jamk.fi

.NET Harjoitustyö

Puskala Juha

Harjoitustyön dokumentaatio
Kirjoituspäivämäärä 22.11.2015

Chat Room sovellus

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Sisällys

1 Asennus

1.1 Serveri

Severin asennuksessa tulee ottaa huomioon palomuuri ja tietoturvaseikat. Jotta asiakasohjelmat voivat yhdistää serveriin on palomuurista hyväksyttävä porttiin 5555 tulevat socket yhteyspyynnöt. Windows työpöytäversiot kysyvät lupaa automaattisesti.

2 Tietoa ohjelmasta

Toteutunut tuotos ei vastaa projektisuunnitelmassa esiteltyä ohjelmaa. Alun perin harjoitustyönä piti toteuttaa ASP.NETillä nettisivu, mutta web socketien käyttö muodostui ongelmaksi, sillä niitä ei voi käyttää ollenkaan Windows 7 ja testaaminen virtuaalikoneen IIS8 ei toiminut halutulla tavalla. Niinpä päädyttiin toteuttamaan Chat Room sovellus WPF:a käyttämällä.

2.1 Toteutuneet toiminnallisuudet

Client sovelluksessa käyttäjä voi:

- Yhdistää käynnissä olevaan serveriin ip+port yhdistelmällä.
- Valita keskustelussa muille näkyvän nimen
- Lähettää viestejä ja nähdä muiden käyttäjien lähettämät viestit

Server sovelluksessa käyttäjä voi:

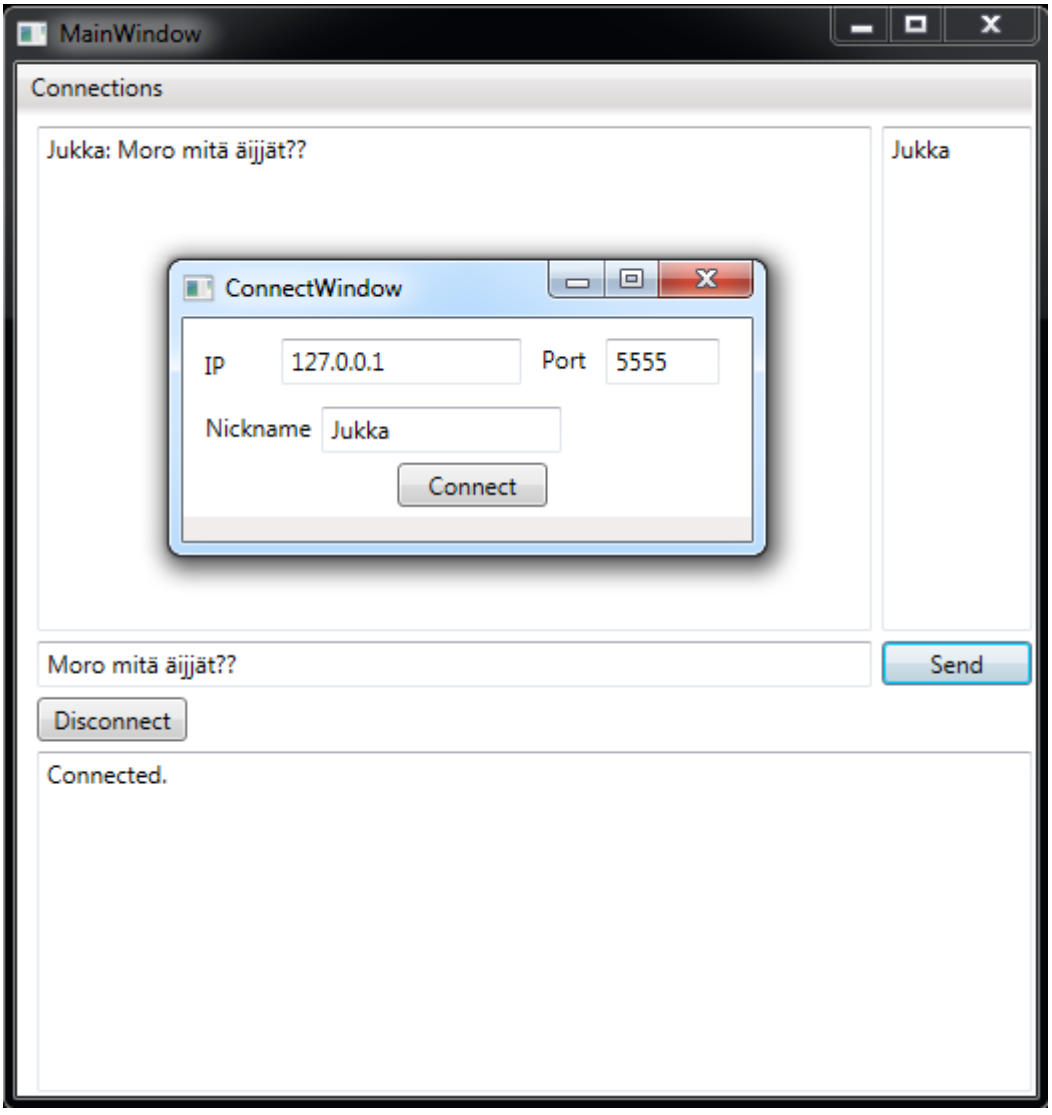
- Tarjota client yhteyksille palvelun, johon voidaan muodostaa tcp socket yhteys
- Nähdä, mistä ip+port yhdistelmästä käyttäjä liittyy

2.2 Suunnitellut, mutta ei toteutetut toiminnallisuudet

Alkuperäiseen suunnitelmaan kuului käyttäjätunnuksien luominen ja tietokantayhteydet, jossa piti muun muassa säilyttää keskustelu logeja. Näitä ei toteutettu ajanpuutteen ja muuttuneiden vaatimusten vuoksi.

Mitä ohjelman jatkokehitykseen tulee, niin olen itse enemmän kiinnostunut esimerkiksi etätyöpöytäyhteyden luomisen muodostavista haasteista kuin tietokantayhteyksistä, joten näitä tuskin toteutetaan ainakaan ihan heti.

3 Client käyttöliittymä



4 Datan lähetys

4.1 Serialisointi

Datan lähetys toteutettiin aluksi pelkästään niin, että ohjelma parsi string muuttujan tavuiksi ja lähetti yhteyden yli serverille. Vaatimusmäärittelyn mukaisen ohjelman pystyy toki toteuttamaan tällä tavalla alusta loppuun, mutta haluttiin tutkia ja löytää jatkokehitykseen sopivia menetelmiä.

Aluksi tutkittiin BinaryFormatter luokkaa. BinaryFormatter luokka tarjoaa seriali- ja deserialisoimiseen kätevät metodit, jotka eivät kuitenkaan toimi sellaisenaan jos niitä käytetään datan lähetykseen verkon yli. BinaryFormatterin serialisointi perustuu vahvasti tyypitykseen ja sen toiminta jäi edelleen hieman hämärän peittoon. Opittiin kuitenkin se, että jos binäärimuotoista serialisointia halutaan käyttää, joudutaan tekemään ylimääräiset Binder luokat, jotka pitävät huolen siitä, että BinaryFormatter osaa seriali- ja deserialisoida luokat oikein, aina.

Esiteltävässä versiossa käytetään XMLSerializer -luokkaa eli XML serialisointia. XML serialisointi on verrattain yksinkertaista ja se on helppo toteuttaa.

4.2 Säikeet ja asynkronisuus

Prosessorin kirjoittaessa dataa puskuriin se ei voi tehdä samalla mitään muuta. Tämä on ominaisuus. Kun halutaan lähettää ja vastaanottaa dataa samanaikaisesti, on puskurointimetodit toteutettava siten, että niiden ajaminen onnistuu lomittain. Yksinkertaisin lähestymistapa olisi luoda jokaiselle datan lähetystä ja vastaanottoa suorittavalle metodille oma System.Thread. Kuten myöhemmin todetaan, tämä on suorituskyvyltään suorastaan surkeaa, joten niin EI tehdä.

Vaikka asiaa tutkittiin hyvinkin tarkasti, tultiin siihen lopputulokseen, että toteutunut ohjelma on niin yksinkertainen, että asynkroniset metodit eivät muodostuneet ongelmaksi.

```
public static ManualResetEvent allDone = new ManualResetEvent(false);

public void StartListening()
{
    TcpListener listener = new TcpListener(System.Net.IPAddress.Any, 5555);
    listener.Start();
    try
    {
        while (true)
        {
            allDone.Reset();
            Console.WriteLine("Waiting for connections...");
            listener.BeginAcceptTcpClient(new
                        AsyncCallback(AcceptCallback), listener);
            allDone.WaitOne();
        }

    }
    catch (SocketException e)
    {
        Console.WriteLine("Socket exeption {0}", e);
    }
    finally
    {
        listener.Stop();
    }
}

public void AcceptCallback(IAsyncResult ar)
{
    allDone.Set();
    var listener = (TcpListener)ar.AsyncState;
    var handler = listener.EndAcceptTcpClient(ar);
    Console.WriteLine(handler.Client.LocalEndPoint.ToString() + " connected.");
    var receiver = new Receiver(listener, handler);
    clients.Add(receiver);
}
```

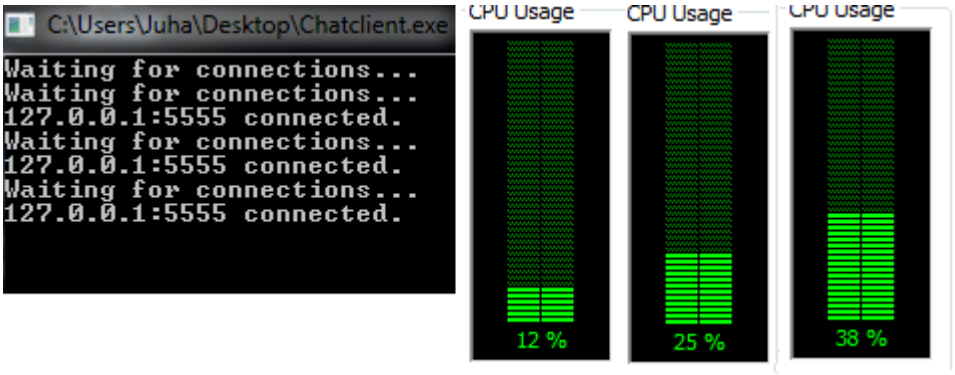
Ohessa yksinkertainen toteutus .NET-kirjaston TcpListener luokalla toteutetusta serveristä, joka kuuntelee TCP yhteyksien luomista ja hoitaa yhdistämisen asynkronisesti. Toteutus käyttää jo vanhahtavaa Begin-End asynkronista ohjelmointitapaa ja ManualResetEvent:ä, joka avulla ajonaikaisesti luotu säie pysäytetään ja uudelleen käynnistetään tarpeen mukaan.

5 Havaintoja

5.1 Säikeistys ja sen onglemat

Jokainen client ja jokainen receiver käynnistää kaksi säiettä, yhden lähetysmetodille ja yhden lukemismetodille. Ohessa nähdään sen vaikutus, ajettaessa samalta prosessorilta sekä serveriä, että useampia clienttejä. Kyseinen lähestymistapa

ohjelman toteuttamiseen on todella huono ja vaihtoehtoisena toteutustapana onkin tutkittu .NETin asynkronisia palautusmetodeja käyttäviä funktioita NetworkStreamin lukemiseen ja kirjoittamiseen.




```
private void SendingMethod()
{
    while (true)
    {
        if (MessageQueue.Count > 0)
        {
            MessageBase msg = MessageQueue[0];
            try
            {
                BinaryFormatter f = new BinaryFormatter();
                f.Serialize(stream, msg);
            }
            catch (IOException e)
            {
                Console.WriteLine("IOException {0}", e);
            }
            MessageQueue.Remove(msg);
        }
        Thread.Sleep(60);
    }
}
```

Asynkronisten palautusmetodien käyttö muodostui toisaalta ongelmaksi jos lähetettävää ja vastaanotettavaa dataa piti serialisoida. .NET kirjastoista löytyy BinaryFormatter –luokka objektien seriali- ja deserialisoimiseen. BinaryFormatterin käyttö asynkronisesti suoraa NetworkStreamin kanssa ei onnistu kovin helpolla, joten sitä ei ainakaan toistaiseksi toteutettu. Yksi tapa olisi toteuttaa serialisointi manuaalisesti, mutta se olisi mielestäni typerää.

Ongelmana edellisessä lähestymistavassa oli se, että jokainen säie pyöri while –silmukassa, nukkui aina 60ms, eikä koskaan pysähtynyt. Tämä kuormittaa prosessoria todella paljon, sillä erilaisia siirtotoimenpiteitä täytyy tehdä todella nopeasti aivan

turhaan. Streamien lukemiseen toteutetut metodit käyttävät paljon korkeampaa teknologiaa, sillä ne pysäyttävät säikeen ja käyttävät lähelle rautaa ohjelmoitua palautusmetodia sen uudelleenkutsumiseen kun luettavaa dataa on saatavilla.

6 Yhteenveto

Alun vaikeuksista huolimatta oli .NET kurssin harjoitustyön tekeminen yllättävän hauskaa. Kurssin sisältö ei omasta mielestäni ollut mitenkään erityisen kiinnostavaa ja olen tyytyväinen, että löysin ja sain toteuttaa työn aiheesta, josta olin oikeasti kiinnostunut.

Opin todella paljon muun muassa tcp protokollasta, socket yhteyksistä, asynkronisuudesta ja siitä miten koodata järkevästi säikeistystä vaativia asioita. Opin Jonkun verran serialisoinnista ja yllättävää kyllä, opin tärkeitä asioita myös perus tietorakenteista ja ylipäätään siitä miten data siirtyy puskurista toiseen. Aloin myös hahmottaa ongelmia, joita syntyy ja joita loppupäässä koodaajan täytyy ottaa huomioon kun dataa siirretään verkon yli ohjelmasta toiseen.