Tentamen
Nätverksprogrammering
Del 2

2014–06–02, 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.

Denna del består av två deluppgifter om 15 respektive 5 poäng. 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.

---

1. Nätverksbaserat bokningssystem

Din uppgift är att (i Java och med de tekniker vi behandlat i kursen) implementera ett enkelt klient-server-baserat system för att hantera bokningar av olika resurser, t.ex. lokaler, över nätet. Det antas att inga bokningar sträcker sig från ett dygn till ett annat. Man ska kunna:

- Lägga till en ny bokningsbar resurs i systemet.
- Skapa en ny bokning av en resurs.
- Ta bort en bokning.
- Visa alla bokningar av en resurs för en viss dag.
- Bevaka bokningsläget för en resurs en viss dag. Detta innebär att varje gång en bokning av resursen tillkommer eller försvinner för den aktuella dagen så visas automatiskt en uppdaterad lista över bokningar.

Skriv ett javaprogram `BookingServer` och ett program `BookingClient` (tillsammans med eventuella hjälpklasser du kan behöva) som implementerar server- respektive klientdelen av systemet. Programmen används genom att ge kommandon i ett kommandoradsfönster på följande sätt (du kan anta att användaren alltid skriver korrekta kommandon formaterade på rätt sätt):

Starta servern

```
java BookingServer
```

Lägga till en ny resurs
`java BookingClient <SERVER> create <RESURSNAMN>`
Byt ut `<SERVER>` mot serverns adress och `<RESURSNAMN>` mot namnet på den nya resursen. Det ska skrivas ut ett meddelande som talar om huruvuda operationen lyckades eller inte. Därefter avslutas programmet.

Ledning: Om du deklarerat din main-metod enligt ”`public static void main(String[] args)`” kommer `args[0]` att innehålla serverns adress, `args[1]` texten ”create” och `args[2]` namnet på den nya resursen.

Skapa en ny bokning
`java BookingClient <SERVER> book <RESURSNAMN> <DATUM> <STARTTID> <SLUTTID> <BESKRIVNING>`
Här byts `<DATUM>`, `<STARTTID>` och `<SLUTTID>` ut mot datum/tid för bokningen på formen ÅÅÅÅ-MM-DD respektive TT:MM. `<BESKRIVNING>` är en beskrivning av bokningen, t.ex. namnet på den som bokar. Om beskrivningen består av flera ord förväntas användaren skriva beskrivningen inom citationstecken; på så sätt kommer hela den beskrivande texten att hamna i `args[6]`.

Det ska skrivas ut ett meddelande som talar om huruvuda bokningen lyckades eller inte. Därefter avslutas programmet.

Ta bort en bokning
`java BookingClient <SERVER> delete <RESURSNAMN> <DATUM> <STARTTID>`
Det ska meddelande huruvida operationen lyckades eller inte. Därefter avslutas programmet.

Visa alla bokningar för en viss dag
`java BookingClient <SERVER> show <RESURSNAMN> <DATUM>`
Lämplig utskrift är de strängar som returneras av metoden `show` i klassen `BookingManager` (se nedan) – en sträng på varje rad. Därefter avslutas programmet.

Bevaka bokningsläget för en viss dag
`java BookingClient <SERVER> watch <RESURSNAMN> <DATUM>`
En likadan utskrift som den som kommandot ”show” resulterade i visas. Till skillnad från övriga kommandon ska förbindelsen till servern bibehållas i väntan på att bokningsläget för den angivna resursen och det angivna datumet ska förändras. När detta sker ska en ny bokningslista skrivas ut, osv, tills man stoppar programmet med CTRL-C. En ny lista ska bara skrivas ut om en ändring gjorts för den aktuella resursen och på det aktuella datumet.

Ledning: Genom att varje gång en ändring gjorts i databasen jämföra den vektor av strängar som metoden `show` i klassen `BookingManager` returnerar med den vektor som returnerades av metoden i samband med den förra utskriften kan man avgöra om en ändring som motiverar ny utskrift skett. För att jämföra innehållet i två vektorer (`v1` och `v2`) med strängar kan du skriva `Arrays.equals(v1,v2)`.

För att hålla reda på alla bokningarna på servern kan du använda nedanstående färdiga klass på serversidan. Observera att klassen *INTE* är trådsäker.

```
class BookingManager {
        /** Skapar ett objekt som håller reda på bokningar av ett antal namngivna
            resurser i en databas. */
        public BookingManager();

        /** Skapar en ny resurs i databasen för en namngiven resurs. Returnerar
            true om resursen kunde skapas. Om en resurs med angivet namn redan
            fanns returneras false. */
        public boolean create(String name);
```

```
        /** Skapar en ny bokning för angiven resurs. Datum anges som en sträng
            på formen ÅÅÅÅ-MM-DD och start- respektive sluttider som strängar på
            formen "TT:MM". En beskrivande text kan även anges. Om bokningen
            lyckades returneras true. Om resursen var upptagen returneras false. */
        public boolean book(String resource, String date, String start_time,
                            String end_time, String description);

        /** Tar bort bokningen av angiven resurs som börjar på angiven tid.
            Om ingen matchande bokning hittas returneras false, annars true. */
        public boolean delete(String resource, String date, String start_time);

        /** Returnerar en vektor med alla bokningar av den angivna resursen på
            den angivna dagen. Varje bokning representeras av en sträng med
            start- och stopptid följd av beskrivningen av aktiviteten. Datum
            anges som för metoden book. Vektorn är tom om inga bokningar hittas. */
        public String[] show(String resource, String date);
}
```

(15p)

2. Adressöversättning

Antag att vi har en lista av symboliska internetadresser, som t.ex. ”cs.lth.se” eller ”login.student.lth.se”, och vill översätta dessa till motsvarande IP-nummer i strängform, som t.ex. ”130.235.209.220” eller ”130.235.35.99”. Då skulle vi kunna skriva en klass enligt följande som gör det åt oss:

```
public class URLMapper {

    /** Translates the symbolic internet addresses in name_list into
        IP number form (as strings). Returns a list with the
        translated addresses. */
    public static List<String> mapper(List<String> name_list)
                            throws InterruptedException, ExecutionException {

        // Put code here

    }

}
```

Ett problem som uppstår om listorna är stora är att varje enskild översättning kommer att kräva att en förfrågan sänds till en DNS-server över nätverket, vilket tar tid. Därför skulle man vilja parallellisera översättningen så att flera förfrågningar kan vara aktiva samtidigt.

Implementera klassen `URLMapper` med metoden `mapper` ovan (och eventuella hjälpklasser) så att den använder Executors[1] för att skapa en trådpool bestående av tre parallella trådar för att göra översättningen.

Om du inte har den senaste versionen av kursboken (upplaga 4) finns det ett utdrag om Executors du kan hämta framme vid låneböckerna.

(5p)

Slut – Glad Sommar!

[1] Därav `throws`-deklarationen av metoden `mapper`.