Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

ДИСЦИПЛИНА Сетевые технологии в АСОИУ

Курсовая работа

«Локальная безадаптерная сеть»

Описание программы
(вид документа)

писчая бумага
(вид носителя)

25
(количество листов)

Выполнили:

ИУ5-63Б		**Богданов Д.А.**
(Группа)	(Подпись, дата)	(Фамилия И.О.)
ИУ5-63Б		**Сёмкин Н.Е.**
(Группа)	(Подпись, дата)	(Фамилия И.О.)
ИУ5-63Б		**Попов М.А.**
(Группа)	(Подпись, дата)	(Фамилия И.О.)

Руководитель курсовой работы:

	Галкин В.А.
(Подпись, дата)	(Фамилия И.О.)

Консультант:

(Подпись, дата)	(Фамилия И.О.)

Москва – 2020г.

Оглавление

1. Введение

Программный продукт написан с использованием среды разработки Visual Studio на языке программирования C#.

Для создания графического интерфейса и взаимодействия с COM-портами использовались стандартные библиотеки и элементы управления. Дополнительные функции, не относящиеся к стандартным, приведены ниже.

2. Классы, используемые в программе

2.1. Класс Connection.cs

Таблица 1. Поля класса Connection.cs

Имя	Описание
STARTBYTE	Стартовый байт
HeaderLenght	Длина заголовка кадра
fileTypeLenght	Длина части кадра, хранящей тип файла
sizeLenght	Длина части кадра, хранящей размер файла
NumOfFrameLenght	Длина части кадра, хрянящей номер посылаемого кадра
InfoLen	Длина кадра
FilePath	Путь к отправляемому файлу на компьютере отправителя
BreakConnection	Флаг наличия обрыва соединения
file_buffer	Буфер для загрузки файла на стороне получателя
ProgressBar	Индикатор выполнения операция
b_ChooseFile	Кнопка выбора файла и последующей отправки

Таблица 2. Перечисления класса Connection.cs

Имя	Описание
FrameType	Содержит типы кадров

Таблица 3. Свойства класса Connection.cs

Имя	Описание
MainForm	Связь с основной формой
Log	Окно вывода логов
Port	Текущий порт
ProgressBar	Индикатор выполнения операции

2.2. Класс Hamming.cs

Таблица 4. Методы класса Hamming.cs

Имя	Описание
ErrorDigit	Возвращает ошибочную цифру
HammingEncode74	Кодирует один информационный байт в 2 закодированных
HammingDecode74	Формирует из информационного байта массив для декодирования
HammingSimptome74	Вычисляет синдром ошибки
HammingCorrection74	Исправляет ошибку
Decode	Декодирование 2 байтов в один информационный

3. Листинг программы

3.1. Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ComPort
{

    public partial class Form1 : Form
    {
        Connection com1 = new Connection();
        //
        Connection com2 = new Connection();

        public Form1()
        {
            InitializeComponent();
            cb_PortNames.Items.AddRange(SerialPort.GetPortNames());
            com1.MainForm = this;
            com1.ProgressBar = progressBar1;
            com1.b_ChooseFile = b_ChooseFile;
            com1.b_Connection = b_Connection;
            com1.b_OpenPort = Open_COM_1;
            b_con.Enabled = false;
            b_ChooseFile.Enabled = true;
            b_Connection.Enabled = false;
            richTextBox1.AppendText("Добро пожаловать!\nПеред началом работы выберите порты
из списка и откройте их.\n\n");




            cb_PortNames2.Items.AddRange(SerialPort.GetPortNames());
            com2.b_OpenPort = Open_COM_2;

        }

        /// <summary>
        /// Пишет в лог, есть ли соединение
        /// </summary>
        private void b_con_Click(object sender, EventArgs e)
        {

            if (com1.IsConnected() && com2.IsConnected())
            {
                richTextBox1.AppendText("[" + DateTime.Now + "]: " +
cb_PortNames.SelectedItem.ToString() + " и " + cb_PortNames2.SelectedItem.ToString() + ":
Соединение установлено\n");


                b_ChooseFile.Visible = true;
                button1.Visible = true;
```

```
                if (com1.Port.PortName == "COM1")
                {
                    b_ChooseFile.Text = "Отправить на второй компьютер";
                }

                if (com1.Port.PortName == "COM2")
                {
                    b_ChooseFile.Text = "Отправить на первый компьютер";
                }

                if (com1.Port.PortName == "COM4")
                {
                    b_ChooseFile.Text = "Отправить на первый компьютер";
                }

                if (com2.Port.PortName == "COM3")
                {
                    button1.Text = "Отправить на третий компьютер";
                }

                if (com2.Port.PortName == "COM5")
                {
                    button1.Text = "Отправить на третий компьютер";
                }

                if (com2.Port.PortName == "COM6")
                {
                    button1.Text = "Отправить на второй компьютер";
                }
            }
            else
            {
                richTextBox1.AppendText("[" + DateTime.Now + "]: " +
cb_PortNames.SelectedItem.ToString() + " и " + cb_PortNames2.SelectedItem.ToString() + ":
Соединение отсутствует\n");

                b_ChooseFile.Visible = false;
                button1.Visible = false;
            }

        }


        /// <summary>
        /// Открывает порт com1
        /// </summary>
        private void open_com_1(object sender, EventArgs e)
        {
            if (cb_PortNames.SelectedItem != null)
            {
                com1.Log = richTextBox1;

                com1.setPortName(cb_PortNames.SelectedItem.ToString());

                if (com1.OpenPort())
                {
                    //com1.Port.DtrEnable = true;

                    cb_PortNames.Enabled = false;

                    Open_COM_1.Enabled = false;

                    Close_COM_1.Enabled = true;

                    richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " +
com1.Port.PortName + " открыт\n");
```

```
                }

            }
            else
            {
                MessageBox.Show("Порт не выбран!", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

            if(com1.Port.IsOpen && com2.Port.IsOpen)
            {
                b_Connection.Enabled = true;
            }


        }



        private void b_ChooseFile_Click(object sender, EventArgs e)
        {
            Console.WriteLine(com1.Port.PortName);
            OpenFileDialog openFileDialog = new OpenFileDialog();
            if (com1.IsConnected())
            {
                if (openFileDialog.ShowDialog() == DialogResult.OK)
                {
                    com1.WriteData(openFileDialog.FileName, Connection.FrameType.FILEOK);
                    richTextBox1.ScrollToCaret();
                }
            }
            else
            {
                MessageBox.Show("Соединение отсутствует!", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

        }

        private void Form1_FormClosing(Object sender, FormClosingEventArgs e)
        {
            com1.ClosePort();
        }

        private void b_About_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Программа реализует взаимодействие 3-х ПК, соединенных через
интерфейс RS-232C,\n" +
                            "с функцией передачи файлов.\n\n" +
                            "Программу разработали студенты МГТУ им. Н.Э.Баумана группы ИУ5-
63:\n\n" +
                            "Богданов Д.А. (физический уровень)\n" +
                            "Попов М.А. (канальный уровень)\n" +
                            "Сёмкин Н.Е. (прикладной уровень)",
                            "О программе",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Information);
        }

        private void b_Connection_Click(object sender, EventArgs e)
        {

            if ((com1.Port.IsOpen&& !com2.Port.IsOpen) ||
(!com1.Port.IsOpen&&com2.Port.IsOpen))
            {
                MessageBox.Show("Откройте второй порт!", "Ошибка", MessageBoxButtons.OK,
```

```
MessageBoxIcon.Error);
            }
            else
            {
                if((!com1.Port.DtrEnable && !com2.Port.DtrEnable) || !com1.Port.DtrEnable ||
!com2.Port.DtrEnable)
                {
                    com1.Port.DtrEnable = true;
                    com2.Port.DtrEnable = true;

                    b_con.Enabled = true;

                    richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " +
com1.Port.PortName + " и порт " + com2.Port.PortName + " готовы к передаче данных \n");

                    b_Connection.Text = "Разорвать соединение";

                    if (com1.IsConnected() && com2.IsConnected())
                    {

                        b_ChooseFile.Visible = true;
                        button1.Visible = true;

                        if (com1.Port.PortName == "COM1")
                        {
                            b_ChooseFile.Text = "Отправить на второй компьютер";
                        }

                        if (com1.Port.PortName == "COM2")
                        {
                            b_ChooseFile.Text = "Отправить на первый компьютер";
                        }

                        if (com1.Port.PortName == "COM4")
                        {
                            b_ChooseFile.Text = "Отправить на первый компьютер";
                        }

                        if (com2.Port.PortName == "COM3")
                        {
                            button1.Text = "Отправить на третий компьютер";
                        }

                        if (com2.Port.PortName == "COM5")
                        {
                            button1.Text = "Отправить на третий компьютер";
                        }

                        if (com2.Port.PortName == "COM6")
                        {
                            button1.Text = "Отправить на второй компьютер";
                        }
                    }



                }
                else
                {
                    com1.Port.DtrEnable = false;
                    com2.Port.DtrEnable = false;
                    b_Connection.Text = "Установить соединение";
                    richTextBox1.AppendText("[" + DateTime.Now + "]: Соединение было
разорвано\n");
                    b_con.Enabled = false;
```

```
                b_ChooseFile.Visible = false;
                button1.Visible = false;
            }

        }

    }



    private void cb_PortNames_SelectedIndexChanged(object sender, EventArgs e)
    {

    }

    private void Open_com_2(object sender, EventArgs e)
    {
        if (cb_PortNames2.SelectedItem != null)
        {
            com2.Log = richTextBox1;

            com2.setPortName(cb_PortNames2.SelectedItem.ToString());

            if (com2.OpenPort())
            {
                //com2.Port.DtrEnable = true;

                cb_PortNames2.Enabled = false;

                Open_COM_2.Enabled = false;

                Close_COM_2.Enabled = true;

                richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " +
com2.Port.PortName + " открыт\n");
            }

        }
        else
        {
            MessageBox.Show("Порт не выбран!", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }

        if (com1.Port.IsOpen && com2.Port.IsOpen)
        {
            b_Connection.Enabled = true;
        }

    }



    private void richTextBox1_TextChanged(object sender, EventArgs e)
    {

    }

    private void progressBar1_Click(object sender, EventArgs e)
    {

    }

    private void label1_Click(object sender, EventArgs e)
    {
```

```
        }

        private void button1_Click(object sender, EventArgs e)
        {
            com1.Port.Close();

            com1.Port.DtrEnable = false;

            richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " + com1.Port.PortName + "
закрыт\n");

            cb_PortNames.Enabled = true;

            Close_COM_1.Enabled = false;
            Open_COM_1.Enabled = true;

            b_Connection.Enabled = false;
            b_ChooseFile.Visible = false;

            b_Connection.Text = "Установить соединение";


        }

        private void Close_COM_2_Click(object sender, EventArgs e)
        {
            com2.Port.Close();

            com2.Port.DtrEnable = false;

            richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " + com2.Port.PortName + "
закрыт\n");

            cb_PortNames2.Enabled = true;

            Close_COM_2.Enabled = false;
            Open_COM_2.Enabled = true;

            b_Connection.Enabled = false;
            b_ChooseFile.Visible = false;

            b_Connection.Text = "Установить соединение";
        }

        private void button1_Click_1(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            if (com2.IsConnected())
            {
                if (openFileDialog.ShowDialog() == DialogResult.OK)
                {
                    com2.WriteData(openFileDialog.FileName, Connection.FrameType.FILEOK);
                    richTextBox1.ScrollToCaret();
                }
            }
            else
            {
                MessageBox.Show("Соединение отсутствует!", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
    }
}
```

3.2. Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ComPort
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
             }
```

3.3. Connection.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO;
using System.IO.Ports;
using System.Windows.Forms;

namespace ComPort
{
    class Connection
    {
        int SuccessfulFrameNumber = 0;
        SerialPort _Port = new SerialPort();
        public SerialPort Port
        {
            get
            {
                return _Port;
            }
            set
            {
                _Port = value;
                if (_Port.IsOpen)
                {
                    _Port.DiscardInBuffer();
                    _Port.DiscardOutBuffer();
                }
            }
        }

        public bool setPortName(string name)
        {
            string[] PortList = SerialPort.GetPortNames();

            if (Port.IsOpen)
            {
                Log.AppendText("[" + DateTime.Now + "] Нельзя менять имя порта, когда он
открыт\n");
                return false;
            }

            if (PortList.Contains(name))
            {
                Port.PortName = name;
                return true;
            }
            Log.AppendText("[" + DateTime.Now + "] Порт" + name + " не найден\n");  //нет
такого порта
            return false;
        }

        public bool OpenPort()
        {
            try
            {
                Port.Open();
                InitializeHandlers();

                return true;
            }

            catch (System.IO.IOException)
            {
```

```
                Log.AppendText("[" + DateTime.Now + "] Порт " + Port.PortName + " не
найден\n");
                return false;
            }

            catch (System.InvalidOperationException) //открыт в этом приложении
            {
                Log.AppendText("[" + DateTime.Now + "] Порт " + Port.PortName + "  уже
открыт\n");
                return false;
            }

            catch (System.UnauthorizedAccessException) //уже открыт в другом
приложении/другим окном
            {
                Log.AppendText("[" + DateTime.Now + "] Порт " + Port.PortName + "  уже
используется\n");
                return false;
            }
        }

        public bool ClosePort()
        {
            if (!Port.IsOpen)
            {
                Log.AppendText("[" + DateTime.Now + "] Порт " + Port.PortName + "  уже
закрыт\n");
                return false;
            }
            Port.Close();
            return true;
        }

        public bool IsConnected() //оба порта открыты и готовы слать данные
        {
            return Port.IsOpen && Port.DsrHolding;
            //return Port.IsOpen;
        }



        //=================================================*/*/*/*

        public const byte STARTBYTE = 0xFF;

        const int HeaderLenght = 2;
        const int fileTypeLenght = 1;
        const int sizeLenght = 10;
        const int NumOfFrameLenght = 7;

        const int InfoLen = HeaderLenght + fileTypeLenght + sizeLenght + NumOfFrameLenght +
NumOfFrameLenght;


        public enum FrameType : byte
        {
            ACK,
            MSG,
            RET_MSG,
            ERR_FILE,
            FILE,
            FRAME,
            FILEOK,
        }

        public static String FilePath;
        public bool BreakConnection = false;
```

```
        public void WriteData(string input, FrameType type)
        {
            byte[] Header = { STARTBYTE, (byte)type };

            byte[] fileId = { 0 };
            byte[] size;
            byte[] NumOfFrames;
            byte[] FrameNumber;

            byte[] BufferToSend;
            byte[] Telegram;
            string Telegram_s;
            string size_s;
            byte[] ByteToEncode;
            byte[] ByteEncoded;


            switch (type)
            {
                case FrameType.ERR_FILE:


                    break;
                case FrameType.MSG:
                    #region MSG
                    if (IsConnected())
                    {
                        // Telegram[] = Coding(input);
                        Telegram = Encoding.Default.GetBytes(input); //потом это кыш

                        BufferToSend = new byte[Header.Length + Telegram.Length]; //буфер
для отправки = заголовок+сообщение
                        Header.CopyTo(BufferToSend, 0);
                        Telegram.CopyTo(BufferToSend, Header.Length);

                        Port.Write(BufferToSend, 0, BufferToSend.Length);
                        Log.AppendText("(" + Port.PortName + ") WriteData: sent message >  "
+ Encoding.Default.GetString(Telegram) + "\n");
                    }
                    break;
                #endregion

                case FrameType.ACK:
                    #region ACK
                    if (IsConnected())
                    {
                        // Telegram[] = Coding(input);
                        Telegram = Encoding.Default.GetBytes(input); //потом это кыш

                        BufferToSend = new byte[Header.Length + Telegram.Length]; //буфер
для отправки = заголовок+сообщение
                        Header.CopyTo(BufferToSend, 0);
                        Telegram.CopyTo(BufferToSend, Header.Length);

                        Port.Write(BufferToSend, 0, BufferToSend.Length);
                        Telegram_s = Encoding.Default.GetString(Telegram);

                    }
                    break;
                #endregion

                case FrameType.FILEOK:
                    #region FILEOK
                    if (IsConnected())
                    {
                        ByteToEncode = File.ReadAllBytes(input);
                        FilePath = input;
```

```
                        size = new byte[sizeLenght];
                        size =
Encoding.Default.GetBytes(((double)ByteToEncode.Length).ToString()); //нужны байты
                        //Telegram = Encoding.Default.GetBytes(size); //потом это кыш

                        BufferToSend = new byte[Header.Length + size.Length]; //буфер для
отправки = заголовок+сообщение
                        Header.CopyTo(BufferToSend, 0);
                        size.CopyTo(BufferToSend, Header.Length);

                        //
                        Console.WriteLine(Port.PortName);

                        Port.Write(BufferToSend, 0, BufferToSend.Length);
                        size_s = Encoding.Default.GetString(size);
                        Log.AppendText("[" + DateTime.Now + "] Отправлена информация о
размере файла: " + size_s + " байт\n");
                        //SuccessfulFrameNumber = int.Parse(Telegram_s);
                    }
                    break;
                #endregion

                case FrameType.FRAME:
                    #region FRAME
                    if (IsConnected())
                    {
                        // Telegram[] = Coding(input);
                        Telegram = Encoding.Default.GetBytes(input); //потом это кыш
                        BufferToSend = new byte[Header.Length + Telegram.Length]; //буфер
для отправки = заголовок+сообщение
                        Header.CopyTo(BufferToSend, 0);
                        Telegram.CopyTo(BufferToSend, Header.Length);

                        Port.Write(BufferToSend, 0, BufferToSend.Length);
                        Telegram_s = Encoding.Default.GetString(Telegram);

                        //SuccessfulFrameNumber = int.Parse(Telegram_s);
                    }
                    else
                    {
                        Log.Invoke(new EventHandler(delegate
                        {
                            Log.AppendText("[" + DateTime.Now + "]: Передача файла
нарушена.\n");
                        }));
                        ProgressBar.Invoke(new EventHandler(delegate
                        {
                            ProgressBar.Visible = false;
                        }));
                        BreakConnection = true;
                        break;
                    }
                    break;
                #endregion

                case FrameType.FILE:
                    #region FILE
                    int i;
                    int parts = 0;
                    int EncodedByteIndex;
                    int Part_ByteEncodedIndex;
                    ByteEncoded = new byte[0];
                    size = new byte[0];
                    NumOfFrames = new byte[0];


                    if (IsConnected())
```

```
                    {

                        ByteToEncode = File.ReadAllBytes(@FilePath);
                        //b_ChooseFile.Invoke(new EventHandler(delegate
                        //{
                        //    b_ChooseFile.Enabled = false;
                        //}));
                        b_ChooseFile.Invoke(new EventHandler(delegate
                        {
                            b_ChooseFile.Enabled = false;
                        }));
                        b_OpenPort.Invoke(new EventHandler(delegate
                        {
                            b_OpenPort.Enabled = false;
                        }));
                        size = new byte[sizeLenght];
                        size =
Encoding.Default.GetBytes(((double)ByteToEncode.Length).ToString()); //нужны байты
                        //WriteData(Encoding.Default.GetString(size), FrameType.FILEOK);
                        NumOfFrames = new byte[NumOfFrameLenght];
                        FrameNumber = new byte[NumOfFrameLenght];

                        string typeFile = @input.Split('.')[1];
                        fileId[0] = TypeFile_to_IdFile(typeFile);


                        ByteEncoded = new byte[ByteToEncode.Length * 2];
                        for (i = 0; i < ByteToEncode.Length; i++)
                        {
                            Hamming.HammingEncode74(ByteToEncode[i]).CopyTo(ByteEncoded, i *
2);
                        }

                        if (ByteEncoded.Length + InfoLen < Port.WriteBufferSize)
                        {
                            BufferToSend = new byte[InfoLen + ByteEncoded.Length];
                            Header.CopyTo(BufferToSend, 0);
                            fileId.CopyTo(BufferToSend, Header.Length);
                            size.CopyTo(BufferToSend, Header.Length + fileId.Length);

                            NumOfFrames = Encoding.Default.GetBytes(1.ToString());
                            NumOfFrames.CopyTo(BufferToSend, Header.Length + fileId.Length +
sizeLenght);

                            FrameNumber = Encoding.Default.GetBytes(1.ToString());
                            FrameNumber.CopyTo(BufferToSend, Header.Length + fileId.Length +
sizeLenght + NumOfFrameLenght);


                            ByteEncoded.CopyTo(BufferToSend, InfoLen);
                            bool flag = false;
                            while (!flag)
                            {

                                if (MessageBox.Show("Отправить?", "Файл",
MessageBoxButtons.YesNo) == DialogResult.Yes)
                                {

                                    flag = true;
                                    Port.Write(BufferToSend, 0, BufferToSend.Length);

                                    //loading.Hide();
                                    MessageBox.Show("Готово!");
                                    //loading.progressBar1.Value = 0;
                                    //loading.i = 1;

                                }
```

```
                                else
                                {
                                    flag = true;
                                    //loading.Hide();
                                    //loading.progressBar1.Value = 0;
                                    MessageBox.Show("Вы отменили передачу файла.");
                                    // loading.i = 1;
                                }
                            }
                        }
                        else
                        {
                            //EncodedByteIndex;
                            //Part_ByteEncodedIndex;

                            parts = (int)Math.Ceiling((double)ByteEncoded.Length /
(double)(Port.WriteBufferSize - InfoLen));
                            ProgressBar.Invoke(new EventHandler(delegate
                            {
                                ProgressBar.Visible = true;
                                ProgressBar.Maximum = parts;
                            }));
                            NumOfFrames = Encoding.Default.GetBytes(parts.ToString());

                            for (i = 0; i < parts; i++)
                            {
                                EncodedByteIndex = i * (Port.WriteBufferSize - InfoLen);
                                Part_ByteEncodedIndex = (Port.WriteBufferSize - InfoLen);

                                byte[] Part_ByteEncoded = new byte[Part_ByteEncodedIndex];

                                int Part_Len = 0;
                                if (((ByteEncoded.Length - EncodedByteIndex) >=
Part_ByteEncodedIndex))
                                {
                                    Part_Len = Part_ByteEncodedIndex;
                                }

                                else if (ByteEncoded.Length - EncodedByteIndex > 0)
                                {
                                    Part_Len = ByteEncoded.Length - i *
(Port.WriteBufferSize - InfoLen);
                                }


                                BufferToSend = new byte[Port.WriteBufferSize];

                                Header.CopyTo(BufferToSend, 0);
                                fileId.CopyTo(BufferToSend, Header.Length);
                                size.CopyTo(BufferToSend, Header.Length + fileId.Length);

                                NumOfFrames.CopyTo(BufferToSend, Header.Length +
fileId.Length + sizeLenght);

                                FrameNumber = Encoding.Default.GetBytes((i + 1).ToString());
                                FrameNumber.CopyTo(BufferToSend, Header.Length +
fileId.Length + sizeLenght + NumOfFrameLenght);

                                Array.ConstrainedCopy(ByteEncoded, EncodedByteIndex,
BufferToSend, InfoLen, Part_Len);


                                if (IsConnected())
                                {
                                    Port.Write(BufferToSend, 0, BufferToSend.Length);
                                }
                                Log.Invoke(new EventHandler(delegate
```

```
                                {
                                    Log.AppendText("[" + DateTime.Now + "]: Отправка кадра "
+ (i + 1).ToString() + "\n");
                                    Log.ScrollToCaret();
                                }));
                                if (ProgressBar.Value != parts)
                                {
                                    ProgressBar.Invoke(new EventHandler(delegate
                                    {
                                        ProgressBar.Value++;
                                    }));
                                }

                                byte[] ByteCheck = new byte[1];

                                if (i > 0 && IsConnected())
                                {
                                    //Thread.Sleep(10);
                                    int WaitTime = 0;
                                    try
                                    {
                                        Port.Read(ByteCheck, 0, 1);
                                    }
                                    catch (Exception e)
                                    {
                                        Log.AppendText(e.Message);
                                        break;
                                    }

                                    while (ByteCheck[0] != STARTBYTE)
                                    {
                                        if (WaitTime <= 100)
                                        {
                                            Thread.Sleep(10);
                                            WaitTime += 10;
                                            Port.Read(ByteCheck, 0, 1);
                                        }
                                        else
                                        {
                                            MessageBox.Show("Передача файла прервана");
                                            break;
                                        }

                                    }
                                    if (IsConnected()) { continue;}
                                    Port.Read(ByteCheck, 0, 1);
                                    if (ByteCheck[0] == (int)FrameType.FRAME)
                                    {
                                        int n = FrameNumber.Length;//Port.BytesToRead;
                                        byte[] msgByteBuffer = new byte[n];

                                        Port.Read(msgByteBuffer, 0, n); //считываем
сообщение
                                        string Message =
Encoding.Default.GetString(msgByteBuffer);

                                        SuccessfulFrameNumber = int.Parse(Message);
                                    }

                                    if (i == SuccessfulFrameNumber)
                                    {
                                        continue;
                                    }



                                }
```

```
                                if (!IsConnected())
                                {
                                    Log.Invoke(new EventHandler(delegate
                                    {
                                        Log.AppendText("[" + DateTime.Now + "]: Передача
файла нарушена\n");
                                    }));
                                    DialogResult result;
                                    while (!IsConnected())
                                    {
                                        result = MessageBox.Show("Соединение прервано.
Передача нарушена.\n"
                                        + "Восстановите соединение и нажмите ОК для докачки
файла.\n"
                                        + "Иначе нажмите ОТМЕНА.",
                                        "Ошибка",
                                        MessageBoxButtons.OKCancel,
                                        MessageBoxIcon.Error);
                                        if (result == DialogResult.Cancel)
                                        {
                                            Log.Invoke(new EventHandler(delegate
                                            {
                                                Log.AppendText("[" + DateTime.Now + "]:
Передача файла отменена\n");
                                            }));
                                            ProgressBar.Invoke(new EventHandler(delegate
                                            {
                                                ProgressBar.Value = 0;
                                            }));
                                            b_ChooseFile.Invoke(new EventHandler(delegate
                                            {
                                                b_ChooseFile.Enabled = true;
                                            }));
                                            return;
                                        }
                                    }
                                    //BreakConnection = true;
                                    i = SuccessfulFrameNumber - 1;
                                    //break;
                                }

                            }
                            Log.Invoke(new EventHandler(delegate
                            {
                                Log.AppendText("[" + DateTime.Now + "]: Файл успешно
передан\n");
                            }));
                            ProgressBar.Invoke(new EventHandler(delegate
                            {
                                ProgressBar.Value = 0;
                            }));
                            b_ChooseFile.Invoke(new EventHandler(delegate
                            {
                                b_ChooseFile.Enabled = true;
                            }));
                            b_OpenPort.Invoke(new EventHandler(delegate
                            {
                                b_OpenPort.Enabled = true;
                            }));

                        }
                    }
                    else
                    {
                        Log.Invoke(new EventHandler(delegate
                        {
                            Log.AppendText("[" + DateTime.Now + "]: Передача файла
```

```
нарушена.\n" + "Последний успешный фрейм: " + SuccessfulFrameNumber.ToString());
                        }));


                        BreakConnection = true;
                        break;
                    }
                    break;
                #endregion

                default:
                    if (IsConnected())
                        Port.Write(Header, 0, Header.Length);
                    break;
            }
//Зачем такая конструкция?

        }


        public void InitializeHandlers()
        {
            Port.DataReceived += new SerialDataReceivedEventHandler(Port_DataReceived);
        }


        private void Port_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            if (Port.ReadByte() == STARTBYTE)
            {
                GetData(Port.ReadByte());
            }
        }


        byte[] file_buffer;

        public void GetData(int typeId)
        {
            FrameType type = (FrameType)typeId;

            byte[] ToDecode;
            byte[] Decoded;



            switch (type)
            {
                case FrameType.MSG:
                    #region MSG
                    if (IsConnected())
                    {
                        int n = Port.BytesToRead;
                        byte[] msgByteBuffer = new byte[n];

                        Port.Read(msgByteBuffer, 0, n); //считываем сообщение
                        string Message = Encoding.Default.GetString(msgByteBuffer);
                        Log.Invoke(new EventHandler(delegate
                        {
                            Log.AppendText("(" + Port.PortName + ") GetData: новое сообщение
> " + Message + "\n");
                        }));

                        WriteData(null, FrameType.ACK);
                    }
                    else
                    {
```

```
                        WriteData(null, FrameType.RET_MSG);
                    }
                    break;
                #endregion
                case FrameType.FILEOK:
                    #region FILEOK
                    if (IsConnected())
                    {
                        int n = Port.BytesToRead;
                        byte[] msgByteBuffer = new byte[n];

                        Port.Read(msgByteBuffer, 0, n); //считываем сообщение
                        string Message = Encoding.Default.GetString(msgByteBuffer);
                        Log.Invoke(new EventHandler(delegate
                        {
                            Log.AppendText("[" + DateTime.Now + "]: Получено предложение на
прием файла размером: " + Message + " байт\n");
                        }));
                        //SuccessfulFrameNumber = int.Parse(Message);
                        int Message_num = int.Parse(Message);
                        double fileSize = Math.Round((double)Message_num / 1024, 3);
                        if (MessageBox.Show("Получено предложение на прием файла. Размер: "
+ fileSize.ToString() + " Кбайт.\nПринять?", "Прием файла", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
                        {
                            WriteData("OK", FrameType.ACK);

                            b_ChooseFile.Invoke(new EventHandler(delegate
                            {
                                b_ChooseFile.Enabled = false;
                            }));
                            b_OpenPort.Invoke(new EventHandler(delegate
                            {
                                b_OpenPort.Enabled = false;
                            }));

                        }

                    }
                    else
                    {
                        MessageBox.Show("Нет соединения!", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    }
                    break;
                #endregion

                case FrameType.FILE:
                    while ((!IsConnected()) && (BreakConnection))
                    {
                        Port.DiscardInBuffer();
                        Log.Invoke(new EventHandler(delegate
                        {
                            Log.AppendText(
                            "[" + DateTime.Now + "]: "
                            + "Ожидание файла..."
                            + "\r\n");
                            Log.ScrollToCaret();
                            Thread.Sleep(1000);
                        }));
                    }
                    #region FILE
                    if (IsConnected())
                    {
                        byte fileId = (byte)Port.ReadByte();
                        string typeFile = TypeFileAnalysis(fileId);
```

```
                        byte[] size = new byte[sizeLenght];
                        Port.Read(size, 0, sizeLenght);
                        int ssize = (int)Double.Parse(Encoding.Default.GetString(size));

                        byte[] byte_NumOfFrames = new byte[NumOfFrameLenght];
                        Port.Read(byte_NumOfFrames, 0, NumOfFrameLenght);
                        int NumOfFrames =
(int)Double.Parse(Encoding.Default.GetString(byte_NumOfFrames));

                        ProgressBar.Invoke(new EventHandler(delegate
                        {
                            ProgressBar.Visible = true;
                            ProgressBar.Maximum = NumOfFrames;
                        }));

                        byte[] byte_FrameNumber = new byte[NumOfFrameLenght];
                        Port.Read(byte_FrameNumber, 0, NumOfFrameLenght);
                        int FrameNumber =
(int)Double.Parse(Encoding.Default.GetString(byte_FrameNumber));


                        if (FrameNumber == 1)
                        {

                            file_buffer = new byte[NumOfFrames * (Port.WriteBufferSize -
27)];

                        }

                        Log.Invoke(new EventHandler(delegate
                        {
                            Log.AppendText(
                            "[" + DateTime.Now + "]: "
                            + "Загружен кадр "
                            + FrameNumber.ToString()
                            + "\r\n");
                            Log.ScrollToCaret();
                        }));
                        int n = Port.WriteBufferSize - InfoLen;
                        byte[] newPart = new byte[n];
                        Port.Read(newPart, 0, n);

                        newPart.CopyTo(file_buffer, n * (FrameNumber - 1));
                        if (ProgressBar.Value != FrameNumber & ProgressBar.Value !=
ProgressBar.Maximum)
                        {
                            ProgressBar.Invoke(new EventHandler(delegate
                            {
                                ProgressBar.Value++;
                            }));
                        }
                        WriteData(FrameNumber.ToString(), FrameType.FRAME);

                        if (FrameNumber == NumOfFrames)
                        {
                            Decoded = new byte[ssize];
                            ToDecode = new byte[2];

                            for (int i = 0; i < ssize; i++)
                            {
                                ToDecode[0] = file_buffer[i * 2];
                                ToDecode[1] = file_buffer[(i * 2) + 1];
                                Decoded[i] = Hamming.Decode(ToDecode);
                            }
                            Log.Invoke(new EventHandler(delegate
                            {
                                Log.AppendText(
```

```
                                "[" + DateTime.Now + "]: "
                                + "Файл успешно получен"
                                + "\r\n");
                                Log.ScrollToCaret();
                                b_ChooseFile.Enabled = true;
                                b_OpenPort.Enabled = true;
                            }));

                            SaveFileDialog saveFileDialog = new SaveFileDialog();

                            MainForm.Invoke(new EventHandler(delegate
                            {
                                saveFileDialog.FileName = "";
                                saveFileDialog.Filter = "TypeFile (*." + typeFile + ")|*." +
typeFile + "|All files (*.*)|*.*";
                                if (DialogResult.OK == saveFileDialog.ShowDialog())
                                {
                                    File.WriteAllBytes(saveFileDialog.FileName, Decoded);
                                    //WriteData(null, FrameType.ACK);
                                    Log.Invoke(new EventHandler(delegate
                                    {
                                        Log.AppendText(
                                        "[" + DateTime.Now + "]: "
                                        + "Файл сохранен"
                                        + "\r\n");
                                        Log.ScrollToCaret();
                                        b_ChooseFile.Enabled = true;
                                        b_OpenPort.Enabled = true;
                                    }));
                                }
                                else
                                {
                                    // MessageBox.Show("Отмена ");
                                    Log.Invoke(new EventHandler(delegate
                                    {
                                        Log.AppendText(
                                        "[" + DateTime.Now + "]: "
                                        + "Вы не сохранили файл"
                                        + "\r\n");
                                        Log.ScrollToCaret();
                                    }));

                                }
                            }));
                            ProgressBar.Invoke(new EventHandler(delegate
                            {
                                ProgressBar.Value = 0;
                            }));
                        }

                    }
                    else
                    {
                        WriteData(null, FrameType.ERR_FILE);
                    }

                    break;
                #endregion
                //=====================================================

                case FrameType.ACK:
                    #region ACK
                    WriteData(FilePath, FrameType.FILE);
                    break;
                #endregion

                case FrameType.RET_MSG:
```

```
            #region RET_MSG
            Log.AppendText("Ошибка отправки! Нет соединения\n");
            break;
        #endregion

        case FrameType.ERR_FILE:
            #region RET_FILE
            Log.AppendText("Ошибка отправки файла! Нет соединения\n");
            break;
            #endregion
    }
}

private RichTextBox _Log; //штука, чтобы видеть, что творится
public RichTextBox Log
{
    get
    {
        return _Log;
    }
    set
    {
        _Log = value;
    }
}

private Button _b_ChooseFile;
public Button b_ChooseFile
{
    get
    {
        return _b_ChooseFile;
    }
    set
    {
        _b_ChooseFile = value;
    }
}

private Button _b_Connection;
public Button b_Connection
{
    get
    {
        return _b_Connection;
    }
    set
    {
        _b_Connection = value;
    }
}

private Button _b_OpenPort;
public Button b_OpenPort
{
    get
    {
        return _b_OpenPort;
    }
    set
    {
        _b_OpenPort = value;
    }
}

private ProgressBar _ProgressBar;
public ProgressBar ProgressBar
```

```
{
    get
    {
        return _ProgressBar;
    }
    set
    {
        _ProgressBar = value;
    }
}
private Form _mainForm;
public Form MainForm
{
    get
    {
        return _mainForm;
    }
    set
    {
        _mainForm = value;
    }
}


private string TypeFileAnalysis(byte fileId)
{
    switch (fileId)
    {
        case 1:
            return "txt";
        case 2:
            return "png";
        case 3:
            return "pdf";
        case 4:
            return "docx";
        case 5:
            return "jpeg";
        case 6:
            return "avi";
        case 7:
            return "mp3";
        case 8:
            return "rar";
        default:
            return "typenotfound";
    }
}

private byte TypeFile_to_IdFile(string str)
{
    switch (str)
    {
        case "txt":
            return 1;
        case "png":
            return 2;
        case "pdf":
            return 3;
        case "docx":
            return 4;
        case "jpeg":
            return 5;
        case "avi":
            return 6;
        case "mp3":
            return 7;
```

```
                case "rar":
                    return 8;
                default:
                    return 9;
            }
        }

    }
}
```

3.4. Hamming.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ComPort
{
    class Hamming
    {
        public static int ErrorDigit(byte Error)
        {
            string tmp = Error.bin();
            int digit = 0;
            for (int i = 0; i < tmp.Length; i++)
            {
                digit += Int32.Parse(tmp[i].ToString());
            }
            return digit;
        }
        /// <summary>
        /// Кодирует один информационный байт в два кодированных
        /// </summary>
        /// <param name="ToBeEncoded">Байт который нужно закодировать</param>
        /// <returns>Массив из двух элементов</returns>
        public static byte[] HammingEncode74(byte ToBeEncoded)
        {
            byte[] Array = new byte[2];
            int i = 0;
            int j = 0;
            StringBuilder temp = new StringBuilder(ToBeEncoded.bin());
            while (temp.Length < 8)
            {
                temp = new StringBuilder("0" + temp);
            }
            for (j = 0; j < 2; j++)
            {
                StringBuilder forHalfByte = new StringBuilder("0000");
                for (i = 0; i < 4; i++)
                {
                    forHalfByte[i] = temp[(j * 4) + i];
                }
                //HalfByte=forHalfByte.ToString().b2();
                StringBuilder xxx = new StringBuilder("000" + forHalfByte);
                xxx[0] = xxx[3];
                xxx[1] = xxx[4];
                xxx[2] = xxx[5];
                xxx[4] = xxx[6];
                xxx[3] = Convert.ToChar(xxx[0] ^ xxx[1] ^ xxx[2]);
                xxx[5] = Convert.ToChar(xxx[0] ^ xxx[1] ^ xxx[4]);
                xxx[6] = Convert.ToChar(xxx[0] ^ xxx[2] ^ xxx[4]);
                Array[j] = xxx.ToString().b2();
            }
            return Array;
        }
        public static string HammingDecode74(byte ToBeDecoded)
        {
            StringBuilder temp = new StringBuilder("0000", 4);
            StringBuilder ToDecode = new StringBuilder(ToBeDecoded.bin());
            if (ToDecode.Length < 7)
                do
                {
                    ToDecode = new StringBuilder("0" + ToDecode.ToString());
                } while (ToDecode.Length < 7);
```

```
            temp[0] = ToDecode[0];
            temp[1] = ToDecode[1];
            temp[2] = ToDecode[2];
            temp[3] = ToDecode[4];
            return temp.ToString();
        }
        public static byte HammingSimptome74(byte ToBeDecoded)
        {
            StringBuilder temp = new StringBuilder("000", 3);
            StringBuilder ToDecode = new StringBuilder(ToBeDecoded.bin());
            if (ToDecode.Length < 7)
                do
                {
                    ToDecode = new StringBuilder("0" + ToDecode.ToString());
                } while (ToDecode.Length < 7);
            temp[2] = Convert.ToChar(((ToDecode[0] ^ ToDecode[2]) ^ (ToDecode[4] ^
ToDecode[6])).ToString());
            temp[1] = Convert.ToChar(((ToDecode[0] ^ ToDecode[1]) ^ (ToDecode[4] ^
ToDecode[5])).ToString());
            temp[0] = Convert.ToChar(((ToDecode[0] ^ ToDecode[1]) ^ (ToDecode[2] ^
ToDecode[3])).ToString());
            return temp.ToString().b2();
        }
        public static byte HammingCorrection74(byte code, int number)
        {
            StringBuilder temp = new StringBuilder(code.bin());
            if (temp.Length < 7)
                do
                {
                    temp = new StringBuilder("0" + temp.ToString());
                } while (temp.Length < 7);
            temp[7 - number] = (char)(temp[7 - number] ^ 1);
            return temp.ToString().b2();
        }
        public static byte Decode(byte[] OneEncodedByteInTwoBytes)
        {
            if (OneEncodedByteInTwoBytes.Length != 2)
            {
                return 0;
            }
            string outgoing = string.Empty;
            for (int i = 0; i < 2; i++)
            {
                byte AfterErrorCode = OneEncodedByteInTwoBytes[i];
                byte Symptom = Hamming.HammingSimptome74(AfterErrorCode); // Определение
симптома
                byte CorrectedCode; // Скорректированный код
                if (Convert.ToBoolean(Symptom)) // Если имеется ненулевой симптом
                {
                    CorrectedCode = Hamming.HammingCorrection74(AfterErrorCode, Symptom); //
Корректируем
                }
                else
                {
                    CorrectedCode = AfterErrorCode; // Не корректируем
                }
                outgoing += Hamming.HammingDecode74(CorrectedCode); // Декодируем
            }
            return outgoing.b2();
        }
    }
    class MyStrComparer : IEqualityComparer<string>
    {
        public bool Equals(string s1, string s2)
        {
            if (s1.Contains(s2)) return true;
            else return false;
```

```
        }
        public int GetHashCode(string st)
        {
            return st.Length;
        }
    }
    static class MyExtensionClass
    {
        public static string bin(this Byte input)
        {
            return Convert.ToString(input, 2);
        }
        public static byte b2(this string input)
        {
            return Convert.ToByte(input, 2);
        }
        public static Int16 b22(this string input)
        {
            return Convert.ToInt16(input, 2);
        }
    }
}
```