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

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

3. Übung „Nebenläufige und verteilte Programmierung“

Abgabe am 13. November in der Vorlesung

Aufgabe 7 **3+2 Punkte**

a) Definieren Sie eine Klasse `Semaphore`, welche einen abstrakten Datentyp für Semaphore mit den Operationen P und V (und auch L für lookup) zur Verfügung stellt.

b) Verwenden Sie die Klasse `Semaphore` zur Implementierung der dinierenden Philosophen mit Deadlockvermeidung durch Zurücklegen.

Aufgabe 8 **5+2+3 Punkte**

a) In der Vorlesung haben wir das Kommunikationskonzept von Java kennen gelernt. Ein alternatives Konzept ist Message Passing. Ein erster Schritt in diese Richtung sind veränderbare Variablen (mutable variables, oder kurz *MVars*). MVars sind vom Prinzip einelementige Puffer. Threads können Werte in MVars schreiben und Werte aus MVars auslesen. Versucht ein Thread aus einer leeren MVar zu lesen, supendiert er. Wird die MVar durch einen anderen Thread gefüllt, so wird einer der suspendierter Leser wieder erweckt und liest die MVar aus. Analoges gilt für das Schreiben in eine gefüllte MVar.

   Definieren Sie eine Klasse `MVar` mit Konstruktoren zur Erzeugung leerer und gefüllten MVars, welche folgende Methoden zur Verfügung stellt:

   - `take` liest eine gefüllte MVar aus; suspendiert, falls die MVar leer ist.
   - `put` füllt eine MVar mit einem Objekt; suspendiert, falls die MVar gefüllt ist.
   - `read` liefert den Wert einer gefüllten MVar, leert die MVar allerdings nicht; suspendiert, falls die MVar leer ist.
   - `swap` ersetzt den Wert einer MVar durch einen anderen Wert; suspendiert, falls die MVar leer ist.
   - `isEmpty` testet, ob die MVar leer ist; suspendiert nie.
   - `tryPut` füllt eine MVar, falls diese leer ist und liefert den booleschen Wert `true`; falls die MVar gefüllt ist liefert die Methode `false`; suspendiert nie.
   - `tryRead` analog zu `tryPut`; suspendiert nie.

Begründen Sie, warum Sie in Ihrer Implementierung `notify()` bzw. `notifyAll()` verwenden.
Optimieren Sie Ihre Implementierung so, daß Sie nur noch `notify()` verwenden.

b) Implementieren Sie das Erzeuger-Verbraucher-Problem unter Verwendung einer MVar als einelementigen Puffer.

Ein Nachteil dieser Implementierung ist, daß der Puffer maximal ein Produkt aufnehmen kann. Erzeuger müssen also mit der weiteren Produktion warten, bis ein Verbraucher den Puffer geleert hat. Optimieren Sie Ihre Implementierung so, daß Sie für jedes erzeugte Produkt einen weiteren Thread zur Hilfe nehmen, welcher die Pufferung des Produkts vornimmt. So kann der Erzeuger sofort fortfahren, ohne auf den Verbraucher zu warten. Mit dieser Technik können Sie einen unbeschränkten Puffer mit Hilfe von Threads implementieren.

c) Erweitern Sie Ihre Implementierung um einen Zeitstemplel (eine laufende Nummer) der Produkte. Verwenden Sie den Zeitstempel, um zu garantieren, daß die Produkte in der Reihenfolge ihrer Produktion verbraucht werden.

Aufgabe 9 **5 Punkte**

Erweitern Sie den graphischen Zähler aus Aufgabe 4 um folgende Buttons:

- Stop: der Zähler bleibt beim aktuellen Wert stehen.
- Start: der Zähler zählt mit seiner vorgegebenen Geschwindigkeit weiter.
- Close: die Darstellung des Zähler wird geschlossen.

Achten Sie auch darauf, daß Ihr Programm terminiert, falls kein Zähler mehr dargestellt wird.