Tentamen
Nätverksprogrammering
Del 2

2012–05–28, 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.

---

1. Buddylist

När man som vi i E-Huset har ett antal olika utspridda datorrum kan det vara både trevligt och också ofta praktiskt att se vilka av ens kompisar/projektkamrater som är inloggade för tillfället och även vid vilken dator de sitter. På så sätt kan man ju till exempel gå dit och fråga dem något eller samla ihop sig i samma rum ifall man arbetar på något gemensamt projekt. Denna uppgift går därför ut på att skriva ett litet Javaprogram som kontinuerligt visar vilka av ens kompisar (som också kör samma program) som för tillfället är inloggade och vid vilken maskin de sitter. När man startar programmet anger man sitt eget namn (”nickname”/ alias) på kommandoraden och detta namn kommer sedan att presenteras för kamraterna.

Den övergripande designen för programmet är given och beskrivs nedan. Din uppgift blir sedan att i ett antal deluppgifter implementera de olika delarna av den givna designen.

Övergripande design

Strategin för programmet blir att med hjälp av multicast (kursboken kapitel 14) regelbundet (cirka en gång i minuten) skicka ut en förfrågan på det lokala nätverket om vilka som för närvarande finns inloggade (och var). När motsvarande program (som kompisarna startat) på de andra datorerna tar emot förfrågan skickar de ett svar, lämpligen i form av ett datagram, till den dator ifrån vilken förfrågan gjordes och i vilket de talar om vem de är (namn eller ”nickname”/alias). Efterhand som svaren anländer till den frågande datorn listas namnen för användaren på lämpligt sätt.

De tre olika faserna ovan (regelbundet skicka ut förfrågan, lyssna efter och svara på andras förfrågan och ta emot svar från andra inloggade) hanteras i programmet av tre separata trådar som beskrivs närmare i deluppgifterna nedan.

Färdig hjälpklass
Ett trevligt sätt att presentera den färdiga listan över ”buddies” vore att ha ett litet fönster på skärmen. Eftersom detta ligger utanför ramen för kursen att skriva koden för detta kan du förutsätta att följande hjälpklass finns tillgänglig. Den visas nedan med en komplett implementation som (för teständamål) bara skriver ut informationen i ett terminalfönster.

```
public class BuddyWindow {

    public BuddyWindow() { }

    public synchronized void clear() {
        System.out.println("--------------------");
    }

    public synchronized void addBuddy(String buddy) {
        System.out.println(buddy);
    }
}
```

Klassen används så här:

När programmet sänder ut en förfrågan på nätverket anropar den även metoden[1] `clear()` i ett objekt av klassen `BuddyWindow`. Då raderas innehållet i det tänkta ”buddiefönstret” (eller som ovan, skriver ut en markering att nu följer namnen på de nu inloggade kompisarna). Därefter, efter hand som svar anländer via nätverket från de andra inloggade, anropas metoden `addBuddy()` med en sträng som parameter som anger namnet på kompisen och vid vilken maskin han sitter. En sådan sträng kan förslagsvis se ut ungefär så här: ”Pelle (hacke-1.student.lth.se)”.

a) Skriv en trådklass kallad `BuddyRequester` som har hand om att regelbundet skicka ut förfrågningar om vilka som är inloggade på nätverket. Låt tråden köra regelbundet, lämpligen cirka var 60:de sekund. Varje gång den kör bör den anropa metoden `clear()` i ditt `BuddyWindow`-objekt för att markera att nu ska gamla uppgifter raderas. Därefter skickar den ut ett multicastmeddelande med lämpligt innehåll på nätverket.

(4p)

b) Skriv en trådklass kallad `BuddyServer` som lyssnar efter förfrågningar skickade med multicast från andra datorer enligt föregående deluppgift. När förfrågan anländer skall svar skickas till den dator från vilken förfrågan kom och på ett sätt som du bestämmer (t.ex. i form av ett datagram skickat till lämplig port). Du bestämmer även innehållet i svarsmeddelandet.

(5p)

c) Skriv en trådklass kallad `BuddyListener` som kontinuerligt lyssnar efter svar på de förfrågningar som trådklassen `BuddyRequester` skickar ut. När ett sådant svar anländer ska kompisens namn (”nickname”/alias) extraheras ur svaret. Det ska även analyseras från vilken dator (datornamn/internetadress) svaret kom. En lämplig presentationssträng ska därefter konstrueras och skickas som parameter till metoden `addBuddy()` i ditt `BuddyWindow`-objekt.

(4p)

d) Skriv en klass `BuddyList` innehållande en main-metod som skapar de objekt du behöver, kopplar ihop objekten på det sätt som kan tänkas behövas samt startar de tre trådarna beskrivna ovan.

(1p)

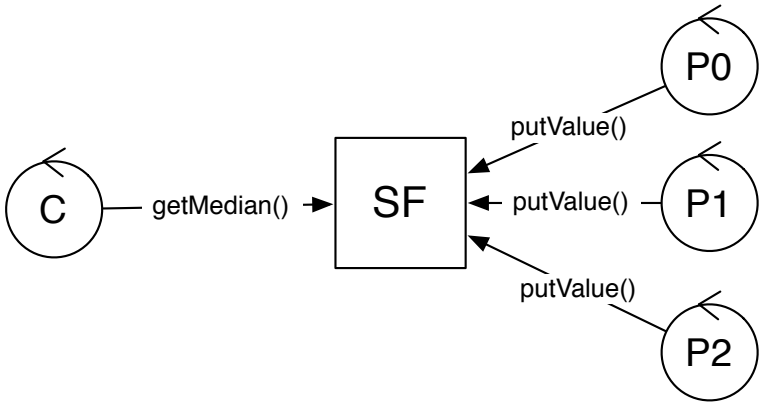
[1] Metoderna `clear()` och `addBuddy()` i klassen `BuddyWindow` är deklarerade `synchronized` för att göra det möjligt att anropa dem från olika trådar utan att riskera eventuella synkroniseringsproblem vid en utbyggnad av klassen. Med den visade implementationen är synchronized-deklarationen faktiskt onödig, men kan eventuellt bli nödvändig om vi ändrar presentationssätt.

2. Filtrering av sensorvärden

Inom processautomation börjar det bli allt vanligare med nätverk av datorer som tillsammans styr processen. För att få indata till styrningen behöver dessa datorer tillgång till sensorer som känner av processens aktuella tillstånd. Värdena från dessa sensorer används sedan för att styra processen. Sensorerna är ofta distribuerade och kopplas även dessa in i systemet via nätverket.

I ett visst system, som uppgiften kommer att handla om, har man anslutit tre oberoende sensorer, som mäter samma sak, till nätverket. Avsikten är att man läser ner ett mätvärde från var och en av sensorerna och sedan använder man sig av medianvärdet av de tre mätvärdena för att styra processen. På detta sätt filtreras felaktiga mätvärden från en enstaka, trasig, sensor bort och påverkar ej processtyrningen.

Filtreringen kan beskrivas av nedanstående figur där P0, P1 och P2 är trådar som läser ner mätvärden via nätverket från sensorerna. C är en tråd som tar emot filtrerade mätvärden från monitorn SF som är av typen `SensorFilter`. Monitorn SF lagrar de tre olika värdena från sensorerna och hanterar synkroniseringen mellan trådarna.



Din uppgift är att implementera klassen `SensorFilter` i Java som beskriver monitorn SF. Använd Javas inbyggda monitorbegrepp för att korrekt synkronisera trådarna. Utgå från följande klasskelett och komplettera med attribut, innehåll i metoderna och andra nödvändiga deklarationer.

```
public class SensorFilter {

    public SensorFilter() {
        ...
    }

    public void putValue(int id, double val) {
        ...
    }

    public double getMedian() {
        ...
    }
}
```

Förklaring av monitormetoderna:

- `void putValue(int id, double val)` — De tre trådarna P0, P1 och P2 innehåller en while-loop som läser in ett värde (av typen `double` från sin respektive sensor. Detta värde läggs in i monitorn SF genom ett anrop av `putValue()`. Som parametrar till anropet ges dels trådens ”id”-nummer (0 för P0, 1 för P1 och 2 för P2) och dels det inlästa sensorvärdet.

  För att undvika att skriva över tidigare sensorvärden som tråden C ännu inte hunnit behandla, samt för att se till att de tre trådarna P0, P1 och P2 går i samma takt, ska det nya värdet inte läggas in förrän tråden C (via ett anrop av `getMedian()`) tagit hand om det tidigare värdet. Om

det finns ett tidigare obehandlat värde i monitorn SF ska alltså tråden som anropar `putValue()` *blockeras* tills C har tagit hand om det tidigare värdet.

- `double getMedian()` — Tråden C anropar denna metod för att dels invänta att mätvärden anlänt från alla tre sensorerna och dels för att beräkna medianvärdet av dessa tre mätvärden. Medianvärdet returneras som resultatet av funktionen. Om mätvärden ännu inte anlänt från alla tre sensorerna ska den anropande tråden (C) *blockeras* tills mätvärden från alla tre sensorerna finns tillgängliga.

(6p)

Glad sommar!