Московский государственный технический университет имени Н. Э. Баумана.

Факультет “Информатика и системы управления”.

Кафедра “Системы обработки информации и управления”.

|  |  |
| --- | --- |
| Утверждаю: |  |
| Галкин В.А. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2016г. |

Курсовая работа

по курсу

Сетевые технологии

«Программа пересылки файлов»

Описание программы

(вид документа)

бумага А4

(вид носителя)

21

(количество листов)

Вариант 25

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛИ: |  |
| студенты группы ИУ5-64 |  |
| Вострокнутов И.Н. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Егоров А.С. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Мусин Т.М. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г. |

Москва 2016

Оглавление

[1. Введение 3](#_Toc450580287)

[2. Класс MainForm:Form 3](#_Toc450580288)

[2.1. Переменные 3](#_Toc450580289)

[2.2. События 4](#_Toc450580290)

[2.3. Методы 5](#_Toc450580291)

[3. Класс SettingsForm:Form 5](#_Toc450580292)

[2.1. Переменные 5](#_Toc450580293)

[2.2. События 5](#_Toc450580294)

[4*.* Листинг 6](#_Toc450580295)

# 1. Введение

Программный продукт написан с использованием технологии Windows Forms 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 MainForm:Form

- класс, определяющий главное окно.

## 2.1. Переменные

* SerialPort COMPort – объект, описывающий COM-порт;
* Thread ConnectionThread – поток для отслеживания соединения;
* Encoding ANSI –установка кодировки ANSI;
* bool BigFile – проверка на величину файла;
* static bool ErrorInfo – информация об ошибке;
* uint Frequency – частота чтения входящего потока;
* byte FinalizationStatus – текущий статус окончания передачи;
* bool Console – вывод в консоль;
* bool ConnStatus – текущий статус порта;
* string AppliedFileName – имя получаемого файла;
* Thread SynchronizationThread – поток на чтение;
* SynchronizationContext UIContext – контекст синхронизации для обращения из одного потока к элементам другого;
* bool RHeader – получен заголовок;
* uint IndexOfInfopacketIn – идентификатор для кадра (получение);
* uint MaxIndexOfInfopacketIn – количество кадров (получение);
* string SendedFileName – имя отправляемого файла;
* bool SHeader – отправлен заголовок;
* byte[][] WriteData – отправляемые данные;
* byte[] byFileData – файл, заносимый в одномерный массив;
* uint IndexOfInfopacketOut – идентификатор для кадра (отправка);
* int CountPackets – максимальный индекс текущего отправляемого файла;

## 2.2. События

* void ChooseFileButton\_Click(object sender, EventArgs e) – событие, возникающее при нажатии на кнопку выбора отправляемого файла:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void AuthorsToolStripMenuItem\_Click(object sender, EventArgs e) – событие, возникающее при нажатии на пункт меню «Об авторах»:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void ExitToolStripMenuItem\_Click (object sender, EventArgs e) – событие, возникающее при нажатии на пункт меню «Выход»:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void ChooseFileToolStripMenuItem\_Click (object sender, EventArgs e) – событие, возникающее при нажатии на пункт меню «Выбор файла…»:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void MainForm\_FormClosing (object sender, EventArgs e) – событие, возникающее при закрытии окна:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void ConnSettingsToolStripMenuItem\_Click (object sender, EventArgs e) – событие, возникающее при нажатии на пункт меню «Свойства соединения»:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void SendFileButton\_Click (object sender, RoutedEventArgs e) – событие, возникающее при нажатии кнопки «Отправить файл»:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void DeclineButton\_Click (object sender, RoutedEventArgs e) – событие, возникающее при нажатии на кнопку «Отклонить»:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void ApplyButton\_Click (object sender, RoutedEventArgs e) – событие, возникающее при нажатии на кнопку «Подтвердить»:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void MainForm\_FormClosed (object sender, RoutedEventArgs e) событие, возникающее после закрытия окна:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;

## 2.3. Методы

* void Connect() – функция, проверки соединения;
* void ReadingThread () – функция, чтения информации из COM-порта;
* bool ChoosePath (bool flag) – функция открытия диалога сохранения файла:
  + bool flag – указан ли путь заранее;
* byte[] PartPacking(byte[] InfByte, char Type, uint Length) – функция для формирования информационного кадра:
  + byte[] InfByte – массив информационных байтов;
  + char Type – тип кадра;
  + uint Length – длина информационного поля кадра;
* void Code(ref byte[] msg) – функция кодирования:
  + ref byte[] msg – массив байтов для кодирования;
* void Decode(ref byte[] msg) – функция декодирования:
  + ref byte[] msg – массив байтов для декодирования;
* byte[] ReadLocalFile(string sLocalFile) – функция чтения из указанного файла:
  + string sLocalFile – путь расположения файла;
* void FileDividing() – функция деления файла на кадры:
* int CheckSize(int size) – функция проверки файла на максимальный размер:
  + int size – размер файла в байтах;
* void GetMessage(bool flag) – функция показа сообщения о приеме файла:
  + bool flag – флаг установки режима (отображение сообщение);
* void NullVariablesHost() – функция обнуления состояния отправляющей стороны;
* void NullVariablesClient() – функция обнуления состояния принимающей стороны.

# 3. Класс SettingsForm:Form

- класс, определяющий окно настроек.

## 2.1. Переменные

* bool flag – флаг изменения настроек;

## 2.2. События

* void CancelButton\_Click(object sender, EventArgs e) – событие, возникающее при нажатии на кнопку отмены настроек:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;
* void OKButton\_Click(object sender, EventArgs e) – событие, возникающее при нажатии на кнопку подвтерждения изменения настроек:
  + object sender – объект, вызывающий событие;
  + EventArgs e – аргументы для события;

# 4*.* Листинг

**MainForm.cs**

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System.IO.Ports;

using System.Threading;

using System.Diagnostics;

using System.Timers;

namespace Kurs\_Project\_var25

{

public partial class MainForm : Form

{

#region Общие глобальные переменные

Thread ConnectionThread; //Поток на отслеживание соединения

Encoding ANSI = Encoding.Default; //С помощью этого задаем кодировку ANSI

bool BigFile = false; //Проверка на величину файла

static bool ErrorInfo = false; //Информация об ошибке

uint Frequency = 100; //Частота чтения входящего потока

byte FinalizationStatus = 1; //Текущий статус окончания передачи

bool ConnStatus = false; //Текущий статус порта

#endregion

#region Переменные, относящиеся к входному потоку (чтение)

string AppliedFileName; //Имя получаемого файла

Thread SynchronizationThread; //Поток на чтение

SynchronizationContext UIContext; //Вещь для синхронизации контролов в форме. Очень нужна для управления контролами из не-родных потоков

bool RHeader = false; //Получен заголовок

uint IndexOfInfopacketIn = 0; //Идентификатор для кадра (получение)

uint MaxIndexOfInfopacketIn = 0; //Количество кадров (получение)

#endregion

#region Переменные, относящиеся к выходному потоку (отправка)

string SendedFileName; //Имя отправляемого файла

bool SHeader = false; //Отправлен заголовок

byte[][] WriteData; //Отправляемые данные

byte[] byFileData = new byte[] { }; //Файл, заносимый в одномерный массив

uint IndexOfInfopacketOut = 0; //Идентификатор для кадра (отправка)

int CountPackets = 0; //Максимальный индекс текущего отправляемого файла

#endregion

public MainForm()

{

InitializeComponent();

GetMessageLabel.Visible = false;

ApplyButton.Visible = false;

DeclineButton.Visible = false;

if (Properties.Settings.Default.FirstLaunch == true)

{

MessageBox.Show("Здравствуйте! Похоже, Вы запускаете программу в первый раз.\n Для начала необходимо выбрать COM-порт, с которым программа будет работать.");

while (Properties.Settings.Default.FirstLaunch == true)

{

var f = new SettingsForm();

f.ShowDialog();

if (f.flag == true)

{

Frequency = Properties.Settings.Default.Frequency;

COMPort.BaudRate = Properties.Settings.Default.BaudRate;

COMPort.PortName = Properties.Settings.Default.ComName;

COMPort.ReadBufferSize = Properties.Settings.Default.InBuffer;

COMPort.WriteBufferSize = Properties.Settings.Default.OutBuffer;

COMPort.ReadTimeout = Properties.Settings.Default.ReadTimeout;

COMPort.WriteTimeout = Properties.Settings.Default.WriteTimeout;

Properties.Settings.Default.FirstLaunch = false;

Properties.Settings.Default.Save();

}

else

{

MessageBox.Show("Пожалуйста, настройте программу для первого запуска.");

}

}

}

COMPort.BaudRate = Properties.Settings.Default.BaudRate;

COMPort.PortName = Properties.Settings.Default.ComName;

COMPort.ReadBufferSize = Properties.Settings.Default.InBuffer;

COMPort.WriteBufferSize = Properties.Settings.Default.OutBuffer;

COMPort.ReadTimeout = Properties.Settings.Default.ReadTimeout;

COMPort.WriteTimeout = Properties.Settings.Default.WriteTimeout;

Frequency = Properties.Settings.Default.Frequency;

COMPort.DtrEnable = true;

COMPort.RtsEnable = false;

COMPort.Handshake = Handshake.None;

UIContext = SynchronizationContext.Current;

ConnectionThread = new Thread(Connect);

SynchronizationThread = new Thread(ReadingThread);

while (true)

{

try

{

COMPort.Open();

break;

}

catch

{

MessageBox.Show("К сожалению, не удалось загрузить нужный порт. Необходимо перенастроить программу.");

var f = new SettingsForm();

f.ShowDialog();

if (f.flag == true)

{

Frequency = Properties.Settings.Default.Frequency;

COMPort.BaudRate = Properties.Settings.Default.BaudRate;

COMPort.PortName = Properties.Settings.Default.ComName;

COMPort.ReadBufferSize = Properties.Settings.Default.InBuffer;

COMPort.WriteBufferSize = Properties.Settings.Default.OutBuffer;

COMPort.ReadTimeout = Properties.Settings.Default.ReadTimeout;

COMPort.WriteTimeout = Properties.Settings.Default.WriteTimeout;

Properties.Settings.Default.FirstLaunch = false;

Properties.Settings.Default.Save();

}

else

{

MessageBox.Show("Пожалуйста, перенастройте программу.");

}

}

}

ConnectionThread.Start();

SynchronizationThread.Start();

}

public byte[] ReadLocalFile(string sLocalFile)

{

using (FileStream oFS = new FileStream(sLocalFile, FileMode.Open, FileAccess.Read))

{

using (BinaryReader oBR = new BinaryReader(oFS))

{

if (oFS.Length < 2147483648)

return oBR.ReadBytes((int)oFS.Length);

else

{

MessageBox.Show("Выберите, пожалуйста, файл, размер которого не превышает 2 Гб.", "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

BigFile = true;

return new byte[] { 0 };

}

}

}

}

private void ChooseFileButton\_Click(object sender, EventArgs e)

{

ChoosePath(false);

}

private void AuthorsToolStripMenuItem\_Click(object sender, EventArgs e)

{

MessageBox.Show("Данный курсовой проект по дисциплине\n\"Сетевые технологии в АСОИУ\" сделали:\nЕгоров Алексей ИУ5-64\nМусин Тимур ИУ5-64\nВострокнутов Илья ИУ5-64", "Авторы", MessageBoxButtons.OK, MessageBoxIcon.None);

}

private void ExitToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (Exit() == true)

{

ConnectionThread.Abort();

SynchronizationThread.Abort();

this.Dispose();

Application.Exit();

}

}

private void ChooseFileToolStripMenuItem\_Click(object sender, EventArgs e)

{

ChoosePath(false);

}

/// <summary>

/// Выбор файла для пересылки/Выбор имени файла и диретории для сохранения

/// </summary>

/// <param name="flag">true - сохранение, false - пересылка</param>

/// <returns>Выполнилась функция или нет</returns>

private bool ChoosePath(bool flag)

{

if (flag == false)

{

if (SendOpenFileDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK) //Костыль для того, чтобы при нажатии "Отмена" путь к папке не "исчезал"

{

if (SendOpenFileDialog.FileName != "")

{

Array.Resize(ref byFileData, byFileData.Length);

byFileData = ReadLocalFile(SendOpenFileDialog.FileName);

FileDividing();

}

if (BigFile == true)

return false;

SendPathTextBox.Text = SendOpenFileDialog.FileName;

SendedFileName = SendOpenFileDialog.SafeFileName;

return true;

}

return false;

}

if (flag == true)

{

AcceptedSaveFileDialog.FileName = AppliedFileName;

if (AcceptedSaveFileDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

AppliedFileName = AcceptedSaveFileDialog.FileName;

return true;

}

return false;

}

return false;

}

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (Exit() == true)

{

ConnectionThread.Abort();

SynchronizationThread.Abort();

this.Dispose();

Application.Exit();

}

else e.Cancel = true;

}

private bool Exit()

{

DialogResult Exit = MessageBox.Show("Вы действительно хотите выйти?", "Подтверждение выхода", MessageBoxButtons.YesNo);

if (Exit == DialogResult.Yes) return true;

else return false;

}

private void ConnSettingsToolStripMenuItem\_Click(object sender, EventArgs e)

{

var f = new SettingsForm();

f.ShowDialog();

if (f.flag == true)

{

DialogResult Reload = MessageBox.Show("Перезагрузить соединение для применения новых параметров?", "Подтверждение перезагрузки", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (Reload == DialogResult.Yes)

{

SynchronizationThread.Abort();

ConnectionThread.Abort();

COMPort.Close();

COMPort.BaudRate = Properties.Settings.Default.BaudRate;

COMPort.PortName = Properties.Settings.Default.ComName;

COMPort.ReadBufferSize = Properties.Settings.Default.InBuffer;

COMPort.WriteBufferSize = Properties.Settings.Default.OutBuffer;

COMPort.ReadTimeout = Properties.Settings.Default.ReadTimeout;

COMPort.WriteTimeout = Properties.Settings.Default.WriteTimeout;

COMPort.Open();

SynchronizationThread = new Thread(ReadingThread);

ConnectionThread = new Thread(Connect);

SynchronizationThread.Start();

ConnectionThread.Start();

}

}

}

//итог: первый байт - количество отбрасываемых бит при переводе в двоичную систему

// второй - количество отбрасываемых бит в декодированном битовом массиве

static void Code(ref byte[] msg)

{

System.Collections.BitArray bitMsg = new System.Collections.BitArray(msg);

int numParts = bitMsg.Length / 11;

if (bitMsg.Length % 11 != 0) numParts++;

byte secondCheck = Convert.ToByte(11 - bitMsg.Length % 11);

System.Collections.BitArray codeBitMsg = new System.Collections.BitArray(numParts \* 15);

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

{

System.Collections.BitArray \_15bitCodeArr = new System.Collections.BitArray(15);

for (int j = 10; j >= 0; --j)

{

bool temp = (bitMsg.Length > i \* 11 + j) ? bitMsg.Get(i \* 11 + j) : false;

if (j >= 4)

{

\_15bitCodeArr.Set(j + 4, temp);

}

else if (j >= 1)

{

\_15bitCodeArr.Set(j + 3, temp);

}

else

{

\_15bitCodeArr.Set(2, temp);

}

}

\_15bitCodeArr[7] = \_15bitCodeArr.Get(14) ^ \_15bitCodeArr.Get(13) ^ \_15bitCodeArr.Get(12) ^ \_15bitCodeArr.Get(11) ^

\_15bitCodeArr.Get(10) ^ \_15bitCodeArr.Get(9) ^ \_15bitCodeArr.Get(8);

\_15bitCodeArr[3] = \_15bitCodeArr.Get(14) ^ \_15bitCodeArr.Get(13) ^ \_15bitCodeArr.Get(12) ^ \_15bitCodeArr.Get(11) ^

\_15bitCodeArr.Get(6) ^ \_15bitCodeArr.Get(5) ^ \_15bitCodeArr.Get(4);

\_15bitCodeArr[1] = \_15bitCodeArr.Get(14) ^ \_15bitCodeArr.Get(13) ^ \_15bitCodeArr.Get(10) ^ \_15bitCodeArr.Get(9) ^

\_15bitCodeArr.Get(6) ^ \_15bitCodeArr.Get(5) ^ \_15bitCodeArr.Get(2);

\_15bitCodeArr[0] = \_15bitCodeArr.Get(14) ^ \_15bitCodeArr.Get(12) ^ \_15bitCodeArr.Get(10) ^ \_15bitCodeArr.Get(8) ^

\_15bitCodeArr.Get(6) ^ \_15bitCodeArr.Get(4) ^ \_15bitCodeArr.Get(2);

for (int j = 0; j < 15; ++j)

{

codeBitMsg.Set(i \* 15 + j, \_15bitCodeArr.Get(j));

}

}

byte firstCheck = Convert.ToByte(8 - codeBitMsg.Length % 8);

int newSize = codeBitMsg.Length / 8 + ((codeBitMsg.Length % 8 == 0) ? 2 : 3);

msg = new byte[newSize];

msg[0] = firstCheck;

msg[1] = secondCheck;

codeBitMsg.CopyTo(msg, 2);

return;

}

static void Decode(ref byte[] msg)

{

byte firstCheck = msg[0];

byte secondCheck = msg[1];

int start = 16;

System.Collections.BitArray bitMsg = new System.Collections.BitArray(msg);

int length = bitMsg.Length - 16 - firstCheck;

int numParts = length / 15;

int errIndx = -1;

System.Collections.BitArray decodeBitMsg = new System.Collections.BitArray(numParts \* 11 - secondCheck);

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

{

int current = start + i \* 15;

bool c1;

bool c2;

bool c3;

bool c4;

c4 = bitMsg.Get(current + 14) ^ bitMsg.Get(current + 13) ^ bitMsg.Get(current + 12) ^ bitMsg.Get(current + 11) ^

bitMsg.Get(current + 10) ^ bitMsg.Get(current + 9) ^ bitMsg.Get(current + 8) ^ bitMsg.Get(current + 7);

c3 = bitMsg.Get(current + 14) ^ bitMsg.Get(current + 13) ^ bitMsg.Get(current + 12) ^ bitMsg.Get(current + 11) ^

bitMsg.Get(current + 6) ^ bitMsg.Get(current + 5) ^ bitMsg.Get(current + 4) ^ bitMsg.Get(current + 3);

c2 = bitMsg.Get(current + 14) ^ bitMsg.Get(current + 13) ^ bitMsg.Get(current + 10) ^ bitMsg.Get(current + 9) ^

bitMsg.Get(current + 6) ^ bitMsg.Get(current + 5) ^ bitMsg.Get(current + 2) ^ bitMsg.Get(current + 1);

c1 = bitMsg.Get(current + 14) ^ bitMsg.Get(current + 12) ^ bitMsg.Get(current + 10) ^ bitMsg.Get(current + 8) ^

bitMsg.Get(current + 6) ^ bitMsg.Get(current + 4) ^ bitMsg.Get(current + 2) ^ bitMsg.Get(current);

if (c1 || c2 || c3 || c4)

{

ErrorInfo = true;

errIndx = ((c1) ? 1 : 0) + ((c2) ? 2 : 0) + ((c3) ? 4 : 0) + ((c4) ? 8 : 0);

bitMsg.Set(current + errIndx, bitMsg.Get(current + errIndx));

}

System.Collections.BitArray \_11bitDecodeArr = new System.Collections.BitArray(11);

\_11bitDecodeArr[0] = bitMsg.Get(current + 2);

\_11bitDecodeArr[1] = bitMsg.Get(current + 4);

\_11bitDecodeArr[2] = bitMsg.Get(current + 5);

\_11bitDecodeArr[3] = bitMsg.Get(current + 6);

\_11bitDecodeArr[4] = bitMsg.Get(current + 8);

\_11bitDecodeArr[5] = bitMsg.Get(current + 9);

\_11bitDecodeArr[6] = bitMsg.Get(current + 10);

\_11bitDecodeArr[7] = bitMsg.Get(current + 11);

\_11bitDecodeArr[8] = bitMsg.Get(current + 12);

\_11bitDecodeArr[9] = bitMsg.Get(current + 13);

\_11bitDecodeArr[10] = bitMsg.Get(current + 14);

for (int j = 0; j < 11; ++j)

{

if (i \* 11 + j < decodeBitMsg.Length)

decodeBitMsg.Set(i \* 11 + j, \_11bitDecodeArr.Get(j));

}

}

int newSize = decodeBitMsg.Length / 8;

msg = new byte[newSize];

decodeBitMsg.CopyTo(msg, 0);

return;

}

private void SendFileButton\_Click(object sender, EventArgs e)

{

if (SendedFileName != null)

{

PartPacking(new byte[] { }, 'H', (uint)SendedFileName.Length);

}

else

MessageBox.Show("Пожалуйста, выберите файл.");

}

private void DeclineButton\_Click(object sender, EventArgs e)

{

GetMessage(false);

if (Properties.Settings.Default.SequentialMode == true)

SequentialHide(false);

//Отправить на другой комп сигнал о том, что принятие файла отклонено

PartPacking(new byte[] { }, 'N', 0);

}

/// <summary>

/// Скрытие контролов, отвечающих за принятие решения по поводу получаемых файлов

/// </summary>

/// <param name="flag">false - скрываем, true - нутыпонял</param>

private void GetMessage(bool flag)

{

if (flag == false)

{

UIContext.Send(io => GetMessageLabel.Visible = false, null);

UIContext.Send(io => ApplyButton.Visible = false, null);

UIContext.Send(io => DeclineButton.Visible = false, null);

}

else

{

UIContext.Send(io => GetMessageLabel.Visible = true, null);

UIContext.Send(io => ApplyButton.Visible = true, null);

UIContext.Send(io => DeclineButton.Visible = true, null);

UIContext.Send(io => GetMessageLabel.Text = "Получен запрос на принятие нового файла. Принять файл?\n\nИмя файла: " + AppliedFileName, null);

}

}

private void ApplyButton\_Click(object sender, EventArgs e)

{

ChoosePath(true);

GetMessage(false);

GetLabel.Text = "Получаемый файл: " + AppliedFileName;

//Отправить сигнал о том, что разрешено отсылать файл

PartPacking(new byte[] { }, 'Y', 0);

}

private void SequentialHide(bool flag)

{

if (flag == true)

UIContext.Send(g => SendGroupBox.Visible = false, null);

else

UIContext.Send(g => SendGroupBox.Visible = true, null);

}

/// <summary>

/// Функция, принимаемая потоком.

/// Служит для соединения двух компьютеров и обозначения текущего статуса соединения

/// </summary>

private void Connect()

{

while (true)

{

Thread.Sleep(2000);

try

{

if (COMPort.IsOpen == true)

{

if (COMPort.DsrHolding == true)

{

ConnStatus = true;

UIContext.Send(d => ConnectionStatusTSSL.Text = "Соединение: активно", null);

}

if (COMPort.DsrHolding == false)

{

ConnStatus = false;

UIContext.Send(d => ConnectionStatusTSSL.Text = "Соединение: отсутствует", null);

}

if (((RHeader == true) || (SHeader == true)) & (ConnStatus == false))

{

UIContext.Send(d => ConnectionStatusTSSL.Text = "Соединение: прервано", null);

RHeader = false;

SHeader = false;

}

}

}

catch (TimeoutException)

{

UIContext.Send(d => ConnectionStatusTSSL.Text = "Соединение: превышено ожидание", null);

}

}

}

private void MainForm\_FormClosed(object sender, FormClosedEventArgs e)

{

ConnStatus = false;

SynchronizationThread.Abort();

ConnectionThread.Abort();

COMPort.Close();

}

private void IntToByte(uint Number, ref int index, byte[] VByte)

{

byte[] helparr = BitConverter.GetBytes(Number); //Запись во вспомогательный массив длины массива (побайтово)

for (int i = 0; i < 4; i++) //Непосредственная запись в основной массив

{

VByte[index] = helparr[i];

index++;

}

}

private uint ByteToInt(byte[] VByte, int index)

{

byte[] qq = new byte[4]; //Вспомогательный массив для выписывания длины инфочасти

for (int i = 0; i < 4; i++)

qq[i] = VByte[index + i];

uint low = 0; //Переменная, в которой будет храниться значение длины инфочасти

low = BitConverter.ToUInt32(qq, 0);

return low;

}

static int CheckSize(int size)

{

return (int)Math.Floor(Math.Floor(((double)size - 17) \* 8.0 / 15.0) \* 11.0 / 8.0);

}

private void FileDividing()

{

int i = CheckSize(Properties.Settings.Default.OutBuffer);

CountPackets = byFileData.Length / i;

bool lastpack = false;

if (byFileData.Length % i != 0)

{

CountPackets++;

lastpack = true;

}

Array.Resize(ref WriteData, CountPackets);

int bytesIndex = 0;

if (i == 0)

{

Code(ref byFileData);

WriteData[0] = byFileData;

return;

}

else

{

for (int IndexOfMas = 0; IndexOfMas < CountPackets; IndexOfMas++)

{

if (lastpack == true && IndexOfMas == CountPackets - 1)

{

int lastLength = byFileData.Length - bytesIndex;

Array.Resize(ref WriteData[IndexOfMas], lastLength);

}

else

Array.Resize(ref WriteData[IndexOfMas], i);

for (int j = 0; j < i && (bytesIndex < byFileData.Length); j++, bytesIndex++)

WriteData[IndexOfMas][j] = byFileData[bytesIndex];

Code(ref WriteData[IndexOfMas]);

}

}

}

private void NullVariablesHost()

{

UIContext.Send(d => SendProgressBar.Value = 0, 0);

SendedFileName = null;

SHeader = false;

WriteData = new byte[][] { };

IndexOfInfopacketOut = 0;

FinalizationStatus = 1;

byFileData = new byte[] { };

CountPackets = 0;

}

private void NullVariablesClient()

{

UIContext.Send(d => GetProgressBar.Value = 0, 0);

AppliedFileName = null;

RHeader = false;

IndexOfInfopacketIn = 0;

MaxIndexOfInfopacketIn = 0;

}

/// <summary>

/// Функция для упаковки информации любого вида в сообщение

/// </summary>

/// <param name="InfByte">Массив данных</param>

/// <param name="Type">Тип передаваемых данных</param>

/// <param name="Length">Длина передаваемых данных</param>

/// <returns>Готовый к отправке кадр с данными</returns>

private byte[] PartPacking(byte[] InfByte, char Type, uint Length)

{

byte[] VByte = new byte[] { };

int index = 0;

switch (Type)

{

#region I

case 'I':

VByte = new byte[Length + 15];

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Старт-байт

index++;

VByte[index] = Convert.ToByte(Type); //Тип кадра

index++;

IntToByte(Length, ref index, VByte);

IntToByte(IndexOfInfopacketOut, ref index, VByte);

IntToByte((uint)CountPackets, ref index, VByte);

for (int j = 0; j < Length; index++, j++) //Запись в массив инфочасти

VByte[index] = InfByte[j];

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Стоп-байт

COMPort.Write(VByte, 0, VByte.Length); //Запись на порт

break;

#endregion

#region A

case 'A':

VByte = new byte[7];

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Старт-байт

index++;

VByte[index] = Convert.ToByte(Type); //Тип кадра

index++;

IntToByte(IndexOfInfopacketIn, ref index, VByte);

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Стоп-байт

COMPort.Write(VByte, 0, VByte.Length); //Запись на порт

break;

#endregion

#region H

case 'H':

VByte = new byte[Length + 3];

char[] FName = SendedFileName.ToCharArray();

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Старт-байт

index++;

VByte[index] = Convert.ToByte(Type); //Тип кадра

index++;

byte[] lol = ANSI.GetBytes(FName);

foreach (byte ch in lol)

{

VByte[index] = ch;

index++;

}

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Стоп-байт

COMPort.Write(VByte, 0, VByte.Length); //Запись на порт

if (SHeader == true)

PartPacking(new byte[] { }, 'A', 0);

break;

#endregion

#region F

case 'F':

VByte = new byte[4];

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Старт-байт

index++;

VByte[index] = Convert.ToByte(Type); //Тип кадра

index++;

if (FinalizationStatus == 1)

{

VByte[index] = FinalizationStatus;

index++;

}

else if (FinalizationStatus == 2)

{

VByte[index] = FinalizationStatus;

index++;

NullVariablesClient();

}

else

{

VByte[index] = FinalizationStatus;

index++;

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Стоп-байт

COMPort.Write(VByte, 0, VByte.Length);

NullVariablesHost();

COMPort.RtsEnable = true;

FinalizationStatus = 1;

UIContext.Send(d => GetLabel.Text = "", 0);

MessageBox.Show("Передача завершена");

break;

}

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Стоп-байт

COMPort.Write(VByte, 0, VByte.Length); //Запись на порт

break;

#endregion

#region Y

case 'Y':

VByte = new byte[3];

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Старт-байт

index++;

VByte[index] = Convert.ToByte(Type); //Тип кадра

index++;

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Стоп-байт

COMPort.Write(VByte, 0, VByte.Length); //Запись на порт

break;

#endregion

#region N

case 'N':

VByte = new byte[3];

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Старт-байт

index++;

VByte[index] = Convert.ToByte(Type); //Тип кадра

index++;

VByte[index] = Byte.Parse("FF", System.Globalization.NumberStyles.AllowHexSpecifier); //Стоп-байт

COMPort.Write(VByte, 0, VByte.Length); //Запись на порт

break;

#endregion

default:

break;

}

return VByte;

}

/// <summary>

/// Функция, отвечающая за чтение и интерпретацию входных данных

/// </summary>

public void ReadingThread()

{

while (true)

{

Thread.Sleep((int)Frequency);

#region Чтение данных из потока

byte[] InfoBuffer = new byte[] { };

try

{

Array.Resize(ref InfoBuffer, COMPort.BytesToRead);

COMPort.Read(InfoBuffer, 0, COMPort.BytesToRead);

}

catch

{

Array.Resize(ref InfoBuffer, COMPort.BytesToRead);

COMPort.Read(InfoBuffer, 0, COMPort.BytesToRead);

}

#endregion

if (InfoBuffer.Length != 0)

{

#region Декодирование

byte[] HelpBuffer = new byte[] { };

if (InfoBuffer.Count() != 0)

{

Array.Resize(ref HelpBuffer, InfoBuffer.Count() - 2);

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

HelpBuffer[i - 1] = InfoBuffer[i];

}

#endregion

#region Основная часть

char TypeOfPacket = Convert.ToChar(HelpBuffer[0]);

switch (TypeOfPacket)

{

#region Информационные кадры

case 'I':

{

MaxIndexOfInfopacketIn = ByteToInt(HelpBuffer, 1);

byte[] ret = new byte[HelpBuffer.Length - 13];

for (int hf = 13; hf < HelpBuffer.Length; hf++)

ret[hf - 13] = HelpBuffer[hf];

Decode(ref ret);

if (ErrorInfo == false)

{

uint MAX = ByteToInt(HelpBuffer, 9);

IndexOfInfopacketIn++;

UIContext.Send(d => GetProgressBar.Value = (int)((IndexOfInfopacketIn / (double)MAX) \* 100), 0);

char[] dof = ANSI.GetChars(ret);

string df = new string(dof);

File.AppendAllText(AppliedFileName, df, ANSI);

}

else

ErrorInfo = false;

if (SHeader == false)

{

PartPacking(new byte[] { }, 'A', 0);

}

else

{

IndexOfInfopacketOut++;

UIContext.Send(d => SendProgressBar.Value = (int)((IndexOfInfopacketOut / (double)CountPackets) \* 100), 0);

if (IndexOfInfopacketOut != CountPackets)

{

PartPacking(WriteData[IndexOfInfopacketOut], 'I', (uint)WriteData[IndexOfInfopacketOut].Length);

}

else

{

NullVariablesHost();

PartPacking(new byte[] { }, 'A', 0);

}

}

}

break;

#endregion

#region ACK-кадры: отвечают за подтверждение принятия кадра или за запрос на повторную передачу

case 'A':

{

IndexOfInfopacketOut = ByteToInt(HelpBuffer, 1);

UIContext.Send(d => SendProgressBar.Value = (int)((IndexOfInfopacketOut / (double)CountPackets) \* 100), 0);

if (IndexOfInfopacketOut != CountPackets)

{

if (MaxIndexOfInfopacketIn != IndexOfInfopacketIn)

{

UIContext.Send(d => GetProgressBar.Value = 0, 0);

IndexOfInfopacketIn = 0;

PartPacking(WriteData[IndexOfInfopacketOut], 'I', (uint)WriteData[IndexOfInfopacketOut].Length);

}

else

PartPacking(WriteData[IndexOfInfopacketOut], 'I', (uint)WriteData[IndexOfInfopacketOut].Length);

}

else

PartPacking(new byte[] { }, 'F', 0);

}

break;

#endregion

#region HEAD-кадры: передают инфо о названии файла

case 'H':

{

RHeader = true;

char[] met = new char[HelpBuffer.Length - 1];

for (int hf = 1; hf < HelpBuffer.Length; hf++)

met[hf - 1] = Convert.ToChar(HelpBuffer[hf]);

AppliedFileName = new string(ANSI.GetChars(HelpBuffer, 1, HelpBuffer.Length - ((SHeader == true) ? 8 : 1)));

GetMessage(true);

if (Properties.Settings.Default.SequentialMode == true)

SequentialHide(true);

}

break;

#endregion

#region FIN-кадры: окончание передачи, закрытие соединения

case 'F':

{

if (HelpBuffer[1] == 1)

{

FinalizationStatus = 2;

PartPacking(new byte[] { }, 'F', 0);

}

else if (HelpBuffer[1] == 2)

{

FinalizationStatus = 3;

PartPacking(new byte[] { }, 'F', 0);

}

else

{

NullVariablesClient();

COMPort.RtsEnable = true;

UIContext.Send(d => GetLabel.Text = "", 0);

MessageBox.Show("Передача завершена");

}

}

break;

#endregion

#region YES-кадры: положительный ответ на запрос о передаче файла

case 'Y':

{

SHeader = true;

PartPacking(WriteData[0], 'I', (uint)WriteData[0].Length);

}

break;

#endregion

#region NO-кадры: отрицательный ответ на запрос о передаче файла

case 'N':

{

SHeader = false;

byFileData = new byte[] { };

}

break;

#endregion

default:

break;

}

#endregion

}

}

}

}

}

**SettingsForm.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO.Ports;

namespace Kurs\_Project\_var25

{

public partial class SettingsForm : Form

{

public bool flag = false;

public SettingsForm()

{

InitializeComponent();

BaudRateComboBox.SelectedIndex = Properties.Settings.Default.BaudRateIndex;

foreach (string s in SerialPort.GetPortNames())

ComPortComboBox.Items.Add(s);

ComPortComboBox.SelectedIndex = Properties.Settings.Default.ComNameIndex;

InBufferNumericUpDown.Value = Properties.Settings.Default.InBuffer;

OutBufferNumericUpDown.Value = Properties.Settings.Default.OutBuffer;

ReadNumericUpDown.Value = Properties.Settings.Default.ReadTimeout;

WriteNumericUpDown.Value = Properties.Settings.Default.WriteTimeout;

FrequencyNUD.Value = Properties.Settings.Default.Frequency;

if (Properties.Settings.Default.SequentialMode == false)

{

ParallelRB.Checked = true;

}

else

{

SequentialRB.Checked = true;

}

}

private void CancelButton\_Click(object sender, EventArgs e)

{

this.Close();

}

private void OKButton\_Click(object sender, EventArgs e)

{

//Проверка сохранения параметров программы (для сохранения настроек)

Properties.Settings.Default.BaudRate = int.Parse(BaudRateComboBox.Text);

Properties.Settings.Default.BaudRateIndex = BaudRateComboBox.SelectedIndex;

Properties.Settings.Default.ComNameIndex = ComPortComboBox.SelectedIndex;

Properties.Settings.Default.ComName = (string)ComPortComboBox.SelectedItem;

Properties.Settings.Default.InBuffer = (int)InBufferNumericUpDown.Value;

Properties.Settings.Default.OutBuffer = (int)OutBufferNumericUpDown.Value;

Properties.Settings.Default.ReadTimeout = (int)ReadNumericUpDown.Value;

Properties.Settings.Default.WriteTimeout = (int)WriteNumericUpDown.Value;

Properties.Settings.Default.Frequency = (uint)FrequencyNUD.Value;

if (ParallelRB.Checked == true)

Properties.Settings.Default.SequentialMode = false;

else

Properties.Settings.Default.SequentialMode = true;

Properties.Settings.Default.Save();

flag = true;

this.Close();

}

}

}