Tentamen
Nätverksprogrammering
Del 2

2007–05–22, 8.00–13.00

Tillåtna hjälpmedel för denna del av tentamen:

- Java snabbreferens
- Kursboken: Java Network Programming av Eliotte Rusty Harold.
- Valfri lärobok i Java
- Utskrift av OH-bilder från föreläsningarna.

Denna tentamen i kursen Nätverksprogrammering består av två delar – en del som innehåller frågor av teoretisk/principiell/utredande karaktär och en del som innehåller praktiska programmeringsuppgifter. Detta är del 2. Den ska du ha erhållit tillsammans med ett färgat tentamensomslag när du lämnade in din lösning på del 1 av tentamen.

För godkänt betyg på tentamen krävs sammanlagt minst 20 poäng på tentamen, varav minst 8 poäng på vardera deltentamen. För högre betyg krävs naturligtvis mer, så gör så många uppgifter du kan.

Denna deltentamen innehåller två uppgifter som vardera kan ge 10 poäng. Uppgifternas ordningsföljd avspeglar inte nödvändigtvis uppgifternas svårighetsgrad.

---

1. Filöverföring med TFTP

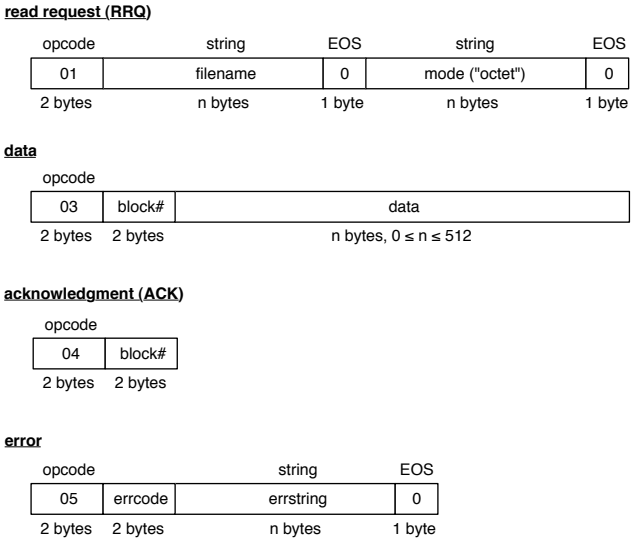
TFTP står för Trivial File Transfer Protocol och är ett enkelt protokoll för filöverföring mellan två datorer (definierat i RFC783). Det är designat för att använda UDP för att skicka över informationen, vilket innebär att protokollet måste hantera omsändning av borttappade paket och liknande. Till skillnad från det mer kända protokollet FTP (som bygger på TCP) innehåller TFTP enbart funktionalitet för att hämta och skicka filer. Protokollet saknar t.ex. funktionalitet för att hantera autentisering av användare och att lista innehållet i kataloger.

Pierre har gett Roger i uppgift att skriva ett klientprogram i Java som kan ladda ner en namngiven fil från en TFTP-server. Programmet ska vara färdigt på fredag, den 25/5, men eftersom Roger förväntar sig vara upptagen med att rätta tentor i Nätverksprogrammering hela veckan har han insett att han inte hinner bli färdig med programmet. Han har dock hunnit skriva en bit av programmet och har haft fräckheten att smyga in uppdraget som en tentamensuppgift i förhoppningen att få hjälp med att skriva färdigt programmet. Det är din uppgift att hjälpa Roger med att komplettera programmet med den saknade delen.

För att kunna lösa uppgiften måste du veta hur TFTP-protokollet fungerar när en fil laddas ner och hur Rogers färdiga kod ser ut. Vi börjar med att gå igenom TFTP-protokollet.

TFTP-protokollet

TFTP använder sig av fem olika typer av paket. Fyra av dem är relevanta för nedladdning av filer från servern[1]. Dessa visas i figuren nedan.



Varje paket inleds med en 16-bitars (2 byte) opkod som talar om vilken typ av paket det är frågan om. Alla 16-bitarsvärden lagras med den mest signifikanta byten först (big-endian). Värdet 256 lagras t.ex. som en byte med värdet 1 följt av en byte med värdet 0.

Ett RRQ-paket (read request) sänds från klienten till servern för att starta en filöverföring. Det specificerar namnet på filen som ska laddas ner och vilken överföringsmod som ska användas. Filnamnet avslutas med en byte med värdet 0 (EOS, End Of String, i figuren). Även överföringsmoden avslutas med en byte med värdet 0. Protokollet definierar två överföringsmoder, *octet* (binärt) och *netascii* (text). Textmoden används för att överföra textfiler och i samband med detta konvertera radslut vid behov. I denna uppgift begränsa vi oss dock till binärmod, i vilken filen överförs exakt som den är.

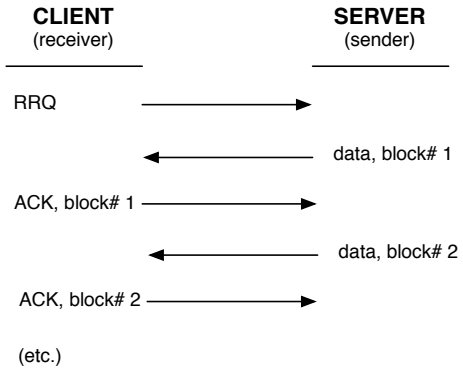
Datapaket innehåller verkliga data tillsammans med ett blocknummer. Längden på datadelen i paketet ligger i intervallet 0 till och med 512 bytes. Om längden på datadelen är mindre än 512 (0..511) bytes anger detta att paketet är det sista datapaketet i filöverföringen. Protokollet skickar alltså alltid fulla 512-byte-paket tills slutet på överföringen nås. Blocknumret används för att upptäcka duplicerade paket och för att kunna bekräfta för sändaren att paketet kommit fram genom att skicka ett ACK-paket. Blocknumren börjar alltid numreras från ett (1) i och med att det första datapaketet i en filöverföring sänds.

Error-paket skickas för att informera klienten om olika felsituationer. Om ett error-paket tas emot avbryts nedladdningen. Man skickar aldrig bekräftelse (ACK) för Error-paket. Möjliga felkoder är:

errcode	Description
0	Not defined, see the *errstring*, if present
1	File not found
2	Access violation
3	Disk full or allocation exceeded
4	Illegal TFTP operation
5	Unknown port number
6	File already exists
7	No such user

[1] Den femte pakettypen används för att initiera uppladdning av en fil och påminner mycket om RRQ-paketet.

Under ideala förhållanden går en filöverföring till så som illustreras i figuren nedan.



Klienten börjar med att skicka ett RRQ-paket till servern. Servern svarar med att skicka tillbaka det första datapaketet. Klienten bekräftar mottagandet genom att skicka ett ACK-paket med blocknumret 1. När servern har mottagit ACK-paketet skickas nästa datapaket som ackas med blocknummer 2, osv. Notera att även det sista datapaketet bekräftas med ett ACK.

Tyvärr medför valet av UDP som transportprotokoll att meddelanden kan försvinna eller fördröjas, vilket måste hanteras. Det gör vi enligt följande:

Om det inledande RRQ-paketet försvinner och klienten inte mottager något inledande datapaket (det kan ju i och för sig också bero på att datapaketet försvann) måste klienten sända om RRQ-paketet. Om inget svar erhålls efter flera försök att sända RRQ-paketet avbryts uppkopplingsförsöket. Vi väljer att vänta högst *2 sekunder* på svar från servern. Om inget svar erhållits efter *10 omsändningar* avbryter vi uppkopplingsförsöket.

Om ett datapaket eller ACK försvinner kommer servern att få en timeout och sända om meddelandet efter en stund. Detta innebär dock att klienten kan få flera kopior av samma datapaket skickat till sig. Klienten ska skicka ACK för varje mottaget datapaket, även dubletter, men om vi mottar ett datapaket som vi redan behandlat ska vi självklart i övrigt ignorera detta. Om klienten inte mottager något datapaket under en längre tid, vi väljer tiden *20 sekunder* för detta, avbryts filöverföringen.

Rogers kod

Programmet Roger ska skriva förväntas ta följande data som parametrar på kommandoraden:

- Adressen till servern på vilken TFTP-servern kör i form av en sträng (t.ex. pierre.cs.lth.se").
- Portnumret på vilken TFTP-servern lyssnar efter trafik.
- Namnet på filen som ska överföras så som den heter *på servern*.
- Det namn filen ska få på den lokala datorn när den har laddats ner.

Roger har skrivit ett huvudprogram som avkodar kommandoradsparametrarna enligt ovan samt öppnar destinationsfilen för skrivning. Om det senare misslyckas ges ett felmeddelande och programmet avbryts. Destinationsfilen representeras av en `FileOutputStream`. En `FileOutputStream` fungerar som vilken annan ström (`OutputStream`) som helst i Java. De bytes som överförs från servern ska alltså skrivas till denna ström.

När parametrarna avkodats och strömmen som representerar den lokala filen skapats anropas en metod `getFile()` som sköter själva överföringen. Denna metod har Roger inte hunnit skriva utan det är alltså din uppgift att skriva denna.

Metoden `getFile()` förväntas ta följande parametrar:

- `server` – Adressen till datorn som kör TFTP-servern.
- `port` – TFTP-serverns portnummer.
- `remoteFileName` – Namnet filen har *på servern*.
- `localFile` – En öppen `FileOutputStream` representerande den lokala filen.

Returvärdet från metoden `getFile()` är en `boolean` som talar om huruvida överföringen lyckades eller inte. Om allt gick bra ska metoden returnera `true`. Om det uppstod något problem, t.ex. att ett nätverksfel inträffade eller om TFTP-servern skickade ett felmeddelande (ERROR-paket), ska metoden returnera `false`. Vi bryr oss dock inte om exakt vilken typ av fel som uppstod.

Dessutom har Roger hunnit skriva metoden `b2i()` som tar två bytes (signed i Java) och omvandlar dem till ett sammansatt 16-bitars heltal. Denna metod kan vara bra att ha för att rekonstruera de blocknummer som finns i DATA-paketen.

```
import java.io.*;
import java.net.*;

public class TFTPGet {
    public static void main(String[] args) {
        if (args.length!=4) {
            System.out.println("Syntax: java TFTPGet <server> <port> "+
                "<remote file name> <local file name>");
            System.exit(1);
        }
        String server = args[0];
        String remoteFileName = args[2];
        String localFileName = args[3];
        int port = 0;
        try {
            port = Integer.parseInt(args[1]);
        } catch(NumberFormatException e) {
            System.out.println("Unknown port number.");
            System.exit(1);
        }
        FileOutputStream file = null;
        try {
            file = new FileOutputStream(localFileName);
        } catch(FileNotFoundException e) {
            System.out.println("Could not create local file "+localFileName+".");
            System.exit(1);
        }

        boolean success = getFile(server,port,remoteFileName,file);

        if (!success) {
            System.out.println("File transfer failed!");
        }
        try {
            file.close();
        } catch (IOException e) { }
    }

    // Converts two signed bytes into a (unsigned) 16-bit int
    private static int b2i(byte b1,byte b2) {
        return (0xff & (int)b1)<<8 + (0xff & (int)b2);
    }

    private static boolean getFile(String server, int port, String remoteFileName,
         FileOutputStream localfile) {

        // TO DO
    }
}
```

Uppgiften
Din uppgift är att komplettera metoden `getFile()` i koden ovan enligt beskrivningen. Metoden är endast avsedd att överföra filer i binärmod (*octet*). Vill du införa extra metoder/klasser är du välkommen att göra det.

Ledning: Med hjälp av metoden `void setSoTimeout(int timeout);` i socket-klassen `DatagramSocket` kan man ange att ett `SocketTimeoutException` ska genereras angivet antal millisekunder efter anrop av `receive()` om inget paket mottagits. Metoden `void setSoTimeout()` kan i sig kasta ett `SocketException`. Se även kursboken sidan 444.

(10p)

2. Datingservice

Ett system för elektronisk dating via chat skall skrivas. En elektronisk date går till så att den som vill chatta med en person av motsatt kön startar programmet Telnet för att ansluta till en datingserver. Väl ansluten till servern börjar man med att ange om man är man eller kvinna.

En date, eller en man och en kvinna, paras ihop när såväl en man som en kvinna har anslutit till datingservern. När detta sker ska en privat chatförbindelse mellan dessa två skapas. Det kan finnas flera sådana parallellt pågående privata chattar. Om en person av motsatt kön inte finns ansluten just nu ska personen som kopplar upp sig informeras om att han måste vänta tills någon lämplig partner kopplar upp sig. När sedan paret är hopkopplat ska de båda informeras om detta så att de kan börja chatta med varandra.

Din uppgift är att skriva en datingserver enligt beskrivningen ovan och nedanstående instruktioner:

1. Servern ska vara så robust så att den överlever om förbindelsen med en klient kopplas ned.
2. Om en klient försvinner (kopplar ned förbindelsen) medan den väntar på någon att chatta med behöver servern inte detektera detta särskilt utan när en person av andra könet så småningom kopplas ihop med den försvunna klienten så räcker det att den nyuppkopplade upplever det som om motparten omedelbart kopplade ner.

(10p)