Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Institut für Informatik
Michael Hanus, Frank Huch

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

7. Übung „Nebenläufige und verteilte Programmierung"

Abgabe am 19. Dezember, Besprechung am 20. Dezember

Aufgabe 1

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 2

Implementieren Sie den Zähler aus der 3. Übung, Aufgabe 2 in Erlang. Die Zählgeschwindigkeiten sollen der Startfunktion als Liste übergeben werden. Eine einfache Zähler-GUI finden Sie auf der Web-Seite zur Vorlesung. Neue Zählerfenster können mit der Funktion

`counterGui:start(`v`,`$clientPid$`)`

geöffnet werden. Hierbei wird v in der GUI angezeigt und die relevanten Events der GUI werden als folgende Nachrichten an den beim Start angegebenen *ClientPid* gesendet:

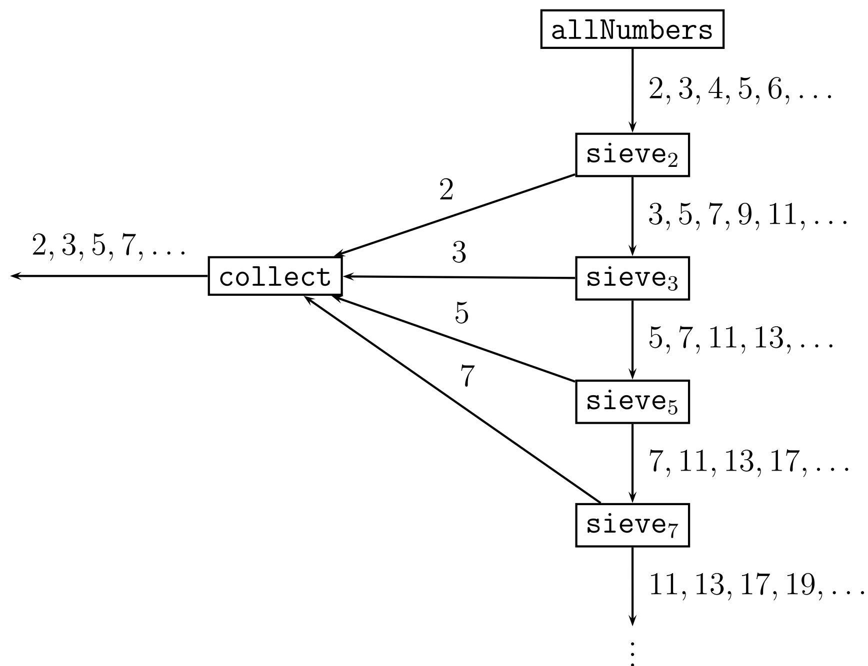
`start`, `stop`, `copy`, `clone` and `close`

Die Funktion `counterGui:start` gibt den Pid des gestarteten Kontrollprozesses zurück. Den Wert des Fensters können Sie verändern, indem Sie an diesen Pid die Nachricht `{setValue,`v`}` schicken, wobei v der anzuzeigende Wert ist. Der Klienten-Pid der Zähler-GUI kann mit der Nachricht `{newClient,`*pid*`}` verändert werden.

Aufgabe 3

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?