Technische Universität München
Fakultät für Informatik
Prof. Dr. M. Broy

WS 2002/2003
Übungsblatt 7
6. Dezember 2002

Übungen zur Vorlesung Einführung in die Informatik III

Aufgabe 25 Monitor

Folgender Monitor sei gegeben:

```
monitor X = /* Counter */
⌈   var int n=0;
    signal si,sd,sp;
    var bool i,d,p:=false,false,false;
    proc next_inc =(): ⌈i:=true;si.signal⌋
    proc next_dec =(): ⌈d:=true;sd.signal⌋
    proc increment =() :
    ⌈if ¬i then si.wait fi;
      i:=false; n:=n+1; p:=true; sp.signal⌋
    proc decrement =() :
    ⌈if ¬d then sd.wait fi;
      d:=false; n:=n-1; p:=true; sp.signal⌋
    proc wait_produced =() :
    ⌈if ¬p then sp.wait fi;
      p:=false⌋
    proc get =(var int v ): ⌈v:=n⌋    ⌋
```

a) Nennen Sie Anweisungen, die „n" enthalten und sich gegenseitig ausschließen.

b) Zeigen Sie, wie Boolesche Semaphore zum gegenseitigen Ausschluss und zur Benachrichtigung eingesetzt werden können. Stellen Sie einen Bezug zu den Eigenschaften und Elementen des Monitorkonzeptes her.

c) Ergänzen Sie folgendes Programm an den mit ... gekennzeichneten Stellen, um einen Monitor, mehrere Schleifen und Anweisungen so, dass ein Programm entsteht, dessen Verhalten dem Petri-Netz aus Aufgabe 20 entspricht. Beachten Sie die Signaldefinition (Vgl. Aufgabe 27a) und die daraus resultierende notwendige „Startbelegung":

```
⌈ monitor⌈ X =/* Counter, wie oben */⌋
    ...
    var int x,y,z := 0,0,0;
```

```
    ⎡ while true do X.increment() od
    ‖ ...
    ⎣‖
⎦
```

Aufgabe 26 Monitore in Java

Die Lösung aus Aufgabe 25 ist in Java zu realisieren:

a) Realisieren Sie eine Klasse `Counter`, so dass ein Objekt x der Klasse `Counter` den Monitor „X" aus Aufgabe 25a) realisiert.

- Verwenden Sie die in Java übliche Schreibweise: Aus „next_inc" wird z.B. `nextInc`.
- Verwenden Sie zusätzliche Boolesche Variablen für die Implementierung der Signale und berücksichtigen Sie folgende Zusammenhänge:

signal s	entspricht	**var bool** s
s.signal	entspricht	s:=**true**
s.wait	entspricht	s:=**false ; await** s **then nop endwait**

- Verwenden Sie nach Bedarf `wait(), notify()` und `notifyAll()`.

b) Schreiben Sie ein Java Programm, indem Sie zwei Objekte x und y der Klasse `Counter` verwenden und für jede der parallelen Anweisungen in Aufgabe 25c) einen neuen Thread starten.

- Lassen Sie die parallele Berechnung von z in einer Schleife 50 mal statt endlos oft durchlaufen.
- Erzeugen Sie nach jeder Neuberechnung von z folgende Ausgabe: »Wert von z« = »Wert von x« + »Wert von y«

(P)
- Schicken Sie das Ausgabeprotokoll und die Sourcen per Email an ihren Tutor.
- Ändern Sie Ihre Lösung so, daß Sie anstatt Boolescher Variablen nun Objekte der Klasse `Object` zur Implementierung der Signale verwenden:

signal s	entspricht dann	`Object s = new Object()`
s.signal	entspricht dann	`s.notify()` oder `s.notifyAll()`
s.wait	entspricht dann	`s.wait()`

  Achten Sie darauf, daß Sie die Monitore nicht so verschachteln, daß eine Verklemmung entsteht.

Aufgabe 27 Leser-Schreiber-Problem

Gegeben seien *n* identische Prozesse die auf eine gemeinsame Variable lesend zugreifen und *m* identische Prozesse, die auf diese Variable schreibend zugreifen. Nun soll der Zugriff auf die gemeinsame Variable so synchronisiert werden, dass zu jedem Zeitpunkt mehrere Prozesse lesen und höchstens ein Prozess schreiben kann. Lesende und schreibende Zugriffe sollen sich im gegenseitigen Ausschluss befinden.

(a) Geben Sie eine Lösung dieses Problems

- mit Hilfe von **`await`**
- unter Verwendung von Semaphoren
- unter Verwendung von Monitoren

an.

(b) Welche Probleme treten bei diesen Synchronisationen auf? Gibt es Prozesse, die benachteiligt bzw. bevorzugt werden, d.h. gibt es unfaire Abläufe? Wie könnte man die Programme modifizieren, um diese Benachteiligungen auszuschliessen? Geben Sie dafür eine Synchronisation der Leser und Schreiber unter Verwendung von Monitoren an.