Journal for I4IKN øvelse 8 og øvelse 9

Gruppe 5

Jeppe Benjaminsen	201500154
Søren Holm	201409556
Mikkel Poulsen	20112893
Morten Christensen	201500162

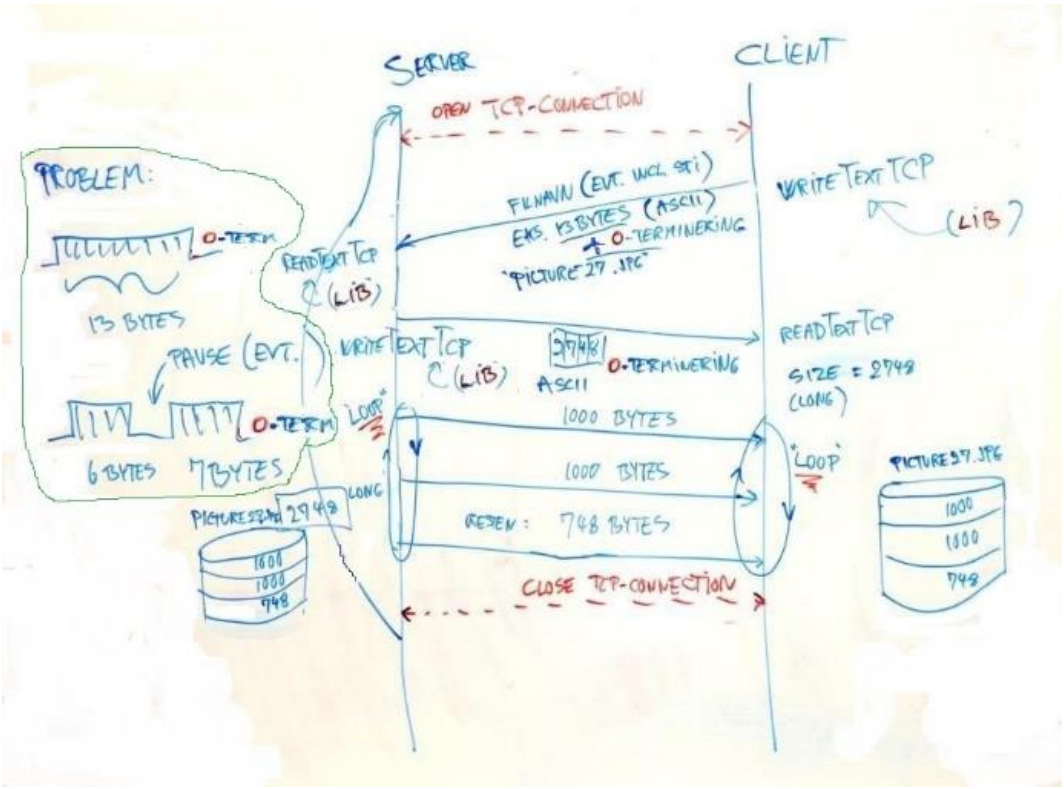
Øvelse 8 – TCP/IP socket programmering

Introduktion

I denne øvelse programmeres en socket til en TCP server og TCP client, og vi har valgt at skrive koden i C#. Det går ud på, at client skal kunne forbinde til server, og downloade en vilkårlig fil herfra. I det efterfølgende beskrives udviklingsforløb og funktionalitet.

Udviklingforløb

Vi delte opgaven med at skrive client og server socket op, så vi havde en 2-mands-gruppe på hver opgave. Vi arbejdede iterativt ud fra skitsen på Figur 1, oppefra og ned, således at vi startede med at etablere TCP-connection, testede den. Dernæst forsøgte vi at requeste en fil, testede dette osv.



Figur 1 Protokol for TCP-overførsel

Funktionalitet

Designet tager udgangspunkt i Figur 1. Først etableres TCP-forbindelsen ved at lave en ny TCP-socket i client:

```
TcpClient NyTcpSocket = new TcpClient ();
NyTcpSocket.Connect (args [0], PORT);
```

Imens står serveren og venter på et TCP-client, som den kan etablere forbindelse med:

```
clientSocket = welcomeScoket.AcceptTcpClient ();
```

Når der er etableret en TCP-forbindelse mellem client og server, kan client requeste om at downloade en vilkårlig fil, enten med eller uden sti:

```
//sæt streameren til at snakke på den nu åbne tcp connection
NetworkStream fileStream = NyTcpSocket.GetStream ();
//Skriv til server at vi ønsker den og den fil.
LIB.writeTextTCP (fileStream, args [1]);
//string ModtagetStatus = LIB.readTextTCP (fileStream);
receiveFile (args [1], fileStream);
//reager på det der kommer tilbage, hvis det ikke er null.
```

Nu leder serveren efter filen, og hvis der ikke er angivet en sti, vil den lede i root som default. Hvis serveren finder filen, vil den returnere størrelsen af filen til client. På den måde kan client vide hvornår overførslen af filen er gennemført. Hvis den ikke finder filen vil den returnere 0, og client ved derfor at filen ikke kunne findes:

```
//Modtager besked fra client
NetworkStream inFromClient = clientSocket.GetStream ();
//Besked læses, og data lægges i bytesFrom
string fileName = tcp.LIB.readTextTCP (inFromClient);

Console.WriteLine ("Client request: " + fileName);

long fileSize = tcp.LIB.check_File_Exists (fileName);

string mySize = fileSize.ToString ();

tcp.LIB.writeTextTCP (inFromClient, mySize);
```

Hvis filen findes vil overførslen påbegyndes. Her sendes filen i masser af små pakker som er op til 1000 bytes stor. Dette sker iterativt inde i sendFile().

```
if (fileSize != 0)
            sendFile (fileName, fileSize, inFromClient);
else
            Console.WriteLine ("File doesn't exist");
```

SendFile() er vist i nedenstående codesnippet. Der oprettes en Filestream, som skal bruges til at tilgå den ønskede fil. I while-løkken sker følgende: Vi læser fra filen, og putter data over i et bytearray på max 1000 bytes (BUFSIZE). Dette sendes afsted til client. Så længe den totale filstørrelse er større end mængden af sendt data, bliver vi ved med at sende pakker af max 1000 bytes.

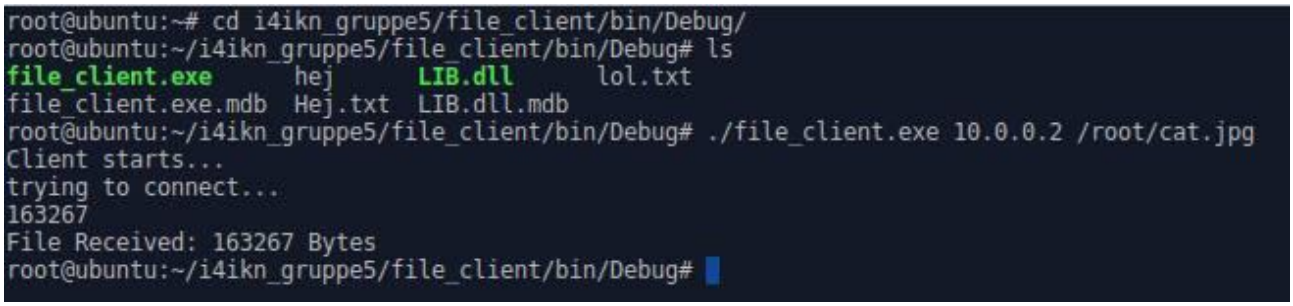
```
private void sendFile (String fileName, long fileSize, NetworkStream io)
{
            int totalSize = (int)fileSize;
            int totalAmountSend = 0;

            byte[] myBytes = new byte[BUFSIZE];
            FileStream Fs = new FileStream (fileName, FileMode.Open,
            FileAccess.Read);

            while (totalAmountSend < totalSize)
            {
                        int bytesRead = Fs.Read (myBytes, 0, BUFSIZE);

                        io.Write (myBytes, 0, bytesRead);
                        totalAmountSend += bytesRead;

                        Console.WriteLine (bytesRead);
            }
            Fs.Close();

}
```

I client er der næsten præcis samme fremgangsmåde, men med omvendt fortegn: I server skrives der fra en fil til en networkstream, mens der i client skrives til en fil fra en networkstream.

Når overførslen er afsluttet lukkes TCP-forbindelsen til client.

Test

På Figur 2 ses et eksempel på en test, hvor vi forsøger at overføre en fil cat.jpg. Det ses på konsoludskriften hvordan klienten først startes, derefter forsøger at oprette forbindelse og requester at hente cat.jpg. Filen findes, så serveren sender tilbage, at filen er 163267 bytes stor, hvorefter overførslen starter. Client udskriver så at filen er modtaget, og hvor mange bytes der blev modtaget.



Figur 2 – Overførsel af billede

Konklusion

Vi kan se at TCP protokollen overfører data pålideligt fra serveren i bider af 1000 bytes. Det er derfor helt tydeligt at TCP anvender handshake mellem client og server.

Øvelse 9 – UDP/IP socket programmering

Introduktion

Der skal nu laves en UDP client og server, som vi vælger også at skrive i C#. Client skal kunne sende en forespørgsel til serveren, og der skal kunne understøttes én enkelt client. Der er mulighed for to forskellige forespørgsler: ”U” for at få information fra */proc/uptime*, og ”L” for */proc/loadavg.*

Funktionalitet

Server

Serveren fungerer iterativt ved hele tiden at vente på forespørgsler fra en klient. Som koden her indikerer så kan det være fra en hvilken som helst klient. Når serveren har fundet til en klient, så gemmes data i et bytearray.

```
ipep = new IPEndPoint (IPAddress.Any, 9000);
socket = new UdpClient (ipep);
byte[] data = new byte[1000];

IPEndPoint sender = new IPEndPoint (IPAddress.Any, 8000);
data = socket.Receive (ref sender);
```

Der modtages en forespørgsel som decodes, for at finde ud af hvad der skal sendes til klienten. Der åbnes en filestream til at læse */proc/loadavg* eller */proc/uptime.* Den information lægges i et bytearray, som sendes tilbage til klienten. Her der bruger man dog ikke *NetworkStream* til at sende data – denne funktionalitet er indbygget i *UdpClient* objektet.
Nu er serveren klar til at modtage en ny forespørgsel.

Klient

Klienten kan sende en forespørgsel til en specifik server ved at indtaste serverens IP samt et ’U’ eller ’L’. I koden er porten hardcoded til 9000.

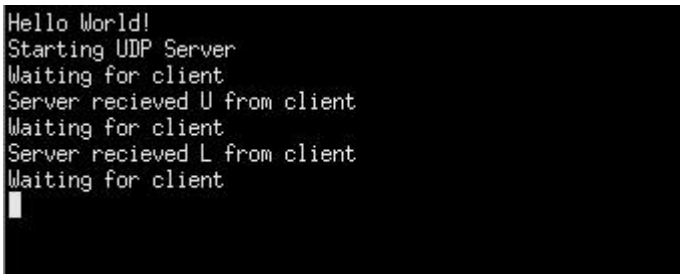
```
UdpClient listener = new UdpClient (Port);
IPEndPoint groupEP = new IPEndPoint (IPAddress.Parse("10.0.0.1"), Port);
```

Forespørgslen encodes til bytes for at kunne sende den. Når forespørgslen er sendt afsted, venter klienten på svar fra serveren. Responsen fra serveren bliver så encodet til en string.

Eksempel på test

Nedenfor kan en test af systemet ses, hvor det ene billede er klient, og det andet er server.

Figur 3 – UDP client



Figur 4 – UDP server

Konklusion

Vi har kodet en UDP client og server. UDP er velegnet til hurtigt men ”usikker” filoverførsel, da der ikke er nogen form for kontrol med, om overførslen gik godt. Det kan også kaldes ”best effort”, da det jo oftest går godt. Fra serverens synspunkt, er det en slags fire and forget, da den er ligeglad med om client modtager filen rigtigt.