(s);
    if (err == SOCKET_ERROR) {
        cout << "Socket closure failed with error: " << WSAGetLastError() << endl;
        if (closeWSA())
            return 12;
        return 2;
    }
    if (closeWSA())
        return 1;
    cout << "Translation complete";
    ExitProcess(0);
}

```