Połączenia protokołem TCP

Zaczynamy od projektu *Tcp* (z pliku *Tcp_0_START.zip*)
W rozwiązaniu są dwie aplikacje: dla Windows (WPF) i dla Androida (Xamarin)
Projekty mają gotowe GUI i metody zdarzeniowe. Prezentują też lokalne adresy IP.

Zaprojektujemy prosty protokół przesyłania wieloliniowych łańcuchów (tekstu) oparty na TCP. Dodane powiadamianie o zakończeniu połączenia.

W ustawieniach rozwiązania - uruchamianie obu projektów: **Multiple startup projects** (w obu Start)

Aplikacja Windows

W aplikacji Windows dodajemy plik *NtppClient.cs* (nazwa od *Network Text Passing Protocol*)

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace JacekMatulewski.Communication.Ntpp
{
    //identyczne jak dane połączenia w Tcp
    public class NtppConnectionData
    {
        private IPAddress ipAddress;
        private string hostname;
        public int Port = 11236; //domyślny port NTPP

        public IPAddress IpAddress
        {
            get
            {
                return ipAddress;
            }
            set
            {
                ipAddress = value;
                IPHostEntry hostEntry = Dns.GetHostEntry(ipAddress);
                hostname = (hostEntry != null) ? hostEntry.HostName :
ipAddress.ToString();
            }
        }

        public string Hostname
        {
            get
            {
                return hostname;
            }
            set
```

```
            {
                hostname = value;
                ipAddress = IPAddress.Parse(hostname);
            }

        }
    }

    public class NtppClient
    {
        public enum ServerState { NotConnected, WaitingForConnection,
Connecting, Connected }
        private ServerState state = ServerState.NotConnected;
        public ServerState State
        {
            get
            {
                return state;
            }
            private set
            {
                ServerState oldState = this.state;
                this.state = value;
                onStateChanged(oldState, this.state);
            }
        }

        public string LastErrorMessage { get; private set; }

        private NtppConnectionData connectionData;
        private TcpClient tcpClient;

        #region Client
        public bool Connect(NtppConnectionData connectionData, int
timeoutMs = 10000) //odpowiednik StartClient
        {
            try
            {
                State = ServerState.Connecting;
                this.connectionData = connectionData;
                //tcpClient = new TcpClient(connectionData.Hostname,
connectionData.Port); //tu nie można ustawić timeoutu
                tcpClient = new TcpClient();
                if (!tcpClient.ConnectAsync(connectionData.Hostname,
connectionData.Port).Wait(timeoutMs))
                {
                    State = ServerState.NotConnected;
                    throw new Exception("Connection to server time out");
                }
                useStreamsFromTcpClient(tcpClient);
                State = ServerState.Connected;
                return true;
            }
```

```
            catch (Exception exc)
            {
                LastErrorMessage = exc.Message;
                return false;
            }
        }

        public static NtppClient ConnectAsClient(NtppConnectionData
connectionData)
        {
            NtppClient ntppClient = new NtppClient();
            if (ntppClient.Connect(connectionData)) return ntppClient;
            else throw new Exception(ntppClient.LastErrorMessage);
        }
        #endregion

        #region Server
        private TcpListener tcpListener = null;
        private Thread listeningLoopThread = null;

        //nie ma funkcji StopServer - za to należy wywołać Disconnect
        public bool WaitForConnection(NtppConnectionData connectionData)
//odpowiednik StartServer
        {
            try
            {
                this.connectionData = connectionData;
                tcpListener = new TcpListener(connectionData.IpAddress,
connectionData.Port); //przyjmuje ze wszystkich (obsłuży wiele klientów)
                tcpListener.Start();
                listeningLoopThread = new Thread(waitingForConnection);
listeningLoopThread.Start();
                State = ServerState.WaitingForConnection;
                return true;
            }
            catch (Exception exc)
            {
                LastErrorMessage = exc.Message;
                return false;
            }
        }

        private void waitingForConnection() //czeka na połączenia
        {
            try
            {
                tcpClient = tcpListener.AcceptTcpClient();
                useStreamsFromTcpClient(tcpClient);
                State = ServerState.Connected;
            }
            catch (System.Threading.ThreadAbortException)
            {
```

```
                LastErrorMessage = "Pętla nasłuchiwacza została
zatrzymana";
            }
            catch (Exception exc)
            {
                LastErrorMessage = exc.Message;
            }
        }

        public static NtppClient ConnectAsServer(NtppConnectionData
connectionData)
        {
            NtppClient ntppClient = new NtppClient();
            if (ntppClient.WaitForConnection(connectionData)) return
ntppClient;
            else throw new Exception(ntppClient.LastErrorMessage);
        }
        #endregion

        public bool Disconnect()
        {
            try
            {
                stopReceivingLoopThread = true;

                if (tcpClient != null)
                {
                    if (State == ServerState.Connected)
                        SendText(NtppHelper.ByeCommand);
                    if (sw != null) { sw.Close(); sw = null; }
                    if (sr != null) { sr.Close(); sr = null; }
                    tcpClient.Close(); tcpClient = null;
                }

                if (tcpListener != null)
                {
                    tcpListener.Stop(); tcpListener = null;
                }

                State = ServerState.NotConnected;
                return true;
            }
            catch (Exception exc)
            {
                LastErrorMessage = exc.Message;
                return false;
            }
        }

        private void useStreamsFromTcpClient(TcpClient tcpClient)
        {
            stopReceivingLoopThread = false;
            NetworkStream tcpClientStream = tcpClient.GetStream();
```

```
            //tcpClientStream.ReadTimeout = 10000;
            //tcpClientStream.WriteTimeout = 10000;
            sw = new StreamWriter(tcpClientStream); sw.AutoFlush = true;
            sr = new StreamReader(tcpClientStream);
            startReceivingText();
        }

        #region Sender
        private StreamWriter sw;

        public bool SendText(string text)
        {
            string _text = NtppHelper.RemoveNewLines(text);
            try
            {
                if (State != ServerState.Connected) throw new
Exception("Connection is requires to send the text");
                sw.Write(_text + Environment.NewLine); //dodaję znak końca
linii, bo w serwerze używam ReadLine (możnaby wysyłać kilka tekstów i
dopiero kończyć znakiem końca linii)
                //sw.Flush(); //niepotrzebne, bo AutoFlush
                return true;
            }
            catch (Exception exc)
            {
                LastErrorMessage = exc.Message;
                return false;
            }
        }
        #endregion

        #region Receiver
        private StreamReader sr;
        private Thread textReceivingThread = null;
        public string LastReceivedText { get; private set; }
        private bool stopReceivingLoopThread = false;

        private void startReceivingText()
        {
            textReceivingThread = new Thread(textReceivingLoop);
            textReceivingThread.Start();
        }

        private void textReceivingLoop()
        {
            while (true)
            {
                if (stopReceivingLoopThread) break;
                try
                {
                    string _text = sr.ReadLine(); //ze względu na ReadLine
wiadomości muszą kończyć się znakiem końca linii; przez \n nie może być w
wiadomości - trzeba wymieniać na inny znak
```

```
                    string text = NtppHelper.RecoverNewLines(_text);
                    if (text == null || text == "") return;
                    switch (text) //komendy
                    {
                        case NtppHelper.ByeCommand:
                            Disconnect();
                            break;
                        default:
                            LastReceivedText = text;
                            onTextReceived(text); //uwaga, to będzie w
osobnym wątku
                            break;
                    }
                }
                catch (Exception exc)
                {
                    //może być błąd przy rozłączaniu
                    LastErrorMessage = exc.Message;
                }
                Thread.Sleep(10);
            }
        }
        #endregion

        #region Events
        public class TextReceivedEventArgs : EventArgs
        {
            public NtppConnectionData ReceivedFrom;
            public string ReceivedText;
        }

        public class StateChangedEventArgs : EventArgs
        {
            public ServerState OldState, NewState;
        }

        public event EventHandler<TextReceivedEventArgs> TextReceived;
        public event EventHandler<StateChangedEventArgs> StateChanged;

        protected void onTextReceived(string text)
        {
            if (TextReceived != null) TextReceived(this, new
TextReceivedEventArgs() { ReceivedFrom = connectionData, ReceivedText =
text });
        }

        protected void onStateChanged(ServerState oldState, ServerState
newState)
        {
            if (StateChanged != null) StateChanged(this, new
StateChangedEventArgs() { OldState = oldState, NewState = newState });
        }
        #endregion
```

```
    }

    static class NtppHelper
    {
        private const string newlineReplacement = "&@nl$;";

        public static string RemoveNewLines(string text)
        {
            return text.Replace(Environment.NewLine, newlineReplacement);
        }

        public static string RecoverNewLines(string text)
        {
            return text.Replace(newlineReplacement, Environment.NewLine);
        }

        public const string ByeCommand = "&@bye$;";
    }
}
```

Plik MainWindow.xaml.cs – aplikacja WPF
(jest gotowy XAML i metody zdarzeniowe przycisków)
Aplikacja Windows będzie działać jako serwer (czeka na połączenie)

```
using System;
using System.Collections.Generic;
using System.Windows;

namespace TcpWindows
{
    using JacekMatulewski.Communication.Ntpp;

    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private NtppClient ntppClient;

        private static string[] getValidIPAddresses()
        {
            List<string> addresses = new List<string>();
            System.Net.IPHostEntry host =
System.Net.Dns.GetHostEntry(System.Net.Dns.GetHostName());
            foreach (System.Net.IPAddress adres in host.AddressList)
                addresses.Add(adres.ToString());
            return addresses.ToArray();
        }

        private void pokażAdresyIp()
        {
            string[] addresses = getValidIPAddresses();
```

```
            string s = "Lokalne adresy IP:\r\n";
            foreach (string address in addresses)
                s += address + "\r\n";
            tbLokalneAdresyIp.Text = s.TrimEnd('\r','\n');
        }

        public MainWindow()
        {
            InitializeComponent();

            pokażAdresyIp();

            ntppClient = new NtppClient();
            ntppClient.StateChanged += NtppClient_StateChanged;
            ntppClient.TextReceived += NtppClient_TextReceived;
        }

        private void NtppClient_StateChanged(object sender,
                                             NtppClient.StateChangedEventArgs e)
        {
            Action a = () => { tbStan.Text = e.NewState.ToString(); };
            if (!Dispatcher.CheckAccess()) Dispatcher.Invoke(a); else a();
        }

        private void NtppClient_TextReceived(object sender,
                                             NtppClient.TextReceivedEventArgs e)
        {
            Action a = () => {
lbOdebraneKomunikaty.Items.Add(e.ReceivedFrom.Hostname + " (" +
DateTime.Now.ToString() + "): " + e.ReceivedText); };
            if (!Dispatcher.CheckAccess()) Dispatcher.Invoke(a); else a();
        }

        private void btnPolacz_Click(object sender, RoutedEventArgs e)
        {
            NtppConnectionData connectionData = new NtppConnectionData()
            {
                Hostname = tbAdres.Text,
                Port = Convert.ToInt32(tbPort.Text)
            };
            ntppClient.WaitForConnection(connectionData);
        }

        private void btnRozlacz_Click(object sender, RoutedEventArgs e)
        {
            ntppClient.Disconnect();
        }

        private void btnWyslij_Click(object sender, RoutedEventArgs e)
        {
            ntppClient.SendText(tbTekst.Text);
        }
    }
```

```
}
```

Ze zdarzeniem `Window.Closed` związać metodę zdarzeniową:

```
private void Window_Closed(object sender, EventArgs e)
{
    //to odłączanie zdarzeń to przejaw paranoi
    ntppClient.StateChanged -= ntppClient_StateChanged;
    ntppClient.TextReceived -= ntppClient_TextReceived;

    btnRozlacz_Click(this, null);
}
```

Aplikacja Android

Do projektu *TcpAndroid* dodajemy plik *NtppClient.cs*. Ale nie dodajemy tylko linkujemy (**Add as Link**)

Plik MainActivity.cs

```
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Widget;

using System;
using System.Collections.Generic;

namespace Tcp
{
    using JacekMatulewski.Communication.Ntpp;

    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme",
MainLauncher = true,
        WindowSoftInputMode = Android.Views.SoftInput.StateHidden)]
    public class MainActivity : AppCompatActivity
    {
        private TextView tvLokalneAdresyIp, tvStan;
        private EditText etAdres, etPort, etTekst;
        private Button btnPolacz, btnRozlacz, btnWyslij;
        private ListView lvOdebraneWiadomosci;

        private NtppClient ntppClient;

        private void pokażAdresyIp()
        {
            List<string> ips = Tcp.AdresyIP.PobierzAdresyIp();
            string sips = "";
            foreach (string ip in ips) sips += ip + "\n";
            tvLokalneAdresyIp.Text = sips.TrimEnd('\n');
        }

        protected override void OnCreate(Bundle savedInstanceState)
```

```
        {
            base.OnCreate(savedInstanceState);
            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.activity_main);

            tvLokalneAdresyIp =
FindViewById<TextView>(Resource.Id.tvLokalneAdresyIp);

            etAdres = FindViewById<EditText>(Resource.Id.etAdres);
            etPort = FindViewById<EditText>(Resource.Id.etPort);
            btnPolacz = FindViewById<Button>(Resource.Id.btnPolacz);
            btnRozlacz = FindViewById<Button>(Resource.Id.btnRozlacz);
            tvStan = FindViewById<TextView>(Resource.Id.tvStan);

            etTekst = FindViewById<EditText>(Resource.Id.etTekst);
            btnWyslij = FindViewById<Button>(Resource.Id.btnWyslij);

            lvOdebraneWiadomosci =
FindViewById<ListView>(Resource.Id.lvOdebraneWiadomosci);

            btnPolacz.Click += btnPolacz_Click;
            btnRozlacz.Click += btnRozlacz_Click;
            btnWyslij.Click += btnWyslij_Click;

            pokażAdresyIp();

            ntppClient = new NtppClient();
            ntppClient.StateChanged += ntppClient_StateChanged;
            ntppClient.TextReceived += ntppClient_TextReceived;
        }

        private void ntppClient_StateChanged(object sender,
                                        NtppClient.StateChangedEventArgs e)
        {
            Action a = () => { tvStan.Text = e.NewState.ToString(); };
            this.RunOnUiThread(a);
        }

        List<string> odebraneWiadomości = new List<string>();

        private void pokażOdebraneWiadomości()
        {
            try
            {
                ArrayAdapter<string> adapter =
                    new ArrayAdapter<string>(
                        this,
                        Android.Resource.Layout.SimpleListItem1,
                        odebraneWiadomości);
                lvOdebraneWiadomosci.Adapter = adapter;
                //scroll down to end
                lvOdebraneWiadomosci.Post(
    () => lvOdebraneWiadomosci.SetSelection(adapter.Count - 1));
```

```
            }
            catch (Exception exc)
            {
                Toast.MakeText(this, "Błąd: " + exc.Message,
                               ToastLength.Long).Show();
            }
        }

        private void ntppClient_TextReceived(object sender,
                                        NtppClient.TextReceivedEventArgs e)
        {
            odebraneWiadomości.Add(e.ReceivedFrom.Hostname + " (" +
DateTime.Now.ToString() + "): " + e.ReceivedText);
            Action a = () => { pokażOdebraneWiadomości(); };
            this.RunOnUiThread(a);
        }

        private void btnPolacz_Click(object sender, System.EventArgs e)
        {
            try
            {
                NtppConnectionData connectionData =
                    new NtppConnectionData()
                {
                    Hostname = etAdres.Text,
                    Port = Convert.ToInt32(etPort.Text)
                };
                ntppClient.Connect(connectionData);
            }
            catch(Exception exc)
            {
                Toast.MakeText(this, "Błąd: " + exc.Message,
                               ToastLength.Short).Show();
            }
        }

        private void btnRozlacz_Click(object sender, System.EventArgs e)
        {
            try
            {
                ntppClient.Disconnect();
            }
            catch (Exception exc)
            {
                Toast.MakeText(this, "Błąd: " + exc.Message,
                               ToastLength.Short).Show();
            }
        }

        private void btnWyslij_Click(object sender, System.EventArgs e)
        {
            ntppClient.SendText(etTekst.Text);
        }
```

```
        protected override void OnDestroy()
        {
            //to odłączanie zdarzeń to przejaw paranoi
            ntppClient.StateChanged -= ntppClient_StateChanged;
            ntppClient.TextReceived -= ntppClient_TextReceived;

            btnRozlacz_Click(this, EventArgs.Empty);

            base.OnDestroy();
        }
    }
}
```