Tentamen
Nätverksprogrammering
Lösningsförslag

2012–05–28, 8.00–13.00

Del 1

1. Time To Live (TTL) anger hur många routrar ett multicastpaket får passera innan det tas bort. Det används för att begränsa spridningen av meddelanden till en begränsad del av nätverket.

2. Den s.k. MIME-taggen används för att tala om vilken typ av innehåll ett dokument har, t.ex. html-kod i textform eller en JPEG-bild.

3. Både SAX och DOM är XML-parsrar, men fungerar lite annorlunda. DOM bygger upp ett träd som representerar XML-dokumentet på vilket man sedan kan utföra godtyckliga operationer (på hela eller delar av trädet) medan SAX inte sparar någon fullständig representation av dokumentet. I det senare fallet kan man utföra operationer efterhand som parsern stöter på de olika taggarna.

4. Busy-wait innebär att en tråd förbrukar all tillgänglig CPU-tid för att vänta på att en händelse ska inträffa, t.e.x att en annan tråd ändrar ett värde. Det leder till dåligt processorutnyttjande och dåliga prestanda.

5. Ett kopplat protokoll, såsom TCP, upprättar en (tänkt) fast förbindelse mellan de två kommunicerande noderna över vilken alla meddelanden sedan går. Man behöver alltså inte ange för varje enskilt meddelande vart det ska skickas. Ett okopplat protokoll, såsom UDP, behandlar varje meddelande som helt oboroende av alla andra meddelanden och därför måste destinationsadressen anges för varje meddelande.

6. I java.nio infördes en mekanism som gör det möjligt för en tråd att vänta på att det ska kunna gå att utföra I/O på någon av flera angivna strömmar. Det går till som så att tråden kopplar (registrerar) varje ström som den är intresserad av till en Selector. Den anger också vilken typ av I/O den är intresserad av att kunna utföra (täsning, skrivning, etc). Sedan anropar tråden metoden select i Selector-objektet. Tråden blockeras då ända tills det går att utföra I/O på någon av de registrerade trådarna utan att I/O-anropet blockeras. Tråden får efter select-anropet information om på vilka strömmar det går att utföra I/O och kan beta av dessa i tur och ordning. Därefter kan den utföra ett nytt select-anrop, osv.

7. a) URI = Uniform Resource Identifier, URL = Uniform Resource Locator
   b) En URI är ett generellt sett att namnge en resurs, men den talar inte nödvändigtvis om hur resursen ska kunna nås via nätverket. En URL identifierar däremot explicit hur en resurs ska kunna nås via nätverket – samtidigt som den naturligtvis fungerar som en ren identifikation. En URL kan därför betraktas som ett specialfall av det mer generella begreppet URI.

8. 1) Sant
   2) Falskt
   3) Falskt
   4) Sant

9. 1) Det som skickas över TCP-ström är en sekvens av bytes (som representerar tecken i vårt fall). Problemet är att det finns inget sätt att avgöra var i strömmen det första kommandot slutar och var det andra börjar. Det som skickas över förbindelsen är alltså STARTLEFTöch det är inte väldefinierat vad anropen av read kommer att returnera. Det första anropet av read kan mycket väl returnera STARTLEFT", STA", STARTLE"eller något liknande.
   2) Idé: Vi lägger till ett radslutstecken som signalerar slutet på ett kommando.

```
os.write("START\n".getBytes());
os.write("LEFTn".getBytes());
os.flush();


byte[] buffer = new byte[128];   // Inga meddelanden > 128 bytes
while(true) {
    int len = 0;
    int c = is.read();
    while(c!='\n' && c!=-1) {
        buffer[len++] = c;
        c = is.read();
    }
    String mess = new String(buffer,0,len);
    performCommand(mess);
}
```

10. Båda teknikerna är javabaserade tekniker som används för att skapa dynamiskt webbinnehåll på serversidan. En servlet implementeras som en klass i Java vilken genererar HTML-kod när den anropas. I JSP skjuter man i stället in fragment av javakod i HTML-koden för en sida. En JSP-specifikation kompileras automatiskt om till en servlet av webbservern när den anropas första gången.

11. 1) Falskt
    2) Sant
    3) Falskt
    4) Falskt

Del 2

1. **Buddylist**
   Vi förutsätter nedan att lämpliga paket importerats (inget poängavdrag för saknad import):

```
import java.net.*;
import java.io.*;
```

a)
```
class BuddyRequester extends Thread {

    BuddyWindow bw;
    MulticastSocket ms;

    public BuddyRequester(MulticastSocket ms,BuddyWindow bw) {
        this.ms = ms;
        this.bw = bw;
    }

    public void run() {
        try {
            byte[] message = new byte[1];
            message[0] = 0;
            InetAddress ia = InetAddress.getByName("experiments.mcast.net");
            DatagramPacket dp = new DatagramPacket(message,message.length,ia,56666);
            while (true) {
                bw.clear();
                ms.send(dp);
                try {
                    sleep(60000);
                } catch (InterruptedException e) { }
            }
        } catch(IOException e) {
            System.err.println("IO error: "+e);
            System.exit(1);
        }
    }
}
```

b)
```
class BuddyServer extends Thread {

    String nick;
    MulticastSocket ms;
    DatagramSocket ds;

    public BuddyServer(MulticastSocket ms,DatagramSocket ds,String nick) {
        this.ms = ms;
        this.ds = ds;
        this.nick = nick;
    }
```

```
        public void run() {
            try {
                byte[] message = new byte[65536];
                while (true) {
                    DatagramPacket dp = new DatagramPacket(message,message.length);
                    ms.receive(dp);
                    byte[] m = nick.getBytes();
                    DatagramPacket reply = new DatagramPacket(m,m.length,dp.getAddress(),56667);
                    ds.send(reply);
                }
            } catch(IOException e) {
                System.err.println("IO error: "+e);
                System.exit(1);
            }

        }
    }
```

c)

```
class BuddyListener extends Thread {

        BuddyWindow bw;
        String nick;
        DatagramSocket ds;

        public BuddyListener(DatagramSocket ds,BuddyWindow bw) {
            this.ds = ds;
            this.bw = bw;
        }

        public void run() {
            try {
                byte[] message = new byte[65536];
                while (true) {
                    DatagramPacket dp = new DatagramPacket(message,message.length);
                    ds.receive(dp);
                    if (dp.getLength()>0) {
                        byte[] m = dp.getData();
                        String nick = new String(m);
                        nick = nick+" ("+dp.getAddress().getHostName()+")";
                        bw.addBuddy(nick);
                    }
                }
            } catch(IOException e) {
                System.err.println("IO error: "+e);
                System.exit(1);
            }
        }
    }
```

d)
```
public class BuddyList {
      public static void main(String [] args) {
          if (args.length!=1) {
              System.err.println("Syntax: java BuddyList <NickName>");
              System.exit(1);
          }
          BuddyWindow bw = new BuddyWindow();
          MulticastSocket ms = null;
          DatagramSocket ds = null;
          try {
              ms = new MulticastSocket(56666);
              InetAddress ia = InetAddress.getByName("experiments.mcast.net");
              ms.setTimeToLive(1);
              ms.joinGroup(ia);
              ds = new DatagramSocket(56667);
          } catch(IOException e) {
              System.err.println("IO error: "+e);
              System.exit(1);
          }
          new BuddyRequester(ms,bw).start();
          new BuddyServer(ms,ds,args[0]).start();
          new BuddyListener(ds,bw).start();
      }
  }
```

2. **Filtrering av sensorvärden**

```
public class SensorFilter {

    private double [] value = new double[3];
    private boolean [] filled = new boolean[3];

    public SensorFilter() {
        /* No code required */
    }

    public synchronized void putValue(int id, double val) {
        while (filled[id]) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        filled[id] = true;
        value[id] = val;
        notifyAll();
    }

    public synchronized double getMedian() {
        while (!filled[0] || !filled[1] || !filled[2]) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        for(int i=0;i<3;i++) {
            filled[i] = false;
        }
        Arrays.sort(value):
        notifyAll();
        return value[1];
    }
}
```