Tentamen
Nätverksprogrammering
Lösningsförslag

2007–05–22, 8.00–13.00

---

1. a) Genom att samla ihop större mängder av data innan det skickas vidare kan man avsevärt minska antalet systemanrop/nätverkspaket som krävs för att överföra datan. Detta brukar leda till kraftig ökning av hastigheten.

   b) Antag att vi har en klient och en server som kommunicerar över TCP. Kommunikationen består i att klienten skickar ett kommando till servern och därefter väntar på svar från servern. Om buffring används kan det hända att kommandot som ska skickas till servern lagras i bufferten, men inte skickas iväg eftersom de underliggande kommunikationslagren väntar på mer data från applikationen innan innehållet i bufferten sänds till servern. Applikationen kommer dock inte att generera mer data eftersom den är blockerad i väntan på svar från servern. Lösningen består i att tvinga fram en tömning av bufferten innan man väntar på svaret från servern. I Java kan man göra detta genom att anropa metoden `flush()` på `Socket`-objektet.

2. a) Document Type Definition

   b) En DTD beskriver strukturen på ett XML-dokument. Den innehåller t.ex. en lista över de element som kan förekomma i XML-dokumentet och hur de kan användas tillsammans.

3. Följande påståenden är sanna: b, e, f

4. RMI är i Java en abstraktion över de grundläggande kommunikationsprotokollet som erbjuder en programmeringsmodell där ett program kan, till synes, exekvera metoder på objekt som i verkligheten finns på en annan dator i nätverket. Metodanrop, med tillhörande parametrar och returvärden översätts av ett "stub"-objekt på den lokala datorn till ett anrop över nätverket till en server där ett "skeleton"-objekt utför anropet på det verkliga objektet.

   När en klient startas är dess första åtgärd normalt att skapa det så kallade "stub"-objektet som representerar det verkliga objektet på servern. För att kunna hitta servern och upprätta förbindelse med denna kontaktas först en speciell server, *rmiregistry*, i vilken servern har registrerat sig. Då överförs all information klienten behöver veta om servern. Servern identifieras i *rmiregistry* av ett namn (textsträng). När sedan själva RMI-anropet sker kommer "stub"-objektet att göra en nätverksuppkoppling till servern, omvandla eventuella parametrar till en form som kan skickas över nätverket, så kallad *marshalling*, och sända dessa. På serversidan omvandlas de tillbaka till originalform och utgör parametrar till den verkliga implementationen av RMI-metoden. Returvärdet från metoden sänds tillbaka till klienten på samma vis.

5. Servlets, JSP, CGI, ASP, PHP...

6. En protocol handler är en abstraktion som isolerar vad det innebär att implementera ett visst protokoll. Den är en modulär enhet som enkelt kan bytas ut när man vill använda ett annat protokoll för att överföra data utan att applikationen i övrigt påverkas. Den betraktar hur saker och ting förs över, inte hur det överförda ska tolkas när det väl är överfört. Till det senare använder man så kallade content handlers.

---

7. Till skillnad från unicast, som innebär att ett datagram sänds genom nätverket till en enda mottagare, innebär multicast att ett meddelande sänds ut till alla mottagare som har registrerat sitt intresse för den aktuella typen av paket. Datagram dupliceras endast om de behöver ta olika vägar genom nätverket.

   Multicast förutsätter dels att routrarna mellan sändare och mottagare stöder/accepterar multicast och att det så kallade TTL-värdet (Time To Live) är tillräckligt stort för att datagrammet ska kunna passera alla routrar på vägen till mottagaren.

8. a) Busy-wait

   b) Den konsumerar all tillgänglig CPU-tid vilket ger dåligt processorutnyttjande och i värsta fall svält bland andra trådar.

   c)
```
public class WaitingThread extends Thread {

    private boolean finished = false;

    public synchronized void releaseThread() {
        finished = true;
        notifyAll(); // I detta fall skulle det räckt med ett notify()
    }

    private synchronized void awaitCompletion() {
        while (!finished) {   // I detta fallet skulle en if-sats fungera lika bra.
            try {
                wait();
            } catch(InterruptedException e) {}
        }
    }

    public void run() {
        ...

        /* Wait for computation to finish. */
        awaitCompletion();

        /* The external thread has now finished its computation */
        ...
    }
}
```

9.
```
private boolean getFile(String server, int port, String remoteFileName,
                        FileOutputStream localfile) {
    DatagramSocket socket = null;
    try {
        InetAddress dest = InetAddress.getByName(server);
            socket = new DatagramSocket();
        byte [] sendBuffer = new byte[516];
        DatagramPacket sendPacket = new DatagramPacket(sendBuffer,
                        sendBuffer.length,dest,port);
        byte [] receiveBuffer = new byte[516];
        DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
                        receiveBuffer.length);

        // Construct RRQ
        int pos = 0;
        sendBuffer[pos++] = 0;
        sendBuffer[pos++] = 1;
        for(int i=0;i<remoteFileName.length();i++) {
            sendBuffer[pos++] = (byte) remoteFileName.charAt(i);
        }
        sendBuffer[pos++] = 0;
        sendBuffer[pos++] = 'o';
        sendBuffer[pos++] = 'c';
        sendBuffer[pos++] = 't';
        sendBuffer[pos++] = 'e';
        sendBuffer[pos++] = 't';
        sendBuffer[pos++] = 0;
        sendPacket.setLength(pos);

        // Send RRQ until response or timeout
        socket.setSoTimeout(2000);
        for(int i=0;i<10;i++) {
            socket.send(sendPacket);
            try {
                socket.receive(receivePacket);
                break;
            } catch(SocketTimeoutException e) { }
        }

        // Handle data packages until EOF or error
        socket.setSoTimeout(20000);
        int expected = 1;
        boolean eof=false;
        do {
            if (receivePacket.getLength()<4) return false; // Sanity check
            if (receiveBuffer[0]!=0 || receiveBuffer[1]!=3)
                return false; // Not data block
            int block = b2i(receiveBuffer[2],receiveBuffer[3]);
            if (block>expected) return false; // Sanity check
            if (block==expected) {
                localfile.write(receiveBuffer,4,receivePacket.getLength()-4);
                expected++;
                eof = (receivePacket.getLength()-4<512);
            }
            // Send ack
            sendBuffer[0] = 0;
            sendBuffer[1] = 4;
            sendBuffer[2] = (byte) (block/256);
            sendBuffer[3] = (byte) (block\% 256);
            sendPacket.setLength(4);
            socket.send(sendPacket);
```

```
                // receive next data block if not eof
                if (!eof) {
                    receivePacket.setLength(516);
                    try {
                        socket.receive(receivePacket);
                    } catch(SocketTimeoutException e) {
                        return false;
                    }
                }
            } while (!eof);

        } catch(IOException e) {
            if (socket!=null) {
                socket.close();
            }
            return false;
        }
        if (socket!=null) {
            socket.close();
        }
        return true;
    }
```

10.
```
import java.net.*;
import java.io.*;
import java.util.*;

public class DatingServer {

    public static void main(String[] args) {
        Queue queue = new Queue();

        try {
                // Skapa ServerSocket för att ta emot uppkopplingar på.
                ServerSocket ss = new ServerSocket(15600);

                // Ta emot nya uppkopplingar och para ihop dem när möjligt.
                while (true) {
                Socket s = null;
                try {
                    // Ta emot uppkoppling
                    s = ss.accept();
                    new SexAnalyzer(s,queue).start();
                } catch(IOException e) { }
            }
        } catch(IOException e) {
                System.out.println("Kunde ej starta servern. Port 15600 upptagen?");
        }
    }
}


class Queue {
    private Vector waiting = new Vector(); // Väntande uppkopplingar
    private int waitingSex = 'M'; // Könet på väntande personer i vektorn

    public synchronized void handleClient(Socket s,int sex) {
        // Finns det någon av motsatt kön som väntar?
        if (waiting.size()>0 && waitingSex!=sex) {
            // Ja, para ihop dem! Skapa ett par nya trådar
            // som förmedlar trafiken i vardera riktningen.
            Socket partner = (Socket) waiting.get(0);
            new TransferThread(s,partner).start();
            new TransferThread(partner,s).start();
            waiting.removeElementAt(0);
        } else {
            // Nej, sätt denna uppkoppling på väntning
            waiting.add(s);
            waitingSex = sex;
            // Informera den väntande
            try {
                OutputStream os = s.getOutputStream();
                os.write("Var god vänta på partner...".getBytes());
                os.write('\n');
                os.flush();
            } catch(IOException e) { }
        }
    }
}
```

```
class SexAnalyzer extends Thread {
    private Socket s;
    private Queue queue;

    public SexAnalyzer(Socket client,Queue queue) {
        s = client;
        this.queue = queue;
    }

    public void run() {
        try {
            // Man eller kvinna?
            OutputStream os = s.getOutputStream();
            InputStream is = s.getInputStream();
            do {
                os.write("Är du man eller kvinna (M/K)?".getBytes());
                os.write('\n');
                os.flush();
                int sex = is.read();
                while (sex==13 || sex==10) {
                    sex = is.read();
                }
            } while (sex!='M' && sex!='K');
            queue.handleClient(s,sex);
        } catch(IOException e) {
            try {
                s.close();
            } catch(IOException exc) {}
        }
    }
}

class TransferThread extends Thread {
    private Socket source;
    private Socket target;

    public TransferThread(Socket source,Socket target) {
        this.source = source;
        this.target = target;
    }

    public void run() {
        // Läs tecken från source och skicka dem till target ända tills
        // något går snett. Stäng då båda socketarna så att förbindelsen
        // och båda klienterna avslutas.
        try {
            InputStream is = source.getInputStream();
            OutputStream os = target.getOutputStream();
            // Meddela klienten att man kan chatta
            os.write("Varsågoda att chatta!".getBytes());
            os.write('\n');
            os.flush();
            int ch = ' ';
            while (ch!=-1) {
                ch = is.read();
                if (ch!=-1) {
                    os.write(ch);
                    os.flush();
                }
            }
        } catch(IOException e) {
        } finally {
```

```
                // Försök alltid att stänga socketerna. Ignorera
                // eventuella exceptions.
                try {
                    source.close();
                } catch(IOException e) {}
                try {
                    target.close();
                } catch(IOException e) {}
            }
        }
    }
```