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 №1**

дисциплина «Сети ЭВМ и телекоммуникации»

тема: «Протокол сетевого уровня IPX»

Выполнил: студент группы ВТ-31

Ковалёв И. Д.

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.

1. **Краткие теоретические сведения**

IPX (англ. internetwork packet exchange — межсетевой обмен пакетами) — протокол сетевого уровня модели OSI в стеке протоколов IPX/SPX. Предназначен для передачи датаграмм. Для передачи данных установки соединения не требуется .

Логическим сетям присваивается уникальный 32-разрядный адрес в диапазоне 0x1 - 0xFFFFFFFE.

Хосты имеют 48-разрядный адрес узла, являющийся MAC-адресом сетевого адаптера. Адрес узла добавляется к адресу сети для создания уникального идентификатора хоста в сети.

Номер сети 00:00:00:00 означает текущую сеть.

Широковещательный адрес — FF:FF:FF:FF.

1. **Основные функции API, использованные в данной работе.**

int IPXOpenSocket(int SocketType, unsigned \*Socket)

Назначение: открывает сокет типа SocketType с номером Socket.

Входные параметры: тип сокета (долгосрочный или короткосрочный), номер сокета.

void IPXCloseSocket(unsigned \*Socket)

Назначение: закрывает сокет типа SocketType с номером Socket.

Входные параметры: номер сокета

void IPXListenForPacket(struct ECB \*RxECB)

Назначение: производит запрос к драйверу IPX на получение входящего пакета, который будет записан в структуру ЕСВ.

Входные параметры: структура ЕСВ.

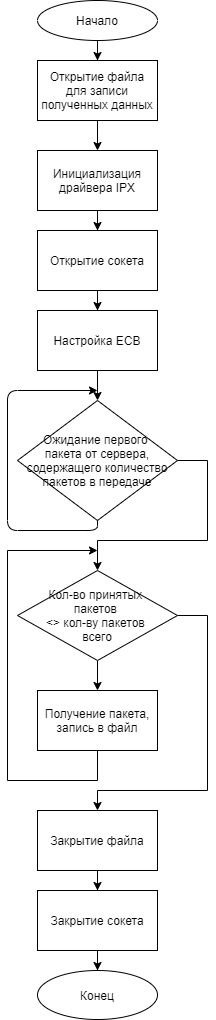
void IPXSendPacket(struct ECB \*TxECB)

Назначение: открывает сокет типа SocketType с номером Socket.

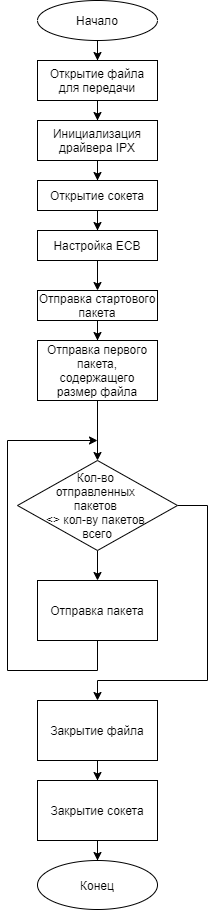
Входные параметры: производит запрос к драйверу IPX на отправку исходящего пакета, который хранится в структуре ЕСВ.

1. **Разработка программы. Блок-схема программы.**

Блок-схема работы программы-клиента:

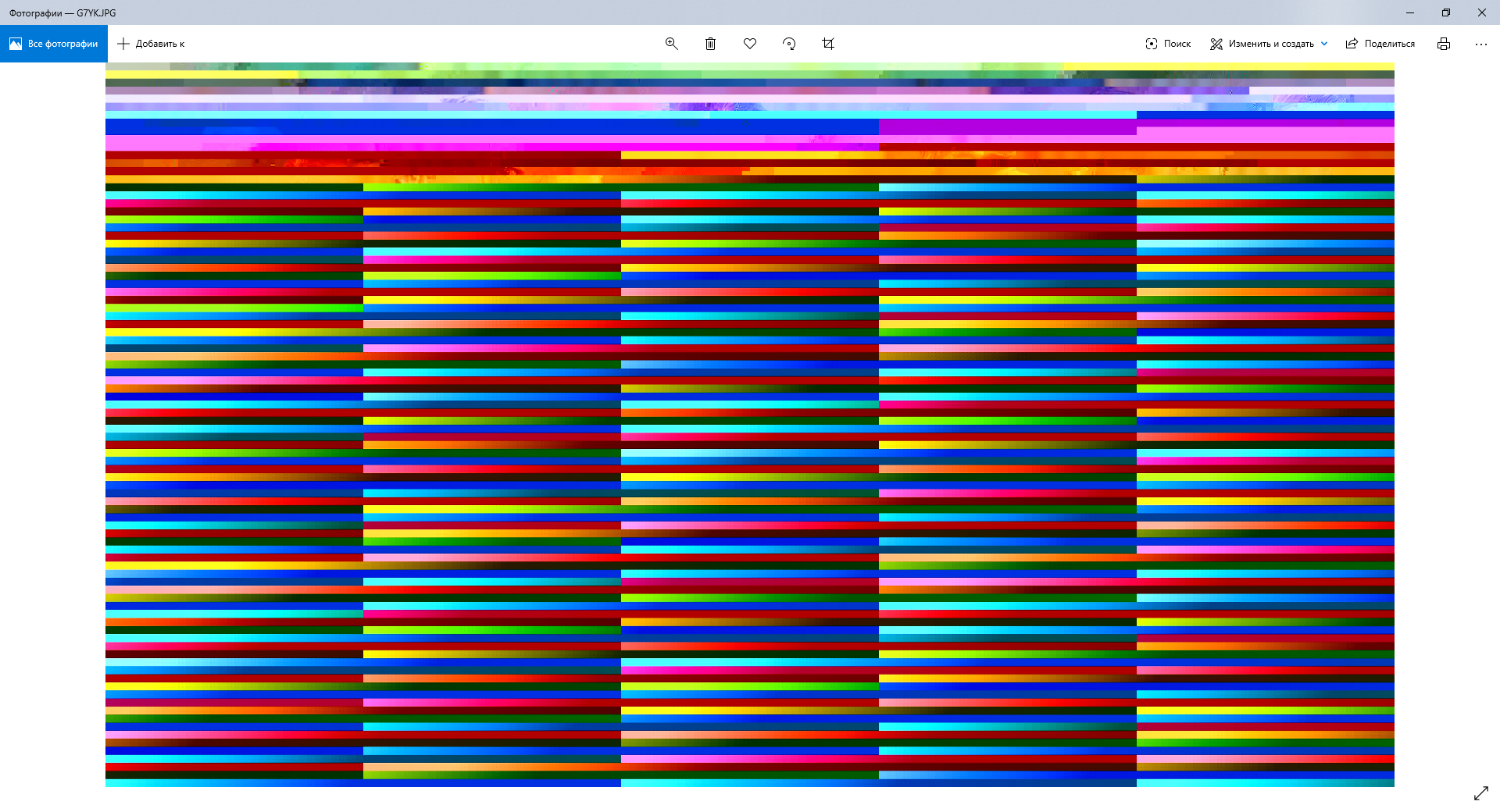
****

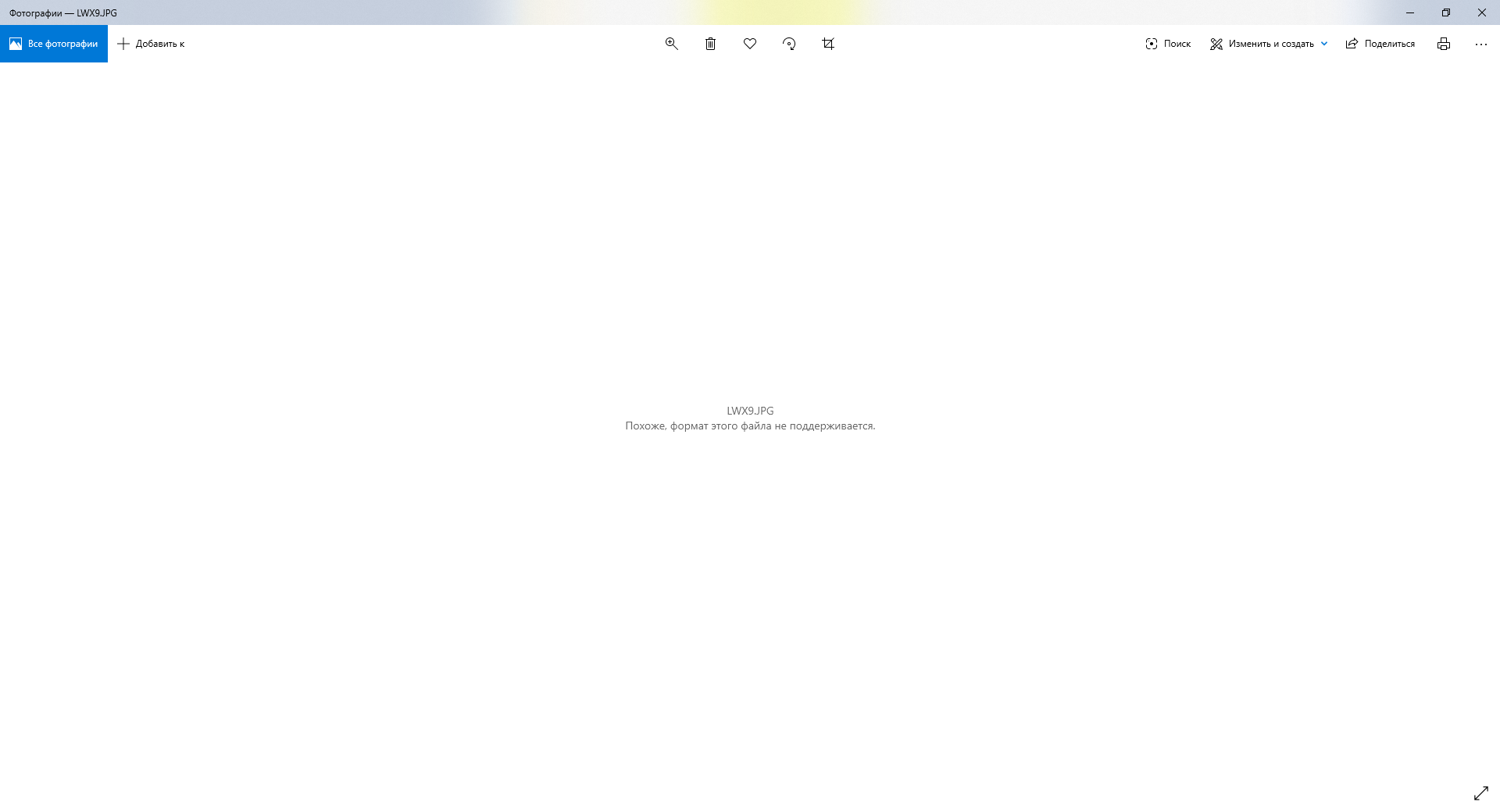
Блок-схема работы программы-сервера:



1. **Анализ функционирования разработанных программ**

В процессе тестирования разработанных программ было выявлено, что передача файлов без задержки после отправки каждого пакета со стороны сервера, данные доходят не полностью, и передаваемое изображение будет получено не полностью.



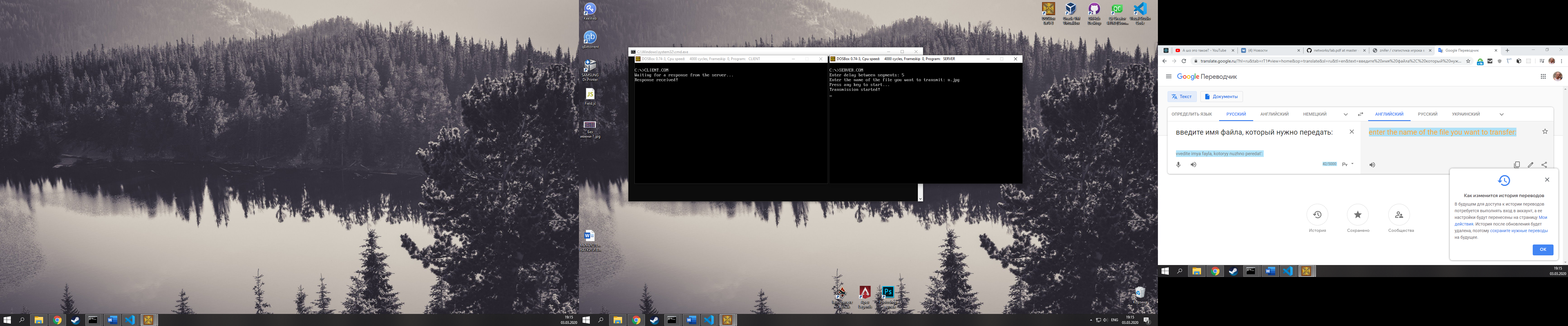


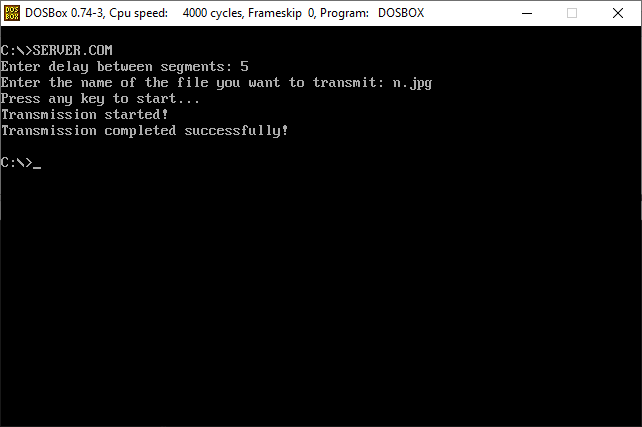
Если же установить задержку более 30 мс, полученное изображение отображается корректно.

1. **Выводы.**

В процессе выполнения лабораторной работы был изучен протокол сетевого уровня IPX, основные функции API драйвера IPX и разработана программу для приема/передачи данных.

1. **Тексты программ, скриншоты программ.**





#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 );

}

**Исходный файл программы-клиента:**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <mem.h>

#include <string.h>

#include "ipx.h"

#include <time.h>

#define BUFFER\_SIZE 500

typedef struct \_fileSegment{

long int size;

long int num;

} fileSegment;

unsigned char segmentBuffer[BUFFER\_SIZE];

void main(void) {

long int SegmentsCounter;

int i,j;

FILE\* file;

fileSegment tmp;

static unsigned Socket = 0x4444;

struct ECB RxECB;

struct IPX\_HEADER RxHeader;

unsigned char RxBuffer[BUFFER\_SIZE];

char name[256] = "result.jpg";

file = fopen(name, "wb+");

if(file==NULL){

printf("Error creating file!");

exit(101);

}

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

printf("Error initializing IPX!\n");

exit(102);

}

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

printf("Error opening socket!\n");

exit(-1);

};

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;

RxECB.Packet[2].Size = 8;

IPXListenForPacket(&RxECB);

printf("Waiting for a response from the server...\n");

while(1) {

if(!RxECB.InUse){

printf("Response received!\n");

break;

};

}

IPXListenForPacket(&RxECB);

while(RxECB.InUse){}

memcpy(&SegmentsCounter, RxBuffer, tmp.size);

rintf("Receiving data...\n");

for(i = 0; i <= SegmentsCounter; i++){

IPXListenForPacket(&RxECB);

while(RxECB.InUse){}

for(j = 0; j < tmp.size; j++){

fputc(RxBuffer[j],file);

};

if(tmp.num == SegmentsCounter){

break;

};

}

fclose(file);

IPXCloseSocket(&Socket);

printf("Transmission completed successfully!\n");

exit(0);

}

**Исходный файл программы-сервера:**

#include <STDIO.H>

#include <STDLIB.H>

#include <CONIO.H>

#include <MEM.H>

#include <STRING.H>

#include <DOS.H>

#include "IPX.H"

#define BUFFER\_SIZE 500

typedef struct \_fileSegment{

long int size;

long int num;

} fileSegment;

unsigned char segmentBuffer[BUFFER\_SIZE];

long int getFileSize(FILE\* stream){

long int size;

fseek(stream,0,SEEK\_END);

size = ftell(stream);

fseek(stream,0,SEEK\_SET);

return size;

};

void main(void) {

long int fileSize;

long int fullSegmentsCounter, segmentsCounter;

long int lastSegmentSize;

FILE\* file;

FILE\* checkFile;

fileSegment tmp;

static unsigned Socket = 0x4444;

struct ECB ServerECB;

struct IPX\_HEADER InHeader, OutHeader;

unsigned char OutBuffer[BUFFER\_SIZE];

int i;

char filename[255] = "image.jpg";

int delayValue = 0;

printf("Enter delay between segments: ");

scanf("%i", &delayValue);

printf("Enter the name of the file you want to transmit: ");

scanf("%s", filename);

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

printf("Error initializing IPX!\n");

exit(101);

}

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

printf("Error opening socket!\n");

exit(102);

}

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

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

if(file != NULL){

fileSize = getFileSize(file);

fullSegmentsCounter = fileSize / (BUFFER\_SIZE - 8);

lastSegmentSize = fileSize % (BUFFER\_SIZE - 8);

segmentsCounter = fullSegmentsCounter + 1;

fclose(file);

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

checkFile = fopen("IMG.jpg", "wb+");

}else{

printf("Error opening file!\n");

exit(103);

};

printf("Press any key to start...\n");

getch();

tmp.size = 0;

tmp.num = -1;

segmentBuffer[0] = 0;

memcpy(OutBuffer,&segmentBuffer,BUFFER\_SIZE);

OutHeader.PacketType = 4;

memset(OutHeader.DestNetwork, 0, 4);

memset(OutHeader.DestNode, 0xFF, 6);

OutHeader.DestSocket = IntSwap(Socket);

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;

ServerECB.Packet[2].Size = 8;

IPXSendPacket(&ServerECB);

printf("Transmission started!\n");

tmp.num = 0;

tmp.size = sizeof(segmentsCounter);

memcpy(segmentBuffer, &segmentsCounter, sizeof(segmentsCounter));

memcpy(OutBuffer, &segmentBuffer, BUFFER\_SIZE);

IPXSendPacket(&ServerECB);

for(i = 1; i <= fullSegmentsCounter; i++){

tmp.num = i;

tmp.size = BUFFER\_SIZE;

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

fwrite(OutBuffer,1,BUFFER\_SIZE,checkFile);

IPXSendPacket(&ServerECB);

delay(delayValue);

};

delay(delayValue);

tmp.num = segmentsCounter;

tmp.size = lastSegmentSize;

fread(OutBuffer,1,lastSegmentSize,file);

fwrite(OutBuffer,1,lastSegmentSize,checkFile);

IPXSendPacket(&ServerECB);

printf("Transmission completed successfully!\n");

fclose(file);

fclose(checkFile);

IPXCloseSocket(&Socket);

exit(0);

}