

I4IKN

Efterår 2015

Øvelse 8 & 9

Socket programmering

Journal

Udført af:

#1	Stud.nr.: 20071526	Navn: Kasper Behrendt
#2	Stud.nr.: 201370045	Navn: Karsten Schou Nielsen
#3	Stud.nr.: 201370904	Navn: Kenn H Eskildsen

9. november 2015

Indhold

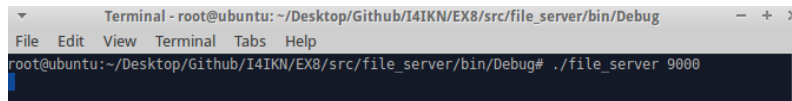
Øvelse 8	3
Fungerende eksempel	3
File_server	5
File_client	8
Øvelse 9	10
Fungerende eksempel	10
File_udp_server	11
File_udp_client	12
Konklusion	13

Øvelse 8

Fungerende eksempel

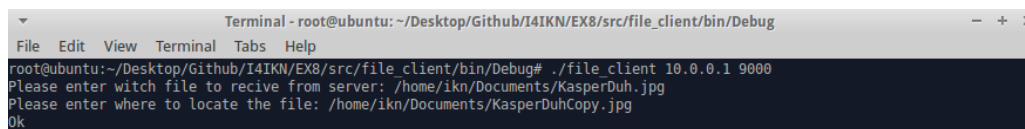
For at gennemgå koden til file_client og file_server vil der først blive gennemgået et fungerende eksempel hvor en fil fra computeren IKN1 overføres til IKN2. Dette gøres ved at starte file_server med port 9000 på IKN1 og file_server på IKN2. Her sendes IP-adressen med på IKN1 samt port 9000.

På [figur 1](#) ses at file_server er startet på IKN1 på port 9000



Figur 1: Screenshot af opstart

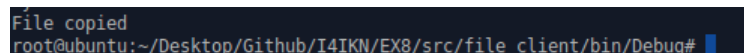
På [figur 2](#) ses at klienten startes med IP-adressen på IKN1 samt port 9000. Vi ønsker at modtage filen /home/ikn/Documents/KasperDuh.jpg fra IKN1 til /home/ikn/Documents/KasperDuhCopy.jpg på IKN2



Figur 2: Screenshot med sti-angivelse på fil der ønskes hentet.

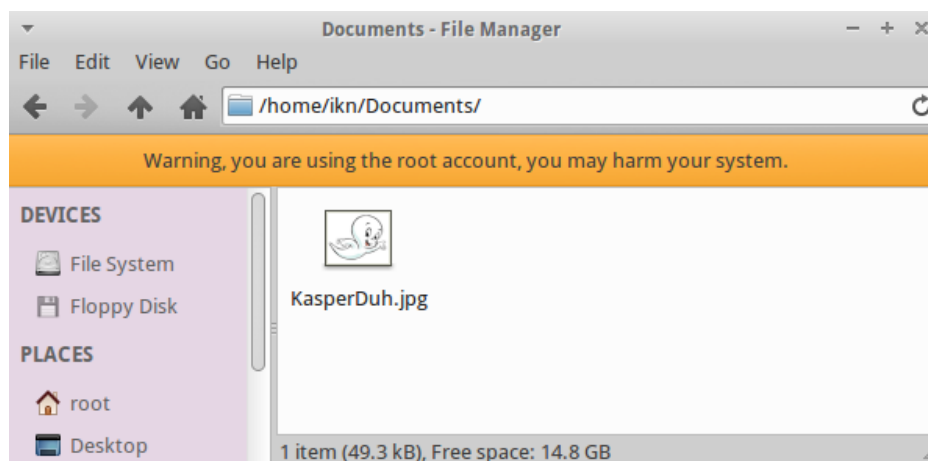
Det ses endvidere på [figur 2](#) at vi får svaret ok fra serveren. Det vil sige at den her siger ok til at filen eksisterer.

På [figur 3](#) ses at vi nu har modtaget filen.



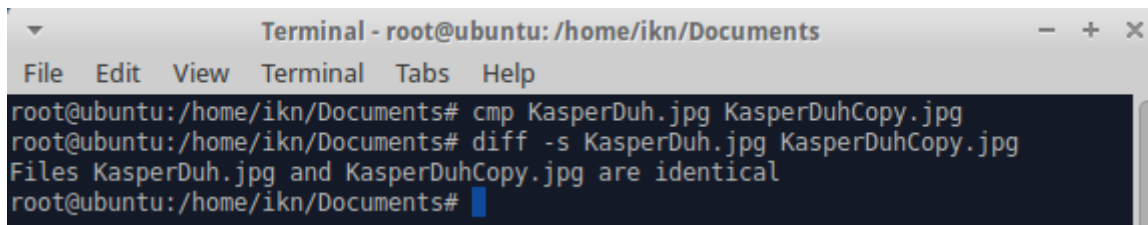
Figur 3: Bekræftelse på kopiering-fuldført

Det ses på [figur 4](#) at filen ligger nu i den ønskede mappe.



Figur 4: Verificering af placering af kopieret fil

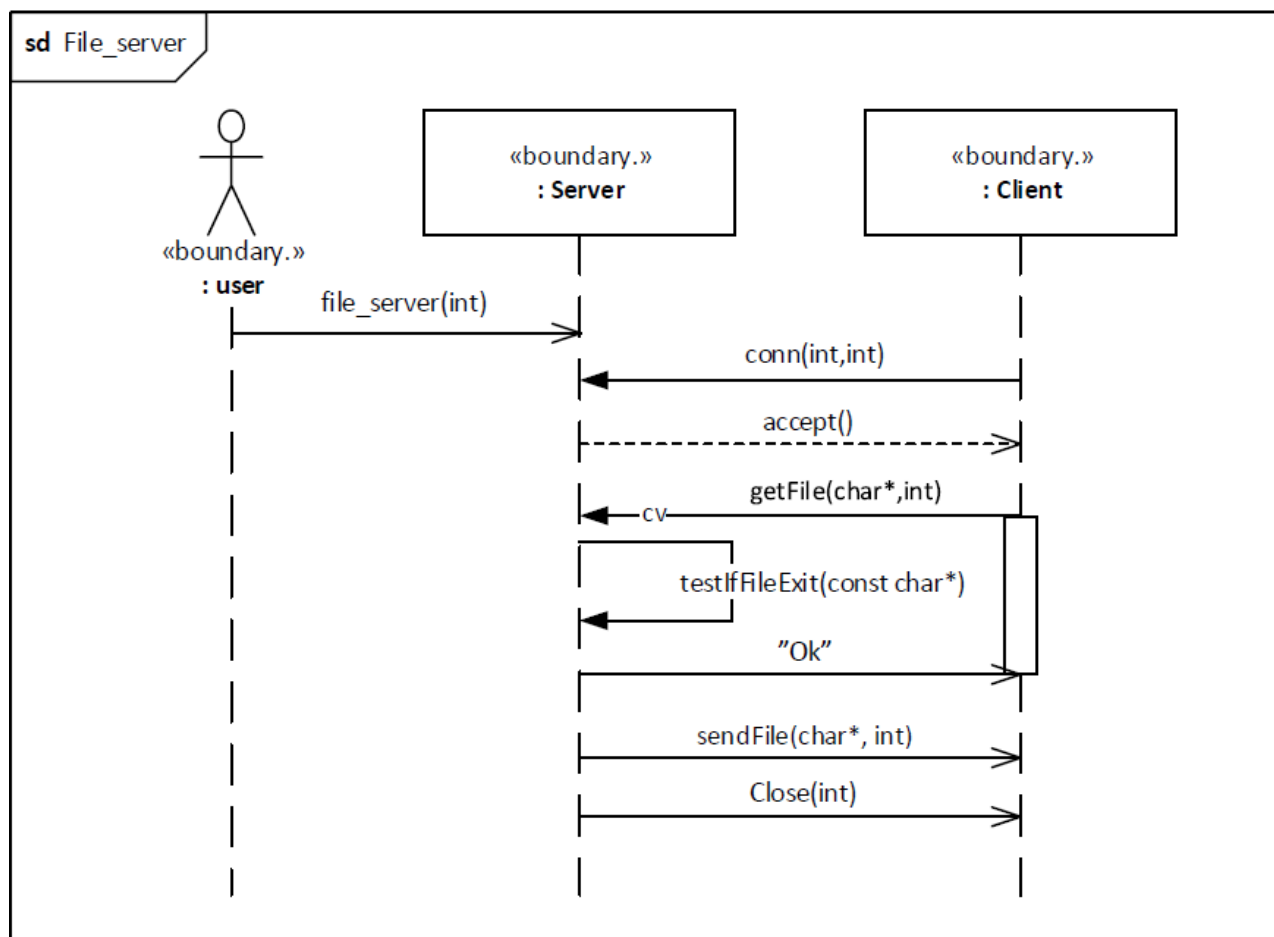
For at eftervise at vores overførsel sker uden fejl, kørte vi både serveren og klienten på IKN1. Herefter brugte vi kommandoerne "cmp" og "diff -s" til at sammenligne filerne. Det ses på [figur 5](#) at filerne er ens.



```
Terminal - root@ubuntu: /home/ikn/Documents
File Edit View Terminal Tabs Help
root@ubuntu:/home/ikn/Documents# cmp KasperDuh.jpg KasperDuhCopy.jpg
root@ubuntu:/home/ikn/Documents# diff -s KasperDuh.jpg KasperDuhCopy.jpg
Files KasperDuh.jpg and KasperDuhCopy.jpg are identical
root@ubuntu:/home/ikn/Documents#
```

Figur 5: Sammenligning af filerne

På [figur 6](#) ses sekvensdiagrammet over filoverførslen.



Figur 6: Sekvensdiagram over transaktion

File_server

I dette afsnit gennemgås koden for file_server for at give et overblik over hvordan dens funktionalitet.

main-funktionen tager argumentet ind for selve porten

```
int main(int argc, char *argv[])
```

argc angiver hvor mange argumenter der er givet ved. Der testes om dette er under 2, for så er der ikke givet et argument med.

```
if (argc < 2) {  
    fprintf(stderr, "ERROR, no port provided\n");  
    exit(1);  
}
```

Efterfølgende oprettes selve socket forbindelsen. Her angiver AF_inet at det er IPV4 protokollen der benyttes og SOCK_STREAM angiver at det er en TCP connection vi vil benytte.

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);  
if (sockfd < 0)  
    error("ERROR opening socket");
```

Portnummer sættes til det angivne portnummer. Atoi konverterer det fra ascii til integer.

```
portno = atoi(argv[1]);
```

Efterfølgende udfyldes serv_addr structen. AF_inet angiver igen det er IPV4 protokollen. INADDR_ANY er ipadressen på computeren serveren kører på og htons konverterer portnummeret til network byte order

```
serv_addr.sin_family = AF_INET;  
serv_addr.sin_addr.s_addr = INADDR_ANY;  
serv_addr.sin_port = htons(portno);
```

Nu kan socket adressen bindes. Giver denne funktion fejl, er det sikkert fordi adressen allerede er brugt

```
if (bind(sockfd, (struct sockaddr *) &serv_addr,  
    sizeof(serv_addr)) < 0){  
    close(sockfd);  
    error("ERROR on binding");  
}
```

Nu venter vi på der kommer en forbindelse fra klienten. 5 angiver det antal forbindelser vi tillader

```
listen(sockfd, 5);
```

Når der kommet et kald skal vi efterfølgende acceptere det hvis det er muligt

```
connfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);  
if (connfd < 0){  
    close(connfd);  
    close(sockfd);  
    error("ERROR on accept");  
}
```

Når forbindelsen er accepteret er forbindelsen oprettet og vi er klar til at modtage beskeder. Read venter indtil at der kommer en besked. Vi udprinter hvilken fil klienten prøver at hente

```

if (read(connfd,buffer,255) < 0)
{
    close(connfd);
    close(sockfd);
    error("ERROR reading from socket");
}
printf("File to send: %s\n",buffer);

```

Der testes efterfølgende om filen findes med funktionen `testIfFileExist()`. Denne funktion vil blive gennemgået senere. Koden siger sig selv.

```

buffer[strlen(buffer)-1]=0;
if (testIfFileExist(buffer))
{
    write(connfd,"\nERROR file does not exist",26);
    close(connfd);
    close(sockfd);
    error("ERROR file does not exist");
}
else
{
    if (write(connfd,"Ok",2) < 0)
    {
        close(connfd);
        close(sockfd);
        error("ERROR writing to socket");
    }
}

```

Når vi har testet at filen eksisterer er vi klar til at sende den. Her bruges funktionen `sendFile()`

```

sendFile(buffer,connfd);

```

`testIfFileExist` er en ret simpel funktion. Det eneste den gør er at teste om den kan åbne filen. Kan den ikke returnerer den 1, så returneres den 0.

```

int testIfFileExist(const char* fileName)
{
    FILE *fp = fopen(fileName,"r");

    if(fp==NULL){
        return 1;
    }

    fclose(fp);
    return 0;
}

```

Funktionen `sendFile` er lidt mere avanceret. Her åbnes filen først og kopieres over i en buffer af størrelsen `BUF_SIZE`. Efterfølgende sendes indholdet af bufferen ud til klienten. Når filen er færdig med at blive sendt tester vi hvor `file-pointeren` står for at sikre der ikke er sket en fejl.

```

void sendFile(const char* fileName, int connfd)
{
    // Fil der ønskes afsendt
    FILE *fp = fopen(fileName, "r");
    if (fp == NULL)
    {
        printf("\nError opening file\n");
        return;
    }

    // data læses fra filen og afsendes
    while(1)
    {
        // Data brydes op i BUF_SIZE stykker
        unsigned char buff[BUF_SIZE] = {0};
        int nread = fread(buff, 1, BUF_SIZE, fp);
        printf("Bytes read %d \n", nread);

        // Hvis læsning lykkes afsendes filen
        if (nread > 0)
        {
            printf("Sending \n");
            write(connfd, buff, nread);
        }

        // Her tjekkes på placering af fp.
        if (nread < BUF_SIZE)
        {
            if (feof(fp))
                printf("End of file\n");
            if (ferror(fp))
                printf("Error reading\n");
            break;
        }
    }

    fclose(fp);
}

```

File_client

Selve forbindelsen til klienten er næsten det samme som i serveren. Den vil derfor ikke blive gennemgået, men i stedet henvises til bilagende.

Når klienten har oprettet forbindelse til serveren skal der først angives hvilken fil der ønskes at blive modtaget. Bzero sætter alle pladser i arrayet file_rc til 0. fgets henter inputtet fra brugeren.

```
printf("Please enter witch file to recive from server: ");
bzero(file_rc,256);
fgets(file_rc,255,stdin);
```

Samme gøres for stien hvor filen ønskes modtaget til. Her bruges scanf i stedet for fgets

```
printf("Please enter where to locate the file: ");
bzero(file_lc,256);
scanf("%s",file_lc);
```

Stinavnet skrives nu til serveren

```
n = write(sockfd,file_rc,strlen(file_rc));
if (n < 0)
{
    close(sockfd);
    printf("ERROR writing to socket");
    exit(1);
}
```

Der læses et svar fra serveren

```
bzero(buffer,256);
n = read(sockfd,buffer,255);
if (n < 0)
{
    close(sockfd);
    printf("ERROR reading from socket");
    exit(1);
}
printf("%s\n",buffer);
```

Der testes om svaret er "ok", hvis ikke så eksisterer filen ikke

```
if (!(buffer[0] == 'O') && (buffer[1]!='k'))
{
    close(sockfd);
    printf("\nERROR file does not exist on server\n");
    exit(1);
}
```

Er svaret "ok" henter vi filen

```
receiveFile(file_lc,sockfd);
```

Funktionen receiveFile() minder meget om sendFile() her modtager vi blot de stykker som blev sendt afsted i størrelsen BUF_SIZE. Både i serveren og klienten er den defineret til 1000 bytes.


```

void receiveFile(const char fileName[], int sockfd)
{
    // Opret en fil hvor dataen vil blive modtaget
    FILE *fp = fopen(fileName, "w");
    if(NULL == fp)
    {
        printf("Error opening file");
        return;
    }

    // Modtag data i stykker af BUF_SIZE i bytes
    int bytesReceived = 0;
    char buff[BUF_SIZE];
    memset(buff, '0', sizeof(buff));
    while((bytesReceived = read(sockfd, buff, BUF_SIZE)) > 0)
    {
        printf("Bytes received %d\n",bytesReceived);
        fwrite(buff, 1,bytesReceived,fp);
    }

    if(bytesReceived < 0)
    {
        printf("\n Read Error \n");
    }

    printf("File copied\n");
}

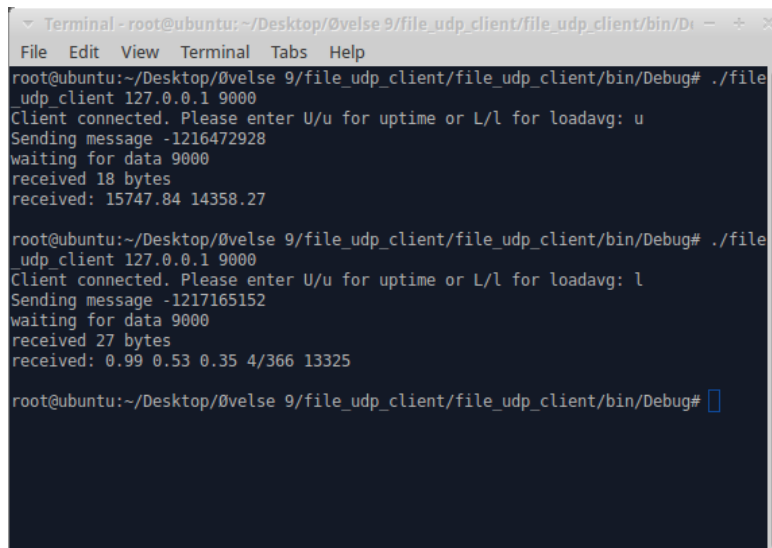
```

Øvelse 9

Fungerende eksempel

Vi vil nu teste en socket forbindelse over UDP, hvor vi modtager indholdet af henholdsvis serverens uptime eller loadavg. Ved start af programmet på klient-siden gives IP-adressen med og portnummeret. Efterfølgende bestemmes om det er uptime eller loadavg der ønskes modtaget, ved brug af parameterne U eller L.

På [figur 7](#) ses at vi starter udp klienten og at vi både kan modtage uptime og loadavg.




```
Terminal - root@ubuntu: ~/Desktop/Øvelse 9/file_udp_client/file_udp_client/bin/Debug - + x
File Edit View Terminal Tabs Help
root@ubuntu:~/Desktop/Øvelse 9/file_udp_client/file_udp_client/bin/Debug# ./file_udp_client 127.0.0.1 9000
Client connected. Please enter U/u for uptime or L/l for loadavg: u
Sending message -1216472928
waiting for data 9000
received 18 bytes
received: 15747.84 14358.27

root@ubuntu:~/Desktop/Øvelse 9/file_udp_client/file_udp_client/bin/Debug# ./file_udp_client 127.0.0.1 9000
Client connected. Please enter U/u for uptime or L/l for loadavg: l
Sending message -1217165152
waiting for data 9000
received 27 bytes
received: 0.99 0.53 0.35 4/366 13325

root@ubuntu:~/Desktop/Øvelse 9/file_udp_client/file_udp_client/bin/Debug#
```

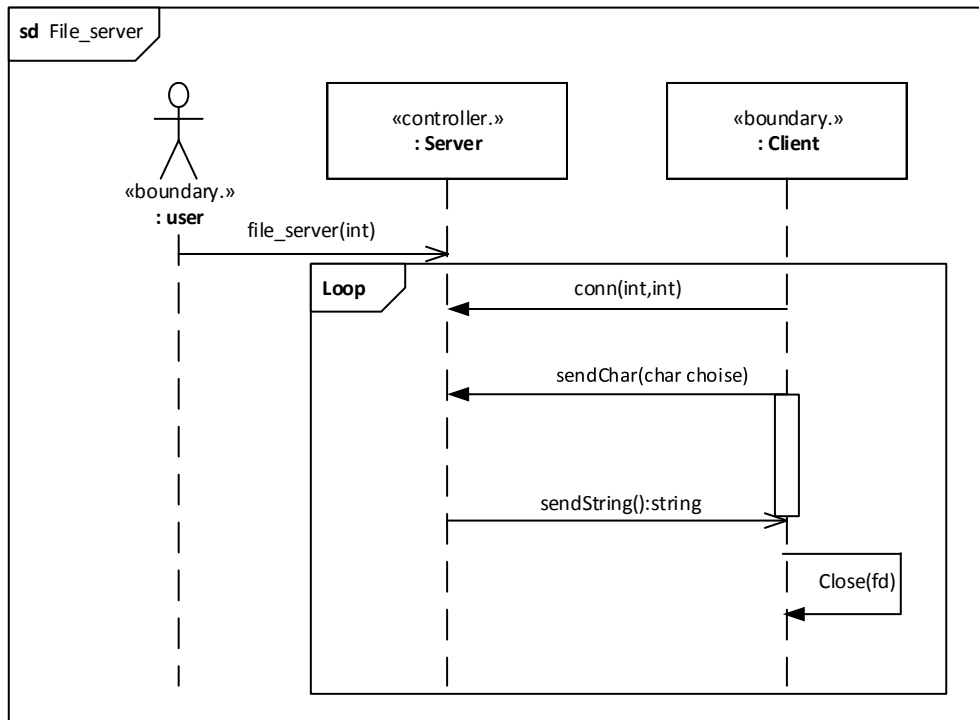
Figur 7: Start af programmet file_udp_client

På [figur 8](#) ses at serveren har skrevet og modtaget 2 gange. Dette er henholdsvis uptime og loadavg der sendes til klienten.



```
Terminal - root@ubuntu: ~/Desktop/Øvelse 9/file_udp_server/file_udp_server/bin/Debug - + x
File Edit View Terminal Tabs Help
root@ubuntu:~/Desktop/Øvelse 9/file_udp_server/file_udp_server/bin/Debug# ./file_udp_server
ERROR, no port provided
root@ubuntu:~/Desktop/Øvelse 9/file_udp_server/file_udp_server/bin/Debug# ./file_udp_server 9000
waiting on port 9000
received 2 bytes
received message: "u"
Bytes read from file 18
waiting on port 9000
received 2 bytes
received message: "l"
Bytes read from file 27
waiting on port 9000
```

Figur 8: Start af programmet file_udp_server



Figur 9: Sekvensdiagram set fra server siden

File_udp_server

Selve serveren minder meget om TCP serveren, dog er den store forskel her at det er UDP protokollen der ønskes at gøre brug af. Derfor ændres funktionen socket til at bruge variablen SOCK_DGRAM.

```
//Her oprettes vores socket. SOCK_DGRAM er UDP connection. AF_INET betyder det er IPV4
protokollen vi benytter
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");
```

Derefter udfyldes structen som i TCP serveren

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
```

Efterfølgende bindes adressen til serveren og vi er nu klar til at oprette forbindelse til klienten. Det ses her at vi springer et led over i forhold til TCP idet der ikke afsendes nogen accept.

```
//bind binder adressen til serveren. Ved fejl er adressen sikkert allerede brugt
if (bind(sockfd, (struct sockaddr *) &serv_addr,
    sizeof(serv_addr)) < 0){
    close(sockfd);
    error("ERROR on binding");
}
```

Den næste del er at vente på forbindelsen og derefter sende det ønskede til klienten. Her oprettes en while løkke som venter på en forespørgsel.

```
while(1 )
{
    printf("waiting on port %d\n", PORT);
    recvlen = recvfrom(sockfd, buffer, BUFSIZE, 0, (struct sockaddr *)&cli_addr,
    &addrlen);
    printf("received %d bytes\n", recvlen);
    if (recvlen > 0)
    {
        buffer[recvlen] = 0;
        printf("received message: \"%s\"\n", buffer);
    }

    if(buffer[0]=='u')
    {
        /* Open the file that we wish to transfer */
        FILE *fp = fopen("/proc/uptime","r");
        if(fp==NULL)
        {
            printf("\nError opening file\n");
            exit(1);
        }
        unsigned char buff[BUF_SIZE]={0};
        int nread = fread(buff,1,BUF_SIZE,fp);
        printf("Bytes read from file %d \n", nread);

        if (sendto(sockfd, buff, nread, 0, (struct sockaddr *)&cli_addr, addrlen)==-1)
        {
            fprintf(stderr, "ERROR in sendto\n");
            exit(1);
        }
    }
}
```

Samme fremgangsmåde ville der ske hvis det var var "l" der blev modtaget, dog ville vi sende filen /proc/loadavg.

File_udp_client

Igen er forskellen ikke det store fra TCP klienten. Det største forskel er at der ikke ventes på nogen accept efter at vi har oprettet forbindelse til serveren som gøres på følgende måde.

```
if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))== -1)
    fprintf(stderr, "ERROR in socket\n");
```

Det ses at der kun er ændret nogle parameter i socket (SOCK_DGRAM og IPPROTO_UDP).

Structen udfyldes som før.

```
memset((char *) &si_other, 0, sizeof(si_other));
si_other.sin_family = AF_INET;
si_other.sin_port = htons(PORT);
if (inet_aton(argv[1], &si_other.sin_addr)==0) {
    fprintf(stderr, "inet_aton() failed\n");
    exit(1);
}
```

Nu er vi klar til at sende og modtage data. Her spørges der efter et input og efterfølgende sendes det ud til serveren.

```
printf("Client connected. Please enter U/u for uptime or L/l for loadavg: ");
bzero(my_message,256);
scanf("%s",my_message);

if(my_message[0] == 'U' || my_message[0] == 'u' )
    my_message[0] = 'u';
else if(my_message[0] == 'L' || my_message[0] == 'l')
    my_message[0] = 'l';
else
{
    fprintf(stderr,"Error in input\n");
    exit(1);
}
printf("Sending message %d\n", i);
if (sendto(s, my_message, strlen(my_message)+1, 0,
    (struct sockaddr *)&si_other, slen)==-1)
{
    fprintf(stderr, "ERROR in sendto\n");
    exit(1);
}

printf("waiting for data %d\n", PORT);
recvlen = recvfrom(s, buf, BUFSIZE, 0, (struct sockaddr *)&si_other,&addrlen );
printf("received %d bytes\n", recvlen);
printf("received: %s\n", buf);
```

Konklusion

Vi har hermed arbejdet med en TCP socket forbindelse og en UDP socket forbindelse. Vi har set at den store forskel er at TCP er connection orienteret hvorimod UDP er connection-less. TCP er derfor mere sikker at bruge til at overføre filer end UDP, da der i UDP kan opstå datatab eller ombytning af pakker.