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)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Сети ЭВМ и телекоммуникации

Лабораторная работа №1

**Протокол сетевого уровня IPX**

Выполнил:

ст. гр. ВТ-31

Клесов М.И.

Проверил:

Федотов Е.А.

Белгород, 2020

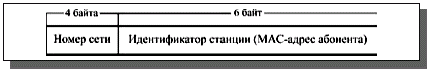
**Цель работы**: изучить протокол сетевого уровня IPX, основные функции API драйвера IPX и разработать программу для приема/передачи данных

Ход работы

Краткие теоретические сведения

Протокол IPX – это протокол сетевого уровня модели взаимодействия открытых систем (OSI) реализующий передачу пакетов (сообщений) между станциями сети на уровне датаграмм (доставка не гарантируется).

Адрес в протоколе IPX состоит из трёх компонент: номер сети, адрес станции в сети и сокет.

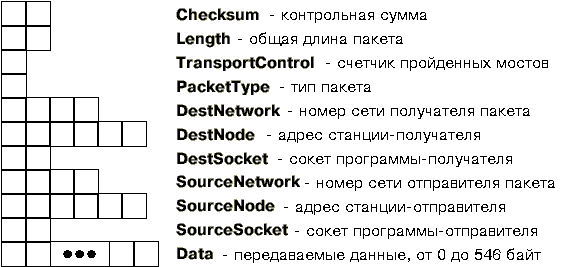


Номер сети – это код, присвоенный каждой конкретной сети, то есть каждой широковещательной области общей, единой сети. 00:00:00:00 – адрес локальной сети.

Адрес станции - это число, которое является уникальным для каждой рабочей станции. При использовании адаптеров Ethernet уникальность обеспечивается изготовителем сетевого адаптера (MAC), FF:FF:FF:FF:FF:FF – широковещательный адрес.

Сокет – число, идентиицирующее программу на рабочей станции.

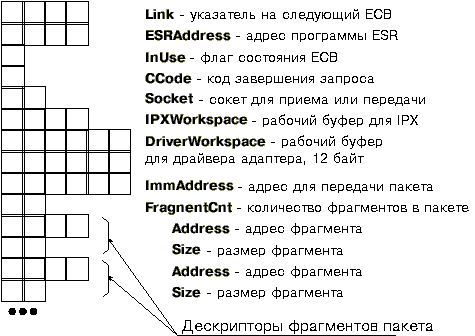
Структура IPX:



Для приема или передачи пакета прикладная программа должна подготовить пакет данных, сформировав его заголовок, и построить так называемый блок управления событием ECB (Event Control Block). В блоке ECB задается адресная информация для передачи пакета, адрес самого передаваемого пакета в оперативной памяти и некоторая другая информация.

Подготовив блок ECB, прикладная программа передает его адрес соответствующей функции IPX для выполнения операции приема или передачи пакета.

Структура ECB:



Основные API

* Int IPXOpenSocket(int SocketType, unsigned \*Socket)

Входные параметры: тип сокета (долгоживущий, короткоживущий), номер сокета

Выходные парметры: если в Socket передано нулевое значение, то через этот параметр возвращается динамический номер сокета.

Назначение: открывает сокет типа Type с номером Socket или динамический сокет (если Socket = 0000h). Возвращает код завершения: 00h - сокет открыт; FFh - этот сокет уже был открыт раньше; FEh - переполнилась таблица сокетов.

* void IPXCloseSocket(unsigned\* Socket)

Входные параметры: номер сокета.

Назначение: закрывает сокет с номером Socket.

* void IPXListenForPacket(struct ECB \*RxECB)

Входные парметры: блок ECB.

Назначение: запрос к драйверу IPX для ожидания получения пакета и запись результата в ECB.

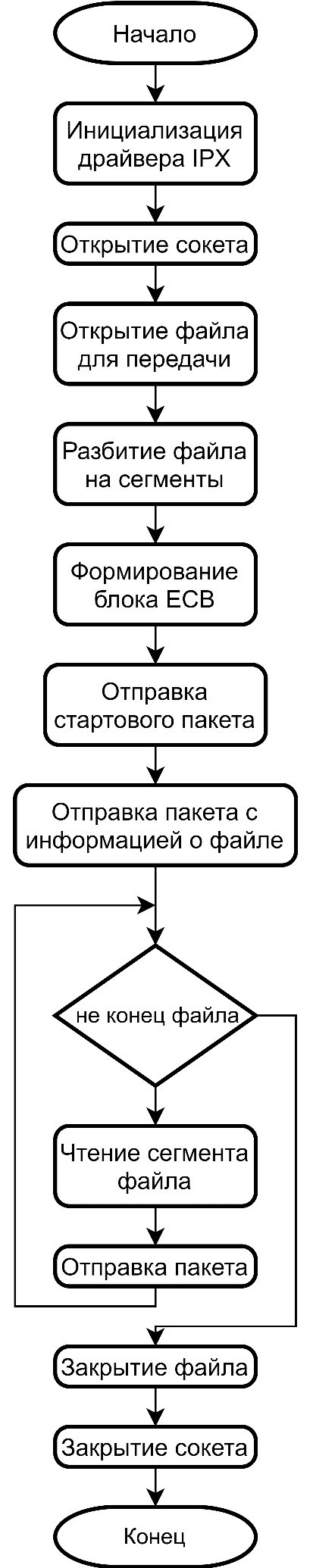
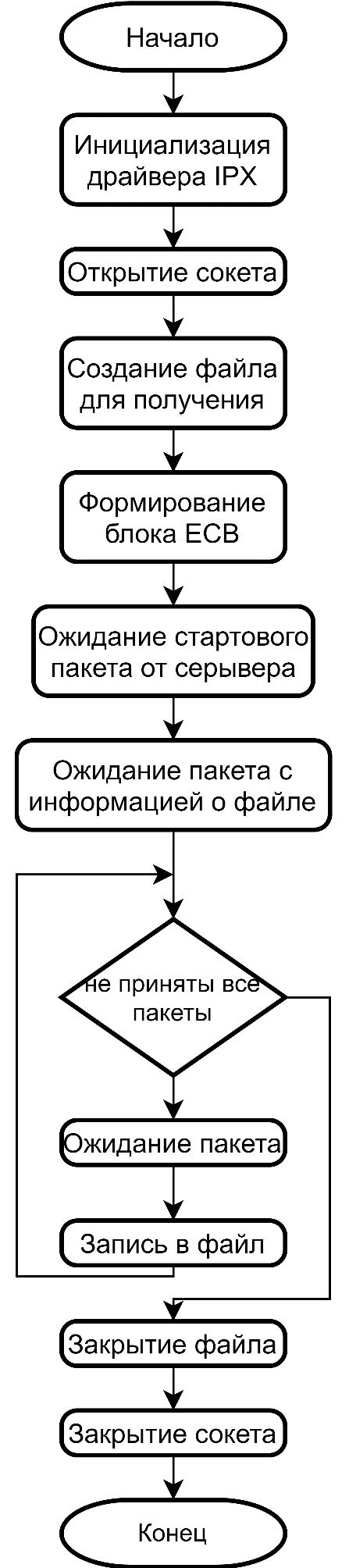
* void IPXSendPacket(struct ECB \*TxECB)

Входные параметры: блок ECB.

Назначение: подготавливает блок ECB и связанный с ним заголовок пакета для передачи пакета по сети.

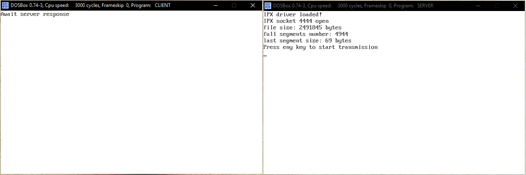
Блок-схемы

Блок схема программы сервер: Блок схема программы клиент:

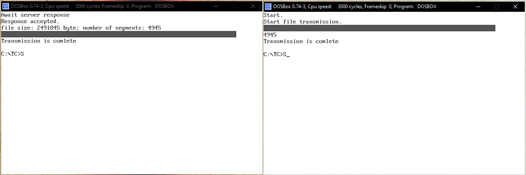
 

Анализ функционирования работы программ и примеры работы

Програмы клиент и сервер до передачи файла:

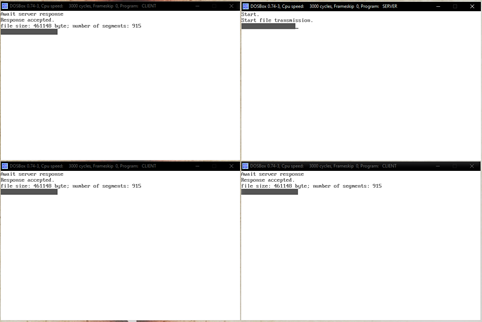


После передачи файла:



Исходное и полученное изображения с задержкой между отправкой пакетов равной 20мс совпадают. При задержке равной 10 мс изображение так же передаётся без искажений. При сокращении задержки до 5 мс и 0 скорость передачи значительно возрастает, но сама передача заканчивается неудачей - изображение получено не полностью и не может быть прочитано.

Передача изображения нескольким клиентам:



Все клиенты получили файл без искажений.

Тексты программ

Код программы клиента (**client.c**):

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <mem.h>

#include <string.h>

#include "ipx.h"

#define BUFFER\_SIZE 504

typedef struct \_file\_segment

{

long int s\_size;

long int s\_num;

} file\_segment;

void main(void) {

long int file\_size;

long int seg\_num;

FILE\* file;

file\_segment tmp\_seg;

int progress\_step;

int i, k;

static unsigned Socket = 0x4444;

struct ECB RxECB;

struct IPX\_HEADER RxHeader;

unsigned char RxBuffer[BUFFER\_SIZE];

char fname[9] = "img.jpg";

file = fopen(fname,"wb");

if(file == NULL){

exit(-1);

}

if(ipx\_init() != 0xff) {

printf("IPX not loaded!\n");

exit(-1);

}

if(IPXOpenSocket(SHORT\_LIVED, &Socket)) {

printf("Socket open error\n");

exit(-1);

};

clrscr();

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

RxECB.Socket = IntSwap(Socket);

RxECB.FragmentCnt= 3;

RxECB.Packet[0].Address = &RxHeader;

RxECB.Packet[0].Size = sizeof(RxHeader);

RxECB.Packet[1].Address = RxBuffer;

RxECB.Packet[1].Size = BUFFER\_SIZE;

RxECB.Packet[2].Address = &tmp\_seg;

RxECB.Packet[2].Size = 8;

//ожидание первого пакета

IPXListenForPacket(&RxECB);

printf("Await server response");

while (RxECB.InUse);

printf("\nResponse accepted.\n");

if (tmp\_seg.s\_num != -1) exit(2);

//ожидание пакета с размером файла

IPXListenForPacket(&RxECB);

while (RxECB.InUse);

if (tmp\_seg.s\_num != 0) exit(2);

memcpy(&file\_size, RxBuffer, sizeof(file\_size));

seg\_num = file\_size / BUFFER\_SIZE + 1;

printf("file size: %ld byte; number of segments: %ld\n", file\_size, seg\_num);

progress\_step = seg\_num/70-1;

for(i = 0; i < seg\_num; i++){

IPXListenForPacket(&RxECB);

while (RxECB.InUse);

for (k = 0; k < tmp\_seg.s\_size; k++) {

fputc(RxBuffer[k], file);

};

if(i % progress\_step == 0)

printf("%c", 219);

if(tmp\_seg.s\_num == seg\_num)

break;

}

printf("\nTransmission is comlete\n");

fclose(file);

IPXCloseSocket(&Socket);

exit(0);

}

Код программы сервера (**server.c**):

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <mem.h>

#include <string.h>

#include <dos.h>

#include "ipx.h"

#define BUFFER\_SIZE 504

typedef struct \_file\_segment{

long int s\_size;

long int s\_num;

} file\_segment;

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 main(void) {

long int file\_size; //размер файла в байтах

long int full\_seg\_num, seg\_num; //кол-во полных сегментов, всего сегментов

long int last\_seg\_size; //размер неполного последнего сегмента (если 0 то сегмента нет)

FILE \*file;

file\_segment tmp\_seg;

static unsigned Socket = 0x4444;

struct ECB ServerECB;

struct IPX\_HEADER InHeader, OutHeader;

unsigned char OutBuffer[BUFFER\_SIZE];

int i, progress\_step;

char filename[255] = "test\_img.jpg";

int value\_delay = 10;

clrscr();

if (ipx\_init() != 0xff) {

printf("IPX driver not loaded!\n");

exit(-1);

}

else {

printf("IPX driver loaded!\n");

}

if(IPXOpenSocket(SHORT\_LIVED, &Socket)) {

printf("IPX socket open error\n");

exit(-1);

}else{

printf("IPX socket %x open\n", Socket);

};

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

//открытие файла и рассчет количества сегментов

file=fopen(filename,"rb+");

if(file!=NULL){

file\_size = get\_file\_size(file);

full\_seg\_num = file\_size / (BUFFER\_SIZE);

last\_seg\_size = file\_size % (BUFFER\_SIZE);

seg\_num = full\_seg\_num + 1;

printf("file size: %ld bytes\n", file\_size);

printf("full segments number: %ld\n", full\_seg\_num);

printf("last segment size: %ld bytes\n", last\_seg\_size);

}else{

printf("File open error!\n");

exit(1);

};

printf("Press eny key to start transmission\n");

getch();

clrscr();

//Пакет "начало передачи"

tmp\_seg.s\_size = 0;

tmp\_seg.s\_num = -1;

memset(OutBuffer, 0, BUFFER\_SIZE);

//заголовок

OutHeader.PacketType = 4; // IPX

memset(OutHeader.DestNetwork, 0, 4); // локальная сеть

memset(OutHeader.DestNode, 0xFF, 6); // всем узлам сети

OutHeader.DestSocket = IntSwap(Socket);

//заполнение ECB

ServerECB.Socket = IntSwap(Socket);

ServerECB.FragmentCnt = 3;

ServerECB.Packet[0].Address = &OutHeader;

ServerECB.Packet[0].Size = sizeof(OutHeader);

ServerECB.Packet[1].Address = OutBuffer;

ServerECB.Packet[1].Size = BUFFER\_SIZE;

ServerECB.Packet[2].Address = &tmp\_seg;

ServerECB.Packet[2].Size = 8;

IPXSendPacket(&ServerECB);

printf("Start.\n");

//Пакет размер файла

tmp\_seg.s\_num = 0;

tmp\_seg.s\_size = sizeof(seg\_num);

memcpy(OutBuffer, &file\_size, sizeof(file\_size));

IPXSendPacket(&ServerECB);

progress\_step = seg\_num/70-1;

printf("Start file transmission.\n");

delay(100);

for(i = 1; i <= full\_seg\_num; i++){

tmp\_seg.s\_num = i;

tmp\_seg.s\_size = BUFFER\_SIZE;

fread(OutBuffer, 1, BUFFER\_SIZE, file);

if(i % progress\_step == 0) printf("%c", 219);

IPXSendPacket(&ServerECB);

delay(value\_delay);

};

tmp\_seg.s\_num = seg\_num;

tmp\_seg.s\_size = last\_seg\_size;

fread(OutBuffer, 1, last\_seg\_size, file);

IPXSendPacket(&ServerECB);

printf("\n%i\n", i);

printf("Transmission is comlete\n");

fclose(file);

IPXCloseSocket(&Socket);

exit(0);

}

**Файл “ipx.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

void far ipxspx\_entry(void far \*ptr);

int ipx\_init(void);

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 RrestOfWorkspace;

unsigned char DriverWorkspace[12];

unsigned char ImmAddress[6];

unsigned int FragmentCnt;

struct {

void far \*Address;

unsigned int Size;

} Packet[3];

};

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);

**Файл “ipx.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) {

struct IPXSPX\_REGS iregs;

iregs.bx = IPX\_CMD\_OPEN\_SOCKET;

iregs.dx = IntSwap(\*Socket);

iregs.ax = SocketType;

ipxspx\_entry((void far \*)&iregs);

\*Socket = IntSwap(iregs.dx);

return(iregs.ax);

}

void IPXCloseSocket(unsigned \*Socket) {

struct IPXSPX\_REGS iregs;

iregs.bx = IPX\_CMD\_CLOSE\_SOCKET;

iregs.dx = IntSwap(\*Socket);

ipxspx\_entry( (void far \*)&iregs );

}

void IPXListenForPacket(struct ECB \*RxECB) {

struct IPXSPX\_REGS iregs;

iregs.es = FP\_SEG((void far\*)RxECB);

iregs.si = FP\_OFF((void far\*)RxECB);

iregs.bx = IPX\_CMD\_LISTEN\_FOR\_PACKET;

ipxspx\_entry( (void far \*)&iregs );

}

void IPXSendPacket(struct ECB \*TxECB) {

struct IPXSPX\_REGS iregs;

iregs.es = FP\_SEG((void far\*)TxECB);

iregs.si = FP\_OFF((void far\*)TxECB);

iregs.bx = IPX\_CMD\_SEND\_PACKET;

ipxspx\_entry( (void far \*)&iregs );

}

void IPXRelinquishControl(void) {

struct IPXSPX\_REGS iregs;

iregs.bx = IPX\_CMD\_RELINQUISH\_CONTROL;

ipxspx\_entry( (void far \*)&iregs );

}

**Файл “ipx\_init.asm”:**

.286

.MODEL SMALL

; Структура для вызова драйвера

IPXSPX\_REGS struc

rax dw ?

rbx dw ?

rcx dw ?

rdx dw ?

rsi dw ?

rdi dw ?

res dw ?

IPXSPX\_REGS ends

.DATA

; Точка входа в драйвер

\_ipxspx\_drv\_entry dd ?

.CODE

;Обьявление процедур

PUBLIC \_ipxspx\_entry, \_ipx\_init

PUBLIC \_ipxspx\_drv\_entry

;Вызов драйвера

\_ipxspx\_entry PROC FAR

;Переносим адрес вершины стека в bp для обращения к аргументам

push bp

mov bp,sp

;Сохраняем регистры в стек

push es

push di

push si

push dx

push cx

push bx

push ax

;Загружаем регистры из структуры, которая передана как аргумент функции

push ds

mov bx, [bp+6] ;смещение

mov ds, [bp+8] ;сегмент

mov es, ds:[bx].res

mov di, ds:[bx].rdi

mov si, ds:[bx].rsi

mov dx, ds:[bx].rdx

mov cx, ds:[bx].rcx

mov ax, ds:[bx].rax

mov bx, ds:[bx].rbx

pop ds

;Вызываем драйвер

call [dword ptr \_ipxspx\_drv\_entry]

;Сохраняем регистры

push ds

push dx

mov dx, bx

;Записываем в структуры возвращенные драйвером

mov bx, [bp+6] ;смещение

mov ds, [bp+8] ;сегмент

mov ds:[bx].rax, ax

mov ds:[bx].rcx, cx

mov ds:[bx].rbx, dx

pop dx

mov ds:[bx].rdx, dx

pop ds

; Восстанавливаем регистры

pop ax

pop bx

pop cx

pop dx

pop si

pop di

pop es

pop bp

retf

\_ipxspx\_entry ENDP

;Инициализация драйвера и получение точки входа

\_ipx\_init PROC NEAR

;Сохранение вершины стека

push bp

mov bp,sp

;Переносим в ax значение отвечающее за драйвер IPX

mov ax, 7a00h

;Вызываем мультиплексорное прерывание

int 2fh

;Прерывание возращает в al точку входа в драйвер или 0xFF если драйвер не обнаружен.

cmp al, 0ffh

jne \_ipx\_init\_exit

;Сохраняем адрес точки входа в переменную

mov word ptr \_ipxspx\_drv\_entry+2, es

mov word ptr \_ipxspx\_drv\_entry, di

\_ipx\_init\_exit:

;В регистре ax прерывание помещает код выхода

mov ah, 0

pop bp

ret

\_ipx\_init ENDP

end