LUNDS TEKNISKA HÖGSKOLA Nätverksprogrammering 2005
Datavetenskap

EDA095

Tentamensskrivning — Nätverksprogrammering 2005-05-27, kl. 8-13

LÖSNINGSFÖRSLAG

DEL 1 – Frågor av teoretisk, principiell eller utredande karaktär

1a. URL = Uniform Resource Locator XML = eXtensible Markup Language
1b. Med TCP (eller UDP) kommunicerar klienten och servern med nätverksströmmar, sockets där man identifierar IP-adresser och portar. Man definierar ett protokoll mellan klienter och servrar som bestämmer kommandon, deras parametrar och ordning.
När man använder RMI är nätverkskommunikationen gömd i underliggande lager, stubbar och man använder metodanrop för att interagera med servern på samma sätt som i en klassisk procedurell programmeringsmodell.

2. En servlet är en liten server som är inbyggd i en webbserver. Webbservern svarar en klient-förfrågan (request) genom att ladda (anropa) en servlet. Klienten skickar förfrågan med dess parametrar i ett ”HTML-formulär” med hjälp av HTTP-protokollen (POST eller GET) och servern svarar med att returnera en HTML-sida (response).
En servlet kodas i Java and Tomcat är en webbserver som kan ladda och köra servlets.

3. Man definierar vilka ”service” servern har i form av metoder som samlas i ett interface. Man implementerar metoderna in en klass som körs av servern. En service identifieras genom en URL på formen ”rmi://host/service”.
Klienten lokaliserar metoder med Naming.lookup() och därefter anropar man som för en lokal metod.
Ett gömt lager handhar kommunikationen som består av två delar: ett skelett och en stub. Bara stubbar är nödvändig med moderna versioner av Java. Man producerar stubbar automatiskt med ett rmic-verktyg från interface.
Klienten lokaliserar en service med hjälp av en broker: registry. Först startar man registryn med rmiregistry. Sedan startar man servern och alla service anmäler sig till registrys.

4a. De tre komponenter är: element, attribut och entiteter (också teckendata, text)
4b. Well-formed: motsvarar generell xml-syntax och balanseringen av parenteser och element. Dokument är ”parsable”.
Valid: alla relationer mellan element och attribut motsvaras av definitioner i DTD.
4c. UTF-8, UTF-16, MacRoman, ISO 8859-1 (Latin 1). ISO 8859-15 (Latin 9), etc.

4d. Ett parsingträd.
4e. Man kan traversera, skriva ut eller transformera trädet. Man kan applicera transformationer till XML-element och presentera ett dokument i HTML.

5a. falskt
5b. korrekt
5c. falskt
5d. korrekt

6a. Monitorbegreppet är en konstruktion för att åstadkomma ömsesidig uteslutning till resurser som delas mellan flera trådar. I Java åstadkoms detta med ordet synchronized. Genom att skriva synchronized(object){block} kommer ”block” endast att exekveras av en tråd i taget. Objektet ”object” kommer då att markeras som upptaget och nya trådar som vill exekvera ”block” kommer där att få vänta i kö på sin tur.
6b. Genom att justera prioriteterna för trådarna i ett system kan man i vissa fall förbättra systemets kapacitet (genomströmning). En annan effekt man kan uppnå genom att ändra prioriteter är att undvika svältning av vissa trådar.

7a. Busy-wait. CPU-krävande och kan i värsta fall orsaka starvation.
7b. Flera trådar kan tillåtas anropa put() samtidigt, vilket kan göra data i mailboxen korrupt. Kan åtgärdas genom att göra metoderna get() och put() synkronizerade.
7c. Starvation.
7d.

```
public class Mailbox {
    private String[] box;
    private int msgNbr;

    public Mailbox(int size) {
      box = new String[size];
      msgNbr = 0;
    }

    public synchronized void post(String msg) {
      while (msgNbr == box.length) {
          try {
            wait();
          } catch (InterruptedException e) {}
      }

      box[msgNbr++] = msg;
      notifyAll();
    }

    public synchronized String get() {
      while (msgNbr == 0) {
          try {
            wait();
          } catch (InterruptedException e) {}
      }

      --msgNbr;
      notifyAll();
      return box[msgNbr];
    }
}
```

DEL 2 – Praktiska programmeringsuppgifter

```
package bulletinboard;

import java.net.*;
import java.io.*;
import java.util.*;
public class ServerBB {
      public static void main(String[] args) {
        BulletinBoard bb = new BulletinBoard(10, 5);
        int ident = 0;
        System.out.println(bb.toString());
        Vector clientList = new Vector();
        try {
            ServerSocket ss = new ServerSocket(25001);
            while (true) {
                Socket client;
                ident++;
                client = ss.accept();
           client.getOutputStream().write(bb.toString().getBytes());
                BulletinBoardSession bbs = new
                BulletinBoardSession(ident, client, bb, clientList);
                bbs.start();
            }
        } catch (IOException e) {
            System.err.println("Network failure...");
            e.printStackTrace();
        }
    }

}
```

```
package bulletinboard;

import java.net.*;
import java.io.*;
import java.util.*;
public class BulletinBoardSession extends Thread {
    private int ident;
    private Socket client;
    private BulletinBoard bb;
    private Vector notices;     // the notice list of the client
    private Vector clientList;  // The list of all clients
    private PrintWriter out;    // The output stream to the client

    /** Creates a new instance of BulletinBoardSession */
    public BulletinBoardSession(int ident, Socket client,
                                BulletinBoard bb, Vector clientList) {
        this.ident = ident;
        this.client = client;
        this.bb = bb;
        this.clientList = clientList;
    }

    public void run() {
        BufferedReader in;
        boolean finished = false;
        notices = new Vector();
```

```
        try {
            in = new BufferedReader(
                     new InputStreamReader(client.getInputStream()));
            out = new PrintWriter(client.getOutputStream());
        } catch (IOException e) {
            try {
                client.close();
            } catch (IOException e2) {
                e2.printStackTrace();
            }
            System.err.println("Exception in client socket");
            e.printStackTrace();
            return;
        }

        // adds the client to the list.
        clientList.add(this);

        while (!finished) {
            String command[];
            boolean completed;

            // Three possible commands: post, remove, and quit
            try {
                command = in.readLine().split(" ");
                if (command[0].equals("post")) {
                    completed = doPost(command);
                } else if (command[0].equals("remove")) {
                    completed = doRemove(command);
                } else if (command[0].equals("quit")) {
                    doQuit();
                    finished = true;
                    out.println("Leaving...");
                    out.flush();
                } else {
                    out.println("Protocol error: post x y xw yh |
remove x y xw yh | quit");
                    out.flush();
                }
                broadcastClients();
            } catch (Exception e) {
                doQuit();
                broadcastClients();
                finished = true;
                System.err.println("Error. Leaving...");
                e.printStackTrace();
            }
        }
        try {
            client.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            clientList.remove(this);
        }
        return;
    }
```

```
    // doPost adds a notice physically in the list of a participant
    public boolean doPost(String [] command) {
        Notice b;
        boolean completed = false;

        if (command.length != 5) {
            out.println("Protocol error: post x y xw yh");
            out.flush();
            return false;
        }
        b = new Notice(ident, Integer.parseInt(command[1]),
Integer.parseInt(command[2]), Integer.parseInt(command[3]),
Integer.parseInt(command[4]));
        completed = bb.post(b);
        if (completed) {
            notices.add(b);
        } else {
            out.println("Could not complete operation!");
        }
        out.flush();
        return completed;
    }

    // doRemove removes a notice physically from the list
    public boolean doRemove(String [] command) {
        Notice b, bc;
        boolean completed = false;

        if (command.length != 5) {
            out.println("Protocol error: remove x y xw yh");
            out.flush();
            return false;
        }
        b = new Notice(ident, Integer.parseInt(command[1]),
Integer.parseInt(command[2]), Integer.parseInt(command[3]),
Integer.parseInt(command[4]));
        for (int i = 0; i < notices.size(); i++) {
            System.out.println(notices.elementAt(i).toString());
            if (((Notice) notices.elementAt(i)).equals(b)) {
                notices.remove(i);
                completed = true;
                break;
            }
        }
        if (completed) {
            bb.remove(b);
        } else {
            out.println("Could not complete operation!");
        }
        out.flush();
        return completed;
    }

    // doQuit removes the notices physically from the list
    public void doQuit() {
        Notice notice;
        while (notices.size() != 0) {
            notice = (Notice) notices.firstElement();
            bb.remove(notice);
            notices.remove(0);
        }
    }
```

```
    public PrintWriter getOutSocket() {
        return out;
    }

    // Sends the updated version of the board to all the clients
    public void broadcastClients() {
        PrintWriter outSocket;
        synchronized (clientList) {
            String boardDisplay = bb.toString(); // One time
assignment to enforce causality. Every client will see all the
modifications
            for (int i = 0; i < clientList.size(); i++) {
                outSocket = ( (BulletinBoardSession)
clientList.elementAt(i)).getOutSocket();
                outSocket.println(boardDisplay);
                outSocket.flush();
            }
        }
    }
}


---------------------------------------------------------------------
package bulletinboard;

public class BulletinBoard {
    private int[][] bb;                    //The actual board
    private int boardWidth, boardHeight;  //The board's sizes

    public BulletinBoard(int boardWidth, int boardHeight) {
        ...
    }

    // Removes a notice from the board: sets the positions to 0
    public synchronized boolean remove(Notice b) {
        ...
    }

    // Posts a notice on the board: sets the positions to the ident
value (identifier of the participant)
    public synchronized boolean post(Notice b) {
        ...
    }

    // Checks if the notice fits on the board surface and does not
overlap
    private boolean available(Notice b) {
        ...
    }

    public synchronized String toString() {
        ...
    }
}
```