Tentamen
Nätverksprogrammering
Del 1

2014–06–02, 08.00–13.00

Tillåtna hjälpmedel för denna del av tentamen: *inga*. Kurslitteratur och andra hjälpmedel för del 2 av tentamen skall förvaras på golvet bredvid bordet eller vid salens vägg.

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 1. När du löst uppgifterna i denna del av tentamen lämnar du in din lösning i det vita tentamensomslaget varvid du erhåller del 2 av tentamen tillsammans med ett nytt, färgat, tentamensomslag som skall användas vid inlämning av din lösning på del 2 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 mer, så gör så många uppgifter du kan.

---

1. Man brukar dela upp mjukvaran som har med nätverkskommunikation i olika lager. Ofta brukar man kalla ett av dessa lager för *transportlagret*.

   a) Ge två exempel på protokoll som vi behandlat i kursen och som brukar hänföras till transportlagret.

   (1p)

   b) Beskriv kortfattat vad som kännetecknar funktionaliteten i transportlagret; vilken abstraktionsnivå ligger de erbjudna tjänsterna på?

   (1p)

2. Vilka av följande sex påståenden är korrekta?

   a) En s.k. RFC (Request For Comments) är ett dokument som till exempel kan beskriva hur ett visst nätverksprotokoll fungerar ända ner på bitnivå.

   b) Portnummer 1-1023 är normalt reserverade för TCP-trafik.

   c) I Java förekommer två mer eller mindre parallella klasshierarkier för I/O, nämligen strömmar respektive readers/writers. Den övergripande skillnaden mellan klasserna i respektive hierarki är att strömmar arbetar med tecken (char/String) medan readers/writers hanterar bytes.

   d) När en webbserver får en begäran från en klient om en JSP-sida kommer webbservern att utifrån en JSP-fil automatiskt generera, kompilera och köra en s.k. servlet.

   e) En s.k. servlet kan vara skriven i olika språk, t.ex. C eller PHP likväl som Java.

   f) RTCP (Real-Time Control Protocol) är en del av protokollet RTP (Real-time Transport Protocol) och används för att kontrollera överföringen av strömmande data, t.ex. starta eller stoppa uppspelningen. Det fungerar ungefär som en fjärrkontroll.

   g) RTP (Real-time Transport Protocol) är baserat på UDP medan det modernare protokollet DASH är baserat på HTTP och TCP.

   h) Ett objekt av typen `MulticastSocket` i Java kan användas för att skicka vanliga datagram (unicast) såväl som multicastdatagram. Det som styr huruvida det blir ett unicast- eller multicastmeddelande är endast IP-numret som angetts som meddelandets mottagare.

   (4p)

3. Antag att vi ska skriva en TCP-baserad server i Java som tar emot en uppkoppling, läser in någon slags begäran från klienten, skickar svar på begäran, tar emot en ny begäran, osv. Slutligen kopplas förbindelsen ned. Vi kan då välja att designa vår server på lite olika sätt. Vi kan t.ex.:

   Låta servern vara enkeltrådad och denna tråd sköter både uppkoppling och sedan även inläsning av begäran/generering av svaret. Först när nedkoppling av första klienten skett behandlar vi nästa klient. Vi använder Javas standardklasser `Socket`, `ServerSocket` och Javas s.k. strömmar för att sköta kommunikationen. Fördelen är att det blir en enkel design som är lätt att implementera. Nackdelen är att vi kan få långa svarstider om en enskild begäran från en klient tar lång tid att behandla – under tiden kan vi ju inte behandla andra klienters uppkopplingar.

   Beskriv kortfattat (med motsvarande abstraktionsgrad som ovan) två (2) andra möjliga, men effektivare, designalternativ för vår server.

   (2p)

4. Per Krona har fått jobb som programmerare på Snålbanken. Eftersom deras datorsystem har krånglat (pengar har mystiskt förvunnit från eller dykt upp på kundernas konton) har man bestämt sig för att själva skriva om hanteringen av kundernas konton i Java. Kontona representeras av objekt av nedanstående Javaklass. Systemet är multitrådat varför man har skyddat operationerna genom att deklarera dem `synchronized`:

```
public class Account {

    /* Saldo. Anges i ören för att undvika avrundningsfel. */
    private long balance;

    public synchronized long getBalance() {
        return balance;
    }

    public synchronized void setBalance(long new_balance) {
        balance = new_balance;
    }
}
```

   Insättningar och uttag genomförs på nedanstående sätt. Variabeln `account` antas vara objekt av typen `Account` och `sum` är summan som ska sättas in eller tas ut.

```
% Insättning
long b = account.getBalance();
b = b + sum;
account.setBalance(b);

% Uttag
long b = account.getBalance();
b = b - sum;
account.setBalance(b);
```

   Din uppgift är att hjälpa Per Krona med att designa/skriva om klassen Account så att insättningar/uttag alltid fungerar korrekt. Ett designkrav är att all eventuell synkronisering hanteras i klassen Account. Svara med en omdesignad/omskriven klass `Account` och uppdaterad exempelkod för hur insättning respektive uttag görs.

   (3p)

5. I Javas klass `InputStream` finns en operation `read` med följande signatur (motsvarande operation finns även i C/C++):

```
public int read(byte[] input,int offset, int length) throws IOException;
```

Antag att vi vill använda operationen för att läsa in 100 byte från en nätverksström `input` till en byte-vektor `buffer` (med start i position 0). Om vi antar att det finns en omgivande try-catch-sats som fångar eventuella exceptions verkar det lockande att skriva så här:

```
input.read(buffer,0,100);
```

a) Varför är det ovan beskrivna sättet att läsa in 100 byte olämpligt?

(1p)

b) Skriv en korrekt kodsekvens som använder operationen `read` som den är deklarerad ovan för att läsa in 100 byte.

(2p)

6. I ett javaprogram behöver en tråd vänta (en gång) tills en annan parallell tråd har avslutat någon typ av beräkning som den första tråden är beroende av innan den kan fortsätta. I ett första förslag till lösning har man skrivit följande Javakod för tråden (för problemet irrelevant kod har ersatts med "..."):

```
public class WaitingThread extends Thread {

    private boolean finished = false;

    public void releaseThread() {
        finished = true;
    }

    public void run() {
        ...

        /* Wait for computation to finish. */
        while (!finished) {}

        /* The external thread has now finished its computation */
        ...
    }
}
```

Koden fungerar så att den parallella, externa, tråden anropar metoden `releaseThread()` när den aktuella beräkningen har avslutats. Om tråden `WaitingThread` kommer fram till while-satsen i metoden `run()` innan den parallella tråden har exekverat `releaseThread()` kommer while-satsen att göra att trådens exekvering hejdas (dvs att efterföljande rader inte exekveras) tills dess `releaseThread()` anropats. Omvänt, om `releaseThread()` anropats innan while-satsen nåtts kommer exekveringen *inte* fördröjas. Tyvärr är lösningen baserad på en s.k. *busy-wait*, vilket gör att mycket CPU-tid går åt i onödan till att vänta på att beräkningen ska avslutas.

Skriv om klassen `WaitingThread` ovan med hjälp av Javas inbyggda monitorbegrepp så att problemet elimineras.

(3p)

7. För att kommunicera över TCP i Java använder man traditionellt klasserna `Socket`, `ServerSocket`, `InputStream` och `OutputStream`. I Java 1.4 introducerades dock paketet `java.nio` med bl.a. klasserna `SocketChannel`, `ServerSocketChannel`, `Selector`, med flera.

   a) Modellen för hur man gör för att läsa från eller skriva till en socket har förändrades ganska radikalt i och med att paketet `java.nio` infördes. Beskriv kortfattat (max tre meningar) hur den nya modellen skiljer sig från den gamla.

   (1p)

   b) Nämn en betydande fördel som kan finnas med att använda det nya paketet (`java.nio`) i stället för det gamla.

   (1p)

   c) Vilken roll har klassen `Selector` i paketet?

   (1p)

Slut på del 1 – lämna in och hämta ut del 2!