Nebenläufigkeit bei Betriebssystemen

Ein Einblick in Prozessnebenläufigkeit

Arne Struck

Universität Hamburg

28. Juli 2014

Betriebsysteme und welche?

Embedded System OS

Desktop OS

Server OS

Betriebsysteme und welche?

Embedded System OS

Desktop OS

Server OS

Betriebsysteme und welche?

Embedded System OS

Desktop OS

Server OS

Nebenläufigkeit und welche?

Speichernebenläufigkeit

Nebenläufigkeit in der CPU

Betriebssystemverteilung in Multiprozessorsystemen

Nebenläufigkeit und welche?

Speichernebenläufigkeit

Nebenläufigkeit in der CPU

Betriebssystemverteilung in Multiprozessorsystemen

Nebenläufigkeit und welche?

Speichernebenläufigkeit

Nebenläufigkeit in der CPU

Betriebssystemverteilung in Multiprozessorsystemen

Threads und Prozesse

Prozess

Programminstanz in Ausführung (inklusive der Speicherinhalte)

Thread

Ausführungsstrang eines Prozesses.

Threads und Prozesse

Prozess

Programminstanz in Ausführung (inklusive der Speicherinhalte)

Thread

Ausführungsstrang eines Prozesses.

Scheduling

- **Allgemeines und Definitionen**
- Scheduling-Beispiel: Round Robin
- Stacksysteme
  - First In, First Out (FIFO)
  - Shortest Job First
  - Shortest Remaining Time Next
- interaktive Systeme
  - Round Robin
  - Priorisiertes Scheduling
  - Klassenbasiertes priorisiertes Scheduling
  - Shortest Process Next
  - Fair-Share

Scheduling

- **Allgemeines und Definitionen**
- **Scheduling-Beispiel: Round Robin**
- Stacksysteme
  - First In, First Out (FIFO)
  - Shortest Job First
  - Shortest Remaining Time Next
- interaktive Systeme
  - Round Robin
  - Priorisiertes Scheduling
  - Klassenbasiertes priorisiertes Scheduling
  - Shortest Process Next
  - Fair-Share

Scheduling

- **Allgemeines und Definitionen**
- **Scheduling-Beispiel: Round Robin**
- **Stacksysteme**
  - First In, First Out (FIFO)
  - Shortest Job First
  - Shortest Remaining Time Next
- **interaktive Systeme**
  - Round Robin
  - Priorisiertes Scheduling
  - Klassenbasiertes priorisiertes Scheduling
  - Shortest Process Next
  - Fair-Share

Scheduling

- **Allgemeines und Definitionen**
- **Scheduling-Beispiel: Round Robin**
- **Stacksysteme**
  - First In, First Out (FIFO)
  - Shortest Job First
  - Shortest Remaining Time Next
- **interaktive Systeme**
  - Round Robin
  - Priorisiertes Scheduling
  - Klassenbasiertes priorisiertes Scheduling
  - Shortest Process Next
  - Fair-Share

Scheduling

- **Echtzeitscheduling**
- Definitionen
- Deadline-Problematik
- Strategien
  - Rate Monotonic Scheduling
  - Earliest Deadline First
- Präemptives Scheduling
- ASAP

Scheduling

- **Echtzeitscheduling**
- Definitionen
- Deadline-Problematik
- Strategien
  - Rate Monotonic Scheduling
  - Earliest Deadline First
- Präemptives Scheduling
- ASAP

Scheduling

- **Echtzeitscheduling**
- Definitionen
- Deadline-Problematik
- Strategien
  - Rate Monotonic Scheduling
  - Earliest Deadline First
- Präemptives Scheduling
- ASAP

Scheduling

- **Echtzeitscheduling**
- Definitionen
- Deadline-Problematik
- Strategien
  - Rate Monotonic Scheduling
  - Earliest Deadline First
- Präemptives Scheduling
- ASAP

Scheduling

- **Echtzeitscheduling**
- Definitionen
- Deadline-Problematik
- Strategien
  - Rate Monotonic Scheduling
  - Earliest Deadline First
- Präemptives Scheduling
- ASAP

Scheduling

- **Echtzeitscheduling**
- Definitionen
- Deadline-Problematik
- Strategien
  - Rate Monotonic Scheduling
  - Earliest Deadline First
- **Präemptives Scheduling**
- ASAP

Scheduling

- **Echtzeitscheduling**
- Definitionen
- Deadline-Problematik
- Strategien
  - Rate Monotonic Scheduling
  - Earliest Deadline First
- **Präemptives Scheduling**
- ASAP

Multiprozessorsysteme

- **Betriebssystemeaufteilung**
  - 1-Kern-1-System-Lösung
  - Master-Slave-Lösung
  - Symmetrische Lösung
- Multiprozessor-Scheduling

Multiprozessorsysteme

- **Betriebssystemeaufteilung**
  - 1-Kern-1-System-Lösung
  - Master-Slave-Lösung
  - Symmetrische Lösung
- Multiprozessor-Scheduling

Multiprozessorsysteme

- **Betriebssystemeaufteilung**
  - 1-Kern-1-System-Lösung
  - Master-Slave-Lösung
  - Symmetrische Lösung
- Multiprozessor-Scheduling

Multiprozessorsysteme

- **Betriebssystemeaufteilung**
  - 1-Kern-1-System-Lösung
  - Master-Slave-Lösung
  - Symmetrische Lösung
- **Multiprozessor-Scheduling**

Scheduling - Definitionen

Scheduling

Aufteilung von CPU-Zeit auf um die Ressource konkurrierende Prozesse oder Threads

Scheduling-Strategie

Verfahren nachdem ein Scheduling vorgenommen wird

Verdrängend und nicht Verdrängend

nicht Verdrängend: Sobald ein Prozess die Priorität zugesprochen bekommt, arbeitet er durch
Verdrängend: Sobald ein Prozess nach Strategie höherer Priorität erscheint, wird der arbeitende Prozess ersetzt

Scheduling - Definitionen

Scheduling

Aufteilung von CPU-Zeit auf um die Ressource konkurrierende Prozesse oder Threads

Scheduling-Strategie

Verfahren nachdem ein Scheduling vorgenommen wird

Verdrängend und nicht Verdrängend

nicht Verdrängend: Sobald ein Prozess die Priorität zugesprochen bekommt, arbeitet er durch
Verdrängend: Sobald ein Prozess nach Strategie höherer Priorität erscheint, wird der arbeitende Prozess ersetzt

Scheduling - Definitionen

Scheduling

Aufteilung von CPU-Zeit auf um die Ressource konkurrierende Prozesse oder Threads

Scheduling-Strategie

Verfahren nachdem ein Scheduling vorgenommen wird

Verdrängend und nicht Verdrängend

nicht Verdrängend: Sobald ein Prozess die Priorität zugesprochen bekommt, arbeitet er durch
Verdrängend: Sobald ein Prozess nach Strategie höherer Priorität erscheint, wird der arbeitende Prozess ersetzt

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

Strategie-Ziele

- **Allgemein**
  - Ressource optimal ausnutzen (Balance)
  - Keinen Prozess benachteiligen (Fairness)
- **Stacksysteme**
  - Maximierung des Durchsatzes
  - Minimierung der Durchlaufzeit
  - Maximierung der CPU-Auslastung
- **Interaktive Systeme**
  - Reduktion der Antwortzeit
  - Proportionalität gewährleisten
- **Echtzeitsysteme**
  - Deadlines einhalten
  - Vorhersagbarkeit

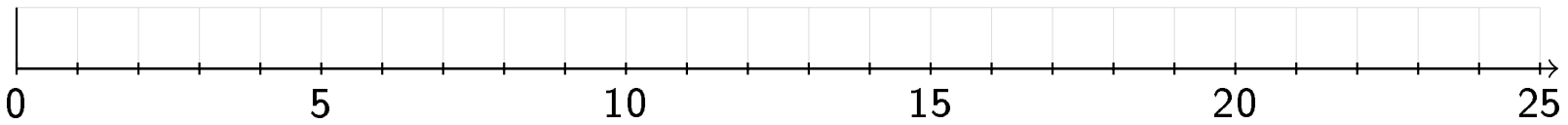
Beispiel: Round Robin

Auftrag	A_1	A_2	A_3	A_4
Ankunftszeit	0	4	5	8
Bedienzeitanforderung	6	4	2	6

Zeitscheibengröße: $\Delta t = 2$

Queue_{t0}:$[A_1]$

Queue_{t2}:[]

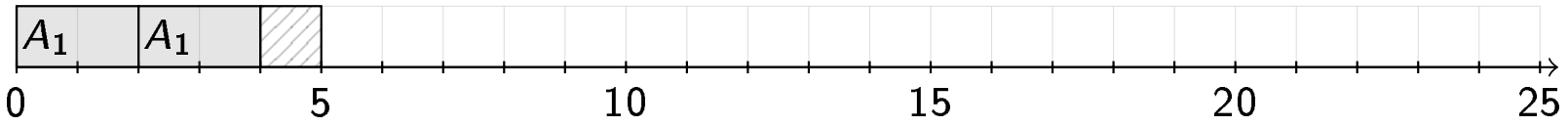


Beispiel: Round Robin

Auftrag	A_1	A_2	A_3	A_4
Ankunftszeit	1	4	5	8
Bedienzeitanforderung	6	4	2	6

Zeitscheibengröße: $\Delta t = 2$

Queue$_{t4}$:$[A_2]$

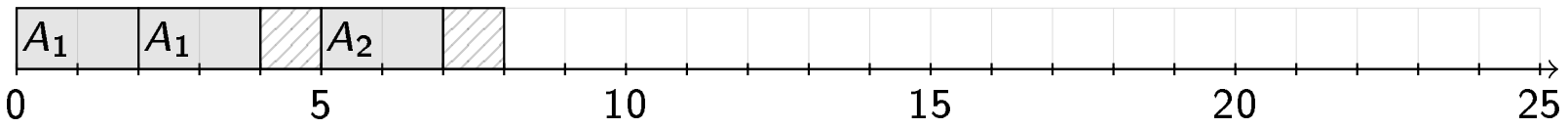


Beispiel: Round Robin

Auftrag	A_1	A_2	A_3	A_4
Ankunftszeit	1	4	5	8
Bedienzeitanforderung	6	4	2	6

Zeitscheibengröße: $\Delta t = 2$

Queue$_{t7}$:$[A_1, A_3]$

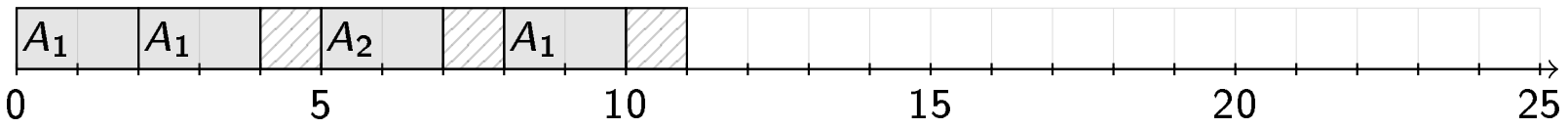


Beispiel: Round Robin

Auftrag	A_1	A_2	A_3	A_4
Ankunftszeit	1	4	5	8
Bedienzeitanforderung	6	4	2	6

Zeitscheibengröße: $\Delta t = 2$

Queue$_{t10}$:$[A_3, A_2, A_4]$

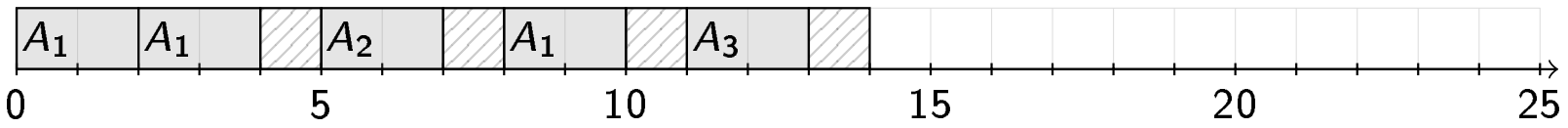


Beispiel: Round Robin

Auftrag	A_1	A_2	A_3	A_4
Ankunftszeit	1	4	5	8
Bedienzeitanforderung	6	4	2	6

Zeitscheibengröße: $\Delta t = 2$

Queue$_{t13}$:$[A_2, A_4]$

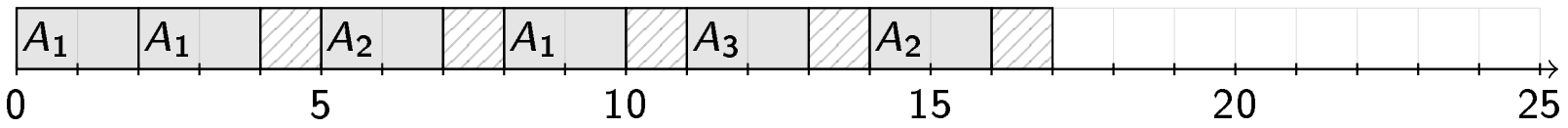


Beispiel: Round Robin

Auftrag	A_1	A_2	A_3	A_4
Ankunftszeit	1	4	5	8
Bedienzeitanforderung	6	4	2	6

Zeitscheibengröße: $\Delta t = 2$

Queue_{t16}:$[A_4]$



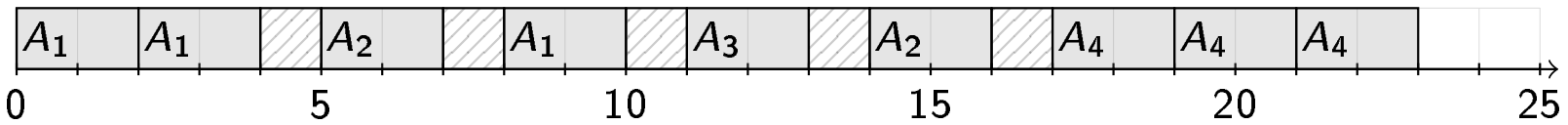
Beispiel: Round Robin

Auftrag	A_1	A_2	A_3	A_4
Ankunftszeit	1	4	5	8
Bedienzeitanforderung	6	4	2	6

Zeitscheibengröße: $\Delta t = 2$

Queue_{t19}:[]

Queue_{t21}:[]



First In, First Out

- nicht verdrängend $\Rightarrow$ geringste Kosten für Prozesswechsel
- einfach vorstellbare Strategie
- Erster rechenbereiter Prozess bekommt als erster Rechenzeit, zweiter als zweites
- durch Queues schnell implementierbar

Shortest Job First

- nicht verdrängende Strategie
- mengenbasiert, Prozess mit geringsten Anforderungen bekommt höchste Priorität
- Durchsatz nachweislich optimal
- nicht Fair (potentiell "verhungernde" Prozess)

Shortest Remaining Time Next

- verdrängende Variante von Shortest Job First
- berücksichtigt wird die verbleibende Anforderung
- Fairnessprobleme bleiben

Round Robin

- verdrängende Strategie
- CPU-Zeit in Zeitscheiben unterteilt
- Queue-basiert
- implizite Annahme: alle Prozesse gleich wichtig

Round Robin

- verdrängende Strategie
- CPU-Zeit in Zeitscheiben unterteilt
- Queue-basiert
- implizite Annahme: alle Prozesse gleich wichtig

Round Robin

- verdrängende Strategie
- CPU-Zeit in Zeitscheiben unterteilt
- Queue-basiert
- implizite Annahme: alle Prozesse gleich wichtig

Round Robin

- verdrängende Strategie
- CPU-Zeit in Zeitscheiben unterteilt
- Queue-basiert
- implizite Annahme: alle Prozesse gleich wichtig

Round Robin

- verdrängende Strategie
- CPU-Zeit in Zeitscheiben unterteilt
- Queue-basiert
- implizite Annahme: alle Prozesse gleich wichtig

Priorisiertes Scheduling

- verdrängende Strategien
- Sammelbezeichnung für bestimmte Strategien
- Idee: Jeder rechenbereite Prozess bekommt eine Prioritätswertung
- Für Interaktivität: dynamisches Anpassen der Priorität

Prioritätsklassen

- Prioritäten errechnen teuer
- Zusammenfassen von Prozessen zu Klassen
- innerhalb der Klassen Round Robin
- Anpassen der Prioritäten, verhindert "verhungern" der unteren Klassen

Shortest Process Next

- Shortest Job First für interaktive Systeme
- Laufzeitinformationen heuristisch ermittelt

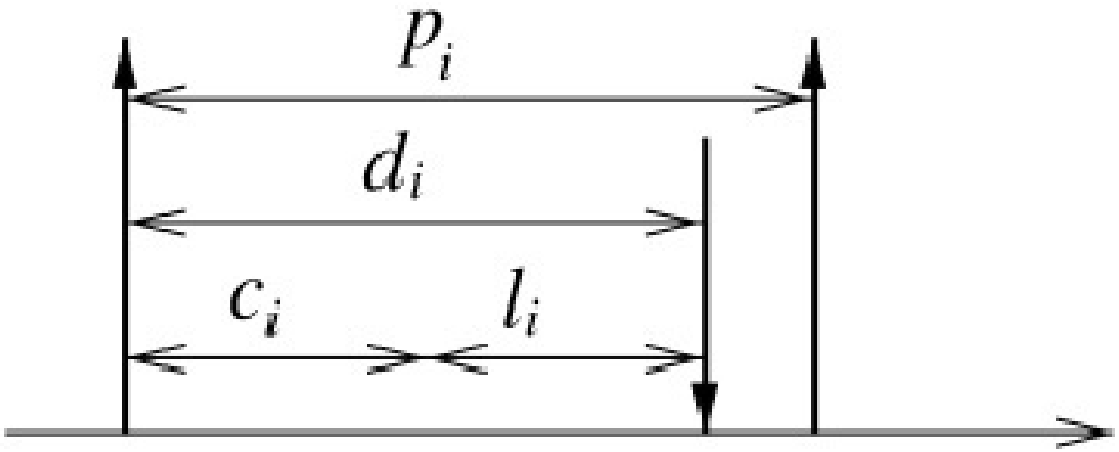
$\Rightarrow$ Profiler benötigt

Fair Share Scheduling

- Betrachtet Nutzerzugehörigkeit
- Stellt Nutzerfairness, nicht Prozessfairness her

Periodizität

Aufgaben spawnen innerhalb bestimmter Zeitabschnitte (Periode, P_i) immer wieder.



[2]

Deadlines

Hard

Harte Deadlineverletzung: Es droht eine Katastrophe, ausgelöst durch Systemabsturz.
$\Rightarrow$ Verletzung nicht tolerabel.

Soft

Weiche Deadlineverletzung: Seltene Fehler verzeihlich.
$\Rightarrow$ Verletzung ist tolerable, sollte aber trotzdem vermieden werden.

Deadlines

Hard

Harte Deadlineverletzung: Es droht eine Katastrophe, ausgelöst durch Systemabsturz.
$\Rightarrow$ Verletzung nicht tolerabel.

Soft

Weiche Deadlineverletzung: Seltene Fehler verzeihlich.
$\Rightarrow$ Verletzung ist tolerable, sollte aber trotzdem vermieden werden.

Schedularisierbarkeit

- Annahme: optimaler Scheduler
- C_i: Zeitanforderung von Task i
- P_i: Periodenlänge von Task i

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Schedularisierbarkeit

- Annahme: optimaler Scheduler
- C_i: Zeitanforderung von Task i
- P_i: Periodenlänge von Task i

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Schedularisierbarkeit

Annahmen:

- Unendlicher Zeitraum
- verschiedene, periodische Tasks
- (implizit) Periodenlänge = Deadline

Schedularisierbarkeit

Schritt 1:

Unterteilung der Unendlichkeit in einzelne, den Periodenlängen angepasste Zeitschritte $\Rightarrow \quad Ch = kgv(P_i)$

Schritt 2:

Berechnung von $a(i) = C_i \cdot \frac{Ch}{P_i}$ für alle Aufgaben

Schritt 3:

Überprüfen: $\sum_{i=1}^{n} a(i) \leq Ch | n =$ Anzahl der Prozesse

Schritt 4 (Überführung):

$$\sum_{i=1}^{n} a(i) \leq Ch \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{a(i)}{Ch} \leq 1$$

$$\Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i \cdot \frac{Ch}{P_i}}{Ch} \leq 1 \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Schedularisierbarkeit

Schritt 1:
Unterteilung der Unendlichkeit in einzelne, den Periodenlängen angepasste Zeitschritte $\Rightarrow \quad Ch = kgv(P_i)$

Schritt 2:
Berechnung von $a(i) = C_i \cdot \frac{Ch}{P_i}$ für alle Aufgaben

Schritt 3:
Überprüfen: $\sum_{i=1}^{n} a(i) \leq Ch | n = \text{Anzahl der Prozesse}$

Schritt 4 (Überführung):

$$\sum_{i=1}^{n} a(i) \leq Ch \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{a(i)}{Ch} \leq 1$$

$$\Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i \cdot \frac{Ch}{P_i}}{Ch} \leq 1 \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Schedularisierbarkeit

Schritt 1:

Unterteilung der Unendlichkeit in einzelne, den Periodenlängen angepasste Zeitschritte $\Rightarrow \quad Ch = kgv(P_i)$

Schritt 2:

Berechnung von $a(i) = C_i \cdot \frac{Ch}{P_i}$ für alle Aufgaben

Schritt 3:

Überprüfen: $\sum_{i=1}^{n} a(i) \leq Ch | n =$ Anzahl der Prozesse

Schritt 4 (Überführung):

$$\sum_{i=1}^{n} a(i) \leq Ch \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{a(i)}{Ch} \leq 1$$

$$\Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i \cdot \frac{Ch}{P_i}}{Ch} \leq 1 \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Schedularisierbarkeit

Schritt 1:

Unterteilung der Unendlichkeit in einzelne, den Periodenlängen angepasste Zeitschritte $\Rightarrow \quad Ch = kgv(P_i)$

Schritt 2:

Berechnung von $a(i) = C_i \cdot \frac{Ch}{P_i}$ für alle Aufgaben

Schritt 3:

Überprüfen: $\sum_{i=1}^{n} a(i) \leq Ch | n =$ Anzahl der Prozesse

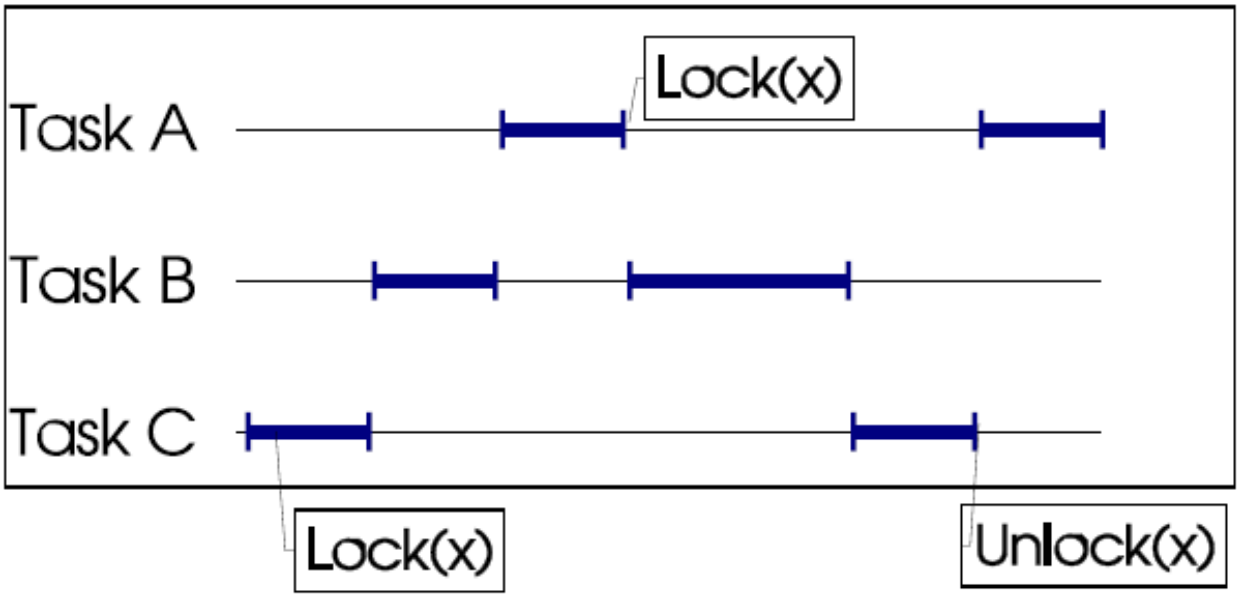
Schritt 4 (Überführung):

$$\sum_{i=1}^{n} a(i) \leq Ch \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{a(i)}{Ch} \leq 1$$

$$\Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i \cdot \frac{Ch}{P_i}}{Ch} \leq 1 \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Prioritätsinversion

- Geteilte Ressource
- führt potentiell zu Problemen (Bsp: A muss vor Darstellungsende terminieren)
- scheinbare Umkehr der Priorität

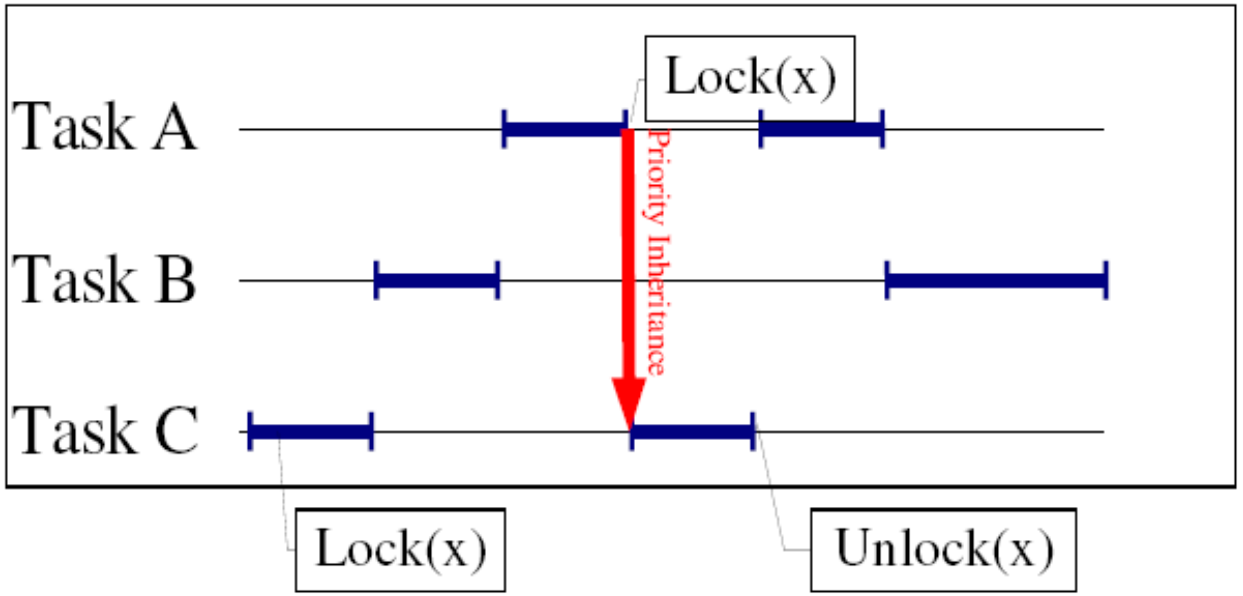


[1]

$A > B > C$ wobei A nach B nach C auftritt

Prioritätsinversion

Lösung: Prioritätsvererbung



[1]

$A > B > C$ wobei A nach B nach C auftritt

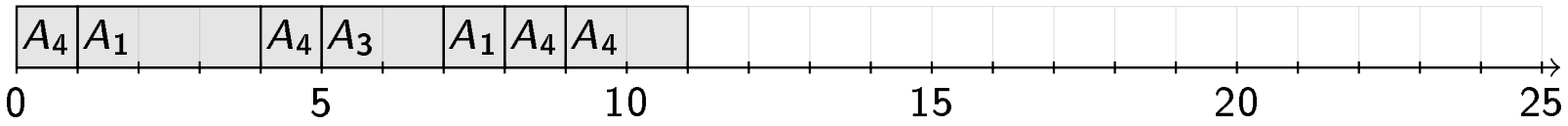
Rate Monotonic Scheduling

- verdrängende Strategie
- statische Prioritäten anhand Periodenlänge
- nicht optimal
- Schedularisierbarkeit: $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n \cdot (2^{\frac{1}{n}} - 1)$ [4]
- nähert sich ln(2) an

RMTS Problembeispiel

Auftrag	A_1	A_2	A_3	A_4
Periodendauer	7	11	9	4
Bedienzeitanforderung	3	1	2	1

$\Rightarrow$ Prioritäten: $A_4 > A_1 > A_3 > A_2$

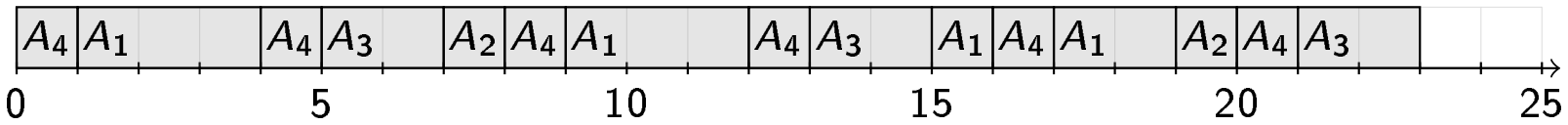


Earliest Deadline First

- verdrängende Strategie
- Prioritätenvergabe anhand der nächsten Deadline
- selbe Bedingung, wie optimaler Scheduler
- Implementation durch Queue
- benötigt nicht zwingend feste Periodenlängen (wie RMTS)
- kann mit aperiodischen Tasks umgehen

EDF schaft RMTS-Problem

Auftrag	A_1	A_2	A_3	A_4
Periodendauer	7	11	9	4
Bedienzeitanforderung	3	1	2	1



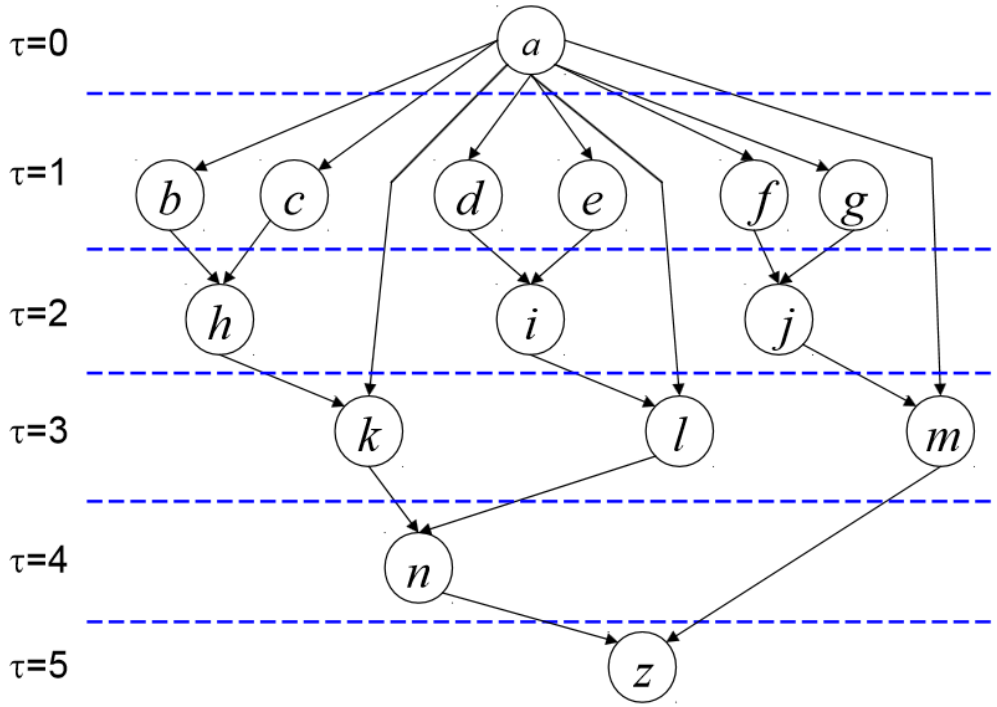
Allgemeines

- Problem: Tasks voneinander abhängig $\Rightarrow$ nicht rechenbereit
- Was passiert bei Deadlines?
- Mit Periodizität der Tasks Schedularisierbarkeit NP-Vollständig [2]

As soon as possible

- Aufbau des Abhängigkeitsgraphen
- Betrachtung der Ursprungsmenge, als berechenbare Elemente
- Beliebiges Scheduling der Menge (da voneinander unabhängig)
- Neue Elemente nun rechenbereit $\Rightarrow$ neue Menge
- Wiederholung bis alle Knoten besucht

As soon as possible Beispiel

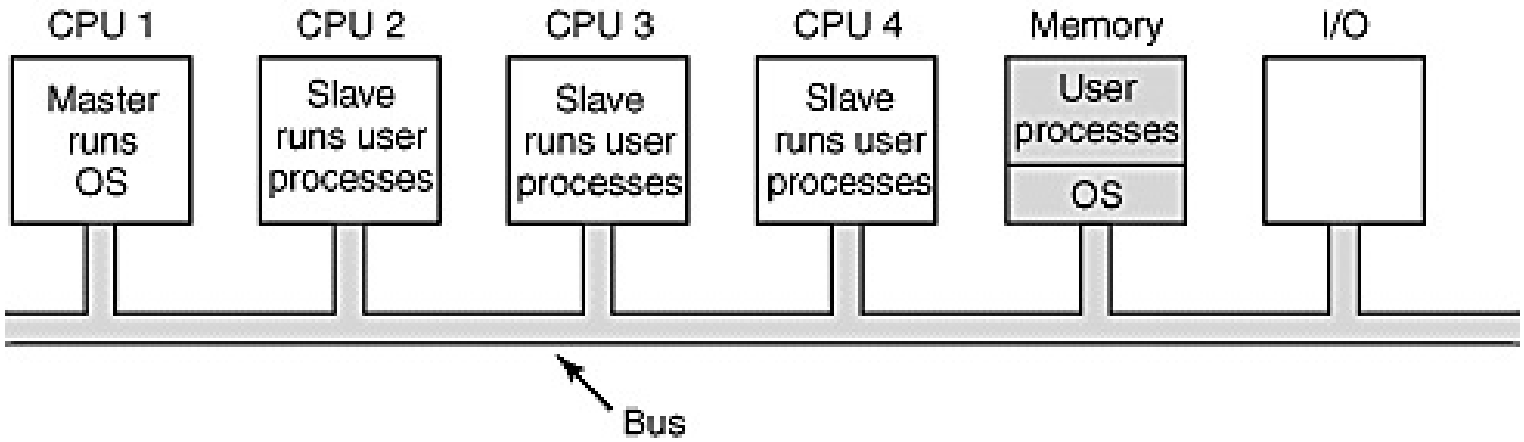


[2]

1-Kern-1-System Modell

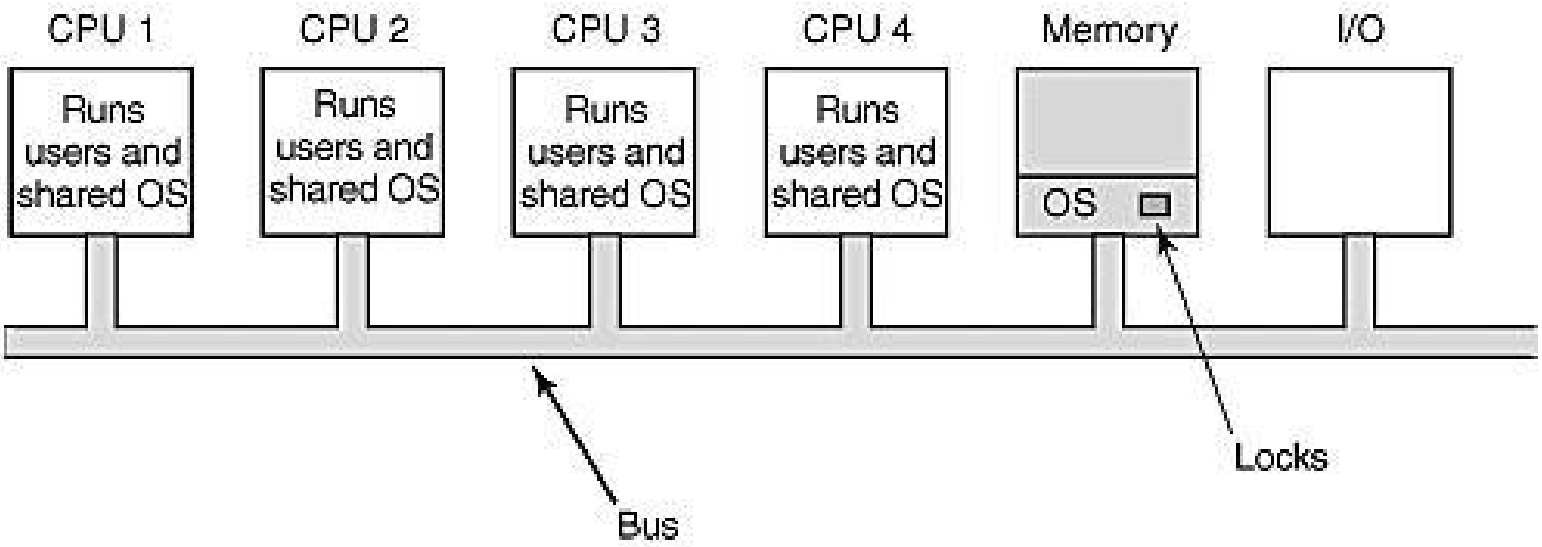
- Jeder Kern hat ein Betriebssystem
- Flüchtiger Speicher wird partitioniert, jeder Kern bekommt seinen Teil
- Restliche Ressourcen geteilt
- Problematisch, da Lastausgleich zwischen Kernen unmöglich
  $\Rightarrow$ nicht mehr verwendeter Ansatz

Master-Slave Modell



[3]

Symmetrisches Modell



[3]

Allgemeines

- Scheduling Strategie bekommt zusätzliche Dimension
- Mögliche Aufteilung von Prozessen auf verschiedene Kerne
- abhängig von Betriebssystemaufteilung:
  - 1-Kern-1-System Modell: Pro Kern ein Scheduler
  - Master-Slave Model: globaler Scheduler
  - Symmetrisches Modell: globaler Scheduler

Schedularisierbarkeit

Optimaler Scheduler:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq MP$$

anstatt:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

wobei MP der Anzahl der Prozessoren entspricht

Fazit

- Betriebsmittelaufteilung und Verwaltung bleibt wichtig, unabhängig von kommenden Systemen
- wenn auch nicht immer unproblematisch.
- Ressourcen der Recheneinheit als Zentraler Baustein wichtig.

$\Rightarrow$ Scheduled fair und anwendungsorientiert.

Quellen I


aicas GmbH.
Jamaicavm 3.4 user documentation: The virtual machine for realtime and embedded systems.
"http://www.aicas.com/jamaica/3.4/doc/html/x4057.html".


Peter Marwedel.
Embedded System Design.
Embedded Systems. Springer US, 3 edition, 2011.


Suresh.
What are the different types of operating systems?
"http://thelinuxdesk.com/2012/09/13/what-are-the-different-types-of-operating-systems/", 09 2012.

Quellen II


A.S. Tanenbaum.
Moderne Betriebssysteme.
Pearson Studium - IT. Pearson Deutschland, 3 edition, 2009.