FU Berlin – WS 08/09

Nachklausur Netzprogrammierung: Lösungen

Maximale Punktzahl: 45 Minimale Punktzahl: 22

Name: **Matr.-Nr.** .

- Die Klausur dauert 90 Minuten.
- Bitte schreiben sie auf jedes Blatt Ihre Matrikelnummer.
- Bitte geben Sie die Klausur komplett und geheftet wieder ab.
- Bitte schreiben Sie mit schwarzem oder blauem Stift; Bleistift wird ignoriert.
- Bei normaler, gepflegter Schrift reicht der jeweils freigelassene Platz für die Beantwortung der Frage aus. Schreiben Sie bei Platzmangel auf der Rückseite weiter und markieren Sie dies deutlich.
- Nicht vergessen: keine Antwort ohne Begründung!

Aufgaben:	**Punkte**	erreicht
1. Klient mit Socket	6	.
2. Server mit Sockets	9	.
3. Java RMI	6	.
4. Klienten und Server	4	.
5. PHP	9	.
6. JavaScript	5	.
7. Unsicherheit im Netz	6	.
Summe	45	

Aufgabe 1: Klient mit Socket (6 P.)

Der *Internet-Dienst* `finger` ist mit TCP über den Port 79 erreichbar. Er liest eine vom Klienten übergebene *userid* und antwortet mit einigen Informationen über diesen Benutzer. Schreiben Sie dafür in Java einen einfachen Klienten `Finger`, der den *lokalen* Dienst benutzt und so aufgerufen wird:

`java Finger` *userid*

Es genügt, wenn Ihr Klient nur die *erste Zeile* der Benutzerinformationen ausgibt.

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

public class Finger {
public static void main (String arg[]) throws IOException {
   Socket socket = new Socket("localhost", 79);
   PrintStream out = new PrintStream(socket.getOutputStream());
   out.println(arg[0]);
   BufferedReader in = new BufferedReader(
         new InputStreamReader(socket.getInputStream()));
   String info = in.readLine();
   System.out.println(info);
   in.close();
   out.close();
   socket.close();
   }
}
```

Aufgabe 2: Server mit Sockets (9 P.)

Internet-Dienste sind häufig durch *nichtsequentielle Server* realisiert, die mehrere Verbindungen nebenläufig verwalten, wobei für jede Verbindung ein eigener Thread eingesetzt wird. Wir unterstellen einen einfachen Frage/Antwort-Dienst mit Port 12345, der die Antwort auf eine Anfrage `query` mittels einer Methode `String answer(String query)` ermittelt. Programmieren Sie einen solchen nichtsequentiellen Server in Java!

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

public class A2 {
public static void main (String arg[]) throws IOException {
       ServerSocket socket = new ServerSocket(12345);
       for (;;) { Socket s = socket.accept();
                  new Service(s).start();   }
       }

   static class Service extends Thread {
          Service(Socket s) { socket = s; }
          Socket socket;
          String answer(String query) { return query+query; }
   public void run() {
          try { BufferedReader in = new BufferedReader(
                                    new InputStreamReader(
                                    socket.getInputStream()));
                String query = in.readLine();
                PrintStream out = new PrintStream(
                                      socket.getOutputStream());
                out.println(answer(query));
                in.close(); out.close();  }
          catch(IOException e) { }
          }
   }
}
```

Aufgabe 3: Java RMI (6 P.)

`x.m(y)` sei ein Fernaufruf für ein Objekt `x` mit Schnittstelle

```
interface I extends Remote {
void m(Arg a) throws RemoteException;
}
```

Wir setzen voraus, dass `y` den Typ `Arg` oder einen Untertyp von `Arg` hat (damit der Code statisch typkorrekt ist). In Abhängigkeit vom Typ `Arg` und vom tatsächlichen Klassentyp des Objekts, auf das `y` verweist, kann `x.m(y)` durchaus unterschiedliche Effekte haben. *Welche 3 Fälle* sind hier möglich, und unter welchen Bedingungen treten sie jeweils ein?

---

1. `m` erhält einen Fernverweis auf das Parameter-Objekt. Dies geschieht, wenn die Klasse des Parameter-Objekts `Remote` implementiert.

2. `m` erhält einen Verweis auf eine lokale Kopie des Parameter-Objekts. Dies geschieht, wenn die Klasse des Parameter-Objekts nicht `Remote`, aber `Serializable` implementiert.

3. `x.m(y)` führt zu einem Laufzeitfehler `MarshalException`, wenn das Parameter-Objekt weder `Remote` noch `Serializable` implementiert.

Aufgabe 4: Klienten und Server (4 P.)

Bei der Netzprogrammierung begegnen wir immer wieder dem *Dienst*-Begriff und damit *Frage/Antwort-Beziehungen* zwischen *Klienten* und *Servern* in verschiedenen Kontexten: 1. TCP Sockets, 2. RMI, 3. CORBA, 4. Web-Dienste. Nennen und erläutern Sie jeweils mindestens einen *konzeptionell bedeutenden* Unterschied zwischen 1 und 2, 2 und 3, 3 und 4!

---

1 versus 2: RMI erlaubt *Fernaufrufe* und ist damit ein programmiersprachlicher Ansatz, der von den vom Betriebssystem an der Systemschnittstelle zur Verfügung gestellten, *nachrichtenorientierten* Sockets abstrahiert.

2 versus 3: Die CORBA IDL ist sprachunabhängig und ermöglicht damit – im Gegensatz zu RMI – Fernaufrufe zwischen unterschiedlichen Programmiersprachen.

3 versus 4: Für die Schnittstellen- und Datenbeschreibung von Web-Diensten wird – anders als bei CORBA – XML eingesetzt, zusammen mit dem HTTP-Protokoll.

Aufgabe 5: PHP (9 P.)

Betrachten Sie diesen Rumpf eines HTML-Dokuments mit PHP:

```
<body>
<?
    function f($x) { return $_GET[$x] + 0; }
    if ($z = ($_GET['y']=='y')) echo '<h2>';
?>
Welcome at BigFive!
<!-- date function delivers date, e.g., 300309 -->
<?= $z ? "</h2><br>".date(dmy) : "" ?>
<br><br>
<?
    $x = f('x');
    for($i=0; $i<$x; $i++) echo '<br>';
?>
What is <?= $_GET['y']."?\n" ?>
</body>
```

Welchen HTML-Text liefert der Server jeweils aus, wenn die folgenden beiden *query strings* zugrundegelegt werden: 1. `"x=5&y=y"` 2. `"x=0&y=z"` ?

1.

```
<body>
<h2>Welcome at BigFive!
<!-- date function delivers date, e.g., 300309 -->
</h2><br>300309<br><br>
<br><br><br><br><br>What is y?
</body>
```

2.

```
<body>
Welcome at BigFive!
<!-- date function delivers date, e.g., 300309 -->
<br><br>
What is z?
</body>
```

Aufgabe 6: JavaScript (5 P.)

Analysieren Sie die folgende *Webseite* und beantworten Sie diese Fragen:

1. Wie ist das *Erscheinungsbild* der Webseite? Zeichnen Sie eine Skizze!
2. Welche *actions* kann der Benutzer auslösen, und was passiert dann?
3. Auf welches *DOM-Objekt* bezieht sich in `<input ...>` das jeweilige `this`?
4. Welche *Eigenschaften* hat dieses Objekt (jeweils Name und Wert)?
5. Kann man `pressed` überall durch `blablabla` ersetzen? Begründung!

```
<HTML><BODY>
<SCRIPT language="JavaScript">
function onSubmit() {
     if(document.pressed == 'Insert')
          document.myForm.action = "insert.html";
     else if(document.pressed == 'Update')
          document.myForm.action = "update.html";
     return true;  }
</SCRIPT>
<FORM name="myForm" onSubmit="return onSubmit();">
     <INPUT type="submit" name="Operation" value="Insert"
          onClick="document.pressed=this.value" >
     <INPUT type="submit" name="Operation" value="Update"
          onClick="document.pressed=this.value" >
</FORM>
</BODY></HTML>
```

---

1. 

2. Der Benutzer kann eine der beiden Tasten *Insert, Update* drücken. Der Effekt ist, dass die Seite `insert.html` bzw. `update.html` angezeigt wird.
3. Das `this` bezieht sich auf das dem Element `<INPUT ...>` zugeordnete Objekt.
4. Das Objekt hat die bei `<INPUT ...>` angegebenen Eigenschaften, also `type="submit", name="Operation", value="Insert", onClick="document.pressed=this.value"` und entsprechend für das andere Objekt.
5. Ja, man kann `pressed` überall durch `blablabla` ersetzen: beliebig benannte Objekteigenschaften können dynamisch hinzugefügt werden; anstelle des ad-hoc gewählten Namens `pressed` hätte auch ein beliebiger anderer Name gewählt werden können.

Aufgabe 7: Unsicherheit im Netz (6 P.)

Sobald man Netzprogrammierung betreibt, sollte man sich der vielfältigen Gefahren bewusst sein, die im Netz drohen, z.B. *Trojanische Pferde* oder die Ausnutzung von Schwachstellen wie *Pufferüberlauf*. Erläutern Sie diese beiden Begriffe und skizzieren Sie typische Möglichkeiten,

1. wie man als *Benutzer* gegen Trojanischen Java-Code kämpfen kann,
2. wie man als *Netzprogrammierer* das Risiko eines Pufferüberlaufs minimiert!

---

1. (Trojanische Pferde)
Trojanische Pferde drohen, wenn man fremden Code aus unzuverlässiger Quelle auf dem eigenen Rechner ausführt: Code mit augenscheinlich korrekter Funktionsweise kann insgeheim auch Schaden anrichten, z.B. Benutzerdaten ausspionieren. Wenn es sich um Java-Code handelt, kann man die Ausführung von einem *Security Manager* überwachen lassen; in einer Richtliniendatei kann der Benutzer festlegen, welche Aktionen dem Code erlaubt werden sollen und/oder von wem der Code kryptographisch signiert sein muss.

2. (Pufferüberlauf)
Pufferüberläufe resultieren aus Programmierfehlern: wenn ein zu schreibender Wert nicht in die Zielvariable passt, werden die Werte benachbarter Variablen in Mitleidenschaft gezogen. In einer sicheren Programmiersprache (wie z.B. Java) ist derartiges von vornherein ausgeschlossen. In unsicheren Sprachen (wie z.B. C) sollte man die „besonders unsicheren" Bibliotheksfunktionen vermeiden – und natürlich besonders sorgfältig programmieren. Auch Werkzeuge für die statische Codeanalyse können bei der Entdeckung möglicher Schwachstellen helfen.

ENDE der Klausur