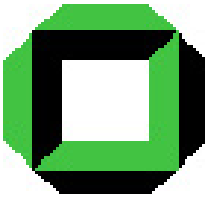

UNIVERSITÄT KARLSRUHE (TH)
Fakultät für Informatik
System Architecture Group
Frank Bellosa, Gerd Liefländer, Philipp Kupferschmied
Dominik Bruhn, Atanas Dimitrov,
Jonathan Dimond, Johannes Weiß

Basispraktikum Systemarchitektur - WS 2008/2009

Kooperation von Threads („Drehkreuz“)

1 Thematik

Zwecks Zusammenarbeit nebenläufiger bzw. paralleler Threads kann man sich auf so genannte gemeinsame Daten abstützen. Man kann in diesem Fall von Kooperation sprechen, da eine vernünftige Zusammenarbeit auf gemeinsamen Daten kritisch sein kann. Während das reine Lesen gemeinsamer Information unkritisch ist, ist das Verändern von gemeinsamer Information im allg. kritisch und muss demzufolge durch besondere Maßnahmen koordiniert werden. Programmabschnitte, die Zugriffe auf gemeinsame Daten beinhalten, nennt man deswegen auch kritische Abschnitte (*critical sections*). Auch gleichzeitiges Lesen und Schreiben kann zu inkonsistenten Daten führen.

Frage 1: Geben Sie ein einfaches Beispiel an, wie gleichzeitiges Lesen und Schreiben auf gemeinsamen zu inkonsistenten Daten führen kann.

Hinweis: Machen Sie sich dabei klar, wie Anweisungen einer höheren Programmiersprache in die jeweilige Maschinensprache (bei Java in den Bytecode) übersetzt werden.

2 Grundlagen

Das gemeinsame Datum im Versuch sei ein Ganzzahlzähler, dessen Wert w, $w \in [0, max]$ einer Raumbenutzung durch w Personen entsprechen soll. Es gebe zwei Türen zum Betreten bzw. Verlassen des Raumes, an denen jeweils Drehzählkreuze angebracht sind. Man entwerfe zwei zyklische Threads, die das Verhalten des Eingangs- bzw. Ausgangsdrehzählkreuzes modellieren. Zufallsgesteuert sollen $x \geq 1$ Personen den Raum betreten bzw. verlassen wollen.

2.1 Modellüberlegungen

Frage 2: Kann es somit prinzipiell zur Überschreitung der maximalen Raumbenutzungszahl max bei der modellierten Benutzerzahl kommen? Begründen Sie Ihre Ansicht!

2.2 Java Monitore

Ein historisch relativ früh entwickeltes Sprachkonzept zur Zwangsserialisierung von kritischen Abschnitten sind Hoare's-Monitore. 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. Mittels des Konzepts der synchronisierten Klassen (*synchronized*) kann man in Java ein Monitorobjekt modellieren. Hierzu sollen Sie zwei Monitorprozeduren `enter()` und `leave()` konstruieren, die das Betreten bzw. Verlassen des Raumes modellieren.

Die Randfälle leerer Raum bzw. überfüllter Raum sind abzufangen, d.h. diese Bedingungen müssen innerhalb der beiden obigen Monitorprozeduren überprüft werden. Sollte beispielsweise jemand den vollen Raum betreten wollen, muss das verhindert werden, indem dieser Thread blockiert wird. Ein blockierter Thread konkurriert solange nicht mehr um den Prozessor, bis er wieder deblockiert, also bereit (*runnable*) wird. Ein Thread kann mittels der Funktion *wait()* an einer Bedingungsvariablen (*condition variable*) blockiert werden und so das Monitorobjekt verlassen, bis durch ein *notify()* bzw. *notifyAll()* in einem anderen Thread dieser blockierte Thread wieder deblockiert wird. Wie üblich garantiert Java nicht, dass der am längsten angehaltene Thread durch ein *notify()* bzw. *notifyAll()* aufgeweckt wird, es wird irgendein wartender 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: *Falls Sie Probleme mit der Java-Syntax haben, dann lesen Sie das Java-Tutorial zum Thema Threads (The Java Tutorial / Essential Java Classes / Threads: Doing Two or more Tasks at Once). Die URL lautet zur Zeit:*
http://java.sun.com/docs/books/tutorial/essential/threads/

2.3 Semaphoren

Semaphoren waren das historisch erste korrekt funktionierende Synchronisationskonzept. 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:

1. Ein positiver Ganzzahlwert repräsentiert die Zahl der Threads, die aktuell in ihrem kritischen Abschnitt eintreten dürfen
2. Ein negativer Ganzzahlwert repräsentiert die Zahl der Threads, die aktuell vor ihrem kritischen Abschnitt warten müssen
3. 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 in Holland häufig anzutreffenden Schleusen dienen.
Die maximale Anzahl der Threads, die gleichzeitig in den kritischen Abschnitt eintreten dürfen, wird durch eine entsprechende Initialisierung der Semaphorenvariable festgelegt.

In Java werden die Semaphorobjekte erst im JDK Version 1.5.0 angeboten. In früheren Versionen existiert nur das sehr viel mächtigere Monitorkonzept und die Semaphoren mussten auf Basis der Monitore implementiert werden. Implementieren und testen Sie Ihre Semaphoren zunächst getrennt.

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

3 Experimente

3.1 Ein Eingangs- und ein Ausgangsdrehkreuz

Die Java-Applikation beinhalte als Aufrufparameter die initiale Zahl von Raumbenutzern, sowie jeweils die maximale Zahl von Zu- und Abgängen.
Implementieren Sie dieses Szenario zunächst direkt mit Monitoren und im Anschluß daran mittels einer selbst implementierten Semaphore – die Auswahl soll mittels eines Aufrufparameters erfolgen.

Frage 3: Wie unterscheiden sich die beiden Varianten Ihrer Lösung ? Welche Gemeinsamkeiten bestehen (warum) ?

3.2 Mehrere Ein-/Ausgangsdrehkreuze

Man erweitere obiges Rahmenprogramm so, dass man mittels Aufrufparameter zusätzlich die Zahl der Ein- bzw. Ausgangsdrehkreuze angeben kann. Auch diese Variante soll sowohl unter Verwendung von Semaphoren, als auch direkt mittels der Java-Monitore realisiert werden.

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

3.3 Ereignisknöpfe mit einem Ein- und einem Ausgangsdrehkreuz.

Entwerfen Sie eine geeignete Visualisierung und Steuerung der Raumbenutzung. Man könnte beispielsweise als graphischen Pegel den jeweiligen Wert des Raumbenutzungszählers visualisieren.
Unter Verwendung des Rahmenprogramms für Experiment 3.1 realisiere man ein Beobachtungsfenster mit zwei Knöpfen (buttons), die bei Anklicken das jeweilige Ereignis Betreten bzw. Verlassen modellieren. Man fange unmögliche Ereignisse ab!

Hinweis: Alternativ zu Aufrufparametern können Sie zum Starten der Experimente auch Eingabedialoge entwerfen. Idealerweise fassen Sie alle Experimente unter eine GUI zusammen.