ALP4 SoSe 2013, Di. 16-18

Lösung Übungsblatt 4

Christoph van Heteren-Frese (Matr.-Nr.: 4465677),
Sven Wildermann (Matr.-Nr.: 4567553)
Tutor: Alexander Steen, eingereicht am 9. Mai 2013

Aufgabe 1

Aufgabe 2

Annahme: Jeder der Philosophen hat nur endlich lange Hunger. Grundlage der Lösung ist die auf Seite 89 im Buch dargestellte Struktur für universelle kritische Abschnitte. Jeder der 5 Philosophen wird durch eine Prozessklasse p_i in Form eines unsigned int repräsentiert (im Uhrzeigersinn von 1 bis 5 durchnumeriert). Der Konstruktor der Struktur **type imp struct** erhält somit für die Anzahl der Prozessklassen den Parameter $nK = 5$. Weiterhin wird ein Array der Größe 5 von booleschen Werten $nP[5]$ benötigt, wobei $nP[k]$ angibt ob der Philosoph der zur Prozessklasse k gehört gerade isst oder nicht. Das Spektrum der Eintrittsbedinungen **c(k uint)** gibt für den Philosophen der Prozessklasse k genau dann true zurück, wenn die Philosophen rechts und links von ihm (also Prozessklasse $k+1 \mod 5$ und $k-1 \mod 5$) sich nicht im kritischen k A befindet (also nicht essen). **in(x Any, k uint)** setzt nP[k] = true und **out(x Any, k uint)** setzt nP[k] = false. Die Funktionen **Enter()**, **Leave()**, **vall()** und **Blocked()** bleiben wie im Buch Seite 90 implementiert.

```
const (p_i = i ; nK = 5)
var (nP[5] bool; x *Imp)

func c(k uint) bool{
        return nP[k+1 mod 5] == false && nP[k-1 mod 5]
}

func in(x Any, k uint){
        nP[k] = true
}

func out(x Any, k uint){
        nP[k] = false
}

func (x *ImpCS) PhilosophEat() { x.Enter(p_i, nil) }

func main() { x = New(nK, c, in, out) }
```

Aufgabe 3

Für jeden Philosophen wird ein 2-faches Semaphor verwendet, wobei der erste Wert des Tupels für seine linke Gabel und der zweite Wert für seine rechte Gabel steht. Da sich je 2 Philosophen eine Gabel teilen, teilen sie sich auch die entsprechenden Variablen innerhalb der Semaphore. Nach Definition wird der kritische Abschnitt von einem Philosophen betreten, sobald sowohl seine linke als auch seine rechte Gabel verfügbar ist (und er eintrittswillig ist). Daher gilt: $S.P :< await : linke{-}Gabel{-}und{-}rechte{-}Gabel{-}frei >$ und $S.V :< linkeGabel.freigeben; rechteGabel.freigeben >$

Aufgabe 4

a)

Das in [**?**] erläuterte Prozessmodel unterscheidet fünf Zustände: *nicht existent*, *bereit*, *aktiv*, *blockiert* und *beendet*. Eine Semaphore kann in Java wie folgt implementiert werden:

```
1  public class Semaphore {
2    private int zaehler = 0;
3      private Thread warteschlange = null;
4      public void init(int i)
5      {
6          zaehler = i;
7          warteschlange = null;
8      }
9      public void P(Thread thread)
10     {
11         zaehler--;
12         if(zaehler<0)
13         {
14             warteschlange = thread;
15             try
16             {
17               warteschlange.suspend();
18             }
19             catch(Exception e)
20             {
21             }
22         }
23     }
24     public void V()
25     {
26         zaehler++;
27 //        if(warteschlange!=null)
28         if(zaehler<=0)
29             warteschlange.resume();
30     }
31 }
```

Listing 1: Implemntierung eines Semaphor in Java nach [**?**]. (Die Methoden `suspend()` und `resume()` sind veraltet und sollten in der Praxis nicht mehr verwendet werden.)

b)

c)

Literatur

[] Christian Maurer. *Nichtsequentille Programmierung mit Go 1 Kompakt*. Springer Vieweg, 2012. ISBN 978-3642299681.

[] Guido Kramann. Implementierung und Anwendung von Semaphoren. URL `http://www.kramann.info/67\_Echtzeitsysteme/05\_Semaphor/03\_Semaphor/`.