

Утверждаю

Лист утверждений

_____ Галкин В.А.

"__" _____ 2020г.

Описание применения

"Локальная безадаптерная сеть"

по курсу "Сетевые технологии в АСОИУ"

Вариант №____

Исполнители:

_____ Матюнин Р.А. гр. ИУ5-61Б

_____ Мелисов А.Е. гр. ИУ5-61Б

_____ Бакланов Н.В. гр. ИУ5-61Б

Москва 2020 г.

Оглавление

1. Введение	3
2. Класс форм	3
2.1. Переменные.....	3
2.2. События	3
2.3. Методы	4
3. Листинг	4

1. Введение

Программный продукт написан с использованием технологии .Net Framework версии 4.8 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 форм

2.1. Переменные

- private SerialComPort serialcomport – объект, описывающий COM-порт;
- private Timer receivedDataTimer – объект, описывающий таймер при приеме данных с COM-порта;
- private Timer replayFileTimer – объект, описывающий таймер при приеме данных;
- private string receivedData - объект, описывающий принятые данные;
- private bool dataReady – объект, описывающий готовность приема данных;
- private StreamReader file – объект описывающий поток данных;

2.2. События

- private void ReceiveDataHandler(string data)– событие, приема передающих через COM-порт данных.
- private void ReceivedDataTimerTick(object sender, EventArgs e)– событие, отсчета времени при приеме данных:
 - object sender – объект, вызывающий событие;
 - EventArgs e – аргументы для события;
- private void ReplayFileTimerTick(object sender, EventArgs e)– событие таймера, возникающее при воспроизведении файла:
 - object sender – объект, вызывающий событие;
 - EventArgs e – аргументы для события;
- private void UpdateDataWindow(string message)– событие, обновления данных на экране.
- private void UpdateWindow(string message)- событие, обновления данных на экране.
- private void SendFileButton(object sender, EventArgs e) - событие, возникающее при нажатии на кнопку «Передать»:
 - object sender – объект, вызывающий событие;
 - EventArgs e – аргументы для события.
- private void ConnectionButton(object sender, EventArgs e)) - событие, возникающее при нажатии на кнопку «Подключиться»:
 - object sender – объект, вызывающий событие;
 - EventArgs e – аргументы для события.
- private void BrowseButton(object sender, EventArgs e) - событие, возникающее при нажатии на кнопку «Путь»:
 - object sender – объект, вызывающий событие;
 - EventArgs e – аргументы для события.
- private void AddInfoToFileButton(object sender, EventArgs e) - событие, возникающее при нажатии на кнопку «Добавить»:
 - object sender – объект, вызывающий событие;

- EventArgs e – аргументы для события.

2.3. Методы

- Close() – закрывает соединение порта, присваивает свойству IsOpen значение false и уничтожает внутренний объект Stream
- Dispose() - Освобождает все ресурсы
- GetPortNames() - Получает массив имен последовательных портов для текущего компьютера.
- Open() - Открывает новое соединение последовательного порта.
- Read(Byte[], Int32, Int32) - Считывает из входного буфера SerialPort определенное число байтов и записывает их в байтовый массив, начиная с указанной позиции.
- Read(Char[], Int32, Int32) - Считывает из входного буфера SerialPort определенное число символов и записывает их в символьный массив, начиная с указанной позиции.
- ReadByte() - Считывает из входного буфера SerialPort один байт в синхронном режиме.
- ReadExisting() - Считывает все непосредственно доступные байты в соответствии с кодировкой из потока и из входного буфера объекта SerialPort.
- ReadLine() - Считывает данные из входного буфера до значения NewLine.
- ReadTo(String) - Считывает из входного буфера строку до указанного значения value.
- ToString() - Возвращает объект String.
- Write(Byte[], Int32, Int32) - Записывает указанное число байтов в последовательный порт, используя данные из буфера.
- Write(Char[], Int32, Int32) - Записывает указанное число символов в последовательный порт, используя данные из буфера.
- WriteLine(String) - Записывает указанную строку и значение NewLine в выходной буфер.

3. Листинг

3.1. Form.cs

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

namespace COMPDT
{
    public partial class Form : System.Windows.Forms.Form
```

```

{

//private void SerialPortConnect(String port, int baudrate, Parity parity, int databits, StopBits stopbits)
//{
//    DateTime dt = DateTime.Now;
//    String dtn = dt.ToShortTimeString();

//    serialport = new System.IO.Ports.SerialPort(port, baudrate, parity, databits, stopbits);
//    try
//    {
//        serialport.Open();
//        Disconnection.Enabled = true;
//        Send.Enabled = true;
//        Connection.Enabled = false;
//        Messeges.AppendText(" [" + dtn + "] " + "Подключен\n");
//        serialport.DataReceived += new SerialDataReceivedEventHandler(SerialPortDataReceived);
//    }
//    catch (Exception ex) { MessageBox.Show(ex.ToString(), "Ошибка"); }
//}
//private void SerialPortDataReceived(object sender, SerialDataReceivedEventArgs e)
//{
//    DateTime dt = DateTime.Now;
//    String dtn = dt.ToShortTimeString();

//    Messeges.AppendText(" [" + dtn + "] " + "Получено: " + serialport.ReadExisting() + "\n");
//}
//private System.IO.Ports.SerialPort serialport;
private SerialComPort serialcomport;
private Timer receivedDataTimer;
private Timer replayFileTimer;
private string receivedData;
private bool dataReady = false;
private StreamReader file;
private void Form_Load(object sender, EventArgs e)
{

}

public Form()
{
    InitializeComponent();
    //Disconnection.Enabled = false;
    SendFile.Enabled = false;
    file = null;
    serialcomport = new SerialComPort();
    serialcomport.RegisterReceiveCallback(ReceiveDataHandler);
    receivedDataTimer = new Timer();
    receivedDataTimer.Interval = 25; // 25 ms
    receivedDataTimer.Tick += new EventHandler(ReceivedDataTimerTick);
    receivedDataTimer.Start();
    replayFileTimer = new Timer();
    replayFileTimer.Interval = 1000; // 1000 ms
    replayFileTimer.Tick += new EventHandler(ReplayFileTimerTick);
    replayFileTimer.Start();
}
private void ReceiveDataHandler(string data)
{
    if (dataReady)

```

```

{
    Debug.Print("Полученные данные были отброшены, потому что буфер строки не очищен");
}
else
{

    dataReady = true;
    receivedData = data;
}
}
private void ReceivedDataTimerTick(object sender, EventArgs e)
{
    string path = textBoxPath.Text + "2.txt";

    if (dataReady)
    {
        string indata = string.Empty;

        Messages.Clear();
        StringBuilder sb = new System.Text.StringBuilder();
        string[] binaryArr1 = new string[sb.Length];
        string[] binaryArr2 = new string[sb.Length];
        string[] residueArr = new string[binaryArr1.Length];
        string[] residueArr2 = new string[binaryArr1.Length];

        string binaryStr = null;
        foreach (byte b in System.Text.Encoding.UTF8.GetBytes(indata))
            for (int k = 0; k < sb.Length; k++)
            {
                sb.Append(Convert.ToString(b, 2).PadLeft(11, '0').PadRight(15, '0'))/*.Append(' ')*/;
                binaryArr1[k] = sb.ToString();
                sb.Append(Convert.ToString(b, 2).PadLeft(11, '0'))/*.Append(' ')*/;
                binaryArr2[k] = sb.ToString();
                //string binaryStr = sb.ToString();
            }
        for (int i = 0; i < binaryArr1.Length; i++)
        {
            int binaryInt = Convert.ToInt32(binaryArr1[i], 2);
            int residue = binaryInt % 19/*10011*/;
            residueArr[i] = Convert.ToString(residue, 2);
            binaryArr1[i] = binaryArr2[i] + residueArr[i] + ' ';
            binaryStr += binaryArr1[i];
        }
        char[] separators = new char[] { ' ' };
        var wordsArray = binaryStr.Split(separators, StringSplitOptions.RemoveEmptyEntries);
        string wordsStr = null;
        string wordsStr2 = null;
        string[] arrayCode = new string[binaryArr1.Length];
        for (int o = 0; o < wordsArray.Length; o++)
        {
            int wordsInt = Convert.ToInt32(wordsArray[o], 2);
            int residue = wordsInt % 19/*10011*/;
            wordsStr = wordsArray[o];
            StringBuilder sb2 = new StringBuilder(wordsStr);
            if (residue != 0) { if (sb2[residue - 2] == 1) { sb2[residue - 2] = '0'; } else { sb2[residue - 2] =
'1'; } }

            wordsStr2 = sb2.ToString();

```

```

        arrayCode[o] = wordsStr2;
    }
    var bytes = arrayCode.Select(s => Convert.ToByte(s, 16));
    var word = Encoding.UTF8.GetString(bytes.ToArray());
    dataReady = false;
    UpdateDataWindow(receivedData);
    Messages.Clear();
    UpdateDataWindow("Данные приняты...");
    StellsBox.Clear();
    UpdateWindow(receivedData);
    using (FileStream file = new FileStream(path, FileMode.Append))
    using (StreamWriter sw = new StreamWriter(file))
        sw.WriteLine(StellsBox.Text);
    }
}
private void ReplayFileTimerTick(object sender, EventArgs e)
{
    if (file != null)
    {
        try
        {
            string message = file.ReadLine();
            if (!file.EndOfStream)
            {
                serialcomport.SendLine(message + "\n");
            }
            else
            {
                file.BaseStream.Seek(0, 0); // start over reading the file
            }
        }
        catch (Exception error)
        {
            Debug.Print(error.Message);
        }
    }
}
private void UpdateDataWindow(string message)
{
    Messages.Text += message;
    Messages.SelectionStart = Messages.TextLength;
    Messages.ScrollToCaret();
}
private void UpdateWindow(string message)
{
    StellsBox.Text += message;
    StellsBox.SelectionStart = StellsBox.TextLength;
    StellsBox.ScrollToCaret();
}
private void UpdateReportWindow(string message)
{
    TransferReport.Text += message;
    TransferReport.SelectionStart = TransferReport.TextLength;
    TransferReport.ScrollToCaret();
}
private void SendMessageButton(object sender, EventArgs e)
{

```

```

DateTime dt = DateTime.Now;
String dtn = dt.ToShortTimeString();

if (serialcomport.IsOpen())
{
    string message = "[" + dtn + "]" + comboBoxPort.Text + ":" + MessageToSend.Text + "\r\n";
    serialcomport.SendLine(message);
    UpdateDataWindow(message);
}
else
{
    UpdateDataWindow "[" + dtn + "]" + "Откройте свой порт\r\n");
}
}
private void SendFileButton(object sender, EventArgs e)
{
    Messages.Clear();
    DateTime dt = DateTime.Now;
    String dtn = dt.ToShortTimeString();

    if (!serialcomport.IsOpen())
    {
        UpdateDataWindow "[" + dtn + "]" + "Откройте свой порт\r\n");
        return;
    }
    if (SendFile.Text == "Передать")
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        DialogResult result = openFileDialog.ShowDialog();
        if (result == DialogResult.OK)
        {
            file = new System.IO.StreamReader(openFileDialog.FileName);
            SendFile.Text = "Остановить передачу";
            UpdateDataWindow("Передача через COM порт: " + openFileDialog.FileName + "\r\n");
        }
    }
    else
    {
        {
            if (file != null)
            {
                file.Close();
                file = null;
                SendFile.Text = "Передать";
                this.timer1.Dispose();
            }
        }
    }
}
private void ConnectionButton(object sender, EventArgs e)
{
    DateTime dt = DateTime.Now;
    String dtn = dt.ToShortTimeString();

    if (comboBoxPort.Text == "" || comboBoxBaudRate.Text == "")
    { Messages.Text = "[" + dtn + "]" + "Пожалуйста заполните настройки порта\n"; }
    else

```



```

{
    // Handles the Open/Close button, which toggles its label, depending on previous state.
    string status;
    if (Connection.Text == "Подключиться")
    {
        status = serialcomport.Open(comboBoxPort.Text, comboBoxBaudRate.Text, "8", "None",
"One");
        if (status.Contains("Открыт"))
        {
            Connection.Text = "Отключиться";
            SendFile.Enabled = true;
        }
        else
        {
            status = serialcomport.Close();
            Connection.Text = "Подключиться";
        }
        UpdateDataWindow(status);
    }
}
private void BrowseButton(object sender, EventArgs e)
{
    using (FolderBrowserDialog fbd = new FolderBrowserDialog() { Description = "Выберите путь,
где хотите сохранить свои файлы:" })
    {
        if (fbd.ShowDialog() == DialogResult.OK)
        {
            FileInfo.Text = "Файлы будут сохранены по пути " + fbd.SelectedPath + "\n";
            textBoxPath.Text = fbd.SelectedPath + "\\";
        }
    }
}
private void AddInfoToFileButton(object sender, EventArgs e)
{
    DateTime dt = DateTime.Now;
    String dtn = dt.ToShortTimeString();

    string path = textBoxPath.Text + textBoxName.Text + ".txt";

    using (FileStream file = new FileStream(path, FileMode.Append))
        using (StreamWriter sw = new StreamWriter(file))
            sw.WriteLine(textBoxMessage.Text);

    using (StreamReader sr = new StreamReader(path, true))
    {
        FileInfo.Clear();
        FileInfo.AppendText("[" + dtn + "] " + textBoxName.Text + ".txt изменен \n");
        FileInfo.AppendText("Содержание:\n" + sr.ReadToEnd());
    }
}
}

```

3.2. Serial_Rx_Tx.cs

```

using System;
using System.IO.Ports;
using System.Diagnostics;

```

```

using System.Text;

namespace SERIAL_RX_TX
{
    public class SerialComPort
    {
        private SerialPort comPort;

        // constructor
        public SerialComPort()
        {
            comPort = new SerialPort();
        }

        ~SerialComPort()
        {
            Close();
        }

        // User must register function to call when a line of text terminated by \n has been received
        public delegate void ReceiveCallback(string receivedMessage);
        public event ReceiveCallback onMessageReceived = null;
        public void RegisterReceiveCallback(ReceiveCallback FunctionToCall)
        {
            onMessageReceived += FunctionToCall;
        }
        public void DeRegisterReceiveCallback(ReceiveCallback FunctionToCall)
        {
            onMessageReceived -= FunctionToCall;
        }

        public void SendLine(string aString)
        {
            try
            {
                if (comPort.IsOpen)
                {
                    comPort.Write(aString);
                }
            }
            catch (Exception exp)
            {
                Debug.Print(exp.Message);
            }
        }

        public string Open(string portName, string baudRate, string dataBits, string parity, string stopBits)
        {
            DateTime dt = DateTime.Now;
            String dtn = dt.ToShortTimeString();

            try
            {
                comPort.WriteBufferSize = 4096;
                comPort.ReadBufferSize = 4096;
                comPort.WriteTimeout = 500;
            }
            catch (Exception exp)
            {
                Debug.Print(exp.Message);
            }
        }
    }
}

```

```

comPort.ReadTimeout = 500;
comPort.DtrEnable = true;
comPort.Handshake = Handshake.None;
comPort.PortName = portName.TrimEnd();
comPort.BaudRate = Convert.ToInt32(baudRate);
comPort.DataBits = Convert.ToInt32(dataBits);

switch (parity)
{
    case "None":
        comPort.Parity = Parity.None;
        break;
    case "Even":
        comPort.Parity = Parity.Even;
        break;
    case "Odd":
        comPort.Parity = Parity.Odd;
        break;
}
switch (stopBits)
{
    case "One":
        comPort.StopBits = StopBits.One;
        break;
    case "Two":
        comPort.StopBits = StopBits.Two;
        break;
}
comPort.Open();
comPort.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler);
}
catch (Exception error)
{
    return error.Message + "\r\n";
}
if (comPort.IsOpen)
{
    return string.Format("[" + dtn + "] " + "{0} Открыт \r\n", comPort.PortName);
}
else
{
    return string.Format("[" + dtn + "] " + "{0} Произошла ошибка \r\n", comPort.PortName);
}
}

public string Close()
{
    DateTime dt = DateTime.Now;
    String dtn = dt.ToShortTimeString();

    try
    {
        comPort.Close();
    }
    catch (Exception error)
    {
        return error.Message + "\r\n";
    }
}

```

```

    }
    return string.Format(" [" + dtn + "] " + "{0} Закрыт\r\n", comPort.PortName);
}

public bool IsOpen()
{
    return comPort.IsOpen;
}

private void DataReceivedHandler(object sender, SerialDataReceivedEventArgs e)
{
    if (!comPort.IsOpen)
    {
        return;
    }
    string indata = string.Empty;

    try
    {
        indata = comPort.ReadLine();

        StringBuilder sb = new System.Text.StringBuilder();
        string[] binaryArr1 = new string[sb.Length];
        string[] binaryArr2 = new string[sb.Length];
        string[] residueArr = new string[binaryArr1.Length];
        foreach (byte b in System.Text.Encoding.UTF8.GetBytes(indata))
            for (int k=0; k < sb.Length; k++)
            {
                sb.Append(Convert.ToString(b, 2).PadLeft(11, '0').PadRight(15, '0'))/*.Append(' ')*/;
                binaryArr1[k]= sb.ToString();
                sb.Append(Convert.ToString(b, 2).PadLeft(11, '0'))/*.Append(' ')*/;
                binaryArr2[k] = sb.ToString();
                //string binaryStr = sb.ToString();
            }
        for (int i = 0; i < binaryArr1.Length; i++)
        {
            int binaryInt = Convert.ToInt32(binaryArr1[i], 2);
            int residue = binaryInt % 19/*10011*/;
            residueArr[i] = Convert.ToString(residue, 2);
            binaryArr1[i] = binaryArr2[i] + residueArr[i] + ' ';
        }
        if (onMessageReceived != null)
        {
            onMessageReceived(indata);
        }
    }
    catch (Exception error)
    {
        Debug.Print(error.Message);
    }
}
}

```