**Betriebssysteme** **WS 2007/2008**
Dr. Henrik Brosenne Georg-August Universität Göttingen
Institut für Informatik

# Übung 7

## Aufgabe 1

Nach einem Wettkampf gehen die Schwimmer $S_1$ ... $S_n$ $(n \epsilon \mathbb{N})$ eines Turniers duschen. In jeder Dusche soll maximal ein Schwimmer gleichzeitig duschen. Da nur zwei Duschen vorhanden sind, soll sichergestellt sein, dass höchstens zwei Schwimmer gleichzeitig duschen. Erstelle dafür eine Lösung mit Semaphoren, die jedem Schwimmer eine freie Dusche zuweist. Gebe hierzu kommentierten Pseudo-Code an für

- die Initialisierung von gemeinsamen Variablen/Semaphoren,
- einen beliebigen Schwimmer $n$.

## Aufgabe 2

Die in Aufgabe 1 von Übung 5 betrachtete Race-Condition soll nun in der Programmiersprache *Java* untersucht werden.

1. Implementiere die beiden konkurrierenden Threads in Java. Experimentiere, welche Größenordnung diesmal `COUNT_MAX` haben sollte, damit auf dem verwendeten Rechner mit hoher Wahrscheinlichkeit eine Race-Condition auftritt.

   Hinweise:

   - Threads können in Java u.a. verwendet werden, indem für jeden benötigten Thread ein Objekt instantiiert wird, das von der Klasse `Thread` abgeleitet ist. Auf einem solchen Objekt kann dann die Methode `start()` aufgerufen werden, um einen neuen Thread zu erzeugen[1]. Die Anweisungen, die als Thread ausgeführt werden, müssen in der Methode `public void run()` desselben Objekts platziert werden. (`start()` ruft dann intern `run()` auf.)
     - Da die Threads von zwei *verschiedenen* Objekten ausgeführt werden, reicht es nicht, die Zählervariable `in` als Attribut innerhalb der von `Thread` abgeleiteten Objekte zu haben. Die saubere Kapselung, die eine objektorientierte Sprache wie Java bietet, erfordert in diesem Fall etwas mehr Aufwand. (Z.B. `static` Attribute oder indem ein separate Klasse `counter` mit einer Methode `increment()` eingeführt wird und beide Threads eine gemeinsame Instanz dieser Klasse verwenden.)

[1] Es ist nicht erlaubt, auf dem selben Objekt mehrmals `start()` aufzurufen. Um mehrere parallele Threads zu erzeugen, müssen daher mehrere solche Objekte instantiiert werden.

- Auf das Ende eines Threads kann gewartet werden, indem auf dem jeweiligen von `Thread` abgeleiteten Objekt `join()` aufgerufen wird. Da `join()` eine `InterruptedException` werfen kann, muss diese abgefangen werden.
- Statt des *C* Datentyps `long int` sollte unter *Java* `long` verwendet werden.

2. Verwende das Monitorkonzept, um den wechselseitigen Ausschluss zu erreichen.

   Hinweis: Alle innerhalb eines Java Objekts als `synchronized`[2] gekennzeichneten Methoden werden als Methoden eines gemeinsamen Monitors aufgefasst: Sobald eine derart gekennzeichnete Methode betreten wird, wird eine Objekt-interne Sperre gesetzt, die erst wieder beim Verlassen dieser Methode zurückgesetzt wird. Jede Instanz hat seine eigene Sperre. Bei `static synchronized` Methoden bezieht sich die Synchronisation auf eine Sperre, die nur einmal pro Klasse vorhanden ist. Die Klassen-Sperre und die Objekt-Sperren sind unabhängig voneinander.

[2]Das Schlüsselwort `synchronized` muss zwischen Rückgabetyp und Methodennamen platziert werden.