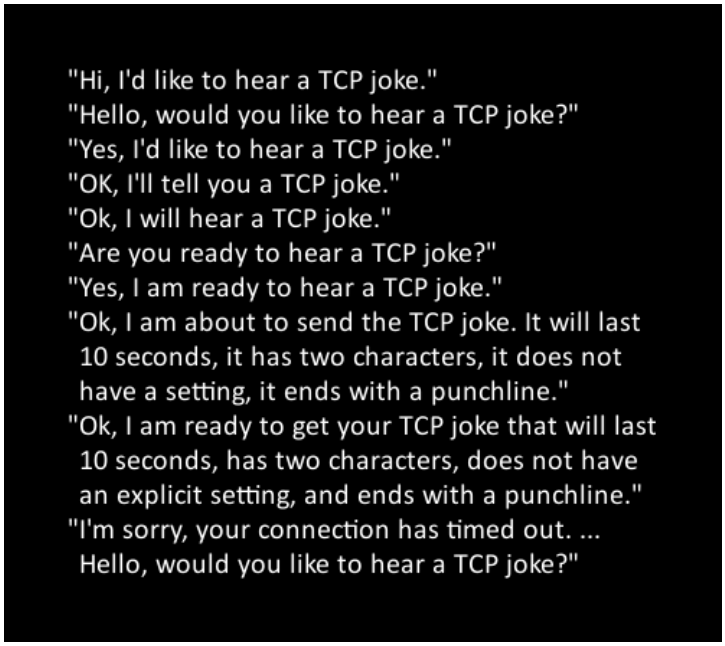
Øvelse 8- TCP



20105969 - Kalle R. Møller

201370050 - Kasper Sejer Kristensen

TCP Socket Programmering

I denne øvelse har vi fået til opgave at programmere et TCP server/client setup der er i stand til at overføre filer af en vilkårlig størrelse. Dertil har vi specificeret en protokol der beskriver de beskeder der kan sendes frem og tilbage.

Protokollen

Vi er har specificeret protokollen således:

Beskedformat:

Klienten sender:

Kommando + filestien + versionen + delimiter

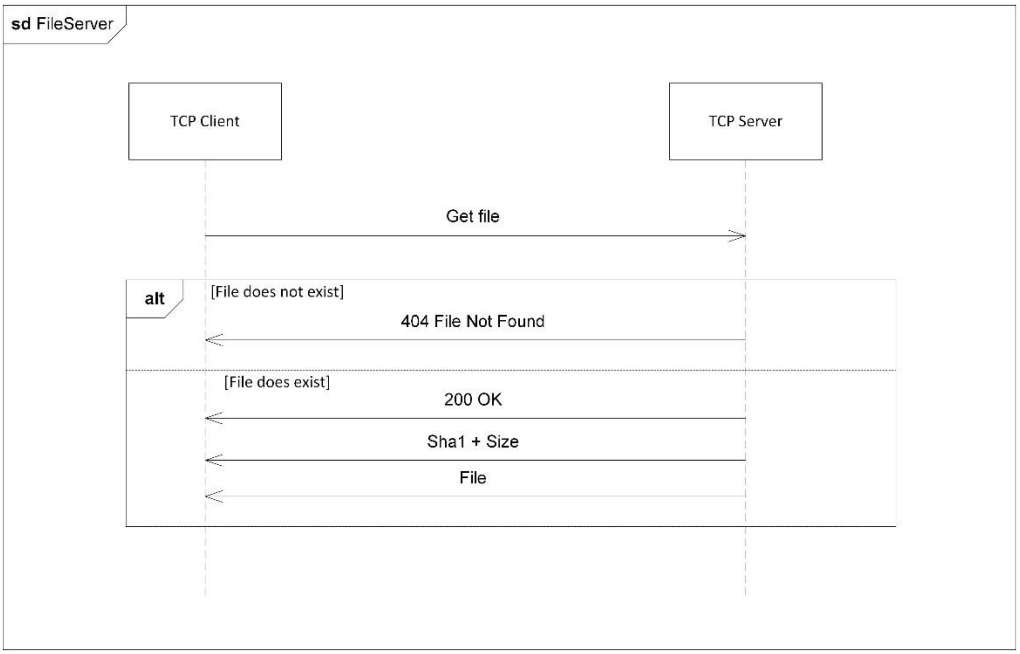
Der er en kommando og det er ”GET”, filstien er stien til den fil der skal overføres fra server til klient, versionen er versionen af protokollen som pt er ”FS/1.0” og delimteren er ”\r\n\r\n”.

Serveren svare så med følgende:

Version + Svar + (Sha1 + Size + File)

Versionen er det samme som før, Svaret kan være enten ”200 OK” eller ”404 File Not Found”, Sha1 er med for at vi kan tjekke at det er den rigtige fil som vi får overført, Size er størrelsen på filen og File er selve filen.

Sekvensen for besked overførsel ses her:



TCP Sockets

Her beskrives kort hvordan vi har programmeret med TCP sockets. Der bliver fremhævet de vigtigste ting i koden der gør at det virker både for server og for klienten.

Server

Serveren er skrevet i python. Socket biblioteket er blevet brugt til at oprette en TCP socket som serveren bruger til at lytte efter klienter med.

```
def __init__(self):
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.socket.bind(('', 9000))
    self.socket.listen(5)
```

I den viste kode kan det ses hvordan der oprettes en socket med parametrene AF_INET og SOCK_STREAM de fortæller at vi ville have en TCP socket der kommunikere over IP. Efter at vi har oprettet en socket binder vi til alle indkommende IP'er på port 9000. Til slut kaldes listen() som gør at vi lytter efter klienter der vil i kontakt med os.

```
def getclient(self):
    (clientsocket, address) = self.socket.accept()
    return clientsocket
```

Efter at socket er oprettet og vi er sat til at lytte sættes serveren til at stå i en while(1) løkke hvor metode i ovenstående kode bliver kaldt den returnere så en ny clientsocket som bruges til resten af kommunikationen med klienten.

Når klienten har sendt sin besked om at få en fil tjekker serveren om beskeden er korrekt og om filen findes hvis filen findes beregnes SHA1 og størrelsen som så sendes til klienten herefter sendes filen som vist i koden herunder.

```
with open(srcfile,'rb') as f:
    buffer = f.read(1000)
    while len(buffer) > 0:
        self.client.send(buffer)
        buffer = f.read(1000)
```

Her læses der 1000 bytes ind ad gangen som så sendes til klienten dette tillader at vi kan sende store filer uden at bruge alt hostens ram.

Klienten

For udfordringens skyld blev klienten skrevet i et andet sprog nemlig c# det er ingen umiddelbar grund til dette og det har ingen betydning for funktionaliteten for fil serveren.

I c# findes der en klasse der hedder TcpClient denne bliver brugt til at forbinde til serveren med.

```
TcpClient _tcpClient = new TcpClient();
private IPAddress _clientAddress;

    _clientAddress = IPAddress.Parse(args[0]);
    _tcpClient.Connect(_clientAddress, Port);
```

Koden herover er udklip men den viser de ting der skal bruges for at lave en forbindelse til serveren.

IPAddress.Parse metoden tager en string som parameter og forsøger at parse en IP adresse, herefter kan Connect metoden kaldes på _tcpClient for at forbinde til serveren.

```
public void SendRequest(string request)
{
    byte[] toServer = ConvertToBytes(request);
    var stream = _tcpClient.GetStream();
    stream.Write(toServer, 0, toServer.Length);
}
```

Metoden i koden herover bruges til at sende GET beskenden til serveren dette gøres ved først at konvertere den streng der indeholder beskeden til et byte array, herefter kaldes GetStream for at få den stream vi kan sende beskeden over til sidst kaldes Write metoden på stream objektet som så sender beskeden til serveren.

Hvis serveren svar at filen findes kaldes følgende metode for at hente filen.

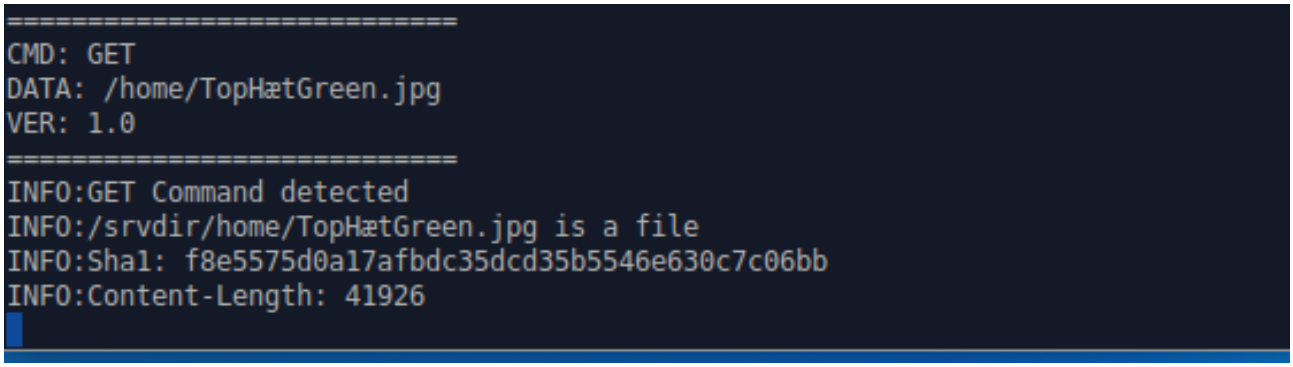
```
public void GetBigFile(int lenght, string path)
{
    var stream = GetStream();
    var fileBytes = new byte[Buffer];
    int bytesToRead = Buffer;
    if(System.IO.File.Exists(path))
        System.IO.File.Delete(path);
    var file = System.IO.File.OpenWrite(path);
    while (lenght > 0)
    {
        var noBytesRead = stream.Read(fileBytes, 0, bytesToRead);
        file.Write(fileBytes, 0, noBytesRead);
        lenght -= noBytesRead;
        noBytesRead = 0;
        if (lenght < Buffer)
        {
            bytesToRead = lenght;
        }
    }
    file.Close();
}
```

Metoden bruge også en stream bare til at læse fra denne gang, den bruger længden på filen til at vide hvor meget den skal læse og der gemmes løbende igen af hensyn til ram.

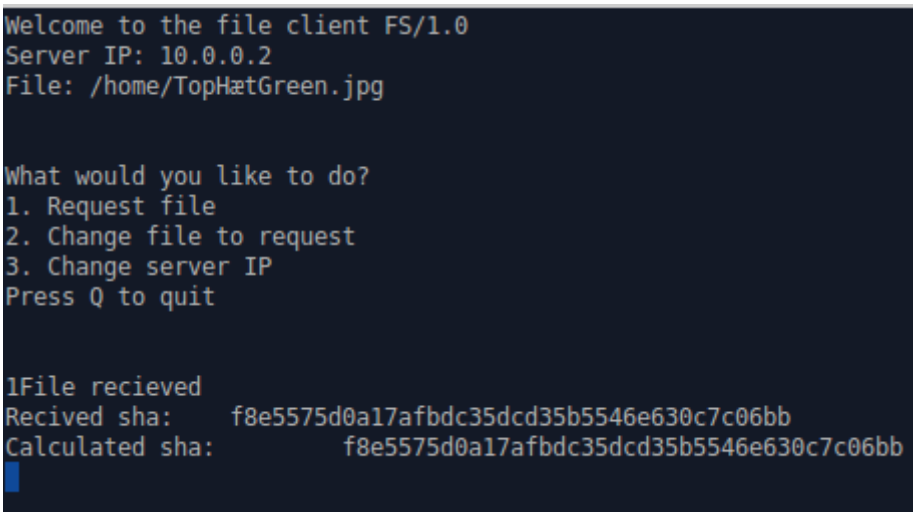
I klienten var det ikke nødvendig at direkte fortælle at vi brugte TCP det blev gjort implicit idet at vi brugte tcpclient klassen. Dog er det mulig at oprette en socket i c# og specificere den selv men tcpclient klassen der til for at gøre livet lettere.

Resultater

Herunder vises et screenshot fra en terminal hvor serveren kører den har netop fået en besked og sender en fil tilbage til klienten:

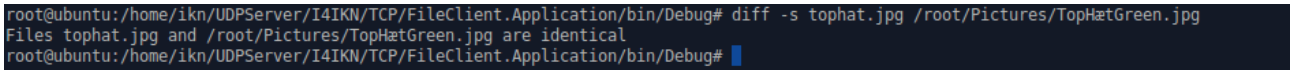


Og her under ses klienten der netop har modtaget filen.



Som det kan ses gik overførslen fint idet at sha er blevet udregnet til det samme på både klient og server.

For god ordensskyld tjekkes det også at filerne er ens:



Som det kan ses her er de to filer ens.