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

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

8. Übung „Nebenläufige und verteilte Programmierung“

Abgabe in der Vorlesung oder bis zum 14. Januar in Raum 704

Aufgabe 19 **6 Punkte**

In der Übung hatten wir begonnen Suchbäume (ohne Höhenbalancierung) in Erlang zu implementieren (siehe `test.erl` auf der Web-Seite zur Vorlesung). Erweitern Sie die Implementierung wie folgt:

- Die Einträge im Suchbaum sollen nicht nur einzelne Werte wie Zahlen sein, sondern ein Paar aus Schlüssel und Wert. Die Sortierung soll über den Schlüssel erfolgen. Passen Sie die definierten Funktionen `insertTree`, `treeToList` und `listToTree` entsprechend an.
- Definieren Sie eine Funktion `lookupTree`, welche zu einem Schlüssel (unter Berücksichtigung der Sortierung) den zugehörigen Wert nachschlägt. Beachten Sie analog zur Funktion `lookup` auf Listen, daß der Schlüssel ggf. auch nicht im Baum vorkommen kann.
- Definieren Sie eine Funktion `deleteTree`, welche das erste Vorkommen eines Schlüssels im Baum löscht. Die Sortierung soll natürlich erhalten bleiben. Nach Möglichkeit soll auch die Struktur des Suchbaums erhalten bleiben.

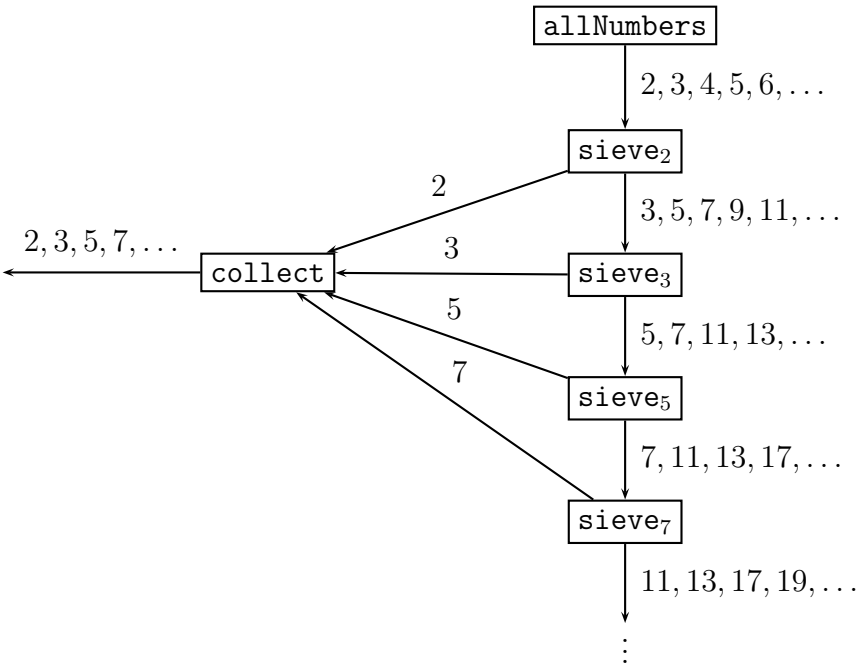
Aufgabe 20 **4 Punkte**

Erweitern Sie die dinierenden Philosophen aus der Vorlesung um die Vermeidung von Deadlocks mittels zurücklegen. Das Programm `diningPhils.erl` finden sie auf der Web-Seite zur Vorlesung.

Aufgabe 21 **8 Punkte**

In dieser Aufgabe sollen Sie mit Hilfe nebenläufiger Erlang Prozesse das Sieb des Eratosthenes implementieren. Definieren Sie ein Modul `primes`, welches einen Prozeß zur Verfügung stellt, der auf Anfrage die Primzahlen der Reihe nach ausgibt.
Definieren Sie zunächst einen Prozeß `allNumbers`, welcher auf Anfrage durch einen anderen Prozeß alle Zahlen (ab zwei) ausgibt. Dann können Sie einen Prozeß `sieve` definieren, welcher von einem anderen Prozeß (initial der Prozeß `allNumbers`) jeweils weitere Zahlen erfragt. Die erste solche Zahl ist immer eine Primzahl. Aus allen weiteren Zahlen soll dieser Prozeß die Vielfachen dieser Primzahl herausfiltern und diese auf Anfrage weiterleiten. Verwenden Sie zum Prüfen ob eine Zahl Vielfaches einer Primzahl ist die Erlang-Infix-Operation `rem`, welche den Divisionsrest liefert.

Folgendes Bild soll das Vorgehen verdeutlichen:



Bei der Implementierung müssen Sie darauf achten, daß ein `sieve` Prozeß einen neuen `sieve` Prozeß erst dann erzeugt, wenn er vom `collect` Prozeß nach seiner ersten Primzahl gefragt wurde. Sonst erzeugen Sie direkt unendliche viele `sieve` Prozesse, was natürlich zu einem Speicherüberlauf führt.
Implementieren Sie auch eine Funktion, welche die Primzahlberechnung startet und die Primzahlen der Reihe nach abfragt und ausgibt.
Wieviele Erlang Prozesse verwendet Ihr System, zum Zeitpunkt bei dem die Primzahl 7727 ausgegeben wird? Wie oft wurde die Nachricht 7727 von einem zu einem anderen Prozeß verschickt? Wird sie bei der Ausgabe weiterer Primzahlen noch einmal verschickt?