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:

ИУ5-65Б		**Усынин Ю. А.**
(Группа)	(Подпись, дата)	(Фамилия И.О.)
ИУ5-65Б		**Камалов М. Р.**
(Группа)	(Подпись, дата)	(Фамилия И.О.)
ИУ5-65Б		**Погосян С. Л.**
(Группа)	(Подпись, дата)	(Фамилия И.О.)

Руководитель курсовой работы:

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

Консультант:

(Подпись, дата)	(Фамилия И.О.)

Москва – 2021г.

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 номер разряда с ошибкой
HammingEncode1511	Кодирует 11-битый информационный вектор в 15
HammingDecode1511	Формирует из заведомо верного 15-битного закодированного вектора 11-битный информационный
HammingSimptome1511	Вычисляет синдром ошибки
HammingCorrection1511	Исправляет ошибку в закодированном 15-битном векторе
Decoded	Получая на вход 15-битный закодированный вектор, исправляет в нем ошибки и возвращает 11-битый информационный вектор

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 ComForm
{

    public partial class Form1 : Form
    {
        Connection com1 = 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 = b_OpenPort;
            b_con.Enabled = false;
            b_ChooseFile.Enabled = false;
            b_Connection.Enabled = false;
            richTextBox1.AppendText("Добро пожаловать!\nПеред началом работы выберите
порт из списка и откройте его.\n\n");
        }

        /// <summary>
        /// Пишет в лог, есть ли соединение
        /// </summary>
        private void b_con_Click(object sender, EventArgs e)
        {
            if (com1.IsConnected())
            {
                richTextBox1.AppendText("[" + DateTime.Now + "]: " +
cb_PortNames.SelectedItem.ToString() + ": Соединение установлено\n");
            }
            else
            {
                richTextBox1.AppendText("[" + DateTime.Now + "]: " +
cb_PortNames.SelectedItem.ToString() + ": Соединение отсутствует\n");
            }
        }

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

```
                com1.Log = richTextBox1;

                if (com1.Port.IsOpen)
                {
                    if (com1.ClosePort())
                    {
                        richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " +
com1.Port.PortName + " закрыт\n");
                        b_con.Enabled = false;
                        b_ChooseFile.Enabled = false;
                        b_Connection.Enabled = false;
                        cb_PortNames.Enabled = true;

                        b_OpenPort.Text = "Открыть порт";
                    }

                }
                else //открываем
                {
                    com1.setPortName(cb_PortNames.SelectedItem.ToString());

                    if (com1.OpenPort())
                    {
                        richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " +
com1.Port.PortName + " открыт\n");
                        b_con.Enabled = true;
                        b_ChooseFile.Enabled = true;
                        b_Connection.Enabled = true;
                        cb_PortNames.Enabled = false;

                        b_OpenPort.Text = "Закрыть порт";
                    }

                }

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

        private void b_ChooseFile_Click(object sender, EventArgs e)
        {
            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 2-х ПК, соединенных через
интерфейс RS-232C,\n" +
                            "с функцией передачи файла и возможностью докачки после
восстановления прерванной связи.\n\n" +
                            "Программу разработали студенты МГТУ им. Н.Э.Баумана группы
ИУ5-65:\n\n" +
                            "Погосян С.Л. (прикладной уровень)\n"
                            + "Камалов М.Р. (канальный уровень)\n"
                            +"Усынин Ю.А. (физический уровень)","О
                            программе",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Information);
        }

        private void b_Connection_Click(object sender, EventArgs e)
        {
            if (!com1.Port.DtrEnable)
            { //если выключен
                com1.Port.DtrEnable = true;
                b_Connection.Text = "Разорвать соединение";

                if (com1.IsConnected())
                {
                    richTextBox1.AppendText("[" + DateTime.Now + "]: Соединение успешно
установлено!\n");
                }
                else
                {
                    richTextBox1.AppendText("[" + DateTime.Now + "]: Порт " +
com1.Port.PortName + " готов к передаче данных, требуется подключение второго порта\n");
                }
            }
            else
            {
                com1.Port.DtrEnable = false;
                b_Connection.Text = "Установить соединение";
                richTextBox1.AppendText("[" + DateTime.Now + "]: Соединение было
разорвано\n");
            }


        }
    }
}
```

3.2. *Program.cs*

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

namespace ComForm
{
    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 ComForm
{
    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;
        }



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

        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);
                        //Log.AppendText("[" + DateTime.Now + "] Отправлено ACK согласие
на принятие файла: " + Telegram_s + "\n");
                    }
                    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);

                        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);
                        //Log.AppendText("[" + DateTime.Now + "] Получен кадр " +
Telegram_s + " .Отправлено подтверждение о получении\n");
                        //SuccessfulFrameNumber = int.Parse(Telegram_s);
                    }
                    else
                    {
                        Log.Invoke(new EventHandler(delegate
                        {
                            Log.AppendText("[" + DateTime.Now + "]: Передача файла
нарушена.");
                        }));
                        //MessageBox.Show("Соединение прервано. Передача нарушена.4");
                        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);

                                //Log.AppendText("[" + DateTime.Now + "]: Отправляется
фрейм: " + (SuccessfulFrameNumber + 1).ToString() + "\n");
                                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);
                                        //Log.Invoke(new EventHandler(delegate
                                        //{
                                        //    Log.AppendText("[" + DateTime.Now + "]
Получено подтверждение об успешной доставке кадра " + Message + "\n");
                                        //}));
                                        SuccessfulFrameNumber = int.Parse(Message);
                                    }

                                    if (i == SuccessfulFrameNumber)
                                    {
                                        continue;
                                    }


                                    //if (i != SuccessfulFrameNumber)
                                    //{

                                    //    MessageBox.Show("Передача файла нарушена.1");
                                    //    break;
                                    //}
                                }
                                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;
                                            }));
                                            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());
                    }));
                    //MessageBox.Show("Соединение прервано. Передача нарушена.3");

                    BreakConnection = true;
                    break;
                }
                break;
            #endregion

            default:
                if (IsConnected())
                    Port.Write(Header, 0, Header.Length);
                break;
        }
//Зачем такая конструкция?
        //Log.Invoke(new EventHandler(delegate
        //{
        //    Log.AppendText("sent frame " + type + "\n"); //всё записываем, мы же
снобы
        //}));
    }


    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;

            /*Log.Invoke(new EventHandler(delegate
            {
                Log.AppendText("get frame " + type +"\n");
            }));*/


            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
                        {
                            Log.AppendText("[" + DateTime.Now + "]: Передача файла
отменена");

                        }

                    }
                    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.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 ComForm
{
    class Hamming
    {
        public static int ErorDigit(char[] Error)
        {


            int digit = 0;
            for (int i = Error.Length - 1; i >= 0; i--)
            {
                int s = int.Parse(Convert.ToString(Error[4 - i + -1]));
                digit += (s * (int)(Math.Pow(2, 4 - i + -1)));

            }
            //Console.WriteLine(digit);
            return digit - 1;
        }
        /// <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 char[] HamingEncode1511(char[] ToBeEncoded)
        {
            char[] Array = new char[15];
            int i = 0;
            int j = 0;
            StringBuilder temp = new StringBuilder(ToBeEncoded.ToString());


            //HalfByte=forHalfByte.ToString().b2();
            StringBuilder xxx = new StringBuilder(Array.ToString());
            Array[0] = ToBeEncoded[0];
            Array[1] = ToBeEncoded[1];
            Array[2] = ToBeEncoded[2];
            Array[3] = ToBeEncoded[3];
            Array[4] = ToBeEncoded[4];
            Array[5] = ToBeEncoded[5];
            Array[6] = ToBeEncoded[6];
            Array[8] = ToBeEncoded[7];
            Array[9] = ToBeEncoded[8];
            Array[10] = ToBeEncoded[9];
            Array[12] = ToBeEncoded[10];
            Array[7] = Convert.ToChar(Array[0] ^ Array[1] ^ Array[2] ^ Array[3] ^ Array[4] ^
Array[5] ^ Array[6]);

            Array[11] = Convert.ToChar(Array[0] ^ Array[1] ^ Array[2] ^ Array[3] ^ Array[8]
^ Array[9] ^ Array[10]);

            Array[13] = Convert.ToChar(Array[0] ^ Array[1] ^ Array[4] ^ Array[5] ^ Array[8]
^ Array[9] ^ Array[12]);
            Array[14] = Convert.ToChar(Array[0] ^ Array[2] ^ Array[4] ^ Array[6] ^ Array[8]
^ Array[10] ^ Array[12]);
            //for (j = 0; j < xxx.Length; j++)
            //{
            //    Array[j]=xxx[j];
            //}
```

```
            return Array;
        }
        public static char[] HamingDecode1511(char[] ToBeDecoded)
        {
            char[] Array = new char[11];
            StringBuilder temp = new StringBuilder(ToBeDecoded.ToString());
            Array[0] = ToBeDecoded[0];
            Array[1] = ToBeDecoded[1];
            Array[2] = ToBeDecoded[2];
            Array[3] = ToBeDecoded[3];
            Array[4] = ToBeDecoded[4];
            Array[5] = ToBeDecoded[5];
            Array[6] = ToBeDecoded[6];
            Array[7] = ToBeDecoded[8];
            Array[8] = ToBeDecoded[9];
            Array[9] = ToBeDecoded[10];
            Array[10] = ToBeDecoded[12];

            return Array;
        }
        public static int HamingSindrome1511(char[] ToBeDecoded)
        {
            int[] Array = new int[4];
            int digit = 0;
            StringBuilder temp = new StringBuilder(ToBeDecoded.ToString());
            Console.WriteLine(ToBeDecoded);
            Array[3] = ((ToBeDecoded[6] ^ ToBeDecoded[5] ^ ToBeDecoded[4] ^ ToBeDecoded[3] ^
ToBeDecoded[2] ^ ToBeDecoded[1] ^ ToBeDecoded[0] ^ ToBeDecoded[7]));
            Array[2] = ((ToBeDecoded[10] ^ ToBeDecoded[9] ^ ToBeDecoded[8] ^ ToBeDecoded[3]
^ ToBeDecoded[2] ^ ToBeDecoded[1] ^ ToBeDecoded[0] ^ ToBeDecoded[11]));
            Array[1] = ((ToBeDecoded[12] ^ ToBeDecoded[9] ^ ToBeDecoded[8] ^ ToBeDecoded[5]
^ ToBeDecoded[4] ^ ToBeDecoded[1] ^ ToBeDecoded[0] ^ ToBeDecoded[13]));
            Array[0] = ((ToBeDecoded[12] ^ ToBeDecoded[10] ^ ToBeDecoded[8] ^ ToBeDecoded[6]
^ ToBeDecoded[4] ^ ToBeDecoded[2] ^ ToBeDecoded[0] ^ ToBeDecoded[14]));
            for (int i = 0; i < 4; i++)
            {

                digit += (Array[i] * (int)(Math.Pow(2, i)));

            }
            Console.WriteLine(Array[3]);
            Console.WriteLine(Array[2]);
            Console.WriteLine(Array[1]);
            Console.WriteLine(Array[0]);
            return 15 - digit;
        }
        public static char[] HamingCorrection1511(char[] code, int number)
        {
            if (number == 0)
            {
                return code;
            }
            else
            {
                if (code[number - 1] == '0')
                {
                    code[number - 1] = '1';
                }
                else
                {
                    code[number - 1] = '0';
                }


            }
            return code;
```

```
        }
        public static char[] Decoded(char[] ToBeDecoded)
        {

            int Sindrome = Hamming.HamingSindrome1511(ToBeDecoded); // Определение синдрома
            char[] CorrectedCode;
            if (Sindrome == 15) // Если имеется ненулевой синдрома
            {
                CorrectedCode = Hamming.HamingCorrection1511(ToBeDecoded, (Sindrome)); //
Корректируем
            }
            else
            {
                CorrectedCode = ToBeDecoded; // Не корректируем
            }
            char[] outgoing = Hamming.HamingDecode1511(CorrectedCode); // Декодируем

            return outgoing;
              }
    }
  }
```