PHILIPPS-UNIVERSITÄT MARBURG D-35032 Marburg
Fachbereich Mathematik und Informatik Hans Meerwein Straße
Prof. Dr. R. Loogen
J.Berthold

06.Juli 2006

11. Übung zu "Parallelität in funktionalen Sprachen", SS 2006

Abgabe schriftlicher Aufgaben: Do, 13.Juli 2006 (vor der Vorlesung)

Hinweise:
Dies ist das letzte Übungsblatt zur Vorlesung.

Aufgaben

11.1 **Dining Philosophers in Haskell** 5 Punkte

Schreiben Sie ein Concurrent-Haskell-Programm zur Simulation des Synchronisationsproblems "Speisende Philosophen" mit Threads:

> An einem runden Tisch sitzen n Philosophen, welche abwechselnd denken und Spaghetti essen. Zwischen je zwei Philosophen liegt eine Gabel. Vor dem Essen nimmt ein Philosoph die Gabeln links und rechts von ihm auf (die er sich mit dem jeweiligen Nachbarn teilt). Nach dem Essen legt er die Gabeln wieder ab und denkt eine zufällig gewählte Zeit lang nach.

Verwenden Sie eine Synchronisation über Semaphore vom Typ `QSem`. Welche unterschiedlichen Möglichkeiten bestehen zur Synchronisation?

Verwenden Sie `threadDelay` sowie `System.Random`, um zufällige Wartezeiten für die Threads zu realisieren.

11.2 **Doctor's Office** 7 Punkte

Simulieren Sie das folgende Synchronisationsproblem in Concurrent Haskell:

Given a set of patients, a set of doctors, and a receptionist, model the following interactions: initially patients are all well, and all doctors are in a FIFO queue awaiting sick patients. At random times, patients become sick and enter a FIFO queue for tratement by one of the doctors. The receptionist handles the two queues, assigning patients to doctors in a first-in-first-out manner. Once a doctor and patient are paired, the doctor diagnoses the illness and cures the patient in a random amount of time. The patient is then released, and the doctor rejoins the doctor's queue to await another patient. The output of the problem is intentially left unspecified.

This is neither an event-driven nor a time-driven simulation. There is no global knowledge, no knowledge of when events will occur, no global clock, and no global communications. You may use any method you wish to decide when a patient becomes sick and how long a patient sees a doctor. The interactions of the patients, doctors, and receptionist should be true to life. Our intent is to test each language's ability to program a set of concurrent, asynchronous processes with circular dependencies.[1]

[1] aus: `http://www.cs.clemson.edu/~steve/Parlib/salishan/problems`