Journal for øvelserne 13 for grp. 22

Lavet i samarbejde med grp. 24

Bjørn Nørgaard Sørensen
stud.nr: 201370248
bjornnorgaard@post.au.dk

Joachim Dam Andersen
stud.nr: 201370031
mr.anderson@post.au.dk

Arni Thor Thorsteinsson
stud.nr: 201370318
mr.thorsteinsson@post.au.dk

December 4, 2015

Indholdsfortegnelse

1 Introduktion **1**

2 Protokol **1**
2.1 Application layer 1
2.2 Transport layer 1
2.3 Link layer 3
2.4 RD232 Driver 3
2.5 Server 4
2.6 Client 4
2.7 Test 5

3 Konklusion **5**

List of Figures

1 1
2 2
3 6
4 6
5 7
6 7
7 7
8 7

1 Introduktion

Formålet med denne øvelse er at udarbejde en protokol stack til overførsel af filer via et RS-232 interface. Systemet gør det muligt at overføre en fil fra en virtuel computer til en anden. Hvor den første virtuelle maskine fungerer som client og den anden som server. Clienten meddeler serveren hvilken fil der skal hentes, hvorefter serveren læser og sender denne, i pakker af 1000 bytes af gangen. Clienten modtager så disse og gemmer dem i en fil. Vi bruger i denne øvelse server og clienten fra øvelse 8. Forskellen fra øvelse 8 er at vi denne gang ikke anvender socket-API til at udføre kald, i denne øvelse bruger vi kald til vores transportlag i en protocolstack som vi selv har udviklet. Brugerfladen er dog ens.

2 Protokol

De enkelte funktioner er delt op efter en lagdelt model, som kan ses på figur 1.

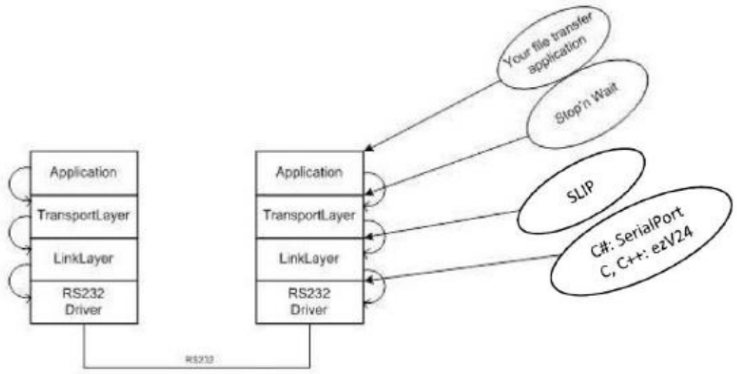


Figure 1

2.1 Application layer

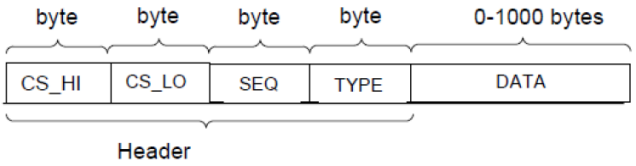
Vores applikationslag fungerer næsten på samme måde som i øvelse 8. Forskellen er dog at vi har udskiftet TCP-kaldet med transportlags kald.

2.2 Transport layer

Transportlaget er implementeret via stop-and-wait protokol. Protokollen anvender en 16-bit checksum. Se på figur 2 at laget overholder følgende segment:

Da vi fra transportlaget ønsker at sende et byte array videre til linklaget, som er større end det array vi modtager fra applikationslaget, indeholder transportlag sit eget buffer array, som er "HeaderSize" større end applikations arrayet. Vi indsætter sekvens i tredje byte og type i fjerde byte. For god ordens skyld initierer vi de to første bytes til 0, men disse bliver overskrevet når vi senere beregner vores checksum. Derefter kopieres arrayet fra applikations laget ind i det nye array. På baggrund af hele det nye array, altså også sekvens nummer og type, beregnes checksummen, og indsættes på de første to pladser. Til sidst sendes det nye array til link laget, og dette fortsættes indtil vi modtager en ack fra modtageren. Det hele ses i funktionen herunder:

```
1 public void send(byte[] buf, int size)
2 {
```



CS_HI:	Den mest betydende del af checksums-beregningen.
CS_LO:	Den mindst betydende del af checksums-beregningen.
SEQ:	Sekvensnummer på den afsendte segment.
TYPE:	0 (DATA) 1 (ACK)
DATA:	Payload på mellem 0-1000 bytes. Skal altid være af længden 0 ved ACK.

Figure 2

```
3       buffer [0] = 0; //checksum high - replaces in calcChecksum
4       buffer [1] = 0; //checksum low - replaces in calcChecksum
5       buffer [2] = seqNo; //seq number
6       buffer [3] = 0; //Type (0 = data / 1 = ack)
7       Array.Copy(buf, 0, buffer, HEADER_SIZE, buf.Length);
8
9       checksum.calcChecksum (ref buffer, size + HEADER_SIZE); //insert checksum
10
11      bool ackReceived = false;
12
13      do
14      {
15          link.send (buffer, size + HEADER_SIZE);
16
17          ackReceived = receiveAck ();
18      } while (!ackReceived);
19  }
```

På modtager siden starter vi med at modtage arrayet fra modtageres link lag. Vi beregner om checksummen stemmer med de overførte data, og sender en ack tilbage baseret på resultatet. Vi kopierer derefter arrayet over i et mindre array for at fjerne headeren. Dette gentages hvis det modtagne data ikke stemte overens med checksummen. Til sidst returneres hvor mange bytes der i alt blev modtaget, minus headerens data som vist herunder:

```
1   public int receive (ref byte[] buf)
2   {
3       int receivedBytes = 0;
4       bool receivedOk = false;
5
6       do
7       {
8           receivedBytes = link.receive (ref buffer);
9           receivedOk = checksum.checkChecksum (buffer, receivedBytes);
10          sendAck (receivedOk);
11          Array.Copy(buffer, HEADER_SIZE, buf, 0, buf.Length);
12      } while(!receivedOk);
13
14      return receivedBytes - HEADER_SIZE;
15  }
```

For at teste om data bliver sendt igen i tilfælde af fejl, har vi lavet følgende test setup hvor der testes med en false ack, samt fejl i checksum:

```
//Test Program
int receivedBytes = link.receive (ref buf);
sendAck (false);
Console.WriteLine ("test - transport sent false ack");

receivedBytes = link.receive (ref buf);
buf [0] = 47;
sendAck( checksum.checkChecksum (buf, receivedBytes));
Console.WriteLine ("test - transport with wrong checksum - !ack sent");

receivedBytes = link.receive (ref buf);
sendAck( checksum.checkChecksum (buf, receivedBytes));
Console.WriteLine ("test - transport with ok checksum - ack sent");

return receivedBytes;
```

På denne måde får vi testet at vores protokol stack overholder 3 af de 4 scenarier fremsat af underviseren. Det sidste scenarie lykkedes desværre ikke at få gennemført. Scenarierne ses på figur 3 på side 6, udleveret af underviser:

2.3 Link layer

I denne øvelse skulle vi udvikle vores egen protokol med start og stop karakter 'A'. Karakter 'A' bliver derfor erstattet med 'BC' og 'B' er erstattes med 'BD' som vist herunder på figur 4.

Her er et eksempel på vores link lag. Her implementeres vores protokol i send funktionen.

```
public void send (byte[] buf, int size)
{
     buffer[0] = 65; //Insert A at start
     int bytesToSendIndex = 1;
     for (int i = 0; i < size; i++)
     {
          if (buf [i] == 65)
          {
               buffer[bytesToSendIndex] = 66;
               bytesToSendIndex++;
               buffer[bytesToSendIndex] = 67;
               bytesToSendIndex++;
          }
          else if (buf [i] == 66)
          {
               buffer[bytesToSendIndex] = 66;
               bytesToSendIndex++;
               buffer[bytesToSendIndex] = 68;
               bytesToSendIndex++;
          }
          else
          {
               buffer [bytesToSendIndex] = buf [i];
               bytesToSendIndex++;
          }
     }
     buffer [bytesToSendIndex] = 65;
     bytesToSendIndex++;
     serialPort.Write (buffer, 0, bytesToSendIndex);
}
```

2.4 RD232 Driver

Her opretter vi det fysiske lag, som I dette tilfælde er vores RS-232-port og vores null-modem.

```
public Link (int BUFSIZE)
```

```
{
    // Create a new SerialPort object with default settings.
    serialPort = new SerialPort("/dev/ttyS1",115200,Parity.None,8,StopBits.One);

    if(!serialPort.IsOpen)
        serialPort.Open();

    buffer = new byte[(BUFSIZE*2)+2];
}
```

2.5 Server

På vores server opretter vi et nyt transportlag med den ønskede bufsize når den startes. Derefter kaldes fileserver med dette transportlag hvorefter der ventes på at modtage en forespørgsel på serveren. Når denne modtages tjekker vi om filen eksisterer, hvis denne findes sendes denne.

```
private file_server (Transport transport)
{
    try
    {
        Console.WriteLine("Server started -- Listening for client");

        //Get filename

        string fileToSend = transport.readText();

        Console.WriteLine("Client trying to retrieve file: " + fileToSend);

        long fileSize = LIB.check_File_Exists (fileToSend);

        if (fileSize != 0)
        {
            transport.sendText("File found");

            sendFile (fileToSend, fileSize, transport);
        }
        else
        {
            transport.sendText("File not found");
            Console.WriteLine ("404 -- File not found");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine (ex.Message);
    }
    finally
    {
        Console.WriteLine ("Exit");
    }
}
```

2.6 Client

På vores klient opretter vi et nyt transportlag med den ønskede bufsize når den startes. Herefter kaldes fileclient med filnavnet på fil den ønskes hentes, samt hvilket transportlag der ønsker kommunikeres over. Et svar modtages om størrelsen på filen modtages og herefter begynder clienten at modtage og gemme filen.

```
private file_client(String[] args, Transport transportStream)
{
```

```
3       try
4       {
5           Console.WriteLine ("Retrieving file");
6
7           string fileToReceive = (args.Length > 0) ? args[0] : "billede.jpg";
8
9           transportStream.sendText(fileToReceive);
10
11          //Read confirmation that file exists
12
13          if (transportStream.readText() == "File found")
14          {
15              receiveFile (fileToReceive, transportStream);
16          }
17          else
18          {
19              Console.WriteLine ("404 - File not found");
20          }
21      }
22      catch (Exception ex)
23      {
24          Console.WriteLine ("Generel exception occured");
25          Console.WriteLine(ex.Message);
26      }
27      finally
28      {}
29  }
```

2.7 Test

Serveren startes og venter på client.
Client sender hvilken fil der ønskes hentet.
Serveren modtager besked fra client og begynder at sende filen.
Client begynder at modtage filen fra serveren.

3 Konklusion

Det praktiske forløb stemte overens med vores teori.

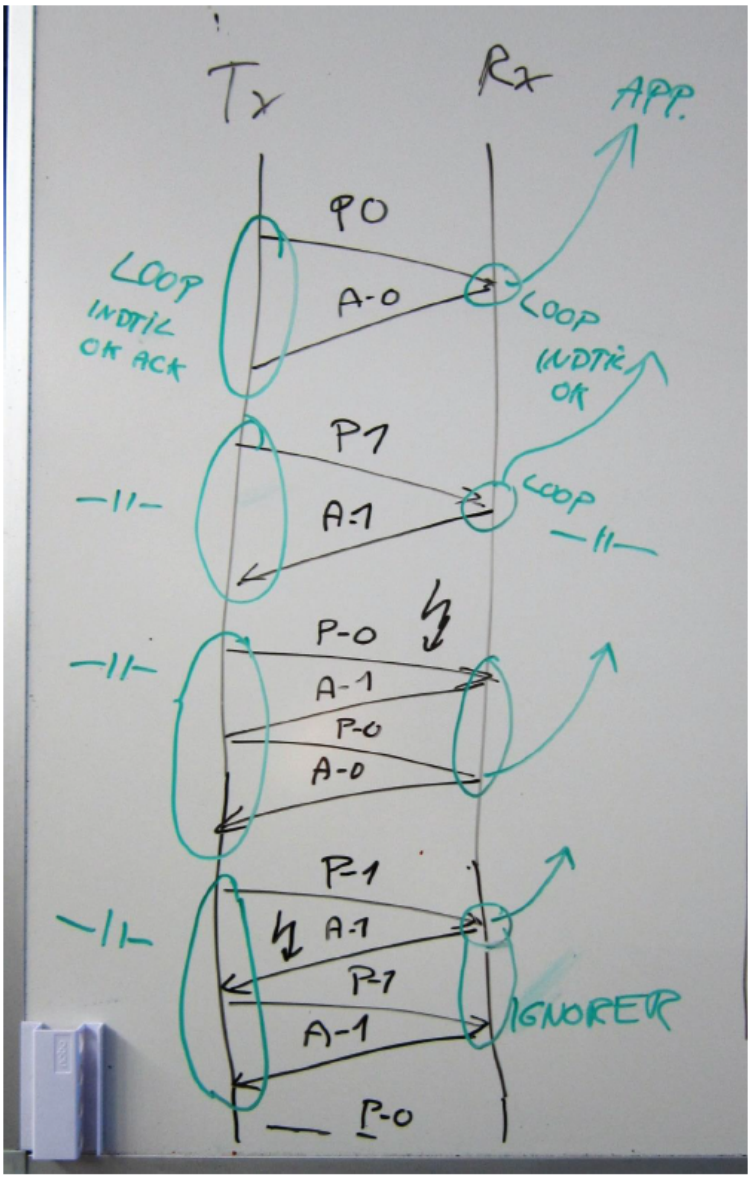


Figure 3

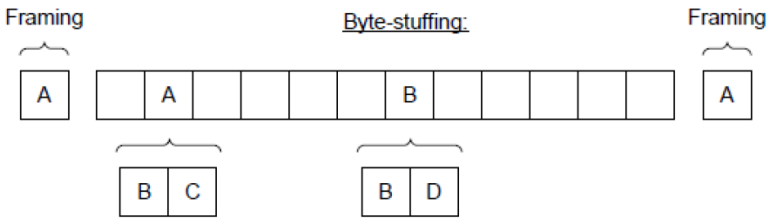


Figure 4

Figure 5

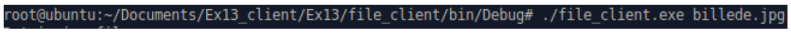


Figure 6

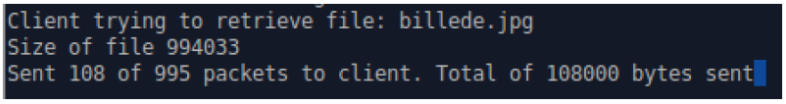


Figure 7

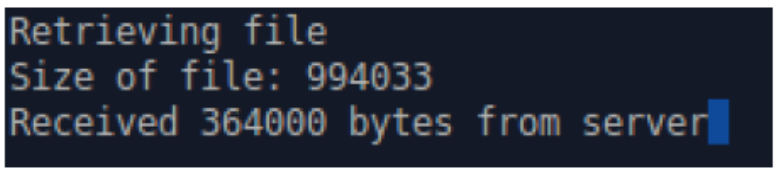


Figure 8