Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Informatik und Prakt. Mathematik
Frank Huch

24118 Kiel
Christian-Albrechts-Platz 4
Tel.: 0431 / 880-7277

11. Übung „Nebenläufige und verteilte Programmierung“

Abgabe in der Vorlesung oder bis zum 4. Februar in Raum 704

Aufgabe 27 **6 Punkte**

Machen Sie den Chat aus Aufgabe 23 robust. Insbesondere soll der Chat-Server terminierte Chat-Room-Server neu starten, ohne daß die Chat-Klienten dies bemerken. Das ist natürlich nur solange möglich, bis keine Knoten mehr für den Start von Chat-Room-Servern zur Verfügung stehen.

Aufgabe 28 **6 Punkte**

Bei der Programmierung verteilter Spiele ist die initiale Phase des Zusammenkommens der Spielpartner ein häufig wiederkehrendes Problem. Definieren Sie ein Erlang-Programm, welches das Zusammentreffen mehrerer Spieler ermöglicht. Ein Spieler soll ein Spiel starten können, welchem sich andere Spieler anschließen können. Das Hinzukommen neuer Spieler soll jedem Spieler angezeigt werden. Neu hinzugekommene Spieler sollen ebenfalls alle bereits vorhandenen Mitspieler angezeigt bekommen. Das Spiel soll starten, wenn ein beliebiger Spieler den Start des Spiels frei gibt. Zu diesem Zeitpunkt soll allen Spielern eine Liste der Pids aller Mitspieler und die ausgezeichnete Pid des initiierenden Spielers zur Verfügung stehen.
Erweitern Sie Ihr Erlang-Programm zu einem Modul, welches von beliebigen Spielen verwendet werden kann. Wie können Sie elegant das konkrete Spiel in Ihr Modul einbinden?
Wer Lust hat kann dieses Modul auch noch robust gestalten, so daß zwischendurch Spieler auch wieder terminieren/abstürzen können.

Aufgabe 29 **8 Punkte**

Implementieren Sie das Linda-Modell in Erlang. Implementieren Sie hierzu ein Modul `tupelServer`, welches die Möglichkeit bietet einen Tupelraum zu starten. Weiterhin soll das Modul die zugehörigen Linda-Funktionen (`in/2`, `out/2` und `rd/2` zur Verfügung stellen. Das zusätzliche Argument dieser Funktionen soll der Tupelraum (genauer seine Pid) sein.
Erlang verfügt zwar über Pattern-Matching, allerdings sind Pattern keine Werte, so daß man sie nicht an den Tupelserver übergeben kann. Stattdessen kann man aber partielle Funktionen übergeben, welche das Pattern-Matching (aber möglicherweise auch kompliziertere Tests) übernehmen, wie beispielsweise:

```
f(E) -> {hello,_,_}=E.

g({hello,X,Y}) -> {X,Y}.
```

```
h({hello,X,Y}) -> case X>Y of
                     true -> {X,Y};
                     false -> throw(noMatch)
                  end.
```

Bei der Funktion `f` wird im Matching-Fall das gesamte Tupel zurück geliefert, während bei `g` nur ein Tupel der Gebundenen Variablen nur ein Tupel der gematchten Variablen zurück gegeben wird. Die Funktion `h` implementiert einen komplizierteren Test, welcher im Linda-Modell nicht ist.
Ein Ansatz zur Programmierung des Linda-Servers ist es zwei Listen im Linda-Server zu verwalten. Die eine Liste enthält die Klienten-Anfragen, welche zu keinem Eintrag im Tupelraum gepaßt haben. Die andere Liste enthält alle Einträge des Tupelraums, welche noch nicht durch einen Klienten mittels `in` ausgelesen wurden. Überlegen Sie genau, wann Sie welche Liste nach passenden Einträgen durchsuchen müssen.
Wie können Sie sowohl für das Schreiben, als auch das Lesen das Tupelraums zusätzlich Timeouts implementieren? Überlegen Sie insbesondere, wie Sie zusätzlichen Rechenaufwand im Tupelserver verhindern können.