**Лабораторна робота №2**

Виконав студент групи КН-21-1

Ільїчов Д. С.

**Мета роботи:** Розробити програму для емуляції дисплейного модуля.

**Етапи виконання лабораторної роботи:**

1. Розробити архітектуру і реалізувати програму з графічним інтерфейсом здатну виконувати команди, наведені в лабораторній роботі №1. Обмеження на тип даних і параметри дивися в описі команд в лабораторній роботі №1.

2. Додати в програму, розроблену в п.1 код для реалізації UDP сервера з лабораторної роботи №1. При спільному використанні як графічного інтерфейсу так і роботи з мережею можливо Вам знадобиться використання додаткових потоків виконання (threads).

3. Після проведення інтеграції(п.2) програма повинна мати можливість приймати команди, описані в лабораторній роботі №1, по протоколу UDP і відображати їх в графічному інтерфейсі.

*Хід роботи*

**Лістинг коду:**

Код для роботи серверу:

Form1.cs

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Drawing.Drawing2D;

namespace CSCS2\_Forms

{

public partial class Form1 : Form

{

static Int16 rotation = 0;

static Int16 penWidth = 2;

static List<Lines> lines = new List<Lines>();

static List<Pixels> pixels = new List<Pixels>();

static List<Rectangles> rectangles = new List<Rectangles>();

static List<Ellipses> ellipses = new List<Ellipses>();

static List<RoundedRectangle> roundedRectangles = new

List<RoundedRectangle>();

static List<Texts> texts = new List<Texts>();

static List<Pictures> pictures = new List<Pictures>(); public Form1()

{

InitializeComponent();

try

{

new Thread(new ThreadStart(ReceiveMessage)).Start();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLASSES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class Pixels

{

public Int16 x0;

public Int16 y0;

public Color argb;

public Pixels(Int16 \_x0, Int16 \_y0, Color \_argb)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.argb = \_argb;

}

}

public class Lines

{

public Int16 x0;

public Int16 y0;

public Int16 x1;

public Int16 y1;

public Color argb;

public Lines(Int16 \_x0, Int16 \_y0, Int16 \_x1, Int16 \_y1, Color \_argb)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.x1 = \_x1;

this.y1 = \_y1;

this.argb = \_argb;

}

}

public class Rectangles

{

public Int16 x0;

public Int16 y0;

public Int16 w;

public Int16 h;

public Color argb;

public bool isfilled;

public Rectangles(Int16 \_x0, Int16 \_y0, Int16 \_w, Int16 \_h, Color

\_argb, bool \_isfilled)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.w = \_w;

this.h = \_h;

this.argb = \_argb;

this.isfilled = \_isfilled;

}

}

public class Ellipses

{

public Int16 x0;

public Int16 y0;

public Int16 radius\_x; public Int16 radius\_y;

public Color argb;

public bool isfilled;

public Ellipses(Int16 \_x0, Int16 \_y0, Int16 \_radius\_x, Int16

\_radius\_y, Color \_argb, bool \_isfilled)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.radius\_x = \_radius\_x;

this.radius\_y = \_radius\_y;

this.argb = \_argb;

this.isfilled = \_isfilled;

}

}

public class RoundedRectangle

{

public Int16 x0;

public Int16 y0;

public Int16 w;

public Int16 h;

public Int16 radius;

public Color argb;

public bool isfilled;

public RoundedRectangle(Int16 \_x0, Int16 \_y0, Int16 \_w, Int16 \_h,

Int16 \_radius, Color \_argb, bool \_isfilled)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.w = \_w;

this.h = \_h;

this.radius = \_radius;

this.argb = \_argb;

this.isfilled = \_isfilled;

}

}

public class Texts

{

public Int16 x0;

public Int16 y0;

public Color argb;

public Int16 fontSize;

public string text;

public Texts(Int16 \_x0, Int16 \_y0, Color \_argb, Int16 \_fontSize,

string \_text)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.argb = \_argb;

this.fontSize = \_fontSize;

this.text = \_text;

}

}

/\*public class TextLines : Texts

{

public List<int[,]> symbols = new List<int[,]>();

public TextLines(Int16 \_x0, Int16 \_y0, Color \_argb, Int16 \_fontSize,

string \_text)

: base(\_x0, \_y0, \_argb, \_fontSize, \_text)

{

Chars s = new Chars();

short x = x0;

foreach (var symbol in text)

{

symbols.Add(s.GetCharCoords(symbol, x, y0, fontSize));

x = Convert.ToInt16(x + fontSize \* 0.8);

}

}

}\*/

public class Pictures

{

public Int16 x0;

public Int16 y0;

public Int16 w;

public Int16 h;

public Color[,] argb;

public Pictures(Int16 \_x0, Int16 \_y0, Int16 \_w, Int16 \_h, Color[,]

\_argb)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.w = \_w;

this.h = \_h;

this.argb = \_argb;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RENDERING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

protected override void OnPaint(PaintEventArgs e)

{

Graphics graphics = e.Graphics;

graphics.SmoothingMode = SmoothingMode.HighQuality;

graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;

graphics.TranslateTransform(this.Width / 2, this.Height / 2);

graphics.RotateTransform(rotation);

graphics.TranslateTransform(-this.Width / 2, -this.Height / 2);

foreach (var pixel in pixels.ToArray())

{

graphics.FillRectangle(new SolidBrush(pixel.argb), pixel.x0 +

this.Width / 2, pixel.y0 + this.Height / 2, 1, 1);

}

foreach (var line in lines.ToList())

{

graphics.DrawLine(new Pen(line.argb, penWidth), line.x0 +

this.Width / 2, line.y0 + this.Height / 2, line.x1 + this.Width / 2, line.y1 +

this.Height / 2);

}

foreach (var rectangle in rectangles.ToList())

{

if (rectangle.isfilled)

{

graphics.FillRectangle(new SolidBrush(rectangle.argb),

rectangle.x0 + this.Width / 2 - rectangle.w / 2, rectangle.y0 + this.Height / 2 -

rectangle.h / 2, rectangle.w, rectangle.h);

}

else

{

graphics.DrawRectangle(new Pen(rectangle.argb, penWidth),

rectangle.x0 + this.Width / 2 - rectangle.w / 2, rectangle.y0 + this.Height / 2 -

rectangle.h / 2, rectangle.w, rectangle.h);

}

}

foreach (var ellipse in ellipses.ToList())

{

if (ellipse.isfilled)

{

graphics.FillEllipse(new SolidBrush(ellipse.argb), ellipse.x0

+ this.Width / 2 - ellipse.radius\_x / 2, ellipse.y0 + this.Height / 2 -

ellipse.radius\_y / 2, ellipse.radius\_x, ellipse.radius\_y);

}

else

{

graphics.DrawEllipse(new Pen(ellipse.argb, penWidth),

ellipse.x0 + this.Width / 2 - ellipse.radius\_x / 2, ellipse.y0 + this.Height / 2 -

ellipse.radius\_y / 2, ellipse.radius\_x, ellipse.radius\_y);

}

}

foreach (var roundedRectangle in roundedRectangles.ToList())

{

if (roundedRectangle.isfilled)

{

graphics.FillPath(new SolidBrush(roundedRectangle.argb),

RoundedRect(new Rectangle(roundedRectangle.x0 + this.Width / 2 -

roundedRectangle.w / 2, roundedRectangle.y0 + this.Height / 2 - roundedRectangle.h

/ 2, roundedRectangle.w, roundedRectangle.h), roundedRectangle.radius));

}

else

{

graphics.DrawPath(new Pen(roundedRectangle.argb, penWidth),

RoundedRect(new Rectangle(roundedRectangle.x0 + this.Width / 2 -

roundedRectangle.w / 2, roundedRectangle.y0 + this.Height / 2 - roundedRectangle.h

/ 2, roundedRectangle.w, roundedRectangle.h), roundedRectangle.radius));

}

}

foreach (var text in texts.ToList())

{

graphics.DrawString(text.text, new Font("Arial", text.fontSize),

new SolidBrush(text.argb), text.x0 + this.Width / 2, text.y0 + this.Height / 2,

new StringFormat());

}

foreach (var picture in pictures.ToList())

{

graphics.SmoothingMode = SmoothingMode.Default;

Int16 x = picture.x0;

Int16 y = picture.y0;

for (int i = 0; i < picture.h; i++)

{

x = picture.x0;

for (int j = 0; j < picture.w; j++)

{

graphics.FillRectangle(new SolidBrush(picture.argb[j, i]),

x + this.Width / 2, y + this.Height / 2, 3, 3);

x += 3;

}

y += 3;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RECEIVE MESSAGE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

private void ReceiveMessage()

{

int port = 1001;

CSCS1.Commands commands = new CSCS1.Commands();

UdpClient receiver = new UdpClient(port);

IPEndPoint remoteIp = new IPEndPoint(IPAddress.Any, 0);

IPEndPoint iPEndPoint;

byte commandNum;

byte command;

Int16 x0, y0;

Int16 x1, y1;

Int16 radius;

string text;

string hexcolor;

Color argb;

try

{

while (true)

{

byte[] data = receiver.Receive(ref remoteIp);

commandNum = data[0];

switch (commandNum)

{

case 1:

commands.ClearDisplayDecode(data, out command, out

hexcolor);

Console.WriteLine($"Recieved command: clear display; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

DeleteAllGraphics();

this.BackColor = argb;

Invalidate();

break;

case 2:

commands.PixelDecode(data, out command, out x0, out

y0, out hexcolor);

Console.WriteLine($"Recieved command: draw pixel; x: { x0}; y: { y0}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

pixels.Add(new Pixels(x0, y0, argb));

Invalidate();

break;

case 3:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: draw line; x0: { x0}; y0: { y0}; x1: { x1}; y1: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

lines.Add(new Lines(x0, y0, x1, y1, argb));

Invalidate();

break;

case 4:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: draw rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

rectangles.Add(new Rectangles(x0, y0, x1, y1, argb,

false));

Invalidate();

break;

case 5:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: fill rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

rectangles.Add(new Rectangles(x0, y0, x1, y1, argb,

true));

Invalidate();

break;

case 6:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: draw ellipse; x: { x0}; y: { y0}; radius x: { x1}; radius y: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, x1, y1, argb,

false));

Invalidate();

break;

case 7:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: fill ellipse; x: { x0}; y: { y0}; radius x: { x1}; radius y: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, x1, y1, argb,

true));

Invalidate();

break;

case 8:

commands.CircleDecode(data, out command, out x0, out

y0, out radius, out hexcolor);

Console.WriteLine($"Recieved command: draw circle; x: { x0}; y: { y0}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, radius, radius,

argb, false));

Invalidate();

break;

case 9:

commands.CircleDecode(data, out command, out x0, out

y0, out radius, out hexcolor);

Console.WriteLine($"Recieved command: fill circle; x: { x0}; y: { y0}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, radius, radius,

argb, true));

Invalidate();

break;

case 10:

commands.RoundedRectDecode(data, out command, out x0,

out y0, out x1, out y1, out radius, out hexcolor);

Console.WriteLine($"Recieved command: draw rounded rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

roundedRectangles.Add(new RoundedRectangle(x0, y0, x1,

y1, radius, argb, false));

Invalidate();

break;

case 11:

commands.RoundedRectDecode(data, out command, out x0,

out y0, out x1, out y1, out radius, out hexcolor);

Console.WriteLine($"Recieved command: fill rounded rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

roundedRectangles.Add(new RoundedRectangle(x0, y0, x1,

y1, radius, argb, true));

Invalidate();

break;

case 12:

commands.TextDecode(data, out command, out x0, out y0,

out hexcolor, out x1, out y1, out text);

Console.WriteLine($"Recieved command: draw text; x: { x0}; y: { y0}; color: 0x{ hexcolor}; font number: { x1}; length: { y1}; text:{ text}; ");

argb = ColorConvert(hexcolor);

texts.Add(new Texts(x0, y0, argb, x1, text));

Invalidate();

break;

case 13:

commands.ImageDecode(data, out command, out x0, out

y0, out x1, out y1, out Color[,] colors);

Console.WriteLine($"Recieved command: draw image; x: { x0}; y: { y0}; width: { x1}; height: { y1}; colors: ");

pictures.Add(new Pictures(x0, y0, x1, y1, colors)); Invalidate();

break;

case 14:

rotation =

BitConverter.ToInt16(data.Skip(1).Take(2).ToArray(), 0);

Console.WriteLine($"Recieved command: set orientation; rotation angle: { rotation}; ");

Invalidate();

break;

case 15:

data =

BitConverter.GetBytes(Convert.ToInt16(this.Width));

Console.WriteLine($"Recieved command: get width;");

iPEndPoint = new IPEndPoint(remoteIp.Address,

remoteIp.Port);

receiver.Send(data, data.Length, iPEndPoint);

break;

case 16:

data =

BitConverter.GetBytes(Convert.ToInt16(this.Height));

Console.WriteLine($"Recieved command: get height;");

iPEndPoint = new IPEndPoint(remoteIp.Address,

remoteIp.Port);

receiver.Send(data, data.Length, iPEndPoint);

break;

case 17:

penWidth =

BitConverter.ToInt16(data.Skip(1).Take(2).ToArray(), 0);

Console.WriteLine($"Recieved command: set pen width; width: { penWidth}; ");

Invalidate();

break;

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

receiver.Close();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SECONDARY FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void DeleteAllGraphics()

{

pixels.Clear();

lines.Clear();

rectangles.Clear();

ellipses.Clear();

roundedRectangles.Clear();

texts.Clear();

pictures.Clear();

}

static public Color ColorConvert(string hexcolor)

{

Int16 color = Convert.ToInt16(hexcolor, 16);

string bits = Convert.ToString(color, 2).PadLeft(16, '0');

int R = Convert.ToInt32(bits.Substring(0, 5).PadRight(8, '0'), 2);

int G = Convert.ToInt32(bits.Substring(5, 6).PadRight(8, '0'), 2);

int B = Convert.ToInt32(bits.Substring(11, 5).PadRight(8, '0'), 2);

return Color.FromArgb(R, G, B);

}

public static GraphicsPath RoundedRect(Rectangle bounds, int radius)

{

int diameter = radius \* 2;

Size size = new Size(diameter, diameter);

Rectangle arc = new Rectangle(bounds.Location, size);

GraphicsPath path = new GraphicsPath();

if (radius == 0)

{

path.AddRectangle(bounds);

return path;

}

// top left arc

path.AddArc(arc, 180, 90);

// top right arc

arc.X = bounds.Right - diameter;

path.AddArc(arc, 270, 90);

// bottom right arc

arc.Y = bounds.Bottom - diameter;

path.AddArc(arc, 0, 90);

// bottom left arc

arc.X = bounds.Left;

path.AddArc(arc, 90, 90);

path.CloseFigure();

return path;

}

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

{

Invalidate();

}

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

{ }

}

}

Form1.Designer.cs

namespace CSCS2\_Forms

{

partial class Form1

{

private System.ComponentModel.IContainer components = null;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(800, 450);

this.Text = "Form1";

}

}

}

Program.cs:

using CSCS2\_Forms;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ServerApp

{

internal static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

Код для роботи клієнту:

Program.cs:

using System;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.IO;

namespace CSCS1

{

class SendProgram

{

static void Main(string[] args)

{

SendMessage();

}

private static void SendMessage()

{

string remoteAddress = "127.0.0.1";

int port = 1001;

Commands commands = new Commands();

UdpClient sender = new UdpClient(0);

IPEndPoint endPoint = new IPEndPoint(IPAddress.Parse(remoteAddress),

port);

Int16 x0, y0;

Int16 x1, y1;

Int16 radius;

string text;

string hexcolor;

try

{

while (true)

{

Console.Write("Введите команду > ");

string commandText = Console.ReadLine();

byte[] commandbyte = new byte[1];

byte[] result = new byte[1] { 0 };

switch (commandText)

{

case "1"://clear display

commandbyte[0] = 1;

hexcolor = ReadHexColor();

result = commands.ClearDisplayEncode(commandbyte[0], hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "2"://draw pixel

commandbyte[0] = 2;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

hexcolor = ReadHexColor();

result = commands.PixelEncode(commandbyte[0], x0, y0, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "3"://draw line

commandbyte[0] = 3;

x0 = ReadNumber("x0", false);

y0 = ReadNumber("y0", false);

x1 = ReadNumber("x1", false);

y1 = ReadNumber("y1", false);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0], x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "4"://draw rectangle

commandbyte[0] = 4; x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("Ширина", true);

y1 = ReadNumber("Высота", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0], x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "5"://fill rectangle

commandbyte[0] = 5;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("Ширина", true);

y1 = ReadNumber("Высота", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0],

x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "6"://draw ellipse

commandbyte[0] = 6;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("Радиус x", true);

y1 = ReadNumber("Радиус y", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0], x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "7"://fill ellipse

commandbyte[0] = 7;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("Радиус x", true);

y1 = ReadNumber("Радиус y", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0], x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "8"://draw circle

commandbyte[0] = 8;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

radius = ReadNumber("Радиус", true);

hexcolor = ReadHexColor();

result = commands.CircleEncode(commandbyte[0], x0, y0, radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "9"://fill circle

commandbyte[0] = 9;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

radius = ReadNumber("Радиус", true);

hexcolor = ReadHexColor();

result = commands.CircleEncode(commandbyte[0], x0, y0, radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "10"://draw rounded rectangle

commandbyte[0] = 10;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("Ширина", true);

y1 = ReadNumber("Высота", true);

radius = ReadNumber("Радиус", true);

hexcolor = ReadHexColor();

result = commands.RoundedRectEncode(commandbyte[0], x0, y0, x1, y1, radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "11"://fill rounded rectangle

commandbyte[0] = 11;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("Ширина", true);

y1 = ReadNumber("Высота", true);

radius = ReadNumber("Радиус", true);

hexcolor = ReadHexColor();

result = commands.RoundedRectEncode(commandbyte[0], x0, y0, x1, y1, radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "12"://draw text

commandbyte[0] = 12;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

hexcolor = ReadHexColor();

x1 = ReadNumber("Толщина", true);

Console.Write("Введите текст > ");

text = Console.ReadLine();

y1 = Convert.ToInt16(text.Length);

result = commands.TextEncode(commandbyte[0], x0, y0, hexcolor, x1, y1, text);

sender.Send(result, result.Length, endPoint);

break;

case "13"://draw image

commandbyte[0] = 13;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("Ширина", true);

y1 = ReadNumber("Высота", true);

text = ReadPath(); result = commands.ImageEncode(commandbyte[0], x0, y0, x1, y1, text);

sender.Send(result, result.Length, endPoint);

break;

default:

Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("Неизвестная оперция. Попробуйте снова");

Console.ResetColor();

break;

}

Console.WriteLine();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sender.Close();

}

}

public static bool IsStringInHex(string text)

{

return System.Text.RegularExpressions.Regex.IsMatch(text, @"\A\b[0-9afA-F]+\b\Z");

}

private static string ReadHexColor()

{

string str;

while (true)

{

Console.Write("Enter RGB565 color > ");

str = Console.ReadLine();

if (IsStringInHex(str) && str.Length <= 4)

{

break;

}

else

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Ошибка типа данных!");

Console.ResetColor();

}

}

return str;

}

private static Int16 ReadNumber(string text, bool onlyPositive = false)

{

string str;

Int16 number;

while (true)

{

Console.Write($"Enter {text} > ");

str = Console.ReadLine();

try

{

number = Int16.Parse(str);

if (onlyPositive)

{

if (number < 0)

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Ошибка! Неверные данные! (От 0 до 32767) Попробуйте снова.");Console.ResetColor();

}

else { break; }

}

else { break; }

}

catch

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Ошибка! Неверные данные! (От -32768 до 32767) Попробуйте снова.");

Console.ResetColor();

}

}

return Convert.ToInt16(str);

}

private static string ReadPath()

{

string str;

while (true)

{

Console.Write("Введите путь > ");

str = Console.ReadLine();

if (File.Exists(str) && IsImage(str))

{

break;

}

else

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Error! File does not exist! Try again.");

Console.ResetColor();

}

}

return @"" + str;

}

public static bool IsImage(string path)

{

return System.Text.RegularExpressions.Regex.IsMatch(path, @"^.\*\.(jpg|JPG|gif|GIF|png|PNG)$");

}

public static void RecieveMessage(UdpClient sender, IPEndPoint endPoint)

{

byte[] data = sender.Receive(ref endPoint);

Console.WriteLine($"Recieved value: {BitConverter.ToInt16(data, 0)}");

}

}

}

MyCommands.cs:

using System;

using System.Drawing;

using System.Linq;

using System.Text;

namespace CSCS1

{

public class Commands

{

//\*\*\*\*\*\*\*\*\*\*\*\*ClearDisplay\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] ClearDisplayEncode(byte command, string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return commandbyte.Concat(BitConverter.GetBytes(color)).ToArray();

}

public void ClearDisplayDecode(byte[] result, out byte command, out string

hexcolor)

{

command = result[0];

hexcolor = ByteToHexColor(result, 1);

}

//\*\*\*\*\*\*\*\*\*\*\*\*Pixel\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] PixelEncode(byte command, Int16 x0, Int16 y0, string

hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(color)).ToArray();

}

public void PixelDecode(byte[] result, out byte command, out Int16 x0, out

Int16 y0, out string hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

hexcolor = ByteToHexColor(result, 5);

}

//\*\*\*\*\*\*\*\*\*\*\*\*FourNumbers\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] FourNumbersEncode(byte command, Int16 x0, Int16 y0, Int16

x1, Int16 y1, string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(BitConverter.GetBytes(color)).ToArray();

}

public void FourNumbersDecode(byte[] result, out byte command, out Int16

x0, out Int16 y0, out Int16 x1, out Int16 y1, out string hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

x1 = BitConverter.ToInt16(result, 5);

y1 = BitConverter.ToInt16(result, 7);

hexcolor = ByteToHexColor(result, 9);

}

//\*\*\*\*\*\*\*\*\*\*\*\*Circle\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] CircleEncode(byte command, Int16 x0, Int16 y0, Int16 radius,

string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(radius)).Concat(BitConverter.GetBytes(color)).ToArray()

;

}

public void CircleDecode(byte[] result, out byte command, out Int16 x0,

out Int16 y0, out Int16 radius, out string hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

radius = BitConverter.ToInt16(result, 5);

hexcolor = ByteToHexColor(result, 7);

}

//\*\*\*\*\*\*\*\*\*\*\*\*RoundedRect\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] RoundedRectEncode(byte command, Int16 x0, Int16 y0, Int16

x1, Int16 y1, Int16 radius, string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(BitConverter.GetBytes(radius)).Concat(BitConverter.GetBytes(color)).ToArray();

}

public void RoundedRectDecode(byte[] result, out byte command, out Int16

x0, out Int16 y0, out Int16 x1, out Int16 y1, out Int16 radius, out string

hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

x1 = BitConverter.ToInt16(result, 5);

y1 = BitConverter.ToInt16(result, 7);

radius = BitConverter.ToInt16(result, 9);

hexcolor = ByteToHexColor(result, 11);

}

//\*\*\*\*\*\*\*\*\*\*\*\*Text\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] TextEncode(byte command, Int16 x0, Int16 y0, string

hexcolor, Int16 x1, Int16 y1, string text)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(color)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(Encoding.Unicode.GetBytes(text)).ToArray();

}

public void TextDecode(byte[] result, out byte command, out Int16 x0, out

Int16 y0, out string hexcolor, out Int16 x1, out Int16 y1, out string text)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

hexcolor = ByteToHexColor(result, 5);

x1 = BitConverter.ToInt16(result, 7);

y1 = BitConverter.ToInt16(result, 9);

text = Encoding.Unicode.GetString(result.Skip(11).Take(y1 \* 2).ToArray());

}

//\*\*\*\*\*\*\*\*\*\*\*\*Image\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] ImageEncode(byte command, Int16 x0, Int16 y0, Int16 x1,

Int16 y1, string data)

{

byte[] commandbyte = { command };

Color[] colors = ColorsEncode(new Bitmap(data, true), x1, y1);

byte[] byteColors = ColorsToByte(colors);

return commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(byteColors).ToArray();

}

public void ImageDecode(byte[] result, out byte command, out Int16 x0, out

Int16 y0, out Int16 x1, out Int16 y1, out Color[,] colors)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

x1 = BitConverter.ToInt16(result, 5);

y1 = BitConverter.ToInt16(result, 7);

colors = ByteToColors(result.Skip(9).Take(x1 \* y1 \* 4).ToArray(), x1,

y1);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SECONDARY FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static string ByteToHexColor(byte[] value, int startIndex)

{

Int16 color = BitConverter.ToInt16(value, startIndex);

return color.ToString("X");

}

public static Color[] ColorsEncode(Bitmap source, Int16 w, Int16 h)

{

Bitmap bmp = new Bitmap(source, w, h); Color[] result = new Color[w \* h];

int counter = 0;

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

{

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

{

result[counter] = bmp.GetPixel(j, i);

counter++;

}

}

return result;

}

public static byte[] ColorsToByte(Color[] colors)

{

int length = colors.Length;

byte[] result = new byte[0];

byte[] Combine(byte[] first, byte[] second)

{

byte[] ret = new byte[first.Length + second.Length];

Buffer.BlockCopy(first, 0, ret, 0, first.Length);

Buffer.BlockCopy(second, 0, ret, first.Length, second.Length);

return ret;

}

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

{

result = Combine(result,

BitConverter.GetBytes(colors[i].ToArgb()));

}

return result;

}

public static Color[,] ByteToColors(byte[] byteColors, Int16 w, Int16 h)

{

Color[,] result = new Color[w, h];

int counter = 0;

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

{

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

{

result[j, i] = Color.FromArgb(BitConverter.ToInt32(byteColors,4 \* counter));

counter++;

}

}

return result;

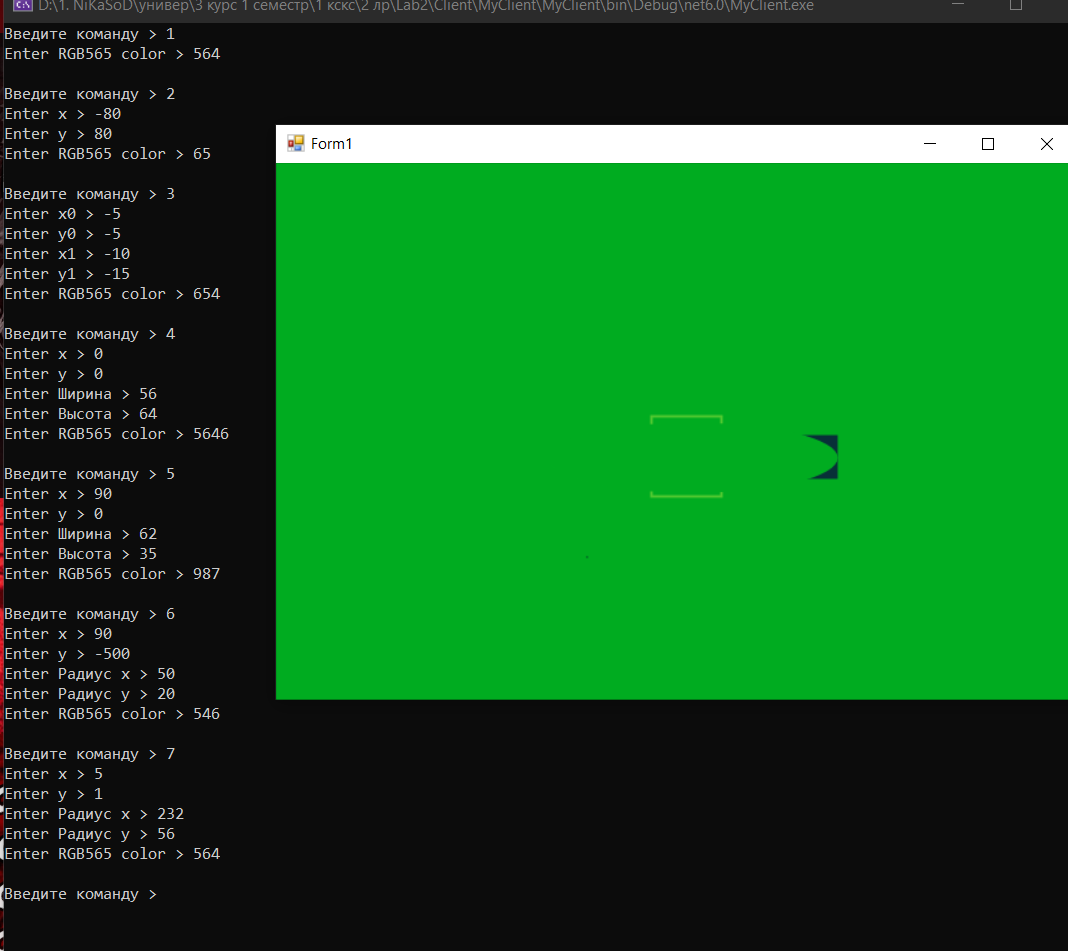
}

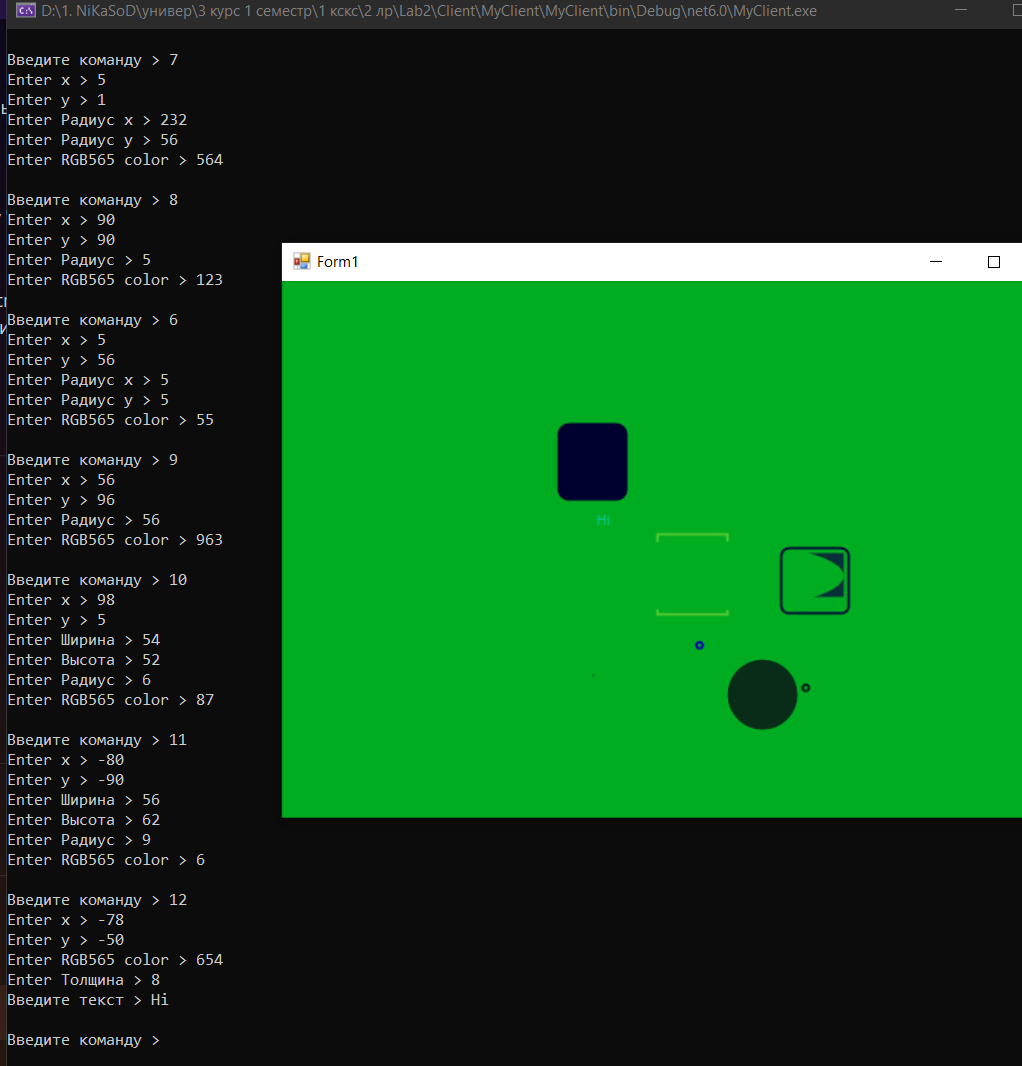
}

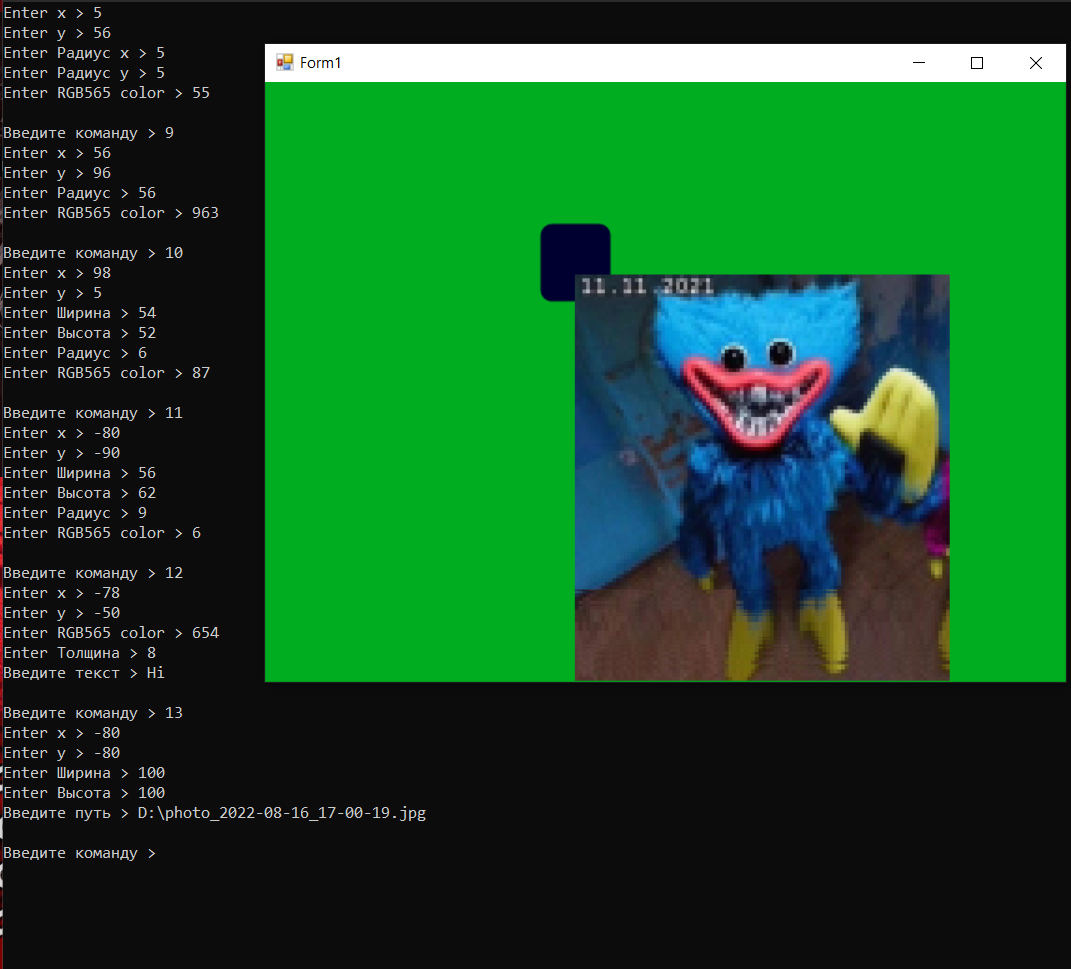
}

Файли проекту знаходяться за посиланням <https://github.com/caniplaysolo/ksks2>

Результат виконання







**Висновок**: розробив програму для реалізації протоколу управління дисплейним модулем.