Parallele Programmierung in Java

Prof. Dr. Uwe Kastens

Sommersemester 2000

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 01

Ziele:

in der Vorlesung:

Verständnisfragen:

Ziele und Durchführung

Ziele:

Die Studierenden sollen

- grundlegende **Konzepte** und höhere Paradigmen zum **Entwurf paralleler Programme**,
- **systematische Methoden** zur Entwicklung paralleler Programme und
- Techniken zur **parallelen Programmierung in Java**

erlernen und **praktisch erproben**.

Durchführung:

- Vorlesung und Übung an zusammenhängendem Termin
- „fliegender Wechsel“ zwischen Vorlesung, Kleingruppenarbeit und Rechnerübung
- Hausaufgaben, um Themen nachzuarbeiten oder vorzubereiten

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 02

Ziele:

Ziele der Vorlesung kennenlernen

in der Vorlesung:

Erläuterungen zu den Zielen und zur verzahnten Durchführung von Vorlesung und Übung.

Verständnisfragen:

Stimmen die Ziele mit Ihren Zielen überein?

Inhalt

Woche	Thema
1	Grundbegriffe zu Prozessen und zu Threads in Java
2	Monitore allgemein und in Java
3	Monitore mit speziellen Wartebedingungen
4	Barrieren
5	Datenparallelität
6	Client-Server-Prinzip
7	Asynchrone Botschaften am Beispiel Client-Server
8	Botschaften in verteilten Systemen
9	Ereignisse in verteilten Systemen und Remote Method Invocation
10	Synchrone Botschaften

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 03

Ziele:

Übersicht über die Vorlesungsthemen

in der Vorlesung:

Erläuterungen zur Auswahl der Themen

Verständnisfragen:

- Welche Themen interessieren Sie jetzt besonders?
- welche weniger?
- Fehlen Ihnen Themen in diesem Plan?

Voraussetzungen

Thema	**z. B. aus der Vorlesung**
praktische Programmiererfahrung mit Java	Software-Entwicklung (SWE)
Grundbegriffe der parallelen Programmierung:	Konzepte und Methoden der Systemsoftware (KMS), SWE
Prozess, Nebenläufigkeit, Parallelität,	KMS 2-5, 2-9,
verzahnte Ausführung,	KMS 2-10
Adressräume, Threads, Prozesszustände	KMS 2-11, 2-12, 2-18
Monitor	KMS 3-37 bis 3-41
Prozess, Nebenläufigkeit, Parallelität,	SWE-131
Threads	SWE-133 bis 137
Synchronisation, Monitore in Java	SWE-142 bis 150

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 04

Ziele:

Quellen zu den Voraussetzungen sichtbar machen

in der Vorlesung:

- Erläuterungen dazu.
- Die Begriffe werden am Anfang dieser Vorlesung kurz wiederholt; in der Form wie inn SWE eingeführt.

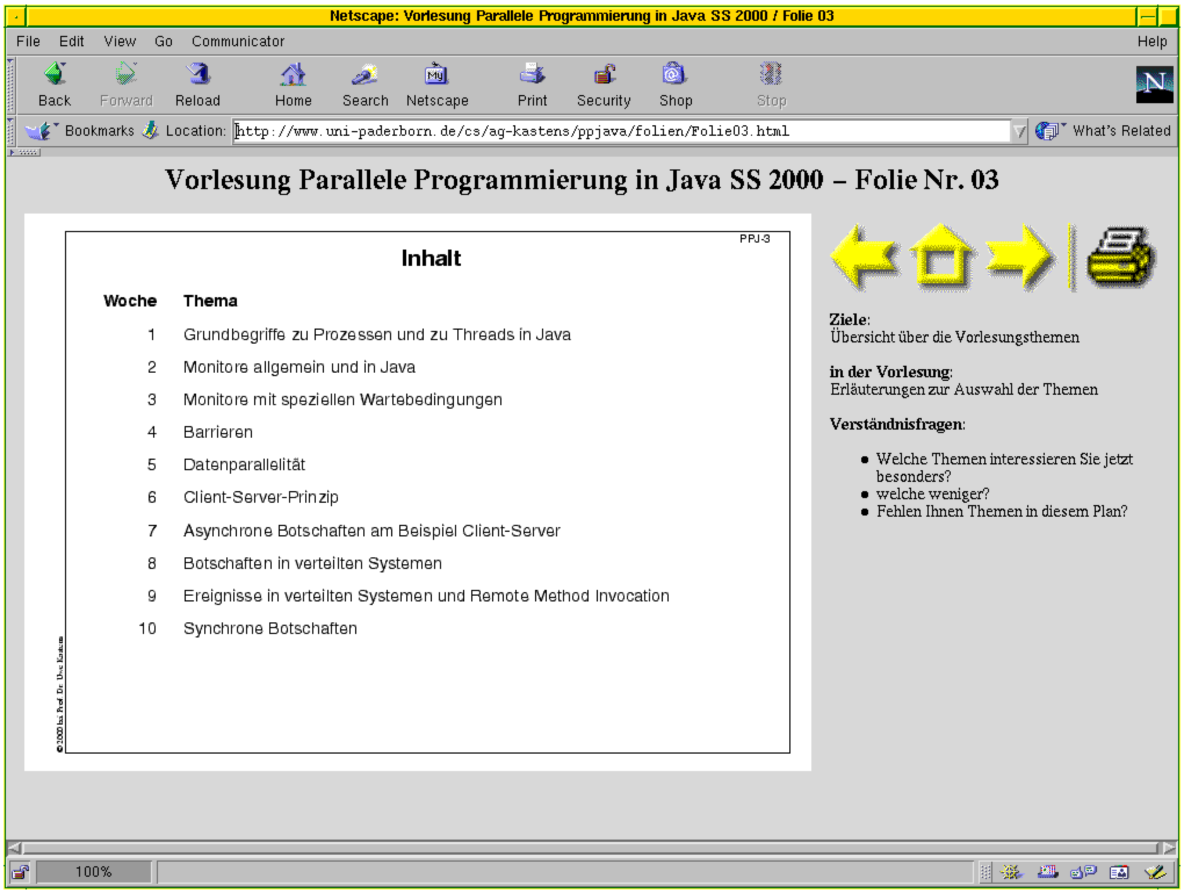
nachlesen:

- Skript zur Vorlesung SWE
- Skript zur Vorlesung KMS

Verständnisfragen:

- Haben Sie die Vorlesungen gehört?
- Wollen Sie den Stoff nachlernen oder wiederholen?

Elektronisches Vorlesungsmaterial im Web



© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 05

Ziele:

Elektronische Vorlesungsmaterial bekannt machen

in der Vorlesung:

- Erläuterungen dazu
- Das Skript wächst mit.

Verständnisfragen:

- Haben sich sich schon in dem Material umgesehen?
- Haben Sie sich Bookmarks für das Skript angelegt?

Literatur

Skript zur Vorlesung „**Parallele Programmierung in Java**“ SS 2000 (wächst mit):
http://www.upb.de/cs/ag-kastens/ppjava

Skript zur Vorlesung „Software-Entwicklung I + II“ WS, SS 1998/1999:
http://www.upb.de/cs/ag-kastens/swei

Skript zur Vorlesung „Konzepte und Methoden der Systemsoftware“ SS 1999:
http://www.upb.de/cs/heiss/lehre/kms/skript.html

Gregory R. Andrews: **Concurrent Programming**, Benjamin Cummings, 1991

Scott Oaks, Henry Wong: **Java Threads**, 2nd ed., O‘Reilly, 1999

Jim Farley: **Java Distributed Computing**, O‘Reilly, 1998

Doug Lea: **Concurrent Programming in Java**, Addison-Wesley, 1996

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 06

Ziele:

Hinweise auf Material und Bücher zur Vorlesung

in der Vorlesung:

Erläuterungen dazu:

- Das Skript zu dieser Vorlesung wächst mit.
- Das Buch von Andrews vermittelt die Konzepte sehr gut und tiefgehend - tiefer als in dieser Vorlesung behandelt.
- Die 3 Bücher zu Java gehen sehr ausführlich auf Prgrammiertechniken ein; viele konkrete Beispiele - allerdings zum Teil nicht besonders übersichtlich.

Verständnisfragen:

Wollen Sie den Stoff anhand einiger Bücher vertiefen?

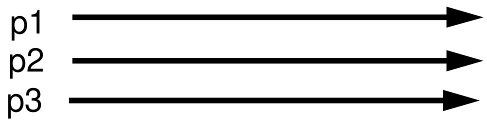
Parallele Prozesse

Prozess:

Ausführung eines sequentiellen Programmstückes in dem zugeordneten Speicher (Adressraum). Veränderlicher Zustand: Speicherinhalt und Programmposition.

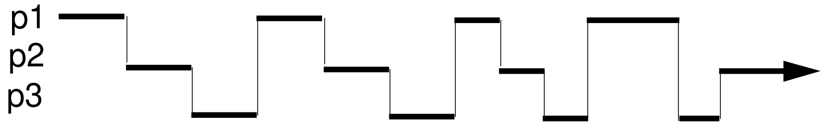
parallele Prozesse:

mehrere Prozesse, die gleichzeitig auf mehreren Prozessoren ausgeführt werden.



verzahnte Prozesse:

mehrere Prozesse, die stückweise abwechselnd auf einem Prozessor ausgeführt werden. Prozessumschaltung durch die Prozessverwaltung oder durch die Prozesse selbst.



Verzahnte Ausführung kann parallele Ausführung simulieren.
Häufiger Wechsel vermittelt den Eindruck, daß alle Prozesse gleichmäßig fortschreiten.

nebenläufige Prozesse:

Prozesse, die parallel oder verzahnt ausgeführt werden können.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 07

Ziele:

Grundbegriffe Paralleler Prozesse wiederholen

in der Vorlesung:

- Begriffe erklären.
- Verzahnte Ausführung verwenden wir auch als Modell um Eigenschaften des Prozesssystems zu beschreiben.

nachlesen:

SWE-131

Verständnisfragen:

- In welchen Situationen veranlasst die Prozessverwaltung eine Prozessumschaltung?

Anwendungen Paralleler Prozesse

- **Benutzungsoberflächen**:
  Die Ereignisse werden von einem speziellen Systemprozeß weitergegeben.
  Aufwendige Berechnungen sollten nebenläufig programmiert werden, damit die Bedienung der Oberfläche nicht blockiert wird.
- **Simulation** realer Abläufe:
  z. B. Produktion in einer Fabrik
- **Animation**:
  Veranschaulichung von Abläufen, Algorithmen; Spiele
- **Steuerung** von Geräten:
  Prozesse im Rechner überwachen und steuern externe Geräte, z. B. Montage-Roboter
- **Leistungssteigerung** durch Parallelrechner:
  mehrere Prozesse bearbeiten gemeinsam die gleiche Aufgabe,
  z. B. paralleles Sortieren großer Datenmengen.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 08

Ziele:

Mit Parallelverarbeitung werden unterschiedliche Ziele verfolgt!

in der Vorlesung:

Es werden Beispiele zu den Anwendungsarten erläutert.

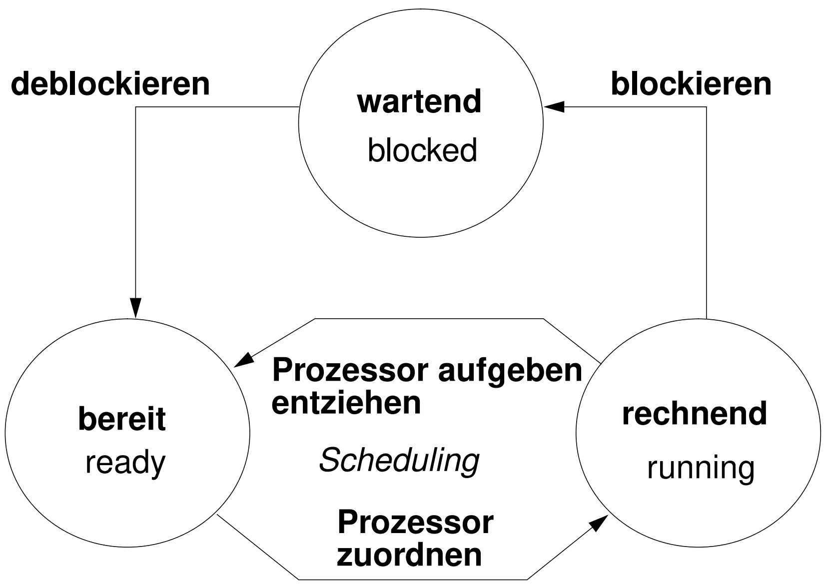
nachlesen:

SWE-132

Verständnisfragen:

Geben Sie weitere Beispiele von parallelen Prozessen in Anwendungen und ordnen Sie sie ein.

Prozesszustände und ihre Übergänge



siehe KMS 2-17, 2-18

Threads (lightweight processes, Leichtgewichtsprozesse):
Prozesse, die parallel oder verzahnt im gemeinsamen Speicher (Adressraum) ablaufen.
Die Prozessumschaltung ist besonders einfach und schnell.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 09

Ziele:

Prozessumschaltung verstehen

in der Vorlesung:

- Erläuterungen der Zustände und Übergänge.
- Rolle des Schedulers.

Verständnisfragen:

- Geben Sie Gründe und Beispiele für die Zustandsübergänge an.

Java Threads erzeugen - Technik Runnable implementieren

Prozesse, Threads in Java:
nebenläufig im **gemeinsamen Speicher** des Programms (o. Applets)
Objekte von Klassen mit bestimmten Eigenschaften

Erste Technik: Eine **Benutzerklasse implementiert das Interface `Runnable`**:

```
class Aufgabe implements Runnable
{ ...
   public void run ()            vom Interface geforderte Methode run
   {...}                         das als Prozess auszuführende Programmstück
   public Aufgabe (...) {...}                                   Konstruktor
}
```

Der Prozess wird als **Objekt der vordefinierten Klasse Thread** erzeugt:

```
Thread auftrag = new Thread (new Aufgabe (...));
```

Erst folgender Aufruf startet dann den Prozess:

`auftrag.start();` Der neue Prozess beginnt, neben dem hier aktiven zu laufen.

Diese Technik (das Interface `Runnable` implementieren) sollte man anwenden, wenn

- der **abgespaltene Prozess nicht weiter beeinflußt** werden muss;
  also einen Auftrag erledigt (Methode `run`) und dann terminiert, oder
- die Benutzerklasse als Unterklasse einer anderen Klasse definiert werden soll.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 10

Ziele:

Deklaration von Prozessklassen verstehen.

in der Vorlesung:

3 Programmierschritte:

- Klasse mit der Methode run deklarieren
- Prozessobjekt erzeugen
- Ausführung des Prozessobjekte starten

Falls in der Benutzerklasse weitere Objektmethoden benötigt würden, wären sie schlecht zugänglich. Dann sollte man die andere Variante verwenden.

nachlesen:

SWE-133

Verständnisfragen:

- Die Klasse Thread hat Klassen- und Objektmethoden. Welche können in der run-Methode auf welche Weise aufgerufen werden?

Java Threads erzeugen - Technik Unterklasse von Thread

Zweite Technik:
Die Benutzerklasse wird als **Unterklasse der vordefinierten Klasse `Thread`** definiert:

```
class DigiClock extends Thread
{ ...
  public void run ()            überschreibt die Thread-Methode run
  {...}                         das als Prozess auszuführende Programmstück
  DigiClock (...) {...}         Konstruktor
}
```

Der Prozess wird als Objekt der Benutzerklasse erzeugt (es ist auch ein **`Thread`**-Objekt):

```
Thread clock = new DigiClock (...);
```

Erst folgender Aufruf startet dann den Prozess:

```
clock.start();
```
der neue Prozess beginnt neben dem hier aktiven zu laufen

Diese Technik (Unterklasse von Thread) sollte man anwenden, wenn
der abgespaltene Prozess weiter beeinflußt werden soll;
also weitere Methoden der Benutzerklasse definiert und von aussen aufgerufen werden,
z. B. zum vorübergehenden Anhalten oder endgültigem Terminieren.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 11

Ziele:

Deklaration von Prozessklassen verstehen.

in der Vorlesung:

3 Programmierschritte:

- Klasse mit der Methode run deklarieren
- Prozessobjekt erzeugen
- Ausführung des Prozessobjektes starten

Gegenüberstellung zur Variante mit Interface Runnable.

nachlesen:

SWE-134

Verständnisfragen:

- Die Klasse Thread hat Klassen- und Objektmethoden. Welche können in der run-Methode auf welche Weise aufgerufen werden?

Wichtige Methoden der Klasse `Thread`

`public void run ();`
wird überschrieben mit der Methode, die die auszuführenden Anweisungen enthält

`public void start ();`
startet die Ausführung des Prozesses

`public void suspend ();`
hält den angegebenen Prozess an: `clock.suspend();`

`public void resume ();`
setzt den angegebenen Prozess fort: `clock.resume();`

`public void join () throws InterruptedException;`
der aufrufende Prozess wartet bis der angegebene Prozess terminiert ist:

```
try { auftrag.join(); } catch (Exception e){}
```

`public static void sleep (long millisec) throws InterruptedException;`
der aufrufende Prozess wartet mindestens die in Millisekunden angegebene Zeit:

```
try { Thread.sleep (1000); } catch (Exception e){}
```

`public final void stop () throws SecurityException;`
nicht benutzen! Terminiert den Prozess u. U. in einem inkonsistenten Zustand

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 12

Ziele:
Übersicht zu Thread-Methoden

in der Vorlesung:
- Erläuterungen zu den Methoden
- Veranschaulichung durch graphische Darstellung der Abläufe der beteiligten Prozesse
- Verweise auf Beispiele

nachlesen:
SWE-137

Übungsaufgaben:
Veranschaulichen Sie die Wirkung der Methoden durch graphische Darstellung der Abläufe der beteiligten Prozesse.

Verständnisfragen:
- An welchen Methodenaufrufen sind zwei, an welchen ist nur ein Prozess beteiligt?

Beispiel: Digitale Uhr als Prozess im Applet (1)

Der Prozess soll in jeder Sekunde Datum und Uhrzeit als Text aktualisiert anzeigen.


```
class DiGiClock extends Thread
{  public void run ()
   {  while (running)                                  iterieren bis von außen terminiert wird
      {  line.setText(new Date().toString());                             Datum schreiben
         try { sleep (1000); } catch (Exception ex) {}                              Pause
      }
   }
                                        Methode, die den Prozess von außen terminiert:
   public void stopIt () { running = false; }
   private boolean running = true;                                     Zustandsvariable

   public DigiClock (Label t) {line = t;}   Label zum Beschreiben übergeben
   private Label line;
}
```

Prozess als Unterklasse von **`Thread`**, weil er

- durch Aufruf von **`stopIt`** terminiert,
- durch weitere **`Thread`**-Methoden unterbrochen werden soll,
- eine andere Oberklasse nicht benötigt.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 13

Ziele:

Erstes vollständiges Prozessbeispiel

in der Vorlesung:

Erläuterungen zur

- Ausführung bis zur Terminierung von außen,
- stopIt-Methode,
- Begründung der Variante "Unterklasse von Thread".

Applet vorführen Digital Clock Process

nachlesen:

SWE-135

Übungsaufgaben:

Installieren Sie das Beispielprogramm und variieren es.

Beispiel: Digitale Uhr als Prozess im Applet (2)

Der Prozess wird in der **init**-Methode der **Applet**-Unterklasse erzeugt:

```
public class DigiApp extends Applet
{  public void init ()
   {  Label clockText = new Label ("--------------------------------");
      add (clockText);

      clock = new DigiClock (clockText);                  Prozess erzeugen
      clock.start();                                       Prozess starten
   }

   public void start ()   { clock.resume(); }            Prozess fortsetzen
   public void stop ()    { clock.suspend(); }            Prozess anhalten
   public void destroy () { clock.stopIt(); }          Prozess terminieren

   private DigiClock clock;
}
```

Prozesse, die in einem Applet gestartet werden,

- sollen angehalten werden (**suspend**, **resume**), solange das Applet nicht sichtbar ist (**stop**, **start**),
- müssen terminiert werden (**stopIt**), wenn das Applet entladen wird (**destroy**).

Andernfalls belasten Sie den Rechner, obwohl sie nicht mehr sichtbar sind.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 14

Ziele:

Prozess aus Applet starten

in der Vorlesung:

Erläuterungen zum Starten, Anhalten, Fortsetzen und Terminieren von Prozessen aus Applets.

nachlesen:

SWE-136

Übungsaufgaben:

Ändern Sie die Klassen dieses Beispiels, so daß DigiClock nicht Unterklasse von Thread ist sondern Runnable implementiert.

Verständnisfragen:

Begründen Sie weshalb die gezeigte Lösung mit einer in der DigiClock Runnable implementiert.

Verzahnung als abstraktes Ausführungsmodell

Zwischen Prozessen, die nicht warten, kann prinzipiell zu **beliebigen Zeitpunkten** umgeschaltet werden. Eine **Scheduling Strategie** schränkt die Beliebigkeit ein.

Beispiel für unterschiedliche Ergebnisse bei unterschiedlicher Verzahnung:
Zwei Prozesse operieren auf einer gemeinsamen Variable:

```
                  konto = 50;
                        a              b             c
                  ____________   ____________   ____________
     Prozess1: t1 = konto; t1 = t1 + 10; konto = t1;

     Prozess2: t2 = konto; t2 = t2 - 5;  konto = t2;
                  ____________   ____________   ____________
                        d              e             f
```

Mit atomaren Zuweisungen *a* - *f* liefert das Programm die gleiche Vielfalt von Ergebnissen wie mit den atomaren Zusammenfassung *<a,b>* und *<d, e>* oder *<b, c>* und *<e, f>*, denn in *b* und *e* kommen keine globalen Variablen vor.

© 2000 bei Prof. Dr. Uwe Kastens

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 15

Ziele:

in der Vorlesung:

- Erläuterungen zu atomaren Operationen.
- Wir sprechen später über Scheduling Strategien.
- Wechselwirkung (Interferenz) zwischen Prozessen über gemeinsame, globale Variable.
- Die gewünschten Programmergebnisse dürfen nicht auf Annahmen über die Verzahnung basieren.
- Alle Ergebnisse, die das Beispiel liefern könnte.

Verständnisfragen:

- Welche Ergebnisse könnte das Beispiel liefern?
- Wie müssen die Operationen atomar zusammengefasst werden, damit das Ergebnis bei jeder noch zulässigen Verzahnung gleich ist?