IKN Exercise 11

# Introduktion

Denne øvelse går ud på at lave en server og klient, der kan sende sende og modtage en vilkårlig fil. Kommunikationen skal ske gennem serielporte. Gruppen skal implementere Linklayer, Transportlayer og Applicationlayer. Det fysiske lag er allerede implementeret i form af to RS-232 porte via et null modem.

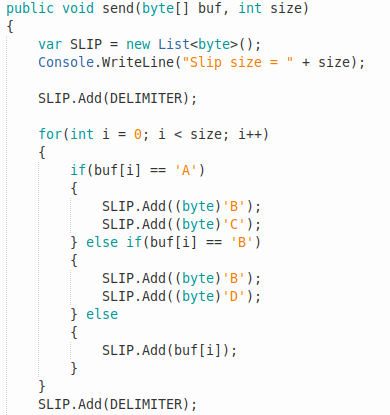
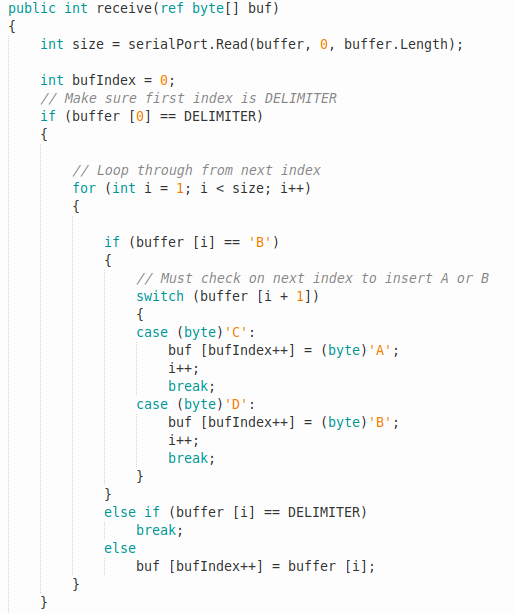
**Smid lidt mere intro på måske.**

# Loopback porte

For at kunne teste overførslerne på en virtual machine, er der brugt loopback porte som simulerer null modem. Da de udleverede loopback porte ikke fungerer, bruges der i stedet et kernemodul med navnet tty0tty[[[1]](#footnote-1)]. For at bygge dette modul skal en dependency ændres. Dette gøres i tty0tty.c hvor ” #include <linux/sched/signal.h>” skal indsættes, og så kan modulet bygges og indsættes. Hver gang maskinen er genstartet skal modules bygges og indsættes på ny. Derefter er det muligt at køre loopback på portene: /dev/tnt0 og /dev/tnt1 med flere.

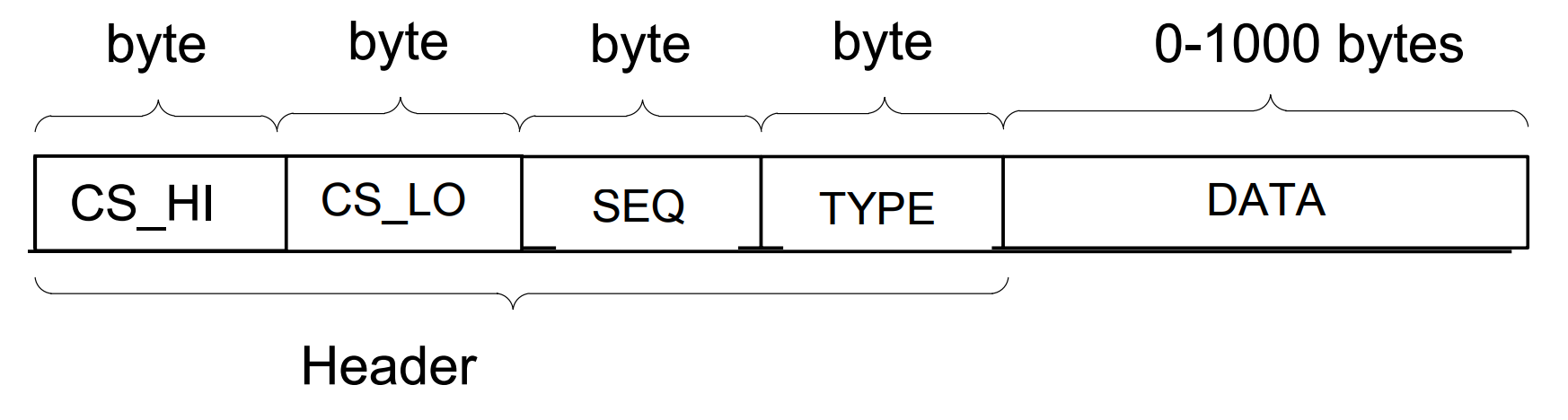
# Link layer

Dette lag ligger lige over det fysiske lag. I dette lag implementeres en SLIP protokol der bruges til at frame en datapakke. Denne pakke bliver framet ved hjælp af to delimiters, der i dette tilfælde er karakteren ’A’. Da denne karakter nu er delimiter, skal alle A’er der findes inde i den framede pakke laves om til ”BC”, og da ’B’ nu bruges til at lave et ’A’ skal ’B’ så udskiftes med ’BC’. Når denne framing er fuldført skal linklaget sende datapakken over det fysiske lag, og derefter skal det igen modtages hos en anden bruger og det modtaget data skal ud af framen og de omdannede karakterer skal laves om til det de var da de blev sendt. I gruppens implementering er en smule fejlhåndtering implementeret, for at løse et problem med en meget fejlfuld overførsel mellem to virtual machines. Hvis modtaget data ikke begynder med delimiteren, så kan hele denne datamængde ikke frames korrekt, og derfor returnerer linklaget med værdien -1.

# Transport layer

I transportlaget implementeres en stop-and-wait protokol som sikrer pålidelig overførsel. Det vil sige at den data der modtages skal være fejlfri og komme i korrekt rækefølge. Dette sikres ved at der på afsenders side udregnes en checksum baseret på dataen der sendes afsted. Denne checksum sendes med i datapakken. På modtagers side udregnes en checksum igen, baseret på den modtagne data (minus den modtagne checksum) og denne sammenlignes med den modtagne checksum. Stemmer disse ikke overens er der sket en fejl under overførslen, og en NAK besked skal derfor sendes tilbage til afsender så dataen kan sendes igen. Stemmer checksummene overens, så sendes en ACK besked tilbage til afsender, og mere data kan herefter sendes til modtageren.



## Sender

I gruppens implementering af transportlagets sender laves der et bytearray som er datapakken der skal sendes afsted. I denne sættes SEQ til det aktuelle sekvensnummer for pakken, og TYPE sættes til 0 for at vise at det er en data transmission og ikke en ACK. Så kopieres dataen ind i bagefter headeren. Efter dette udregnes checksum og indsættes i de første to bytes. Ønskes en simuleret datafejl ændres checksummen her. Derefter sendes datapakken til link laget.

Efter afsendelsen ventes der på en ACK besked. Hvis denne modtages korrekt (har en rigtig checksum) og indeholder det samme sekvensnummer som datapakken der senest blev afsendt, så har modtageren modtaget datapakken korrekt. Er der fejl i checksummen eller er sekvensnummer forkert, sendes seneste datapakke afsted igen. Hvis der ikke er modtaget en ACK besked inden for et vist tidsrum, så sendes datapakken også afsted igen.

## Modtager

I implementeringen af transportlagets modtager læses der først data fra link laget. Checksummen udregnes og sammenlignes med modtaget checksum. Stemmer denne checksum ikke overens, sendes en NAK besked (en ACK men med et forkert sekvensnummer). Er checksummen korrekt, så tjekkes sekvensnummeret. Er dette det samme som senest modtaget datapakke, ignoreres dataene og en ACK sendes tilbage. Er sekvensnummeret det næste i rækken, så sendes ACK beskes, og dataene returneres til applikationslaget. Modtages værdien -1 fra linklaget, så venter modtageren i 250 ms. Dette sikrer at en hel frame er sendt over, modtageren sender en NAK, og dermed bliver framen reset.

# Application layer

I applikationslaget implementeres selve applikationen der skal bruge den pålidelige seriel kommunikation fra de andre lag. I dette tilfælde skal dette være filserver og filklient. Filserveren skal modtage et filnavn, og sende filen til filklienten. Filklienten skal sende filnavnet og derefter modtage og gemme filen. Et lignende program er implementeret i øvelse 6, men dette skete over en TCP IP forbindelse, og nu over en seriel overførsel som gruppen selv har implementeret.

## Server

I gruppens implementering af filserveren modtages der et filnavn gennem transportlaget. Når denne er modtaget checkes der om en fil med det modtagne filnavn findes i mappen: ”/root/ServerData/”. Hvis filen findes sendes filens størrelse afsted til klienten. Findes filen ikke sendes en filstørrelse med værdien 0 afsted, og serveren venter på at modtage et nyt filnavn.

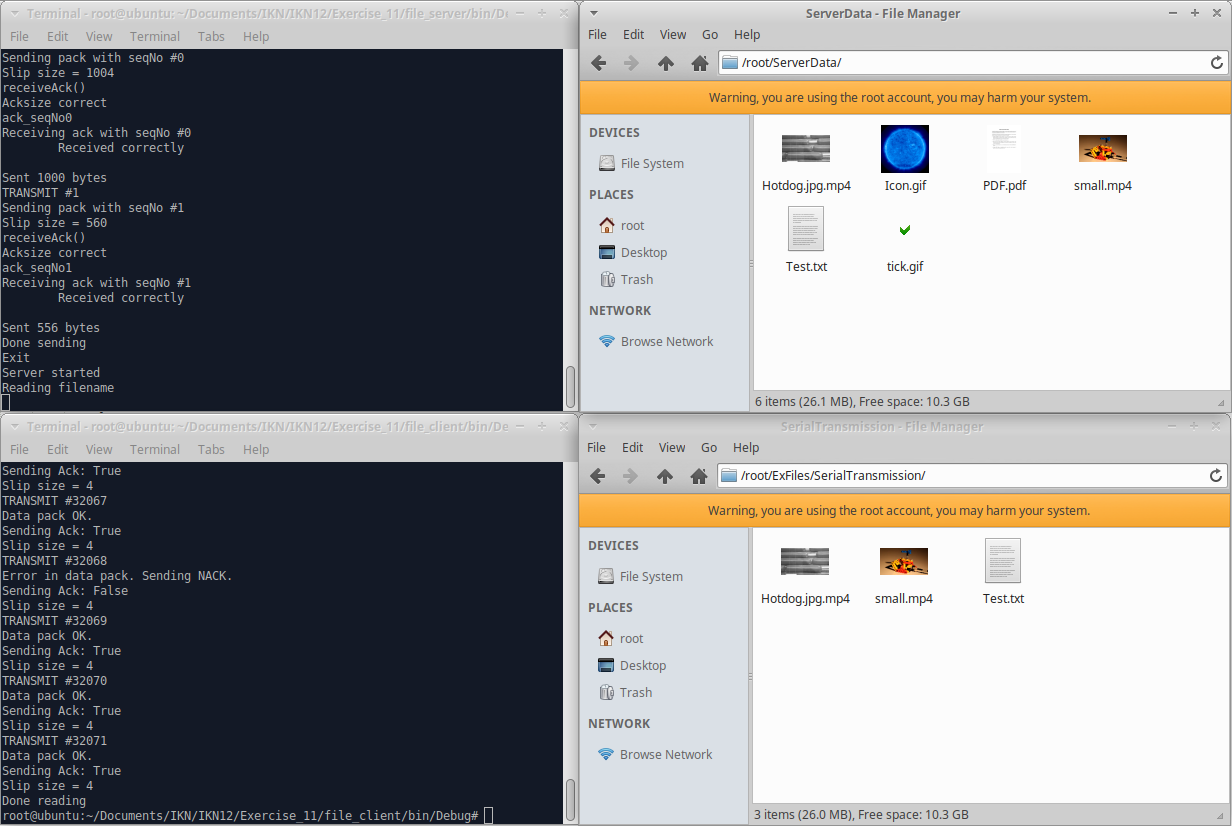
Når et korrekt filnavn er modtaget, og filstørrelse er sendt, indlæses filen i en buffer i bider på maks 1000 bytes. Hver bid sendes så gennem transportlaget indtil hele filen er sendt til klienten. Serveren resetter sine variabler og er derefter klar på at modtage et filnavn igen.

## Klient

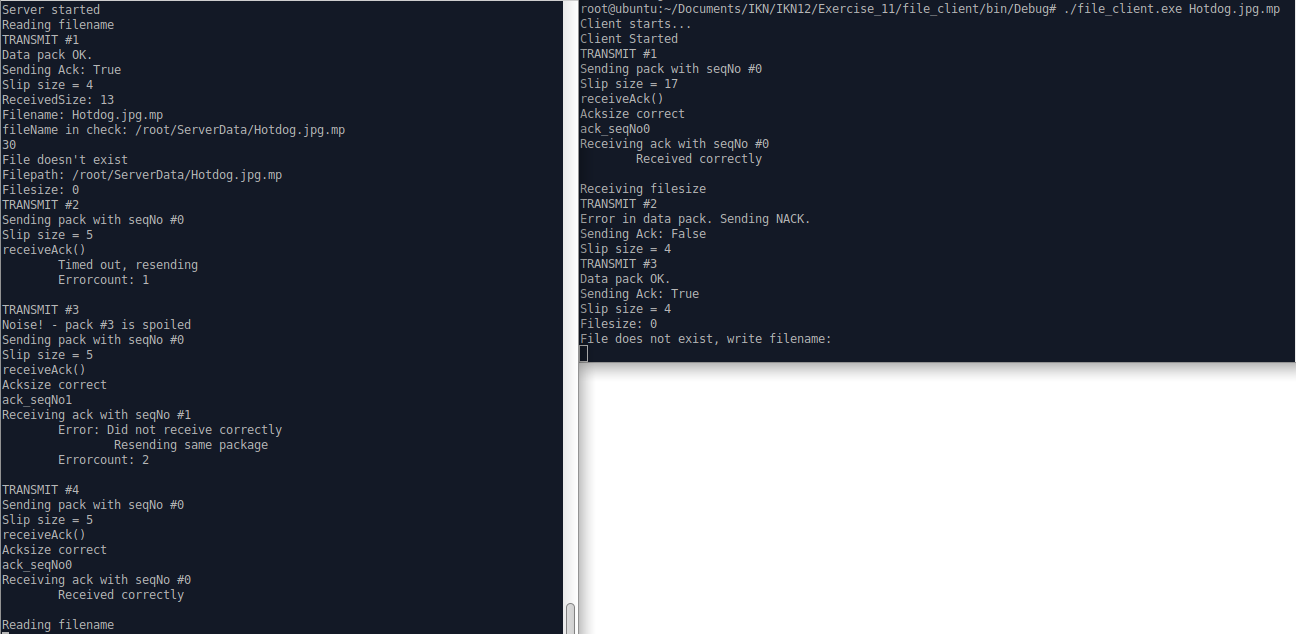
I gruppens implementering af filklienten tages et filnavn som startargument. Gives der ikke et startargument, udskrives en besked og programmet lukker. Er der et filnavn som argument, sendes dette over til filserveren gennem transportlaget, og herefter modtages en filstørrelse fra serveren. Er filstørrelsen på 0, bedes brugeren om at skrive et nyt filnavn og dette sendes derefter over til serveren. Er filstørrelsen større end 0, startes en filestream der vil gemme den modtagne data i en fil. Klienten modtager data fra serveren i bider af maks. 1000 bytes, og gemmer disse data i en fil med samme navn som filen hedder på serveren. Når klienten har modtaget en datamængde af samme størrelse som filstørrelsen, lukkes klientprogrammet og filen ligger nu i mappen: "/root/ExFiles/SerialTransmission/".

# Resultater

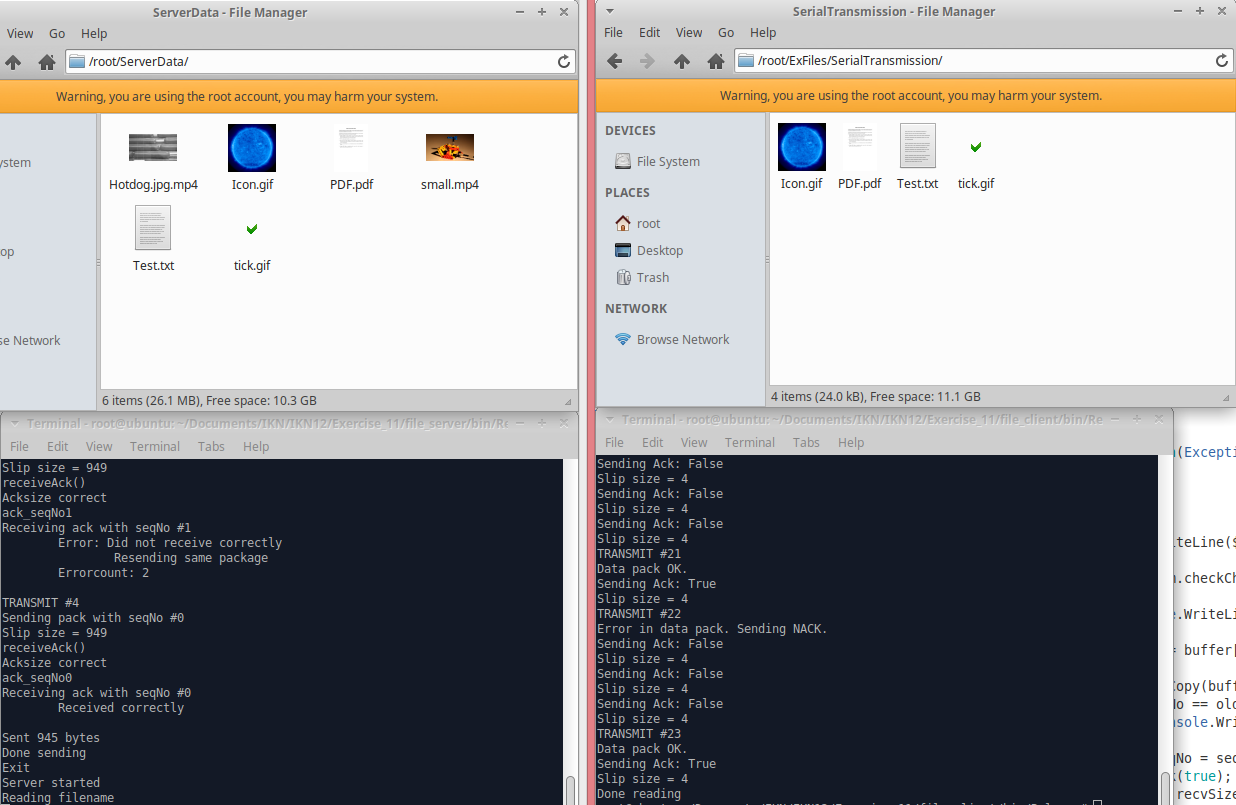
Filoverførsel gennem loopback portene virker rigtig godt. Her kan alle filer sendes hurtigt og stabilt. En 25mb video fil kan sendes i løbet af få sekunder, selv med simulerede bitfejl.



Sendes et filnavn der ikke findes på serveren, så gør serveren igen klar til at modtage filnavn og klient gør klar til at indlæse nyt filnavn fra brugeren.



På det simulerede null modem mellem to virtual machines kan filer også sendes og modtages. Her er fire filer sendt mellem server og klient, hvor server har kørt iterativt.



Det simulerede null modem mellem to virtual machines giver problemer på store filer. Dette skyldes at der kommer mange fejl på disse forbindelser, der ødelægger framingen. Ved hjælp af lidt fejlhåndtering på linklaget er dette blevet fikset lidt, men sker stadig på filer over nogle 100 kilobytes.

# Konklusion

Gruppen kan konkludere at linklaget, transportlaget og applikationslaget kan snakke sammen, og sikre at selv store filer kan overføres korrekt med bitfejl, når kommunikationen sker over loopback porte.

Mellem virtual machines virker overførslen mindre stabil og kan kun håndtere mindre filer, grundet ødelagte frames. Det er ikke lykkedes gruppen at løse problemet helt, selv efter lidt hjælp fra vejleder.

1. https://sourceforge.net/projects/tty0tty/files/ [↑](#footnote-ref-1)