Übungsblatt 6

Ausgabe: 03.06.2008
Abgabe: 11.06.2008, 16:00 Uhr

Aufgabe 1 (6 P)

Für den Leser/Schreiber-Ausschluss sollen geeignete Sperroperationen bereitgestellt werden, mit denen man eine Lese- bzw. Schreibsperre setzen und wieder aufheben kann, z.B. so:

```
lock1.Rlock();
... // read data protected by lock1
lock1.unlock();
lock2.Wlock();
... // modify data protected by lock2
lock2.unlock();
```

Für Objekte wie das hier verwendete lock ist eine Klasse `RWlockImpl implements RWlock` mit folgender Spezifikation zu entwickeln:

```
interface RWlock { // für Lese/Schreib-Sperrobjekte
  void Rlock() throws LockErrorException;
  // setzt Lesesperre, sobald keine Schreibsperre gesetzt ist;
  // Ausnahmemeldung, wenn laufender Faden bereits eine Sperre ge

  void Wlock() throws LockErrorException;
  // setzt Schreibsperre, sobald keine Lese/Schreibsperre gesetzt
  // Ausnahmemeldung, wenn laufender Faden bereits eine Sperre ge

  void unlock()throws LockErrorException;
  // löscht die vom laufenden Faden gesetzte Sperre;
  // Ausnahmemeldung, wenn laufender Faden keine Sperre gesetzt h
}
```

Implementieren Sie `RWlockImpl` in Java (mit **wait**/**notify**/**notifyAll**)! Erweitern Sie dafür die Methodensignaturen so, dass auch eine `InterruptedException` geworfen werden kann.Welche Fairness-Eigenschaften hat Ihre Implementierung?

Achtung: Lassen Sie sich nicht zu sehr von java.util.concurrent.locks beeinflussen! Die dortigen Methoden haben eine etwas andere Semantik.

Aufgabe 2 (10 P)

a) (4 P)

Implementieren Sie in Java ein boolesches Semaphor (pro P() und V() kann man immer

nur eine Flagge kriegen/freigeben, siehe Folien 3.3). Sie dürfen hierfür Synchronisationsmechanismen von Java verwenden. Benutzen Sie außerdem den folgenden Klassenrumpf:

```
class BooleanSemaphore {
  ...
  BooleanSemaphore(int init) {...}
  void P() {...}
  void V() {...}
}
```

b) (6 P)

Implementieren Sie nun unter Einsatz Ihrer booleschen Semaphoren ein allgemeines Semaphor (pro P und V können mehrere Flaggen angefragt/freigegeben werden). Benutzen Sie hierfür keine zusätzlichen Synchronisationsmechanismen von Java. Sie sollten jedoch das boolesche Semaphor aus Aufgabenteil a verwenden. Orientieren Sie sich an dem folgenden Klassenrumpf:

```
class ExtendedSemaphore {
  ...
  ExtendedSemaphore(int init) {...}
  void P(int n) {...}
  void V(int n) {...}
}
```

Hinweise zur Abgabe:

Die Abgabe der Lösungen erfolgt auf zwei Wegen:

- Abgabe auf Papier:

  Bitte schreiben Sie zur jeder Aufgabe eine Dokumentation, in der Sie anhand von *relevanten* Codefragmenten Ihre Implementierung erläutern. Die Codefragmente sollten möglichst kurz gehalten werden und nur der Orientierung für den Leser dienen; entscheidend sind Ihre Erläuterungen, was diese Fragmente machen und wieso Sie sich für diese Art der Implementierung entschieden haben. Die Dokumentation ist *bis Mittwoch 16:00 Uhr in den Tutorenfächern* abzugeben .

- Abgabe per E-Mail:

  Schicken Sie ein ZIP-Archiv an Ihren Tutor, welches den vollständigen Quellcode sowie ausführbare JAR-Dateien für die jeweiligen Aufgabenteile enthält. Der Betreff der E-Mail sollte wie folgt aussehen: "[ALP4] Blatt X - Namen Y".

Zum Bestehen des Übungsblattes müssen 50% der Punkte erreicht werden.