Tentamen
Nätverksprogrammering
Del 2

2009–06–01, 14.00–19.00

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 kan ge 15 respektive 5 poäng vardera. Uppgifternas ordningsföljd avspeglar inte nödvändigtvis deras svårighetsgrad.

---

1. Filöverföring

Ett programsystem för överföring av filer från skivminnet på en dator till skivminnet på en annan dator via nätverket ska skrivas. Systemet innefattar två program som ska kommunicera med varandra: en serverdel (FileServer.java) som startas på den ena av datorerna och en klientdel (FileClient.java) som startas på den andra datorn och från vilken överföringen styrs.

En session ska kunna gå till så här:

1. Användaren startar programmet FileServer på den ena datorn (i detta exempel antar vi att datorn heter ”gote.cs.lth.se”):

   ```
   gote(roger)% java FileServer
   ```

2. Användaren startar programmet FileClient på den andra datorn (i vårt exempel heter den "ygg.cs.lth.se"))och anger samtidigt namnet på den första datorn:

   ```
   ygg(roger)% java FileClient gote.cs.lth.se
   ```

3. Klientprogrammet ska nu vänta på kommandon från användaren via tangentbordet. Vi antar att användaren vill flytta över en fil till datorn med serverprogrammet på. Han skriver då (om han till exempel vill överföra filen FileClient.java):

   ```
   s FileClient.java
   ```

   Filen ska nu överföras till serverdatorn och lagras på skivminnet med samma namn som originalfilen hade. Klientprogrammet ska därefter vara berett att ta emot ett nytt kommando.

4. Om vi antar att användaren därefter vill hämta en fil från serverdatorn (till exempel FileServer.java) skriver han:

   `g FileServer.java`

   Filen FileServer.java hämtas därvid via nätverket och lagras på det lokala skivminnet under samma namn.

5. Användaren fortsätter att ge kommandon för att hämta/sända filer så länge han vill. Därefter avslutar han både klient- och serverprogrammet genom att ge kommandot "q" till klienten.

Skriv de två programmen FileServer.java respektive FileClient.java!

Tips och anvisningar

- Serverprogrammet ska kunna hantera flera samtidigt aktiva klienter.
- Välj själv lämpligt kommunikationsprotokoll samt definiera lämpligt applikationsprotokoll.
- Felhanteringen behöver inte vara helt perfekt; skulle ett nätverksfel uppstå räcker det att avbryta klienten på ett kontrollerat sätt med ett lämpligt felmeddelande. Du kan också anta att användaren ger korrekta kommandon. Servern måste dock vara så pass robust att den klarar att förbindelsen till en klient bryts utan att andra uppkopplingar påverkas.
- Du behöver inte kunna hantera sökvägar till filerna som flyttas, utan vi kan anta att alla filer ligger/placeras i den katalog som klienten/respektive servern startades i.
- Filer på skivminnet kan läsas och skrivas som vilken ström som helst med hjälp av klasserna FileInputStream respektive FileOutputStream i paketet java.io. Du kan öppna en fil för läsning genom att skriva:

  ```
  String filnamn = ...
  FileInputStream file = null;
  try {
      file = new FileInputStream(filnamn);
  } catch(FileNotFoundException e) { ... }
  ```

  Att öppna en fil för skrivning gör du på motsvarande sätt genom att skriva:

  ```
  String filnamn = ...
  FileOutputStream file = null;
  try {
      file = new FileOutputStream(filnamn);
  } catch(FileNotFoundException e) { ... }
  ```

  I övrigt fungerar filerna som vilken ström som helst, se kapitel 4 i Java Network Programming.

- Det kan det vara praktiskt – men inte strikt nödvändigt – att kunna avgöra hur stor en fil är (= hur många bytes den innehåller). Detta kan göras genom att man anropar operationen `available()` på ett objekt av typen `FileInputStream` direkt efter att det skapats. Observera att available() kan generera ett `IOException`.

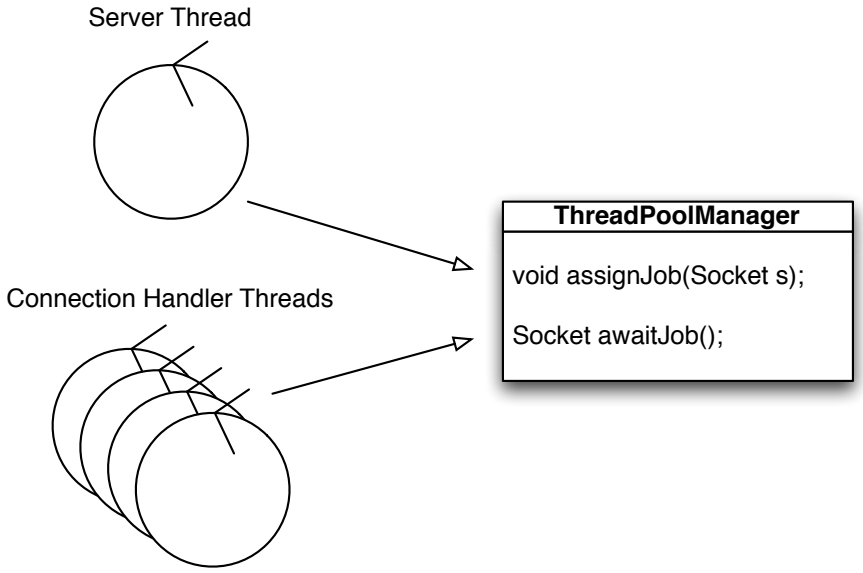
  Exempel:

  ```
  String filnamn = ...
  FileInputStream file = null;
  int fileLength = 0;
  try {
      file = new FileInputStream(filnamn);
  } catch(FileNotFoundException e) { ... }
  try {
      fileLength = file.available();
  } catch(IOException e) { ... }
  ```

(15p)

2. Trådpool i en server

På en av föreläsningarna i kursen diskuterade vi olika sätt att göra tråddesignen för en server som ska kunna betjäna flera klienter samtidigt. Ett vanligt designmönster är att ha en pool av trådar som står beredda att serva en klient vid behov. När en klient kopplar upp sig mot servern utses en av de lediga trådarna i trådpoolen att hantera just den klienten. När klienten kopplar ned sig återgår den utsedda tråden till att vänta på att få ett nytt uppdrag. En TCP-baserad server kan då se ut ungefär som i följande figur:



Här finns det en tråd, benämnd ”Server Thread” i figuren, som väntar på nya TCP-uppkopplingar (`accept();`). När en ny uppkoppling har skett anropar tråden metoden `assignJob()` i ett objekt av klassen `ThreadPoolManager` och skickar med en referens till den socket som motsvarar uppkopplingen. Trådarna benämnda ”Connection Handler Threads” i figuren tar i sin tur hand om uppkopplingarna. En ledig tråd utses att ta hand om klienten som kopplade upp sig. När en av dessa trådar inte har något att göra anropar den metoden `awaitJob()` i `ThreadPoolManager`. Tråden blockeras därmed tills den utsetts att betjäna en ny uppkoppling varvid en referens till motsvarande socket returneras. Om ingen tråd finns tillgänglig för att betjäna klienten för tillfället köas begäran upp tills någon tråd anropar `awaitJob()`.

Uppgift

Implementera klassen `ThreadPoolManager` fullständigt med hjälp av Javas inbyggda monitorbegrepp.

(5p)

Slut – Glad sommar!