



IKN ØVELSE 13

FIL OVERFØRSEL

VIA RS232

Udvikling af protokol-stack

Deltagere

Kenn Hedegaard Eskildsen

-201370904

Kasper Behrendt

-20071526

Karsten Schou Nielsen

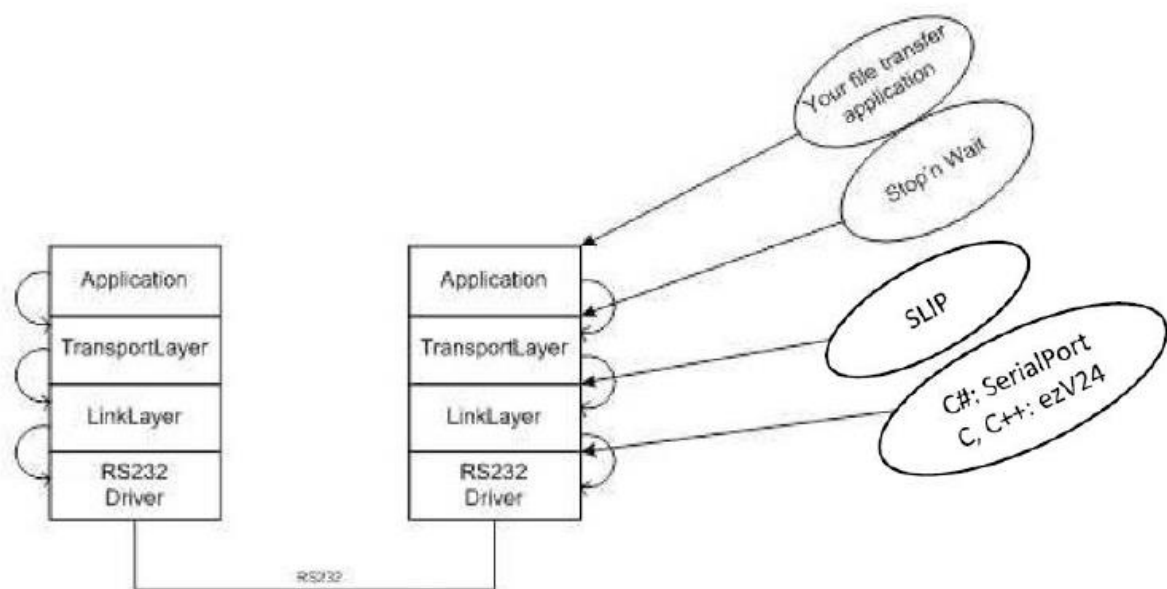
-201370045

Indhold

Indledning.....	2
Linklayer.....	3
Transportlayer	4
Applikationslag	6
Sekvensdiagrammer	7
Test	9

I denne øvelse skal designs og implementeres mulighed for at overføre en fil vha. den serielle kommunikations-port i en virtuel maskine. Det serielle interface er i denne øvelse et RS-232 interface. Microcontroller-baseret embedded udstyr har ofte kun mulighed for at kommunikere med omverdenen via et serielt interface. Derfor er problematikken i denne øvelse relevant. Det er yderligere yderst lærerigt at udvikle en protocol stack, hvilken er hovedformålet med øvelsen.

Client og server kan også have samme brugerflade som client-/server-applikationerne i øvelse 8 (TCP/IP-baseret filoverførsel).

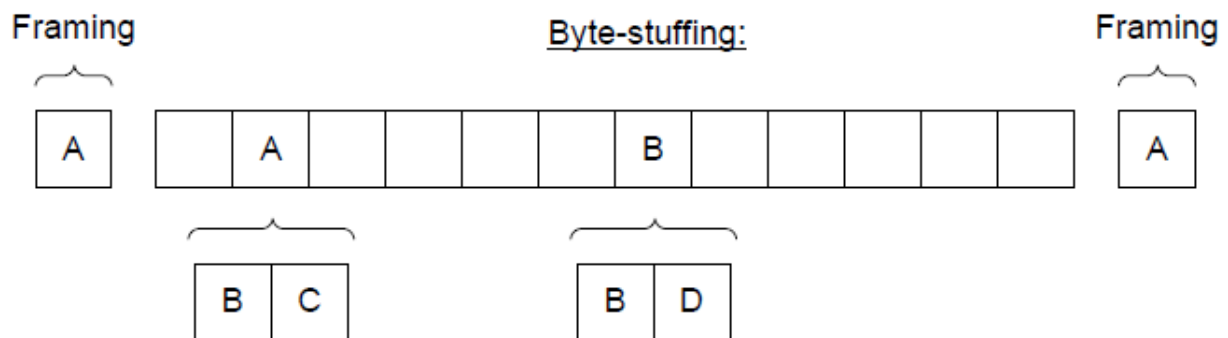


Figur 1: Oversigt over protokollag

Linklag

Linklayeret implementeres med en SLIP-protokol.

SLIP protokollen laver 'A' om til "BC" og 'B' om til "DB". På denne måde kan linklaget modtager-funktionen benytte 'A' som start og stop bit og derved vide hvor lang transmissionen er.



Figur 2: SLIP-protokol

Sender-funktionen er den nemmeste at implementere. Dette gøres ved blot at løbe arrayet igennem med data, som der ønskes sent og spørge efter 'A' eller 'B'. Der oprettes et nyt array, som er dobbelt så stort for at gøre plads til at alle bytes kan være 'A'. Herved menes at hvis et array som sendes alle indeholder 'A' skal hele arrayet laves om til "BC" og derved være dobbelt så stort.

```
for (i = 0; i < size; i++)
{
    if (!(buf[i] == 'A' || buf[i] == 'B'))
        buffer[i + a] = buf[i];

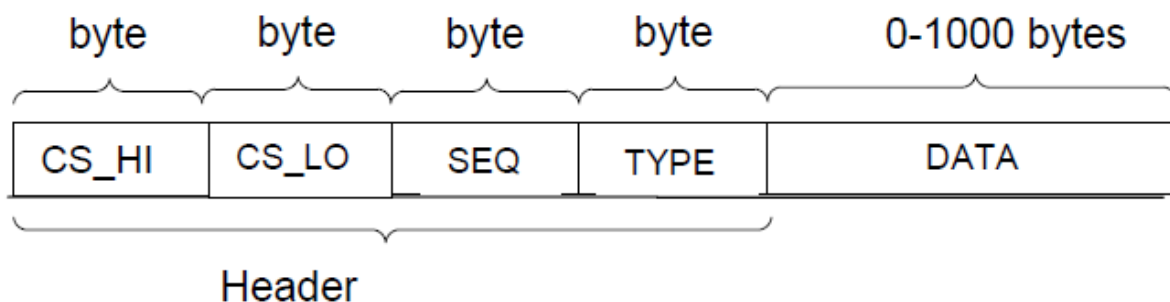
    else if (buf[i] == 'A')
    {
        buffer[i + a] = 'B';
        buffer[i + a + 1] = 'C';
        a++;
    }
    else
    {
        buffer[i + a] = 'B';
        buffer[i + a + 1] = 'D';
        a++;
    }
}
```

Modtager-funktionen er lidt mere kompleks. Da den skal gøre arrayet mindre og returnere størrelsen på det oprindelige data. Her testes for "BC" og "BD" findes de lavet de om til enten 'B' eller 'A'.

```
int a = 0;
for (int i = 0; i < size; i++)
{
    if (!(buffer[i + a] == 'B' && buffer[i + 1 + a] == 'C') && !(buffer[i + a] == 'B' &&
buffer[i + 1 + a] == 'D') && !(buffer[i + a] == 'A'))
        buf[i] = buffer[i + a];
    else if (buffer[i + a] == 'B' && buffer[i + 1 + a] == 'C')
    {
        buf[i] = 'A';
        a++;
    }
    else if (buffer[i + a] == 'B' && buffer[i + 1 + a] == 'D')
    {
        buf[i] = 'B';
        a++;
    }
}
```

Transportlag

Transportlaget implementeres som en stop-and-wait protokol. Det vil sige at klienten står og læser indtil at der kommer noget fra serveren. Desværre har vi ikke implementeret en timeout hvilket resulterer i nogle problemer som vi vil beskrive senere i applikationslaget. Når transport laget sender dens data, sendes der en 16bit checksum med samt sekvensnummer (1 eller 0) samt type (DATA eller ACK)



Figur 3: Transport lag

Sender-funktionen står for at udregne en 16 bit checksum samt sikre det korrekte sekvensnummer og type.

Når senderfunktionen modtager en buffer kopierer den, den over i en ny, hvor de første 4 pladser er reserveret til vores header. Herved kan checksum instansen modtage pointeren til den nye buffer og skrive checksummen til de reservede pladser. Når checksummen er udregnet sendes hele pakken og der ventes på et ACK. Kommer der et NACK sendes pakken igen.

```
void Transport::send(char buf[], short size)
{
    memcpy(this->buffer + ACKSIZE, buf, size);

    buffer[SEQNO] = seqNo;
    buffer[TYPE] = DATA; //type = data
    checksum->calcChecksum(buffer, size + ACKSIZE);
}
```

```

link->send(this->buffer, size + ACKSIZE); //header + data

while (!receiveAck())
{
    std::cout << "Error did not receive ACK in transportlayer\r\n";
    link->send(this->buffer, size + ACKSIZE); //header + data
}

std::cout << "ACK received in transportlayer\r\n";

return;
}

```

Modtager-funktionen modtager i en buffer som sendes med som parameter, samt den maksimale størrelse der ønskes modtaget. Funktionen starter med at læse fra linklaget og teste om checksummen fra dataen er ens med den i headeren. Er den ikke sendes der et *NACK* og dette gøres indtil der modtages en korrekt datasekvens. Er checksummen ens sendes der et *ACK* og pakken er herved modtaget.

```

short Transport::receive(char buf[], short size)
{
    short bytesRead;

    memset(this->buffer, 0, sizeof(this->buffer));

    bytesRead = link->receive(this->buffer, size + ACKSIZE);

    while (!checksum->checkChecksum(this->buffer, bytesRead))
    {
        std::cout << "Error in checksum\r\n";
        std::cout << "Bytes read: " << bytesRead - ACKSIZE << "\r\n";
        sendAck(false);
        bytesRead = link->receive(this->buffer, size + ACKSIZE);
    }
    sendAck(true);
    std::cout << "Checksum Ok\r\n";
    memcpy(buf, buffer + ACKSIZE, bytesRead - ACKSIZE);
    return bytesRead - ACKSIZE;
}
}

```

Applikationslag

Applikationslaget sørger for at sende og modtage vores filer. Funktionen sendfile sørger for at få en fil korrekt afsendt.

Funktionen starter med at åbne for den fil der skal sendes.

```
FILE *fp = fopen(buf, "r");
if (fp == NULL)
{
    printf("\nError opening file\n");
    return;
}
```

Når det er gjort deles den op i stykker af maksimal BUFSIZE.

```
// Data brydes op i BUF_SIZE stykker
char buff[BUFSIZE] = { 0 };

int nread = fread(buff, 1, BUFSIZE, fp);
printf("Bytes read %d \n", nread);

// Hvis læsning lykkes afsendes filen
if (nread > 0)
{
    printf("Sending \n");
    transportlayer->send(buff, nread);
}
printf("File sent \n");
```

Transportlaget sørger nu for sikker transport af filen. Vi kan dog risikere at programmet står og blokerer hvis der ikke modtages ACK fra klienten.

I modtagerfunktionen startes det også med at åbne den fil der vil modtages i. Eksisterer den ikke oprettes den.

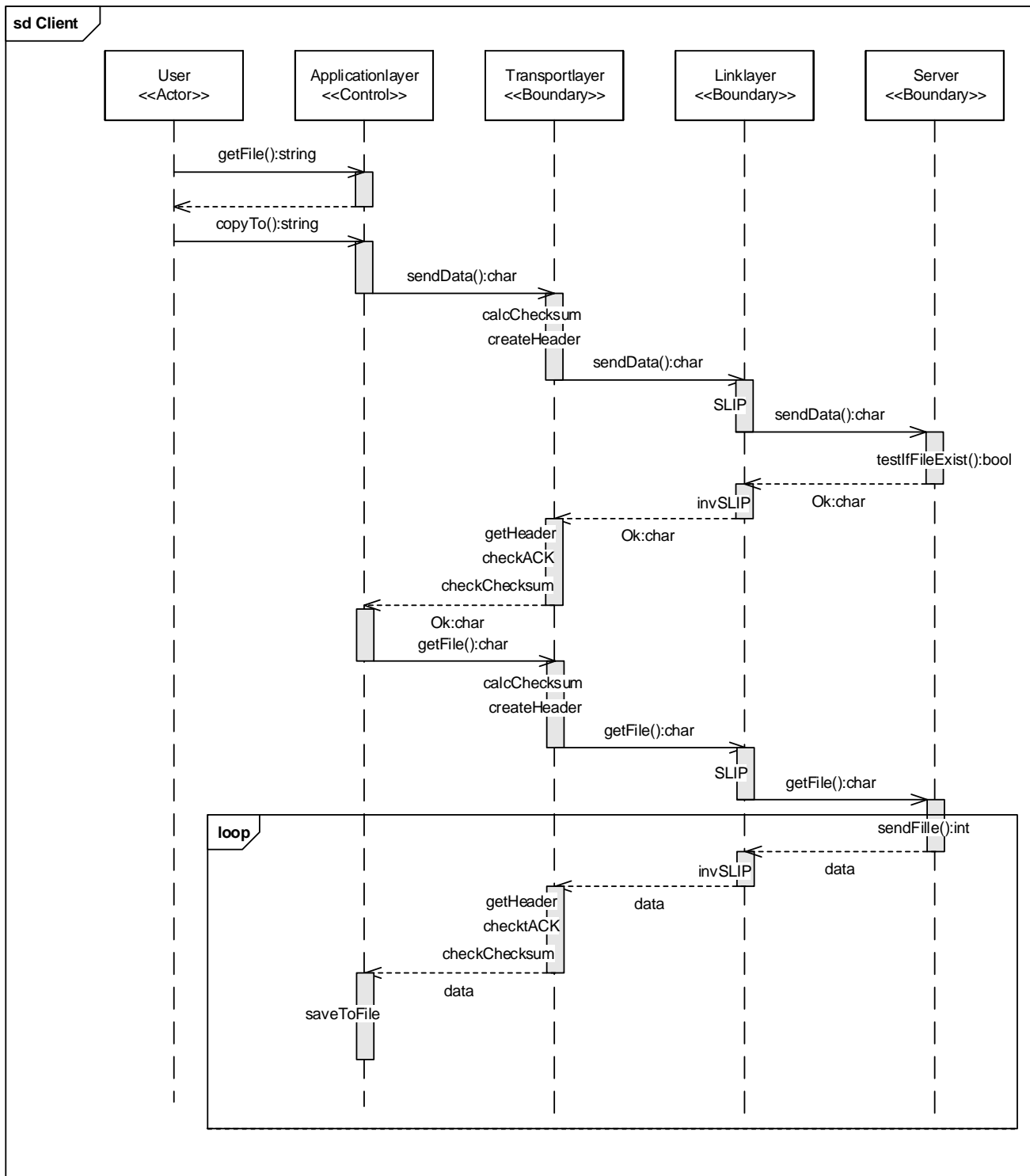
```
// Opret en fil hvor dataen vil blive modtaget
int fd = open(fileName, O_RDONLY);
if (NULL == fd)
{
    cout << "Error creating file";
    return;
}
```

Når filen skal modtages åbner vi fra "end of file" og skriver den modtagende data og lukker igen. Dette gøres da der ikke er implementeret en timeout i transportlaget. Dette vil resultere i at vi aldrig kommer ud at while-løkken og derved aldrig for lukker for filen. Vi vil derfor se at den sidste sekvens som der modtages aldrig bliver gemt i filen.

```
do
{
    fd = open(fileName, "a");
    bytesReceived = transportlayer->receive(buff, BUFSIZE);
    cout << "Bytes received: " << bytesReceived << endl;
    n = write(buff, sizeof(char), bytesReceived, fd);
    close(fd);
}
while (bytesReceived);
```

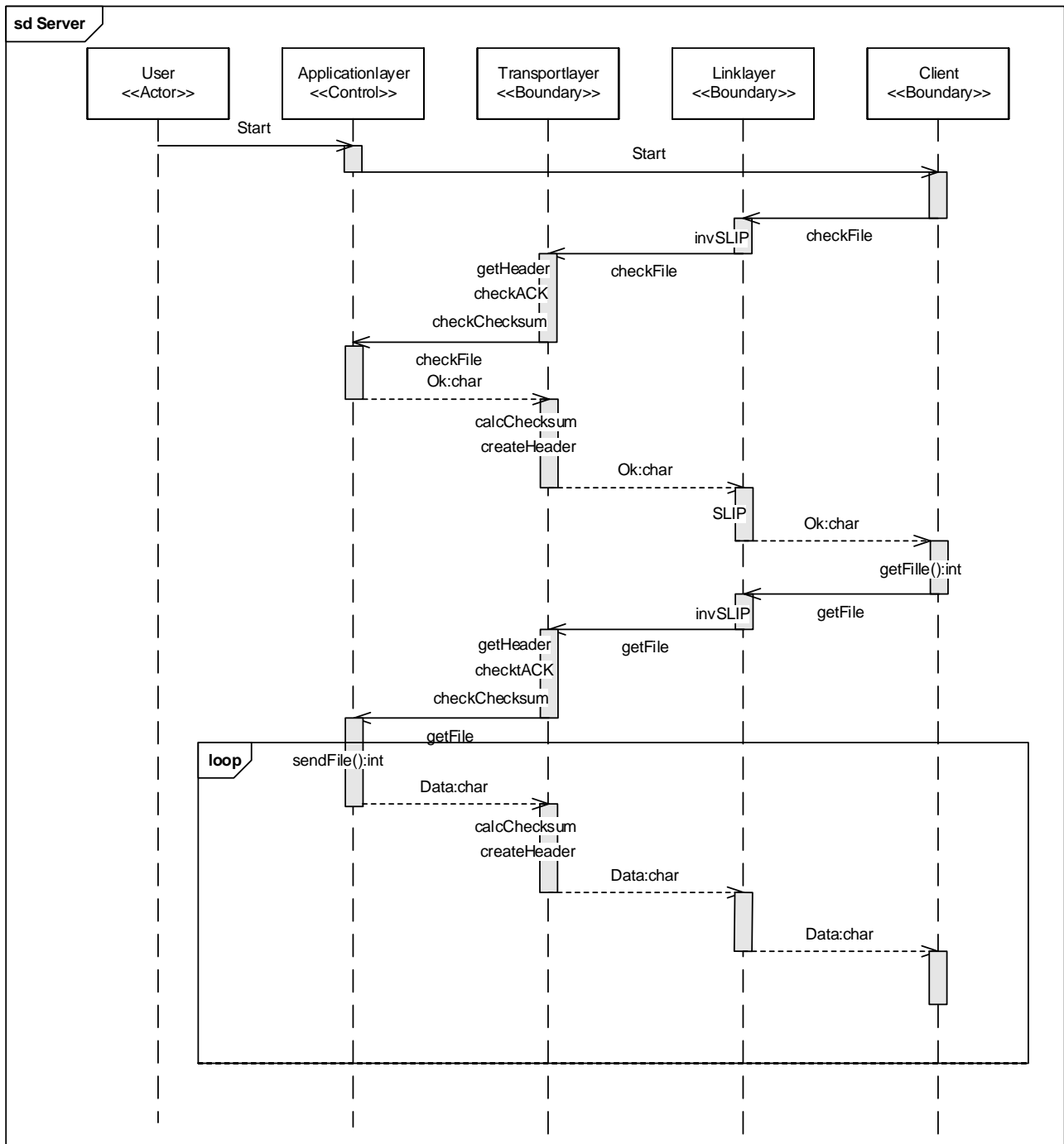
Sekvensdiagrammer

Nedenfor ses sekvensdiagrammet over klienten.



Figur 4: sekvensdiagram over klient

Nedenfor ses sekvensdiagrammet over serveren.

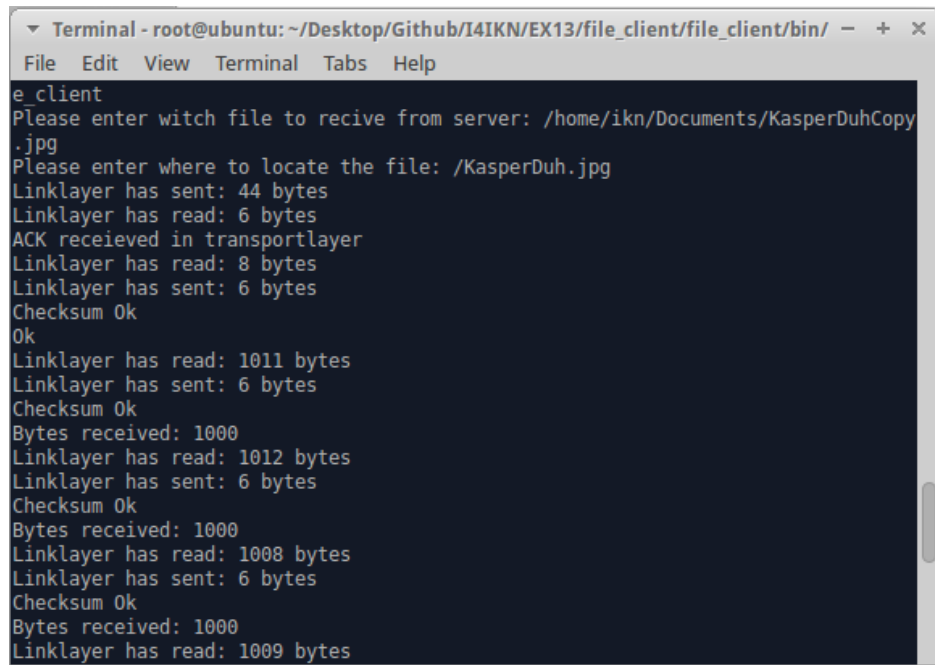


Figur 5: Sekvensdiagram over server

Test

Vi prøver at overføre et billede fra serveren til klienten for at se om vi kan kopiere store data over.

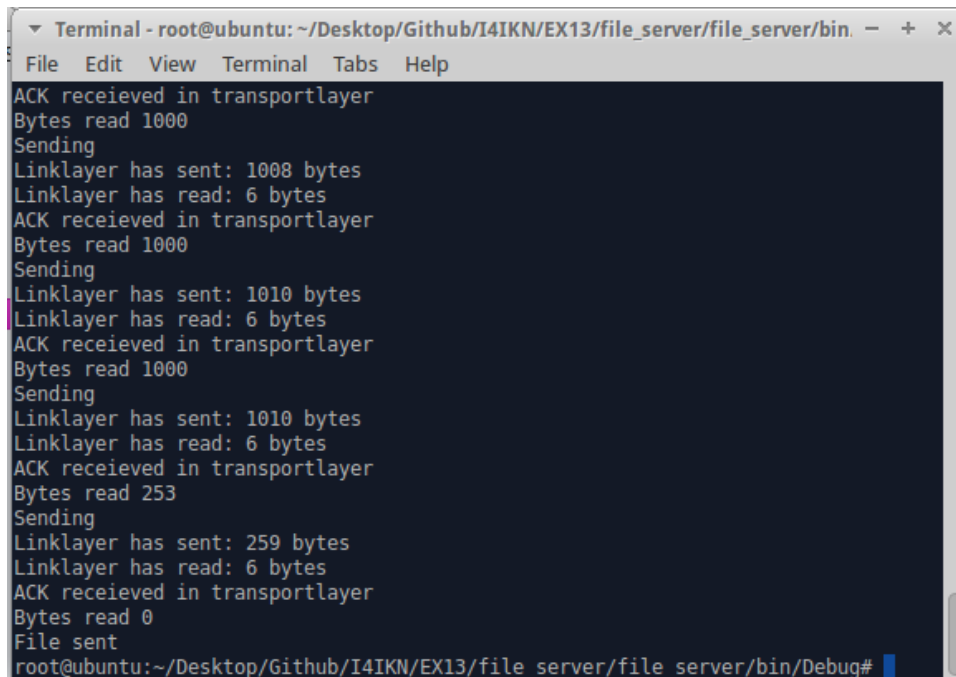
Filen er lokaliseret i **/home/ikn/Documents/KasperDuhCopy.jpg** på server siden og den kopieres over i **/KasperDuh.jpg** på klient siden.



```
Terminal - root@ubuntu: ~/Desktop/Github/I4IKN/EX13/file_client/file_client/bin/
File Edit View Terminal Tabs Help

e_client
Please enter witch file to recive from server: /home/ikn/Documents/KasperDuhCopy
.jpg
Please enter where to locate the file: /KasperDuh.jpg
Linklayer has sent: 44 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Linklayer has read: 8 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Ok
Linklayer has read: 1011 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Bytes received: 1000
Linklayer has read: 1012 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Bytes received: 1000
Linklayer has read: 1008 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Bytes received: 1000
Linklayer has read: 1009 bytes
```

Figur 6: Filen som ønskes modtaget skrives til klienten



```
Terminal - root@ubuntu: ~/Desktop/Github/I4IKN/EX13/file_server/file_server/bin.
File Edit View Terminal Tabs Help

ACK receieved in transportlayer
Bytes read 1000
Sending
Linklayer has sent: 1008 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Bytes read 1000
Sending
Linklayer has sent: 1010 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Bytes read 1000
Sending
Linklayer has sent: 1010 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Bytes read 253
Sending
Linklayer has sent: 259 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Bytes read 0
File sent
root@ubuntu:~/Desktop/Github/I4IKN/EX13/file_server/file_server/bin/Debug#
```

Figur 7: Serveren har sendt filen

Filen åbnes og det ses at billedet er intakt.



Figur 8: KasperDuh.jpg

Vi fremprovokerer en fejl ved afsendelse nummer 5 i serveren. På den måde tester vi at vi får en genafsendelse.

```
// Fremprovokeret fejl
errorCount++;
if(errorCount == 5)
    buffer[0]++;

link->send(this->buffer, size+ACKSIZE); //header + data

if(errorCount==5)
    buffer[0]--;
```

På [figur 9](#) ses test af fejloverførsel.

```
Terminal - root@ubuntu: ~/Desktop/Github/I4IKN/EX13/file_server/file_server/bin. - + x
File Edit View Terminal Tabs Help
ACK receieved in transportlayer
Bytes read 1000
Sending
Linklayer has sent: 1012 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Bytes read 1000
Sending
Linklayer has sent: 1008 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Bytes read 1000
Sending
Linklayer has sent: 1009 bytes
Linklayer has read: 6 bytes
Error did not receive ACK in transportlayer
Linklayer has sent: 1009 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
Bytes read 1000
Sending
Linklayer has sent: 1010 bytes
Linklayer has read: 6 bytes
ACK receieved in transportlayer
```

Figur 9: Server-Retransmission ved pakke nummer 5

```
Terminal - root@ubuntu: ~/Desktop/Github/I4IKN/EX13/file_client/file_client/bin/ - + x
File Edit View Terminal Tabs Help
ACK receieved in transportlayer
Linklayer has read: 8 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Ok
Linklayer has read: 1011 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Bytes received: 1000
Linklayer has read: 1012 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Bytes received: 1000
Linklayer has read: 1008 bytes
Linklayer has sent: 6 bytes
Checksum Ok
Bytes received: 1000
Linklayer has read: 1009 bytes
Error in checksum
Bytes read: 1000
Linklayer has sent: 6 bytes
Linklayer has read: 1009 bytes
Linklayer has sent: 6 bytes
Checksum Ok
```

Figur 10: Klient- Error i checksum i pakke 5

Konklusion:

Øvelsen forløb som forventet, dog havde vi en del problemer med at sende flere pakker over linklaget, hvor sidst afsendte pakke altid gav "Error in checksum", det viste sig at være en fejl i linklagets *reseive()* hvor der blev skrevet til en buffer der var for lille, samt at arrayet blev gennemløbet med fast størrelse på *size*, dette medførte at checksummet blev forkert beregnet. Efter løsning af dette problem, kunne vi gennemføre test , samt fremprovokation af fejl i transportlaget.