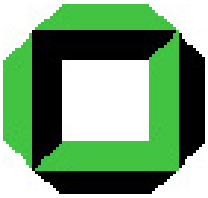

UNIVERSITÄT KARLSRUHE (TH)
Fakultät für Informatik
System Architecture Group
Frank Bellosa, Gerd Liefländer, Philipp Kupferschmied
Dominik Bruhn, Atanas Dimitrov,
Jonathan Dimond, Johannes Weiß

Basispraktikum Systemarchitektur - WS 2008/2009

Erzeugung und Abwicklung von Threads

1 Thematik

Ein Prozess ist in der Linux Terminologie ein Objekt, das aus einem ausführbaren Anwenderprogramm und dessen Ausführungsumgebung (insbesondere einem Adressraum) besteht. Ein Prozess im engeren Sinne ist dann die Abarbeitung dieses Anwenderprogramms auf dem System, wobei pro Prozess über einen Befehlszähler die Abarbeitung der Befehlssequenz dieses Anwenderprogramms gesteuert wird. In der Programmiersprache Java wird zur Erzielung paralleler Abläufe auf Anwenderebene das so genannte „Fadenkonzept“ (*thread concept*) angeboten. Der Begriff Thread bedeutet wörtlich übersetzt Faden. Hierdurch wollten die Begriffsschöpfer darauf hinweisen, dass Threads im Gegensatz zum klassischen Unix-Prozess nicht über einen eigenen Adressraum, wohl aber über einen eigenen Befehlszähler und einen eigenen Stapel (*stack*) verfügen.

In diesem Versuch soll zunächst eruiert werden, wie Java-Threads auf dem Betriebssystem Linux (evtl. auch auf Windows NT/2000/XP) erzeugt und parametrisiert werden können. Ferner soll beobachtet werden, wie diese Threads auf dem jeweiligen Betriebssystem abgewickelt werden.

2 Grundlagen

Entwerfen Sie eine Java-Anwendung (*application*), die aus dem Hauptprogramm $\mathbf{1 \leq n \leq 5}$ so genannte zyklische Anwenderthreads erzeugt. Jeder Anwenderthread bestehe aus einer for-Schleife, wobei die Anzahl Schleifendurchläufe einstellbar sein soll.

3 Experimente

Hinweise:

1. Entwickeln Sie Ihre Java-Anwendungen systematisch. Kümmern Sie sich beispielsweise im Experiment 3.1 zuerst um die **korrekte Parameterübergabe**, bevor Sie sich um die Programmierung der Threaderzeugung und deren Ausgabe bzw. deren Visualisierung des Prozessfortschritts kümmern.
2. Alle Parameter müssen auf der Kommandozeile übergeben werden; es genügt ***nicht***, dass die Parameter im Quellcode geändert werden können.
   Erklären Sie sich die Rolle des Eingabeparameters der main-Methode in einer Java-Anwendun. (String[] args)

3. Eine besonders elegante Variante wäre es alle Experimente aus einer Klasse heraus zu starten und nur mittels eines zusätzlichen Parameters zwischen den Experimenten zu unterscheiden! (keine Pflicht)

3.1 Erzeugen von n Threads und Ausgabe der Threadidentifikation

Experimentieren Sie so, dass Sie sowohl hinsichtlich der Anzahl der Anwenderthreads, als auch hinsichtlich deren Länge (Anzahl der Schleifendurchläufe) variieren können. Die vorbelegte Bearbeitungszeit eines Anwenderthreads betrage mindestens 30 Sekunden auf einem der Poolrechner. Damit Sie eine einigermaßen reproduzierbare Last auf den heutigen Prozessoren erzielen können, sollten Sie für einen entsprechend rechenintensiven Schleifenkörper sorgen, der nicht allzu stark von Zufallszahlen abhängen sollte. Für diese Last gibt es viele Möglichkeiten, z.B.:

- Implementieren Sie einen einfachen Algorithmus zur Suche nach Primzahlen, d.h. bei jedem Schleifendurchlauf sucht der Algorithmus nach der größten Primzahl zwischen 2 und einer vorgegebenen Obergrenze.
- Implementieren Sie einen Algorithmus, der quadratische Matrizen potenziert
- Lassen Sie sich selbst etwas einfallen, wie man durch Berechnungen eine *bemerkbare* CPU-Last erzeugen kann.

Jede Java-Applikation beinhaltet neben den von Ihnen programmierten Threads (*main* + n Anwenderthreads) auch solche, die bereits in der virtuellen Maschine enthalten sind, u.a. einen Thread für die Speicherbereinigung (*garbage collection*). Wie macht sich dieses bemerkbar?
Halten Sie die Beobachtungs- bzw. Messergebnisse in einem Testprotokoll fest. Aufrufparameter dieser Java-Applikation sind:

- n = Anzahl zu erzeugender Anwenderthreads
- Bearbeitungszeiten der Anwenderthreads in Anzahl Schleifendurchläufe

3.2 Erzeugen von n Threads mit mehrfacher Ausgabe

In diesem Experiment enthalte jeder Thread neben der obigen Schleife eine äußere Schleife. Die innere Schleife ist von allen Threads genau $m \geq 1$ mal zu durchlaufen, m sei Aufrufparameter. Der „Vorbelegungswert“ für die Anzahl der äußeren Schleifendurchläufe ist in diesem Experiment und in den folgenden stets 100. Jeder Thread soll nach jedem m-ten Durchlauf der inneren Schleife den Schleifenzähler der äußeren Schleife und seine eigene Threadkennung (*thread identifier*) ausgeben.

Führen Sie einige Testläufe durch, wobei sie die Parameter n und m systematisch variieren sollten, und halten Sie die Ergebnisse in geeigneter Form fest. Hierzu bietet das Linuxsystem u.a. die Ausgabeumlenkung an, welche die Bildschirmausgabe in eine Protokolldatei umlenkt. Aufrufparameter dieser Java-Applikation sind:

- n = Anzahl zu erzeugender Anwenderthreads
- m = Bearbeitungszeit der inneren Schleife (für alle Threads gleich)
- optional[2]: Anzahl Schleifendurchläufe der äußeren Schleife für die $n' \leq n$ Threads

Hinweise:

1. Sofern nur für n' < n die äußere Schleife per Parameter spezifiziert wird, sollen für die letzten n - n' Threads der Vorbelegungswert gelten.
2. Optional: Parameter können vom Benutzer eingegeben, aber auch weggelassen werden.

3.3 Abwicklung von einem Ausgabe- und n Anwenderthreads

In diesem Experiment soll untersucht werden, in wie weit eine Aufteilung von Anwender- und Ausgabethreads deren Abwicklung beeinflusst. Hierzu wird vom Hauptprogramm *main()* ein globales Feld *buffer* als zentrales Objekt zwischen den Threads eingerichtet. Die n Anwenderthreads legen gemäß ihrer Threadkennungen i im *buffer[i]* ihre äußeren Schleifenzähler (siehe Experiment 3.2) ab.

In regelmäßigen Abständen soll ein ebenfalls vom Hauptprogramm erzeugter Ausgabethread den in *buffer[1..5]* abgelegten Zustand der n Anwendungsthreads ausgeben, wobei zum einen ein relativer Zeitstempel und zum anderen die Einträge von *buffer* zeilenweise ausgegeben werden sollen. Aufrufparameter dieser Java-Applikation sind:

- t = Zeitabstand zwischen zwei Aktivitäten des Ausgabethreads
- n = Anzahl der zu erzeugenden Anwenderthreads
- m = Bearbeitungszeit der inneren Schleife (für alle Threads gleich)
- optional: Anzahl Schleifendurchläufe der äußeren Schleife für die n' ≤ n Threads

Lösungshinweise:

1. Zum regelmäßigen Zeitabstand t: In Java gibt es in der Klasse *java.lang.Thread* die Methode `sleep(time)`, die es gestattet, einen Thread `time` Millisekunden anzuhalten. Nach Ablauf der Frist **kann** der so angehaltene Thread wieder fortgesetzt werden.
2. Zum Zeitstempel: In Java gibt es in der Klasse *java.lang.System* die Methode `currentTimeMillis()`, die als Ergebnis die seit einem bestimmten Datum verstrichene Zeit in Millisekunden liefert.
3. Ohne zusätzliche Koordinierungsmaßnahmen bzw. ohne Prioritäten ist diese Aufgabe in der gewünschten Form nicht lösbar. Können Sie hierfür eine Begründung abgeben? Bewahren Sie sich gleichwohl eine Programmversion für dieses Experiment!

3.4 Abwicklungsbeeinflussung durch Prioritäten

In Java kann man Threads mit der Methode `setPriority()` der Klasse *java.lang.Thread* unterschiedliche Prioritäten zuordnen. Hiermit kann man die Abarbeitung von wichtigen gegenüber weniger wichtigen Threads beeinflussen. Wenden Sie das Prioritätskonzept auf die Threads von Experiment 3.3 so an, dass die in 3.3 gewünschte Ausgabe erreicht wird.

3.5 Graphische Ausgabe von Threadzuständen

Ausgehend von den Anwenderthreads aus den Experimenten 3.2 bis 3.4 soll nun der durch den äußeren Schleifenzähler bestimmte Threadfortschrittszustand graphisch ausgegeben werden. Die Ausgabe selbst soll wieder mittels eines separaten Ausgabethreads durchgeführt werden.

Entwerfen Sie eine Klasse *ThreadBuffer*, welche das Interface *Buffer* implementiert. Ergänzen Sie ferner die Run-Methode der Klasse *GraphicalUserInterface*.

Hinweis:

- Das Interface *Buffer* bzw. die Klasse *GraphicalUserInterface* sind im Verzeichnis */home/basis/java/src/versuch0* im Poolraum des Instituts (1.Stock) abgelegt.

3.6 Erzeugen von n Threads mit n Protokolldateien

Neben der Erzeugung von bis zu n unterschiedlich langen Threads ist in diesem Experiment pro Thread eine Protokolldatei anzulegen. Jeder Thread wird nach jedem m-ten Durchlauf der inneren Schleife in seiner Protokolldatei den äußeren Schleifenzähler als Threadfortschrittszustand und einen aktuellen Zeitstempel ablegen. Entsprechende Methoden sind in der Klasse *java.io.PrintWriter* enthalten. Beispielsweise erfolgt das Erzeugen einer neuen Protokolldatei *filename.txt* durch den Aufruf: *logfile = new PrintWriter(new FileOutputStream("filename.txt"))*.

3.7 Graphische Ausgabe und Protokolldateien

Legen Sie wie in Experiment 3.6 in den Protokolldateien den äußeren Schleifenzähler als Threadfortschrittszustand und jeweils den aktuellen Zeitstempel ab und geben Sie zusätzlich analog zu Experiment 3.5 den dazugehörigen Threadfortschrittszustand auch graphisch aus. Diesmal sollen Sie keinen separaten Ausgabethread verwenden, sondern die Ausgabe soll direkt in den Anwenderthreads angestoßen werden. Verwenden Sie hierzu die Klasse *GraphicalUserInterface* im Verzeichnis */home/basis/java/src/versuch0* im Poolraum des Instituts (1.Stock).

Letzter Vorführtermin: 07. November 2008, 15:00 Uhr