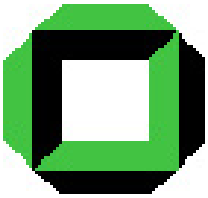

UNIVERSITÄT KARLSRUHE (TH)
Fakultät für Informatik
System Architecture Group
Frank Bellosa, Gerd Liefländer, Philipp Kupferschmied
Dominik Bruhn, Atanas Dimitrov,
Jonathan Dimond, Johannes Weiß

Basispraktikum Systemarchitektur - WS 2008/2009

Schleusenmodellierung

1 Thematik

In der Literatur zum parallelen Programmieren (*concurrent programming*) spielen Dijkstra's Semaphoren eine herausragende Rolle, stellen sie doch das historisch erste brauchbare Konzept zur Behandlung von parallelen Aktivitäten dar. Motiviert wurden sie durch die auf Hollands Kanälen sehr häufig anzutreffenden Schleusen. Auch bei diesen ist die Nutzung einer beschränkten Ressource zu verwalten.

2 Grundlagen

2.1 Semaphoren

Eine Zählsemaphore S ist ein Synchronisationsobjekt, das nach seiner Initialisierung nur über die beiden atomaren und wechselseitig-exklusiven Methoden p(S) und v(S) verwendet werden kann. Die Semantik der Semaphore wird durch folgende Ganzzahlwerte charakterisiert:

- Ein positiver Ganzzahlwert repräsentiert die Zahl der Threads, die aktuell in ihrem kritischen Abschnitt eintreten dürfen
- Ein negativer Ganzzahlwert repräsentiert die Zahl der Threads, die aktuell vor ihrem kritischen Abschnitt warten müssen
- Hat die Semaphore den Wert 0, dann ist aktuell die erlaubte Anzahl von Threads in ihren kritischen Abschnitten und es wartet auch kein anderer Thread.

Die Namen der Semaphormethoden stammen von den holländischen „Passeren" und „Verlaaten", welche als Signale zum Einfahren in bzw. zum Verlassen der schon erwähnten Schleusen verwendet werden.
Die maximale Anzahl der Threads, die gleichzeitig in den kritischen Abschnitt eintreten dürfen, wird durch eine entsprechende Initialisierung der Semaphorenvariable festgelegt.

2.2 Modellüberlegungen

In Java werden die Semaphorobjekte erst im JDK Version 1.5.0 angeboten. In früheren Versionen existiert nur das sehr viel mächtigere Monitorkonzept. Ein Monitor entspricht einem Objekt, dessen Schnittstellenfunktionen (*member functions*) unter gegenseitigem Ausschluss stehen, d.h. so beschaffen sind, dass höchstens ein Thread eine Monitorschnittstellenfunktion ausführen kann. Monitorobjekte werden in Java als *synchronized class* modelliert.

Ein Thread kann mittels der Funktion *wait()* an einer Bedingungsvariablen (*condition variable*, in Java das Monitorobjekt) blockiert werden und so den Monitor temporär verlassen, bis er durch ein *notify()* bzw. *notifyAll()* in einem anderen Thread wieder deblockiert wird. Wie üblich garantiert Java nicht, dass der am längsten angehaltene Thread durch ein *notify()* bzw. *notifyAll()* aufgeweckt wird, es wird irgendeiner der wartenden Thread bei *notify()* deblockiert.

Eine typische Programmstelle innerhalb einer synchronisierten Monitorprozedur eines Monitorobjekts sieht wie folgt aus:

while (condition) try {wait();} catch (Interrupted Exception e} {}

Man beachte, dass an obiger Programmstelle der aufrufende Thread angehalten wird, was gleichbedeutend mit einer Threadumschaltung ist.

Hinweis: *Bei der Lösung der Aufgaben dürfen Sie die in jdk 1.5 angebotene Implementierung des Semaphorkonzepts nicht verwenden. Sie müssen die Semaphoren selbst entwickeln.*

2.2 Schleusenmodellierung

Modellieren Sie mittels der von Ihnen implementierten Semaphoren sowie direkt mit Java-Monitoren verschiedene Schleusen unterschiedlicher maximaler Kapazität.

Frage: Wie behandeln Sie in den beiden Fällen eine volle Schleuse (möglichst sinnvoll)?

3 Experimente

3.1 Schleuse mit Kapazität n>1

Modellieren Sie *mittels der von Ihnen implementierten Klasse Semaphoren* eine Schleuse mit der Kapazität n und lassen sie eine Menge von zufällig ankommenden Schiffen korrekt abarbeiten. Wenden sie ihre Lösung sowohl auf eine Schleuse an, die nur in einer Richtung befahren wird, als auch auf eine, die in beiden Flussrichtungen „vernünftig“ funktioniert. Realisieren Sie diese Schleusen im Anschluß auch unter direkter Verwendung der Java-Monitore.

Frage: Gibt es Ähnlichkeiten zwischen der Lösung mit Semaphoren und der Lösung unter direkter Verwendung der Monitore ? Wenn ja, warum ?

Wichtig: Achten Sie darauf, dass die korrekte Funktionsweise Ihres Programms sowohl aus dem Code als auch aus der Textausgabe ersichtlich ist.

3.2 Graphische Schleuse mit Kapazität n>1

Denken Sie sich eine geeignete Visualisierung und Steuerung der Schleusenbenutzung aus. Wenden sie ihre Lösung sowohl auf eine Schleuse an, die nur in einer Richtung befahren wird, als auch auf eine, die in beiden Flussrichtungen „vernünftig“ funktioniert, sowohl in der Variante mit Semaphoren als auch der mit Monitoren.
Ein graphischer Dialog zur Auswahl von Schleusenkapazität und Synchronisations-Variante ist genauso zulässig (aber bevorzugt), wie die Verwendung von Aufrufparametern.