Tentamen
EDA095 Nätverksprogrammering
Del 2

2016–05–31, 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.

Observera: Läs igenom båda deluppgifterna i denna tentamen innan du börjar lösa dem.

---

1. Radio över Internet

När kursansvarig för kursen Nätverksprogrammering började doktorera 1991 noterade han att någon på institutionen vid något tillfälle hade lött ihop en kabel som gjorde att man kunde koppla valfri audioutrustning som var försedd med ett hörlursuttag till ljudingången på de Sun 3/60-maskiner institutionen då hade. På hyllan bredvid stod en gammal transistorradio.

Lite efterforskning visade dessutom att det var ganska lätt att använda sig av ljudingången (och ljudutgången) mjukvarumässigt. Den inbyggda hårdvaran omvandlade den analoga inkommande signalen till en ström av bytes med hastigheten 8192 bytes/sekund (8-bitars sampel med en samplingsfrekvens på ca 8 kHz). För att spela upp denna digitaliserade ljudsignal i den inbyggda högtalaren i datorn behövde man bara i sin tur skicka denna insamplade dataström tillbaka till ljudhårdvaran som omvandlade den till en analog signal som spelades upp i högtalaren.

Idén var given: Varför inte ansluta transistorradion till ljudingången på en maskin, skriva ett serverprogram (som körde på denna maskin) som läste in den digitaliserade ljudsignalen och skickade ut den via nätverket till alla på institutionen som ville lyssna på radio? Varje användare startade ett klientprogram som kopplade upp sig mot servern och tog emot ljuddata som sedan skickades till högtalaren. Internetradion var född![1]

(fortsättning på nästa blad)

[1] När applikationen började bli populär förbättrades den dessutom ytterligare genom att servern höll reda på vid vilka tidpunkter som Sveriges Radio sände nyheter och började då spela in dessa genom att spara den digitaliserade ljudströmmen till en fil. Detta gjorde att alla sedan alltid kunde lyssna på de senast sända nyheterna när de ville.

Uppgiften består av att i ett par steg implementera ett klient/server-system i Java enligt beskrivningen ovan. Till din hjälp har du följande färdigskrivna klass som kan användas för att läsa in digitaliserat ljud till en ström av bytes eller omvänt skicka den digitala strömmen till högtalaren:

```
class SoundIO {
        /** Skapar ett objekt med vars hjälp man kan läsa in en digitaliserad
            signal från ljudingången eller skicka motsvarande signal till
            högtalaren */
        public SoundIO();

        /** Läser in så många bytes som har hunnit samplas in sedan operationen
            anropades senast, och lagrar dessa i vektorn "data". Om fler bytes
            finns tillgängliga än vad som ryms i "data" lagras endast så många
            bytes som får plats i "data". Resten sparas till nästa anrop av "read".
            Det antal bytes som verkligen lagrades i "data" returneras som resultat
            av anropet. Skulle inga data finnas tillgängliga blockerar anropet av "read"
            tills minst 128 bytes data finns tillgängliga. */
        public int read(byte [] data);

        /** Skickar de "n" stycken första bytes i vektorn "data" till högtalaren
            för omvandling till en analog signal och uppspelning. */
        public void write(byte [] data,int n);
}
```

Av effektivitetsskäl kan det vara lämpligt att *försöka* läsa och skriva 1024 bytes eller mer varje gång man anropar read/write ovan. Storleken på den vektor data som används som inparameter bör alltså vara minst 1024 bytes.

a) Skriv ett klientprogram som kopplar upp sig via nätverket till angiven servermaskin och börjar ta emot en kontinuerlig ström av ljuddata från denna. Skicka dataströmmen till högtalaren. Klientprogrammet ska kunna startas genom att man skriver

   `java RadioClient server`

   där server är namnet på den maskin där radioservern kör. För att avsluta klientprogrammet är det acceptabelt att man helt enkelt stoppar det genom att trycka ctrl-c (dvs tvingar programmet att avslutas utifrån). Du behöver alltså då inte själv hantera detta.

   Bestäm själv lämpligt protokoll och implementationsteknik, men klienten ska fungera ihop med servern som implementeras i nästa deluppgift. *Läs därför igenom även b-uppgiften innan du designar din klient*.

   (5p)

(fortsättning på nästa blad)

b) Skriv ett serverprogram som fungerar tillsammans med klientprogrammet du utvecklade i uppgift a. Vi ställer följande krav på serverprogrammet:

1) Ljuddata ska (naturligtvis) spelas upp i samma ordning som den generades. Bitar av ljudströmmen får alltså aldrig kastas om.
2) Flera klienter ska kunna vara anslutna till servern samtidigt och få alla ljuddata skickade till sig. Det vill säga att de bytes som läses med hjälp av SoundIO.read ovan måste skickas ut till samtliga anslutna klienter (de måste alltså dela på ljudströmmen).
   Ledning: Man kan alltså inte ha en tråd för varje ansluten klient där varje tråd anropar SoundIO.read. Då kommer ju ljudströmmen att delas upp i fragment som skickas ut till olika klienter.
3) Servern ska vara robust i den meningen att den ska klara av att förbindelsen till en klient bryts (t.ex. genom att användaren stoppar klientprogrammet med ctrl-c). Servern ska då sluta skicka data till den klienten.
4) För att minska CPU-belastningen på den maskinen som kör serverprogrammet ska servern vara skriven så att om ingen klient är ansluten ska den sluta anropa operationen SoundIO.read för att läsa in mer ljuddata. När någon klient därefter kopplar upp sig ska servern återigen börja anropa SoundIO.read för att erhålla data att skicka till den anslutna klienten.

Poängguide
Försök implementera kraven ovan så väl som möjligt! Gör du avsteg från kraven kommer poängavdrag att göras, men det är bättre att lösa en del av uppgiften än att inte lösa den alls. Nedan ges exempel på hur olika tänkbara inskränkningar påverkar maximalpoängen för uppgiften:

- En korrekt lösning, men som bortser från krav nummer 4 (effektivitetskravet) ger ett avdrag på 2 poäng.
- En korrekt lösning som fungerar tillsammans med flera klienter, men som ignorerar eventuella synkroniseringsbehov mellan eventuella ingående trådar ger 5 poängs avdrag.
- En korrekt lösning som kan hantera flera klienter samtidigt, men som inte klarar av att klienter kopplas ned ger 5 poängs avdrag.
- En korrekt lösning, men som bara kan hantera en uppkopplad klient åt gången ger 10 poängs avdrag.

(15p)

Slut!