**Московский государственный технический университет имени Н. Э. Баумана.**

**Факультет “Информатика и системы управления”.**

**Кафедра “Системы обработки информации и управления”.**

|  |  |
| --- | --- |
| Утверждаю: |  |
| Галкин В.А. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

**Курсовая работа**

**по курсу**

**Сетевые технологии в АСОИУ**

**«Программа пересылки писем»**

Описание программы

(вид документа)

бумага А4

(вид носителя)

(количество листов)

48

Вариант 17

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛИ: |  |
| студенты группы ИУ5-64 |  |
| Меркулова Н.А. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Зубков А.Д. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Турусов В.И. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |

**Москва - 2020**

Оглавление

[1. Введение 3](#_Toc40045142)

[2. Класс Form1 3](#_Toc40045143)

[2.1. Переменные 3](#_Toc40045144)

[2.2. События 4](#_Toc40045145)

[2.3. Методы 5](#_Toc40045146)

[3. Класс Form2 5](#_Toc40045147)

[3.1. Переменные 5](#_Toc40045148)

[3.2. События 5](#_Toc40045149)

[3.3. Методы 6](#_Toc40045150)

[4. Класс Form3 6](#_Toc40045151)

[4.1. Переменные 6](#_Toc40045152)

[4.2. События 6](#_Toc40045153)

[4.3. Методы 6](#_Toc40045154)

[5*.* Листинг 7](#_Toc40045155)

[5.1. Form1 7](#_Toc40045156)

[5.2. Form2 42](#_Toc40045157)

[5.3. Form3 46](#_Toc40045158)

# Введение

Программный продукт написан с использованием технологии Microsoft 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.

# Класс Form1

Класс, определяющий главное окно программы.

## 2.1. Переменные

* public Dictionary<string, string> Auth\_status – словарь, который хранит флаг получения логина
* public Dictionary<string, string> – словарь для хранения логинов пользователей
* public bool DEBUG\_MODE – флаг отключения режима отладки
* public bool RET\_ALLOWED – флаг использования ret в ARQ
* public enum Connection\_Status – возможные состояния соединений
* public Mutex Auth\_status\_mutex – семафор для доступа к Auth\_status
* public Connection\_Status Phys\_status1 – статус абонента 1 на линии
* public Connection\_Status Phys\_status2 - статус абонента 2 на линии
* public Mutex Phys\_status1\_mutex - семафор
* public Mutex Phys\_status1\_mutex – семафор
* One\_Task LastFrameSenttoPort1 – последний не доставленный кадр в порт 1
* One\_Task LastFrameSenttoPort2 - последний не доставленный кадр в порт 2
* public Mutex LastFrame\_ToSend\_mutex – семафор
* public Int32 Ack1\_awaited\_Auth - флаг ожидания ack1
* public Int32 Ack2\_awaited\_Auth – флаг ожидания ack2
* public Mutex Ack1\_mutex\_Auth – семафор
* public Mutex Ack2\_mutex\_Auth – семафор
* string SelectedPort1Name – хранит имя COM-порта 1
* Mutex SelectedPort1Name\_mutex – семафор
* string SelectedPort2Name – хранит имя COM-порта 2
* Mutex SelectedPort2Name\_mutex – семафор
* string SelectedBaudrate – хранит скорость работы COM-портов
* Mutex SelectedBaudrate\_mutex – семафор
* public Mutex ReceivedFrames\_mutex1 – семафор
* public Mutex ReceivedFrames\_mutex2
* public List<byte> ReceivedFrames1 – список1 принятых байт
* public List<byte> ReceivedFrames2 – список2 принятых байт
* public struct One\_Task – хранит кадр с портом назначения
* public List<One\_Task> TasksReceived – список принятых кадров
* public Mutex TaskReceived\_mutex - семафор
* public List<One\_Task> TasksToSend – список кадров для отправки
* public Mutex TaskToSend\_mutex – семафор
* public Mutex AuthData\_mutex - семафор
* class inbox\_class – для хранения принятого письма
* class outbox\_class - для хранения отправленного письма
* public Mutex Inbox\_update\_mutex - семафор
* public Mutex Outbox\_update\_mutex - семафор
* public bool Inbox\_update\_needed - флаг для обновления папки входящие
* public bool Outbox\_update\_needed - флаг для обновления папки исходящие

## 2.2. События

* private void Form1\_FormClosing(object sender, FormClosingEventArgs e)- событие возникающее при закрытии формы:
  + object sender – объект, вызывающий событие;
  + FormClosingEventArgs *e* – аргументы для события;
* private void button3\_Click(object sender, EventArgs e) – событие, возникающее при нажатии кнопки авторизации:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void toolStripComboBox1\_SelectedIndexChanged(object sender, EventArgs e) – событие, возникающее при выборе имени порта1:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void toolStripComboBox3\_SelectedIndexChanged(object sender, EventArgs e) – событие, возникающее при выборе скорости:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void button4\_Click(object sender, EventArgs e) - событие возникающее при нажатии кнопки закрыть порты:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void toolStripComboBox2\_SelectedIndexChanged(object sender, EventArgs e) – событие, возникающее при выборе имени порта2:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void toolStripMenuItem2\_Click(object sender, EventArgs e) – событие, возникающее при выборе пункта меню “справка”:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void button1\_Click(object sender, EventArgs e) – событие, возникающее при нажатии кнопки открытия портов:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;

## 2.3. Методы

* public void FrameReceivedInformation(DefaultFrame local\_frame)
* public void FrameReceivedOpenLetter(DefaultFrame frame) - обработчик кадра OPENLETTER
* public void FrameReceivedRet(DefaultFrame frame) - обработчик кадра RET
* public void FrameReceivedAck(DefaultFrame ReceivedFrameStruct) - обработчик кадра ACK
* public void FrameReceivedLogin(DefaultFrame ReceivedFrame) - обработчик кадра LOGIN
* public void FrameReceivedLogout(DefaultFrame frame)- обработчик кадра LOGOUT
* public void TaskHandler() – обработчик полученных кадров
* public void TaskToSendHandler() – метод для отправки кадров
* public void Establish\_Logical() – метод установления логического соединения
* public void Wait\_for\_info\_ack1() – отсылка письма в порт1 с ожиданием подтверждения
* public void Wait\_for\_info\_ack2()– отсылка письма в порт2 с ожиданием подтверждения
* public void settextbox1(string text)- запись строки в textbox1
* public void setport1state(string text) - запись строки в port1state\_label
* public void OpenSerial1() – открытие порта 1
* public void OpenSerial2()– открытие порта 2
* public void serial1\_monitor() - отслеживает физическое состояние портов
* public void serial2\_monitor()
* public void Set\_AuthDisconnectButton(bool state) – устанавливает режим кнопки AuthDisconnectButton
* public void Set\_AuthСonnectButton(bool state) – устанавливает режим кнопки AuthСonnectButton
* public void Auth\_Tracker()- метод, следящий за Авторизацией

# 3. Класс Form2

Класс, определяющий окно для отображения принятых писем.

## 3.1. Переменные

* public bool UpdateThread\_to\_close – флаг о необходимости закрытия формы
* public bool Inbox\_update\_needed – флаг о необходимости обновления таблицы писем
* private delegate void UpdateDataGridDeleg(List<inbox> list) - делегат для внесения изменений в таблицу писем
* private delegate void SetTextDeleg(string text) - делегат для обновления элементов GUI из потока LookForInboxUpdatesthr
* public Form1 form1 – объект главной формы

## 3.2. События

* private void Form2\_FormClosing(object sender, FormClosingEventArgs e) – событие, возникающее при закрытии формы:
  + object sender – объект, вызывающий событие;
  + FormClosingEventArgs *e* – аргументы для события;
* private void Form2\_Load(object sender, EventArgs e) – событие, возникающее при открытии формы:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e) – событие, возникающее при открытии письма:
  + object sender – объект, вызывающий событие;
  + DataGridViewCellEventArgs *e* – аргументы для события;

## 3.3. Методы

* public void LookForInboxUpdates() – для запуска обновления данных таблицы
* public void UpdateDataGrid1(List<inbox> list) - обновление содержимого таблицы отправленных писем
* public void Show\_Inbox() - отображает таблицу отправленных писем

# Класс Form3

Класс, определяющий окно для отображения принятых писем.

## 4.1. Переменные

* public bool UpdateThread\_to\_close – флаг о необходимости закрытия формы
* public bool Inbox\_update\_needed – флаг о необходимости обновления таблицы писем
* private delegate void UpdateDataGridDeleg(List<inbox> list) - делегат для внесения изменений в таблицу писем
* private delegate void SetTextDeleg(string text) - делегат для обновления элементов GUI из потока LookForOutboxUpdatesthr
* public Form1 form1 – обьект главной формы

## 4.2. События

* private void Form3\_FormClosing\_1(object sender, FormClosingEventArgs e) – событие, возникающее при закрытии формы:
  + object sender – объект, вызывающий событие;
  + FormClosingEventArgs *e* – аргументы для события;
* private void Form3\_Load\_1(object sender, EventArgs e) – событие, возникающее при открытии формы:
  + object sender – объект, вызывающий событие;
  + EventArgs *e* – аргументы для события;
* private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e) – событие, возникающее при открытии письма:
  + object sender – объект, вызывающий событие;
  + DataGridViewCellEventArgs *e* – аргументы для события;

## 4.3. Методы

* public void LookForOutboxUpdates() – для запуска обновления данных таблицы
* public void UpdateDataGrid1(List<inbox> list)- обновление содержимого таблицы отправленных писем
* public void Show\_Outbox() – отображает таблицу отправленных писем

# 5*.* Листинг

## 5.1. Form1

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System.IO.Ports;

using System.Threading;

using System.Data.SQLite;

using System.Data;

using System.Linq;

using System.Data.Common;

using Newtonsoft.Json;

namespace main\_application

{

public partial class Form1 : Form

{

public bool DEBUG\_MODE = false;

public bool RET\_ALLOWED = false;

// Использется для передачти байтов по компорту

public static Encoding WIN1251 = Encoding.GetEncoding("windows-1251");

public static Encoding ASCII = Encoding.ASCII;

public Dictionary<string, string> Auth\_status = new Dictionary<string, string> {

{ "ACK1", "undef" },

{ "ACK2", "undef" },

{ "ACK\_local", "undef" }

};

public Mutex Auth\_status\_mutex = new Mutex();

// Перечисление состояний соединения

public enum Connection\_Status

{

CONNECTION\_WAIT,

CONNECTED,

DISCONNECTION\_WAIT,

DISCONNECTED,

};

//Флаги наличия абонентов на линии

public Connection\_Status Phys\_status1 = Connection\_Status.DISCONNECTED;

public Connection\_Status Phys\_status2 = Connection\_Status.DISCONNECTED;

public Mutex Phys\_status1\_mutex = new Mutex();

public Mutex Phys\_status2\_mutex = new Mutex();

// Здесь отмечаются только сообщения информационного кадра (пока так)

One\_Task LastFrameSenttoPort1 = new One\_Task(null, null);

One\_Task LastFrameSenttoPort2 = new One\_Task(null, null);

public Mutex LastFrame\_ToSend\_mutex = new Mutex();

public Int32 Ack1\_awaited\_Auth = 0;

public Int32 Ack2\_awaited\_Auth = 0;

public Mutex Ack1\_mutex\_Auth = new Mutex();

public Mutex Ack2\_mutex\_Auth = new Mutex();

//Парамаетры компортов

string SelectedPort1Name;

Mutex SelectedPort1Name\_mutex = new Mutex();

string SelectedPort2Name;

Mutex SelectedPort2Name\_mutex = new Mutex();

string SelectedBaudrate;

Mutex SelectedBaudrate\_mutex = new Mutex();

//Мьютекс для согласования: чтения, записи, и изменения в списке ReceivedFrames

public Mutex ReceivedFrames\_mutex1 = new Mutex();

//Мьютекс для согласования: чтения, записи, и изменения в списке ReceivedFrames2

public Mutex ReceivedFrames\_mutex2 = new Mutex();

//Списки принятых данных для обмена между потоками Serial1\_receiving1() и FindFrame1()

public List<byte> ReceivedFrames1 = new List<byte>();

//Списки принятых данных для обмена между потоками Serial1\_receiving2() и FindFrame2()

public List<byte> ReceivedFrames2 = new List<byte>();

// Структура для хранения задания(кадра)

public struct One\_Task

{

public string PortNum;

public byte[] Frame;

public One\_Task(string name, byte[] frame)

{

PortNum = name;

Frame = frame;

}

}

// Список заданий используется четырмя потоками- По два на каждый порт

// Формат записи заданий One\_Task("Номер порта", Байты[] пришедшие в порт)

//Список заданий, новое задание помещается в конец списка, выполненное удаляется из начала списка

// Список заданий содержит задания из первого и второго порта

public List<One\_Task> TasksReceived = new List<One\_Task>();

//Мьютекс для согласования: чтения, записи, и изменения заданий

// Использется для первого и второго порта одновременно

public Mutex TaskReceived\_mutex = new Mutex();

public List<One\_Task> TasksToSend = new List<One\_Task>();

//Мьютекс для согласования: чтения, записи, и изменения заданий

// Использется для первого и второго порта одновременно

public Mutex TaskToSend\_mutex = new Mutex();

public Dictionary<string, string> AuthData = new Dictionary<string, string>

{

{ "Port1", null },

{ "Port2", null },

{ "local", null }

};

public Mutex AuthData\_mutex = new Mutex();

#region Описание Модели Данных

class inbox\_class

{

public string id { get; set; }

public string sender { get; set; }

public string recepient { get; set; }

public string re { get; set; }

public string msg { get; set; }

public string status { get; set; }

public string date\_received { get; set; }

public string foreign\_id { get; set; }

public inbox\_class()

{ }

public inbox\_class(inbox letter)

{

this.foreign\_id = letter.id.ToString();

this.sender = letter.sender.ToString();

this.recepient = letter.recepient.ToString();

this.re = letter.re.ToString();

this.msg = letter.msg.ToString();

this.status = letter.status.ToString();

this.date\_received = letter.date\_received.ToString();

this.id = "";

}

};

class outbox\_class

{

public string id { get; set; }

public string sender { get; set; }

public string recepient { get; set; }

public string re { get; set; }

public string msg { get; set; }

public string status { get; set; }

public string date\_sent { get; set; }

public outbox\_class()

{ }

public outbox\_class(outbox letter)

{

this.id = letter.id.ToString();

this.sender = letter.sender.ToString();

this.recepient = letter.recepient.ToString();

this.re = letter.re.ToString();

this.msg = letter.msg.ToString();

this.status = letter.status.ToString();

//this.date\_sent = letter.date\_sent.ToString();

}

};

#endregion

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ФЛАГИ ДЛЯ ОБНОВЛЕНИЯ ПАПОК С ПИСЬМАМИ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Mutex Inbox\_update\_mutex = new Mutex();

public Mutex Outbox\_update\_mutex = new Mutex();

public bool Inbox\_update\_needed = false;

public bool Outbox\_update\_needed = false;

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^//

#region КОДИРОВАНИЕ СООБЩЕНИЙ

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

КОДИРОВАНИЕ СООБЩЕНИЙ И СБОРКА КАДРОВ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Codings

{

public static ushort SetDatalen(byte len1, byte len2)

{

return BitConverter.ToUInt16(new byte[] { len1, len2 }, 0);

}

// Таблица для определения синдрома ошибки

public static Dictionary<byte, byte> synd\_table = new Dictionary<byte, byte>

{

{ 0x01, 0x01 },

{ 0x02, 0x02 },

{ 0x04, 0x04 },

{ 0x03, 0x08 },

{ 0x06, 0x10 },

{ 0x07, 0x20 },

{ 0x05, 0x40 }

};

//Получение битов контрольной суммы из исходных 4 бит //ОК

public static byte GetChecksum(byte data)

{

byte temp = data;

//Порождающий полином

byte poly = 0x58;

short shift\_count = 0;

byte mask = 0x40;

short counter = 20;

byte checksum;

while (counter > 0)

{

temp = (byte)(temp ^ poly);

if ((temp & mask) == mask)

{

continue;

}

else {

if (shift\_count < 3)

{

mask = (byte)(mask >> 1);

poly = (byte)(poly >> 1);

shift\_count++;

}

else { break; }

}

counter--;

}

checksum = temp;

return checksum;

}

//OK

public static byte EncodeDataBits(byte decoded\_value)

{

byte checksum = GetChecksum((byte)(decoded\_value << 3));

byte encoded\_value = (byte)((decoded\_value << 3) + checksum);

return encoded\_value;

}

//Принимает байт с контрольной суммой, возвращает полубайт

public static byte DecodeDataBits(byte encoded\_value)

{

byte decoded\_value;

byte checksum = GetChecksum(encoded\_value);

if (checksum == 0)

{

decoded\_value = (byte)(encoded\_value >> 3);

}

else

{

decoded\_value = (byte)((synd\_table[checksum] ^ encoded\_value) >> 3);

}

return decoded\_value;

}

//Принимает массив полубайтов win1251 -> выдает строку на русском языке в utf-8

public static string ByteMessageToString(byte[] message\_data)

{

int Message\_Len = (message\_data.Length);

byte[] bytestostring = new byte[Message\_Len / 2];

for (int i = 0; i < Message\_Len / 2; i++)

{

// Проверка crc в принятых байтах.

if (GetChecksum((byte)(message\_data[i \* 2] & 0x0F)) != (byte)(message\_data[i \* 2] >> 3))

{

return null;

}

if (GetChecksum((byte)(message\_data[i \* 2 + 1] & 0x0F)) != (byte)(message\_data[i \* 2 + 1] >> 3))

{

return null;

}

}

for (int i = 0; i < Message\_Len / 2; i++)

{

byte first = (byte)((message\_data[i \* 2] & 0x78) << 1);

byte second = (byte)((message\_data[i \* 2 + 1] & 0x78) >> 3);

byte code = (byte)(first + second);

bytestostring[i] = code;

}

byte[] utf8Bytes = Encoding.Convert(WIN1251, Encoding.UTF8, bytestostring);

string Received\_String = Encoding.UTF8.GetString(utf8Bytes);

return Received\_String;

}

////Принимает строку на русском языке в utf-8 -> Выдает массив полубайтов win1251

public static byte[] StringToByteMessage(string StringToSend)

{

byte[] BytesToSend = new byte[(StringToSend.Length) \* 2];

byte[] utf8bytes = Encoding.UTF8.GetBytes(StringToSend);

byte[] win1251Bytes = Encoding.Convert(Encoding.UTF8, WIN1251, utf8bytes);

string win1251string = WIN1251.GetString(win1251Bytes);

for (int i = 0; i < win1251Bytes.Length; i++)

{

byte nibble1 = EncodeDataBits((byte)(win1251Bytes[i] >> 4));

byte nibble2 = EncodeDataBits((byte)(win1251Bytes[i] & 0x0F));

BytesToSend[i \* 2] = nibble1;

BytesToSend[(i \* 2) + 1] = nibble2;

}

return BytesToSend;

}

};

// Определение типов кадров

public enum FrameType : byte

{

MEETING = 0x01,

DISCONNECT = 0x02,

LOGIN = 0x03,

LOGOUT = 0x04,

INFORMATION = 0x06,

OPENLETTER = 0x07,

ACK = 0x08,

RET = 0x09

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

СОЗДАНИЕ И РАЗБОР КАДРОВ ЛЮБОГО ТИПА

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//сборка кадра для отправки в порт

public byte[] CreateNewFrame(FrameType type, string senderstr, string datalenstr, string receiverstr, string payload, bool encoding = false)

{

List<byte> framebytes = new List<byte>();

framebytes.Add((byte)0xFF);

framebytes.Add((byte)type);

byte sender;

byte receiver;

if (byte.TryParse(senderstr, out sender) && byte.TryParse(receiverstr, out receiver))

{

framebytes.Add((byte)sender);

framebytes.Add((byte)receiver);

}

else {

MessageBox.Show("FormNewFrame(): Не удалось преобразовать адрес машины в байт", "Error!");

return null;

}

//Если тип кадра должен содержать данные

if (type == FrameType.INFORMATION || type == FrameType.LOGIN || type == FrameType.OPENLETTER || type == FrameType.MEETING)

{

UInt16 datalen;

if (UInt16.TryParse(datalenstr, out datalen))

{

byte highbyte = (BitConverter.GetBytes(datalen))[1];

byte lowbyte = (BitConverter.GetBytes(datalen))[0];

framebytes.Add((byte)highbyte);

framebytes.Add((byte)lowbyte);

//Если требуется кодирование данных

if (encoding == true)

{

byte[] double\_encoded\_payload = Codings.StringToByteMessage(payload);

framebytes.AddRange(double\_encoded\_payload);

}

else

{

byte[] utf8bytes = Encoding.UTF8.GetBytes(payload);

byte[] win1251Bytes = Encoding.Convert(Encoding.UTF8, WIN1251, utf8bytes);

framebytes.AddRange(win1251Bytes);

}

}

else {

MessageBox.Show("CreateNewFrame(): Не удалось преобразовать datalen в Uint16 ", "Error!");

return null;

}

}

//Добавление стопового байта и возврат сформированного кадра в виде byte[]

framebytes.Add((byte)0xFE);

return framebytes.ToArray();

}

//разбор кадра, полученного из порта. Возвращает структуру,заполненную принятыми данными

public DefaultFrame ParseReceivedFrame(byte[] frame, bool encoding = false)

{

DefaultFrame ParsedFrame = new DefaultFrame();

if (frame == null) { ParsedFrame.ResultOfParsing = "Fail"; return ParsedFrame; }

if (frame.Length < 5)

{

MessageBox.Show("ParseReceivedFrame(). Длина кадра меньше минимальной", "Error!");

ParsedFrame.ResultOfParsing = "Failed";

return ParsedFrame;

}

ParsedFrame.Startbyte = frame[0];

ParsedFrame.Frametype = frame[1];

ParsedFrame.OriginPort = ((UInt16)(frame[2])).ToString();

ParsedFrame.DestinationPort = ((UInt16)(frame[3])).ToString();

ParsedFrame.Stopbyte = frame[(frame.Length - 1)];

byte type = ParsedFrame.Frametype;

ParsedFrame.ResultOfParsing = "OK";

// Кадры данных типов могут быть носителями payload'а

if (type == (byte)FrameType.INFORMATION || type == (byte)FrameType.LOGIN || type == (byte)FrameType.OPENLETTER || type == (byte)FrameType.MEETING)

{

List<byte> data = new List<byte>();

// Старший байт - 4, младший - 5. В кадре передается в формате big-endian

ParsedFrame.datalen = BitConverter.ToUInt16(new byte[] { frame[5], frame[4] }, 0);

if (encoding == false)

{

// проверка на совпадение размера данных с указанным в кадре значением

// 2 флаговых байта, 2 - адресация, 2 - колчество символов win 1251 , 1 - тип кадра

if (frame.Length != (7 + ParsedFrame.datalen))

{

MessageBox.Show("ParseReceivedFrame(enc==false) Длина поля данных в кадре не совпадает с указанной", "Error!");

//Установка флага "Пришел битый кадр"

ParsedFrame.ResultOfParsing = "Fail";

return ParsedFrame;

}

else {

for (int i = 6; i < frame.Length - 1; i++)

{

data.Add(frame[i]);

}

}

ParsedFrame.MessageData = WIN1251.GetString(data.ToArray());

}

else if (encoding == true)

{

List<byte> doubled\_frame\_data = new List<byte>();

if (frame.Length != (7 + ParsedFrame.datalen \* 2))

{

MessageBox.Show("ParseReceivedFrame(enc==true) Длина поля данных в кадре не совпадает с указанной ", "Error!");

//Установка флага "Пришел битый кадр"

ParsedFrame.ResultOfParsing = "Fail";

return ParsedFrame;

}

for (int i = 6; i < frame.Length - 1; i++)

{

doubled\_frame\_data.Add(frame[i]);

}

//Возвращенная строка в UTF-8

string decoded\_from\_doubled = Codings.ByteMessageToString(doubled\_frame\_data.ToArray());

if (decoded\_from\_doubled == null)

{

MessageBox.Show("ParseReceivedFrame(encoding==true) Ошибка в crc кадра", "Error!");

//Установка флага "Пришел битый кадр"

ParsedFrame.ResultOfParsing = "Fail";

return ParsedFrame;

}

else

{ ParsedFrame.MessageData = decoded\_from\_doubled; }

}

}

//Если кадр не должен содержать данные

else

{

if (frame.Length > 5)

{

ParsedFrame.ResultOfParsing = "Fail";

return ParsedFrame;

}

}

return ParsedFrame;

}

//Структура для хранения результатов разбора любого кадра

public struct DefaultFrame

{

public byte Startbyte;

public byte Frametype;

public string OriginPort;

public string DestinationPort;

public UInt16 datalen;

public string MessageData;

public byte Stopbyte;

public string ResultOfParsing;

public string PortName;

}

/\*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

КОДИРОВАНИЕ СООБЩЕНИЙ

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\*/

#endregion

public Form1()

{

InitializeComponent();

string[] portslist = SerialPort.GetPortNames();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ЗАДАНИЕ ПАРАМЕТРОВ КОМПОРТА

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//// Начальное задание параметров ком портов

//Соединенные пары портов { COM3 <-> COM4, COM6 <-> COM7,COM8 <-> COM9 }

serialPort1.Encoding = WIN1251;

serialPort1.BaudRate = 9600;

serialPort1.DataBits = 8;

serialPort1.Parity = Parity.None;

serialPort1.StopBits = StopBits.One;

serialPort1.Handshake = Handshake.RequestToSend;

serialPort1.PortName = "COM3";

serialPort2.Encoding = WIN1251;

serialPort2.BaudRate = 9600;

serialPort2.DataBits = 8;

serialPort2.Parity = Parity.None;

serialPort2.StopBits = StopBits.One;

serialPort2.Handshake = Handshake.RequestToSend;

serialPort2.PortName = "COM6";

toolStripComboBox3.SelectedItem = "9600";

SelectedBaudrate = "9600";

toolStripComboBox1.Items.AddRange(SerialPort.GetPortNames());

toolStripComboBox2.Items.AddRange(SerialPort.GetPortNames());

//Port1

toolStripComboBox1.SelectedItem = "COM3";

SelectedPort1Name = "COM3";

//Port2

toolStripComboBox2.SelectedItem = "COM6";

SelectedPort2Name = "COM6";

this.button1.Enabled = true;

this.button4.Enabled = false;

this.AuthConnectButton.Enabled = true;

this.AuthDisconnectButton.Enabled = false;

/\*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

ЗАДАНИЕ ПАРАМЕТРОВ КОМПОРТА

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\*/

}

private void Form1\_Load(object sender, EventArgs e)

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ПОЛУЧЕНИЕ КАДРОВ ИЗ ПОРТОВ И ЗАПОЛНЕНИЕ СПИСКА ЗАДАНИЙ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public enum ReceiveState { SOF\_FOUND, EOF\_FOUND, FREE }

// Функции для считывания данных из ком порта в отдельном потоке

public void Serial1\_StartReceiving()

{

if (!serialPort1.IsOpen)

{

{ MessageBox.Show("Порт Закрыт!", serialPort1.PortName); }

}

//Очистка буфера перед началом нового сеанса

serialPort1.DiscardInBuffer();

int bytestoread;

while (serialPort1.IsOpen)

{

bytestoread = serialPort1.BytesToRead;

//Буфер для чтения из порта

byte[] ReceivedBytes = new byte[bytestoread];

if (bytestoread > 0)

{

try

{

//Чтение принятых данных из компорта в буферный массив принятых байтов

serialPort1.Read(ReceivedBytes, 0, bytestoread);

//Вход в критическую секцию

//разделяемый ресурс- Список байт, принятых из порта ReceivedFrames

ReceivedFrames\_mutex1.WaitOne();

//Запись новых данных из компорта в глобальный Массив Принятых данных

ReceivedFrames1.AddRange(ReceivedBytes);

ReceivedFrames\_mutex1.ReleaseMutex();

//Выход из критической секции

}

catch (InvalidOperationException ex)

{MessageBox.Show(ex.Message, "Error!"); }

}

Thread.Sleep(20);

}

}

// просматривает Список ReceivedFrames и разбирает найденные кадры

// Читает поток входящих байт и порта 1 и заполняет список заданий в формате One\_Task(Номер порта, Байты кадра)

public void FindFrameInPort1()

{

int startbyte;

int stopbyte;

List<byte> Frame = new List<byte>();

//Наачльное состояние функции при запуске

ReceiveState State = ReceiveState.FREE;

while (true)

{

//Вход в критическую секцию

//разделяемый ресурс- Список байт, принятых из порта

ReceivedFrames\_mutex1.WaitOne();

//Выполнение условия, только если новый кадр ещё не поступил, а старый уже обработан

if (State == ReceiveState.FREE)

{

startbyte = ReceivedFrames1.IndexOf(0xFF);

stopbyte = ReceivedFrames1.IndexOf(0xFE);

//Если Начало кадра найдено, то отмечаем новое состояние

if (startbyte != -1)

{

State = ReceiveState.SOF\_FOUND;

//Удаляем всё, что было до начала кадра, т.е. теперь буфер кадров содержит только начало кадра и возможно конец

ReceivedFrames1.RemoveRange(0, startbyte);

//Далее, если конец кадра не обнаружен, то вырезаем всё, что накопилось в буфере кадров в локальный контейнер для кадра

if (stopbyte == -1)

{

Frame.AddRange(ReceivedFrames1.GetRange(0, ReceivedFrames1.Count));

ReceivedFrames1.RemoveRange(0, ReceivedFrames1.Count);

}

//Иначе вырезаем лишь часть и разбираем кадр(Передача контейнера кадра на разбор )

else

if (stopbyte != -1)

{

//Получение новых индексов для урезанного списка

startbyte = ReceivedFrames1.IndexOf(0xFF);

stopbyte = ReceivedFrames1.IndexOf(0xFE);

if (stopbyte == -1)

{

Frame.AddRange(ReceivedFrames1.GetRange(0, ReceivedFrames1.Count));

ReceivedFrames1.RemoveRange(0, ReceivedFrames1.Count);

}

else {

Frame.AddRange(ReceivedFrames1.GetRange(0, stopbyte - startbyte + 1));

ReceivedFrames1.RemoveRange(0, stopbyte - startbyte + 1);

State = ReceiveState.EOF\_FOUND;

//Запуск разбора кадра Frame, после этого перевод в состояние FREE И очистка контейнера кадра

//Запись найденного кадра в список заданий

TaskReceived\_mutex.WaitOne();

TasksReceived.Add(new One\_Task("Port1", Frame.ToArray()));

TaskReceived\_mutex.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

}

}

}

//Если Начало кадра не найдено и при этом система готова к приему нового кадра,

// значит в порт пришел мусор, его удаляем из буфера кадров

else

{ ReceivedFrames1.Clear();}

}

//Если уже был обнаружен стартовый байт, значит принимаем всё до конца буфера или до конечного байта

//Заход в эту область происходит, если размер кадра оказался больше, чем буфер приема

else if (State == ReceiveState.SOF\_FOUND)

{

stopbyte = ReceivedFrames1.IndexOf(0xFE);

//Если не найден стоповый байт, достаём из буфера кадров всё, что там есть

if (stopbyte == -1)

{

startbyte = ReceivedFrames1.IndexOf(0xFF);

if ((startbyte != -1) && State == ReceiveState.SOF\_FOUND)

{

MessageBox.Show("Функция FindFrame1(), найден кадр без стопового байта", "Error!");

ReceivedFrames\_mutex1.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

continue;

}

Frame.AddRange(ReceivedFrames1.GetRange(0, ReceivedFrames1.Count));

ReceivedFrames1.RemoveRange(0, ReceivedFrames1.Count);

}

//Если стоповый байт найден, тогда вырезаем всё, что расположено до стопового байта, далее разбор кадра

else if (stopbyte != -1)

{

startbyte = ReceivedFrames1.IndexOf(0xFF);

if ((startbyte != -1) && State == ReceiveState.SOF\_FOUND)

{

MessageBox.Show("Функция FindFrame1(), найден кадр без стопового байта", "Error!");

ReceivedFrames\_mutex1.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

continue;

}

stopbyte = ReceivedFrames1.IndexOf(0xFE);

//Добавляем в контейнер кадра до стопового байта, остальное переносим в начало буфера

Frame.AddRange(ReceivedFrames1.GetRange(0, stopbyte + 1));

ReceivedFrames1.RemoveRange(0, stopbyte + 1);

State = ReceiveState.EOF\_FOUND;

//Запуск разбора кадра Frame, после этого перевод в состояние FREE И очистка контейнера кадра

//Запись найденного кадра в список заданий

TaskReceived\_mutex.WaitOne();

TasksReceived.Add(new One\_Task("Port1", Frame.ToArray()));

TaskReceived\_mutex.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

}

}

// Переход в эту область не должен происходить

else if (State == ReceiveState.EOF\_FOUND)

{

MessageBox.Show(" Функция FindFrame1.\r\n Состояние осталось EOF\_FOUND в начале прохода цикла", "Error!");

Frame.Clear();

State = ReceiveState.FREE;

continue;

}

ReceivedFrames\_mutex1.ReleaseMutex();

Thread.Sleep(10);

}

}

//Всё тоже самое, что и для первого порта(Serial1\_StartReceiving).

public void Serial2\_StartReceiving()

{

if (!serialPort2.IsOpen)

{ MessageBox.Show("Порт Закрыт!", serialPort2.PortName);}

//Очистка буфера перед началом нового сеанса

serialPort2.DiscardInBuffer();

int bytestoread;

while (serialPort2.IsOpen)

{

bytestoread = serialPort2.BytesToRead;

//Буфер для чтения из порта

byte[] ReceivedBytes = new byte[bytestoread];

if (bytestoread > 0)

{

try

{

//Чтение принятых данных из компорта в буферный массив принятых байтов

serialPort2.Read(ReceivedBytes, 0, bytestoread);

//Вход в критическую секцию

//разделяемый ресурс- Список байт, принятых из порта ReceivedFrames

ReceivedFrames\_mutex2.WaitOne();

//Запись новых данных из компорта в глобальный Массив Принятых данных

ReceivedFrames2.AddRange(ReceivedBytes);

ReceivedFrames\_mutex2.ReleaseMutex();

//Выход из критической секции

}

catch (InvalidOperationException ex)

{ MessageBox.Show(ex.Message, "Error!"); }

}

Thread.Sleep(20);

}

}

//Всё тоже самое, что и для первого порта(FindFrameInPort1).

public void FindFrameInPort2()

{

int startbyte;

int stopbyte;

List<byte> Frame = new List<byte>();

//Наачльное состояние функции при запуске

ReceiveState State = ReceiveState.FREE;

while (true)

{

//Вход в критическую секцию

//разделяемый ресурс- Список байт, принятых из порта

ReceivedFrames\_mutex2.WaitOne();

//Выполнение условия, только если новый кадр ещё не поступил, а старый уже обработан

if (State == ReceiveState.FREE)

{

startbyte = ReceivedFrames2.IndexOf(0xFF);

stopbyte = ReceivedFrames2.IndexOf(0xFE);

//Если Начало кадра найдено, то отмечаем новое состояние

if (startbyte != -1)

{

State = ReceiveState.SOF\_FOUND;

//Удаляем всё, что было до начала кадра, т.е. теперь буфер кадров содержит только начало кадра и возможно конец

ReceivedFrames2.RemoveRange(0, startbyte);

//Далее, если конец кадра не обнаружен, то вырезаем всё, что накопилось в буфере кадров в локальный контейнер для кадра

if (stopbyte == -1)

{

Frame.AddRange(ReceivedFrames2.GetRange(0, ReceivedFrames2.Count));

ReceivedFrames2.RemoveRange(0, ReceivedFrames2.Count);

}

//Иначе вырезаем лишь часть и разбираем кадр(Передача контейнера кадра на разбор )

else

if (stopbyte != -1)

{

//Получение новых индексов для урезанного списка

startbyte = ReceivedFrames2.IndexOf(0xFF);

stopbyte = ReceivedFrames2.IndexOf(0xFE);

if (stopbyte == -1)

{

Frame.AddRange(ReceivedFrames2.GetRange(0, ReceivedFrames2.Count));

ReceivedFrames2.RemoveRange(0, ReceivedFrames2.Count);

}

else {

Frame.AddRange(ReceivedFrames2.GetRange(0, stopbyte - startbyte + 1));

ReceivedFrames2.RemoveRange(0, stopbyte - startbyte + 1);

State = ReceiveState.EOF\_FOUND;

//Запуск разбора кадра Frame, после этого перевод в состояние FREE И очистка контейнера кадра

//Запись найденного кадра в список заданий

TaskReceived\_mutex.WaitOne();

TasksReceived.Add(new One\_Task("Port2", Frame.ToArray()));

TaskReceived\_mutex.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

}

}

}

//Если Начало кадра не найдено и при этом система готова к приему нового кадра,

// значит в порт пришел мусор, его удаляем из буфера кадров

else

{

ReceivedFrames2.Clear();

}

}

//Если уже был обнаружен стартовый байт, значит принимаем всё до конца буфера или до конечного байта

//Заход в эту область происходит, если разиер кадра оказался больше, чем буфер приема

else if (State == ReceiveState.SOF\_FOUND)

{

stopbyte = ReceivedFrames2.IndexOf(0xFE);

//Если не найден стоповый байт, достаём из буфера кадров всё, что там есть

if (stopbyte == -1)

{

startbyte = ReceivedFrames2.IndexOf(0xFF);

if ((startbyte != -1) && State == ReceiveState.SOF\_FOUND)

{

MessageBox.Show("Функция FindFrame2(), найден кадр без стопового байта", "Error!");

ReceivedFrames\_mutex2.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

continue;

}

Frame.AddRange(ReceivedFrames2.GetRange(0, ReceivedFrames2.Count));

ReceivedFrames2.RemoveRange(0, ReceivedFrames2.Count);

}

//Если стоповый байт найден, тогда вырезаем всё, что расположено до стопового байта, далее разбор кадра

else if (stopbyte != -1)

{

startbyte = ReceivedFrames2.IndexOf(0xFF);

if ((startbyte != -1) && State == ReceiveState.SOF\_FOUND)

{

MessageBox.Show("Функция FindFrame2(), найден кадр без стопового байта", "Error!");

ReceivedFrames\_mutex2.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

continue;

}

stopbyte = ReceivedFrames2.IndexOf(0xFE);

//Добавляем в контейнер кадра до стопового байта, остальное переносим в начало буфера

Frame.AddRange(ReceivedFrames2.GetRange(0, stopbyte + 1));

ReceivedFrames2.RemoveRange(0, stopbyte + 1);

State = ReceiveState.EOF\_FOUND;

//Запуск разбора кадра Frame, после этого перевод в состояние FREE И очистка контейнера кадра

//Запись найденного кадра в список заданий

TaskReceived\_mutex.WaitOne();

TasksReceived.Add(new One\_Task("Port2", Frame.ToArray()));

TaskReceived\_mutex.ReleaseMutex();

Frame.Clear();

State = ReceiveState.FREE;

}

}

// Переход в эту область не должен происходить

else if (State == ReceiveState.EOF\_FOUND)

{

MessageBox.Show(" Функция FindFrame2.\r\n Состояние осталось EOF\_FOUND в начале прохода цикла", "Error!");

Frame.Clear();

State = ReceiveState.FREE;

continue;

}

//Выход из критической секции

ReceivedFrames\_mutex2.ReleaseMutex();

Thread.Sleep(10);

}

}

/\*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Получение кадров из ком портов и внесение их в список заданий

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ВЫПОЛНЕНИЕ ЗАДАНИЙ НА ПРИЕМ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Новые задания появляются в списке TasksReceived из функции FindFrame1 и FindFrame2

public void TaskHandler()

{

bool task\_received = false;

while (true)

{

task\_received = false;

One\_Task task = new One\_Task();

//Получение доступа к списку заданий

TaskReceived\_mutex.WaitOne();

if (TasksReceived.Count != 0)

{

task = TasksReceived[0];

//Удаление кадра и списка заданий

TasksReceived.RemoveAt(0);

TaskReceived\_mutex.ReleaseMutex();

task\_received = true;

}

else

{ TaskReceived\_mutex.ReleaseMutex(); }

if (task\_received == true)

{

byte[] frame = task.Frame;

// не Number, а Name

//имена = {"Port1","Port2"}

string PortNumber = task.PortNum;

DefaultFrame ReceivedFrameStruct = ParseReceivedFrame(frame);

if (ReceivedFrameStruct.ResultOfParsing == "OK")

{

ReceivedFrameStruct.PortName = PortNumber;

byte frametype = ReceivedFrameStruct.Frametype;

//ОПРЕДЕЛЕНИЕ НЕОБХОДИМОГО ОБРАБОТЧИКА

if (frametype == (byte)FrameType.ACK || frametype == (byte)FrameType.RET)

{

if (frametype == (byte)FrameType.ACK)

FrameReceivedAck(ReceivedFrameStruct);

else

FrameReceivedRet(ReceivedFrameStruct);

}

if (frametype == (byte)FrameType.MEETING || frametype == (byte)FrameType.DISCONNECT)

{

if (DEBUG\_MODE) MessageBox.Show("Получен кадр типа meeting(не используется)", "Error!");

}

if (frametype == (byte)FrameType.LOGIN || frametype == (byte)FrameType.LOGOUT)

{

if (frametype == (byte)FrameType.LOGIN)

FrameReceivedLogin(ReceivedFrameStruct);

else

FrameReceivedLogout(ReceivedFrameStruct);

}

if (frametype == (byte)FrameType.OPENLETTER || frametype == (byte)FrameType.INFORMATION)

{

if (frametype == (byte)FrameType.OPENLETTER)

FrameReceivedOpenLetter(ReceivedFrameStruct);

else

FrameReceivedInformation(ReceivedFrameStruct);

}

}

//Если принятый кадр оказался битым, то отправка

//отправка RET кадра в PortName или машине OriginPort

else {

if (RET\_ALLOWED)

{

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(new One\_Task(

PortNumber, new byte[] { 0xFF, (byte)FrameType.RET, 0x00, 0x00, 0xFE }));

TaskToSend\_mutex.ReleaseMutex();

}

}

}

Thread.Sleep(20);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ОБРАБОТЧИКИ СОБЫТИЙ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void FrameReceivedLogin(DefaultFrame ReceivedFrame)

{

////Отправка ACK кадра в PortName

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(

new One\_Task(

ReceivedFrame.PortName, CreateNewFrame(FrameType.ACK, "0", null, "0", null, false)

));

TaskToSend\_mutex.ReleaseMutex();

if (ReceivedFrame.PortName == "Port1")

{

AuthData\_mutex.WaitOne();

AuthData["Port1"] = ReceivedFrame.MessageData;

AuthData\_mutex.ReleaseMutex();

}

if (ReceivedFrame.PortName == "Port2")

{

AuthData\_mutex.WaitOne();

AuthData["Port2"] = ReceivedFrame.MessageData;

AuthData\_mutex.ReleaseMutex();

}

string port1\_auth\_data, port2\_auth\_data;

AuthData\_mutex.WaitOne();

port1\_auth\_data = AuthData["Port1"];

port2\_auth\_data = AuthData["Port2"];

AuthData\_mutex.ReleaseMutex();

//Если все login'ы были получены

if (port1\_auth\_data != null && port2\_auth\_data != null)

{

Auth\_status\_mutex.WaitOne();

Auth\_status["ACK\_local"] = "Received";

Auth\_status\_mutex.ReleaseMutex();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Логины абонентов получены\r\n" });

}

}

//Готово

public void FrameReceivedAck(DefaultFrame ReceivedFrameStruct)

{

string status\_1\_auth;

string status\_2\_auth;

//Получение состояния авторизации

Auth\_status\_mutex.WaitOne();

status\_1\_auth = Auth\_status["ACK1"];

status\_2\_auth = Auth\_status["ACK2"];

Auth\_status\_mutex.ReleaseMutex();

if (status\_1\_auth == "undef" || status\_2\_auth == "undef")

{

if (ReceivedFrameStruct.PortName == "Port1")

{

Ack1\_mutex\_Auth.WaitOne();

Ack1\_awaited\_Auth = 0;

Ack1\_mutex\_Auth.ReleaseMutex();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{ "Получен отчет о приеме Login из Port 1 \r\n" });

Auth\_status\_mutex.WaitOne();

Auth\_status["ACK1"] = "Received";

Auth\_status\_mutex.ReleaseMutex();

}

if (ReceivedFrameStruct.PortName == "Port2")

{

Ack2\_mutex\_Auth.WaitOne();

Ack2\_awaited\_Auth = 0;

Ack2\_mutex\_Auth.ReleaseMutex();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{ "Получен отчет о приеме Login из Port 2 \r\n" });

Auth\_status\_mutex.WaitOne();

Auth\_status["ACK2"] = "Received";

Auth\_status\_mutex.ReleaseMutex();

}

}

bool auth\_stat = false;

Auth\_status\_mutex.WaitOne();

if (Auth\_status["ACK1"] != "undef" && Auth\_status["ACK\_local"] != "undef" && Auth\_status["ACK2"] != "undef")

{

auth\_stat = true;

}

Auth\_status\_mutex.ReleaseMutex();

//Если авторизован, тогда приходящие ACK подтверждают прием сообщения infoframe

if (auth\_stat)

{

if (ReceivedFrameStruct.PortName == "Port1")

{

LastFrame\_ToSend\_mutex.WaitOne();

LastFrameSenttoPort1.PortNum = null;

LastFrame\_ToSend\_mutex.ReleaseMutex();

}

if (ReceivedFrameStruct.PortName == "Port2")

{

LastFrame\_ToSend\_mutex.WaitOne();

LastFrameSenttoPort2.PortNum = null;

LastFrame\_ToSend\_mutex.ReleaseMutex();

}

}

// Отметка отправленного сообщения как доставленного

// Далее, открыть возможность оправлять новые сообщения

}

public void FrameReceivedRet(DefaultFrame frame)

{

// Здесь получаем значение last send frame о отправляем его вкомпорт

LastFrame\_ToSend\_mutex.WaitOne();

One\_Task LastFrameSent\_local1 = LastFrameSenttoPort1;

One\_Task LastFrameSent\_local2 = LastFrameSenttoPort2;

LastFrame\_ToSend\_mutex.ReleaseMutex();

TaskToSend\_mutex.WaitOne();

if (frame.PortName == "Port1")

{

if (LastFrameSent\_local1.PortNum != null && LastFrameSent\_local1.Frame != null)

TasksToSend.Add(LastFrameSent\_local1);

}

if (frame.PortName == "Port2")

{

if (LastFrameSent\_local1.PortNum != null && LastFrameSent\_local1.Frame != null)

TasksToSend.Add(LastFrameSent\_local2);

}

TaskToSend\_mutex.ReleaseMutex();

}

public void FrameReceivedLogout(DefaultFrame frame)

{

AuthData\_mutex.WaitOne();

AuthData["Port1"] = null;

AuthData["Port2"] = null;

AuthData["local"] = null;

AuthData\_mutex.ReleaseMutex();

Auth\_status\_mutex.WaitOne();

Auth\_status["ACK\_local"] = "undef";

Auth\_status["ACK1"] = "undef";

Auth\_status["ACK2"] = "undef";

Auth\_status\_mutex.ReleaseMutex();

BeginInvoke(new Set\_ButtonState(Set\_AuthConnectButton), new object[] { true });

BeginInvoke(new Set\_ButtonState(Set\_AuthDisconnectButton), new object[] { false });

BeginInvoke(new SetTextDeleg(addtotextbox1),

new object[] { "Получен кадр о деавторизации, логины сброшены" });

}

// Обработчик открытия письма (готов)

// В БД ищется письмо, затем отмечается как прочитанное

public void FrameReceivedOpenLetter(DefaultFrame frame)

{

//Подключение к бд, поиск письма с тем id, который указан в принятом пакете

//После этого обновление таблицы принятых сообщений на ui (установкой Outbox\_update\_needed= true)

long LetterId = long.Parse(frame.MessageData);

using (CourseDB db = new CourseDB())

{

var result = db.outbox.SingleOrDefault(x => x.id == LetterId);

if (result != null)

{

if (result.status == "Прочитано")

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { $" Сообщение : письмо с id = {LetterId} уже было прочитано\r\n" });

}

else {

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { $" Сообщение : письмо с id = {LetterId} было прочитано\r\n" });

result.status = "Прочитано";

db.SaveChanges();

//Отметка для обновления формы отправленных сообщений

Outbox\_update\_mutex.WaitOne();

Outbox\_update\_needed = true;

Outbox\_update\_mutex.ReleaseMutex();

}

}

}

}

//Готов

public void FrameReceivedInformation(DefaultFrame local\_frame)

{

byte[] framedata = WIN1251.GetBytes(local\_frame.MessageData);

string framestr = WIN1251.GetString(framedata);

inbox\_class message\_data = JsonConvert.DeserializeObject<inbox\_class>(local\_frame.MessageData);

if (local\_frame.ResultOfParsing == "OK")

{

BeginInvoke

(new SetTextDeleg(addtotextbox1),

new object[] { $"Получено письмо от {message\_data.sender}" + "\r\n" });

long foreign\_id = long.Parse(message\_data.id);

inbox received\_letter = new inbox();

received\_letter.foreign\_id = foreign\_id;

received\_letter.re = message\_data.re;

received\_letter.msg = message\_data.msg;

received\_letter.recepient = message\_data.recepient;

received\_letter.sender = message\_data.sender;

received\_letter.status = "Принято";

using (CourseDB db = new CourseDB())

{

bool letter\_already\_exists = false;

try

{

inbox letter = db.inbox.FirstOrDefault(x => x.foreign\_id == received\_letter.foreign\_id);

if (letter.foreign\_id == received\_letter.foreign\_id)

{ letter\_already\_exists = true; }

}

catch

{ letter\_already\_exists = false; }

if (!letter\_already\_exists)

{

db.inbox.Add(received\_letter);

db.SaveChanges();

}

db.Dispose();

}

Inbox\_update\_mutex.WaitOne();

Inbox\_update\_needed = true;

Inbox\_update\_mutex.ReleaseMutex();

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(new One\_Task(local\_frame.PortName,

CreateNewFrame(FrameType.ACK, "0", null, "0", null, false))

);

TaskToSend\_mutex.ReleaseMutex();

}

else

{

if (RET\_ALLOWED)

{

// Если сообщение кривое, то отправляем RET в порт из которого он был принят

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(new One\_Task(local\_frame.PortName,

CreateNewFrame(FrameType.RET, "0", null, "0", null, false)

));

TaskToSend\_mutex.ReleaseMutex();

}

}

}

/\*

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

функции для исполнения заданий, полученных из списка с заданиями

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ОТПРАВКА НОВОГО ПИСЬМА

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//Источник особой уличной магии

private volatile Type \_dependency;

public void MyClass()

{ \_dependency = typeof(System.Data.Entity.SqlServer.SqlProviderServices); }

public void Wait\_for\_info\_ack1()

{

int counter = 0;

while (true)

{

One\_Task frame\_acked1;

LastFrame\_ToSend\_mutex.WaitOne();

frame\_acked1 = LastFrameSenttoPort1;

LastFrame\_ToSend\_mutex.ReleaseMutex();

bool frame\_is\_to\_resend = true;

if (frame\_acked1.PortNum == null)

{ frame\_is\_to\_resend = false; }

if (!frame\_is\_to\_resend)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{ "Сообщение доставлено"});

using (CourseDB db = new CourseDB())

{

DefaultFrame a = ParseReceivedFrame(LastFrameSenttoPort1.Frame);

if (a.ResultOfParsing != "OK") { return; }

string id\_string = JsonConvert.DeserializeObject<outbox\_class>(a.MessageData).id;

long id\_val = long.Parse(id\_string);

var last\_letter = db.outbox.FirstOrDefault<outbox>(x => x.id == id\_val);

last\_letter.status = "Доставлено";

db.SaveChanges();

}

Outbox\_update\_mutex.WaitOne();

Outbox\_update\_needed = true;

Outbox\_update\_mutex.ReleaseMutex();

return;

}

if (frame\_is\_to\_resend)

{

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(frame\_acked1);

TaskToSend\_mutex.ReleaseMutex();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{$"сообщение(попытка:{counter}, порт:{frame\_acked1.PortNum}"});

counter++;

}

if (counter > 10 && frame\_is\_to\_resend)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{ "Сообщение не было доставлено"});

using (CourseDB db = new CourseDB())

{

DefaultFrame a = ParseReceivedFrame(LastFrameSenttoPort1.Frame);

string id\_string = JsonConvert.DeserializeObject<outbox\_class>(a.MessageData).id;

long id\_val = long.Parse(id\_string);

var last\_letter = db.outbox.FirstOrDefault<outbox>(x => x.id == id\_val);

last\_letter.status = "Не Доставлено";

db.SaveChanges();

}

Outbox\_update\_mutex.WaitOne();

Outbox\_update\_needed = true;

Outbox\_update\_mutex.ReleaseMutex();

return;

}

Thread.Sleep(4000);

}

}

public void Wait\_for\_info\_ack2()

{

int counter = 0;

while (true)

{

One\_Task frame\_acked2;

LastFrame\_ToSend\_mutex.WaitOne();

frame\_acked2 = LastFrameSenttoPort2;

LastFrame\_ToSend\_mutex.ReleaseMutex();

bool frame\_is\_to\_resend = true;

if (frame\_acked2.PortNum == null)

{ frame\_is\_to\_resend = false; }

if (!frame\_is\_to\_resend)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{ "Сообщение доставлено"});

using (CourseDB db = new CourseDB())

{

DefaultFrame a = ParseReceivedFrame(LastFrameSenttoPort2.Frame);

string id\_string = JsonConvert.DeserializeObject<outbox\_class>(a.MessageData).id;

long id\_val = long.Parse(id\_string);

var last\_letter = db.outbox.FirstOrDefault<outbox>(x => x.id == id\_val);

last\_letter.status = "Доставлено";

db.SaveChanges();

}

Outbox\_update\_mutex.WaitOne();

Outbox\_update\_needed = true;

Outbox\_update\_mutex.ReleaseMutex();

return;

}

if (frame\_is\_to\_resend)

{

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(frame\_acked2);

TaskToSend\_mutex.ReleaseMutex();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{$"сообщение(попытка:{counter}, порт:{frame\_acked2.PortNum}"});

counter++;

}

if (counter > 10 && frame\_is\_to\_resend)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]

{ "Сообщение не было доставлено"});

using (CourseDB db = new CourseDB())

{

DefaultFrame a = ParseReceivedFrame(LastFrameSenttoPort2.Frame);

string id\_string = JsonConvert.DeserializeObject<outbox\_class>(a.MessageData).id;

long id\_val = long.Parse(id\_string);

var last\_letter = db.outbox.FirstOrDefault<outbox>(x => x.id == id\_val);

last\_letter.status = "Не Доставлено";

db.SaveChanges();

}

Outbox\_update\_mutex.WaitOne();

Outbox\_update\_needed = true;

Outbox\_update\_mutex.ReleaseMutex();

return;

}

Thread.Sleep(4000);

}

}

public void SendNewLetterButton\_Click(object sender, EventArgs e)

{

string Re\_string = ReTextbox.Text;

string Receiver\_name = ReceiverComboBox.SelectedItem.ToString();

string Letter\_Message = LetterTextBox.Text;

Phys\_status1\_mutex.WaitOne();

Connection\_Status p1 = Phys\_status1;

Phys\_status1\_mutex.ReleaseMutex();

Phys\_status2\_mutex.WaitOne();

Connection\_Status p2 = Phys\_status2;

Phys\_status2\_mutex.ReleaseMutex();

if (p1 != Connection\_Status.CONNECTED || p2 != Connection\_Status.CONNECTED)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "физическое соединение не установлено\r\n" });

return;

}

if (AuthData["Port1"] != null && AuthData["Port2"] != null && AuthData["local"] == null)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Логины не определены\r\n" });

if (DEBUG\_MODE)

{

MessageBox.Show("SendNewLetterButton\_Click(): логическое соединение не установлено", "Error!");

}

return;

}

using (CourseDB db = new CourseDB())

{

outbox letter = new outbox();

letter.re = Re\_string;

letter.sender = AuthData["local"];

letter.recepient = Receiver\_name;

letter.status = "Отправлено";

letter.msg = Letter\_Message;

db.outbox.Add(letter);

db.SaveChanges();

}

long max;

outbox a;

using (CourseDB db = new CourseDB())

{

max = db.outbox.Max(x => x.id);

a = db.outbox.FirstOrDefault(x => x.id == max);

}

string letter\_local\_id = a.id.ToString();

string receiver\_port = AuthData.FirstOrDefault(x => x.Value == Receiver\_name).Key;

outbox\_class letter\_payload\_obj = new outbox\_class(a);

string letter\_payload\_string = JsonConvert.SerializeObject(letter\_payload\_obj);

string letter\_len = letter\_payload\_string.Length.ToString();

byte[] Letter\_frame\_to\_send = CreateNewFrame(

FrameType.INFORMATION, "0", letter\_len, "0", letter\_payload\_string, false);

if (receiver\_port == "Port1")

{

LastFrame\_ToSend\_mutex.WaitOne();

LastFrameSenttoPort1 = new One\_Task(receiver\_port, Letter\_frame\_to\_send);

LastFrame\_ToSend\_mutex.ReleaseMutex();

Thread Wait\_for\_info\_ack1thr = new Thread(Wait\_for\_info\_ack1);

Wait\_for\_info\_ack1thr.IsBackground = true;

Wait\_for\_info\_ack1thr.Start();

}

if (receiver\_port == "Port2")

{

LastFrame\_ToSend\_mutex.WaitOne();

LastFrameSenttoPort2 = new One\_Task(receiver\_port, Letter\_frame\_to\_send);

LastFrame\_ToSend\_mutex.ReleaseMutex();

Thread Wait\_for\_info\_ack2thr = new Thread(Wait\_for\_info\_ack2);

Wait\_for\_info\_ack2thr.IsBackground = true;

Wait\_for\_info\_ack2thr.Start();

}

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(new One\_Task(receiver\_port, Letter\_frame\_to\_send));

TaskToSend\_mutex.ReleaseMutex();

Outbox\_update\_mutex.WaitOne();

Outbox\_update\_needed = true;

Outbox\_update\_mutex.ReleaseMutex();

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

// Делегат используется для записи в UI control из потока не-UI

private delegate void SetTextDeleg(string text);

private delegate void FillComboBoxDeleg();

//Заполняет список авторизованных пользователей

public void FillReceiverComboBox()

{

ReceiverComboBox.Items.Clear();

AuthData\_mutex.WaitOne();

string[] values = new string[] { AuthData["Port1"], AuthData["Port2"] };

ReceiverComboBox.Items.AddRange(values);

AuthData\_mutex.ReleaseMutex();

}

public void settextbox1(string text)

{ textBox1.Text = text; }

public void addtotextbox1(string text)

{

textBox1.AppendText(text);

}

public void OpenSerial1()

{

//Установка параметров компорта1

SelectedPort1Name\_mutex.WaitOne();

try

{ serialPort1.PortName = SelectedPort1Name; }

catch (Exception ex)

{ MessageBox.Show(ex.Message, serialPort1.PortName); }

SelectedPort1Name\_mutex.ReleaseMutex();

SelectedBaudrate\_mutex.WaitOne();

serialPort1.BaudRate = int.Parse(SelectedBaudrate);

SelectedBaudrate\_mutex.ReleaseMutex();

try

{

serialPort1.Open();

serialPort1.DiscardOutBuffer();

serialPort1.DiscardInBuffer();

serialPort1.DtrEnable = true;

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Порт1 был открыт\r\n" });

return;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, serialPort1.PortName);

return;

}

}

public void OpenSerial2()

{

//Установка параметров компорта2

SelectedPort2Name\_mutex.WaitOne();

try

{ serialPort2.PortName = SelectedPort2Name; }

catch (Exception ex)

{ MessageBox.Show(ex.Message, serialPort1.PortName); };

SelectedPort2Name\_mutex.ReleaseMutex();

// serialPort2.PortName = "COM6";

SelectedBaudrate\_mutex.WaitOne();

serialPort2.BaudRate = int.Parse(SelectedBaudrate);

SelectedBaudrate\_mutex.ReleaseMutex();

try

{

serialPort2.Open();

serialPort2.DiscardOutBuffer();

serialPort2.DiscardInBuffer();

serialPort2.DtrEnable = true;

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Порт2 был открыт\r\n" });

return;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, serialPort2.PortName);

return;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ПРОВЕРКА СОСТОЯНИЯ ПОРТОВ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Делегат используется для записи в UI control из потока не-UI

private delegate void SetPortState(string text);

public void setport1state(string text)

{ port1state\_label.Text = text; }

public void setport2state(string text)

{ port2state\_label.Text = text; }

//Мониторит физическое состояние портов

public void serial1\_monitor()

{

while (true)

{

if (serialPort1.IsOpen)

{

try

{

if (serialPort1.CDHolding || serialPort1.DsrHolding)

{

BeginInvoke(new SetTextDeleg(setport1state), new object[] { "Подключен" });

Phys\_status1\_mutex.WaitOne();

Phys\_status1 = Connection\_Status.CONNECTED;

Phys\_status1\_mutex.ReleaseMutex();

}

else {

BeginInvoke(new SetTextDeleg(setport1state), new object[] { "Отключен" });

Phys\_status1\_mutex.WaitOne();

Phys\_status1 = Connection\_Status.CONNECTION\_WAIT;

Phys\_status1\_mutex.ReleaseMutex();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Data.ToString(), "Error!");

}

}

else

{

BeginInvoke(new SetTextDeleg(setport1state), new object[] { "Порт1 закрыт" });

Phys\_status1\_mutex.WaitOne();

Phys\_status1 = Connection\_Status.DISCONNECTED;

Phys\_status1\_mutex.ReleaseMutex();

}

Thread.Sleep(200);

}

}

public void serial2\_monitor()

{

while (true)

{

if (serialPort2.IsOpen)

{

try

{

if (serialPort2.CDHolding || serialPort2.DsrHolding)

{

BeginInvoke(new SetTextDeleg(setport2state), new object[] { "Подключен" });

Phys\_status2\_mutex.WaitOne();

Phys\_status2 = Connection\_Status.CONNECTED;

Phys\_status2\_mutex.ReleaseMutex();

}

else {

BeginInvoke(new SetTextDeleg(setport2state), new object[] { "Отключен" });

Phys\_status2\_mutex.WaitOne();

Phys\_status2 = Connection\_Status.CONNECTION\_WAIT;

Phys\_status2\_mutex.ReleaseMutex();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Data.ToString(), "Error!");

}

}

else

{

BeginInvoke(new SetTextDeleg(setport2state), new object[] { "Порт2 закрыт" });

Phys\_status2\_mutex.WaitOne();

Phys\_status2 = Connection\_Status.DISCONNECTED;

Phys\_status2\_mutex.ReleaseMutex();

}

Thread.Sleep(200);

}

}

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^//

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ЗДЕСЬ НАЧАЛО ВСЕГО

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

//Открытие портов

Thread SerOpenthread1 = new Thread(OpenSerial1);

Thread SerOpenthread2 = new Thread(OpenSerial2);

SerOpenthread1.IsBackground = true;

SerOpenthread1.Start();

SerOpenthread2.IsBackground = true;

SerOpenthread2.Start();

Thread.Sleep(300);

if (!serialPort1.IsOpen || !serialPort2.IsOpen)

{

if (DEBUG\_MODE) MessageBox.Show("Не удалось открыть порты", "Warning");

if (serialPort1.IsOpen) serialPort1.Close();

if (serialPort2.IsOpen) serialPort2.Close();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Не удалось открыть порты\r\n" });

return;

}

this.button1.Enabled = false;

this.button4.Enabled = true;

//После запуска потока, метод StartReceiving Осуществляет прием из входного буфера в программный буфер принятых байтов

Thread ReceiverThr1 = new Thread(Serial1\_StartReceiving);

ReceiverThr1.IsBackground = true;

ReceiverThr1.Start();

//FindFrame просматривает програмный буфер принятых байтов и составляет кадры, затем помещает в список кадров(заданий)

Thread ParseFrameThr1 = new Thread(FindFrameInPort1);

ParseFrameThr1.IsBackground = true;

ParseFrameThr1.Start();

//После запуска потока, метод StartReceiving Осуществляет прием из входного буфера в программный буфер принятых байтов

Thread ReceiverThr2 = new Thread(Serial2\_StartReceiving);

ReceiverThr2.IsBackground = true;

ReceiverThr2.Start();

//FindFrame просматривает програмный буфер принятых байтов и составляет кадры, зате помещает в список кадров(заданий)

Thread ParseFrameThr2 = new Thread(FindFrameInPort2);

ParseFrameThr2.IsBackground = true;

ParseFrameThr2.Start();

//Потоки нужны для определения состояния абонентов на физическом уровне

//Состояние пишут в UI и в Phys\_status 1|2

Thread serial1\_mon\_thr = new Thread(serial1\_monitor);

serial1\_mon\_thr.IsBackground = true;

serial1\_mon\_thr.Start();

Thread serial2\_mon\_thr = new Thread(serial2\_monitor);

serial2\_mon\_thr.IsBackground = true;

serial2\_mon\_thr.Start();

//Обработка заданий, принятых из портов

Thread TaskHandlerThr = new Thread(TaskHandler);

TaskHandlerThr.IsBackground = true;

TaskHandlerThr.Start();

//Обработка заданий на отправку

Thread TaskToSendHandlerThr = new Thread(TaskToSendHandler);

TaskToSendHandlerThr.IsBackground = true;

TaskToSendHandlerThr.Start();

}

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^//

//При закрытии формы порты тоже закрываются

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

while (serialPort1.IsOpen || serialPort2.IsOpen)

{

if (serialPort1.IsOpen)

{

serialPort1.Close();

}

if (serialPort2.IsOpen)

{

serialPort2.Close();

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ВЫПОЛНЕНИЕ ЗАДАНИЙ НА ОТПРАВКУ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void TaskToSendHandler()

{

while (true)

{

//Получение задания из очереди на отправку кадра в порт

TaskToSend\_mutex.WaitOne();

if (TasksToSend.Count != 0)

{

One\_Task TaskToSend = TasksToSend[0];

TasksToSend.RemoveAt(0);

TaskToSend\_mutex.ReleaseMutex();

string PortName = TaskToSend.PortNum;

byte[] frametosend = TaskToSend.Frame;

if (PortName == "Port1")

{

try

{ serialPort1.Write(WIN1251.GetString(frametosend)); }

catch (Exception ex)

{ MessageBox.Show(ex.ToString(), "Error!"); }

}

else if (PortName == "Port2")

{

try

{ serialPort2.Write(WIN1251.GetString(frametosend)); }

catch (Exception ex)

{ MessageBox.Show(ex.ToString(), "Error!"); }

}

else { MessageBox.Show("TaskToSendHandler() Нет такого порта", "Error!"); }

}

else {

TaskToSend\_mutex.ReleaseMutex();

}

Thread.Sleep(20);

}

}

/\*

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\*/

// По кнопке начинает отсылать логины

public void Establish\_Logical()

{

//Получение локальногго логина

AuthData\_mutex.WaitOne();

string local\_auth\_name = AuthData["local"];

AuthData\_mutex.ReleaseMutex();

Ack1\_mutex\_Auth.WaitOne();

Ack1\_awaited\_Auth = 0;

Ack1\_mutex\_Auth.ReleaseMutex();

Ack2\_mutex\_Auth.WaitOne();

Ack2\_awaited\_Auth = 0;

Ack2\_mutex\_Auth.ReleaseMutex();

int ack1;

int ack2;

int counter = 0;

while (true)

{

Auth\_status\_mutex.WaitOne();

string s1, s2;

s1 = Auth\_status["ACK1"];

s2 = Auth\_status["ACK2"];

Auth\_status\_mutex.ReleaseMutex();

//Если ACK1 или ACK2 были получены то прерываем попытки отправлять логины

if (s1 != "undef" && s2 != "undef")

{

BeginInvoke(new SetTextDeleg(addtotextbox1),

new object[] { "Локальный логин был доставлен \r\n" });

Thread.Sleep(100);

return;

}

//Начальная попытка отправить LOGIN

if (counter == 0)

{

Ack1\_mutex\_Auth.WaitOne();

ack1 = Ack1\_awaited\_Auth;

Ack1\_mutex\_Auth.ReleaseMutex();

Ack2\_mutex\_Auth.WaitOne();

ack2 = Ack2\_awaited\_Auth;

Ack2\_mutex\_Auth.ReleaseMutex();

//Если это первая попытка отправки, то в порты отсылаются логины

if (ack1 == 0 && ack2 == 0)

{

TaskToSend\_mutex.WaitOne();

byte[] frame1 = CreateNewFrame(FrameType.LOGIN, "0",

(local\_auth\_name.Length).ToString(), "0", local\_auth\_name);

TasksToSend.Add(new One\_Task("Port1", frame1));

//Установка флага, что ожидается ack1

Ack1\_mutex\_Auth.WaitOne();

Ack1\_awaited\_Auth = 1;

Ack1\_mutex\_Auth.ReleaseMutex();

TaskToSend\_mutex.ReleaseMutex();

counter++;

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] {

" Попыток логин соединения (пк на порт1) : " + counter.ToString() + "\r\n" });

TaskToSend\_mutex.WaitOne();

byte[] frame2 = CreateNewFrame(FrameType.LOGIN, "0",

(local\_auth\_name.Length).ToString(), "0", local\_auth\_name);

TasksToSend.Add(new One\_Task("Port2", frame2));

//Установка флага, что ожидается ack2

Ack2\_mutex\_Auth.WaitOne();

Ack2\_awaited\_Auth = 1;

Ack2\_mutex\_Auth.ReleaseMutex();

TaskToSend\_mutex.ReleaseMutex();

counter++;

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { " Попыток логин соединения (пк на порт2) : " + counter.ToString() + "\r\n" });

Thread.Sleep(4000);

continue;

}

}

// Повторные попытки отправить логин

if (counter < 10 && counter != 0)

{

//Получение текущего статуса доставки

Ack1\_mutex\_Auth.WaitOne();

ack1 = Ack1\_awaited\_Auth;

Ack1\_mutex\_Auth.ReleaseMutex();

Ack2\_mutex\_Auth.WaitOne();

ack2 = Ack2\_awaited\_Auth;

Ack2\_mutex\_Auth.ReleaseMutex();

//Если ack 1&2 были получены

if (ack1 == 0 && ack2 == 0)

{

Auth\_status\_mutex.WaitOne();

Auth\_status["ACK1"] = "Received";

Auth\_status\_mutex.ReleaseMutex();

Auth\_status\_mutex.WaitOne();

Auth\_status["ACK2"] = "Received";

Auth\_status\_mutex.ReleaseMutex();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Переданные логины доставлены \r\n" });

return;

}

if (ack1 == 1)

{

TaskToSend\_mutex.WaitOne();

byte[] frame1 = CreateNewFrame(FrameType.LOGIN, "0", (local\_auth\_name.Length).ToString(), "0", local\_auth\_name);

TasksToSend.Add(new One\_Task("Port1", frame1));

Ack1\_mutex\_Auth.WaitOne();

Ack1\_awaited\_Auth = 1;

Ack1\_mutex\_Auth.ReleaseMutex();

TaskToSend\_mutex.ReleaseMutex();

counter++;

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { " Попыток передать логин (пк на порт1) : " + counter.ToString() + "\r\n" });

}

if (ack2 == 1)

{

TaskToSend\_mutex.WaitOne();

byte[] frame2 = CreateNewFrame(FrameType.LOGIN, "0", (local\_auth\_name.Length).ToString(), "0", local\_auth\_name);

TasksToSend.Add(new One\_Task("Port2", frame2));

Ack2\_mutex\_Auth.WaitOne();

Ack2\_awaited\_Auth = 1;

Ack2\_mutex\_Auth.ReleaseMutex();

TaskToSend\_mutex.ReleaseMutex();

counter++;

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { " Попыток передать логин (пк на порт2) : " + counter.ToString() + "\r\n" });

}

}

else if (counter != 0)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { " Передать логин не удалось. Попыток: " + counter.ToString() + "\r\n" });

return;

}

Thread.Sleep(4000);

}

}

private delegate void Set\_ButtonState(bool text);

public void Set\_AuthDisconnectButton(bool state)

{

this.AuthDisconnectButton.Enabled = state;

}

public void Set\_AuthConnectButton(bool state)

{

this.AuthConnectButton.Enabled = state;

}

// Поток, следящий за Авторизацией

// Если пользователи авторизованы в сети, тогда дается уведомление и заполняется список юзверей

public void Auth\_Tracker()

{

bool stat = false;

while (true)

{

Auth\_status\_mutex.WaitOne();

if (Auth\_status["ACK1"] != "undef" && Auth\_status["ACK\_local"] != "undef" && Auth\_status["ACK2"] != "undef")

{ stat = true; }

Auth\_status\_mutex.ReleaseMutex();

if (stat == true)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]{

"Авторизация всех пользователей прошла успешно\r\n" });

BeginInvoke(new FillComboBoxDeleg(FillReceiverComboBox), new object[] { });

BeginInvoke(new Set\_ButtonState(Set\_AuthConnectButton), new object[] { false });

BeginInvoke(new Set\_ButtonState(Set\_AuthDisconnectButton), new object[] { true });

return;

}

Thread.Sleep(200);

}

}

//Кнопка попытки авторизации

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

{

//Получение значений статуса физ подключения (берется из serial\_monitor)

Phys\_status1\_mutex.WaitOne();

Connection\_Status local\_status1 = Phys\_status1;

Phys\_status1\_mutex.ReleaseMutex();

Phys\_status2\_mutex.WaitOne();

Connection\_Status local\_status2 = Phys\_status2;

Phys\_status2\_mutex.ReleaseMutex();

if (local\_status1 != Connection\_Status.CONNECTED || local\_status2 != Connection\_Status.CONNECTED)

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Проверьте физическое соединение \r\n" });

return;

}

//Получение логина

string local\_login = LogintextBox.Text.Trim();

if (string.IsNullOrWhiteSpace(local\_login))

{

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Проверьте введенный логин \r\n" });

return;

}

else

{

AuthData\_mutex.WaitOne();

AuthData["local"] = local\_login;

AuthData\_mutex.ReleaseMutex();

}

//Запуск потока, следящего за ходом авторизации

Thread Auth\_Trackerthr = new Thread(Auth\_Tracker);

Auth\_Trackerthr.IsBackground = true;

Auth\_Trackerthr.Start();

//Попытка отправить свой логин

Thread connect\_logicalthr = new Thread(Establish\_Logical);

connect\_logicalthr.Start();

}

//Кнопка деавторизации, Disconnect

private void AuthDisconnectButton\_click(object sender, EventArgs e)

{

AuthData\_mutex.WaitOne();

AuthData["local"] = null;

AuthData["Port1"] = null;

AuthData["Port2"] = null;

AuthData\_mutex.ReleaseMutex();

Auth\_status\_mutex.WaitOne();

Auth\_status["ACK1"] = "undef";

Auth\_status["ACK2"] = "undef";

Auth\_status["ACK\_local"] = "undef";

Auth\_status\_mutex.ReleaseMutex();

TaskToSend\_mutex.WaitOne();

TasksToSend.Add(new One\_Task("Port1", CreateNewFrame(FrameType.LOGOUT, "0", null, "0", null, false)));

TasksToSend.Add(new One\_Task("Port2", CreateNewFrame(FrameType.LOGOUT, "0", null, "0", null, false)));

TaskToSend\_mutex.ReleaseMutex();

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[] { "Вы были деавторизованы\r\n" });

this.AuthDisconnectButton.Enabled = false;

this.AuthConnectButton.Enabled = true;

}

private void toolStripComboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

string selectedState = toolStripComboBox1.SelectedItem.ToString();

SelectedPort1Name\_mutex.WaitOne();

SelectedPort1Name = selectedState;

SelectedPort1Name\_mutex.ReleaseMutex();

}

private void toolStripComboBox3\_SelectedIndexChanged(object sender, EventArgs e)

{

string selectedState = toolStripComboBox3.SelectedItem.ToString();

SelectedBaudrate\_mutex.WaitOne();

SelectedBaudrate = selectedState;

SelectedBaudrate\_mutex.ReleaseMutex();

}

private void toolStripComboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

string selectedState = toolStripComboBox2.SelectedItem.ToString();

SelectedPort2Name\_mutex.WaitOne();

SelectedPort2Name = selectedState;

SelectedPort2Name\_mutex.ReleaseMutex();

}

//Закрытие портов

private void button4\_Click(object sender, EventArgs e)

{

if (serialPort1.IsOpen)

{

try

{

serialPort1.Handshake = Handshake.None;

Thread.Sleep(300);

serialPort1.Close();

if (DEBUG\_MODE) MessageBox.Show("Порт 1 был Закрыт", "Error!");

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]{

"Порт 1 был Закрыт\r\n" });

serialPort1.Handshake = Handshake.RequestToSend;

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString(), "Error!");

}

}

if (serialPort2.IsOpen)

{

try

{

serialPort2.Handshake = Handshake.None;

Thread.Sleep(300);

serialPort2.Close();

if (DEBUG\_MODE) MessageBox.Show("Порт 2 был Закрыт", "Error!");

BeginInvoke(new SetTextDeleg(addtotextbox1), new object[]{

"Порт 2 был Закрыт\r\n" });

serialPort2.Handshake = Handshake.RequestToSend;

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString(), "Error!");

}

}

if (!serialPort1.IsOpen && !serialPort1.IsOpen)

{

this.button1.Enabled = true;

this.button4.Enabled = false;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ОТКРЫТИЕ ПАПОК С ПИСЬМАМИ

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Открытие папки входящие

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

{

Form2 inbox\_folder = new Form2(this);

inbox\_folder.Show();

}

//Инициация обновления формы inbox

private void button2\_Click(object sender, EventArgs e)

{

Inbox\_update\_mutex.WaitOne();

Inbox\_update\_needed = true;

Inbox\_update\_mutex.ReleaseMutex();

}

//Открытие папки исходящие

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

{

Form3 outbox\_folder = new Form3(this);

outbox\_folder.Show();

}

//Инициация обновления формы outbox

private void button5\_Click(object sender, EventArgs e)

{

Outbox\_update\_mutex.WaitOne();

Outbox\_update\_needed = true;

Outbox\_update\_mutex.ReleaseMutex();

}

private void toolStripMenuItem2\_Click(object sender, EventArgs e)

{

MessageBox.Show(

"Работу выполнили:\r\n" +

" Пупчин П.Н.\r\n" +

" Пирмамедов М.Э.\r\n" +

" Сметанкин К.И.", "Справка");

}

}

}

## 5.2. Form2

namespace main\_application

{

public partial class Form2 : Form

{

public Form1 form1;

public bool UpdateThread\_to\_close = false;

public Form2(Form1 form)

{

UpdateThread\_to\_close = false;

this.form1 = form;

InitializeComponent();

Thread LookForInboxUpdatesthr = new Thread(LookForInboxUpdates);

LookForInboxUpdatesthr.IsBackground = true;

LookForInboxUpdatesthr.Start();

}

public void Show\_Inbox() {

using (CourseDB db = new CourseDB())

{

dataGridView1.DataSource = db.inbox.ToList();

}

dataGridView1.Columns[4].Visible = false;

dataGridView1.Columns[6].Visible = false;

dataGridView1.Columns[0].Width = 20;

dataGridView1.Columns[1].Width = 50;

dataGridView1.Columns[2].Width = 50;

dataGridView1.Columns[3].Width = 80;

dataGridView1.Columns[5].Width = 90;

dataGridView1.Columns[7].Width = 70;

foreach (DataGridViewRow row in dataGridView1.Rows)

{

if (row.Cells[5].Value.ToString() == "Принято")

{

row.DefaultCellStyle.BackColor = Color.AliceBlue;

}

if (row.Cells[5].Value.ToString() == "Прочитано")

{

row.DefaultCellStyle.BackColor = Color.LightCyan;

}

}

dataGridView1.Update();

dataGridView1.Refresh();

}

private void Form2\_Load(object sender, EventArgs e)

{

Show\_Inbox();

}

private delegate void SetTextDeleg(string text);

public void setlabel4state(string text)

{

label4.Text = text;

}

private delegate void UpdateDataGridDeleg(List<inbox> list);

public void UpdateDataGrid1(List<inbox> list)

{

dataGridView1.DataSource = list;

foreach (DataGridViewRow row in dataGridView1.Rows)

{

if (row.Cells[5].Value.ToString() == "Принято")

{

row.DefaultCellStyle.BackColor = Color.AliceBlue;

}

if (row.Cells[5].Value.ToString() == "Прочитано")

{

row.DefaultCellStyle.BackColor = Color.LightCyan ;

}

}

dataGridView1.Update();

dataGridView1.Refresh();

}

public void LookForInboxUpdates(){

while(true){

form1.Inbox\_update\_mutex.WaitOne();

if (form1.Inbox\_update\_needed)

{

form1.Inbox\_update\_needed = false;

form1.Inbox\_update\_mutex.ReleaseMutex();

BeginInvoke(new SetTextDeleg(setlabel4state), new object[] { "обновляется" });

using (CourseDB db = new CourseDB())

{

BeginInvoke(new UpdateDataGridDeleg(UpdateDataGrid1), new object[] { db.inbox.ToList() });

}

}

else

{

form1.Inbox\_update\_mutex.ReleaseMutex();

}

Thread.Sleep(1000);

BeginInvoke(new SetTextDeleg(setlabel4state), new object[] { "ожидание" });

if (UpdateThread\_to\_close)

{

break;

}

}

}

//Обработка клика на письмо

private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

//gets a collection that contains all the rows

DataGridViewRow row = this.dataGridView1.Rows[e.RowIndex];

//Отображение письма в текстБоксы

try { textBox1.Text = row.Cells[1].Value.ToString(); }

catch (Exception ex) { textBox1.Text = ""; }

try { textBox2.Text = row.Cells[3].Value.ToString(); }

catch (Exception ex) { textBox2.Text = ""; }

try { textBox3.Text = row.Cells[4].Value.ToString(); }

catch (Exception ex) { textBox3.Text = ""; }

try { textBox4.Text = row.Cells[7].Value.ToString();}

catch (Exception ex){textBox4.Text = "";}

try

{ textBox5.Text = row.Cells[5].Value.ToString(); }

catch (Exception ex) { textBox5.Text = ""; }

//Попытка отправить уведомление о прочтении

bool foreign\_id\_exists = false;

string foreign\_id\_string="";

string sender\_string = "";

string status\_string = "";

try {

foreign\_id\_string = row.Cells[6].Value.ToString();

sender\_string = row.Cells[1].Value.ToString();

status\_string = row.Cells[5].Value.ToString();

foreign\_id\_exists = true;

}

catch(Exception ex)

{

foreign\_id\_exists = false;

}

if (foreign\_id\_exists)

{

if (form1.AuthData["Port1"] != null && form1.AuthData["Port2"] != null && form1.AuthData["local"] != null)

{

if (status\_string != "Прочитано")

{

int len = foreign\_id\_string.Length;

// Здесь ожидаем получения мьютекса на создание заданий

// После получения, используя методы Form1 создается кадр об открытии письма

// Необходимо получить значение "Port1/2" по ключу "sender",

// а также номер машины, за которой сидит этот sender

var Receiver\_Port = form1.AuthData.FirstOrDefault(x => x.Value == sender\_string).Key;

//either "Port1" or "Port2"

string Port = Receiver\_Port.ToString();

//Конвертирование из utf-8 в win1251 уже предусмотрено в функции

byte[] frame = form1.CreateNewFrame(Form1.FrameType.OPENLETTER, "0", len.ToString(), "0", foreign\_id\_string, false);

Form1.One\_Task openletterframe = new Form1.One\_Task();

form1.TaskToSend\_mutex.WaitOne();

form1.TasksToSend.Add(new Form1.One\_Task(Receiver\_Port, frame));

form1.TaskToSend\_mutex.ReleaseMutex();

using (CourseDB db = new CourseDB())

{

var foreign\_id\_num = long.Parse(foreign\_id\_string);

var result = db.inbox.SingleOrDefault(x => x.foreign\_id == foreign\_id\_num);

if (result != null)

{

result.status = "Прочитано";

db.SaveChanges();

form1.Inbox\_update\_mutex.WaitOne();

form1.Inbox\_update\_needed = true;

form1.Inbox\_update\_mutex.ReleaseMutex();

}

}

}

else

{

}

}

}

}

}

private void Form2\_FormClosing(object sender, FormClosingEventArgs e)

{

UpdateThread\_to\_close = true;

Thread.Sleep(2500);

}

}

}

## 5.3. Form3

namespace main\_application

{

public partial class Form3 : Form

{

public Form1 form1;

public bool UpdateThread\_to\_close = false;

public Form3(Form1 form)

{

UpdateThread\_to\_close = false;

this.form1 = form;

InitializeComponent();

Thread LookForOutboxUpdatesthr = new Thread(LookForOutboxUpdates);

LookForOutboxUpdatesthr.IsBackground = true;

LookForOutboxUpdatesthr.Start();

}

public void Show\_Outbox()

{

using (CourseDB db = new CourseDB())

{

dataGridView1.DataSource = db.outbox.ToList();

}

dataGridView1.Columns[4].Visible = false;

dataGridView1.Columns[6].Visible = false;

dataGridView1.Columns[0].Width = 20;

dataGridView1.Columns[1].Width = 50;

dataGridView1.Columns[2].Width = 50;

dataGridView1.Columns[3].Width = 80;

dataGridView1.Columns[5].Width = 190;

dataGridView1.Columns[7].Width = 90;

foreach (DataGridViewRow row in dataGridView1.Rows)

{

if (row.Cells[5].Value.ToString() == "Принято")

{

row.DefaultCellStyle.BackColor = Color.AliceBlue;

}

if (row.Cells[5].Value.ToString() == "Прочитано")

{

row.DefaultCellStyle.BackColor = Color.Azure;

}

}

dataGridView1.Update();

dataGridView1.Refresh();

}

private delegate void SetTextDeleg(string text);

public void setlabel4state(string text)

{

label5.Text = text;

}

private delegate void UpdateDataGridDeleg(List<outbox> list);

public void UpdateDataGrid1(List<outbox> list)

{

dataGridView1.DataSource = list;

foreach (DataGridViewRow row in dataGridView1.Rows)

{

if (row.Cells[5].Value.ToString() == "Принято")

{

row.DefaultCellStyle.BackColor = Color.AliceBlue;

}

if (row.Cells[5].Value.ToString() == "Прочитано")

{

row.DefaultCellStyle.BackColor = Color.Azure;

}

}

dataGridView1.Update();

dataGridView1.Refresh();

}

public void LookForOutboxUpdates()

{

while (true)

{

form1.Outbox\_update\_mutex.WaitOne();

if (form1.Outbox\_update\_needed)

{

form1.Outbox\_update\_needed = false;

form1.Outbox\_update\_mutex.ReleaseMutex();

BeginInvoke(new SetTextDeleg(setlabel4state), new object[] { "обновляется" });

using (CourseDB db = new CourseDB())

{

BeginInvoke(new UpdateDataGridDeleg(UpdateDataGrid1), new object[] { db.outbox.ToList() });

}

}

else

{

form1.Outbox\_update\_mutex.ReleaseMutex();

}

Thread.Sleep(1000);

BeginInvoke(new SetTextDeleg(setlabel4state), new object[] { "ожидание" });

if (UpdateThread\_to\_close)

{

break;

}

}

}

//Обработка клика на письмо

private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

//gets a collection that contains all the rows

DataGridViewRow row = this.dataGridView1.Rows[e.RowIndex];

//Отображение письма в текстБоксы

try { textBox1.Text = row.Cells[2].Value.ToString(); }

catch (Exception ex) { textBox1.Text = ""; }

try { textBox2.Text = row.Cells[3].Value.ToString(); }

catch (Exception ex) { textBox2.Text = ""; }

try { textBox3.Text = row.Cells[4].Value.ToString(); }

catch (Exception ex) { textBox3.Text = ""; }

try { textBox4.Text = row.Cells[7].Value.ToString(); }

catch (Exception ex) { textBox4.Text = ""; }

try

{ textBox5.Text = row.Cells[5].Value.ToString(); }

catch (Exception ex) { textBox5.Text = ""; }

}

}

private void Form3\_Load\_1(object sender, EventArgs e)

{

Show\_Outbox();

}

private void Form3\_FormClosing\_1(object sender, FormClosingEventArgs e)

{

UpdateThread\_to\_close = true;

Thread.Sleep(2500);

}

}

}