Tentamen
Nätverksprogrammering
Lösningsförslag

2018–08–23, 8.00–13.00

Del 1

1. URI och URL

   a) En URI är en unik identifierare för en resurs, t.ex. isbn-numret på en bok. En URL beskriver var en resurs finns, t.ex. på viken server och sökväg den kan hämtas.

   (1p)

   b) En URL består av `protokoll://server:port/sökväg?parametrar` Protokoll anger vilket nätverksprotokoll som används vid kommunikationen. Server är ett IP-nummer eller ett DNS-namn som identifierar en server. Sökväg anger var på serven resursen finns. Parametrar innehåller ytterligare data som serven behöver för att generera/hitta resursen.

   (2p)

2. Trådar och blockering

   a) Trådar läggs i tillståndet *waiting* när de anropar `sleep(time)`. De väcks efter en viss tid. Trådar hamnar även i tillståndet när de anropar `wait()` inuti en monitor. Tr[darna väcks då när någon annan tråd anropar notify()/notifyAll() i den monitorn.

   Trådar läggs i tillståndet *blocked* när de tvingas vänta på att komma in i en monitor vid anropar till en monitor-metod, d.v.s. vid anrop till en metod som är `synchronized`. Detta inträffar när monitorn ägs av en annan tråd och de blockerade trådarna väcks när tråden som var i monitorn lämnar den (låser upp monitorlåset).

   (3p)

   b) Ja, det finns två scenarier. Det räcker med att beskriva ett för att få full poäng.

   1) Om en tråd anropar `sleep()` från en monitormetod. Tråden är då i tillsåndet *waiting* samtidigt som den håller monitorlåset. Andra trådar hindras från att komma in i monitorn och systemt fryser. Om tråden istället anropar `wait()` släpps monitorlåset automatiskt när tråden sover.

   2) Nästlade monitoranrop. Vi har två monitorer: MonitorA och MonitorB. Om en tråd anropar en metod i MonitorA från en metod i MonitorB kommer den att hålla låset för MonitorB när den väntar på att få komma in i MonitorA. Detta kan leda till deadlock (om en annan tråd äger MonitorB och behöver MonitorA innan den släpper Bs lås samtidigt som den första tråden äger MonitorA och vill in i MonitorB)

   (1p)

3. XML

   a) *Valid* innebär att dokumentet är korrekt formaterat (en rot, korrekt nästlade taggar). *Verified* innebär även att dokumentet följer gramatiken som beskrivs av en DTD (anger namnen på taggar och hur de får nästlas).

(1p)

b) Ett XML-dokument kan betraktas som ett träd. XPATH anger mönster/sökvägar i det trädet. XPATH används för att navigera och söka i ett XML-dokument. XPATH används även vid transformeringar av XML-dokument (XSLT).

(1p)

4. HTML och CSS

a) inline css `<span style="color: #F00;">spaceman spiff was here</span>`

(1p)

b) `<span class=red">spaceman spiff was here</span>`
In css-file, alternative 1, use classes: `red { color: #F00;}`
In css-file, alternative 2, make every `<span>` block red: `span { color: #F00; }`

(2p)

5. a) *falskt* – UDP garanterar inte att paketen kommer fram, men innehållet i paket som gör det är korrekt.

b) *falskt* – I ett XML-dokument kan endast text lagras. Base64-kodning behövs för binär data.

c) *sannt* – I en monitor garanteras ömsesidig uteslutning.

d) *falskt* – De vanligast förekommande tecknen använder 8 bitar, men andra använder fler.

e) *falskt* – UDP-paket kan skickas till alla portar.

f) *sannt* – En html-fil kan innehålla JavaScript.

(3p)

6. JavaScript

a) Operationen `===` utför ingen typkonvertering, d.v.s. returnerar true on både värde och typ är samma. Operationen `==` utför typkonvertering innan värdet jämförs. `3.14 === '3.14'` är falskt medans `3.14 == '3.14'` är sant.

(1p)

b) Inga exekveringsfel genereras. Icke definierade egenskaper (attribut) evalueras till `undefined`, som är falskt. Objektet som returneras är:

```
{
  name: 'Kalle',
  age: 6,
  grade = 'u'
}
```

(2p)

7. Klassen `Communication` är inte trådsäker. `while(bank.lock) { }` är en *bussy-wait* som kan leda till svält. Det finns även kapplöpning: Om ett trådbyte sker efter att en tråd lämnat `while(bank.lock)`, men innan `bank.lock = true` kan flera trådar samtidigt anropa `setBalance()/getBalance()` på samma konto.

(2p)

Del 2 – Programmeringstävlingen

1. implementering

```
public class ServerMain {
  public final int telnetPort = 23;

  public static void main(String[] args) {
    new ServerMain();
  }

  public ServerMain() {
    // monitors
    ResultMonitor resultMonitor = new ResultMonitor();

    // wait for new connections
    try(ServerSocket serverSocket = new ServerSocket(telnetPort)){
      while(true) {
        Socket socket = serverSocket.accept();
        new ConnectionHandler(resultMonitor, socket).start();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public class ResultMonitor {
    private Results results = new Results();

    synchronized public boolean newResult(int team, int problems, int time) {
      boolean updated = results.newResult(team, problems, time);
      if(updated) {
        notifyAll();
      }
      return updated;
    }

    synchronized public void printResults(OutputStream os) {
      results.printResults(os);
    }

    synchronized public void awaitNewResult() {
      try {
        wait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
```

```
  class ConnectionHandler extends Thread {

    private ResultMonitor mon;
    private Socket socket;

    ConnectionHandler(ResultMonitor mon, Socket socket) throws IOException {
      this.mon = mon;
      this.socket = socket;
    }

    public void run() {
      try {
        BufferedReader is = new BufferedReader(
                                          new InputStreamReader(
                                               socket.getInputStream()));
        String command = is.readLine();
        if(command.startsWith("list")) {
          mon.printResults(socket.getOutputStream());
        } else if (command.startsWith("station")) {
          while(true) {
            mon.awaitNewResult();
            mon.printResults(socket.getOutputStream());
          }
        } else {
          String[] numbers = command.split("\\s");
          mon.newResult(Integer.parseInt(numbers[0]),
                        Integer.parseInt(numbers[1]),
                        Integer.parseInt(numbers[2]));
        }
        socket.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }

}
```