Prosta aplikacja klient - serwer na bazie protokołu UDP. Sprawozdanie.

Autor Pierwszy, Autor Drugi, Autor Trzeci

Styczeń 2012r

Wyższa Szkoła Biznesu w Dąbrowie Górniczej
Informatyka, I rok Studiów UM

Spis treści

1 Temat

W niniejszym ćwiczeniu należało przygotować aplikację typu klient - serwer z użyciem protokołu UDP. Ma w niej istnieć możliwość przesłania wiadomości "witaj świecie" z klienta do serwera, a serwer ma zidentyfikować nadawcę wiadomości i wyświetlić ją. Dodatkowo należało:

1. Zmodyfikować program serwera, aby po uruchomieniu wypisywał własny adres IP jako pierwszą pozycję na liście.
2. Przetestować działanie aplikacji poprzez łączenie się wielu klientów z jednym serwerem.
3. Dodać w obu procesach: klienta i serwera możliwość wskazania numeru portu, przez który nastąpi komunikacja.
4. Dodatkowo - aplikacja klienta ma umożliwiać wysłanie komunikatu o dowolnej treści.

2 Analiza i projekt

Aplikacja składa się z dwóch części - serwera oraz klienta. Klient, wykorzystując protokół UDP przesyła do programu serwera wiadomość w postaci łańcucha znaków. Serwer po odebraniu wiadomości wyświetla ją na ekranie. Aplikacja zrealizowana jest z wykorzystaniem platformy .NET, w języku programowania C#. Wszystkie klasy, nie zastosowane, a nie zdefiniowane w niniejszym ćwiczeniu należą do biblioteki .NET Framework.

2.1 Algorytmy, struktury danych i zmiany w specyfikacji

Specyfikacja aplikacji określona w instrukcji laboratoryjnej została rozwinięta o następujące zachowania:

1. Zastosowano wielowątkowość po stronie serwera.
2. Dodano przyciski, umożliwiające uruchomienie/zatrzymanie serwera.
3. Program serwera posiada wbudowanego klienta, co pozwala zrezygnować z przygotowywania osobnej aplikacji klienta i umożliwia symetryczną komunikację między wieloma programami.

2.1.1 Serwer

Działanie programu serwera oparte jest o wykorzystanie klasy UdpClient. Klasa ta umożliwia wysyłanie i odbieranie ciągu bajtów przez wybrane gniazdo sieciowe, określone przez adres IP oraz port, z wykorzystaniem protokołu UDP. Takie rozwiązanie umożliwia realizację komunikacji klient - serwer, narzuca jednak pewne ograniczenia.

- Odbieranie danych jest realizowane przy pomocy metody UdpClient.Receive(), która jest metodą blokującą. Program wstrzymuje swoje działanie w systemowej funkcji odbierającej dane z gniazda. W związku z tym w czasie oczekiwania na dane program nie reaguje na polecenia użytkownika. Aby umożliwić interakcję z programem wykorzystaliśmy udostępniany w platformie interfejs wielowątkowości, z wykorzystaniem obiektu BackgroundWorker. Blokująca metoda Receive() wywoływana jest w metodzie BackgroundWorker.DoWork(), która wykonywana jest przez dodatkowy wątek. Program wykonywany jest przez dwa wątki, z których jeden zajmuje się obsługą formatki, a drugi odbiera dane przesyłane gniazdem sieciowym.

- Aby umożliwić zmianę wykorzystywanego do komunikacji portu potrzebna jest możliwość zatrzymania działania serwera. Wątek, w którym obsługiwany jest odczyt danych z gniazda jest zatrzymany w metodzie Receive(). Dlatego też, aby go zatrzymać musimy mieć możliwość wyjścia z tej metody. W prezentowanym przykładzie zrealizowano to poprzez wysłanie komunikatu kończącego działanie serwera, przez klienta wbudowanego w program serwera. Klient ten jest tworzony i wywoływany po naciśnięciu przycisku „Zatrzymaj".

- Zastosowanie wielowątkowości wymaga implementacji delegata, który umożliwi wątkowi realizującemu obsługę odbierania danych zapisywanie do pola należącego do wątku obsługującego formatkę aplikacji. Delegat ten to SetText, który umożliwia dopisanie łańcucha znaków do pola w obiekcie lbConnections. Drugim, zdefiniowanym w aplikacji delegatem jest ResetText, który usuwa zawartość tekstowego pola lbConnections.

- Aby uniknąć prób wielokrotnego wywołania metody BackgroundWorker.DoWork(), co doprowadziłoby do błędu programu, zastosowano prosty mechanizm kontrolny, polegający na deaktywacji przycisku Uruchom po uruchomieniu serwera. Przycisk ten aktywowany jest ponownie po zatrzymaniu serwera. Podobnie dla przycisku zatrzymaj oraz pola numerycznego określającego port, na którym działa serwer.

- Windows 7 standardowo nie umożliwia korzystania z słowa „localhost" jako określenia sieciowego urządzenia zwrotnego, dlatego w kodzie ręcznie wprowadzono adres IPv4 127.0.0.1, wskazujący na to urządzenie.

- W Windows 7 metoda GetHostName() zwraca adres IPv6. Aby otrzymać adres IPv4, wprowadzona została metoda LocalIPadres(), która przeszukuje wszystkie adresy lokalne i zwraca pierwszy adres, który nie jest adresem IPv6.

2.1.2 Klient

Realizacja programu klienta także została oparta o klasę UdpClient. Ponieważ jednak protokół UDP nie wymaga potwierdzenia doręczenia wiadomości, metoda Send() nie jest metodą blokującą i trudności z takimi metodami programu klienta nie dotyczą.

2.2 Dyskusja możliwych rozwiązań

2.2.1 Problem blokowania

Aby wyeliminować problem blokowania w metodzie Receive() możnaby pokusić się o zastosowanie alternatywnych, nieblokujących metod odbierania danych. Wówczas program nie wymagałby dodatkowych komunikatów, wysyłanych by zatrzymać serwer oczekujący na pakiety UDP. Można zrealizować to przy pomocy metody ReceiveAsync(), jednak metoda ta oparta jest o zdarzenia (Events). Implementacja jej istotnie skomplikowałaby przykład.

3 Specyfikacja zewnętrzna

3.1 Obsługa programu

Rysunek 1 przedstawia wygląd aplikacji klienta/serwera UDP. Składa się z następujących obiektów, pogrupowanych wierszami:

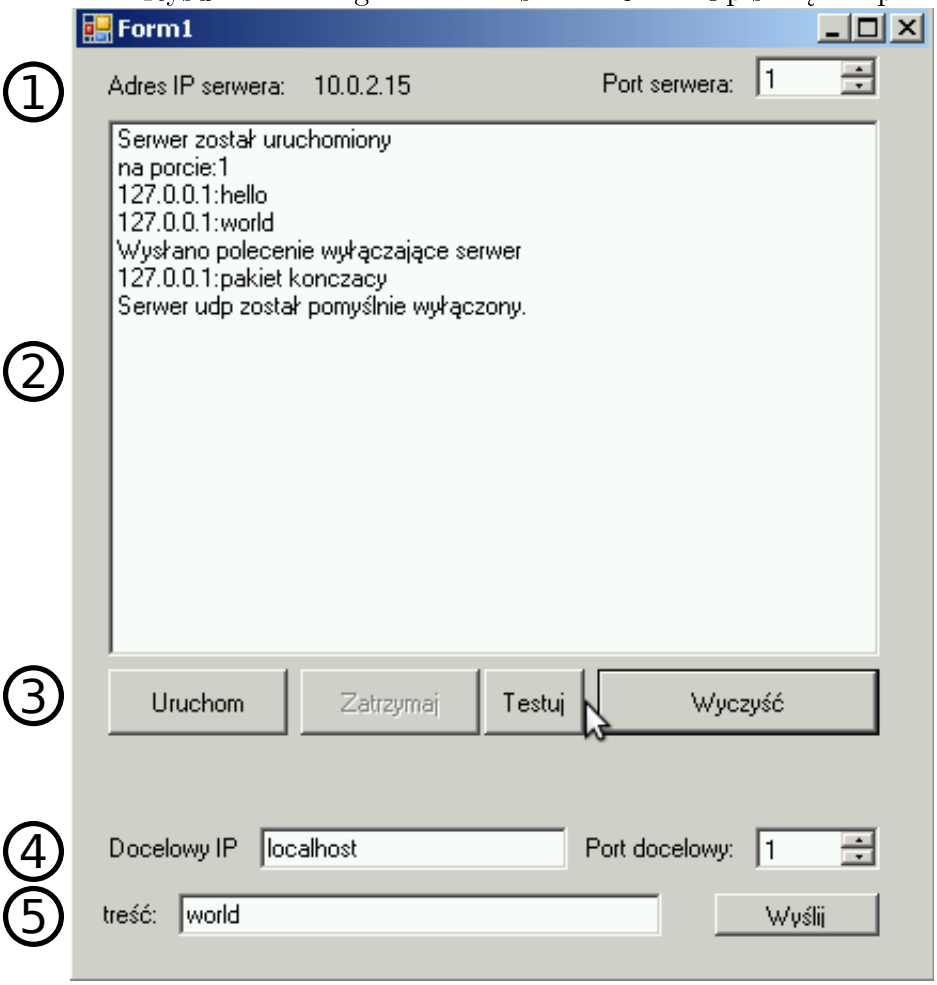
1. Adres i port serwera, do którego można wysyłać komunikaty. Pole wyświetlające adres nie jest edytowalne, a zawierające port - można edytować, jeżeli serwer jest wyłączony. Port jest opisany liczbą całkowitą P taką, że $1 \leq P \leq 65535$.

2. Pole wyświetla komunikaty, które odebrał serwer.

3. Przyciski sterujące działaniem serwera: Uruchom, Zatrzymaj, Testuj i Wyczyść. Działanie przycisków Uruchom i Zatrzymaj jest oczywiste. Przycisk Testuj wysyła testowy komunikat do serwera. Przycisk Wyczyść usuwa zawartość okienka komunikatów odebranych przez serwer.

4. Pola konfigurujące klienta: adres IP i port, do którego mają zostać wysłane komunikaty. Pole port docelowy przyjmuje takie same wartości jak pole port serwera.

Rysunek 1: Program klient-serwer UDP. Opis części aplikacji w tekście.



5. Pole treść, zawierające treść komunikatu do wysłania i przycisk wyślij, który ten komunikat wysyła.

3.2 Komunikaty

Kliencka część aplikacji nie generuje żadnych komunikatów - protokół UDP nie wymaga potwierdzenia doręcznia wiadomości, a tym bardziej jej poprawności, więc wysłanie wiadomości nie generuje nigdy błędu.

Część aplikacji, realizująca funkcje serwera wyświetla komunikaty w tym samym oknie, co komunikaty przychodzące z gniazda sieciowego. Komunikaty te mogą dotyczyć uruchomienia serwera (zawierający numer portu, na którym serwer jest uruchomiony), próby jego zatrzymania (i potwierdzenia zatrzymania jeżeli się ono powiodło), być komunikatem testowym (o treści „serwer pracuje poprawnie") lub komunikatem pochodzącym od aplikacji klienta o dowolnej treści, poprzedzony numerem IP klienta.

4 Specyfikacja wewnętrzna

Program ze względu na swoją prostotę zaimplementowany jest jako publiczna klasa Form1, dziedzicząca z klasy Form biblioteki .NET Framework.

4.1 Metody

W niniejszej części zaprezentowane są najważniejsze metody wykorzystane w aplikacji.

private void btUruchom_Click(object sender, EventArgs e)
Metoda obsługująca uruchomienie serwera nasłuchującego komunikatów UDP. Uruchamia wątek Backgroundworker1 metodą RunWorkerAsync(), wyświetla komunikat o uruchomieniu serwera w oknie komunikatów aplikacji, deaktywuje przycisk uruchom i pole **nPort** oraz aktywuje przycisk Zatrzymaj.

private string LocalIPAddress()
Metoda pobierająca adres IP serwera. Przeszukuje wszystkie adresy lokalne zwracane przez *ipLocalHost.AddressList* i **zwraca** pierwszy adres, który nie jest adresem IPv4 jako łańcuch string.

private void Form1_Load(object sender, EventArgs e)
Metoda inicjuje działanie aplikacji. Wyświetla w polu adresu IP serwera odpowiedni adres IP.

private void SetText(string text) **private** void ResetText(string text)
Metody odpowiadają za zawartość okna komunikatów odebranych przez serwer. Pierwsza z metod do listy wyświetlonych komunikatów dodaje tekst *text*. Druga usuwa zawartość okna komunikatów.

private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
Metoda odczytująca komunikaty przychodzące do gniazda sieciowego. W nieskończonej pętli while(true) powtarza próbę odczytu z gniazda sieciowego. Przed każdym odczytem sprawdza, czy flaga backgroundWorker1.CancellationPending ma wartość true. Jeżeli tak, to kończy działanie serwera. Jeżeli nie, to pętla jest wykonywana.

private void btZatrzymaj_Click(object sender, EventArgs e)
Metoda zatrzymująca działanie serwera. Wywołując metodę BackgroundWorker1.CancelAsync() ustawia flagę BackgroundWorker1.CancellationPending na true. Następnie wysyła komunikat do serwera, co pozwala wątkowi serwera opuścić blokującą metodę Receive(). Wysyła komunikat informujący o zakończeniu działania serwera, deaktywuje przycisk Zakończ i aktywuje przycisk Uruchom oraz pole port serwera.

private void btWyczysc_Click(object sender, EventArgs e)
Metoda usuwa zawartość okna z komunikatami odebranymi przez serwer.

private void btTestuj_Click(object sender, EventArgs e)
Metoda wysyła przy pomocy lokalnie utworzonej instancji klasy UdpClient komunikat testowy do serwera, korzystając z portu i adresu serwera.

private void btclientSend_Click(object sender, EventArgs e)
Metoda realizuje działanie klienta UDP. Lokalnie tworzy instancję klasy UdpClient i korzystając z gniazda sieciowego określonego przez wartości pól port docelowy i adres docelowy wysyła komunikat zawarty w polu treść.

5 Kod programu (najważniejsze fragmenty)

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

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;


namespace prostySerwerUDP
{
    public partial class Form1 : Form
    {
        delegate void SetTextCallback(string text);

        public Form1()
        {
            InitializeComponent();
        }

        private void btUruchom_Click(object sender, EventArgs e)
        {
            backgroundWorker1.RunWorkerAsync();
            lbConnections.Items.Add("Serwer zosta uruchomiony");
            btUruchom.Enabled = false;
            nPort.Enabled = false;
            btZatrzymaj.Enabled = true;
            this.Refresh();
        }

        private string LocalIPAddress()
        {
            string ipv4 = "";
            IPHostEntry ipLocalHost = Dns.GetHostEntry(Dns.GetHostName());
```

```
        foreach (IPAddress ipAdress in ipLocalHost.AddressList)
        {
            if (ipAdress.IsIPv6LinkLocal == false)
            {
                ipv4 = ipAdress.ToString();
                break;
            }
        }
        return ipv4;
    }


    private void Form1_Load(object sender, EventArgs e)
    {
        label1.Text = LocalIPAddress();
    }

    private void SetText(string text)
    {
        if (this.lbConnections.InvokeRequired)
        {
            SetTextCallback d = new SetTextCallback(SetText);
            this.Invoke(d, new object[] { text });
        }
        else
            this.lbConnections.Items.Add(text);
    }


    private void ResetText(string text)
    {
        if (this.lbConnections.InvokeRequired)
        {
            SetTextCallback d = new SetTextCallback(ResetText);
            this.Invoke(d, new object[] { text });
        }
```

```
            else
                this.lbConnections.Items.Clear();
        }


        private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {

            int calkPort = (int)nPort.Value;
            UdpClient udpServer = new UdpClient(calkPort);
            SetText("na porcie:"+ calkPort.ToString("D"));
            while (true)//!backgroundWorker1.CancellationPending)
                if (!backgroundWorker1.CancellationPending)
                {
                    IPEndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Any,
0);

                    Byte[] odebraneDane = udpServer.Receive(ref RemoteIpEndPoint);
                    string daneZwrotne = Encoding.ASCII.GetString(odebraneDane);
                    SetText(RemoteIpEndPoint.Address.ToString() + ":"+ daneZwrotne);

                }
                else
                {
                   break;
                }
            udpServer.Close();
            SetText("Serwer udp zosta pomylnie wyczony.");

        }

        private void btZatrzymaj_Click(object sender, EventArgs e)
        {
            backgroundWorker1.CancelAsync();

            int calkPort = (int)nPort.Value;
            UdpClient udpClient = new UdpClient("127.0.0.1", calkPort);
```

```
            Byte[] sendBytes = Encoding.ASCII.GetBytes(''pakiet konczacy'');
            udpClient.Send(sendBytes, sendBytes.Length);
            // backgroundWorker1.Dispose();
            SetText(''Wysano polecenie wyczajce serwer'');
            btUruchom.Enabled = true;
            btZatrzymaj.Enabled = false;
            nPort.Enabled = true;

        }

        private void btWyczysc_Click(object sender, EventArgs e)
        {
            ResetText('''');
        }

        private void btTestuj_Click(object sender, EventArgs e)
        {
            int calkPort = (int)nPort.Value;
            UdpClient udpClient = new UdpClient(''127.0.0.1'', calkPort);
            Byte[] sendBytes = Encoding.ASCII.GetBytes(''Serwer pracuje poprawnie'');
            udpClient.Send(sendBytes, sendBytes.Length);
        }

        private void btclientSend_Click(object sender, EventArgs e)
        {

            int calkPort = (int)targetPort.Value;
            UdpClient udpClient = new UdpClient(targetIP.Text, calkPort);
            Byte[] sendBytes = Encoding.ASCII.GetBytes(clientText.Text);
            udpClient.Send(sendBytes, sendBytes.Length);
        }
    }
}
```

6 Testowanie

Działanie aplikacji przetestowano na komputerach z zainstalowanym systemem Windows XP oraz Windows 7. Aby aplikacja mogła się poprawnie komunikować, niezbędne było wyłączenie Firewalla w zakresie portów wykorzystywanych przez aplikacje.