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)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №1

по дисциплине: Компьютерные сети
тема: «Протокол сетевого уровня IPX»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверили:
Рубцов Константин Анатольевич

Белгород 2025 г.

Цель работы: изучить протокол сетевого уровня IPX, основные функции API драйвера IPX и разработать программу для приема/передачи данных. Изменяем конфигурацию для использования протокол IPX в DosBox.

Краткие теоретические сведения

Протокол IPX – это протокол сетевого уровня модели взаимодействия открытых систем (OSI) реализующий передачу пакетов (сообщений) между станциями сети на уровне датаграмм. Датаграмма – это сообщение, доставка которого получателю не гарантируется. Следовательно, для обеспечения надежной работы нужно предусмотреть схему уведомления других станций о том, что переданные ими пакеты успешно приняты и обработаны. Более того, последовательность отправления пакетов передающим узлом может отличаться от последовательности приема этих пакетов, что также необходимо учитывать [1, 3, 7].

В процессе обмена сообщениями на уровне сеанса связи участвуют только две станции сети. На уровне датаграмм есть возможность посылать сообщение одновременно всем станциям сети.

Система адресов, используемая в протоколе IPX, представлена несколькими компонентами: это номер сети, адрес станции в сети и идентификатор программы на рабочей станции.

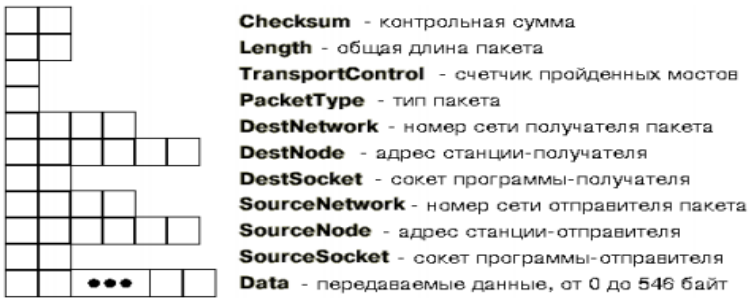
Номер сети - это номер сегмента сети, определяемого системным администратором. Если в общей сети есть мосты, каждая отдельная сеть, подключенная через мост, должна иметь свой, уникальный номер сети.

Адрес станции - это число, которое является уникальным для каждой рабочей станции. При использовании адаптеров Ethernet уникальность обеспечивается изготовителем сетевого адаптера. Специальный адрес FFFFFFFFFFFFh используется для рассылки
данных всем станциям данной сети одновременно.

Идентификатор программы на рабочей станции (сокет) - число, которое используется для адресации определенной программы, работающей на компьютере. В среде мультизадачных операционных систем, на каждой рабочей станции в сети одновременно может быть запущено несколько приложений. Для того, чтобы послать

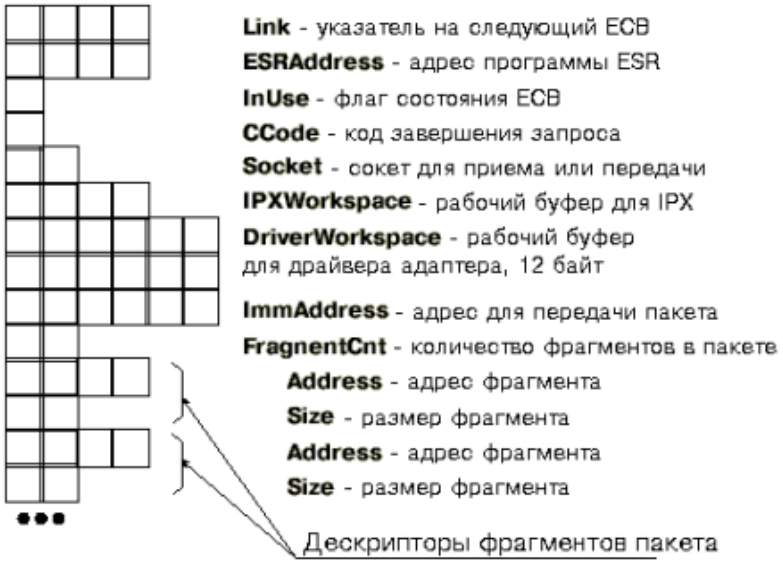
данные конкретной программе, используется идентификация программ при помощи сокетов. Каждая программа, желающая принимать или передавать данные по сети, должна получить свой, уникальный для данной рабочей станции, идентификатор - сокет.

Структура пакета



Прикладное ПО взаимодействует с сетевым драйвером через драйвер IPX. Управление драйвером происходит с помощью блока ECB(Блок контроля события).

Структура блока ECB



Для осуществления взаимодействия с драйвером IPX нужно получить его адрес с помощью прерывания 2f поместив в ax 7A00h. А затем через эту точку входа в API вызывать функции для работы с IPX. Функциям передаются различные параметры(Блок ECB, номер сокета, тип сокета).

Номер сокета передается в перевернутом виде, поэтому используем IntSwap

Основные функции API, использованные в данной работе.

Для работы с IPX используются следующие основные функции:

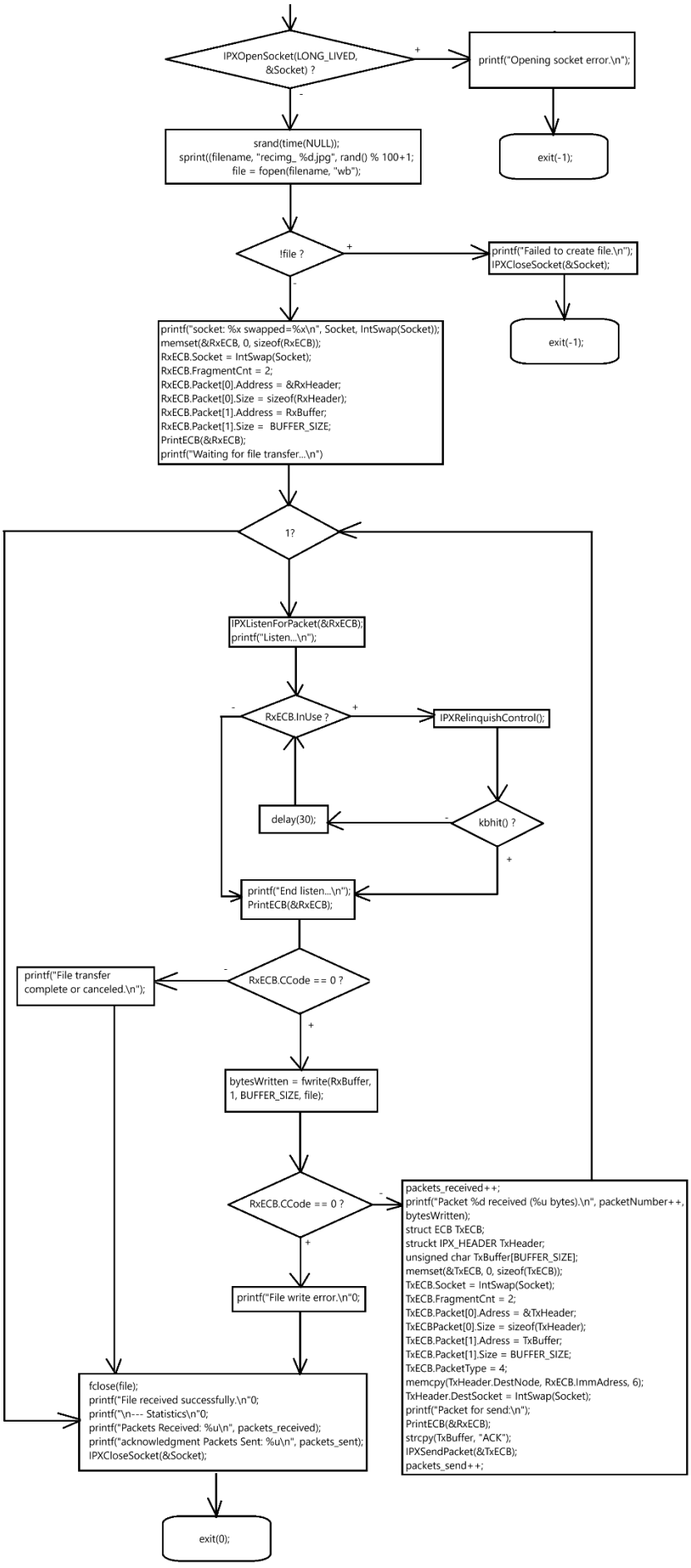
Функция **IPXOpenSocket** предназначена для получения сокетов. На входе BX = 00h, AL = Тип сокета: 00h - короткоживущий; FFh - долгоживущий, DX = Запрашиваемый номер сокета или 0000h, если требуется получить динамический номер сокета. Байты номера сокета находятся в перевернутом виде. На выходе AL = Код завершения (00h - сокет открыт; FFh - этот сокет уже был открыт раньше; FEh - переполнилась таблица сокетов), DX = Присвоенный номер сокета.

Функция **IPXCloseSocket** предназначена для освобождения сокетов, т.к. сокеты являются ограниченным ресурсом. На входе BX = 0lh, DX = Номер закрываемого сокета. На выходе регистры не используются.

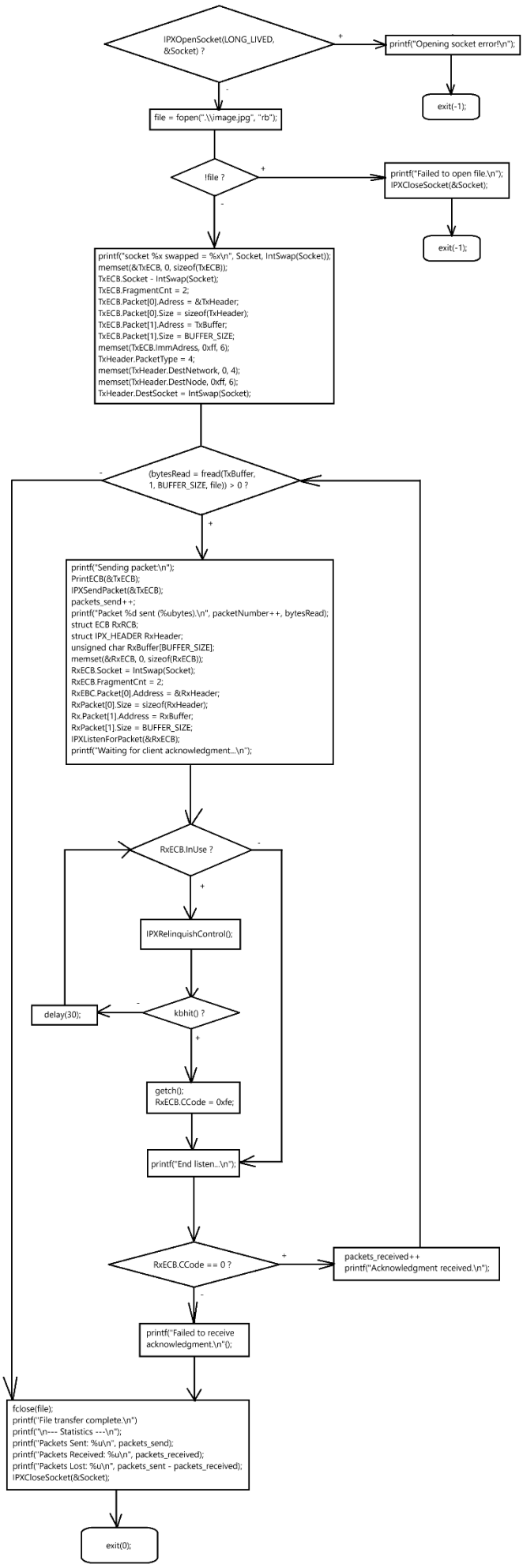
Функция **IPXGetLocalTaget** применяется для вычисления значения непосредственного адреса, помещаемого в поле ImmAddress блока ECB перед передачей пакета. На входе BX = 09h, ES:SI = Указатель на буфер длиной 10 байт, в который будет записан адрес станции, на которой работает данная.

Разработка программы. Блок-схемы программы

Client.c



Server.c


IPXOpenSocket(LONG_LIVED, &Socket) ?
+
printf("Opening socket error!\n");
exit(-1);
-
file = fopen(".\\image.jpg", "rb");
!file ?
+
printf("Failed to open file.\n");
IPXCloseSocket(&Socket);
exit(-1);
-
printf("socket %x swapped = %x\n", Socket, IntSwap(Socket));
memset(&TxECB, 0, sizeof(TxECB));
TxECB.Socket - IntSwap(Socket);
TxECB.FragmentCnt = 2;
TxECB.Packet[0].Adress = &TxHeader;
TxECB.Packet[0].Size = sizeof(TxHeader);
TxECB.Packet[1].Adress = TxBuffer;
TxECB.Packet[1].Size = BUFFER_SIZE;
memset(TxECB.ImmAdress, 0xff, 6);
TxHeader.PacketType = 4;
memset(TxHeader.DestNetwork, 0, 4);
memset(TxHeader.DestNode, 0xff, 6);
TxHeader.DestSocket = IntSwap(Socket);
(bytesRead = fread(TxBuffer, 1, BUFFER_SIZE, file)) > 0 ?
-
+
printf("Sending packet:\n");
PrintECB(&TxECB);
IPXSendPacket(&TxECB);
packets_send++;
printf("Packet %d sent (%ubytes).\n", packetNumber++, bytesRead);
struct ECB RxRCB;
struct IPX_HEADER RxHeader;
unsigned char RxBuffer[BUFFER_SIZE];
memset(&RxECB, 0, sizeof(RxECB));
RxECB.Socket = IntSwap(Socket);
RxECB.FragmentCnt = 2;
RxEBC.Packet[0].Address = &RxHeader;
RxPacket[0].Size = sizeof(RxHeader);
Rx.Packet[1].Address = RxBuffer;
RxPacket[1].Size = BUFFER_SIZE;
IPXListenForPacket(&RxECB);
printf("Waiting for client acknowledgment...\n");
RxECB.InUse ?
-
+
IPXRelinquishControl();
delay(30);
-
kbhit() ?
+
getch();
RxECB.CCode = 0xfe;
printf("End listen...\n");
RxECB.CCode == 0 ?
+
packets_received++
printf("Acknowledgment received.\n");
-
printf("Failed to receive acknowledgment.\n"();
fclose(file);
printf("File transfer complete.\n")
printf("\n--- Statistics ---\n");
printf("Packets Sent: %u\n", packets_send);
printf("Packets Received: %u\n", packets_received);
printf("Packets Lost: %u\n", packets_sent - packets_received);
IPXCloseSocket(&Socket);
exit(0);

Задание к работе

1. Разработать программу “Сервер”, которая посылает клиентам сети файл с использованием протокола IPX в среде DOS на языке программирования Pascal или C.
2. Разработать программу “Клиент”, которая принимает от сервера файл на языке программирования Pascal или C.
3. Провести анализ функционирования разработанных программ при передаче файла в формате *.jpg размером не менее 1 Мб (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

Настраиваем конфигурационный файл для использования протокола IPX в DosBox.

```
[ipx]
# ipx: Enable ipx over UDP/IP emulation.

ipx=true

core=auto
cputype=auto
cycles=max
cycleup=10
cycledown=20
```

Листинг программы:

IPX.H

```
#ifndef IPX_H
#define IPX_H

#include <dos.h>

#define IPX_CMD_OPEN_SOCKET             0x00
#define IPX_CMD_CLOSE_SOCKET            0x01
#define IPX_CMD_GET_LOCAL_TARGET        0x02
#define IPX_CMD_SEND_PACKET            0x03
#define IPX_CMD_LISTEN_FOR_PACKET       0x04
#define IPX_CMD_SCHEDULE_IPX_EVENT      0x05
#define IPX_CMD_CANCEL_EVENT            0x06
#define IPX_CMD_GET_INTERVAL_MARKER     0x08
#define IPX_CMD_GET_INTERNETWORK_ADDRESS 0x09
#define IPX_CMD_RELINQUISH_CONTROL      0x0A
#define IPX_CMD_DISCONNECT_FROM_TARGET  0x0B

#define NO_ERRORS               0
#define ERR_NO_IPX             1
#define ERR_NO_SPX             2
#define NO_LOGGED_ON           3
```

```
#define UNKNOWN_ERROR          0xFF

#define SHORT_LIVED            0
#define LONG_LIVED             0xFF
#define IPX_DATA_PACKET_MAXSIZE 546

struct IPXSPX_REGS {
    unsigned int ax;
    unsigned int bx;
    unsigned int cx;
    unsigned int dx;
    unsigned int si;
    unsigned int di;
    unsigned int es;
};

struct IPX_HEADER {
    unsigned int Checksum;
    unsigned int Length;
    unsigned char TransportControl;
    unsigned char PacketType;
    unsigned char DestNetwork[4];
    unsigned char DestNode[6];
    unsigned int DestSocket;
    unsigned char SourceNetwork[4];
    unsigned char SourceNode[6];
    unsigned int SourceSocket;
};

struct ECB {
    void far* Link;
    void far(*ESRAddress)(void);
    unsigned char InUse;
    unsigned char CCode;
    unsigned int Socket;
    unsigned int ConnectionId;
    unsigned int RestOfWorkspace;
    unsigned char DriverWorkspace[12];
    unsigned char ImmAddress[6];
    unsigned int FragmentCnt;
    struct {
        void far* Address;
        unsigned int Size;
    } Packet[2];
};

unsigned IntSwap(unsigned i);
int IPXOpenSocket(int SocketType, unsigned* Socket);
void IPXCloseSocket(unsigned* Socket);
void IPXListenForPacket(struct ECB* RxECB);
void IPXRelinquishControl(void);
void IPXSendPacket(struct ECB* TxECB);
void PrintECB(const struct ECB* ecb);

#endif // IPX_H
```

IPX.C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "IPX.H"

unsigned IntSwap(unsigned i) {
    return ((i >> 8) | (i & 0xFF) << 8);
}

int IPXOpenSocket(int socketType, unsigned* socket) {
    union REGS inregs, outregs;
    struct SREGS segregs;

    inregs.x.bx = IPX_CMD_OPEN_SOCKET;
    inregs.x.dx = IntSwap(*socket);
    inregs.x.ax = socketType;

    int86x(0x7A, &inregs, &outregs, &segregs);

    *socket = IntSwap(outregs.x.dx);
    return outregs.x.ax;
}

void IPXCloseSocket(unsigned* socket) {
    union REGS inregs, outregs;
    struct SREGS segregs;

    inregs.x.bx = IPX_CMD_CLOSE_SOCKET;
    inregs.x.dx = IntSwap(*socket);

    int86x(0x7A, &inregs, &outregs, &segregs);
}

void IPXListenForPacket(struct ECB* rxECB) {
    union REGS inregs, outregs;
    struct SREGS segregs;

    segregs.es = FP_SEG((void far*)rxECB);
    inregs.x.si = FP_OFF((void far*)rxECB);
    inregs.x.bx = IPX_CMD_LISTEN_FOR_PACKET;

    int86x(0x7A, &inregs, &outregs, &segregs);
}

void IPXSendPacket(struct ECB* txECB) {
    union REGS inregs, outregs;
    struct SREGS segregs;

    segregs.es = FP_SEG((void far*)txECB);
    inregs.x.si = FP_OFF((void far*)txECB);
    inregs.x.bx = IPX_CMD_SEND_PACKET;

    int86x(0x7A, &inregs, &outregs, &segregs);
```

```
}

void IPXRelinquishControl(void) {
    union REGS inregs, outregs;
    struct SREGS segregs;

    inregs.x.bx = IPX_CMD_RELINQUISH_CONTROL;

    int86x(0x7A, &inregs, &outregs, &segregs);
}

void PrintECB(const struct ECB* ecb) {
    unsigned int i;
    printf("ECB Structure:\n");
    printf("  Link: %Fp\n", ecb->Link); // %Fp для far-указателя
    printf("  ESRAddress: %Fp\n", ecb->ESRAddress); // %Fp для far-указателя
    printf("  InUse: 0x%02X\n", ecb->InUse);
    printf("  CCode: 0x%02X\n", ecb->CCode);
    printf("  Socket: 0x%04X\n", ecb->Socket);
    printf("  ConnectionId: 0x%04X\n", ecb->ConnectionId);
    printf("  RestOfWorkspace: 0x%04X\n", ecb->RestOfWorkspace);

    printf("  DriverWorkspace: ");
    i = 0;
    while(i < 12) {
    printf("%02X ", ecb->DriverWorkspace[i]);
    i++;
    }
    printf("\n");

    printf("  ImmAddress: ");
    i = 0;
    while(i < 6) {
    printf("%02X ", ecb->ImmAddress[i]);
    i++;
    }
    printf("\n");

    printf("  FragmentCnt: %u\n", ecb->FragmentCnt);
    i = 0;
    while (i < 2) {
    printf("  Packet[%d]:\n", i);
    printf("    Address: %Fp\n", ecb->Packet[i].Address); // %Fp для far-указателя
    printf("    Size: %u\n", ecb->Packet[i].Size);
    i++;
    }
}
```

SERVER.C:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "IPX.H"

#define BUFFER_SIZE 512
#define FILE_SIZE (1024 * 1024)

unsigned int packets_sent = 0;
unsigned int packets_received = 0;

void main(void) {
    static unsigned Socket = 0x4567;
    struct ECB TxECB;
    struct IPX_HEADER TxHeader;
    unsigned char TxBuffer[BUFFER_SIZE];
    FILE* file;
    size_t bytesRead;
    unsigned int packetNumber = 0;

    printf("\n*Server IPX*\n\n");

    if (IPXOpenSocket(LONG_LIVED, &Socket)) {
        printf("Opening socket error!\n");
        exit(-1);
    }

    file = fopen(".\\image.jpg", "rb");
    if (!file) {
    printf("Failed to open file.\n");
    IPXCloseSocket(&Socket);
    exit(-1);
    }

    printf("socket: %x swapped=%x\n", Socket, IntSwap(Socket));

    memset(&TxECB, 0, sizeof(TxECB));
    TxECB.Socket = IntSwap(Socket);
    TxECB.FragmentCnt = 2;
    TxECB.Packet[0].Address = &TxHeader;
    TxECB.Packet[0].Size = sizeof(TxHeader);
    TxECB.Packet[1].Address = TxBuffer;
    TxECB.Packet[1].Size = BUFFER_SIZE;

    memset(TxECB.ImmAddress, 0xff, 6);

    TxHeader.PacketType = 4;
    memset(TxHeader.DestNetwork, 0, 4);
    memset(TxHeader.DestNode, 0xff, 6);
    TxHeader.DestSocket = IntSwap(Socket);
```

```
    while ((bytesRead = fread(TxBuffer, 1, BUFFER_SIZE, file)) > 0) {
    printf("Sending packet:\n");
    PrintECB(&TxECB);

    IPXSendPacket(&TxECB);
    packets_sent++;
    printf("Packet %d sent (%u bytes).\n", packetNumber++, bytesRead);

    {
        struct ECB RxECB;
        struct IPX_HEADER RxHeader;
        unsigned char RxBuffer[BUFFER_SIZE];

        memset(&RxECB, 0, sizeof(RxECB));
        RxECB.Socket = IntSwap(Socket);
        RxECB.FragmentCnt = 2;
        RxECB.Packet[0].Address = &RxHeader;
        RxECB.Packet[0].Size = sizeof(RxHeader);
        RxECB.Packet[1].Address = RxBuffer;
        RxECB.Packet[1].Size = BUFFER_SIZE;

        IPXListenForPacket(&RxECB);

        printf("Waiting for client acknowledgment...\n");
        while (RxECB.InUse) {
        IPXRelinquishControl();
        if (kbhit()) {
            getch();
            RxECB.CCode = 0xfe;
            break;
        }

    delay(30);
        }

        printf("End listen...\n");
        PrintECB(&RxECB);

        if (RxECB.CCode == 0) {
        packets_received++;
                printf("Acknowledgment received.\n");
            }
            else {
                printf("Failed to receive acknowledgment.\n");
                break;
            }
        }
    }

    fclose(file);
    printf("File transfer complete.\n");

    printf("\n--- Statistics ---\n");
    printf("Packets Sent: %u\n", packets_sent);
    printf("Packets Received: %u\n", packets_received);
```

```
    printf("Packets Lost: %u\n", packets_sent - packets_received);

    IPXCloseSocket(&Socket);
    exit(0);
}
```

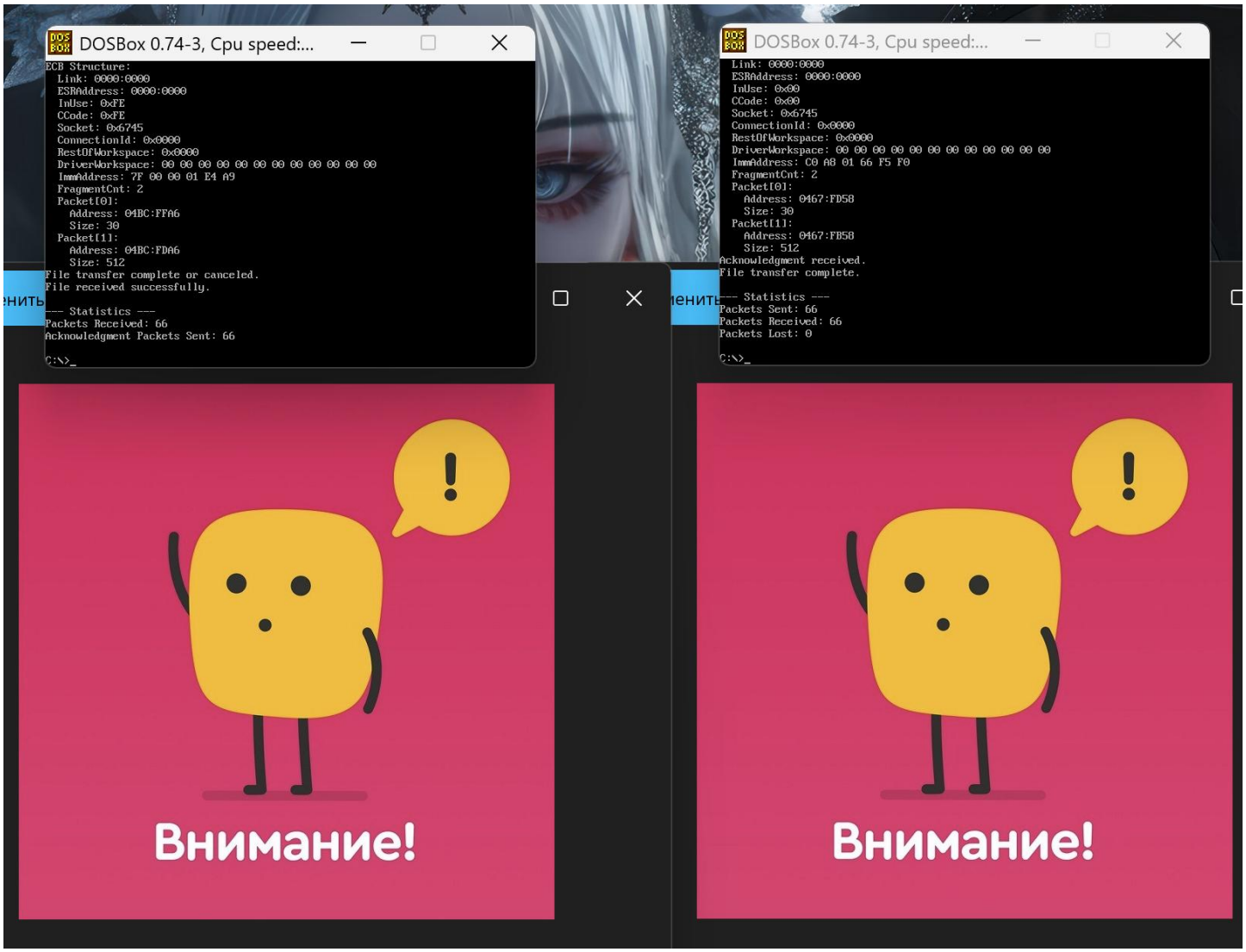
CLIENT.C:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "IPX.H"

#define BUFFER_SIZE 512
#define FILE_SIZE (1024 * 1024)

unsigned int packets_received = 0;
unsigned int packets_sent = 0;

void main(void) {
    static unsigned Socket = 0x4567;
    struct ECB RxECB;
    struct IPX_HEADER RxHeader;
    unsigned char RxBuffer[BUFFER_SIZE];
    FILE* file;
    size_t bytesWritten;
    unsigned int packetNumber = 0;
    char filename[20];

    printf("\n*Client IPX*\n\n");

    if (IPXOpenSocket(LONG_LIVED, &Socket)) {
        printf("Opening socket error.\n");
        exit(-1);
    }

    srand(time(NULL));
    sprintf(filename, "recimg_%d.jpg", rand() % 100 + 1);
    file = fopen(filename, "wb");
    if (!file) {
        printf("Failed to create file.\n");
        IPXCloseSocket(&Socket);
        exit(-1);
    }

    printf("socket: %x swapped=%x\n", Socket, IntSwap(Socket));

    memset(&RxECB, 0, sizeof(RxECB));
    RxECB.Socket = IntSwap(Socket);
    RxECB.FragmentCnt = 2;
    RxECB.Packet[0].Address = &RxHeader;
    RxECB.Packet[0].Size = sizeof(RxHeader);
```

```
    RxECB.Packet[1].Address = RxBuffer;
    RxECB.Packet[1].Size = BUFFER_SIZE;

    PrintECB(&RxECB);

    printf("Waiting for file transfer...\n");

    while (1) {
        IPXListenForPacket(&RxECB);
        printf("Listen...\n");

        while (RxECB.InUse) {
            IPXRelinquishControl();
            if (kbhit()) {
                getch();
                RxECB.CCode = 0xfe;
                break;
        }

            delay(30);
        }

        printf("End listen...\n");
        PrintECB(&RxECB);

        if (RxECB.CCode == 0) {
            bytesWritten = fwrite(RxBuffer, 1, BUFFER_SIZE, file);
            if (bytesWritten < BUFFER_SIZE) {
                printf("File write error.\n");
                break;
            }

            packets_received++;
            printf("Packet %d received (%u bytes).\n", packetNumber++, bytesWritten);

            {
                struct ECB TxECB;
                struct IPX_HEADER TxHeader;
                unsigned char TxBuffer[BUFFER_SIZE];

                memset(&TxECB, 0, sizeof(TxECB));
                TxECB.Socket = IntSwap(Socket);
                TxECB.FragmentCnt = 2;
                TxECB.Packet[0].Address = &TxHeader;
                TxECB.Packet[0].Size = sizeof(TxHeader);
                TxECB.Packet[1].Address = TxBuffer;
                TxECB.Packet[1].Size = BUFFER_SIZE;

                TxHeader.PacketType = 4;
                memcpy(TxHeader.DestNode, RxECB.ImmAddress, 6);
                TxHeader.DestSocket = IntSwap(Socket);

                printf("Packet for send:\n");
                PrintECB(&RxECB);
```

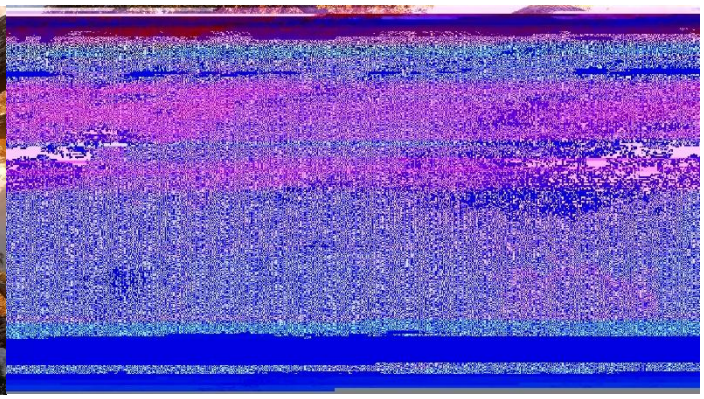
```
                strcpy(TxBuffer, "ACK");
                IPXSendPacket(&TxECB);
                packets_sent++;
            }
        }
        else {
            printf("File transfer complete or canceled.\n");
            break;
        }
    }

    fclose(file);
    printf("File received successfully.\n");

    printf("\n--- Statistics ---\n");
    printf("Packets Received: %u\n", packets_received);
    printf("Acknowledgment Packets Sent: %u\n", packets_sent);

    IPXCloseSocket(&Socket);
    exit(0);
}
```

Результат работы:



Если количество клиентов 2, то изображения передаются с помехами:

Оригинал: Переданное:

При числе клиентов 3 и выше, изображение невозможно открыть (выводит, что изображение повреждено)

Анализ широковещательной рассылки всем пользователям

Размер файла	Число пакетов	Число получателей	Число утерянных пакетов	Процент потерь	Время
1.81 МБ (1 898 957 байт)	3709	1	0	0%	4:20
	7418	2	68	0.92%	3:40
	14836	4	675	4.55%	3:05
	Невозможно подключить ещё устройства	5	-	-	-

По мере увеличения кол-ва клиентов кол-во потерянных пакетов растет

Вывод: в ходе работы были изучены и применены на практике основные функции API драйвера для работы с протоколом сетевого уровня IPX. Получены навыки использования IPX. Проведены замеры, показывающие, что при использовании IPX возможны потери пакетов. Чем больше клиентов – тем больше потерь пакетов. И чем больше клиентов, тем меньше время передачи.