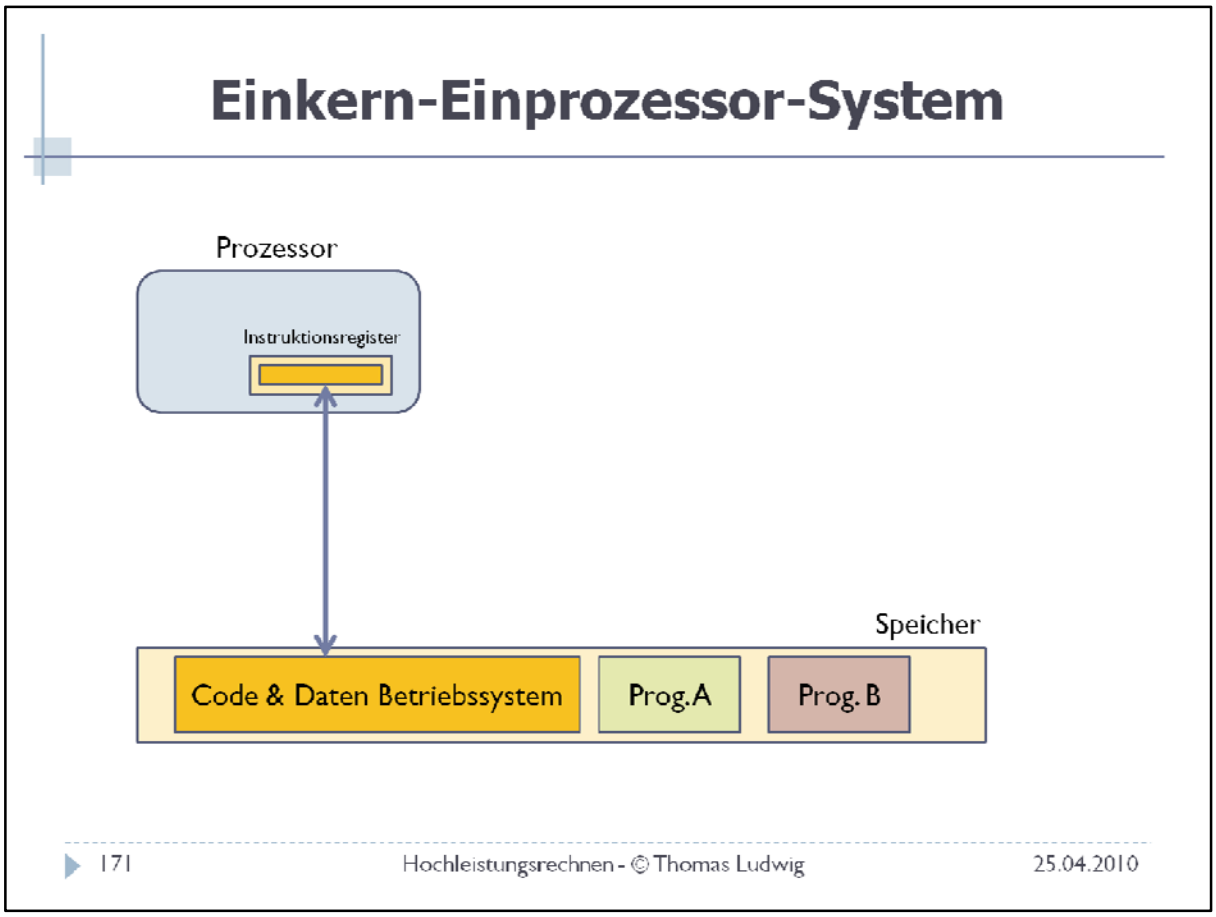
Betriebssystemaspekte

- Einprozessor-Einkern-Systeme
- Mehrprozessor/Mehrkern-Systeme
- SMP-Hardware
- SMP-Betriebssystem
- Synchronisationsmechanismen
- Eweiterte Betriebssystemfunktionalität

Betriebssystemaspekte
Die zehn wichtigsten Fragen

- Wie funktioniert Synchronisation im traditionellen UNIX-Kern?
- Was versteht man unter Wiedereintrittsfähigkeit?
- Wie werden Maschinenbefehle in Mehrprozessor/Mehrkern-Systemen abgearbeitet?
- Was benötige ich für ein Mehrprozessor-System?
- Was kennzeichnet ein SMP-Betriebssystem?
- Welche Varianten von Sperren finden wir im SMP-Betriebssystem?
- Wie wird ein Betriebssystem SMP-fähig?
- Wie funktioniert Synchronisation mittels Semaphor?
- Wie funktioniert Synchronisation mittels Spinlock?
- Welche weiteren Funktionen muß ein SMP-Betriebssystem aufweisen?



Der Cache wird hier immer vernachlässigt, da er zunächst nur auf der Ebene der Prozessorhardware eine Rolle spielt.

Moduswechsel

- Der Prozessor arbeitet
  - **entweder** im Betriebssystemmodus
    - und bearbeitet Code des Betriebssystems
  - **oder** im Benutzermodus
    - und bearbeitet Code des Benutzerprozesses

Wechsel des Modus

- Systemaufruf im Programm (synchron)
- Unterbrechung (asynchron)

172 Hochleistungsrechnen - © Thomas Ludwig 25.04.2010

Siehe:
- http://en.wikipedia.org/wiki/Usermode
- http://en.wikipedia.org/wiki/System_call
- http://en.wikipedia.org/wiki/Interrupt

Bei modernen Betriebssystemen ist die Wahrheit ein bißchen komplizierter, insbesondere, welche Adressen der Prozessor im Betriebssystemmodus bearbeitet. Der Code des Betriebssystem wird hier in den Adreßraum des Anwendungsprozeß eingeblendet und da im Systemmodus abgearbeitet. Davon wollen wir hier abstrahieren.

Moduswechsel sind teuer und sollten vermieden werden, wenn es geht. Dummerweise entsteht bei jeder E/A-Operation ein Moduswechsel.

Synchronisation im traditionellen UNIX-Kern

Feststellung: der UNIX-Kern ist wiedereintrittsfähig (reentrant)

- Mehrere Prozesse arbeiten im Kern zur selben Zeit und evtl. im selben Code-Bereich
- Natürlich nicht echt gleichzeitig sondern verschränkt !
  - Es gibt ja nur einen Prozessor

Frage: Könnte es zu inkonsistenten Daten kommen? Wenn ja, wie vermeidet man es?

- Die Situation könnte auftreten, wenn zwei Prozesse dieselben Daten, z.B. Puffer, benutzen
- Natürlich darf Dateninkonsistenz nicht auftreten

173 Hochleistungsrechnen - © Thomas Ludwig 25.04.2010

Siehe: http://en.wikipedia.org/wiki/Reentrant_%28subroutine%29

Man beachte: Bibliotheken werden natürlich nur einmal in den Hauptspeicher geladen, möglicherweise aber von mehreren Prozessen verwendet. Auch hier muß der Code wiedereintrittsfähig (reentrant) sein.

Synchronisation im traditionellen UNIX-Kern ...

Vermeidung von Inkonsistenzen

- Das Betriebssystem ist zunächst einmal nicht unterbrechbar (non-preemptive)
- D.h. eine BS-Aktivität wird zuendegeführt, auch wenn dadurch die Zeitscheibe des zugehörigen Prozesses überschritten wird
- Nach dem Ende sind alle Datenstrukturen konsistent und ein anderer Prozeß kann unbedenklich damit arbeiten

Aber ...

Siehe: http://en.wikipedia.org/wiki/Preemption_%28computing%29

Synchronisation im traditionellen UNIX-Kern ...

Problem: Unterbrechungen

- Während der Aktivität im BS-Kern könnte eine Unterbrechung erfolgen
- Die Unterbrechungsbehandlungsroutine könnte zufällig dieselben Datenstrukturen bearbeiten

Lösung:

- Vorübergehendes Verbieten von Unterbrechungen durch Erhöhung des *ipl* (interrupt priority level)
- Kritischer Bereich mit Erhöhung/Verringerung eingerahmt

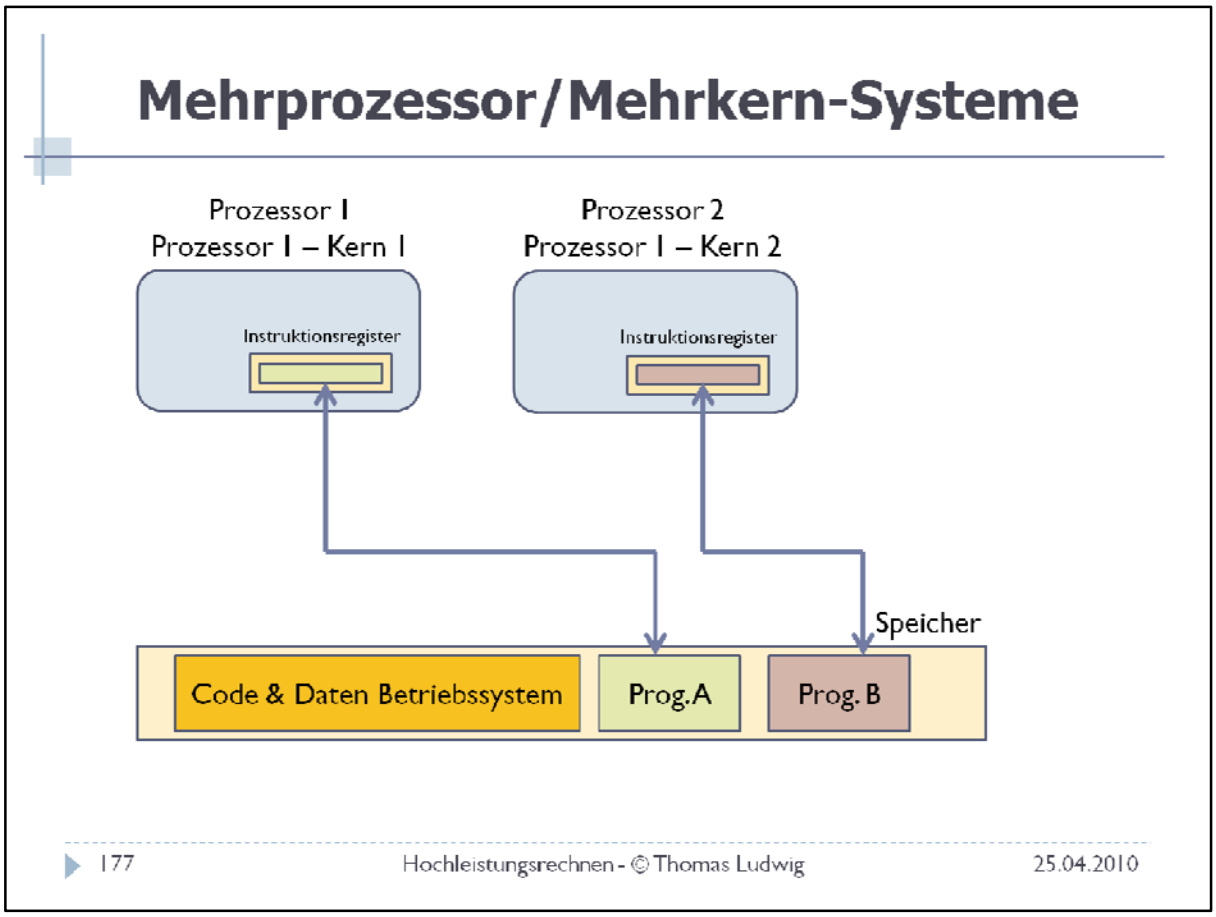
Synchronisation im traditionellen UNIX-Kern ...

Beim Einprozessorsystem

- Synchronisation problemlos möglich
- Nur kleinere Probleme
- Ununterbrechbarkeit des Kerns ist starker Schutz

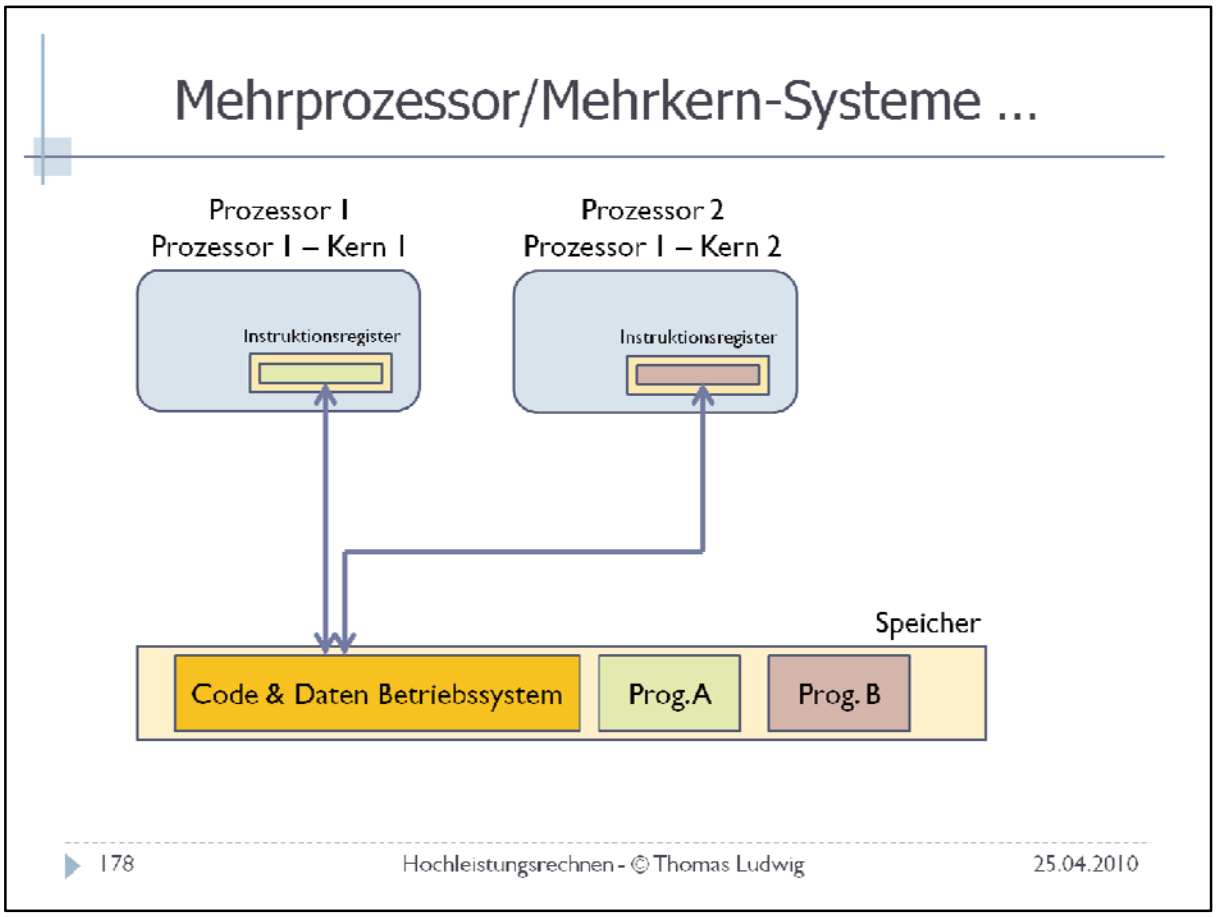
(Bei Echtzeitbetriebssystemen ist die Ununterbrechbarkeit des Kern nicht mehr gegeben – damit neue Probleme)

Siehe: http://en.wikipedia.org/wiki/Realtime_operating_system



Siehe: http://en.wikipedia.org/wiki/Multi-core_processor

Unkritischer Fall: Prozessoren bearbeiten Code von verschiedenen Programmen/Prozessen.



Kritischer Fall: Beide Prozessoren arbeiten gleichzeitig im Code des Betriebssystem.

Mehrprozessor/Mehrkern-Systeme ...

Unkritisch

- Beide Prozessoren bearbeiten Code verschiedener Programme/Prozesse

Beginn aller Probleme

- Ein Prozessor wechselt in den Systemmodus
  - Wegen Systemaufruf oder Unterbrechung
- Frage: Was macht anderer Prozessor?
  - Systemmodus verbieten? Ineffizient!
  - Systemmodus erlauben? Gefährlich!

Mehrprozessor/Mehrkern-Systeme ...

Was benötige ich alles?

- Spezial-Hardware zur Unterbrechungssteuerung an jedem einzelnen Prozessor
- Cache-Kohärenz-Mechanismen
- **Nebenläufig ausführbares Betriebssystem**
  - **Threads im Betriebssystem**

180 Hochleistungsrechnen - © Thomas Ludwig 25.04.2010

Siehe: http://en.wikipedia.org/wiki/Thread_%28computer_science%29

„Nebenläufig“ (concurrent) bedeutet, daß zwei Aktionsstränge gleichzeitig ablaufen, diese aber nicht notwendigerweise miteinander zu tun haben. Von „parallel“ spricht man, wenn die Abläufe zu einem Programm gehören und logisch miteinander verknüpft sind, indem z.B. Synchronisationen erfolgen.

SMP-Hardware

SMP Symmetric Multiprocessing

- Variante des Betriebssytems, die mehrere Prozessoren/Kerne unterstützen kann

Intels Multiprocessor Specification MPS 1.4

- Ist die heutige gültige Referenz
- Beschreibt Intel-SMP-Systeme
- Festlegung der BIOS-Eigenschaften
- Beschreibung des APIC
  Advanced Programmable Interrupt Controller
- Cache-Kohärenz, MESI-Protokoll
- In der Praxis typisch 2- und 4-Wege-Systeme

Siehe:
- http://en.wikipedia.org/wiki /Symmetric_multiprocessing
- http://en.wikipedia.org/wiki/MultiProcessor_Specification
- http://en.wikipedia.org/wiki/Intel_APIC_Architecture

SMP-Betriebssystem

Zunächst eine Begriffsbestimmung

- In einem SMP-System sind alle Prozessoren gleichberechtigt
- Sie greifen gemeinsam auf denselben Code und dieselben Daten des Betriebssystemkerns zu und stehen im Wettbewerb um Systemressourcen
- Jeder Benutzerprozeß kann auf jedem Prozessor zur Ausführung gebracht werden

SMP-Betriebssystem ...

Was bedeutet das?

- Tatsächlich ist das SMP-Betriebssystem ein paralleles Programm auf einer Maschine mit gemeinsamem Speicher
- Der Code wird nebenläufig ausgeführt
  - (vgl. verteiltes Betriebssystem)
- Die Daten können beliebig manipuliert werden
- Inkonsistenzen vermeidet man durch Sperren
  - Sperren von Code-Abschnitten
  - Sperren von Datenbereichen

183 Hochleistungsrechnen - © Thomas Ludwig 25.04.2010

In einem **verteilten** Betriebssystem werden Funktionen des BS auf **unterschiedlichen** Rechnern abgewickelt.

SMP-Betriebssystem ...

Alles dreht sich um die Sperren

- Eine große Sperre (sog. giant lock):
  Nur ein Prozessor kann in das BS
  - Problem gelöst; Effizienz vernichtet
- Viele kurze Sperren:
  Daten bzgl. der Konsistenz geschützt
  - Gute Nebenläufigkeit bei geringen Kosten
- Ganz viele sehr kurze Sperren:
  Daten geschützt
  - Bessere Nebenläufigkeit bei höheren Kosten

Siehe: http://en.wikipedia.org/wiki/Giant_lock

SMP-Betriebssystem ...

Wie mache ich mein Einprozessor-Einkern-Betriebssystem SMP-fähig?

- Analysiere alle Datenstrukturen und Abläufe im BS-Code
  - Zunächst Subsysteme wie Speicherverwaltung, Scheduling, Ein-/Ausgabe etc.
- Schütze kritische Bereich vor gleichzeitigem Zugriff
- Verfeinere den Schutz (mehr Nebenläufigkeit)
  - Inkrementelle Parallelisierung

Synchronisationsmechanismen

- Es müssen jetzt verschiedenste Datenstrukturen geschützt werden, die bei einem einzelnen Prozessor unkritisch waren
- Einfache Flags reichen nicht aus, da sie gleichzeitig manipuliert werden könnten
- Unterbrechungssperren müssen global gesetzt werden können
- Traditioneller Sleep/wakeup-Mechanismus auf Mehrprozessor-Systemen unbrauchbar
- Wiederaufwecken von Threads kritisch
  - Thundering-herd-problem

186 Hochleistungsrechnen - © Thomas Ludwig 25.04.2010

Siehe: http://en.wikipedia.org/wiki/Thundering_herd_problem

Thundering-herd-problem: Beim Freiwerden einer Ressource werden viele Threads gleichzeitig deblockiert, nur um alle bis auf einen sofort wieder blockiert zu werden.

Synchronisationsmechanismen...

Essentiell: Hardware-Unterstützung

- Atomares Testen-und-Setzen
  - Testet Bit, setzt Wert auf ´1´, gibt alten Wert zurück
  - Nach Abschluß ist das Bit ´1´
  - Rückgabewert ´0´: man hat jetzt Zugriff, Ressource war frei
  - Rückgabewert ´1´: Ressource von anderen belegt
- Anweisung kann nicht einmal durch eine Unterbrechung unterbrochen werden
- In vielen Systemen sogar atomare Nutzung des Speicherbusses

Siehe: http://en.wikipedia.org/wiki/Atomic_operation

Synchronisation mittels Semaphor

Standardverfahren:

- P() dekrementiert Semaphor und blockiert, wenn Wert kleiner 0 wird
- V() inkrementiert Semaphor und weckt Thread auf, wenn Wert kleiner oder gleich 0 ist

BS-Kern garantiert Atomarität der Aktion

- Einprozessor-Einkern-System:
  durch Ununterbrechbarkeit der BS-Kerns
- Mehrprozessor-System:
  durch tieferliegende unteilbare Aktion

188 Hochleistungsrechnen - © Thomas Ludwig 25.04.2010

Siehe: http://en.wikipedia.org/wiki/Semaphore_%28programming%29

Synchronisation mittels Semaphor...

Problem

- Blockieren und Aufwecken erfordert Kontextwechsel im Betriebssystem: langsam
- Nicht akzeptabel für kurze Blockierungen
- Weniger aufwendiger Mechanismus benötigt

Synchronisation mittels Spinlock

Einfachster Sperrenmechanismus: Spinlock

- engl: *spin lock, simple lock, simple mutex*
- Skalare Variable
  - ´0´ bedeutet verfügbar
  - ´1´ bedeutet belegt
- Manipulation mittels aktivem Warten (busy-wait) und atomarem Testen-und-Setzen

Siehe: http://en.wikipedia.org/wiki/Spinlock

Synchronisation mittels Spinlock...

```
void spin_lock (spinlock_t *s)  {
  while (test_and_set (s) != 0)  /* belegt */
    ;                            /* warte auf Freigabe */
}

void spin_unlock (spinlock_t *s)  {
  *s = 0;
}
```

Möglicher Nachteil: Busblockierung

Synchronisation mittels Spinlock...

```
void spin_lock (spinlock_t *s)  {
  while (test_and_set (s) != 0) /*belegt*/
    while (*s != 0);
           /* warte auf Freigabe */
}

void spin_unlock (spinlock_t *s)  {
  *s = 0;
}
```

Synchronisation mittels Spinlock...

Verwendung von Spinlocks

- Kurzzeitige Sperre kritischer Bereiche im Betriebssystem-Code
- Niemals für blockierenden Code einsetzen

Analyse

- Aktives Warten (normalerweise unerwünscht)
- Sehr billig, wenn Ressource nicht belegt ist
- Insgesamt billig bei geringem Belegt-Grad

SMP-Betriebssystem... Linux

- Bei Linux dauert(e) dieser Vorgang Jahre!
  - Erstmals SMP-fähig in Kernel 2.2
- Im Kernel 2.4
  - Alle Subsysteme durch feingranulare Sperren abgesichert
  - „Kernel subsystems are fully threaded“
- Skalierbarkeit bzgl. Anzahl der Prozessoren ist problematisch
  - Bisherige Grenze: 4 (?)

Siehe: http://en.wikipedia.org/wiki/Linux_kernel#Architecture

Erweiterte Betriebssystemfunktionalität

Was brauchen wir sonst noch?

- Feingranulare Ausführungsobjekte
  - Kernel-Threads
    Werden im BS-Code erzeugt und teilen sich mit ihm den Adreßraum
- Speicherverwaltung für Mehrprozessor-Systeme
  - Verwaltung mehrerer Speicherbänke
- Scheduling für Mehrprozessor-Systeme

Erweiterte Betriebssystemfunktionalität...

Mehrprozessor-Scheduling

- Prozessor-Affinität

  Ein Prozeß oder ein Thread sollte auf dem Prozessor fortgesetzt werden, auf dem er zuletzt lief

  Grund: Gültiger Inhalt im L2-Cache

- Gang-Scheduling

  Alle Threads eines Prozesses sollten zusammen zugeteilt werden

  Grund: Verzögerungen an gemeinsam genutzten Sperren vermeiden

Siehe:
- http://en.wikipedia.org/wiki/Processor_affinity
- http://en.wikipedia.org/wiki/Gang_scheduling

Betriebssystemaspekte
Zusammenfassung

- Synchronisation in Einprozessor-Einkern-Systemen fast ohne Probleme
- Bei SMP-Systemen läuft das Betriebssystem auf allen Prozessoren
- Hauptproblem: innere Synchronisation und Schutz der Datenstrukturen
- Betriebssystem-Code wird durch den Einbau von Sperren parallelisiert
- Feinere Sperren bedeuten mehr Nebenläufigkeit
- Hardware-Unterstützung für die Sperren notwendig
- Semaphor ist zu kostspielig
- Spinlock ist das wichtigste Synchronisationskonzept
- Scheduling von Threads ist sehr komplex