Лабораторная работа #1

Изучение протокола IPX

Выполнил: студент группы ВТ-31 Солодилов М.А

Проверил: Федотов Е.А

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнение:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

БГТУ 2016

**Протокол сетевого уровня IPX**

**Цель работы:** изучить протокол сетевого уровня IPX, основные

функции API драйвера IPX и разработать программу для

приема/передачи данных.

**Краткие теоретические сведения**

Протокол IPX – это протокол сетевого уровня моделивзаимодействия открытых систем (OSI) реализующий передачупакетов (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.

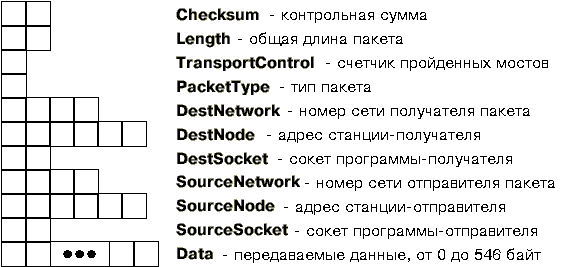
**Номер сети** - это номер сегмента сети, определяемого системным администратором. Если в общей сети есть мосты, каждая отдельная сеть, подключенная через мост, должна иметь свой, уникальный номер

сети.

**Адрес станции** - это число, которое является уникальным для каждой рабочей станции. При использовании адаптеров Ethernet уникальность обеспечивается изготовителем сетевого адаптера.

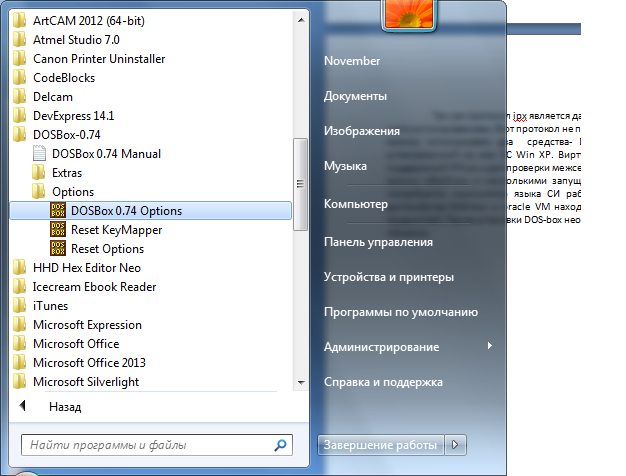
Специальный адрес FFFFFFFFFFFFh используется для рассылки данных всем станциям данной сети одновременно.

**Идентификатор программы на рабочей станции (сокет)** -число, которое используется для адресации определенной программы, работающей на компьютере. В среде мульти задачных операционныхсистем, на каждой рабочей станции в сети одновременно может бытьз апущено 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

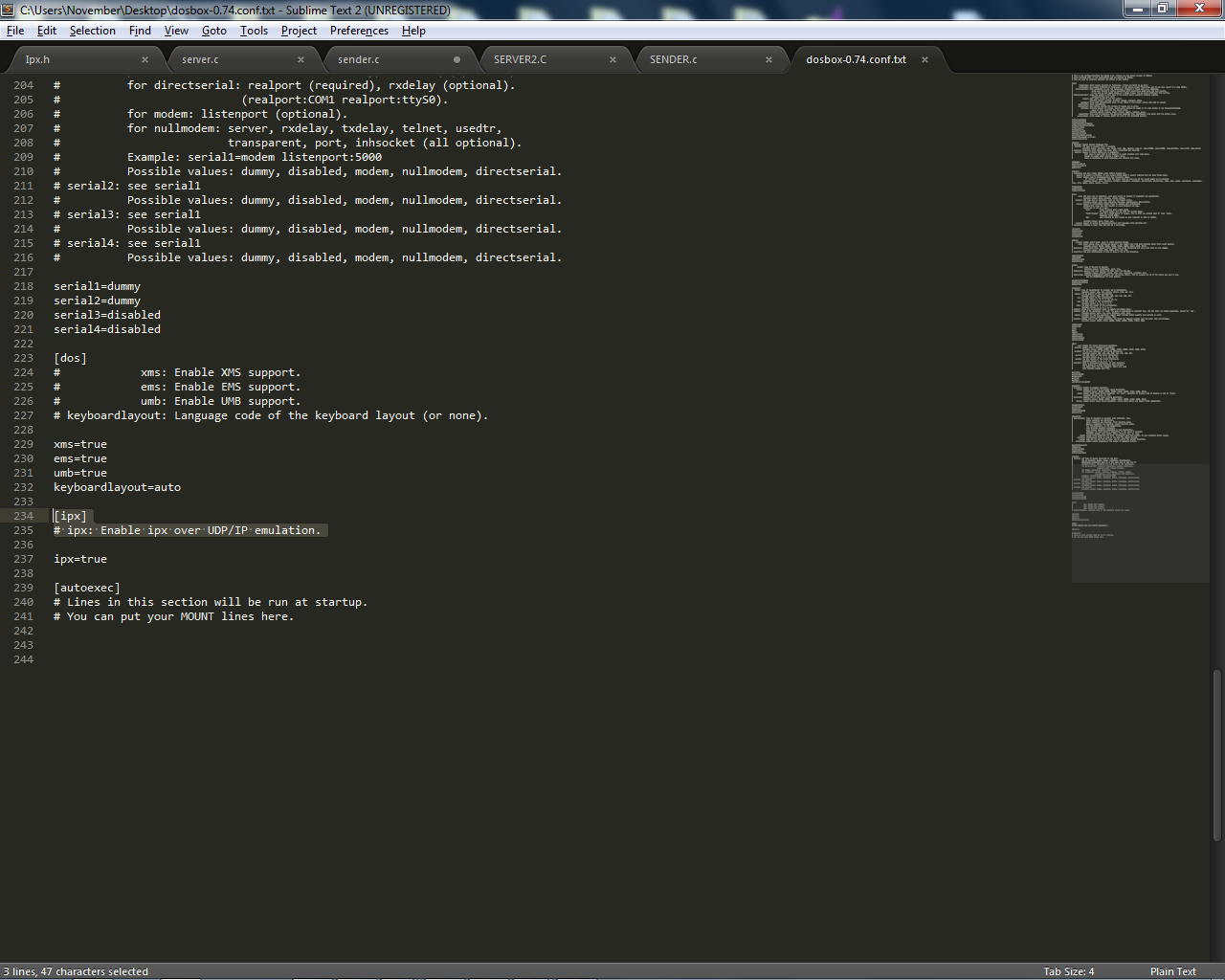


**Ход выполнения работы**

Так как протокол ipx является далеко не самым новым протоколом, возникают определенные трудности с его использованием. Этот протокол не поддерживается версиями Windows выше XP. Для решения проблемы можно использовать два средства- DOS BOX и виртуальную машину(я использовал машину Oracle с установленной на нее ОС Win XP. Виртуальная машина пригодится нам не только для запуска приложений с поддержкой IPX но и для проверки межсетевого взаимодействия написанных программ(что весьма удобно, хотя можно обойтись и несколькими запущенными версиями DOS-box. Начнем с подготовительного этапа- нам потребуется компилятор языка СИ работающий из под DOS(Turbo-C отлично подойдет для этих целей), дистрибутив DOS-box и oracle VM находятся в свободном доступе, их загрузка и установка не представляют трудностей. После установки DOS-box необходимо внести изменения в его конфигурационный файл следующим образом.



**Рис.1**



**Рис.2**

По умолчанию в настройках Dos-box выключена эмуляция IPX. Знание этой тонкости может сэкономить очень и очень много времени.

Теперь перейдем непосредственно к самой программе и разбору ее функционирования. Функции драйвера IPX вызываются через программное прерывание которое может быть вызвано либо ассемблерной вставкой либо функцией библиотеки DOS.H int86. В библиотеке DOS.H описаны структуры и заголовки для работы с регистрами процессора и для вызова прерываний, список прерываний и параметров для их вызова будет приведен в приложении. Для начала разберем небольшую программу проверяющую наличие драйвера IPX

#include <dos.h>

#include <stdio.h>

#include "Ipx.h"

void main()

{

union REGS inregs,outregs;

inregs.x.ax=0x7A00;

int86(IPX\_interrupt,&inregs,&outregs);

clrscr();

printf("%x\n",outregs.h.al);

system("pause");

getch();

return;

}

В данной программе вызывается прерывание ,в регистр ax загружено значение 0x7a00 ,если по завершению работы функции в регистре al будет значение ff- все отлично, драйвер загружен. В противном случае необходимо найти причину его отсутствия (ошибка в конфигурационном файле или вовсе его отсутствие в ОС). Если драйвер не загружен ,то лучше не пытаться выполнить какие либо другие функции связанные с драйвером IPX , это может повлечь разного рода последствия.  
  
Следующий участок кода отвечает за пример сообщения:

#include <stdio.h>

#include <string.h>

#include <dos.h>

#include <ipx.h>

#define TargetSocket 0x4567

void main ()

{

unsigned char RecieveString [80];

IPXHeader Header; //Заголовок

ECB\_2 My\_ECB;

ECB\_2 far \*My\_ECB\_ptr; //указатель на блок ECB

union REGS inregs, outregs //объединения входных/выходных значений регистров;

struct SREGS segreg; //структура для хранения сегмента смещения

//Заполняем блок ECB,здесь ничего необычного

My\_ECB.ESRAddress = NULL;

My\_ECB.SocketNumber = TargetSocket;

My\_ECB.FragmentCount = 2;

My\_ECB.FragmentAddress\_1 = &Header;

My\_ECB.FragmentSize\_1 = header\_length;

My\_ECB.FragmentAddress\_2 = &RecieveString;

My\_ECB.FragmentSize\_2 = sizeof RecieveString;

//открываем сокет для приема файла

inregs.x.bx = IPX\_open;

inregs.h.al = 0xFF; //параметр FF означает что сокет требует закрытия вручную

inregs.x.dx = TargetSocket; //в регистр DX записывается номер открываемого сокета в

int86 (IPX\_interrupt, &inregs, &outregs); //обратном порядке,затем вызываем прерывание

if (outregs.h.al != 0x00) //если в результате в регистер al 0x00-произошла ошибка

printf("Socket opening error,exit code:%x\n",outregs.h.al);

else printf("Socket opening succes!");

inregs.x.bx = IPX\_listen; //формируем значения регистров для получения пакета

My\_ECB\_ptr = &My\_ECB; //IPX\_listen-константа описанная в IPX.H

inregs.x.si = FP\_OFF ( My\_ECB\_ptr ); //в регистровую пару es:si записываем сегмент и

segreg.es = FP\_SEG ( My\_ECB\_ptr ); //смещение адреса блока ECB

int86x (IPX\_interrupt, &inregs, &outregs, &segreg);//и вызываем прерывание

printf("wait for packet"); //ожидаем пакет

while (My\_ECB.InUseFlag)

{};

if (My\_ECB.CompletionCode != 0) //если код завершения не равен нулю значит произошла ошибка

printf("Message recieving error,exit code: %x\n",My\_ECB.CompletionCode);

else

printf ("%s \n", RecieveString); //иначе просто выводим сообщение

inregs.x.bx = IPX\_close; //и закрываем сокет так же используя прерывание

inregs.x.dx = TargetSocket;

int86 (IPX\_interrupt, &inregs, &outregs);

}

Теперь рассмотрим код приложения клиента:

#include <stdio.h>

#include <string.h>

#include <dos.h>

#include <ipx.h> //заголовочный файл с константами

#define TargetSocket 0x4567 //целевой сокет

void main ()

{

unsigned char DispString [80]; //вводимая строка

IPXHeader Header;

ECB\_2 My\_ECB;

ECB\_2 far \*My\_ECB\_ptr;

int Length;

union REGS inregs, outregs;

struct SREGS segreg;

printf ("Input your message \n");

gets(DispString); //ввод строки

Length = strlen (DispString);

printf("your message : %s",DispString);

memset(&My\_ECB,0,sizeof(My\_ECB)); //обнуление блока ECB

My\_ECB.ESRAddress = NULL; //ESR=NULL так как не используются другие блоки

My\_ECB.SocketNumber = TargetSocket; //Заполняем сокет,число пакетов,адрес в памяти и размер

My\_ECB.FragmentCount = 2;

My\_ECB.FragmentAddress\_1 = &Header;

My\_ECB.FragmentSize\_1 = header\_length;

My\_ECB.FragmentAddress\_2 = &DispString;

My\_ECB.FragmentSize\_2 = Length+1;

memset(My\_ECB.ImmediateAddress,0xff,6); //заполняем адрес ,ffffffff для широкополосного вещания

memset(Header.DestantionNetwork,0,4); //сеть =0 т.к используем локальную сеть

memset(Header.DestantionNode,0xFF,6); //узел ffffff для использования всех узлов

Header.DestantionSocket = TargetSocket; //сокет=целевой сокет

Header.PacketType = 0;

inregs.x.si = FP\_OFF ( My\_ECB\_ptr ); //заполняем адрес ECB в ES;SI

inregs.x.bx = IPX\_send; //сегмент и смещение

My\_ECB\_ptr = &My\_ECB;

segreg.es = FP\_SEG ( My\_ECB\_ptr );

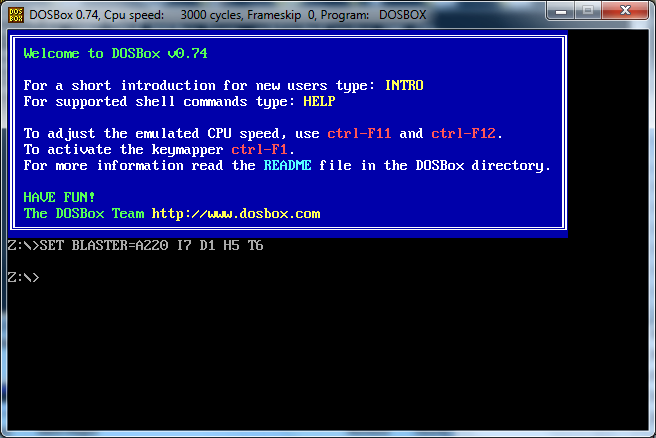
int86x (IPX\_interrupt, &inregs, &outregs, &segreg);

while (My\_ECB.InUseFlag){};

printf ("Message has been sended,exit code: %x\n",My\_ECB.CompletionCode); //вывод сообщения и

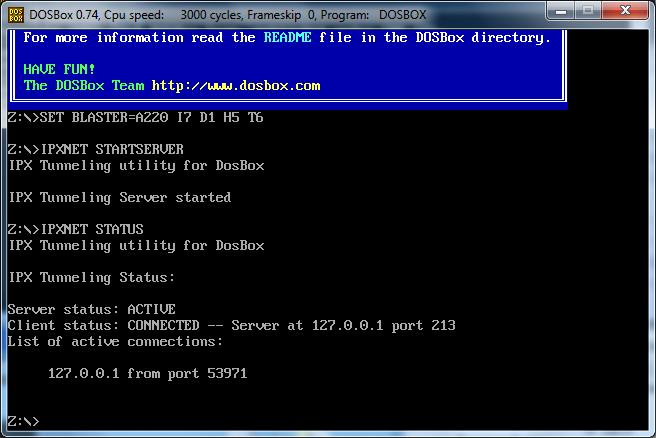
//кода завершения

}

Рассмотрим процесс работы программы, запускаем 2 процесса dos-box(клиент и сервер)  


**Рис.3**

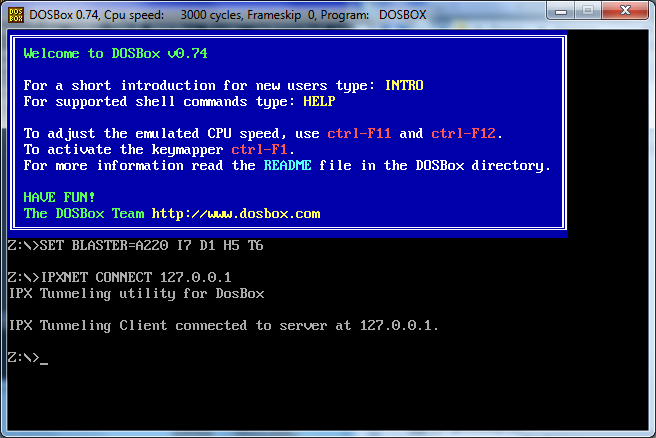
Введем команду IPXNET STARTSERVER для инициализации сервера на одном клиенте



**Рис.4**

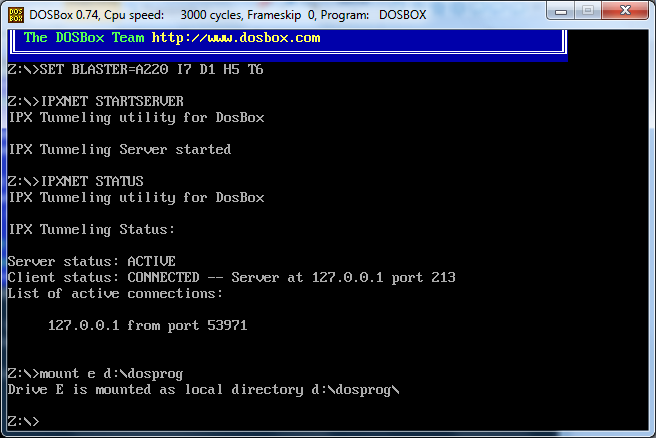
Можно запускать виртуальную машину и после настройки локальной сети подключить клиент к серверу или же подключить один процесс к другому на одной машине, сперва рассмотрим пример работы на одной машине.

Запустим еще один экземпляр dos-box и выполним подключение к уже существующему серверу



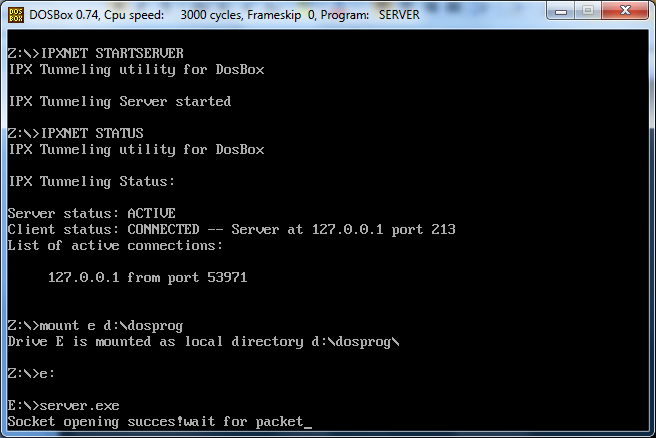
**Рис.5**

Монтируем области диска :

****

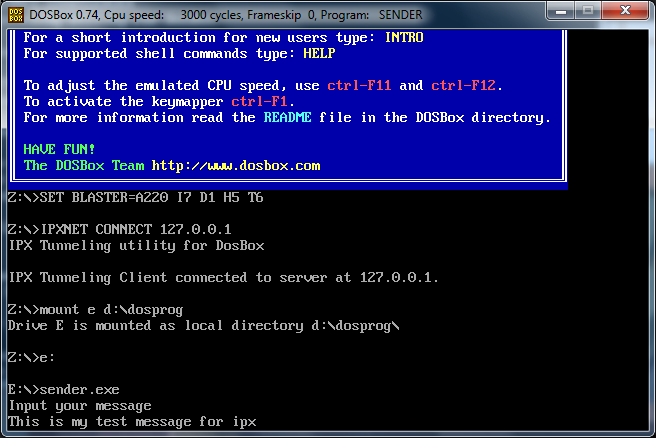
**Рис.6**

Запускаем приложение сервер

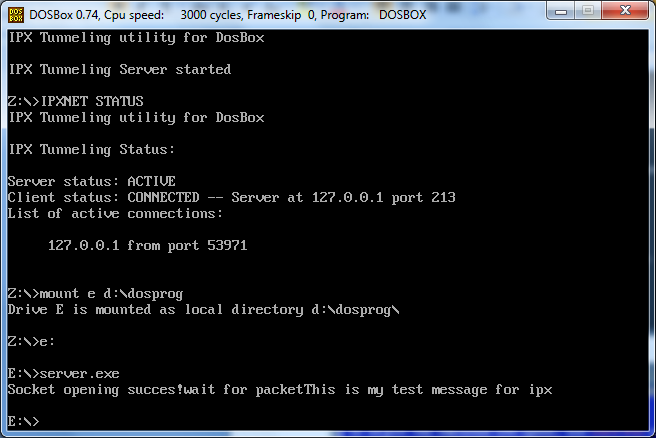
****

**Рис.7**

И запускаем клиент

****

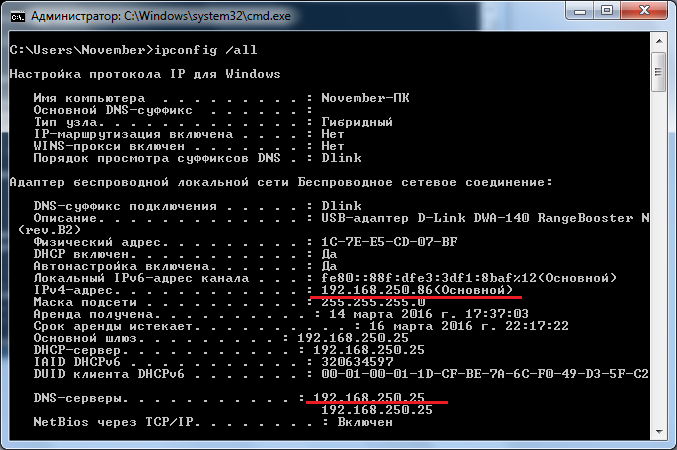
**Рис.8**

Отправляем сообщение и смотрим результат в сервере ****

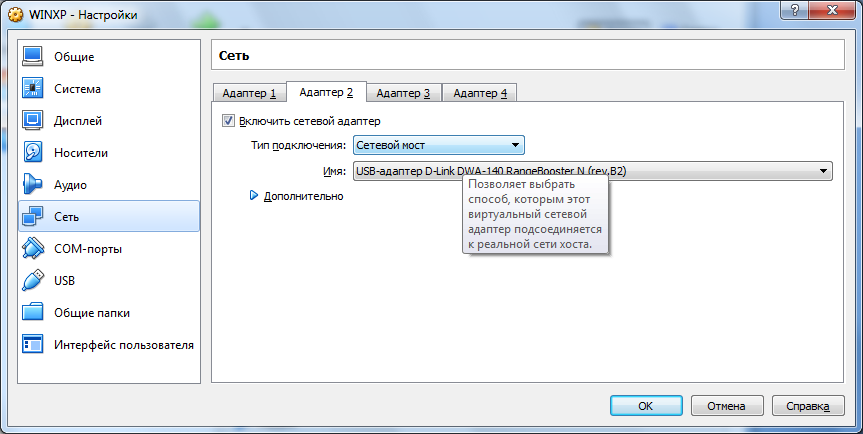
**Рис.9**

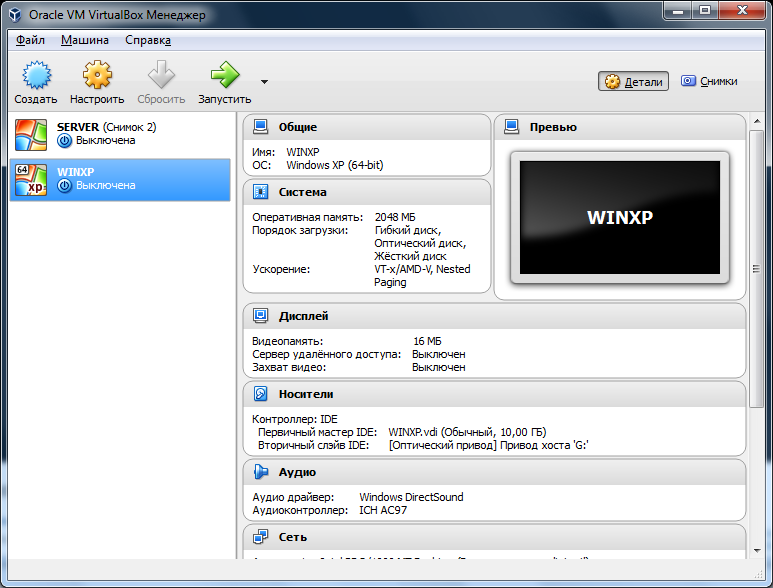
Для подключения через виртуальную машину требуется выполнить следующие действия

Запускаем командную строку:

****

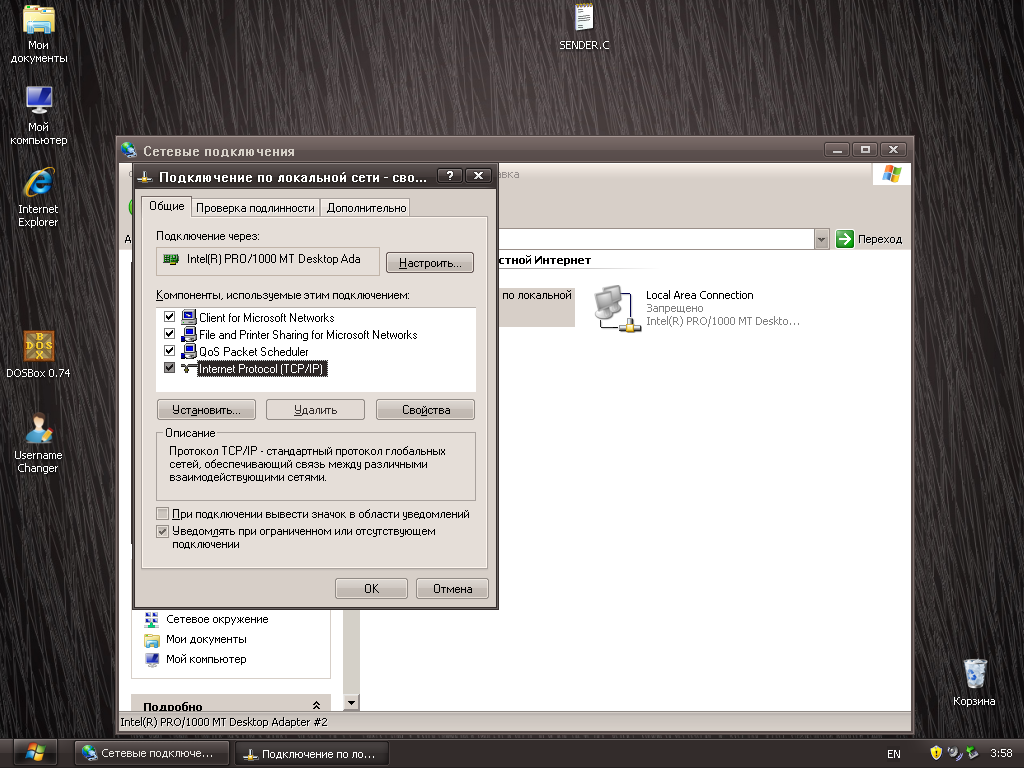
**Рис.10**

**Настраиваем подключение в oracle VirtualMachineРис.11**

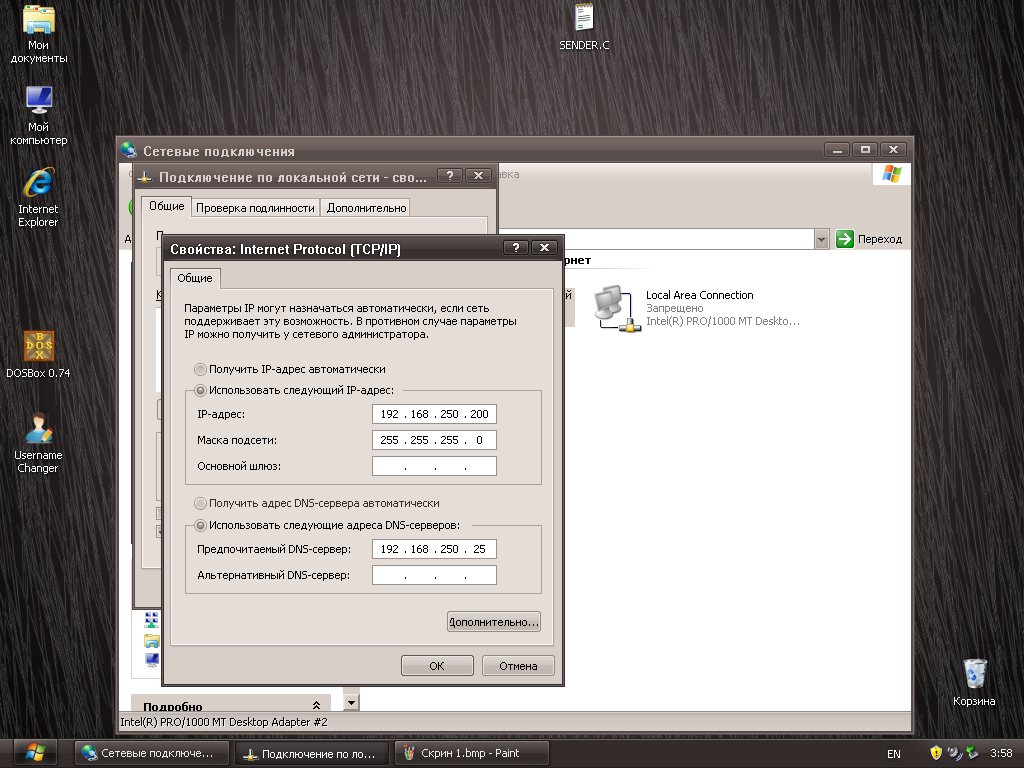
****

**Рис.12**

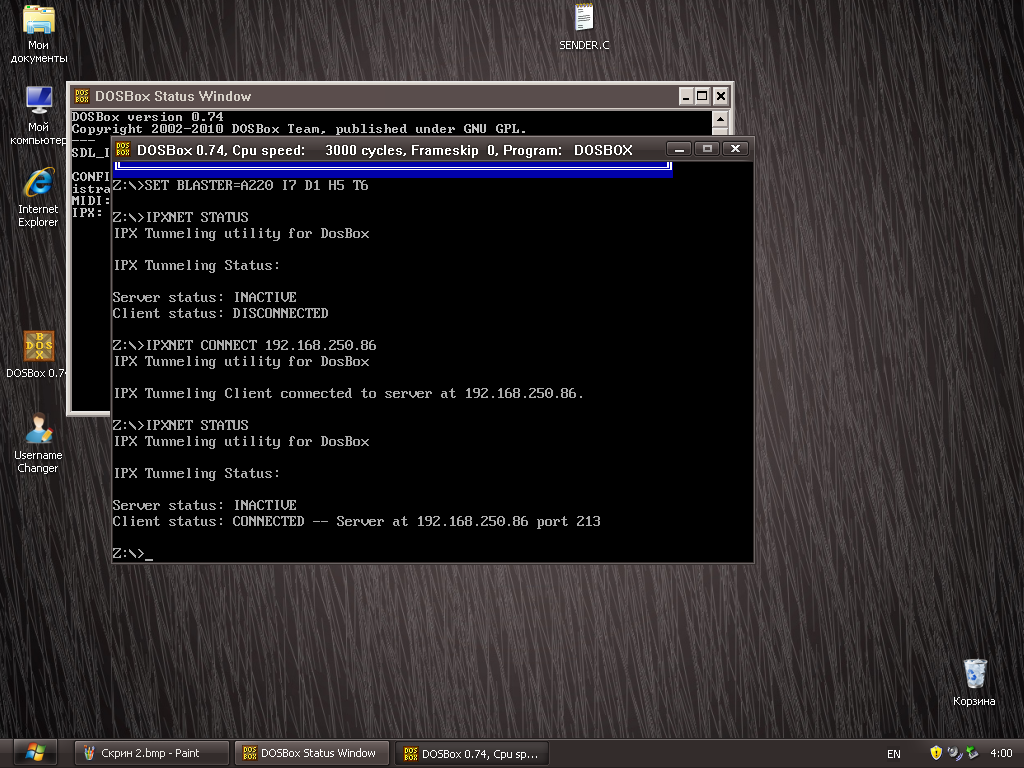
Теперь настраиваем локальную сеть в WinXP

**Значение вводить для своего сетевого адаптера,данные о котором получаем из командной строки командной ipconfig  
**

**Рис.13**

****

**Рис.14**

****

**Рис.14**

Таким образом на одной машине можно запустить несколько версий приложения для проверки работоспособности приложения.

Теперь перейдем к следующему этапу выполнения работы. Нам необходимо передать по сети файл (изображение размером более 2 МБ). Из этого вытекают следующие моменты:

1. Максимальный размер пакета 546 байт, следовательно для передачи данных необходимо последовательно считывать из файла (читать как бинарный ) блоки размером 546 байт и последовательно передать их.
2. Для завершения процесса передачи необходимо, каким-то образом сообщить принимающей стороне о завершении процесса.

Решение первой задачи выглядит следующим образом. Необходимо описать конкретную функцию, формирующую заголовок и блок EBX, входным параметром будет адрес буферной строки считанных байтов из бинарного файла.

void SendMessage(char\*Packet,int Length)

{

union REGS inregs, outregs;

struct SREGS segreg;

IPXHeader Header;

ECB\_2 My\_ECB;

ECB\_2 far \*My\_ECB\_ptr;

memset(&My\_ECB,0,sizeof(My\_ECB));

My\_ECB.ESRAddress = NULL;

My\_ECB.SocketNumber = TargetSocket;

My\_ECB.FragmentCount = 2;

My\_ECB.FragmentAddress\_1 = &Header;

My\_ECB.FragmentSize\_1 = header\_length;

My\_ECB.FragmentAddress\_2 = Packet;

My\_ECB.FragmentSize\_2 = Length;

memset(My\_ECB.ImmediateAddress,0xff,6);

Header.PacketType = 0;

memset(Header.DestantionNetwork,0,4);

memset(Header.DestantionNode,0xFF,6);

Header.DestantionSocket = TargetSocket;

inregs.x.bx = IPX\_send;

My\_ECB\_ptr = &My\_ECB;

inregs.x.si = FP\_OFF ( My\_ECB\_ptr );

segreg.es = FP\_SEG ( My\_ECB\_ptr );

int86x (IPX\_interrupt, &inregs, &outregs, &segreg);

while (My\_ECB.InUseFlag)

{};

}

Отправку пакетов организуем циклически. При чтении бинарного файла есть следующие тонкости:

1. Если считывать сразу блок размером 546 байт- возникают пробелы в информативной части связанные с разного рода ошибками чтения
2. Если считывать файл до конца, то программа натыкается на битовое представление символа конца файла и прекращает чтение, по этой причине необходимо узнать размер файла в байтах и считывать его от начала и до конца.

FileLength=filelength(fileno(image));

printf("filesize=%ld",FileLength);

printf("\nPacketCount=%d",FileLength/544);

system("pause");

for(i=0;i<FileLength;i+=1)

{

fread(Symbol,1,1,image);

DispString[packet\_size]=Symbol[0];

packet\_size+=1;

if(packet\_size==546)

{

SendMessage(DispString,546);

strnset(DispString,'\0',546);

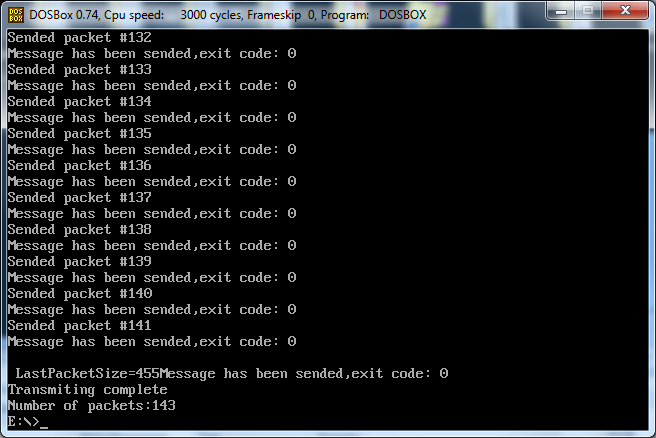
counter++;

packet\_size=0;

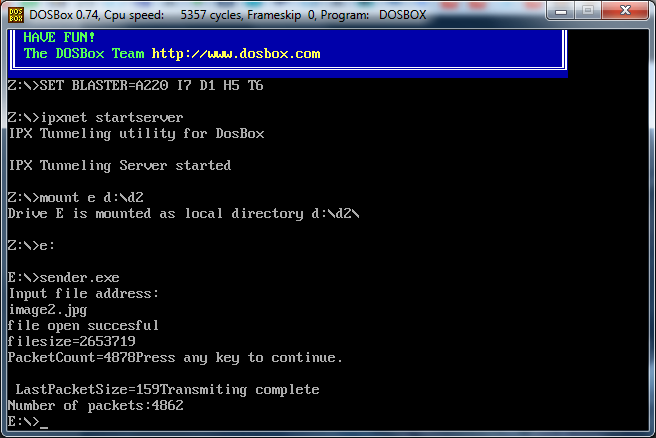
}

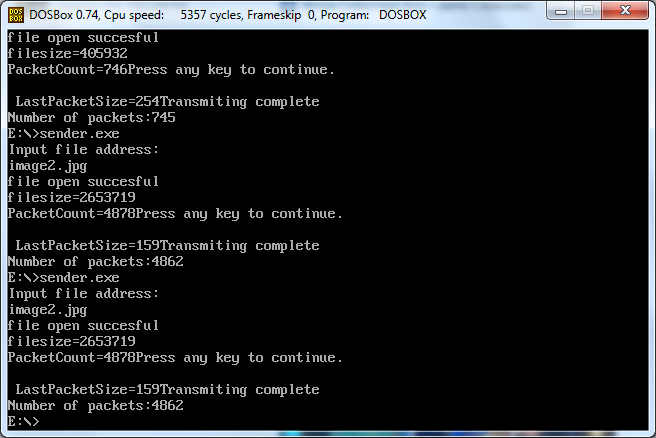
}

На стороне приемника изображения признаком конца файла будет файл, размер которого меньше чем 576(546 информационных +30 байт заголовка) байт. Так как заголовок блока ECB хранит данные в обратном порядке, необходимо узнать число которое соответствует полному пакету. В отдельной программе я передал блок размером 576 байт и в принимающей программе проверил размер этого блока (узнал его представление в обратном порядке). Таким образом, если на стороне приемника размер полученного блока в поле Length заголовка Header не равен 16386, значит пакет не полный и прием окончен. Для подстраховки случая, когда файл нацело поделился на 546, программа отправитель отсылает страховочный пакет размером в 250 байт и заполненный символами конца строки.

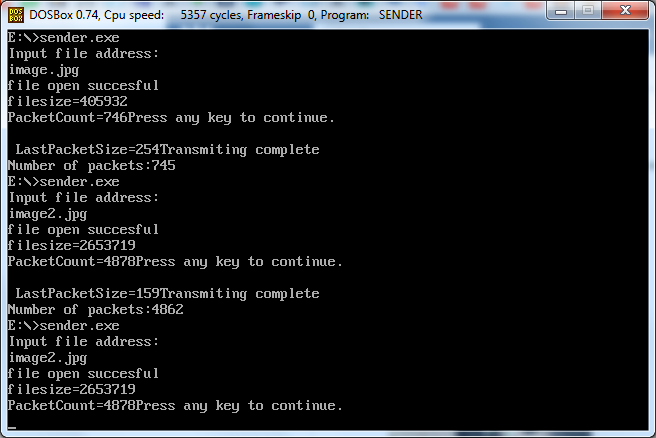
Программа приемник работает аналогично предыдущей версии, за исключением того, что она принимает пакеты до тех пор, пока не получит не полный пакет и записывает данные в бинарный файл в формате, указанном пользователем. Полные тексты программы отправителя и получателя будут приведены в конце работы.**Рис.15**



**Рис.16****Рис.17**



**Рис.18**

**Рис.19**

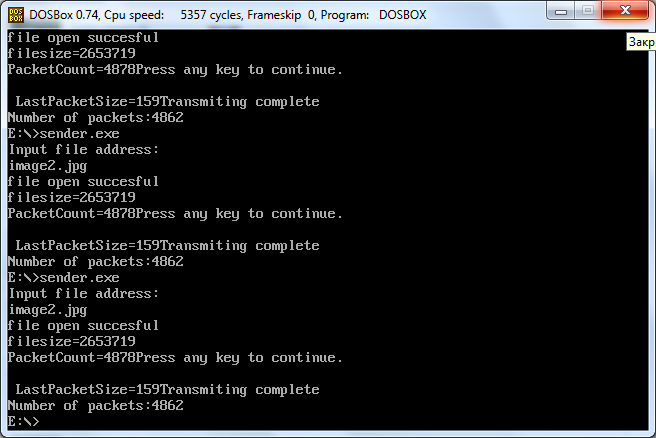
**Рис.20**

Таблица статистики передачи данных:

Отчет о потерях при передаче данных при передаче в пределах 1 пк

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер файла | Число пакетов | Число получателей | Число утерянных пакетов | Процент потерь |
| 75кб | 142 | 1 | 0 | 0 |
| 2.13 мб | 3916 | 2 | 92 | 0% в одном процесса и 3.4 % в другом |

Потери при передаче по каналу WI-fi между 2 компьютерами

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер файла | Число пакетов | Число получателей | Число утерянных пакетов | Процент потерь |
| 2.53 мб | 4862 | 1 | 31 | 0.6% |
| 2.53 мб | 4862 | 1 | 0 | 0 |
| 2.53 мб | 4862 | 2 | 3  2 | 0.07%  И 0.05 % |
| 2.53 мб | 4862 | 4 | 1  0  1  1 | 0.03%  0  0.03%  0.03% |
| 2.53 мб | 4862 | 6 | 30  30  29  29 | 0.61%  0.61%  0.58%  0.58% |

**Исходные коды программ**

**SENDER.C**

#include <stdio.h>

#include <string.h>

#include <dos.h>

#include <ipx.h>

#include <io.h>

#define TargetSocket 0x4567

#define TargetNode 0xFF

void SendMessage(char\*Packet,int Length)

{

union REGS inregs, outregs;

struct SREGS segreg;

IPXHeader Header;

ECB\_2 My\_ECB;

ECB\_2 far \*My\_ECB\_ptr;

memset(&My\_ECB,0,sizeof(My\_ECB));

My\_ECB.ESRAddress = NULL;

My\_ECB.SocketNumber = TargetSocket;

My\_ECB.FragmentCount = 2;

My\_ECB.FragmentAddress\_1 = &Header;

My\_ECB.FragmentSize\_1 = header\_length;

My\_ECB.FragmentAddress\_2 = Packet;

My\_ECB.FragmentSize\_2 = Length;

memset(My\_ECB.ImmediateAddress,0xff,6);

Header.PacketType = 0;

memset(Header.DestantionNetwork,0,4);

memset(Header.DestantionNode,0xFF,6);

Header.DestantionSocket = TargetSocket;

inregs.x.bx = IPX\_send;

My\_ECB\_ptr = &My\_ECB;

inregs.x.si = FP\_OFF ( My\_ECB\_ptr );

segreg.es = FP\_SEG ( My\_ECB\_ptr );

int86x (IPX\_interrupt, &inregs, &outregs, &segreg);

while (My\_ECB.InUseFlag)

{};

}

void main ()

{

long counter=0,packet\_size=0,i=0;

long FileLength=0;

unsigned char DispString [546]; /\*545 to transmite\*/

unsigned char Address[255];

unsigned char NewName[255];

unsigned char Symbol[1];

union REGS inregs, outregs;

struct SREGS segreg;

IPXHeader Header;

ECB\_2 My\_ECB;

ECB\_2 far \*My\_ECB\_ptr;

FILE \*image;

FILE \*image\_copy;

printf("Input file address:\n");

gets((char\*)Address);

image=fopen((char\*)Address,"rb");

if(image==NULL)

printf("this file does not exist\n");

else

{

printf("file open succesful\n");

FileLength=filelength(fileno(image));

printf("filesize=%ld",FileLength);

printf("\nPacketCount=%d",FileLength/544);

system("pause");

for(i=0;i<FileLength;i+=1)

{

fread(Symbol,1,1,image);

DispString[packet\_size]=Symbol[0];

packet\_size+=1;

if(packet\_size==546)

{

SendMessage(DispString,546);

strnset(DispString,'\0',546);

counter++;

packet\_size=0;

}

}

SendMessage(DispString,packet\_size);

printf("\n LastPacketSize=%d",packet\_size);

strnset(DispString,'\n',544);

SendMessage(DispString,0);

counter+=2;

printf("Transmiting complete\nNumber of packets:%ld",counter);

}

}

**SERVER.C**

#include <stdio.h>

#include <string.h>

#include <dos.h>

#include <ipx.h>

#define TargetSocket 0x4567

void main () {

unsigned char RecieveString [546];

unsigned char Address[255];

int Length=0;

int Flag=1,Count=0;

union REGS inregs, outregs;

struct SREGS segreg;

FILE \*image\_copy;

IPXHeader Header;

ECB\_2 My\_ECB;

ECB\_2 far \*My\_ECB\_ptr;

My\_ECB.ESRAddress = NULL;

My\_ECB.SocketNumber = TargetSocket;

My\_ECB.FragmentCount = 2;

My\_ECB.FragmentAddress\_1 = &Header;

My\_ECB.FragmentSize\_1 = header\_length;

My\_ECB.FragmentAddress\_2 = &RecieveString;

My\_ECB.FragmentSize\_2 = sizeof RecieveString;

inregs.x.bx = IPX\_open;

inregs.h.al = 0xFF;

inregs.x.dx = TargetSocket;

My\_ECB\_ptr = &My\_ECB;

printf("Input file address:\n");

gets((char\*)Address);

image\_copy=fopen((char\*)Address,"wb");

int86 (IPX\_interrupt, &inregs, &outregs);

if (outregs.h.al != 0x00)

printf("Socket opening error,exit code:%x\n",outregs.h.al);

else printf("Socket opening succes!");

while(Flag==1)

{

inregs.x.bx = IPX\_listen;

inregs.x.si = FP\_OFF ( My\_ECB\_ptr );

segreg.es = FP\_SEG ( My\_ECB\_ptr );

int86x (IPX\_interrupt, &inregs, &outregs, &segreg);

while (My\_ECB.InUseFlag){}

if (My\_ECB.CompletionCode != 0)

printf("Message recieving error,exit code: %x\n",My\_ECB.CompletionCode);

else

{

Count++;

if(Header.Length!=16386)

Flag=0;

else

fwrite(RecieveString,1,546,image\_copy);

}

}

fputs(RecieveString,image\_copy);

printf("\nRecieved packets:%d",Count);

fclose(image\_copy);

inregs.x.bx = IPX\_close;

inregs.x.dx = TargetSocket;

int86 (IPX\_interrupt, &inregs, &outregs);

}

**IPX.H**

#define IPX\_interrupt 0x7A

#define header\_length 30

#define IPX\_open 0x00

#define IPX\_close 0x01

#define IPX\_localTarget 0x02

#define IPX\_send 0x03

#define IPX\_listen 0x04

#define IPX\_event 0x05

#define IPX\_cancel 0x06

#define IPX\_specialEvent 0x07

#define IPX\_interval 0x08

#define IPX\_GetInternetwork 0x09

#define IPX\_control 0x0A

#define IPX\_disconnect 0x0B

typedef struct{

unsigned int Checksum;

unsigned int Length;

unsigned char TransportControl;

unsigned char PacketType;

unsigned char DestantionNetwork [4];

unsigned char DestantionNode [6];

unsigned int DestantionSocket;

unsigned char SourceNetwork [4];

unsigned char SourceNode [6];

unsigned int SourceSocket;

} IPXHeader;

typedef struct {

void far \*LinkAddress;

void far \*ESRAddress;

unsigned char InUseFlag;

unsigned char CompletionCode;

unsigned int SocketNumber;

unsigned char IPXWorkspace [4];

unsigned char DriverWorkspace [12];

unsigned char ImmediateAddress [6];

unsigned int FragmentCount;

void far \*FragmentAddress;

unsigned int FragmentSize;

} ECB\_1;

typedef struct {

void far \*LinkAddress;

void far \*ESRAddress;

unsigned char InUseFlag;

unsigned char CompletionCode;

unsigned int SocketNumber;

unsigned char IPXWorkspace [4];

unsigned char DriverWorkspace [12];

unsigned char ImmediateAddress [6];

unsigned int FragmentCount;

void far \*FragmentAddress\_1;

unsigned int FragmentSize\_1;

void far \*FragmentAddress\_2;

unsigned int FragmentSize\_2;

} ECB\_2;

typedef struct {

void far \*LinkAddress;

void far \*ESRAddress;

unsigned char InUseFlag;

unsigned char WorkSpace [5];

} SpecialECB;