2016-06-17

Podstawy Teleinformatyki

Projekt

Monitorowanie pracowni laboratoryjnej – dokumentacja

Sebastian Karkos, Adam Pluciński

Spis treści

1. Temat

Celem naszego projektu było stworzenie aplikacji umożliwiającej monitorowanie komputerów w obrębie jednej sieci (np. komputerów znajdujących się w pracowni laboratoryjnej).

Projekt składa się z dwóch typów aplikacji: *aplikacji zarządzającej* (master), która umożliwia użytkownikowi podgląd monitorowanych komputerów, oraz z *aplikacji klienckiej* (client), która wysyła obraz z komputera, na której została uruchomiona, do aplikacji zarządzającej. Oprócz transmisji obrazu pozwalającej na obserwację
komputera, aplikacja kliencka dostarcza także do aplikacji nadzorującej listy procesów oraz historie otwartych stron przeglądarki internetowej.

Wybraliśmy ten temat ze względu na to, że interesuję nas szeroko pojęty streaming danych, a sam projekt chcielibyśmy rozwijać w przyszłości. Oczywiście chęć głębszego przestudiowania programowania wielowątkowego oraz ogólnie pojętego programowania związanego z sieciami również odegrała dużą rolę.

2. Funkcje systemu

a) Aplikacja master

- prosty i przejrzysty interfejs,
- aplikacja po włączeniu odpowiada na broadcast sygnalizując kto jest “masterem”,
- posiada listę aplikacji klienckich(wraz z ich adresami IP), które połączyły się z aplikacją master,
- po kliknięciu na konkretny adres IP wyświetla się miniaturowy podgląd z pulpitu aplikacji klienckiej, który aktualizowany jest co 3 sekundy,
- po kliknięciu na konkretny podgląd aplikacja otwiera nowe okno, w którym obraz przesyłany jest około 30 razy szybciej,
- listę nadzorowanych urządzeń można na bieżąco zmieniać poprzez cofanie ich z powrotem do listy aktywnych urządzeń

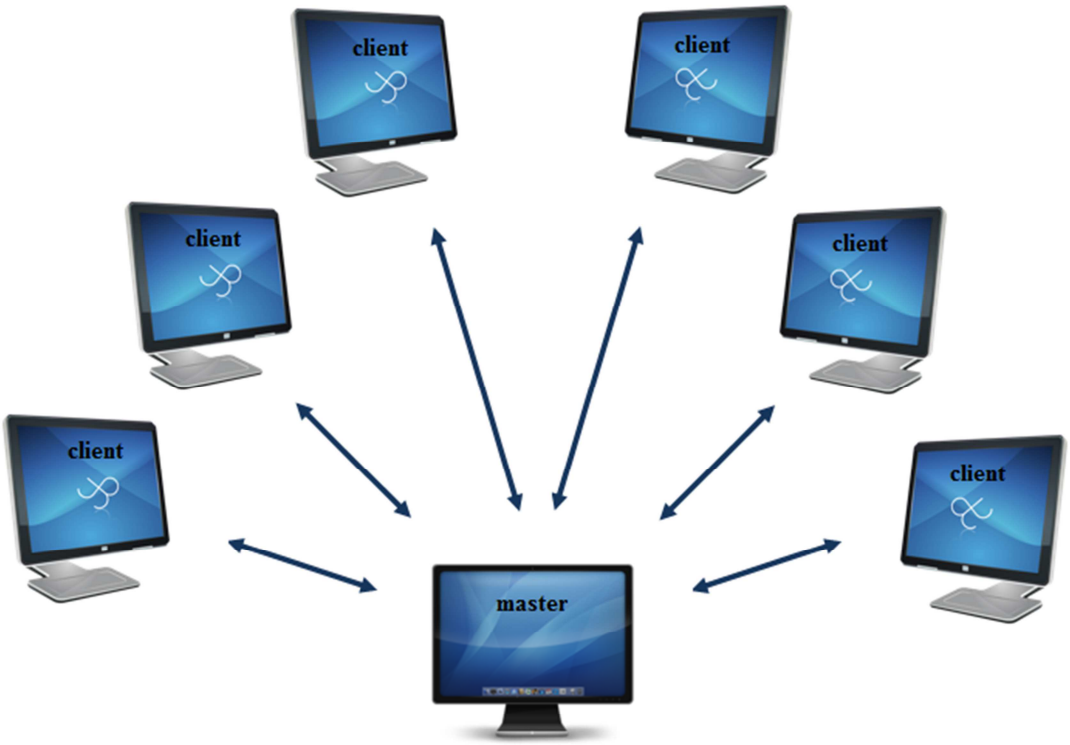
b) Aplikacja client

- aplikacja po włączeniu wysyła broadcast z zapytaniem “WhoIsMaster?”,
- przysyła listę aktywnych procesów,
- pracuję w dwóch trybach:
    - tryb wolnego przesyłania obrazu pulpitu (co 3 sekundy),
    - tryb szybkiego przesyłania obrazu (co 100 ms),

3. Podział prac

Imię i nazwisko	Zadania
Sebastian Karkos	Implementacja funkcji odpowiadających za broadcast, przesyłanie obrazu, mechanizm obsługiwanych komunikatów, szeroko pojęta komunikacja master<->klient
Adam Pluciński	Zaprojektowanie i implementacja UI, funkcję związane z background workerem, wielowątkowość

4. Architektura systemu



5. Napotkane problemy

Z pewnością sporym wyzwaniem było zapewnienie satysfakcjonującej wydajności przesyłania obrazu. Na początku przygody z streamingiem pojawił się pomysł całkowitej kontroli nad manipulowaniem przy obrazie - żeby uzyskać takie efekty potrzeba było autorskich metod do kompresji, dzielenia i przesyłania obrazu. Po dwóch miesiącach postanowiliśmy skorzystać z gotowych rozwiązań jakie oferuje TCP Bin Formatter. Do uzyskania podglądu co dzieje się na komputerze ta metoda jest wystarczająca, lecz nie daję płynności obrazu. W przyszłości na pewno przy podobnych aplikacjach zastosujemy FFMPEG z kodekiem h264 (dużo lepsza płynność, spora kompresja). Problemów dostarczył nam również wielowątkowy charakter aplikacji i nasze doświadczenie w tym zakresie.

6. Środowisko i narzędzia

- język programowania - C#, XAML (WPF)
- środowisko programistyczne - Microsoft Visual Studio 2012

7. Sposób użytkowania

1. Uruchamiamy aplikacje master*
2. Uruchamiamy aplikacje client*
3. Po pojawieniu się pełnego adresu mastera w oknie client klikamy połącz (adres można również wpisać ręcznie)
4. Adres clienta wyświetla się na liście znajdującej się po prawej stronie aplikacji master
5. Klikamy na przycisk “plus” znajdujący się obok adresu by uzyskać wolny podgląd pulpitu urządzenia o podanym adresie (pojawi się element na liście głównej)
6. Klikając na element na liście głównej otworzy się nowe okno z podglądam w trybie szybkim. Z tego okna możliwe jest także otwarcie listy procesów klienta oraz
7. Przycisk “X” przy podglądzie działa odwrotnie do kroku opisanego w punkcie 5)

*Kolejność kroków 1) i 2) nie ma znaczenia.

8. Implementacja

1) CLIENT

1.1)Zmienne publiczne

```
private readonly TcpClient client = new TcpClient();

    private NetworkStream mainStream;
       Thread FastConnectionThread;
       private readonly UdpClient BroadcastClient = new UdpClient();
       Thread BroadcastThread;
       bool Broadcastbool = false;
       public int PortToConnect = 0;

       public System.Windows.Forms.Timer timer1;

       public static Socket Client = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);

       public static IPEndPoint PClientIEP;

       DispatcherTimer SlowConnectionTimer = new DispatcherTimer();
       DispatcherTimer FastConnectionTimer = new DispatcherTimer();
```

- definicja klienta TCP, za pomocą którego przesyłany będzie obraz
- definicja strumienia, którym będzie wysyłany obraz,,
- definicja wątku odpowiadającego za szybsze przesyłanie obrazu,
- definicja klienta, który ma za zadanie wysyłać broadcast,
- definicja wątku, który odpowiada za uruchomienie funkcji z wysyłaniem broadcastu,
- definicja flagi, która wstrzymuję broadcast i zmienia wartość na true w przypadku gdy zostanie nawiązane połączenie z masterem
- definicja portu, z którym łączy się klient,
- definicja zegara, który wspomaga funkcję broadcastowe,
- definicja gniazda, który obsługuję komunikację krótkich komunikatów między masterem a klientem,
- definicja publicznej zmiennej IPEndPoint wykorzystywanej do komunikacji z serwerem,
- definicja zmiennych służących do przekazywania danych między wątkami

1.2)SendDesktopImage()

```
       private void SendDesktopImage()
              {
                  try
                  {
                      BinaryFormatter binFormatter = new BinaryFormatter();
                      mainStream = client.GetStream();
                      binFormatter.Serialize(mainStream, CaptureScreen(true));
                  }
                  catch
                  {
                      SlowConnectionTimer.Stop();
                      FastConnectionTimer.Stop();
                  }
```

```
        }
```

Funkcja odpowiadająca za wysyłanie obrazu pochodzącego z funkcji CaptureScreen(), obraz jest serializowany do strumienia i wysyłany do aplikacji master na konkretny adres i port.

1.3)FastConnectionTimerTick()

```
private void FastConnectionTimerTick(object sender, EventArgs e)
    {
        SendDesktopImage();
    }
```

Funkcja wywołująca SendDesktopImage() co określony czas(używana przy szybszym podglądzie).

1.4)SlowConnectionTimerTick()

```
    private void SlowConnectionTimerTick(object sender, EventArgs e)
    {
        SendDesktopImage();
    }
```

Funkcja wywołująca SendDesktopImage() co określony czas (używana przy wolniejszym podglądzie).

1.5)Broadcast()

```
private void Broadcast()
    {
        while (true)
        {
            if (Broadcastbool != true)
            {
                IPEndPoint BroadcastIP = new IPEndPoint(IPAddress.Broadcast, 415);
                byte[] Info = GetBytes("WhoIsMaster");
                BroadcastClient.Send(Info, Info.Length, BroadcastIP);

                Thread.Sleep(3000);
            }
            else if (Broadcastbool == true)
            {
            }
        }

    }
```

Funkcja odpowiadająca za wysyłanie broadcastu w przypadku, gdy klient nie połączył się z żadną aplikacją master, w razie braku połączenia wznawiana jest ona po 3 sekundach.

1.6)CheckFastConnection()

```
private void CheckFastConnection()
        {
            while (true)
            {

            IPEndPoint ServerIEP = new IPEndPoint(IPAddress.Any, Port--);
            EndPoint ServerEP = (EndPoint)(ServerIEP);
            byte[] Response = new byte[200];

                Client.ReceiveFrom(Response,ref ServerEP);
                string rgvtvyv = GetString(Response);
                string[] messageReceived = SplitString2(rgvtvyv);
                if (messageReceived[0].Contains("STARTFASTCONNECTION"))
                {
                    SlowConnectionTimer.Stop();
                    FastConnectionTimer.Start();
                }
                else if (messageReceived[0].Contains("STOPFASTCONNECTION"))
                {
                    FastConnectionTimer.Stop();
                    SlowConnectionTimer.Start();
                }
                else if (messageReceived[0].Contains("SHOWPROCESESS"))
                {
                    Process[] process = Process.GetProcesses();
                    string processes = "PROCESESS;";
                    foreach (Process prs in process)
                    {
                        double size = prs.PrivateMemorySize64 / 1024;
                        processes += prs.ProcessName + ";" + size+ ";";
                    }

                    IPEndPoint ServerIEPPROC = new
IPEndPoint(IPAddress.Parse(SERVIP), 415);
                    EndPoint ServerEPPRC = (EndPoint)(ServerIEPPROC);
                    byte[] pro = GetBytes(processes);
                    Client.SendTo(pro, ServerEPPRC);
                }
            }
        }
```

Funkcja odpowiadająca za ustanawianie szybszego przesyłania obrazu. Na początku inicjowany jest obiekt, który przechowuje adres serwera na który klient nasłuchuje w celu przechwycenia pakietów z informacjami odnośnie dalszych instrukcji(szybsze przesyłanie, wysyłanie procesów). W tej metodzie odpowiednio uruchamiane są timery, które odpowiadają za włączenie funkcji wysyłające obraz w konkretnej prędkości. Ostatni fragment kodu odpowiada za pobranie listy procesów oraz wysłanie ich do serwera.

1.7)ConnectionAfterBroadcast()

```
private void ConnectionAfterBroadcast()
       {
           IPEndPoint ServerIEP = new IPEndPoint(IPAddress.Any, 415);


           EndPoint ServerEP = (EndPoint)(ServerIEP);
           byte[] Response = new byte[200];
           Response = BroadcastClient.Receive(ref ServerIEP);
           string rgvtvyv = GetString(Response);
           string[] messageReceived = SplitString2(rgvtvyv);
               if (messageReceived[0] == "CONNECTIONPORT" && messageReceived[1] !=
"")
                {
                       PortToConnect = Int32.Parse(messageReceived[1]);

                       HostIP_TB.Text = messageReceived[2];
                       SERVIP = HostIP_TB.Text;
                       Broadcastbool = true;
                       ConfirmButton.IsEnabled = true;
                }

                }
```

Funkcja odpowiadająca za wyświetlenie adresu mastera i wyzwolenie przycisku połącz - w tym momencie klient ma już wszystkie dane potrzebne do kontaktu z serwerem i może rozpocząć wymianę danych.

1.8)GetMyComputer_LanIP()

```
private string GetMyComputer_LanIP()
   {
       string strHostName = System.Net.Dns.GetHostName();

       IPHostEntry ipEntry = System.Net.Dns.GetHostEntry(strHostName);

       foreach (IPAddress ipAddress in ipEntry.AddressList)
       {
           if (ipAddress.AddressFamily.ToString() == "InterNetwork")
           {
               return ipAddress.ToString();
           }
       }

       return "-";
   }
```

Funkcja odpowiadająca za pobranie aktualnie przydzielonego adresu IP(obojętnie czy adres jest przydzielony statycznie, czy dynamicznie poprzez DHCP).

1.9)RegisterInStartup

```
private void RegisterInStartup(bool isChecked)
   {
       RegistryKey registryKey = Registry.CurrentUser.OpenSubKey
              ("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true);
       if (isChecked)
       {
                          registryKey.SetValue("LANSupervisorClient",
    System.Reflection.Assembly.GetExecutingAssembly().Location);
       }
       else
       {
           registryKey.DeleteValue("LANSupervisorClient");
       }
   }
```

Funkcja odpowiadająca za uruchomienie aplikacji LANSupervisorClient przy starcie systemu(domyślnie wyłączona).

1.10)ConfirmButton_Click()

```
private void ConfirmButton_Click(object sender, RoutedEventArgs e)
   {
       string HostIP = HostIP_TB.Text.Trim();
       int PortNumber = 12300;
       string[] words = HostIP.Split('.');

       if (words.Length != 4 || words[3] == "")
       {
           System.Windows.MessageBox.Show("Wpisz poprawny adres IP");
       }
       else
       {
           try
           {
               IPAddress IpServer = IPAddress.Parse(HostIP);
               IPEndPoint ServerIEP = new IPEndPoint(IpServer, 415);
               EndPoint ServerEP = (EndPoint)(ServerIEP);



                   client.Connect(HostIP, PortToConnect);
                   SlowConnectionTimer.Start();
                   ConnectedTB.Text += " " + HostIP;
                   ConnectedTB.Visibility = Visibility.Visible;
                   //System.Windows.MessageBox.Show("Connected!");
                   Broadcastbool = true;
               //}

           }

           catch (Exception ee)
           {
               System.Windows.MessageBox.Show("Nie można połączyć z nadzorcą. Sprawdź
poprawność adresu IP oraz czy aplikacja nadzorcy jest włączona." + ee.Message);
           }

       }
   }
```

Przycisk “Połącz” - odpowiada za zatwierdzenie otrzymanych danych i wykorzystanie ich do połączenia z serwerem. W tym przypadku zmienna Broadcastbool zostaje ustanowiona na true, czyli klient przestaje wysyłać komunikaty do sieci w poszukiwaniu mastera.

1.11)GetString()

```
public static string GetString(byte[] bytes)
   {
       char[] chars = new char[bytes.Length / sizeof(char)];
       System.Buffer.BlockCopy(bytes, 0, chars, 0, bytes.Length);
       return new string(chars);
   }
```

Funkcja konwertuje tablice bajtów do łańcucha znaków(gwarantuje większą szybkość niż funkcję “Encoding”).

1.12)GetBytes()

```
public static byte[] GetBytes(string str)
   {
       byte[] bytes = new byte[str.Length * sizeof(char)];
       System.Buffer.BlockCopy(str.ToCharArray(), 0, bytes, 0, bytes.Length);
       return bytes;
   }
```

Funkcja konwertuje łańcuch znaków do tablicy bajtów(gwarantuje większą szybkość niż funkcję “Encoding”).

1.13)CaptureScreen()

```
public static Bitmap CaptureScreen(bool CaptureMouse)
   {
       Bitmap result = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
Screen.PrimaryScreen.Bounds.Height, System.Drawing.Imaging.PixelFormat.Format24bppRgb);

       try
       {
           using (Graphics g = Graphics.FromImage(result))
           {
               g.CopyFromScreen(0, 0, 0, 0, Screen.PrimaryScreen.Bounds.Size,
CopyPixelOperation.SourceCopy);

               if (CaptureMouse)
               {
                   CURSORINFO pci;
                   pci.cbSize =
System.Runtime.InteropServices.Marshal.SizeOf(typeof(CURSORINFO));
```

```
                    if (GetCursorInfo(out pci))
                    {
                        if (pci.flags == CURSOR_SHOWING)
                        {
                            DrawIcon(g.GetHdc(), pci.ptScreenPos.x, pci.ptScreenPos.y,
pci.hCursor);
                            g.ReleaseHdc();
                        }
                    }
                }
            }
        }
        catch
        {
            result = null;
        }

        return result;
    }
```

Funkcja odpowiadająca za robienie zrzutów ekranu i zwracanie ich w formie Bitmap. W tym przypadku zwraca obraz o rozdzielczości odpowiadającej rozdzielczości urządzenia na którym działa aplikacja. Precyzujemy tu również format pikseli w którym ma zostać zrobiony zrzut. Zaletą funkcji jest nanoszenie kursora myszki na obraz, gdyż kursor stanowi inną warstwę w systemie niż pulpit(inne funkcję do zrzutów ekranu nie uwzględniały kursora).

1.14)

```
[StructLayout(LayoutKind.Sequential)]
    struct CURSORINFO
    {
        public Int32 cbSize;
        public Int32 flags;
        public IntPtr hCursor;
        public POINTAPI ptScreenPos;
    }

    [StructLayout(LayoutKind.Sequential)]
    struct POINTAPI
    {
        public int x;
        public int y;
    }

    [DllImport("user32.dll")]
    static extern bool GetCursorInfo(out CURSORINFO pci);

    [DllImport("user32.dll")]
    static extern bool DrawIcon(IntPtr hDC, int X, int Y, IntPtr hIcon);

    const Int32 CURSOR_SHOWING = 0x00000001;
```

Dane potrzebne do lokalizowania kursora myszki na pulpicie w celu naniesienia go na zrzut ekranu(GetCursorInfo oraz DrawIcon - rysowanie kursora).

2) Master

2.1) Zmienne publiczne

```
public static Socket Server = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);

private BackgroundWorker FirstBackgroundWorker = null;

public static string ConnectionMessage = "CONNECTIONPORT_";
public static int AvaiblePort;

public BitmapImage toSet { get; set; }
DispatcherTimer ReloadImagesTimer = new DispatcherTimer();

public Thread MainThread;
public Thread Receive;


public List<MainListObject> mainlist = new List<MainListObject>();
```

- definicja gniazda odpowiadającą za wymianę krótkich komunikatów za pośrednictwem protokołu UDP pomiędzy masterem a clientem
- definicja background workera odpowiadającego za odświeżanie obrazu w obiektach(każdy obiekt ma dwa pola Bitmap oraz Adres - w tym przypadku nowo otrzymany obraz zastępuję poprzedni i zostaję wyświetlony w podglądzie), które reprezentują danego klienta
- definicja łańcucha znaków używanego do przesyłania portu na którym następuję połączenie,
- zmienna pomocnicza typu Bitmap,
- timer, który służy do przeładowania obrazu w wolnym podglądzie(jego wartość odpowiada czasowi, co jaki aktualizowane są obrazy),
- definicja dwóch wątków, jednego głównego odpowiadającego za funkcję związaną z komunikacją między masterem a clientem, a drugiego dotyczącego odbierania danych przez serwer,
- definicja listy obiektów, z której każdy obiekt odpowiada unikalnemu klientowi, który podłączy się do aplikacji master

2.2) Pozostałe dane

```
IPAddress ServerIP = IPAddress.Parse(GetMyIp());
          IPEndPoint ServerIEP = new IPEndPoint(ServerIP, 415);
          Server.Bind(ServerIEP);

          AvaiblePort = 500;
```

Dane, które łączone są z gniazdem serwera oraz numer portu, od którego można łączyć się z serwerem(wartość ta narasta wraz ze zwiększającą się liczbą podłączonych klientów).

```
if (null == FirstBackgroundWorker)
            {
                FirstBackgroundWorker = new BackgroundWorker();
                FirstBackgroundWorker.DoWork += new
DoWorkEventHandler(FirstBackgroundWorker_DoWork);
                FirstBackgroundWorker.ProgressChanged += new
ProgressChangedEventHandler(FirstBackgroundWorker_ProgressChanged);
                FirstBackgroundWorker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(FirstBackgroundWorker_RunWorkerCompleted);
                FirstBackgroundWorker.WorkerReportsProgress = true;
                FirstBackgroundWorker.WorkerSupportsCancellation = true;
            }

            FirstBackgroundWorker.RunWorkerAsync();

            ReloadImagesTimer.Interval = TimeSpan.FromSeconds(3);
            ReloadImagesTimer.Tick += ReloadImagesTimerTick;
            ReloadImagesTimer.Start();
```

Implementacja funkcjonalności background workera, dotyczą aktualizacji obrazów w wolnym podglądzie(domyślnie ustawione na 3 sekundy).

2.3)FirstBackgroundWorker_ProgressChanged()

```
void FirstBackgroundWorker_ProgressChanged(object sender, ProgressChangedEventArgs
e)
        {
            ReceivedObject obj = (ReceivedObject)e.UserState;
            Bitmap a = obj.Image;
            string IP = obj.IP;
            if (mainlist.Exists(aa => aa.IP == IP))
            {
               ((MainListObject)mainlist.Find(b => b.IP == IP)).Image =
ToBitmapImage(a);
            }
            else
            {
                MainListObject neww = new MainListObject();
                neww.IP = IP;
                neww.Image = ToBitmapImage(a);
                mainlist.Add(neww);

            }
        }
```

Funkcja odpowiadająca za nadpisywanie obrazu w odpowiednich obiektach. Jeżeli aplikacja odbierze nowy obraz, sprawdza, czy obiekt o danym adresie istnieje już na liście klientów, jeżeli tak to nadpisuję obraz na poprzedni, jeżeli nie - tworzy nowy obiekt z IP klienta oraz przesyłanym obrazem i dodaję go do listy.

2.4)ToBitmapImage

```
public static BitmapImage ToBitmapImage(Bitmap bitmap)
        {
            using (var memory = new MemoryStream())
            {
                bitmap.Save(memory, ImageFormat.Png);
```

```
                memory.Position = 0;
                var bitmapImage = new BitmapImage();
                bitmapImage.BeginInit();
                bitmapImage.StreamSource = memory;
                bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
                bitmapImage.EndInit();

                return bitmapImage;
            }
        }
```

Funkcja dzięki której ze zmiennej Bitmap otrzymywaliśmy BitmapImage - było to niezbędne w ustawieniu źródła kontrolki Image w WPF'ie, gdyż tylko taki format jest przez tę kontrolkę akceptowalny.

2.5) FirstBackgroundWorker_RunWorkerCompleted()

```
void FirstBackgroundWorker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            // a to odpala go na nowo
            FirstBackgroundWorker.RunWorkerAsync();
        }
```

Kolejna funkcja dotycząca Background Workera - zadaniem tej jest uruchomienie go na nowo po wypełnieniu podanego zadania(ProgressChanged()).

2.6) ReloadImagesTimerTick()

```
private void ReloadImagesTimerTick(object sender, EventArgs e)
        {
            List<MainListObject> tempList = new List<MainListObject>();
            List<MainListObject> tempList2 = new List<MainListObject>();
            foreach (MainListObject ob in mainlist)
            {
                if (ob.DisplayedOnMainList == true)
                {
                    tempList.Add(ob);
                }
                else
                {
                    tempList2.Add(ob);
                }
            }
            MainListBox.ItemsSource = null;
            MainListBox.ItemsSource = tempList;

            AvailableComputers.ItemsSource = null;
            AvailableComputers.ItemsSource = tempList2;
        }
```

Funkcja odpowiadająca za równoczesne aktualizowanie obrazów w aktualnie "podglądanych" klientach - zapobiega migotaniu obrazu.

2.7)CheckMessageType()

```
public void CheckMessageType()
        {
            while (true)
```

```
            {
                byte[] Info = new byte[200];
                IPEndPoint RandomClient = new IPEndPoint(IPAddress.Any, 415);
                EndPoint RandomClientEP = (EndPoint)(RandomClient);
                Server.ReceiveFrom(Info, ref RandomClientEP);
                string ReceivedInfo = GetString(Info);
                if (ReceivedInfo.Contains("STARTCONNECTION"))
                {
                    //ODSYLA PORT I OTWIERA TCPLISTENER NA TEN PORT
                    string FirstResponseSTR = "CONNECTIONPORT_" +
AvaiblePort.ToString() + "_";
                    byte[] FirstResponse = GetBytes(FirstResponseSTR);
                    Server.SendTo(FirstResponse,RandomClientEP);

                    string HostIP = RandomClientEP.ToString();
                    string[] IPArray = HostIP.Split(':');

                    Thread ListeningThread = new Thread(()=>
OpenListener(AvaiblePort, IPArray[0]));
                    ListeningThread.Start();
                    AvaiblePort++;


                }
                else if (ReceivedInfo.Contains("STOPCONNECTION"))
                {
                    //ZAMYKA WATEK Z TYM TCPLISTENEREM


                }
                else if (ReceivedInfo.Contains("WhoIsMaster"))
                {
                    string FirstResponseSTR = "CONNECTIONPORT_" +
AvaiblePort.ToString() + "_" + GetMyIp() + "_";
                    byte[] FirstResponse = GetBytes(FirstResponseSTR);
                    Server.SendTo(FirstResponse, RandomClientEP);

                    string HostIP = RandomClientEP.ToString();
                    string[] IPArray = HostIP.Split(':');


                    Thread ListeningThread = new Thread(() =>
OpenListener(AvaiblePort, IPArray[0]));
                    ListeningThread.Start();
                    AvaiblePort++;
                }
            }
```

Funkcja, która odpowiada za ciągłe nasłuchiwanie pakietów na ustalonym porcie. W przypadku odebranego pakietu sprawdza jego zawartość i na podstawie instrukcji warunkowych wykonuję odpowiednie funkcję.
STARTCONNECTION - rozpoczyna transmisje danych z klientem, otwiera nowy wątek z TCPListenerem, który nasłuchuje na odpowiedni port w celu otrzymania transmisji obrazu,
STOPCONNECTION - zamyka wątek z TCPListenerem
WhoIsMaster - wysyła odpowiedź na broadcasty z zapytaniem, która aplikacja jest masterem

2.8)OpenListener()

```
public void OpenListener(int Port, string IP)
        {
            Port--;
            TcpListener Listener = new TcpListener(IPAddress.Any, Port);
            TcpClient Client = new TcpClient();
            NetworkStream mainStream;
            Listener.Start();
            Client = Listener.AcceptTcpClient();
            BinaryFormatter binFormatter = new BinaryFormatter();
            while (Client.Connected)
            {
                mainStream = Client.GetStream();
                try
                {
                    Bitmap a = (Bitmap)binFormatter.Deserialize(mainStream);
                    if (a != null)
                    {
                        ReceivedObject neww = new ReceivedObject();
                        neww.Image = a;
                        neww.IP = IP;
                        FirstBackgroundWorker.ReportProgress(0, neww);
                    }
                }
                catch
                {
                    mainlist.Remove(mainlist.Find(a=> a.IP == IP));
                    Thread.CurrentThread.Abort();
                }
            }
```

Funkcja wywoływana przez CheckMessageType() w przypadku, gdy podłączy się nowy klient. Otwierana jest w nowym wątku na konkretnym porcie przydzielonym przez aplikacje master. W przypadku zerwaniu połączenia wątek zostaje anulowany a obiekt odpowiadający danemu klientowi usunięty z listy podłączonych klientów.

2.9)AddItem_Click()

```
private void AddItem_Click(object sender, RoutedEventArgs e)
        {
            Button btn = sender as Button;

            if (btn != null)
            {
                object item = btn.DataContext;
                if (item != null)
                {
                    ((MainListObject)item).DisplayedOnMainList = true;
                    RefreshLists();
                }
            }
        }
```

Funkcja przycisku pojawiającego się obok każdego adresu klienta figurującego na liście podłączonych klientów. Po kliknięciu zostaje utworzony wolny podgląd w oknie głównym(klient/adres znikają z listy podłączonych klientów).

2.10)CloseItemOnList_Click()

```
private void CloseItemOnList_Click(object sender, RoutedEventArgs e)
        {
            Button btn = sender as Button;

            if (btn != null)
            {
                object item = btn.DataContext;
                if (item != null)
                {
                    ((MainListObject)item).DisplayedOnMainList = false;
                    RefreshLists();
                }
            }
        }
```

Funkcja z zadaniem odwrotnym do AddItem_Click() - przenosi klienta z powrotem na listę podłączonych klientów(usuwając tym samym wolny podgląd).

9. Podsumowanie

Rezultat końcowy pracy nad projektem oceniamy jako zadowalający. W kontekście założeń początkowych udało zapewnić się nam większość funkcjonalności. Niezrealizowanym przez nas zadaniem było jednak zapewnienie możliwości definiowania listy stron internetowych jakie użytkownik może odwiedzać po stronie aplikacji klienckiej.

Praca zespołowa przebiegała sprawnie i bezkonfliktowo, choć dość nieregularnie. Wiązało się to z innymi projektami, które tworzyliśmy równolegle w ramach studiów. Udało się nam odpowiednio rozdzielić zadania dzięki czemu mogliśmy pracować również zdalnie. Największe postępy jednak uzyskiwaliśmy podczas pracy wspólnej.

Z całą pewnością praca nad projektem pozwoliła zdobyć nam cenne doświadczenie i zdobyć nowe umiejętności. Wiedzę tą zdążyliśmy wykorzystać już w ramach realizacji innych projektów.