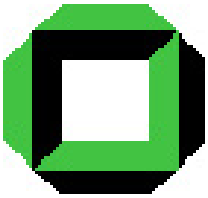

UNIVERSITÄT KARLSRUHE (TH)
Fakultät für Informatik
System Architecture Group
Frank Bellosa, Gerd Liefländer, Philipp Kupferschmied
Dominik Bruhn, Atanas Dimitrov,
Jonathan Dimond, Johannes Weiß

Basispraktikum Systemarchitektur - WS 2008/2009

Modellierung einer Bibliothek / I/O-Scheduling

Ähnlich wie beim Zuteilen des Prozessors an mehrere quasi-gleichzeitig ausgeführte Threads, muss das Betriebssystem konkurrierende Zugriffe der laufenden Prozesse/Threads auf Ein-/ Ausgabe-Geräte (allgemein Peripherie) koordinieren. Beispielsweise würden gleichzeitige Zugriffe auf einen Drucker zu völlig unbrauchbaren Resultaten führen. Ebenso würden gleichzeitige Zugriffe auf Dateien, ohne zusätzliche Koordinierungsmaßnahmen, eher zur Zerstörung wertvoller Datenbestände als zu deren sinnvoller Verarbeitung führen (siehe Leser-Schreiber-Problem).

In dieser Aufgabe soll an einem konkreten Beispiel der quasi-gleichzeitige Zugriff mehrerer Nutzer (Anwendungsthreads) auf eine Speicherressource modelliert werden. Zusätzlich soll dabei die Rolle der Optimierung beim Zugriff auf Peripherie untersucht werden, da diese bei heutigen Rechnern im Vergleich zum Prozessor um Größenordnungen langsamer arbeitet (der Strohhalm für den von-Neumann-Flaschenhals).

Szenariobeschreibung

Als Hintergrund des Modells dient folgende -archaisch entstellte- Universitätsbibliothek: Der Bestand an Literatur beschränkt sich auf 1000 Bücher, die in einem geschlossenen Magazin auf 5 Stockwerke verteilt sind (1.OG: Bücher 1-200, 2. OG 201-400 ...). Die Bibliotheksnutzer greifen auf diesen Bestand zu, indem sie dem im Eingangsbereich wartenden HiWi eine Bestellung (=Nummer des Buches) auftragen, bzw. sich zunächst in die Schlange der Wartenden einreihen. Dieser setzt sich umgehend (aus dem Erdgeschoss) in Bewegung und benötigt zum Erklimmen eines Stockwerks 200ms. An der richtigen Stelle angekommen, entnimmt er in 20ms ein Buch aus den Regalen und macht sich auf den Rückweg, wobei er die gleiche Zeit pro Stockwerk benötigt (die anderen Bibliotheksmitarbeiter sind bereits der Raserei auf den Treppen zum Opfer gefallen). Wieder unten angekommen übergibt er das geholte Buch dem Auftraggeber und nimmt die nächste Bestellung entgegen.

Die Tatsache, dass ein Buch nur von einem Nutzer gleichzeitig ausgeliehen werden kann, sowie die Rückgabe der Bücher kehren wir dabei unter den virtuellen Teppich.

Frage 0.1.1: Welches real vorkommende Peripheriegerät könnte man mit dieser Bibliothek vergleichen? Beschreiben Sie kurz das Zeitverhalten des realen Geräts und vergleichen Sie es mit dem Modell.

Frage 0.1.2: Welche Bücher würden Sie im 1.OG einsortieren, welche im 5.OG? Warum?

Modellierungsvorgaben

Jeder Bibliotheksnutzer soll durch einen eigenständigen Thread modelliert werden. Die Realisierung des HiWis sowie das Verhalten der Nutzer wird jeweils in den folgenden Einzelaufgaben vorgegeben. Offensichtlich ist aber immer zu gewähren, dass in der Modellwelt des Java-Programms immer nur ein HiWi gleichzeitig existiert bzw. arbeitet. Verwenden Sie dazu die von Java bereitgestellten Synchronisationsmechanismen synchronized, wait() und notify().

Hinweis 1: Entwerfen Sie ihre Klassen und Methoden mit dem Ziel der Wiederverwendbarkeit. Verwenden Sie möglichst viel gemeinsamen Code zwischen den Teilaufgaben. Sie sparen sich dabei viel Arbeit, wie das mehrfache Einpflegen kleinster Änderungen in duplizierte Klassen mit der gleichen Funktionalität.

Aufgabe 1: Erste Version des HiWi auf Ebene von exklusiven Methoden

Legen Sie dazu zunächst eine klare Schnittstelle zwischen Anwendungsthread und Bibliothek fest (am besten als interface in java). Damit könnnen Sie in den weiteren Aufgabenteilen die Implementierung des HiWi austauschen, ohne die Anwendungsthreads ändern zu müssen.
Entwickeln Sie nun Methoden, die die Funktionalität und das Zeitverhalten des HiWi abbilden. Orientieren sie sich dabei an der Szenariobeschreibung. Diese sollen von den Anwendungsthreads direkt aufgerufen werden und müssen daher so angelegt sein, dass sie nur ein Thread gleichzeitig betreten kann.
Schreiben Sie dann einen kleinen Test-Bibliotheksnutzer(-Thread), der ein festgelegtes (kurzes) Muster von Büchern abfragt und die während seines Ablaufs verstrichene Realzeit misst. Resultierend sollen sie ein Programm erhalten, das 2<=m<=5 Test-Nutzer startet und die Zeitmessungen ausgibt.
Die Ausgabe soll zunächst textuell erfolgen, denken Sie aber schon jetzt auch an die zur graphischen Visualisierung notwendigen Komponenten.

Frage 1.1: Schätzen Sie die Laufzeit (Realzeit) ihres Test-Threads im worst- und best-case ab und beschreiben Sie jeweils, wie es zu diesem Verhalten kommt. Verhält sich ihr Programm entsprechend? Erkennen Sie eine Tendenz zu einem der Extreme unter Verwendung der classic-VM ??

Aufgabe 2: Bibliotheksnutzerthreads mit klassischen I/O-Mustern

Schreiben sie jetzt einige Anwendungsthreads, die typische Ein-/Ausgabeverhalten simulieren und ihre Laufzeit messen. Die Aktivitätsmuster der Threads sollten festgelegt sein (keine zufälligen Veränderungen), damit die Zeitmessungen sinnvolle Vergleiche unterschiedlicher Randbedingungen zulassen.
Zu implementieren sind: Ein rechenintensiver Thread, der nur selten auf Bücher zugreift, in einer Variante mit Büchern aus hohen und einer mit Büchern aus niedrigen Stockwerken, sowie ein I/Ointensiver Thread, der fast ausschliesslich mit dem Ausleihen von Büchern aus unterschiedlichsten Stockwerken beschäftigt ist.
Überlegen sie sich ein weiteres, in einer realen Bibliothek denkbares Zugriffsmuster und implementieren sie auch dieses als Anwendungsthread.

Experimentieren Sie nun mit der „realistischen“ Situation, dass viele unterschiedliche Benutzer die Bibliothek gleichzeitig nutzen. Schreiben sie als Frontend zur schnellen Durchführung solcher Versuche eine GUI, in der Sie die Anzahl und die Art (auch gemischt) der Nutzerthreads einstellen können.

Frage 2.1: Welche Art von Bibliotheksnutzer wird in der Konkurrenzsituation am wenigsten, welche am meisten zu ungunsten seiner Laufzeit beeinträchtigt?
Begründen sie ihre Antwort und bestätigen oder widerlegen sie sie mit Zeitmessungen.

Aufgabe 3: Bibliotheks-HiWi als eigenständiger Thread mit Warteschlange

Realisieren Sie nun den Laufburschen der Bibliothek als eigenständigen Thread. Die Nutzerthreads reihen sich bei Bestellung eines Buches in eine Warteschlange ein, die der HiWi-Thread der Reihe nach abarbeiten und dabei jeweils die belieferten Threads wieder aufwecken soll. Beachten Sie auch hier: Ein Thread kann nach einer Buchbestellung erst weiterarbeiten, wenn er das gewünschte Buch erhalten hat.

Hinweis: Das Einreihen in die Warteschlange implementieren sie am besten als von den Nutzerthreads aufgerufene Methode, analog zum Vorgehen in Aufgabe 1.

Frage 3.1: Beobachten sie Laufzeitunterschiede im Vergleich zur Situation ohne eigenen HiWi-Thread. Wenn ja, wie erklären sie sich diese? Und treten sie bei unterschiedlichen „Mischungen“ von Threads in gleicher Ausprägung auf ?

Aufgabe 4: Optimierung

Intuitiv ist klar, dass niemand für jedes einzelne Buch ein gigantisches Treppenhaus erklimmen würde. Eher würde man mehrere Buchbestellungen zusammenfassen.
Erweitern Sie Aufgabe 3 entsprechend diesem Konzept, wobei einstellbar viele Buchbestellungen zusammengefasst werden. Nehmen sie ein möglichst optimales Durchlaufen der Stockwerke bei der Berechnung der Bearbeitungszeit an.

Frage 4.1: Wieviele Nutzerthreads müssen gleichzeitig laufen, um eine merkbare Beschleunigung zu erreichen. Welchen Einfluß haben darauf die unterschiedlichen Typen von Anwendungsthreads. Nennen sie zusätzlich zu ihren Beobachtungen jeweils eine(n) Erklärung(sversuch).

Aufgabe 5: Visualisierung

Entwerfen Sie eine graphische Ausgabe zur Visualisierung der Vorgänge.
Dargestellt werden sollen:
- listenartig die Wartschlange mit Name/Nummer und Typ der wartenden Threads
- eine Zugriffsstatistik über die Stockwerke
- pro Thread sein Fortschritt, Zustand (wartend/rechnend) sowie Rechen- und I/O-Zeit

Mit Rechenzeit ist die Realzeit (wall time) gemeint, die der Thread bisher außerhalb der I/O-Routinen für andere Operationen zur Verfügung hat. Die tatsächliche CPU-Zeit des Threads muss nicht erfasst werden.

Kombinieren Sie die bisherigen Aufgabenteile nun zu einem „Gesamtkunstwerk“:
Entwerfen Sie eine Starter-GUI, aus der die Aufgabenteile 1 bis 4 aufgerufen werden können. Die Visualisierung während des Ablaufs muss entsprechend dem Aufgabenteil eingeschränkt werden (z.B. keine Warteschlange bis Aufgabe 4).

Hinweis: System.currentTimeMillis() liefert die aktuelle Zeit in Millisekunden zurück.