LÖSNINGAR

Tentamensskrivning — Nätverksprogrammering (EDA095) 2004-05-27, kl 8-13

DEL 1 - Frågor av teoretisk, principiell eller utredande karaktär

1. **a)** Buffring ökar effektiviteten i överföringen genom att skicka större paket av data åt gången. Det medför mindre overhead och högre överföringshastigheter.

b) Ett problem som kan uppstå är att data som ska skickas buffras i väntan på mer data och inte skickas iväg direkt. Om vi har en dialog över nätverket mellan två datorer kan detta medföra dödläge genom att den sändande datorn väntar på ett svar som aldrig kommer eftersom den mottagande datorn aldrig får någon begäran om svar. Vi kan tvinga strömmen att skicka iväg allt som finns buffrat genom att anropa metoden `flush()`.

2. a) Remote Method Invocation.

b) 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. RMI är Javas motsvarighet till RPC - Remote Procedure Call.

3. Session Initiation Protocol. SIP används för att upprätta och stänga ner förbindelser över ett nätverk och för att förhandla om eventuella parametrar för överföringen.

4. a) Ett exempel på en fördel med CGI är att det är oberoende av programmeringsspråk.

b) En fördel med servlets är att en servlet inte behöver startas om för varje anrop utan kan snabbt svara på nästa begäran och även att den kan behålla ett tillstånd i minnet mellan varje begäran.

5. En metod är att spara information om den aktuella sessionen som dolda fält i formulär i den HTML-sida som returneras till klienten. När nästa begäran kommer kan fälten läsas av och informationen behandlas av servern. Antingen kan servern välja att lagra all nödvändig information på detta sätt eller också kan den nöja sig med att lagra ett identifikationsnummer eller liknande som den sedan kan matcha mot en informationsdatabas på servern. En annan variant är att använda s.k. cookies på motsvarande sätt. Abstraktionen som representeras av klassen `HttpSession` är också ett alternativ, vilket implementationstekniskt bygger på någon av de förstnämnda teknikerna.

6. 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.

7. a) Ett problem Niklas kan få är att meddelanden försvinner eftersom `set()` skulle kunna anropas två gånger i följd av samma tråd. Ett annat problem som kan uppstå är att den läsande tråden stjäl all tillgänglig CPU-tid (busy-wait) vilket kan göra att andra trådar, t.ex. den som anropar `set()` aldrig får köra.

b)

```
public class Buffer() {
    private String message;

    public synchronized void set(String m) {
        while (m!=null) {
            try {
                wait();
            } catch(InterruptedException e) {}
        }
        message = m;
        notifyAll();
    }

    public synchronized String get() {
        while (m==null) {
            try {
                wait();
            } catch(InterruptedException e) {}
        }
        String m = message;
        message = null;
        notifyAll();
        return m;
    }
}
```

DEL 2 - Praktiska programmeringsuppgifter

1. Synkronisering av klockor

TimeSyncClient.java

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

/**
    A class used to keep the local clock synchronized with the clock of
    a central server.
*/
public class TimeSyncClient extends Thread {
    private InetAddress timeServer;
    private int serverPort;
    private int period;
    private long serialNo;

    /**
        Constructor. The “serverAddr” and “portno” parameters determines the
        location of the corresponding server. The “period” parameter gives
        the interval (in minutes) between two synchronization attempts.
    */
    public TimeSyncClient(String serverAddr,int portno,int period) {
        try {
            timeServer = InetAddress.getByName(serverAddr);
            serverPort = portno;
            this.period = period;
            serialNo = 0;
            start(); /* Activate the thread. */
        } catch(UnknownHostException) {
            /* Error detected. Print an error message, but let the application
            continue without automatic clock synchronization. */
            System.err.println(“TimeSyncClient: “+e);
        }
    }
```

```
    /**
        Main thread method.
    */
    public void run() {
        /* Set our priority to maximum to minimize interference from
           other threads. */
        setPriority(MAX_PRIORITY);
        /* Create a DatagramSocket to be used for communicating with the
           server. */
        DatagramSocket mySocket = null;
        try {
            mySocket = new DatagramSocket();
            mySocket.setSoTimeout(500); /* receive times out after 500 ms. */
        } catch(SocketException e) {
            System.err.println("TimeSyncClient: "+e);
            return; /* Terminate this thread only. */
        }
        /* Request synchronization periodically. */
        byte[] buffer = new byte[100];
        while (true) {
            /* Create a message with a unique serial number and send it. */
            String sNo = Long.toString(++serialNo);
            DatagramPacket dp = new DatagramPacket(sNo.getBytes(),
                                   sNo.getBytes().length,timeServer,serverPort);
            try {
                long t1 = System.currentTimeMillis();
                long t2;
                mySocket.send(dp);
                /* Wait for answer from the server. If there is an exception
                   due to timeout, i/o error, or malformed messages, abort
                   the synchronization attempt and wait the stipulated
                   amount of time before trying again. */
                try {
                    String serial;
                    String answer;
                    int x;
                    /* Receive messages until the message corresponding
                       to our latest request arrives (typically the
                       first one). */
                    do {
                        dp = new DatagramPacket(buffer,buffer.length);
                        mySocket.receive(dp);
                        t2 = System.currentTimeMillis();
                        answer = new String(dp.getData(),0,dp.getLength());
                        x = answer.indexOf(' ');
                        serial = answer.substring(0,x);
                    } while (Long.parseLong(serial)<serialNo);
                    String serverTime = answer.substring(x+1,answer.length());
                    long newTime = Long.parseLong(serverTime)+(t2-t1)/2;
                    System.setCurrentTimeMillis(newTime);
                } catch(NumberFormatException e) {
                } catch(InterruptedIOException e) {
                }
            } catch (IOException e) { }
            /* Wait until next period... */
            try {
                sleep(60*1000*(long)period);
            } catch(InterruptedException e) { }
        }
    }
}
```

TimeSyncServer.java

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

/**
    A class used for keeping the local clocks of a set of client nodes
    synchronized with the clock of this node.
*/
public class TimeSyncServer extends Thread {

    private DatagramSocket mySocket;

    /**
        Constructor. The parameter indicates which port this server should
        listen on.
    */
    public TimeSyncServer(int portno) {
        try {
            /* Create DatagramSocket for receiving requests. */
            mySocket = new DatagramSocket(portno);
            start(); /* Activate the thread. */
        } catch(SocketException e) {
            /* An error occurred! Abort the server thread without terminating
               the main application. */
            System.err.println("TimeSyncServer: "+e);
        }
    }

    public void run() {
        /* Set our priority to maximum to minimize interference from
           other threads. */
        setPriority(MAX_PRIORITY);

        /* Receive requests one by one and return the current time together
           with the client-specified serial number. */
        byte[] b = new byte[100];
        while(true) {
            DatagramPacket dp = new DatagramPacket(b,b.length);
            try {
                mySocket.receive(dp);
                String serial = new String(dp.getData(),0,dp.getLength());
                InetAddress sender = dp.getAddress();
                int senderPort = dp.getPort();
                String timeString = serial+" "+
                                    Long.toString(System.currentTimeMillis());
                byte[] time = timeString.getBytes();
                dp = new DatagramPacket(time,time.length,sender,senderPort);
                mySocket.send(dp);
            } catch(IOException e) {
                /* An error occurred. Ignore the request and wait for the
                   next one. */
            }
        }
    }

}
```

2. Distribuerad high-score-lista via RMI

Server- och klientsida

```
public class HiScores implements serializable{
    public String[] player;
    public int[] score;

    public HiScores() {
        player = new String[10];
        score = new int[10];
    }
}

public interface HiScoreList extends Remote {
    public HiScores getHighScores() throws RemoteException;
    public void sendScore(String name,int score) throws RemoteException;
}
```

Klientsidan

```
public class HiScoreCommunication {
    private HiScoreList list;

    public HiScoreCommunication() {
        try {
            list = (HiScoreList) Naming.lookup("rmi://gote.cs.lth.se/hiscore");
        } catch(Exception e) {
            System.err.println(e);
            System.exit(1);
        }
    }

    public HiScores fetchHiScores() {
        HiScores scores = null;

        try {
            scores = list.getHighScores();
        } catch(Exception e) {
            System.err.println(e);
            System.exit(1);
        }
        return scores;
    }

    public void sendNewScore(String name,int score) {
        try {
            list.sendScore(name,score);
        } catch(Exception e) {
            System.err.println(e);
            System.exit(1);
        }
    }
}
```

Serversidan

```
import java.rmi.*;

public class HiScoreListServer {
    public static void main(String [] args) {
        try {
            HiScoreList hiscore = new HiScoreListImpl();
            Naming.rebind("hiscore", hiscore);
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}


import java.rmi.*;

public class HiScoreListImpl extends java.rmi.server.UnicastRemoteObject
                                                        implements HiScoreList {
    private HiScores hiscores;

    public HiScoreListImpl() throws RemoteException {
        hiscores = new HiScores();
    }

    public void sendScore(String name, int score) throws RemoteException {
        String tmpName;
        int tmpScore;
        for(int i=0;i<hiscores.player.length;i++) {
            if (hiscores.score[i]<score) {
                tmpName = hiscores.player[i];
                tmpScore = hiscores.score[i];
                hiscores.player[i] = name;
                hiscores.score[i] =score;
                name = tmpName;
                score = tmpScore;
            }
        }
    }

    public HiScores getHighScores() throws RemoteException {
        HiScores res = new HiScores();
        for(int i=0;i<hiscores.player.length;i++) {
            res.player[i] = new String(hiscores.player[i]);
            res.score[i] = hiscores.score[i];
        }
        return res;
    }
}
```