LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME UND DATA MINING

Nebenläufigkeit

SEP 2018

Tobias Lingelmann

2018-04-17

DBS

Wissenschaftliche Betreuer:
Daniel Kaltenthaler, Johannes Lohrer
Verantwortlicher Professor:
Prof. Dr. Peer Kröger

Übersicht

- Nebenläufigkeit
- Prozesse und Threads
- Probleme bei Nebenläufigkeit
- Thread und Runnable
- Synchronisation

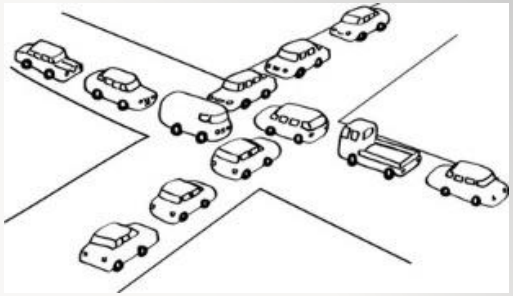
Nebenläufigkeit

- Parallelität
- Gleichzeitige Ausführung mehrerer Berechnungen oder Befehle.
- Unterscheidung zwischen scheinbarer Nebenläufigkeit und echter Nebenläufigkeit.
- Sowohl für unterschiedliche Aufgaben als auch für gemeinsame Aufgaben geeignet.

Prozesse und Threads

- Prozesse
  - Verschiedene Prozesse haben je einen separaten Speicherbereich.
  - Erstellen eines Prozesses vergleichsweise teuer.
- Threads
  - Verschiedene Threads (eines Prozesses) teilen sich einen Speicherbereich.
  - Kann einfach erstellt werden.
  - „Leichtgewichtiger“ Prozess.

Probleme



- Verklemmung („deadlock“)
  - Gegenseitiges Warten auf (bereits) exklusiv belegte Betriebsmittel.
- Verlorenes Update („lost update“)
  - Überschreiben eines Wertes basierend auf veralteten Informationen.
- Wettlaufsituation („race condition“)
  - Ergebnis abhängig von zeitlicher Abfolge
- Verhungern („starvation“)
  - Nichtzuteilung von CPU-Zeit

Java: Thread & Runnable

- MyThread `extends` Thread
  - Überschreiben der `run()`-Methode.
  - `(new MyThread()).start();`
- MyRunnable `implements` Runnable
  - Implementierung der `run()`-Methode
  - Benötigt dannach Thread zum Starten
  - `(new Thread(new MyRunnable())).start();`

Darüber hinaus

java.util.concurrent

- `Executors / ExecutorService`
- `Callable`
- `Future`
- `FutureTask`

javafx.concurrent

- `Worker` (`Task`, `Service`)

Synchronisation

- „Synchronized methods“
  - `public synchronized void inc()`
  - Nur ein Thread kann auf synchronized-Methoden einer Klasse zugreifen.
- „Synchronized statements“
  - `synchronized(Object) { … }`
  - Anstatt einer ganzen Klasse sind nur Statements, die das selbe Monitor-Objekt nutzen, gesperrt.

Synchronisation

- Warten auf einen Thread: `Thread.join(…)`
- Thread warten lassen, bis ein Ereignis eintritt:
  - In `synchronized(Object)`-Block:
    - `wait()` lässt Thread warten.
    - `notify()` weckt einen Wartenden auf.
    - `notifyAll()` weckt alle Wartenden auf.
  - Eventuelle Alternative:
    - `java.concurrent.BlockingQueue`

Linksammlung

- https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
- https://docs.oracle.com/javase/8/javafx/interoperability-tutorial/concurrency.htm
- http://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/
- https://www.artima.com/insidejvm/ed2/threadsynch.html