Tentamen
Nätverksprogrammering
Lösningsförslag

2013–06–03, 8.00–13.00

Del 1

1. a) Ett applikationsprotokoll ligger på en högre nivå än de s.k. transportprotokoll vi annars behandlat. Ett applikationsprotokoll är speciellt designat för att överföra information som är specifik för en viss applikationen. De storheter som förekommer på applikationsnivån avspeglar sig normalt i protokollets utformning och det utgör alltså inte bara ett sätt att överföra godtyckliga bytes.
   b) Ett exempel är HTTP – Hypertext Transfer Protocol

2. a) Korrekt
   b) Fel
   c) Fel
   d) Korrekt
   e) Fel
   f) Korrekt

3. a) Det första felet yttrar sig genom att det ser ut som om vi bara kan ta emot kortare och kortare meddelanden. Det andra felet yttrar sig som att det ser ut att skrivas ut skräptext i slutet av meddelandena.
   b) Det första felet åtgärdas genom att David återställer längden på datagrampaketet `dp` före varje anrop av `receive()`: `dp.setLength(buf.length);`. Det andra felet rättas genom att strängen skapas med an annan konstruktor: `new String(dp.getData(),0,dp.getLength());`.

4. 1) RTSP
   2) RTP

5. a) Ett registry är ett separat serverprogram som används för att en RMI-klient och en RMI-server ska kunna koppla ihop sig med varandra. Det fungerar som en förteckning över vilka tjänster som olika RMI-servrar erbjuder.
   b) Serialisering innebär att javaobjekt kodas om till en seriell ström av bytes – typiskt i syfte att överföras via ett nätverk. På mottagarsidan gör man sedan den omvända operationen.
   c) Stub/skeleton är de programkomponenter på klient- respektive serversidan som implementerar RMI-anropen. På klienten representerar en s.k. stubb det distribuerade objektet. När metoder anropas på stubben omvandlas dessa till nätverksanrop som tas emot av ett s.k. skeletten på serversidan. Detta senare objekt gör sedan det verkliga metodanropet på det distribuerade objektet.

6. a) 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.

   b) SAX är oftast snabbare än DOM. Å andra sidan är DOM normalt sett kraftfullare då ett komplett parseträd byggs upp på vilket man kan göra mera omfattande analyser eller transformationer.

7. a) Unicode är en familj av kodningsstandarder som bygger på en gemensam teckentabell innehållande en stor mängd med tecken ur olika alfabet. Varje tecken motsvaras av en särskild kod. Unicode säger dock inget om exakt hur ett tecken ska representeras på bit-nivå. För detta behöver man en specifik kodningsstandard, t.ex. UTF-8 som i detalj specificerar hu den teckenkod i Unicode som representerar ett visst teckna ska lagras.

   b) I UTF-8 lagras ett tecken som en följd av en eller flera bytes. De vanligaste tecknen representeras av en enkel byte medan de mera ovanliga representeras av en sekvens av upp till fyra bytes.

8. Proceduren börjar med att den dator som vill hitta sin motpart på nätet skickar ut ett datagram i form av ett multicast på en utvald multicastadress. Den andra datorn förväntas lyssna efter multicast på just denna multicast och till ett utvalt portnummer. För att göra detta i Java behövs en öppen `MulticastSocket` som har anslutit sig till multicastadressen genom ett anrop av `JoinGroup()`. När dator nummer två tar emot multicastpaketet kan den analysera paketet för att ta reda på adressen till den dator som sände paketet. Nu känner dator två till dator ett. För att även dator ett ska få kunskap om sin motpart skickar den andra datorn ett vanligt unicastpaket till en utvald port på den första datorn. När detta paket tas emot avslöjar paketets avsändare var på nätet dator nummer två finns och båda datorerna känner nu till varandra.

Del 2 – Strömmande video

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

public class VideoReceiver {
    private Socket sock;
    private InputStream inp;
    private OutputStream outp;

    /** Create an object which allows the client to connect to a camera server
        and request images at will. The address to the server is given as the
        argument to the constructor. Returns as soon as the object is initialized. */
    public VideoReceiver(String address) throws IOException {
        /* Open a connection to the server. We choose to use port 33333. */
        sock = new Socket(address,33333);
        inp = sock.getInputStream();
        outp = sock.getOutputStream();
    }

    /** Send a request to the server to send the next image over the network.
        Then, the method blocks until the image arrives. The image is returned
        as a byte vector with a length (<30000 bytes) exactly fitting the size
        of the JPEG image. */
    public byte[] receive() throws IOException  {
        /* Request a new image. */
        outp.write('X');
        outp.flush();

        /* Receive image length encoded as two bytes. */
        int l1 = inp.read();
        int l2 = inp.read();
        int length = l1*256+l2;

        /* Create and read a  byte array with the length calculated above. */
        byte[] b = new byte[length];
        int read = 0;
        int result = 0;
        while (read<length && result!=-1) {
            result = inp.read(b,read,length-read);
            if (result!=-1) {
                read = read+result;
            } else {
                /* The problem statement does not explicitly state how to handle */
                /* a failed connection, but we obviously need to signal it to    */
                /* the application somehow. We therefore choose to throw a       */
                /* general IOException.                                          */
                throw new IOException("Connection lost.");
            }
        }
        return b;
    }

    /** Disconnects from the server. */
    public void close() throws IOException {
        sock.close(); /* Implicitly closes inp and outp as well. */
    }
}
```

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

public class VideoTransmitter {
    private byte[] image = null;

    /** Create an object capable of sending images to supervision clients, if
        connected. Throws an IOException if there is problems initializing the
        object due to network problems. Returns as soon as the object is initialized. */
    public VideoTransmitter() throws IOException {
        new ConnectionThread(this).start();
    }

    /** Informs the object that a new image is available for sending to the
        supervision clients if they should so request. */
    public synchronized void send(byte[] image) {
        /* Instead of making the send method synchronized you can enclose the content */
        /* of the method in a synchronized block or call a private method declared
*/
        /* synchronized.
*/
        this.image = image;
        notifyAll();
    }

    /** Return the next image != latest. Wait if necessary. */
    public synchronized byte[] getImage(byte[] latest) {
        while (image==latest) {
            try {
                wait();
            } catch(InterruptedException e) {}
        }
        return image;
    }
}


class ConnectionThread extends Thread{
    private VideoTransmitter vt;
    private ServerSocket serverSock;
    private byte[] latest;

    public ConnectionThread(VideoTransmitter vt) throws IOException {
        this.vt = vt;
        serverSock = new ServerSocket(33333);
        latest = null;
    }

    public void run() {
        while (true) {
            try {
                Socket s = serverSock.accept();
                new ClientThread(s,vt).start();
            } catch(IOException e) {
                /* Really should not happen, but print an error message anyway and */
                /* continue.                                                       */
                e.printStackTrace();
            }
        }
    }
}
```

```
class ClientThread extends Thread{
    private Socket sock;
    private VideoTransmitter vt;

    public ClientThread(Socket sock, VideoTransmitter vt) {
        this.sock = sock;
        this.vt = vt;
    }

    public void run() {
        try {
            byte[] latest = null;
            InputStream inp = sock.getInputStream();
            OutputStream outp = sock.getOutputStream();
            while (true) {
                inp.read(); /* Any byte received is interpreted as an image request. */
                byte[] b = vt.getImage(latest);
                latest = b;

                /* Send the length of the image encoded in two bytes. */
                int length = b.length;
                int l1 = length/256;
                int l2 = length%256;
                outp.write(l1);
                outp.write(l2);

                /* Send the actual image. */
                outp.write(b);
                outp.flush();
            }
        } catch(IOException e) {
            try {
                /* Attempt to close the socket and terminate. */
                sock.close();
            } catch(IOException e2) { }
        }
    }
}
```