

I4IKN- Øvelse 7

Application Layer, Transport Layer, TCP/IP Socket Programming

Jonas Lind	201507296
Tais Hjortshøj	201509128

Problemformulering

I denne øvelse skal der udvikles en iterativ server og en client som skal køre i hver deres virtuelle Linux-maskine. Serveren skal kunne modtage en tekststreng fra én client ad gangen. Tekststrengen som clienten sender skal indeholde filnavn og eventuelt sti angivelse og herved udpege en fil af en vilkårlig type/størrelse i serveren, som den tilsluttede client ønsker at hente fra serveren. Filen skal overføres fra server til client i segmenter på 1000 bytes ad gangen indtil filen er overført fuldstændigt. Der returneres en fejlmeddelelse fra serveren til clienten hvis den ønskede fil ikke findes i serveren. Efter filoverførslen er færdig skal serveren kunne håndtere en ny forespørgsel fra en client.

Som kvalitetskontrol for client/server systemet skal den overførte fil kunne sammenlignes med den oprindelige fil. Der må ikke være nogle forskelle i filerne mht. til størrelse eller mht. indhold.

Server

Snippet 1 viser koden for initiering af vores socket på serveren. Vi udskriver til terminalen hvilken adresse serveren er forbundet til. SOCK_STREAM gør at det er en TCP forbindelse. Vi håndterer fejl ved at udskrive en fejlmeddelelse, lukke socketten og forlade programmet.

```
PORT = 9000
HOST = ''
BUFSIZE = 1000

def main(argv):
    try:
        #Initiate server
        serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        #Bind socket to PORT and localhost
        serversocket.bind((HOST, PORT))
        serversocket.listen(1) #We want to queue up to just one client
        print 'Socket connection on: ', serversocket.getsockname()

    except socket.error as msg:
        print 'Error connecting with serversocket: %s\n Terminating program.'
            %msg
        serversocket.close()
        sys.exit()
```

Snippet 1 – Oprettelse af serversocket.

Snippet 2 viser vores while-løkke som håndtere én connection fra en client for derefter at starte forfra. Dette gør vores server iterativ. Vi bruger funktioner fra det udleverede Lib, som er blevet lavet som et object bibliotek og derfor skal vi kalde objektet Lib.funktion for at gøre brug af disse.

```
while(1):
    #Wait for connections
    (clientsocket, address) = serversocket.accept()
    print 'Incomming connection from', address

    filename = Lib.readTextTCP(clientsocket) #Connect with client
    print '1 ', filename

    filesize = Lib.check_File_Exists(filename) #Returns size
    print '2 filesize ', filesize

    if filesize != 0:
        sendFile(filename, filesize, clientsocket)
    else:
        Lib.writeTextTCP("0", clientsocket)
        Lib.writeTextTCP("File " + filename + "doesnt exist.", clientsocket)

    print '3 Closing connection ', address
    clientsocket.close()
```

Snippet 2 – Iterativ server der ikke lukker ned efter end filoverførsel til client.

Snippet 3 viser vores sendFile funktion som vi bruger til at sende den fil der er blevet anmodet om. Den skal bruge filnavn, filstørrelse og en socket at sende på. Vi starter med at sende filstørrelsen som tekststreng og derefter sender vi den anmodede fil i bider på 1000 bytes (BUFSIZE). Den udskriver i terminalen hvor mange packets der er sendt og at filen er sendt til sidst.

```
def sendFile(fileName, fileSize, conn):
    i = 0
    text = "."

    #Sending size of requested file
    Lib.writeTextTCP(str(fileSize), conn)

    with open(fileName, "rb") as file_obj:
        while 1: #text not == "":
            text = file_obj.read(BUFSIZE)
            i = i + 1
            if text == "":
                conn.send(text)
                break
            conn.send(text)
            print 'Packets sent: ', i

    print 'File sent: ', fileName
```

Snippet 3 – Funktion til at sende fil.

Client

Vi har udviklet en client som opfylder at den skal have to argumenter når den kaldes – en filsti og hvor denne skal hentes.

```
PORT = 9000
HOST = ''
BUFSIZE = 1000

def main(argv):
    #File to be downloaded
    fileName = argv[0]    #Filepath as argument
    print '1 Filename: ', fileName

    if argv[1]:
        HOST = argv[1]
        print HOST

    #Creating socket
    clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientsocket.connect((HOST, PORT))
    print '2 '
```

Snippet 4 – Oprettelse af clientsocket.

Snippet 4 viser initiering af client socket. SOCK_STREAM gør at det er TCP.

```
#Request server for file through the socket
Lib.writeTextTCP(fileName, clientsocket)
print '3 '

#Wait to receive file
receiveFile(fileName, clientsocket)
print '5 File received. Closing connection.\n '

clientsocket.close()
```

Snippet 5 – Client anmoder server om fil gennem socket.

Snippet 5 viser at vi initiere TCP forbindelsen ved at anmode om en fil. Derefter kalder vi funktionen receiveFile(). Når vi har modtaget filen lukkes socketten og programmet lukkes.

```
def receiveFile(fileName, conn):
    size = Lib.readTextTCP(conn) #Save recieved message
    fileName = Lib.extractFilename(fileName) #Remove path
    print '4 Data size: ', long(size)

    if long(size) == 0:
        print 'File', fileName, 'doesn\'t exist. Closing connection.'
        sys.exit()

    text_obj = open(fileName, "w") #Make new file

    i = 0

    while i < long(size):
        text = conn.recv(BUFSIZE)
        text_obj.write(text) #Write message to file
        i = i + len(text)
        print 'Text received: ', i, long(size)
        if i >= size:
            break

    print 'Text obj: ', text_obj,
    text_obj.close() #Save file
    filesize = Lib.check_File_Exists(fileName)
    print '\nSize of file: ', filesize
```

Snippet 6 – Modtage fil fra server

Snippet 6 viser vores funktion til at modtage filen. Vi starter med at modtage størrelsen på filen som en tekststreng. Der tages hensyn til om filen ikke eksisterer da filstørrelsen så vil være 0. Derefter opretter vi en fil og skriver alle tegn vi modtager ind i denne indtil den korrekte filstørrelse er opnået. Derefter gemmes filen ved at lukke denne og vi tjekker størrelsen på den nye fil.

Test

```
root@ubuntu:~/Desktop/IKN# python file_client_2.py /root/Downloads/images.jpeg 1
0.0.0.2
1 Filename: /root/Downloads/images.jpeg
10.0.0.2
2
3
4 Data size: 2647
Text received: 1000 2647
Text received: 2000 2647
Text received: 2647 2647
Text obj: <open file 'images.jpeg', mode 'w' at 0xb7238390>
Size of file: 2647
5 File received. Closing connection.

root@ubuntu:~/Desktop/IKN# python file_client_2.py /root/Downloads/images.jpg 10
0.0.0.2
1 Filename: /root/Downloads/images.jpg
10.0.0.2
2
3
4 Data size: 0
File images.jpg doesn't exist. Closing connection.
root@ubuntu:~/Desktop/IKN#
```

Figur 1 – Succesfuld anmodning og modtagelse af fil i klientens terminal. Der ses at vi kan håndtere hvis filen ikke eksisterer og der vil ikke blive oprettet en tom fil.

```
root@ubuntu:~/Desktop/IKN# python file_server.py
Socket connection on: ('0.0.0.0', 9000)
Incoming connection from ('10.0.0.1', 57576)
1 /root/Downloads/images.jpeg
2 filesize 2647
Packets sent: 1
Packets sent: 2
Packets sent: 3
File sent: /root/Downloads/images.jpeg
3 Closing connection ('10.0.0.1', 57576)
Incoming connection from ('10.0.0.1', 58070)
1 /root/Downloads/images.jpg
2 filesize 0
3 Closing connection ('10.0.0.1', 58070)
```

Figur 2 - Succesfuld modtagelse af anmodning og afsendelse af fil. Serveren fungerer iterativt og kan håndtere at filen ikke eksisterer. Vi kan også se at vores TCP forbindelse ligger på port 9000, og at der oprettes nye porte til at sende på. (57576 og 58070).

```
root@ubuntu:~/Desktop/IKN# diff images.jpeg /root/Downloads/images.jpeg
root@ubuntu:~/Desktop/IKN#
```

Figur 3 - Vi downloader den originale fil fra nettet og sammenligner den med den vi har hentet gennem vores TCP. Der er ingen forskel.

Konklusion

Vores TCP server virker fint. Den kan håndtere mange typer fejl, deriblandt at filen ikke eksisterer. De filer vi modtager er identiske til de oprindelige filer vi sender hvilket betyder at det virker som det skal. Vi kan sende alle slags filer da vi splitter filen op inden den sendes. Vi har her valgt at inkludere testen med at sende et billede samt fejl i stien som derved henviser til en ikke-eksisterende fil.

Det virker mellem to forskellige virtuelle maskiner på samme PC.

Den fulde kode er vedlagt i bilag.

I4IKN- Øvelse 8

UDP/IP Socket Programming

UDP server

Vi vil udvikle en iterativ server som kan modtage en kommando fra en client. Kommandoerne som genkendes er "U", "u", "L" og "l". Vi håndterer at der modtages andre kommandoer.

Snippet 7 viser initiering af vores socket. SOCK_DGRAM gør den til en UDP forbindelse. Vi sætter socket options til at være genbrugelig for at den ikke er lukket i en periode efter vores program crasher. Dette er primært brugbart under tests med kodeændring. Vi fejlhåndterer og udskriver fejlmeddelelse hvis dette sker.

```
def main(argv):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        s.bind((HOST, PORT))

    except socket.error as msg:
        print 'Error 1 encountered: ', msg
        s.close()
        sys.exit()
```

Snippet 7 – Initiering af socket.

Snippet 8 viser vores while-loop som gør vores server iterativ. Der bliver også her fejlhåndteret. Den kan primært 3 ting. Enten anmodes om en af 2 filer eller også kan den svare med ukendt anmodning.

```
try:
    while(1):
        print '\nServer ready to receive data.'
        data, address = s.recvfrom(BUFSIZE)
        print 'Message from: ', address
        print 'Message received: ', data

        if data == "L" or data == "l":
            sendFile("/proc/loadavg", address, s)
        elif data == "U" or data == "u":
            sendFile("/proc/uptime", address, s)
        else: print 'Command not recognized: ', data
            socket.sendto("Command not recognized", address)

    except socket.error as msg:
        print 'Error 2 encountered: ', msg
        s.close()
        sys.exit()
```

Snippet 8 – Iterativ server der ikke lukker ned efter end filoverførsel til client.

```
def sendFile(file_, address, socket):  
    text = openFile(file_)  
    socket.sendto(text, address)  
    print 'Returning ', file_  
  
def openFile(file_) :  
    with open(file_, "rb") as file_obj:  
        text = file_obj.read()  
    return text
```

Snippet 9 – Vi ser her vores to funktioner til at åbne en fil og sende en fil.

Test

```
root@ubuntu:~/Desktop/IKN/Ovelse_8# python server_udp.py  
  
Server ready to receive data.  
Message from: ('10.0.0.2', 9000)  
Message received: l  
Returning /proc/loadavg  
  
Server ready to receive data.  
Message from: ('10.0.0.2', 9000)  
Message received: L  
Returning /proc/loadavg  
  
Server ready to receive data.  
Message from: ('10.0.0.2', 9000)  
Message received: U  
Returning /proc/uptime  
  
Server ready to receive data.  
Message from: ('10.0.0.2', 9000)  
Message received: u  
Returning /proc/uptime  
  
Server ready to receive data.
```

Figur 1 – Vi kan modtage 4 kommandoer og der returneres det korrekte. Serveren er iterativ.

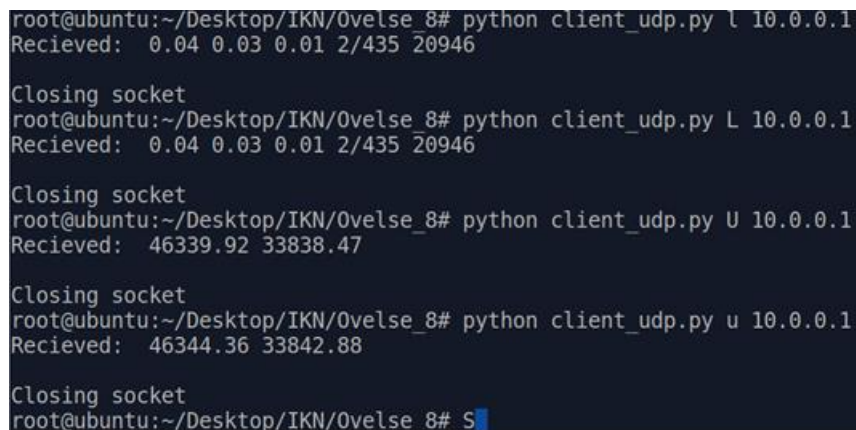
UDP Client

Vi vil udvikle en client som sender ved hjælp af UDP. Den modtager to argumenter – en kommando som skal sendes og hvor denne skal sendes til.

```
def main(argv):  
  
    #Create UDP socket  
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
    sock.bind((UDP_HOST, UDP_PORT))  
  
    cmd = argv[0]  
    addr = argv[1]  
  
    #Send data  
    sock.sendto(cmd, (addr, UDP_PORT))  
  
    #Recieve response  
    data, addr = sock.recvfrom(BUFFER_SIZE)  
    print 'Recieved: ', data  
  
    #Closing Socket  
    print 'Closing socket'  
    sock.close()
```

Snippet 10 – Koden for vores UDP client. Vi initiere en socket. Der modtages to argumenter som behandles. Data modtaget udskrives i terminalen.

Test



```
root@ubuntu:~/Desktop/IKN/Ovelse_8# python client_udp.py l 10.0.0.1  
Recieved: 0.04 0.03 0.01 2/435 20946  
  
Closing socket  
root@ubuntu:~/Desktop/IKN/Ovelse_8# python client_udp.py L 10.0.0.1  
Recieved: 0.04 0.03 0.01 2/435 20946  
  
Closing socket  
root@ubuntu:~/Desktop/IKN/Ovelse_8# python client_udp.py U 10.0.0.1  
Recieved: 46339.92 33838.47  
  
Closing socket  
root@ubuntu:~/Desktop/IKN/Ovelse_8# python client_udp.py u 10.0.0.1  
Recieved: 46344.36 33842.88  
  
Closing socket  
root@ubuntu:~/Desktop/IKN/Ovelse_8# S
```

Figur 2 – Vi ser at alle 4 kommandoer returnerer hvad de skal. Programmet lukker derefter ned og er klar til at køres igen.

Konklusion

Vi har nu lavet både en UDP server og en TCP server som vi har fået til at virke. Vi har prøvet at håndtere fejl der kan opstå.

Fuld kode er vedhæftet i bilag.