Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра «Информатика и вычислительная техника»

Курс «Парадигмы и конструкции языков программирования»

Отчет по домашней работе

Выполнил:
студент группы РТ5-31:
Слкуни Г.Г.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2023

Текст программы:

MainWindow.xaml:

```
<Window x:Class="ServerSearcher.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ServerSearcher"
        mc:Ignorable="d"
        Title="ServerSearch" Height="380" Width="500">
    <Grid Margin="10">
        <StackPanel>
            <StackPanel Orientation="Horizontal">
                <StackPanel Margin="0, 0, 10, 10">
                    <Label>Диапозоны IP</Label>
                    <TextBox x:Name="IpRangeBox" AcceptsReturn="True" Width="300"
Height="100"></TextBox>
                </StackPanel>
                <StackPanel>
                    <Label>Проверяемый порт</Label>
                    <TextBox x:Name="PortBox">7777</TextBox>
                    <Label>Кол-во потоков</Label>
                    <TextBox x:Name="ThreadBox">50</TextBox>
                </StackPanel>
            </StackPanel>
            <Button x:Name="LaunchButton" Content="Начать"
Click="LaunchButton_Click"></Button>
            <StackPanel Orientation="Horizontal">
                <Label>Осталось проверить:</Label>
                <Label x:Name="StatusLabel"></Label>
            </StackPanel>
            <TextBox x:Name="ResultBox" Height="120" AcceptsReturn="True"
VerticalScrollBarVisibility="Visible"></TextBox>
        </StackPanel>
    </Grid>
</Window>
```

IPManager.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ServerSearcher
{
    internal class IPManager
    {

        private static List<IPRange> ipRangeList = new List<IPRange>();
        private static IPStats ipStats = new IPStats();
        private static Mutex mutex = new Mutex();
        private static bool needCalculateTotal = true;

        public static void Reset()
        {
            ipRangeList.Clear();
            needCalculateTotal = true;
            ipStats.Reset();
        }
```

```
        public static String GetIP()
        {
            mutex.WaitOne();
            while (ipRangeList.Count > 0) {
                String[] fromIp, toIp;
                if (needCalculateTotal)
                {
                    ipStats.totalIp = 0;
                    ipStats.ipChecked = 0;
                    needCalculateTotal = false;
                    fromIp = ipRangeList[0].ipFrom.Split('.');
                    toIp = ipRangeList[0].ipTo.Split('.');
                    while (fromIp[0] != toIp[0] || fromIp[1] != toIp[1] || fromIp[2]
!= toIp[2] || fromIp[3] != toIp[3])
                    {
                        int[] newValues = { int.Parse(fromIp[0]),
int.Parse(fromIp[1]), int.Parse(fromIp[2]), int.Parse(fromIp[3]) + 1 };
                        for (int i = 3; i >= 0; i--)
                        {
                            if (newValues[i] > 255)
                            {
                                newValues[i] = 0;
                                newValues[i - 1]++;
                            }
                        }
                        for (int i = 0; i <= 3; i++)
                        {
                            fromIp[i] = newValues[i].ToString();
                        }
                        ipStats.totalIp++;
                    }
                }
                fromIp = ipRangeList[0].ipFrom.Split('.');
                toIp = ipRangeList[0].ipTo.Split('.');
                if (fromIp[0] != toIp[0] || fromIp[1] != toIp[1] || fromIp[2] !=
toIp[2] || fromIp[3] != toIp[3])
                {
                    int[] newValues = { int.Parse(fromIp[0]), int.Parse(fromIp[1]),
int.Parse(fromIp[2]), int.Parse(fromIp[3]) + 1 };
                    for (int i = 3; i >= 0; i--)
                    {
                        if (newValues[i] > 255)
                        {
                            newValues[i] = 0;
                            newValues[i - 1]++;
                        }
                    }
                    for (int i = 0; i <= 3; i++)
                    {
                        fromIp[i] = newValues[i].ToString();
                    }
                    ipStats.ipChecked++;
                    ipRangeList[0].ipFrom = String.Join(".", fromIp);
                    mutex.ReleaseMutex();
                    return ipRangeList[0].ipFrom;
                }
                else
                {
                    needCalculateTotal = true;
                    ipRangeList.RemoveAt(0);
                }
            }
            mutex.ReleaseMutex();
            return null;
```

```
        }

        public static void AddIPRange(IPRange range)
        {
            ipRangeList.Add(range);
        }

        public static IPStats GetStats()
        {
            return ipStats;
        }

    }

    internal class IPStats
    {
        public int ipChecked { get; set; } = 0;
        public int totalIp { get; set; } = 0;

        public void Reset()
        {
            ipChecked = 0;
            totalIp = 0;
        }
    }

    internal class IPRange
    {
        public string ipFrom { get; set; } = "";
        public string ipTo { get; set; } = "";

        public IPRange(string ipFrom, string ipTo)
        {
            String[] ipFromParts = ipFrom.Split('.');
            ipFromParts[3] = (int.Parse(ipFromParts[3]) - 1).ToString();
            this.ipFrom = String.Join(".", ipFromParts);
            this.ipTo = ipTo;
        }
    }
}
```

MainWindow.xaml.cs:

```
using SAMP;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace ServerSearcher
{
```

```
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    ///

    public partial class MainWindow : Window
    {
        private List<Thread> threadList = new List<Thread>();

        public void Worker(object param)
        {
            int port = (int)param;
            String ip = null;
            while ((ip = IPManager.GetIP()) != null)
            {
                Dispatcher.Invoke(() =>
                {
                    IPStats stats = IPManager.GetStats();
                    StatusLabel.Content = String.Format("{0}/{1}", stats.ipChecked,
stats.totalIp);
                });
                try
                {
                    Query query = new Query(ip, port);
                    query.Send('i');
                    String[] data = query.Store(query.Recieve());
                    String result = String.Format("{0}:{1} ({2}, {3}/{4})\n", ip,
port, data[3], data[1], data[2]);
                    Dispatcher.Invoke(() =>
                    {
                        ResultBox.Text += result;
                    });
                    using (StreamWriter sw = File.AppendText("./result.txt"))
                    {
                        sw.Write(result);
                    }
                }
                catch (Exception)
                {
                    continue;
                }
            }
        }

        public void SoftwareWorker()
        {
            while (true)
            {
                Thread.Sleep(1000);
                Dispatcher.Invoke(() =>
                {
                    if (threadList.Count > 0)
                    {
                        threadList.RemoveAll(x => !x.IsAlive);
                        if (threadList.Count == 0)
                        {
                            LaunchButton.Content = "Начать";
                        }
                    }
                });
            }
        }

        public MainWindow()
        {
```

```
            InitializeComponent();
            Thread thr = new Thread(new ThreadStart(SoftwareWorker));
            thr.Start();
        }

        private void LaunchButton_Click(object sender, RoutedEventArgs e)
        {
            if (threadList.Count > 0)
            {
                for (int i = 0; i < threadList.Count; i++)
                {
                    threadList[i].Abort();
                }
                threadList = new List<Thread>();
                LaunchButton.Content = "Начать";
            }
            else
            {
                LaunchButton.Content = "Завершить";
                IPManager.Reset();
                foreach (String line in IpRangeBox.Text.Split('\n'))
                {
                    String ipRange = line.Replace("\r", "").Replace("\t", "");
                    String[] ips = ipRange.Split('-');
                    IPManager.AddIPRange(new IPRange(ips[0], ips[1]));
                }
                int port = int.Parse(PortBox.Text);
                int threads = int.Parse(ThreadBox.Text);
                for (int i = 0; i < threads; i++)
                {
                    Thread thr = new Thread(new ParameterizedThreadStart(Worker));
                    thr.Start(port);
                    threadList.Add(thr);
                    Thread.Sleep(10);
                }
            }
        }
    }
}
```

Результат выполнения: