Московский государственный технический университет имени Н. Э. Баумана.

Факультет “Информатика и системы управления”.

Кафедра “Системы обработки информации и управления”.

|  |  |
| --- | --- |
| Утверждаю: |  |
| Галкин В.А. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2018г. |

Курсовая работа

по курсу

Сетевые технологии в АСОИУ

«Программа пересылки файлов»

Описание программы

(вид документа)

бумага А4

(вид носителя)

17

(количество листов)

Вариант 16.

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛИ: |  |
| студенты группы ИУ5-63 |  |
| Шубникова М.В. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Костенкова Ю.В. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Егоров С.А. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г. |

Москва 2018

Оглавление

[1. Введение 3](#_Toc305591231)

[2. Класс mainwindow:window 3](#_Toc305591232)

[2.1. Переменные 3](#_Toc305591233)

[2.2. События 3](#_Toc305591234)

[2.3. Методы 4](#_Toc305591235)

[3. Листинг 5](#_Toc305591236)

# Введение

Программный продукт написан с использованием технологии Windows Presentation Foundation на языке программирования C#.

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

# 2. Класс MainWindow:Window

- класс определяющий главного окна.

## 2.1. Переменные

* *SynchronizationContext UIContext* – контекст синхронизации для обращения из одного потока к элементам другого;
* *bool Conn* – состояние соединения;
* *SerialPort ComPort* – объект, описывающий COM-порт;
* *Thread RThread* – объект, описывающий параллельный поток чтения данных из буфера COM-порта;
* *Thread CThread* – объект, описывающий параллельный поток проверки соединения;
* *FileStream SFileStream* – файловый поток для чтения;
* *FileStream PFileStream* – файловый поток для записи;
* *string SourcePath* – путь на передающем ПК к передаваемому файлу;
* *string PurposePath* – путь на принимающем ПК для сохранения принимаемого файла;
* *string size* – размер уже полученной части файла;
* *long p* – последняя записываемая позиция чтения из файлового потока;
* *bool ZagR* – принят заголовок файла;
* *bool InfR* – принят файл;
* *bool ZagS* – заголовок файла доставлен;
* *bool InfS* – файл доставлен;
* *bool Podtv* – необходимость подтверждения отправки;
* *int ChisloPovtorov* – счетчик подряд идущих ошибок передач.

## 2.2. События

* *private void Window\_Loaded(object sender, RoutedEventArgs e*) – событие, возникающее при загрузке окна:
  + *object sender* – объект, вызывающий событие;
  + *RoutedEventArgs e* – аргументы для события;
* *private void Window\_Closed(object sender, EventArgs e)* – событие, возникающее при закрытии окна:
  + *object sender* – объект, вызывающий событие;
  + *RoutedEventArgs e* – аргументы для события;
* *private void button1\_Click(object sender, RoutedEventArgs e)* – событие, возникающее при нажатии кнопки «Отправить файл»:
  + *object sender* – объект, вызывающий событие;
  + *RoutedEventArgs e* – аргументы для события;
* *private void button\_Click(object sender, RoutedEventArgs e)* – событие, возникающее при нажатии кнопки «Отправить файл пофторно»:
  + *object sender* – объект, вызывающий событие;
  + *RoutedEventArgs e* – аргументы для события;
* *private void MenuItemClose\_Click(object sender, RoutedEventArgs e*) – событие, возникающее при нажатии на кнопку «Закрыть» в меню «Действия»:
  + *object sender* – объект, вызывающий событие;
  + *RoutedEventArgs e* – аргументы для события;
* *private void MenuItemHelpA\_Click(object sender, RoutedEventArgs e)* – событие, возникающее при нажатии на кнопку «О программе» в меню «Справка»:
  + *object sender* – объект, вызывающий событие;
  + *RoutedEventArgs e* – аргументы для события;
* *private void MenuItemHelpD\_Click(object sender, RoutedEventArgs e) событие*, возникающее при нажатии на кнопку «Разработка» в меню «Справка»:
  + *object sender* – объект, вызывающий событие;
  + *RoutedEventArgs e* – аргументы для события.

## 2.3. Методы

* *public void Connect()* – функция, проверки соединения;
* *public void Read()* – функция, чтения информации из COM-порта;
* *void ACK()* – функция, отправки положительной квитанции;
* *void NAK()* – функция, отправки отрицательной квитанции;
* *void YES()* – функция, отправки согласия на прием файла;
* *void NO()* – функция, отправки сообщения о несогласии на прием файла;
* *void EOT()* – функция, отправки сообщения о завершении передачи;
* *bool? Sohranenie(string Zagolovok)* – функция открытия диалога сохранения файла:
  + *string Zagolovok* – заголовок принимаемого файла;
* *bool? Dokachka(string Zagolovok*) – функция открытия диалога сохранения файла:
  + *string Zagolovok* – заголовок принимаемого файла;
* *byte[] Upakovat(char Type)* – функция для формирования неинформационного кадра:
  + *char Type* – тип кадра;
* *byte[] Upakovat(byte[] InfByte, char Type, long Length)* – функция для формирования информационного кадра, кадра докачки и кадра размера уже полученного файла:
  + *byte[] InfByte* – массив информационных байтов;
  + *char Type* – тип кадра;
  + *long Length* – длина информационного поля кадра;
* *byte[] Kodir(byte[] Ish, int k, string PorPolinom)* – функция кодирования:
  + *byte[] Ish* – массив байтов для кодирования;
  + *int k* – число кодируемых битов;
  + *string PorPolinom* – порождающий полином;
* *string VichOstatka(string Delimoe, string PorPolinom*) – функция вычисление остатка после деления по модулю 2 двоичного числа на порождающий полином:
  + *string Delimoe* – делимое;
  + *string PorPolinom* – порождающий полином;
* *byte[] Dekodir(string Ish, int k, string PorPolinom)* – функция декодирования:
  + *string Ish* – декодируемая строка;
  + *int k* – число кодируемых битов;
  + *string PorPolinom* – порождающий полином;
* *bool Proverka(string Ish, int k, string PorPolinom*) – функция проверки закодированного кадра на наличие ошибок:
  + *string Ish* – декодируемая строка
  + *int k* – число кодируемых битов;
  + *string PorPolinom* – порождающий полином;

# 3*.* Листинг

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.IO;

using System.IO.Ports;

using System.Threading;

using Microsoft.Win32;

namespace NetworkTechnology

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

SynchronizationContext UIContext;

bool Conn = false; //состояние соединения

SerialPort ComPort;

Thread RThread;

Thread CThread;

FileStream SFileStream;

FileStream PFileStream;

string SourcePath;

string PurposePath;

string size; //вес уже полученной части файла

long p; //последняя записываемая позиция

bool ZagR = false; //получен заголовок

bool InfR = false; //получение информационной части

bool ZagS = false; //отправлен заголовок

bool InfS = false; //отправка иноформационной части

bool Podtv = false; //необходимо подтверждение

int a = 0;

int ChisloPovtorov = 0;

public MainWindow()

{

InitializeComponent();

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

button1.IsEnabled = false;

button.IsEnabled = false;

UIContext = SynchronizationContext.Current;

RThread = new Thread(Read);

CThread = new Thread(Connect);

ComPort = new SerialPort("COM5", 9600, Parity.None, 8, StopBits.One);

ComPort.Handshake = Handshake.None;

ComPort.DtrEnable = true;

ComPort.RtsEnable = false;

ComPort.ReadTimeout = 500;

ComPort.WriteTimeout = 500;

ComPort.Open();

CThread.Start();

RThread.Start();

}

private void Window\_Closed(object sender, EventArgs e)

{

Conn = false;

RThread.Abort();

CThread.Abort();

ComPort.Close();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog sourceBrowse = new OpenFileDialog();

sourceBrowse.Multiselect = false;

sourceBrowse.Title = "Выберите передаваемый файл";

bool? BrowseOp = sourceBrowse.ShowDialog();

string FileName;

if (BrowseOp == true)

{

SourcePath = sourceBrowse.FileName;

SFileStream = new FileStream(SourcePath, FileMode.Open, FileAccess.Read);

progressBar1.Maximum = (SFileStream.Length / 50) + 1;

progressBar1.Value = 0;

FileName = SourcePath.Substring(SourcePath.LastIndexOf('\\') + 1);

byte[] Zagolovok = new byte[FileName.Count()];

for (int i = 0; i < FileName.Count(); i++)

{

Zagolovok[i] = Convert.ToByte(FileName[i]);

}

byte[] telegram = Kodir(Upakovat(Zagolovok, 'I', Zagolovok.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

a = 1;

}

}

private void button\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog sourceBrowse = new OpenFileDialog();

sourceBrowse.Multiselect = false;

sourceBrowse.Title = "Выберите файл для повторной отправки";

bool? BrowseOp = sourceBrowse.ShowDialog();

string FileName;

if (BrowseOp == true)

{

SourcePath = sourceBrowse.FileName;

SFileStream = new FileStream(SourcePath, FileMode.Open, FileAccess.Read);

progressBar1.Maximum = (SFileStream.Length / 50) + 1;

progressBar1.Value = 0;

FileName = SourcePath.Substring(SourcePath.LastIndexOf('\\') + 1);

byte[] Zagolovok = new byte[FileName.Count()];

for (int i = 0; i < FileName.Count(); i++)

{

Zagolovok[i] = Convert.ToByte(FileName[i]);

}

byte[] telegram = Kodir(Upakovat(Zagolovok, 'D', Zagolovok.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

a = 1;

}

}

public void Connect()

{

while (true)

{

if ((ComPort.DsrHolding == true) & (ZagR == false) & (ZagS == false))

{

Conn = true;

UIContext.Send(d => label2.Content = "Активно", null);

UIContext.Send(d => MenuItem\_Send.IsEnabled = true, null);

UIContext.Send(d => MenuItem\_Resum.IsEnabled = true, null);

UIContext.Send(d => button1.IsEnabled = true, null);

UIContext.Send(d => button.IsEnabled = true, null);

}

if ((ComPort.DsrHolding == false) & (ZagR == false) & (ZagS == false))

{

Conn = false;

UIContext.Send(d => label2.Content = "Отсутствует", null);

UIContext.Send(d => MenuItem\_Send.IsEnabled = false, null);

UIContext.Send(d => MenuItem\_Resum.IsEnabled = false, null);

UIContext.Send(d => button1.IsEnabled = false, null);

UIContext.Send(d => button.IsEnabled = false, null);

}

if (((ZagR == true) || (ZagS == true)) & (ComPort.DsrHolding == false))

{

Conn = false;

UIContext.Send(d => progressBar1.Visibility = Visibility.Hidden, null);

UIContext.Send(d => label4.Visibility = Visibility.Hidden, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = true, null);

UIContext.Send(d => button1.IsEnabled = true, null);

UIContext.Send(d => button.IsEnabled = true, null);

MessageBox.Show("Во время передачи возникла ошибка!\nПередача прервана!", "Компьютер1.");

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

if (a == 1)

{

SFileStream.Close();

SFileStream.Dispose();

}

else

{

PFileStream.Close();

PFileStream.Dispose();

}

}

Thread.Sleep(1000);

}

}

public void Read()

{

while (true)

{

while ((Conn == true) & (ComPort.CtsHolding == true))

{

string mess = "";

string message = "";

string s1 = "";

for (int i = 0; ComPort.BytesToRead > 0; i++)

{

s1 = Convert.ToString(ComPort.ReadByte(), 2);

if (s1.Count() < 8)

{

for (int j = 0; s1.Count() < 8; j++)

{

s1 = "0" + s1;

}

}

message += s1;

Thread.Sleep(30);

}

if (message.Count() > 0)

{

if (Proverka(message, 4, "1011") == true)

{

ChisloPovtorov++;

if (ChisloPovtorov >= 5)

{

UIContext.Send(d => label4.Visibility = Visibility.Hidden, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = true, null);

UIContext.Send(d => button1.IsEnabled = true, null);

UIContext.Send(d => button.IsEnabled = true, null);

MessageBox.Show("Во время передачи возникли ошибки!\nПередача прервана!", "Компьютер1.");

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

PFileStream.Close();

PFileStream.Dispose();

ChisloPovtorov = 0;

}

NAK();

}

else

{

byte[] DekMes = Dekodir(message, 4, "1011");

if (DekMes[1] == Convert.ToByte('A'))

{

ChisloPovtorov = 0;

ComPort.RtsEnable = false;

if ((ZagS == true) & (InfS == true) & (Podtv == false))

{

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

UIContext.Send(d => progressBar1.Visibility = Visibility.Hidden, null);

UIContext.Send(d => button1.IsEnabled = true, null);

UIContext.Send(d => button.IsEnabled = true, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = true, null);

MessageBox.Show("Файл передан!", "Компьютер1.");

SFileStream.Close();

SFileStream.Dispose();

}

if ((ZagS == true) & (InfS == false) & (Podtv == true))

{

UIContext.Send(d => progressBar1.Value++, null);

p = SFileStream.Position;

long k = SFileStream.Length - p;

if (k > 0)

{

byte[] inf;

if (k > 50)

{

inf = new byte[50];

for (int i = 0; i < 50; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

else

{

inf = new byte[k];

for (int i = 0; i < k; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

byte[] telegram = Kodir(Upakovat(inf, 'I', inf.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

}

else

{

InfS = true;

Podtv = false;

EOT();

}

}

if ((ZagS == false) & (Podtv == true))

{

ZagS = true;

}

}

if (DekMes[1] == Convert.ToByte('R'))

{

ChisloPovtorov++;

if (ChisloPovtorov >= 5)

{

UIContext.Send(d => progressBar1.Visibility = Visibility.Hidden, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = true, null);

UIContext.Send(d => button1.IsEnabled = true, null);

UIContext.Send(d => button.IsEnabled = true, null);

MessageBox.Show("Во время передачи возникла ошибка!\nПередача прервана!", "Компьютер1.");

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

ChisloPovtorov = 0;

SFileStream.Close();

SFileStream.Dispose();

}

else

{

if (ZagS == true)

{

byte[] inf;

long k = SFileStream.Length - p;

SFileStream.Position = p;

if (k > 50)

{

inf = new byte[50];

for (int i = 0; i < 50; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

else

{

inf = new byte[k];

for (int i = 0; i < k; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

byte[] telegram = Kodir(Upakovat(inf, 'I', inf.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

}

else

{

string FileName = SourcePath.Substring(SourcePath.LastIndexOf('\\') + 1);

byte[] Zagolovok = new byte[FileName.Count()];

for (int i = 0; i < FileName.Count(); i++)

{

Zagolovok[i] = Convert.ToByte(FileName[i]);

}

byte[] telegram = Kodir(Upakovat(Zagolovok, 'I', Zagolovok.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

}

}

}

if (DekMes[1] == Convert.ToByte('Y'))

{

UIContext.Send(d => button1.IsEnabled = false, null);

UIContext.Send(d => button.IsEnabled = false, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = false, null);

UIContext.Send(d => progressBar1.Visibility = Visibility.Visible, null);

byte[] inf;

p = SFileStream.Position;

long k = SFileStream.Length - p;

if (k > 50)

{

inf = new byte[50];

for (int i = 0; i < 50; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

else

{

inf = new byte[k];

for (int i = 0; i < k; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

Podtv = false;

byte[] telegram = Kodir(Upakovat(inf, 'I', inf.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

}

if (DekMes[1] == Convert.ToByte('N'))

{

MessageBox.Show("Принимающая сторона отказывается принимать файл!", "Компьютер1.");

SFileStream.Close();

SFileStream.Dispose();

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

}

if (DekMes[1] == Convert.ToByte('E'))

{

InfR = true;

UIContext.Send(d => label4.Visibility = Visibility.Hidden, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = true, null);

UIContext.Send(d => button1.IsEnabled = true, null);

UIContext.Send(d => button.IsEnabled = true, null);

MessageBox.Show("Файл принят!", "Компьютер1.");

ACK();

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

PFileStream.Close();

PFileStream.Dispose();

}

if (DekMes[1] == Convert.ToByte('S'))

{

UIContext.Send(d => button1.IsEnabled = false, null);

UIContext.Send(d => button.IsEnabled = false, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = false, null);

UIContext.Send(d => progressBar1.Visibility = Visibility.Visible, null);

for (int i = 0; i < Convert.ToInt32(DekMes[2]); i++)

{

mess = mess + Convert.ToChar(DekMes[3 + i]);

}

for (int i = 0; i == (Convert.ToInt32(mess)/50); i++)

{

UIContext.Send(d => progressBar1.Visibility = Visibility.Hidden, null);

}

SFileStream.Position = Convert.ToInt32(mess);

byte[] inf;

p = SFileStream.Position;

long k = SFileStream.Length - p;

if (k > 50)

{

inf = new byte[50];

for (int i = 0; i < 50; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

else

{

inf = new byte[k];

for (int i = 0; i < k; i++)

{

inf[i] = Convert.ToByte(SFileStream.ReadByte());

}

}

Podtv = false;

byte[] telegram = Kodir(Upakovat(inf, 'I', inf.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

}

if ((DekMes[1] == Convert.ToByte('I')) || (DekMes[1] == Convert.ToByte('D')))

{

if (DekMes.Count() == DekMes[2] + 4)

{

if (ZagR == false)

{

for (int i = 0; i < Convert.ToInt32(DekMes[2]); i++)

{

mess = mess + Convert.ToChar(DekMes[3 + i]);

}

ZagR = true;

ACK();

if (DekMes[1] == Convert.ToByte('D'))

{

if (MessageBox.Show("Продолжить получать файл " + mess + "?", "Компьютер1.Согласие на передачу", MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

if (Dokachka(mess) == true)

{

ZagR = true;

UIContext.Send(d => button1.IsEnabled = false, null);

UIContext.Send(d => button.IsEnabled = false, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = false, null);

UIContext.Send(d => label4.Visibility = Visibility.Visible, null);

size = Convert.ToString(PFileStream.Length);

byte[] FileSize = new byte[size.Count()];

for(int i = 0; i < size.Count(); i++)

{

FileSize[i] = Convert.ToByte(size[i]);

}

byte[] telegram = Kodir(Upakovat(FileSize, 'S', FileSize.Count()), 4, "1011");

ComPort.RtsEnable = false;

Podtv = true;

ComPort.Write(telegram, 0, telegram.Count());

Thread.Sleep(100);

ComPort.RtsEnable = true;

}

else

{

NO();

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

}

}

else

{

NO();

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

}

}

else

{

if (MessageBox.Show("Принять файл " + mess + "?", "Компьютер1.Согласие на передачу", MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

if (Sohranenie(mess) == true)

{

ZagR = true;

UIContext.Send(d => button1.IsEnabled = false, null);

UIContext.Send(d => button.IsEnabled = false, null);

UIContext.Send(d => MenuItem\_Action.IsEnabled = false, null);

UIContext.Send(d => label4.Visibility = Visibility.Visible, null);

YES();

}

else

{

NO();

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

}

}

else

{

NO();

ZagR = false;

InfR = false;

ZagS = false;

InfS = false;

}

}

}

else

{

ACK();

PFileStream.Write(DekMes, 3, Convert.ToInt32(DekMes[2]));

}

}

}

}

}

}

}

}

void ACK()

{

ComPort.RtsEnable = false;

ComPort.Write(Kodir(Upakovat('A'), 4, "1011"), 0, Kodir(Upakovat('A'), 4, "1011").Count());

ComPort.RtsEnable = true;

}

void NAK()

{

ComPort.RtsEnable = false;

ComPort.Write(Kodir(Upakovat('R'), 4, "1011"), 0, Kodir(Upakovat('R'), 4, "1011").Count());

ComPort.RtsEnable = true;

}

void YES()

{

ComPort.RtsEnable = false;

ComPort.Write(Kodir(Upakovat('Y'), 4, "1011"), 0, Kodir(Upakovat('Y'), 4, "1011").Count());

ComPort.RtsEnable = true;

}

void NO()

{

ComPort.RtsEnable = false;

ComPort.Write(Kodir(Upakovat('N'), 4, "1011"), 0, Kodir(Upakovat('N'), 4, "1011").Count());

ComPort.RtsEnable = true;

}

void EOT()

{

ComPort.RtsEnable = false;

ComPort.Write(Kodir(Upakovat('E'), 4, "1011"), 0, Kodir(Upakovat('E'), 4, "1011").Count());

ComPort.RtsEnable = true;

}

bool? Sohranenie(string Zagolovok)

{

SaveFileDialog purposeBrowse = new SaveFileDialog();

purposeBrowse.AddExtension = true;

purposeBrowse.Title = "Выберите место для сохранения файла";

purposeBrowse.FileName = Zagolovok;

bool? BrowseWr = purposeBrowse.ShowDialog();

if (BrowseWr == true)

{

PurposePath = purposeBrowse.FileName;

PFileStream = new FileStream(PurposePath, FileMode.Create, FileAccess.Write);

}

return BrowseWr;

}

bool? Dokachka(string Zagolovok)

{

OpenFileDialog purposeBrowse = new OpenFileDialog();

purposeBrowse.AddExtension = true;

purposeBrowse.Title = "Выберите недокаченный файл";

purposeBrowse.FileName = Zagolovok;

bool? BrowseWr = purposeBrowse.ShowDialog();

if (BrowseWr == true)

{

PurposePath = purposeBrowse.FileName;

PFileStream = new FileStream(PurposePath, FileMode.Open, FileAccess.Write);

PFileStream.Seek(0, SeekOrigin.End);

}

return BrowseWr;

}

byte[] Upakovat(char Type)

{

byte[] VByte;

if (Type == 'A' || Type == 'R' || Type == 'Y' || Type == 'N' || Type == 'E')

{

VByte = new byte[3];

VByte[0] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier);

VByte[1] = Convert.ToByte(Type);

VByte[2] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier);

}

else

{

VByte = new byte[0];

}

return VByte;

}

byte[] Upakovat(byte[] InfByte, char Type, long Length)

{

byte[] VByte;

if (Type == 'I' || Type == 'D' || Type =='S')

{

VByte = new byte[Length + 4];

VByte[0] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier);

VByte[1] = Convert.ToByte(Type);

VByte[2] = Convert.ToByte(Length);

for (int i = 3, j = 0; i < Length + 3; i++, j++)

{

VByte[i] = InfByte[j];

}

VByte[Length + 3] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier);

}

else

{

VByte = new byte[0];

}

return VByte;

}

byte[] Kodir(byte[] Ish, int k, string PorPolinom)

{

byte[] Vih;

int n = Ish.Count();

string s = "";

string p = "";

string s1 = "";

for (int i = 0; i < n; i++)

{

s1 = Convert.ToString(Ish[i], 2);

for (int j = 0; s1.Count() < 8; j++)

{

s1 = "0" + s1;

}

s = s + s1;

}

string Nuli = "";

for (int i = 0; i < PorPolinom.Count() - 1; i++)

{

Nuli = Nuli + "0";

}

for (int inach = 0, ikon = 0; ikon < s.Count(); ikon++)

{

if ((ikon + 1) % k == 0)

{

p = p + s.Substring(inach, k) + VichOstatka(s.Substring(inach, k) + Nuli, PorPolinom);

inach = ikon + 1;

}

}

int U = 8 - p.Count() % 8;

for (int i = 0; i < U; i++)

{

p = p + "0";

}

Vih = new byte[p.Count() / 8];

for (int i = 0; i < p.Count() / 8; i++)

{

Vih[i] = Convert.ToByte(p.Substring(8 \* i, 8), 2);

}

return Vih;

}

string VichOstatka(string Delimoe, string PorPolinom)

{

string Chastnoe;

int n = PorPolinom.Count();

for (int i = 0; i < Delimoe.Count() - n + 1; i++)

{

if (Delimoe[i] == PorPolinom[0])

{

Delimoe = Delimoe.Remove(i, 1);

Delimoe = Delimoe.Insert(i, "0");

for (int j = 1; j < n; j++)

{

if (Delimoe[i + j] == PorPolinom[j])

{

Delimoe = Delimoe.Remove(i + j, 1);

Delimoe = Delimoe.Insert(i + j, "0");

}

else

{

Delimoe = Delimoe.Remove(i + j, 1);

Delimoe = Delimoe.Insert(i + j, "1");

}

}

}

}

Chastnoe = Delimoe.Substring(Delimoe.Count() - n + 1);

return Chastnoe;

}

byte[] Dekodir(string Ish, int k, string PorPolinom)

{

byte[] Vih;

string p = "";

for (int i = 0; i < Ish.Count() - k - PorPolinom.Count() + 1; i += k + PorPolinom.Count() - 1)

{

p = p + Ish.Substring(i, k);

}

Vih = new byte[p.Count() / 8];

for (int i = 0; i < p.Count() / 8; i++)

{

Vih[i] = Convert.ToByte(p.Substring(8 \* i, 8), 2);

}

return Vih;

}

bool Proverka(string Ish, int k, string PorPolinom)

{

bool Oshibka = false;

string Nuli = "";

for (int i = 0; i < PorPolinom.Count() - 1; i++)

{

Nuli = Nuli + "0";

}

for (int i = 0; i < Ish.Count() - k - PorPolinom.Count() + 1; i += k + PorPolinom.Count() - 1)

{

if (VichOstatka(Ish.Substring(i, k + PorPolinom.Count() - 1), PorPolinom) != Nuli)

{

Oshibka = true;

break;

}

}

return Oshibka;

}

private void MenuItemClose\_Click(object sender, RoutedEventArgs e)

{

Close();

}

private void MenuItemHelpA\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Программа предназначена для передачи файла между двумя ЭВМ, соединенными нуль-модемно, через COM-порт.\nДля передачи файла нажмите кнопку \"Отправить файл\" или выберите пункт \"Отправить файл\" в меню \"Действия\"", "О программе");

}

private void MenuItemHelpD\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Выполнена в рамках курса \"Сетеввые технологии в АСОИУ\"\nИсполнители:\tЕгоров С.А. ИУ5-63\n\t\tКостенкова Ю.В. ИУ5-63\n\t\tШубникова М.В. ИУ5-63\nПреподаватель:\tГалкин В.А.", "Разработка");

}

}

}