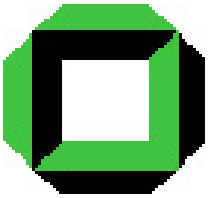

UNIVERSITÄT KARLSRUHE (TH)
Fakultät für Informatik
System Architecture Group
Frank Bellosa, Gerd Liefländer, Philipp Kupferschmied
Dominik Bruhn, Atanas Dimitrov,
Jonathan Dimond, Johannes Weiß

Basispraktikum Systemarchitektur - WS 2008/2009

Gezielte Threadumschaltung

1 Thematik

Java-Threads werden auf unterschiedlichen Plattformen unterschiedlich abgewickelt. Das teilweise von sonstigen Systemaktivitäten beeinflusste Threadverhalten (siehe Versuch 0: Ausgabeintensität von Texten bzw. graphischen Objekten) soll durch einen eigenen Umschaltbaustein beeinflusst werden können. Entwickeln Sie mit Hilfe der im Versuch 0 benutzten Funktionen zur Manipulation von Threads bzw. mit sonstigen Operationen der Threadklasse einen Umschaltmechanismus, der Ihre Java-Anwendungsthreads nach einem Zeitscheibenverfahren umschaltet, also so wie beispielsweise Java-Threads in WindowsXP abgewickelt werden.

Hinweis: Der von Ihnen zu programmierende Umschalter ist selbst als Java-Thread zu entwerfen! Beachten Sie ferner, dass die Zeitscheiben mindesten 10 msec lang sind!

2 Grundlagen

In der einschlägigen Literatur sind verschiedene zeitscheibenorientierte Umschaltverfahren *(time slice scheduling*) vorgeschlagen worden. Prinzipiell geht es darum, dass ein Prozess bzw. ein Anwenderthread ohne Inspektion durch den Umschalter höchstens Δt Zeiteinheiten auf dem Zentralprozessor arbeiten kann, d.h. nur eine Zeitscheibe des Zentralprozessors für seinen Aktivitätsfortschritt erhält. Sofern der Anwenderthread bedingt umschalten muss, etwa weil er auf eine Nachricht von einem anderen Thread oder auf die Beendigung einer synchronen Ein-/Ausgabeoperation warten muss, geht man bei der nächsten Umschaltung auf diesen Anwenderthread vereinfachend davon aus, dass er wieder eine ganze Zeitscheibe verbrauchen darf. Man unterscheidet hierbei *statische* und *dynamische* Zeitscheibenverfahren. Erstere bieten für alle Anwenderthreads nur eine einheitliche, systemweite Zeitscheibengröße an. Letztere können auch threadspezifische Zeitscheibengrößen unterstützen.

Bei Kernel-Level Threads funktioniert dieses Zeitscheibenverfahren wie folgt: Der Kernablaufplaner (*kernel scheduler*) schaltet auf einen neuen Thread um, wobei er in dem Zeitgeberbaustein (*timer)* die entsprechende Zeitscheibengröße –umgerechnet in Anzahl Tics- festlegt. Wenn dieser asynchron mitlaufende Timer auf Null heruntergezählt hat, gibt es eine durch den Timer ausgelöste Zeitscheibenunterbrechung (*timer interrupt*), in deren Verlauf der Kernelscheduler wieder aktiviert wird und eine neue Threadumschaltung durchführt.

3 Experimente

Machen Sie sich zunächst mit den grundlegenden Synchronisationsmechanismen in Java vertraut, indem Sie z.B. die entsprechenden Kaptitel im Java-Tutorial oder entsprechende Literatur lesen. Ihnen sollten mittlerweile das Schlüsselwort *synchronized* sowie die Methoden *wait()* und *notify()* der Klasse java.lang.Thread ein Begriff sein. Eine detaillierte Beschreibung sowie die theoretischen Grundlagen zu den entsprechenden Problemen finden Sie in den Versuchsbeschreibungen der entsprechenden Experimente.

Hinweis: Wenn Sie den Umschalterthread vor dessen Start mittels `setDeamon()` auszeichnen, dann bricht das Anwendungssystem ab, wenn alle Anwendungsthreads beendet sind.

3.1 Gezielte Threadumschaltung mit konstanter Zeitscheibe

Lassen Sie sich etwas einfallen, wie die Threads aus Versuch 3.5 des ersten Übungsblattes sich gezielt umschalten lassen, so dass jeder Thread eine gewisse, zu Beginn festgelegte Zeit $\Delta t > 0$ läuft (die zugebilligte Zeit bezeichnet man auch als Zeitscheibe). Ist die Zeit Δt verstrichen, soll der Thread anhalten (bzw. vom zu implementierenden Scheduler angehalten werden) und ein anderer Thread wieder t Sekunden lang laufen, bis auch dieser wieder angehalten wird und wieder der nächste an der Reihe ist.
Die Reihenfolge, in der die Threads an die Reihe kommen, sollte so vom Scheduler festgelegt sein, dass alle Threads gleich oft an die Reihe kommen.

Verwenden Sie für die Realisierung der Thread-Umschaltung die Funktionen `wait()` und `notify()`der Klasse java.lang.Thread .

3.2 Zeitscheibenverfahren mit individuellen Zeitscheibengrößen

Wiederum sind bis zu zehn unterschiedlich lange, rechenintensive Anwenderthreads mit threadspezifischen Zeitscheibengrößen umzuschalten. Wie die unterschiedlichen Zeitscheibengrößen festgelegt werden bleibt ihnen überlassen, sollte aber im Protokoll beschrieben werden. Der Fortschrittszustand der Anwenderthreads ist wiederum graphisch anzuzeigen!

3.3 Bevorzugung von interaktiven Threads

Im System gebe es eine minimale Zeitscheibengröße. Alle längeren Zeitscheiben seien ein ganzzahliges Vielfaches dieser minimalen Zeitscheibe. Wiederum sind bis zu zehn unterschiedlich lange Anwenderthreads mit threadspezifischen Zeitscheibengrößen umzuschalten. Dabei gebe es zwei Arten von Anwenderthreads: Die einen seien interaktiv und die anderen rechenintensiv. Letztere verbrauchen grundsätzlich ihre Zeitscheiben, während erstere an geeigneten Stellen eine Pseudoausgabe* produzieren, deren Zeitdauer ebenfalls ein ganzzahliges Vielfaches der minimalen Zeitscheibe sein soll. Während dieser Pseudoausgabe soll der nächste Thread mit seiner Arbeit beginnen können (dazu muss der die Pseudoausgabe initiierende Thread den Prozessor verlassen). Es soll zum Start einstellbar sein, mit welcher Wahrscheinlichkeit Prozesse interaktiv sind. Der Fortschrittszustand der Anwenderthreads ist wieder graphisch anzuzeigen!

3.4 Dreistufiges Multiebenenmodell

Diesmal sind bis zu zwanzig Anwenderthreads umzuschalten. Diese Threads besitzen thread-spezifische minimale Zeitscheibengrößen t_n (n = Thread-ID), die zum Programmstart beispielsweise durch Eingabeparameter festgelegt werden. Die Anwender-threads werden initial alle als interaktiv eingestuft. Die Anwenderthreads besitzen folgende Eigenschaften:

1. Hat einer der Anwenderthreads seine Zeitscheibe ohne Pseudoausgabe aufgebraucht, wird er als *ausgeglichener* Thread eingestuft und erst wieder aufgenommen, wenn kein interaktiver Thread fortsetzbar ist.
2. Ausgeglichene Threads erhalten eine Zeitscheibe von $2*t_i$. Sofern ein ausgeglichener Thread seine Zeitscheibe wiederum vollständig ohne Pseudoausgabe verbraucht, wird er schließlich zum rechenintensiven Thread.
3. Rechenintensive Threads erhalten eine Zeitscheibe von $4*t_i$. Sie werden nur dann vom Scheduler ausgewählt, wenn weder ein interaktiver noch ein ausgeglichener Thread fortgesetzt werden kann.
4. Nachdem ein ausgeglichener bzw. ein rechenintensiver Thread eine Pseudoausgabe durchgeführt hat, wird er sofort interaktiv.

Hat ein Thread seine Aufgabe fertig berechnet, dann wechselt er in den Status „beendet“, egal, ob er zuvor interaktiv, ausgeglichen oder rechenintensiv war.
Stellen Sie in geeigneter Form den Fortschritt der Anwenderthreads dar (Benutzen Sie z.B. verschiedene Farben für die unterschiedlichen Threadstufen und für die Pseudoausgabe).

Letzter Vorführtermin: Mittwoch, 03. December 2008, 15:00 Uhr

*Pseudoausgabe: E/A, die eine feste Anzahl an Zeiteinheiten benötigt. Wird eine Pseudoausgabe gestartet, so muss im TCB vermerkt werden, dass dieser Thread gerade „mit einer Pseudo-ausgabe beschäftigt ist“ und somit bis zum Ende dieser Pseudoausgabe vom Scheduler nicht berücksichtigt wird. Eine Pseudoausgabe bedeutet in diesen Versuchen, dass der Thread eine Meldung ausgibt, dass er mit der Ausgabe beginnt und dann keine weitere Rechenleistung mehr beansprucht. Er sollte ab dann also so lange, wie die Pseudoausgabe dauern soll, einfach schlafen und so die Rechenleistung der CPU freigeben.