Inhaltsverzeichnis

1 Grundbegriffe

1.1 Threads

Definition 1 (Prozess)**.** *Sequentieller Rechenvorgang*

Definition 2 (sequentiell)**.** *Alle Rechenschritte laufen nacheinander in einer vorgegebenen Reihenfolge ab.*

Definition 3 (Thread)**.** *„leichte“ Variante eines Prozesses*

Allgemeine Tendenz:

1. Systemkern möglichst „schlank“ halten
2. Systemkern möglichst selten betreten

Unterschied zu Prozess:

- Kein eigener Speicherbereich
- Üblicherweise nicht vom Systemkern verwaltet („leight-weight process“), vom Systemkern verwaltet

Vorteile:

- Wechsel zwischen Threads weniger aufwändig als Wechsel zwischen Prozessen
- Threads benötigen weniger Speicher
- Man kann viel mehr Threads (≈ 10.000) als Prozesse (≈ 100) laufen lassen.

Nachteil:
Anwendungsprogrammierer muss sich um Verwaltung der Threads kümmern.

Viele Programmiersprachen bieten heutzutage Programmbibliotheken für Threads an (Beispiel: *PThread* in C). Wir verwenden in dieser Veranstaltung *Java* als Programmiersprache.

Definition 4 (parallel)**.** *Mehrere Threads laufen gleichzeitig auf verschiedenen Rechnerkernen.*

Definition 5 (verschränkt (engl. interleaved))**.** *Threads laufen abwechselnd je ein Stück weit.*

Definition 6 (nebeneinander laufend (auch: nebenläufig, engl. concurrent))**.** *Mehrere Threads laufen parallel oder miteinander verschränkt.*

Auch Mischformen sind möglich.

Unterschied:

Definition 7 (Rechenzeit (cpu time))**.** *Zeit, die der Prozessor mit Rechnen zubringt.*

Definition 8 (Bearbeitungszeit (wall clock time))**.** *Umfasst auch Wartezeiten*

Amdahlsches Gesetz (Gene Amdahl, 1967):

Wenn eine Aufgabe die Bearbeitungszeit a benötigt und der Anteil $0 \leq p \leq 1$ davon parallelisierbar ist, dann benötigt sie auf n Prozessoren die Bearbeitungszeit

$$a\left(1 - p + \frac{p}{n}\right). \tag{1.1}$$

Beispiel:
$p = \frac{9}{10}$, $n = 100$

Beschleunigung (speed up):

$$\frac{a}{a\left(1 - p + \frac{p}{n}\right)} = \frac{1}{1 - \frac{9}{10} + \frac{9}{1000}} \approx 9,17$$

$$\text{Sogar } \lim_{n\to\infty} \frac{1}{1 - p + \frac{p}{n}} = \frac{1}{1 - p} = 10$$

Fazit: Der nicht-parallelisierbare Anteil dominiert die Bearbeitungszeit.

1.2 Nicht-Determinismus

Definition 9 (Nicht-Determinismus)**.** *Das Verhalten eines Systems hat Freiheitsgrade.*

Nicht-Determinismus hat zwei Anwendungen:

1. Möglichkeiten des Verhaltens der Systemumgebung zusammenfassen (engl. don't know nondeterminism)

2. Spielraum für Implementierungen vorsehen (engl. don't care nondeterminism)

Hier: System von Threads
Man muss davon ausgehen, dass die Rechenschritte der Threads beliebig miteinander verschränkt sind. Die Reihenfolge der Schritte eines Threads ist durch sein Programm vorgegeben („Programm-Reihenfolge").
Der Zeitplaner (engl. scheduler) legt zur Laufzeit fest, in welcher Reihenfolge die Schritte

zweier Threads zueinander ablaufen. Man möchte den Zeitplaner in seiner Entscheidungsfreiheit nicht unnötig einschränken, sondern einen möglichst großen Spielraum lassen. Man verlangt deshalb, dass das System von Threads korrekt zusammenarbeitet unabhängig davon, wie der Zeitplaner die Verschränkung bildet. Don't know nondeterminism aus der Sicht des Anwendungsprogrammierers, don't care nondeterminism aus der Sicht des Zeitplaners.

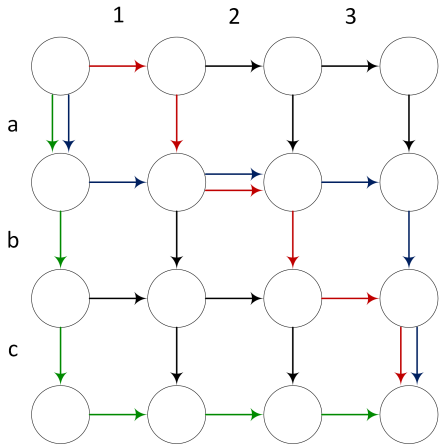
Beispiel:
Thread 1 führt aus: (1) (2) (3)
Thread 2 führt aus: (a) (b) (c)

Beispiele für mögliche Abläufe:

- (1) (a) (2) (b) (3) (c)
- (1) (a) (2) (b) (3) (c)
- (1) (a) (2) (b) (3) (c)

. . .

Mögliche Abläufe visualisiert (Beispiele sind farblich markiert):



Da bei jedem Test der Zeitplaner eine andere Ausführungsreihenfolge (Umstände des Wettrennens, engl. race conditions) wählen kann, ist der Test praktisch nicht reproduzierbar. Wegen der großen Anzahl möglicher Abläufe ist ein systematisches Testen aussichtslos („Zustandsexplosion").

1.3 Kritische Bereiche

1.4 Sperren

2 Verifikation

2.1 Zeitliche Abläufe

2.2 Serielle Abläufe

2.3 Faire Mischung

2.4 Sicherheits- und Liveness-Eigenschaften

2.5 Modellierung

3 Synchronisation

4 Feinkörnige Nebenläufigkeit

4.1 Methoden

4.2 Beispiel: Mengen, grobkörnig

4.3 Beispiel: Mengen, feinkörnig

4.4 Beispiel: Mengen, optimistisch

4.5 Beispiel: Mengen, faul

5 Implementierung

5.1 Atomare Befehle

5.2 Konsenszahlen

5.3 Zwischenspeicher

5.4 Bäckerei-Algorithmus

6 Transactional Memory

6.1 Probleme mit Sperren

6.2 Transaktionen

6.3 Software Transactional Memory (STM)

6.3.1 Transaktionsstatus

6.3.2 Transactional Thread

6.3.3 Zwei Implementierungen

Literaturverzeichnis

[1] Maurice Herlihy, Nir-Shavit: „The Art of Multiprocessor Programming“, Morgan Kaufmann, 2008.

[2] Kalvin Lin, Larry Snyder: „Principles of Parallel Programming“, Addison Wesley.

[3] Greg Andrews: „Concurrent Programming“, Addison Wesley, 1991.

[4] Brian Goetz, u.a.: „Java Concurrency in Practice“, Addison Wesley.