

Echtzeitsysteme – Nicht- blockierende Synchronisationsverfahren

Benedikt Dremel

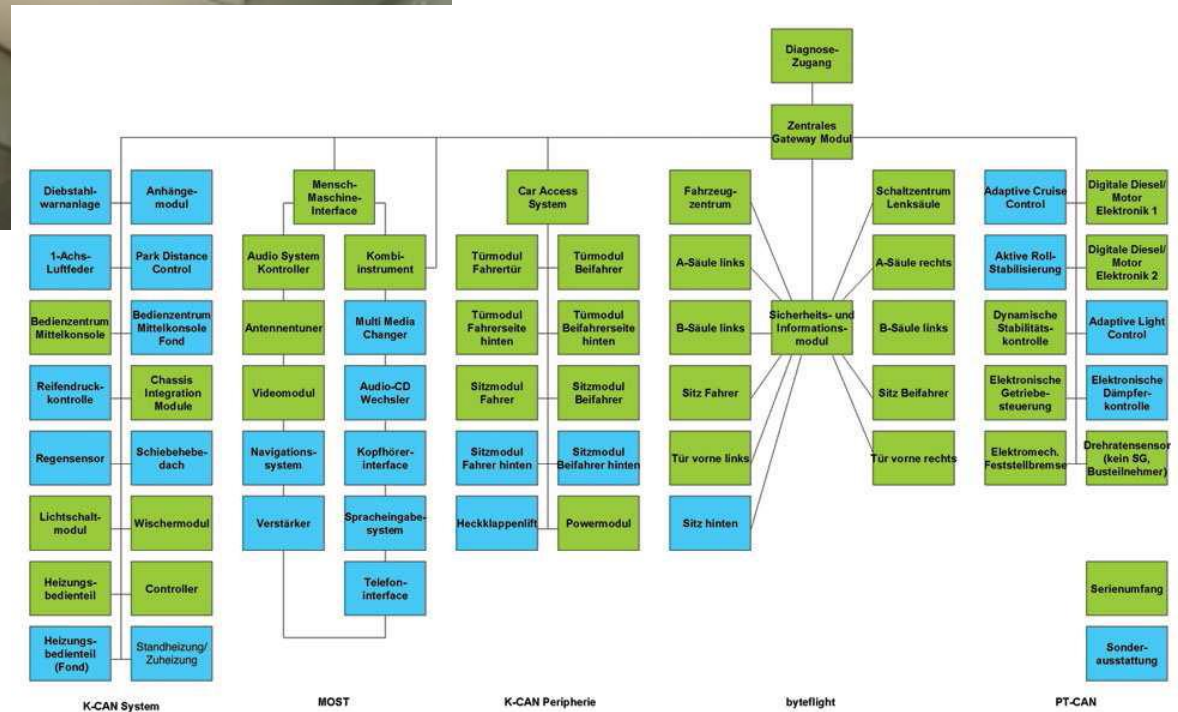
02.07.2009

Ausgewählte Kapitel der Systemsoftware: Multicore- und
Manycore-Systeme



- Echtzeitfähige eingebettete Systeme weit verbreitet
- Ansteigende Anforderungen an Performanz

➔ Zentralisieren auf wenige Recheneinheiten mit Mehrkernprozessoren



Agenda

- ➔ • Grundlagen
 - Der CAS-Befehl
 - Echtzeitproblematik
 - Prioritätenumkehr
- Algorithmen zum Datenaustausch
 - Mit CAS-Befehl
 - Helping mit Wartestapel
 - Single-Server
- Echtzeitfähigkeit
- Multiprozessorfähigkeit
- Fazit

Der CAS-Befehl

- „Compare and Swap“ - Instruktion
- Befehl des Prozessors
- Garantiert atomares Vergleichen und Schreiben

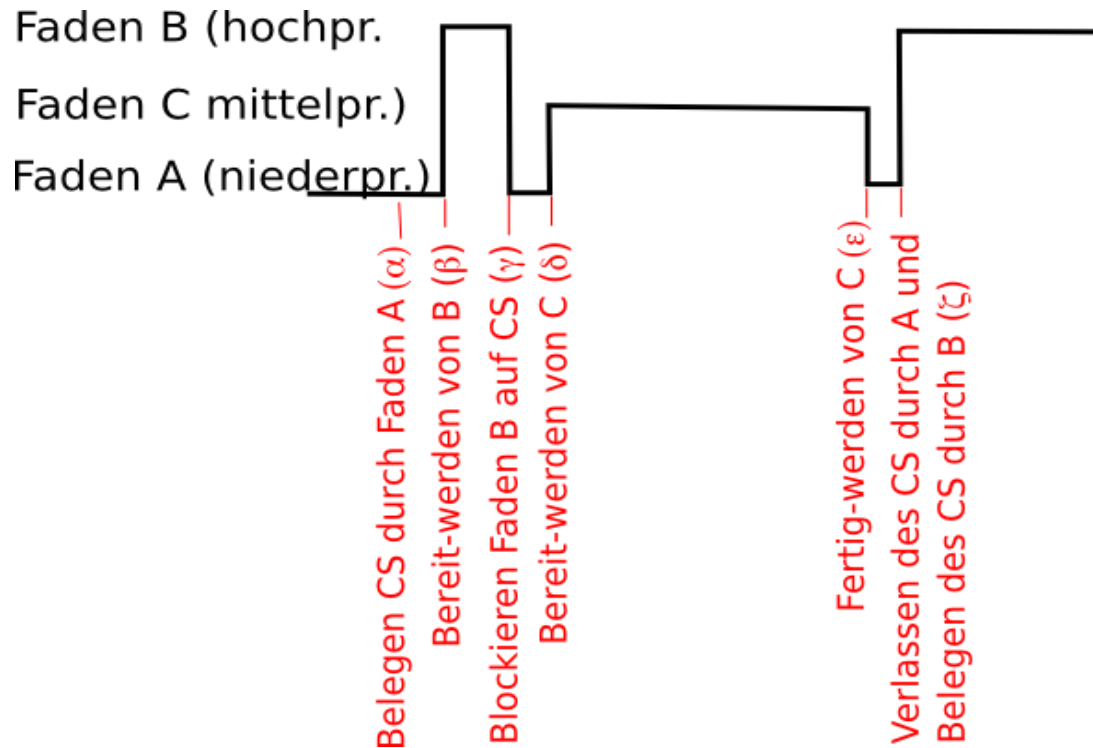
```
1  bool CAS(int *speicherstelle, int alt, int neu)
2  {
3      if(*speicherstelle != alt)
4          {
5              return false;
6          }
7      *speicherstelle = neu;
8      return true;
9  }
10
```

Echtzeitproblematik


- Jeder Auftrag hat Termin (Deadline)
 - muss eingehalten werden
- Häufig prioritätsbasierte Einplanungsstrategie
- Bestimmung der maximale Laufzeit (WCET)
- Problem: Kommunikation zwischen Fäden
 - Konsistenzwahrung durch atomaren Zugriff
 - Verzögerung durch Warten
 - ➔ Termine trotzdem einhaltbar?

Unbeschränkte Prioritätenumkehr

- Mittelpriore Faden C verzögert hochprioren Faden B
- Faden B von niederpriorem Faden A abhängig



Agenda

- Grundlagen
 - Der CAS-Befehl
 - Echtzeitproblematik
 - Prioritätenumkehr
-  • Algorithmen zum Datenaustausch
 - Mit CAS-Befehl
 - Helping mit Wartestapel
 - Single-Server
- Echtzeitfähigkeit
- Multiprozessorfähigkeit
- Fazit

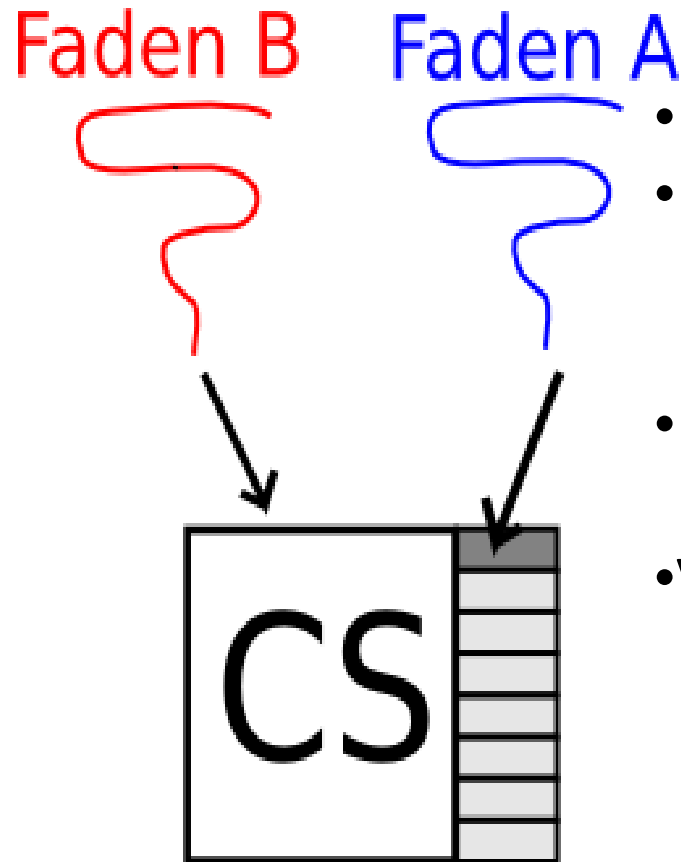
Nicht-blockierende Synchronisation mit CAS (1)

- Pointer zu Datenstruktur global
- Beim Schreiben der Daten:
 - Kopie der Daten und des Pointers
 - Veränderung der lokalen Kopie
 - Umsetzen des Pointers mittels CAS-Instruktion
- Kein Blockieren eines Fadens
 - ➔ Deadlocks ausgeschlossen
- Ansatz einiger Nicht-Echtzeit-Systeme

Nicht-blockierende Synchronisation mit CAS (2)

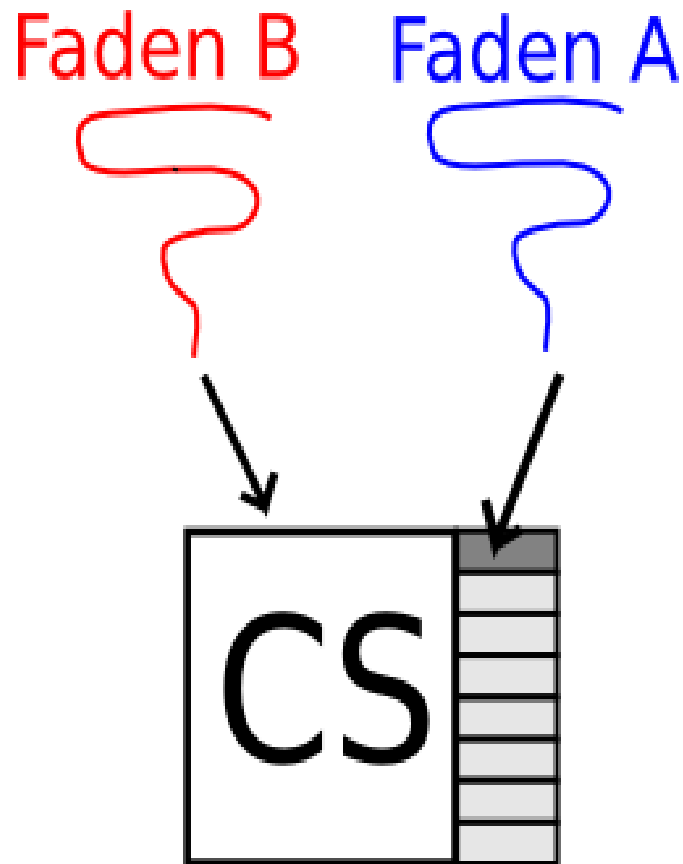
- Problem: CAS scheitert
 - Zugriff auf Daten durch anderen Faden
 - Pointer unterschiedlich → CAS scheitert
- Lösung: Berechnung wiederholen, erneut versuchen
 - Bei mehr als zwei Konkurrenten:
Erneutes scheitern mind. eines
 - Warten mit Backoff-Algorithmus
 - ➔ Streuen der Wiederholungen

Helping mit Wartestapel (1)



- Kritische Abschnitte (CS) geschützt
- Faden A: Versuchter Zugriff auf CS
 - Speicherung auf Stapel
 - Warten bis Faden B fertig
- Übergabe der CPU-Zeit
 - Beschleunigung von B
- Vererbung der Priorität von A an B
 - keine unbeschränkte Prioritätenumkehr

Helping mit Wartestapel (2)



Warum Stapel?

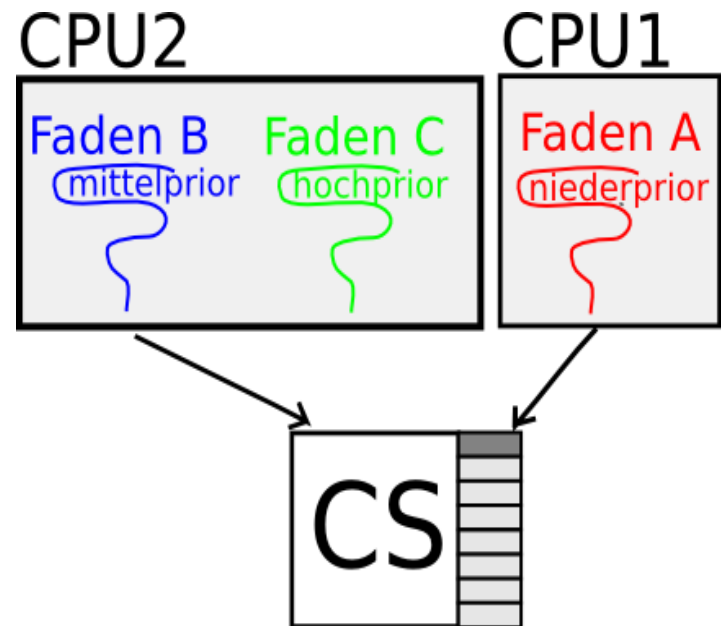
System fester Prioritäten

- Unterbrechung nur durch höherprioreren Faden
- Höchstpriorer Faden immer oben auf Stapel
- Keine Prioritätsverletzung

Helping mit Wartestapel (3)

Helping Varianten:

- Local Helping ($B \rightarrow CPU \text{ von } A$)
- Remote Helping ($A \rightarrow CPU \text{ von } B$)



Helping mit Wartestapel (4)

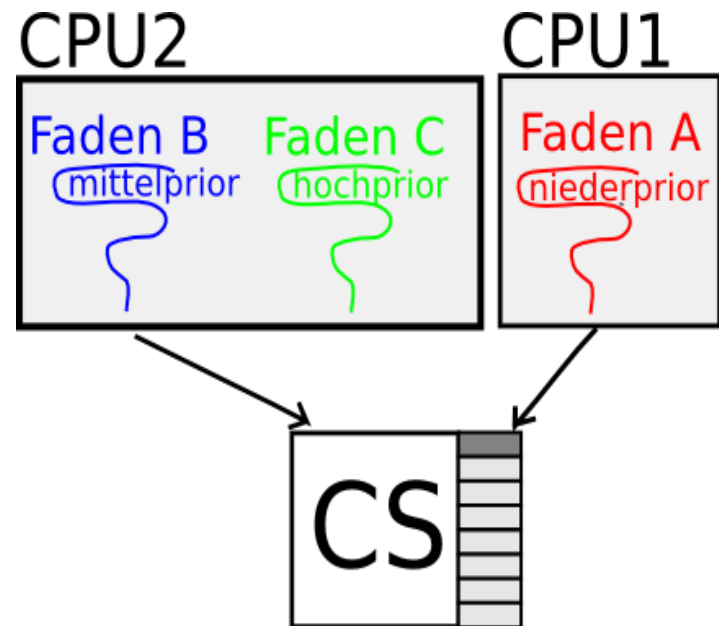
Helping Varianten:

- Local Helping ($B \rightarrow \text{CPU von A}$)
- Remote Helping ($A \rightarrow \text{CPU von B}$)

Beispiel:

- Belegung der CS durch Faden B
- Verdrängung Faden B durch Faden C
- Belegungsversuch der CS durch Faden A

→ Helping durch Faden A



Helping mit Wartestapel (5)

Helping Varianten:

- Local Helping ($B \rightarrow \text{CPU von A}$)
- Remote Helping ($A \rightarrow \text{CPU von B}$)

Unterschied zwischen beiden Varianten:

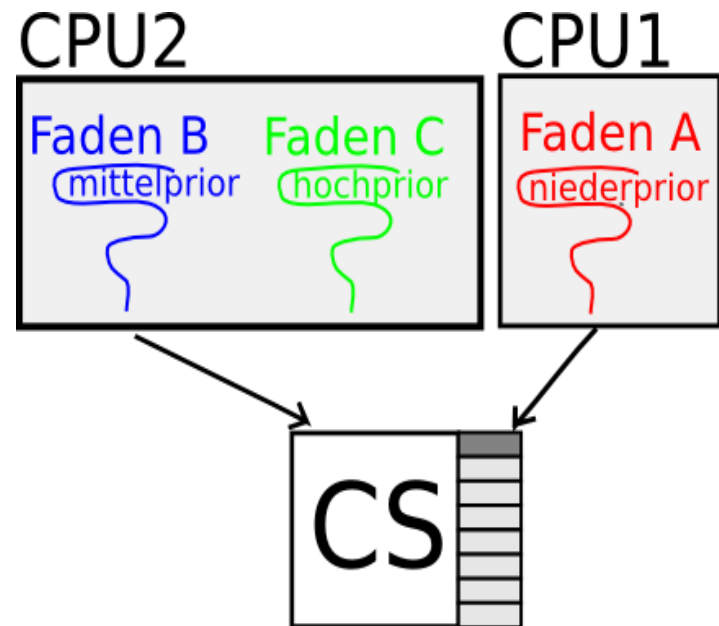
Beispiel:

Verdrängung Faden B durch Faden C

Local Helping: Fortschritt bei Faden B

Remote Helping: kein Fortschritt bei B

→ Local Helping hier besser



Helping mit Wartestapel (6)

Verhalten beim Warten während des Helfens:

- Sleep and Callback (Wenn fertig , bescheid geben)
- Polling (Nachfragen bei jeder Reaktivierung)

Interprozessor-Interrupt bei Variante 1 nötig → teuer

→ Polling trotz Wartezeit besser für die meisten Systeme

Scheduling nach Helping:

Scheduling nach Ende eines Helping-Vorgangs nötig

→ Migration des wartenden Fadens möglich

Single-Server

- Jeder Faden kann Objekte besitzen (Daten, Code)
- Jedes Objekt hat exakt einen Besitzer
- Belegen eines kritischen Abschnitts
 - Besitzer des kritischen Abschnitts ändern
 - Sich selbst verschließen
 - Besitzer der eigenen Objekte nichtmehr änderbar
 - Ausführen des kritischen Abschnitts
 - Sich selbst wieder öffnen
- Wenn kritischer Abschnitt bereits belegt → Helping

Agenda

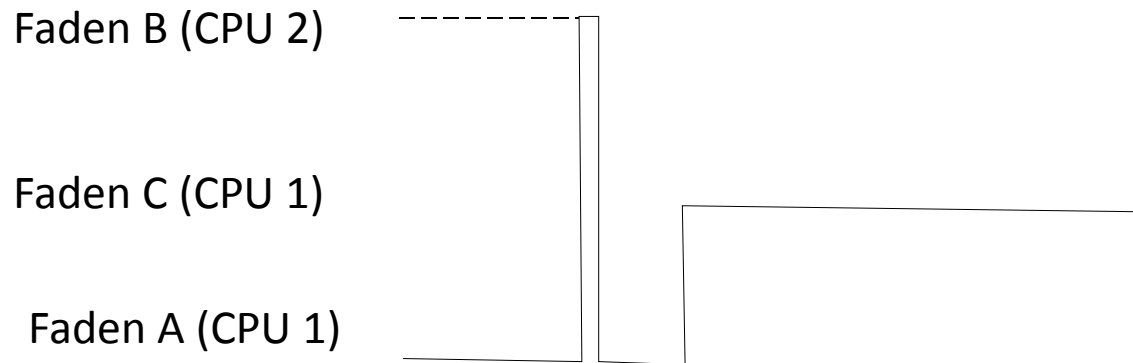
- Grundlagen
 - Der CAS-Befehl
 - Echtzeitproblematik
 - Prioritätenumkehr
- Algorithmen zum Datenaustausch
 - Mit CAS-Befehl
 - Helping mit Wartestapel
 - Single-Server



- **Echtzeitfähigkeit**
- **Multiprozessorfähigkeit**
- **Fazit**

Echtzeitfähigkeit

Vermeidung von Prioritätsverletzungen über
Prozessorgrenzen hinweg



Echtzeiteigenschaften der Algorithmen

- CAS-Befehl:
 - Unbegrenzte Anzahl an Wiederholungen beim Fehlschlagen des CAS-Befehls
 - ➔ Nicht echtzeitfähig
- Helping mit Wartestapel:
 - Prioritätsverletzungen ausgeschlossen
 - ➔ echtzeitfähig
- Single-Server:
 - Prioritätsverletzungen ausgeschlossen
 - ➔ echtzeitfähig
- Vermeidung des Verhungerns muss auf Anwendungsebene sichergestellt werden

Multiprozessorsysteme

- Prozessorlokale Datenstrukturen schneller
 - Interprozessor Interrupt besonders teuer
 - Speicherzuordnung einiger Prozessoren
- Analyse im Kontext von Echtzeit schwerer
 - echte Parallelität
 - Keine Aussagen mit Hilfe des Schedulings möglich

Multiprozessorkompatibilität

Algorithmen an sich multiprozessorfähig

ABER:

Zumindest keine harte Echtzeitfähigkeit mehr

→ Implementierung aktuelles Forschungsthema

→ Aktuell keine echtzeitfähig Variante

Fazit

- Helping mit Wartestapel und Single-Server echtzeitfähig (zumindest im Uniprozessorfall)
- Variante mit CAS-Befehl für Echtzeitsysteme ungeeignet
- Wartestapel mit Helping und Single-Server:
 - keine nicht-blockierende Synchronisation
 - nicht-blockierend implementiert
- In Zukunft noch Forschungsarbeit nötig
- Problematik nimmt mit steigender Prozessoranzahl zu

Vielen Dank für die Aufmerksamkeit

Fragen?