# Echtzeitsysteme
# Zusammenfassung des Wichtigsten

## Wintersemester 2015/2016

Dr. Peter Ulbrich

Autor:

- Christian Strate

# Inhaltsverzeichnis

# 1 Organisatorisches

Note:

Bei dieser Zusammenfassung handelt es sich um eine inoffizielle Mitschrift von Studenten. Entsprechend sind weder Garantie auf Vollständigkeit noch auf Korrektheit gewährleistet.

- Übungsabnahme alle 14d
- 7 Übungsaufgaben
- 5 ECTS - 20 min Prüfung, 7.5 ECTS - 30 min Prüfung

# 2 Organisation und Einleitung

Eine Übersicht über die mathematischen Abkürzungen finden sich am Ende (table 1)
Echtzeitsysteme:

- Eingebettet in die Umwelt und abhängig von der Hardware
- Gekoppelt an die Realzeit
- Steuerung und Regelung von physikalischen Prozessen

Echtzeitbetrieb:

Verfügbarkeit des Ergebnisses innerhalb einer vorgegebenen Zeitspanne bei stets betriebsbereiter Programme. **Rechtzeitigkeit**

Problematisch bei Laufzeitabschätzungen sind Interrupts. Interne Zeitskala muss nicht mit der Realzeit übereinstimmen. Entsprechend muss eine Abbildung existieren.

Terminüberschreitung

- *Weich - schwach - soft*
  Ergebnis verliert an Wert. Überschreitung ist tolerierbar
- *Fest - stark - firm*
  Ergebnis wird wertlos. Überschreitung tolerierbar, Abbruch der Ausführung der Ergebnisberechnung
- *Hart - strikt - hard*
  Katastrophe. Nicht tolerierbar

Reaktion bei Verletzung

- *Fest*
  Abbrechen des Arbeitsauftrags, planmäßiges Starten des nächsten Auftrags, Transparenz für Anwendung
- *Hart*
  BS löst, eine für die Anwendung intransparente, Ausnahmesituation aus. Anwendung behandelt Ausnahme

Determiniertheit:
: Identische Eingaben ermöglichen unterschiedliche Abläufe mit stets korrektem Resultat.

Determinismus:
: Identische Eingaben implizieren identische Abläufe.

Vorhersagbarkeit:
: Ablauf ist unabhängig von Eingabe und Zustand. Ablauf ist zu jedem Zeitpunkt exakt bestimmbar.

Hartes Echtzeitrechensystem:

- mindestens ein harter Termin
- ausnahmslos deterministisches Laufzeitverhalten

Weiches Echtzeitrechensystem:

- ohne harten Termin

Aktive Zeitspanne:
: Zur Berechnung benötigte Zeit.

Inaktive Zeitspanne:
: Zuteilung von Betriebsmitteln. Für weitere Abschätzung ist die Vorhersagbarkeit dieser Zeitspanne erforderlich.

Ergebnisbehandlung:

- *Rein zyklisch*
- *Meist zyklisch*
  zusätzlich Reaktion auf externe Ereignisse
- *Asynchron-vorhersagbar*
  Zeitdifferenzen haben obere Grenze/bekannte Statistik, stark schwankender Betriebsmittelbedarf
- *Asynchron-nicht vorhersagbar*
  Ausschließlich externe Ereignisse.

Gegenseitige Bedingung von Verlässlichkeit und Rechtzeitigkeit.

# 3 Physikalisches System ↔ Kontrollierendes Rechensystem

## 3.1 Zusammenspiel

Zeit ist keine funktionale Eigenschaft.

Übertragungsfunktion:
Verändert den Zustand des kontrollierten Objekts

Antwortfunktion:
Beschreibung der Zustandsänderung

$d^{object}$:
Zeit bis zum Beginn der Lageänderung. Also bis Kräfte Wirkung entfalten.(Prozessverzögerung). Ist durch Objekt-Trägheit bestimmt.

$d^{rise}$:
Zeitdauer bis nach einsetzen einer Wirkung erneut ein Gleichgewichtszustand erreicht ist. (Einschwingverhalten)

$d^{cpu}$:
Zeitdauer bis zur Ausgabe des Ergebnisses. Randbedingung: $d^{cpu} < d^{sample}$

$\Delta d^{cpu}$:
$|min(d^{cpu}) - max(d^{cpu})|$, es sollte gelten $\Delta d^{cpu} << d^{cpu}$, um eine möglichst konstante und bekannte Verzögerung zu haben. Die Schwankungen nennt man auch jitter

$d^{dead}$:
Zeitintervall zwischen Berechnungsbeginn und spürbarer Reaktion nach Steuerung. $= d^{cpu} + d^{object}$

Um Unterbrechungsbehandlung berechenbar zu machen sind Werte mit fester oberer Schranke notwendig (Herstellerangaben):

- Prozessor mir respektivem Rechensystem
- Echtzeitbetriebssystem
- Echtzeitanwendung

Dies übermittelt einem maximal über die Anwendung die volle Kontrolle

$d^{control}$:

konstanter Zeitabstand zwischen zwei Regelschritten - Faustregel: $d^{sample} < \left(\frac{d^{rise}}{10}\right)$

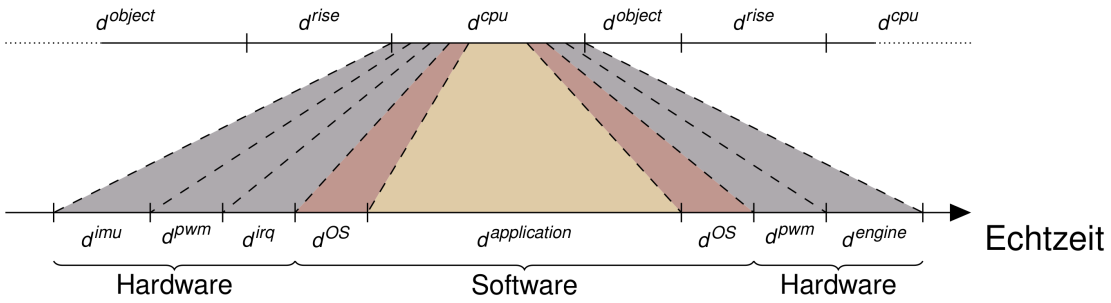



Abbildung 1: Komponenten von $d^{cpu}$

Also die Komponenten:

- Sensoren/Aktoren
- Mikrocontroller
- Betriebssystem
- Anwendung

Termine und zeitliche Rahmen werden maßgeblich von der Objektdynamik definiert.

## 3.2 Echtzeitanwendung

bestimmtes Ereignis:

Auftreten lässt sich als Funktion der physikalischen Zeit beschreiben. Sonst ist ein Ereignis ungewiss

Ereignis-Auslöser:

event trigger:

Ableitung der Ereignisse anhand von physikalsicher Umwelt oder kontrollierenden Rechensystemen.

time trigger:

Ableitung anhand von physikalisch voranschreitender Zeit.

Job:

Schedulbare Arbeitseinheit

Task:
: Menge zusammenhängender Jobs, die zusammen System-Funktionalität anbieten

Ablaufsteuerung:
: Zeitpunkt und Ort der Arbeitsauftrags-Ausführung
Also Abbildung von Arbeit/Arbeitsaufträgen $\mapsto$ Fäden und
Fäden $\mapsto$ Prozessor/Kerne

Vorranggesteuerte Einplanung:
: Vergabe der Prioritäten erfolgt häufig off-line, wohingegen die Einplanung on-line erfolgt. Die on-line Einplanung erzwingt explizite Überprüfung der Termin-Einhaltung.

Einfache Aufgaben = simple task:
: Kommen ohne Synchronisationspunkt aus. Die Ausführung ist blockadefrei und unabhängig von Fortschritten anderer Aufgaben.

Komplexe Aufgaben = complex task:
: Mit Synchronisationspunkten. Benötigt Betriebsmittel. Fortschritt abhängig von anderen Aufgaben

Latenz = latency:
: Zeitdauer zwischen Auslösezeit und Abarbeitungsbeginn

Antwortzeit = response time:
: Zeitdauer zwischen Auslösung und Arbeitsauftrag-Terminierung

Schlupfzeit = slack time:
: Zeitdauer zwischen voraussichtlichem Ausführungsende und Termin eines sich in der Bearbeitung befindlichen Arbeitsauftrags

Gültigkeit eines Ablaufplans = valid:

- Zu jedem Zeitpunkt existiert maximal ein Arbeitsauftrag pro CPU
- Jeder separate Arbeitsauftrag läuft für jeden Zeitpunkt auf maximal einer CPU
- Auslösezeitpunkt notwendig vor Einplanung
- Zuteilung der tatsächlichen/maximalen Ausführungszeit (nicht zwingend rechtzeitig)
- Erfüllung sämtlicher Abhängigkeiten

Zulässigkeit eines Ablaufplans = feasible:
Ist Gültig. Alle Arbeitsaufträge werden termingerecht eingeplant

Optimalier Einplanungsalgorithmus = optimal:
findet für eine Menge Aufgaben einer gewissen Klasse von Aufgaben, einen zulässigen Ablaufplan, sofern existent.

## 3.3 Zeitliche Analyse von Echtzeitanwendungen

- Best Case Execution Time (BCET)
- Worst Case Execution Time (WCET)
  Wird bestimmt durch Summation der Ausführungszeiten des längsten Pfades (erfordert Aufzählung sämtlicher Pfade)
  Entsprechende Abbildung auf Flussprobleme als Vereinfachung
- Best Observed Execution Time (BOET)
- Worst Observed Execution Time (WOET)
- Statische WCET-Analyse liefert Schranken vgl. fig. 2
- Bestimmung der exakten Ausführungszeit ist aufgrund von Prozessor-Magie (fast) unmöglich
- Entsprechend zur Laufzeit dynamische Beobachtung der Ausführungszeiten
- kann aber allenfalls eine Ergänzung zur statischen WCET-Analyse sein, da häufig das gesamte System gemessen wird.
  Gewählte Testdaten führen auch nicht zwingend zum längsten Pfad.
  Insbesondere im Hinblick auf Ausnahmebehandlung
- Analyse ist vollständig (sound), falls Upper Bound $\geq$ WCET

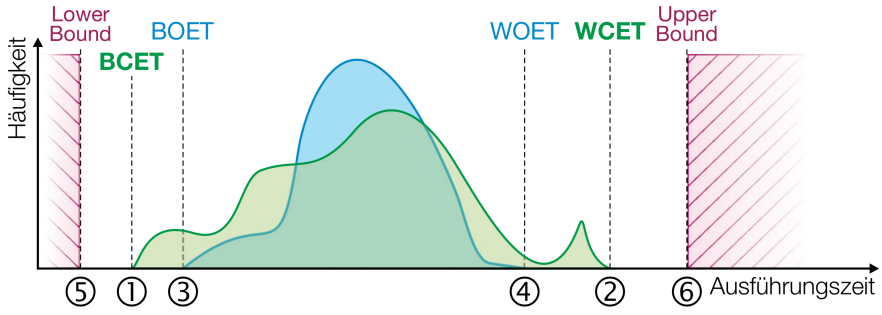



Abbildung 2: Statische WCET-Analyse.
Der Ausschlag zwischen 4 und 2 ist typisch für Fehlerbehandlung

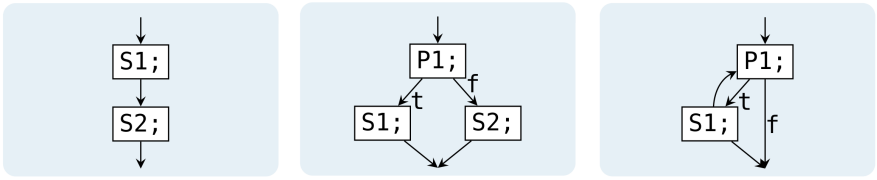



Abbildung 3: Timing Scheme - vlnr. Sequenz, Verzweigung, Schleife

Beispiele für Vereinfachung der Pfadsemantik: Timing Schema und IPET

Timing Schema:

Ist unabhängig von der verwendeten Sprache.
Summiert rekursiv (bootom-up) sämtliche Kanten (Abzweigungen) der Pfade zusammen.
Unterschieden wird hier zwischen drei verschiedenen Typen. vgl. fig. 3

- Sequenzen, `S(1); S2();` Schlicht Addition der beiden $e_{seq} = e_{S1} + e_{S2}$
- Verzweigung, `if(P1()) S1(); else S2();` Addition des teureren Pfades $e_{cond} = e_{P1} + \max(e_{S1}, e_{S2})$
- Schleifen, `while(P1())S1();` Addition des multiplizierten Wertes $e_{loop} = e_{p1} + n(e_{P1} + e_{S1})$
- Vorteile
  - Skaliert gut mit Programmgröße
  - Einfaches Verfahren
- Nachteile
  - Durch das Zusammenfassen (Aggregation) entsteht Informationsverlust
    Nicht Berücksichtigbarkeit von sich ausschließenden Zweigen

  - unerreichbare Pfade werden nicht ausgeschlossen. Entstehende Überapproximation

Implicit Path Enumeration Technique (IPET):

Erfordert Vereinfachung.
Überführt Kontrollflusgraphen in ein ganzzahliges, lineares Optimierungsproblem (ILP) und betrachtet das System erneut Global:

1. Zeitanalysegraph (s.u.) aus dem Kontrollflussgraphen bestimmen
2. Abbildung auf lineares Optimierungsproblem
3. Annotation von Flussrestriktionen
4. Lösung des Optimierungsproblems

Zeitanalysegraph:

- Gerichteter Graph mit Knotenmenge $\mathcal{V} = \{V_i\}$ und Kantenmenge $\mathcal{E} = \{E_i\}$
- Enthält genau eine Quelle und eine Senke
- Kosten werden an den Kanten angegeben, weshalb Dummy-Kanten $d_i$ notwendig sind um die Kosten der Bedingungsprüfungen darstellen zu können. vgl. fig. 4.
- Ursprung ist der Kontrollflussgraph aus dem für jeden Knoten eine Kante im Zeitanalysegraph angelegt wird.
- Zirkulation ist eine Abbildung $f : \mathcal{E} \mapsto \mathcal{R}$, die den Fluss erhält vgl. fig. 5
- Kanten wird die Ausführungsanzahl $f_i$ als Fluss zugeordnet
  Wodurch eine Rückkehrkante $E_e, f_e = 1$ notwendig wird
- Vorteile
  - komplexe Flussrestriktionen möglich
  - Definition von Nebenbedingung ist leicht
  - Verfügbarkeit vieler Werkzeuge
- Nachteile
  - Lösen eines ILP ist meist NP-hart
  - Flussrestriktion hilft nicht bei Ausführungsreihenfolgen

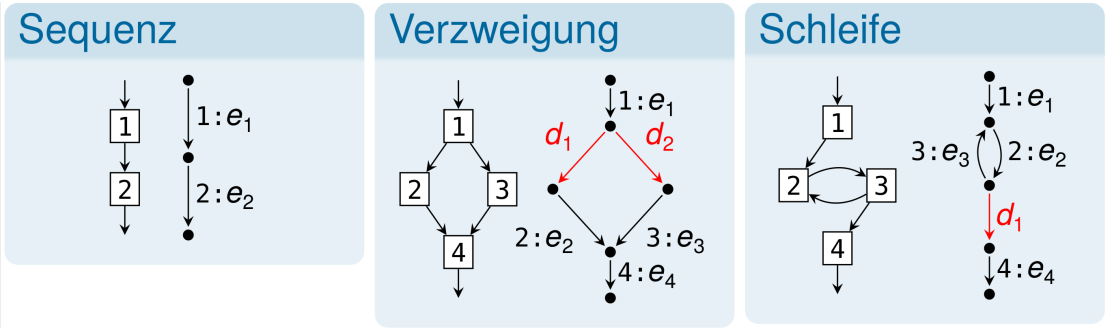



Abbildung 4: Zeitanalysegraph mit Dummy-Kanten $d_i$ für die Bedingungsprüfungen

Irgendwie erscheint mir die Übersetzung der Schleife grad als falsch. Das linke ist eine do-while Schleife bei der 2 und 3 den Rumpf bilden (2 und 3 werden zwingend ausgeführt). Es scheint fast so als e2 gleich die Kosten für die Knoten 2 und 3, dann müsste e3 doch aber eine Dummy-Kante für die Fallunterscheidung sein.

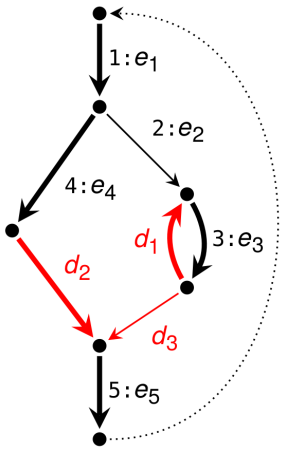



Abbildung 5: Zeitanalysegraph Zirkulation

Garantie der Zirkulation durch

$$f_1 = f_2 + f_4$$

‚gültige Zirkulation:

$$\{E_1, E_4, d_2, E_5, E_e\} \cup \{E_3, d_1\}$$

, aber keine gültige Abarbeitung.
Behebung durch Flussrestriktion

$$f_3 \leq 5 f_2$$

- als Nebenbedingung

Hardware-Analyse:

- Integration von Pfad- und Cache-Analyse
  Beachtung von Pipeline und Caches und Integration derer in das Optimierungsproblem
- Separate Pfad- und Cache-Analyse
  Anschließende Berücksichtigung der Cache-Analyse, die mithilfe einer Datenflussanalyse getroffen wurde
- Cache-Analyse - Unterscheidung von Annahmen bzgl. vorliegen eines Datums im Cache
  - must: garantiert im Cache
  - may: vielleicht im Cache
  - persistent: verbleibt im Cache

# 4 Periodisches Echtzeitsysteme

## 4.1 Abfertigung periodischer Echtzeitsysteme

Periodische Aufgaben $T_i$:

ist eine in regelmäßig Zeitintervallen wiederkehrende Abfolge von Arbeitsaufträgen $(J_{i,j})$ mit vorgegebenen zeitlichen Eigenschaften.

$T_i = (p_i, e_i, D_i, \phi_i)$

$J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$

| | | | |
|---|---|---|---|
| $p_i$ | Periode | $r_i$ | Auslösezeitpunkt |
| $e_i$ | WCET | | |
| $D_i$ | Relativer Termin | $d_i$ | Absoluter Termin |
| $\phi_i$ | Phase | | |

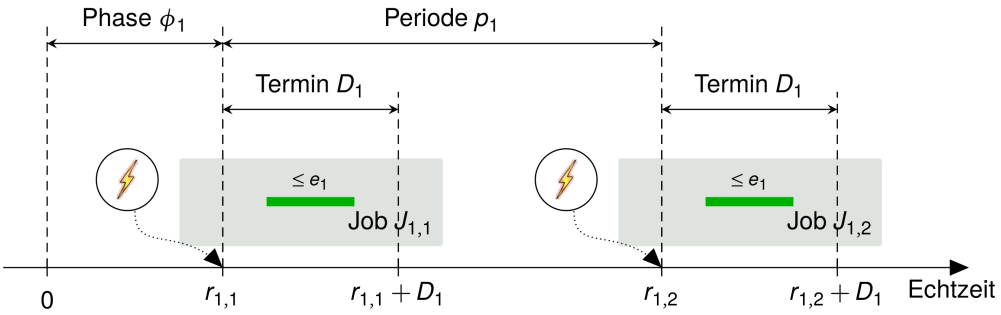
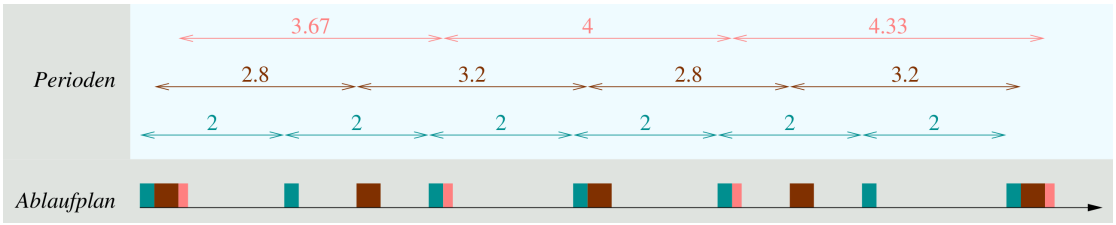
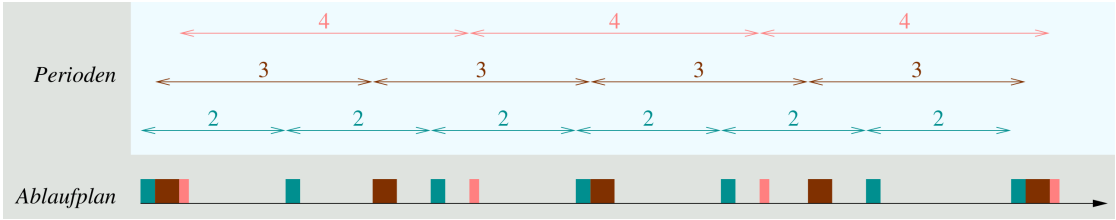



Abbildung 6: periodische Aufgabe. - Abkürzungen: table 1
Auflösung einer Überlappung durch Phasenverschiebung




(a) Schwankende periodische Aufgaben




(b) Periodisierte periodische Aufgaben

Abbildung 7: Entfernung von Schwankung periodischer Aufgaben

Hyperperiode $H$:
: Periode in der ALLE Unter-Perioden auftreten. Hierbei liegt häufig ein Phasenversatz vor, der Einlastungszeiten schwanken lässt falls zeitgleich mehrere Arbeitsaufträge anstehen.

- Mathematische Ansätze zur Analyse gehen häufig mit unrealistischen Einschränkungen einher
  1. Alle Aufgaben sind periodisch
  2. Alle Arbeitsaufträge können an ihren Auslösezeitpunkten eingeplant und ausgeführt werden
  3. Termine und Perioden sind identisch
  4. Kein Arbeitsauftrag gibt die Kontrolle über den Prozessor ab
  5. Alle Aufgaben sind unabhängig
  6. Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar
  7. Alle Aufgaben verhalten sich voll-präemptiv
- Treten Konflikte auf und sind verfügbare Slots zu klein, so müssen einige Aufgaben womöglich händisch in mehrere Teile untergliedert werden
- Bei einem statischen Ablaufplan führen unvorhergesehene Ausnahmen zur Terminüberschreitung
- Bei einer statischen Ablauftabelle entspricht jeder Tabelleneintrag einer Einplanungsentscheidung zu einem Zeitpunkt
- Entscheidungszeitpunkte haben stets in Taktzahl vorzuliegen
- Im Abfragebetrieb muss die WCET die Fehlerbehandlung mit einschließen
- Beim Unterbrechenden Zeitgeber hingegen erfolgt der Abbruch des Arbeitsauftrags als Folge einer Zeitgeberunterbrechung. Im Bezug auf die WCET ist dies ein Ausnahmefall.
- Batch Processing führt einen Job nach dem anderen aus, wodurch lange Jobs kürzere verzögern
- Bei Stapelbasierte Abarbeitung verdrängt der eingelastete Job den aktuell ausgeführten Job. Hierbei ist die Laufzeitprotokollierung elementar.

DH auch lange Jobs verdrängen die kürzeren? Dann werden kurze auch dann verzögert, wenn ein langer kurz nach ihnen dispatched wurde?

- Ereignisorientierte Einplanung. Ereignisse besitzen Prioritäten, keine Endlosschleife in der Anwendung. Starten von run-to-completion Fäden.
- Feste Priorität vs dynamische Priorität
- Zuweisung zu Prioritäten erfolgt zum Auslösezeitpunkt.
- Auf Jobebene sind Prioritäten fest, auf Taskebene aber variabel. (dynamisch)
- Verdrängbar (preemptable)
  an beliebigen Stellen (fully preemptive), vs.
  an ausgewiesenen Stellen (preemption points)
- Ablaufliste - Aufwand
  Auslösezeitpunkt - Linear
  Auswahlzeitpunkt - Konstant
- Ablauftabelle - Aufwand
  Auslösezeitpunkt - Konstant
  Auswahlzeitpunkt - Linear

## 4.2 Ereignisgesteuerte Ablaufplanung periodischer Echtzeitsysteme

Kriterien der Priotiätsvergabe:

Statisch:

- Rate Monotonic (RM) - Ratenmonoton
  Je kürzer die Periode, desto höher die Priorität. Vermeidet Idlen
  Rate ist $\frac{1}{p_i}$
  Ist optimal für Systeme mit folgenden Eigenschaften:
  - Mathematische Bedingungen (item 1) A1-A7 sind erfüllt
  - Aufgaben sind in Phase (synchron), $\phi_i = 0$
- Deadline Monotonic (DM) - Fristenmonoton
  Je kürzer der relative Termin, desto höher die Priorität. Vermeidet Idlen
  Ist bei variierenden relativen Terminen besser als RM und sinnvoll bei Perioden variabler Länge. $D_i = p_i \Rightarrow DM = RM$
  Ist optimal für Systeme mit folgenden Eigenschaften:
  - Mathematische Bedingungen (item 1) A1,A2,A4-A7 sind erfüllt
  - Für die Termine gilt: $D_i \leq p_i$

Dynamisch:

- Earliest Deadline First (EDF)
  Je früher der Termin, desto höher die Priorität. Vermeidet Idlen. Ist optimal für Systeme mit folgenden Eigenschaften:
  - Mathematische Bedingungen (item 1) A2,A4-A7 sind erfüllt
  - Aufgaben haben beliebige Auslösezeiten
  - Aufgaben besitzen beliebige Termine

  Jeder zulässige Ablaufplan für solche Probleme lässt sich in einen EDF-Ablaufplan umformen

- Latest Release-Time First (LRT)
  Erledigt die Aufgaben möglichst zeitnah an ihrer Deadline. Also es wird von hinten nach vorne eingeplant. Kann Idlen verursachen.

- Least Slack-Time First (LST)

  $$\begin{aligned} slack(J_i,t) &= r_i + D_i - t - maturity(J_i,t) \\ maturity(J_i,t) &= e_i - elapsedtime(J_i,t) \end{aligned}$$

  Vermeidet Idlen

Diese vielen Prioritätsebenen lassen sich im allgemeinen nicht in der Realität abbilden. Die Abbildung auf Systemprioritäten kann, aber muss nicht gleichmäßig erfolgen. Allgemein werden bei gleichzeitiger Laufbereitschaft derjenige Kontrollfluss gewählt, der früher Laufbereit war, da dieser früher eingeplant wurde.

Planbarkeitsanalysen:

- CPU-Auslastung $\rightsquigarrow$ dynamische Prioritäten
  $u_{[t_1,t_2[} = \frac{h_{[t_1,t_2[}}{t_2-t_1}$ der Arbeitsaufträge in dem Interval (jeweils bis zur nächsten Deadline).
  $h_{[t_1,t_2[}$ Rechenzeitbedarf für diese Arbeitsauftragsmenge.
  $u = \max_{0 \leq t1 \leq t2} u_{[t_1,t_2[}$ maximale CPU-Auslastung, wichtig für Zulässigkeit der Aufgabenmenge $T$

- Zeitbedarfsanalyse $\rightsquigarrow$ dynamische Prioritäten
  $h_{[t_1,t_2[} = \sum_{t_1 \leq r_k, D_k \leq t_2} e_k$ Rechenzeitbedarf für das Interval

- Antwortzeitanalyse $\rightsquigarrow$ statische Prioritäten
  Schwierig die maximale Antwortzeit zu bestimmen.
  Antwortzeit $\omega_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k; 0 < t \leq p_i$
  Es wird für Alle Aufgaben Schrittweise die jeweils nächste Deadline (nicht zwingend die eigene) betrachtet und inkrementell getestet ob die eigene Ausführung bis dorthin möglich ist. Falls nein, probiere die nächste, sofern diese noch vor der eigenen Deadline ist. Falls ja, plane dich selbst ein.
  Kritischer Zeitpunkt ist der Punkt mit maximaler Antwortzeit.

- Simulation $\rightsquigarrow$ statische Prioritäten, Beobachten, ob Deadline verfehlt wird.
  Analyse schwierig, Planung einfach

Kritischer Zeitpunkt:

> Ein Punkt, zu dem ein zu diesem Zeitpunkt eingelasteter Job die maximale Antwortzeit benötigt. Ist für Systeme mit dynamischen Prioritäten nicht bestimmbar.

Systeme, die den Bedingungen A1-A7 genügen sind mit polynomiellem Aufwand analysierbar. Lockerung der Regeln zerstört diesen Aufwand und beeinflusst die Zulässigkeitstests.

- A3 $\rightsquigarrow$ NP-hart
- A4 $\rightsquigarrow$ NP-hart
- A5 $\rightsquigarrow$ NP-hart
- A7 $\rightsquigarrow$ NP-hart

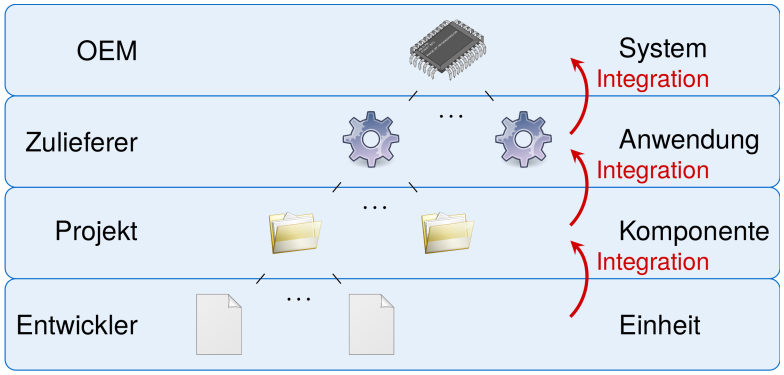



Abbildung 8: Ablaufplanung - Integration - Bootom-Up
Jeder Intergrationsschritt beeinflusst die zeitlichen Parameter
- Bündelung von Softwareeinheiten zu Komponenten
- Implementierung von Arbeitsaufträgen in Anwendungen durch Komponenten
- Einplanung der Arbeitsaufträge in statischer Ablauftabelle

## 4.3 Zeitgesteuerte Ablaufplanung periodischer Echtzeitsysteme

Ablaufplanung:

- Manuelle Ablaufplanbestimmung ist Prüfungsrelevant
- Die Ablaufplanung ist der letzte Schritt der Systemerstellung
- Nachträgliche Änderungen (fig. 8) erfordern erheblichen Aufwand (WCET-Bestimmung etc.), da teils viele Entwicklungsschritte wiederholt werden müssen
- Deswegen wäre es praktisch zeitliches Verhalten von Software-Einheiten und Komponenten vorab zu spezifizieren, was genau das Folgende tut

Spezifikation - fig. 9:

- Dies ist selten ein möglicher Ansatz, da für die globalen Zuweisungen Vorabwissen erforderlich ist, welches sich durch iterative Prozesse oder Erfahrung bestimmen lässt.
- Erfordert eine lange Anpassungs-Sequenz, bis gültige Pläne entstehen

Rahmenlänge $f$:

- Unterteilt die Echtzeitachse in Intervalle fester Länge $f$
- Die Beschränkung von Einplanungsentscheidungen nur zu bestimmten Zeitpunkten erleichtert erheblich die Analysierbarkeit

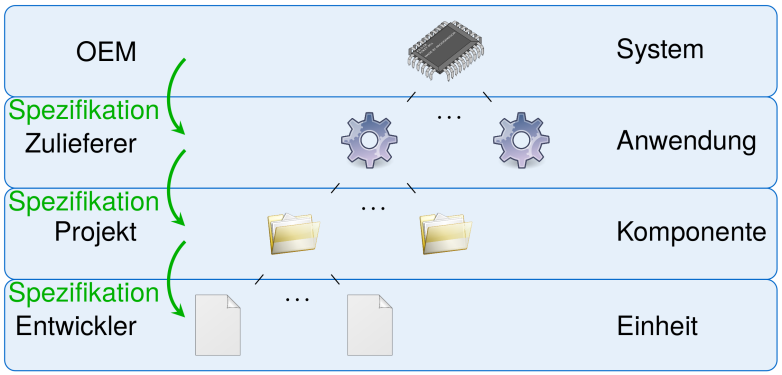



Abbildung 9: Spezifikation - Top-Down
- OEM weist den Anwendungen Zeitschlitze im Ablaufplan zu
- Anwendungen verteilen Rechenzeit auf Softwarekomponenten
- Komponenten und Einheiten müssen die Rechenzeit einhalten

- Einplanungsentscheidungen erfolgen lediglich am Rahmenanfang
- Auslösung der Aufträge erfolgt am Rahmenanfang
- Verdrängung innerhalb eines Rahmens ist nicht möglich $\Rightarrow$ Phase einer periodischen Aufgabe ist ein Vielfaches von $f$
- Dies erleichtert WCET-Analyse. Rahmenzeit sollte gewählt werden mit $\geq \max(WCET)$, um Verdrängung von Jobs zu vermeiden, also jeder Job innerhalb einer Rahmenlänge sicher fertig wird.
- $f$ teilt die Hyperperiode $H$, so dass die zyklische Ausführung möglich ist:
  $\left\lfloor \frac{p_i}{f} \right\rfloor - \frac{p_i}{f} = 0$
- Um die Terminüberwachung zu vereinfachen ist das rechtzeitige Auslösen notwendig: $f \leq p_i, 1 \leq i \leq n \Rightarrow \boldsymbol{2f - \gcd(p_i, f) \leq D_i}$
  $\Rightarrow 2f - 1 \leq D_i, f \leq D_i$
  Es soll also jedesmal mindestens ein Rahmen zwischen Auslösung und Termin liegen
- $\Rightarrow f \leq \min\{p_i | p_i \in \Phi\}$
- $\boldsymbol{\Rightarrow \max(WCET) \leq f \leq \min\{p_i | p_i \in \Phi\}}$
- Falls diese Bedingungen nicht erfüllbar sind, müssen Arbeitsaufträge in Scheiben geschnitten werden, sonst kann dieses Einlastmodell nicht verwendet werden
- Da die Überprüfung der Verletztung nun Rahmenweise erfolgt muss Fehlerbehandlung bei Verletzung für alle Beteiligten erfolgen, da keine Zuweisung möglich ist

Entwurfs- bzw. Ausführungsmodelle (Übung):

- zeitgesteuert
- ereignisgesteuert
- Rahmenmodell

nicht wie wir dachten:

- ereignisorientiert Ausführungsmodell (run-to-completion)
- prozessorientiertes Ausführungsmodell (Endlosschleife, Anwendungsprogrammierer muss für Deadline-Überprüfung sorgen)
- Statische Ablaufplanung sind umfangreich und Hyperperioden werden schnell groß
- Algorithmische Ablaufplanung ist schwierig, meist NP-hart (Heuristiken notwendig, nicht zu pessimistische WCET-Annahmen treffen)

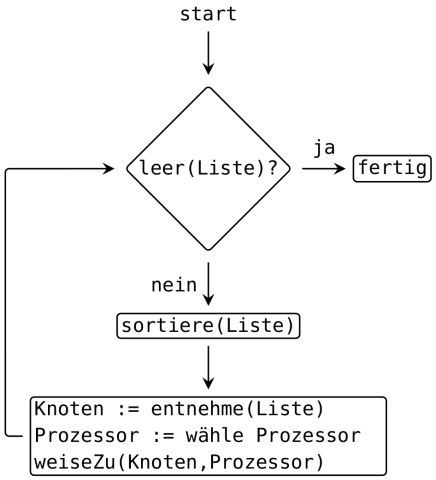



Abbildung 10: List-Scheduling, Heuristik
Falls die Zuweisen fehlschlägt, muss eine Umsortierung erfolgen
Sortiert wird mittels eines "directed acyclic graph" (DAG), bei dem die Knotengewichte Berechnungen und Kantengewichte Kommunikation repräsentieren
t-Level: Längster Pfad von Wurzel zum betrachteten Knoten (dynamisch Variabel - Kommunikation verschwindet u.U.)
b-Level: längster Pfad vom Endknoten zum betrachteten Knoten (i.d.R. konstant)

Beispiel-Algorithmen:

- Highest Level First Estimated Times (HLFET)
  ignoriert Kommunikationskosten, verwendet ausschließlich b-Level
- Insertion Scheduling Heuristic (ISH)
  Selbes Sortierkriterium, wie HLFET.
  Mögliche Erzeugung von untätigen Intervallen.
- Dynamic Level Scheduling (DLS)
  Sortiert nach Dynamic Level: |b-Level − frühester Prozessor-Startpunkt|

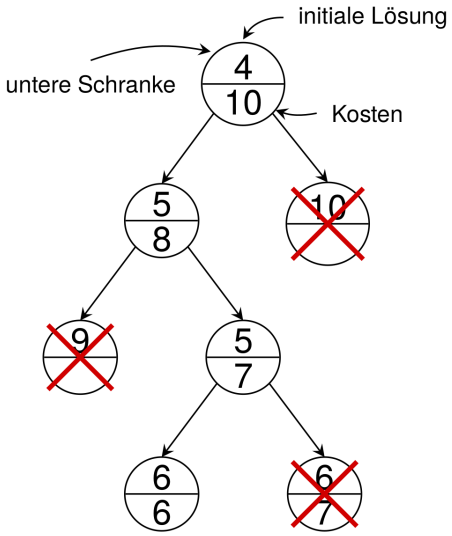



Abbildung 11: Branch and Bound
Die Knoten unterhalb des initialen sind hergeleitete Verbesserungen, die Schritt für Schritt eliminiert (Bound) und im Branch-Schritt mit initialen Lösungen versehen werden:
$10 \geq 10$ nicht besser als bisherige Kosten
$9 \geq 8$ nicht besser als bisherige Kosten
$7 \geq 6$ tatsächliche Kosten sind im anderen Fall besser
Optimalität wird erreicht indem im Branch-Schritt eine komplette Untersuchung aller möglichen Verbesserungen einer Lösungen erfolgt

Branch and Bound - statische Ablaufplanung - fig. 11:

- initiale Lösungen sind vollständig gültige Ablaufpläne (nicht zwingend zulässig)
- Kosten sind vorhandene Verspätungen (Maximale Terminüberschreitung aller Jobs)
- Untere Schranken durch Vereinfachung des Planungsproblems, ohne Verletzung der Optimalität des Algorithmus
- Verbesserung durch Manipulation des Planungsproblems durch frühere Einplanung der Arbeitsaufträge mit größerer Terminüberschreitung

Algorithmus von Abdelzaher und Shin:

- Bearbeitung von
  - Auslöse-, Ausführungszeiten, Termine
  - (Un)gerichtete Abhängigkeiten
  - Verteilte Systeme und nachrichtenbasierte Kommunikation
- Kosten mittels EDF bestimmten
- untere Schranke: Vereinfachung bis EDF optimal (Entfernung ungerichteter bzw. kernübergreifender Abhängigkeiten)

Statische Ablauftabelle:

- Auch bei seltener Auslösung wird stets mit WCET gerechnet
- Entflechtung der Arbeitsaufträge, indem sich diese nur dann in einer gemeinsamen Ablauftabelle befinden, wenn sie auch zusammen ausgelöst werden können

Rekonfiguration des Aufgabensystems (neuer statischer Ablaufplan):

- Zerstören und Erzeugung periodischer Aufgaben
- Aktivierung der neuen Ablauftabelle
- nichtperiodisch durch speziellen Arbeitsauftrag (mode-change job) kann der Wechsel wie folgt erfolgen
  - aperiodisch mit weichem Termin
    Kommt vor allen aperiodischen Aufträgen
    Notfalls hinauszögern von aperiodischen Aufgaben bis zum Abschluss des Wechsels
    Ziel ist die Minimierung der Antwortzeit für den Betriebswechsel
  - Sporadisch, Betriebswechsel mit hartem Termin
    Notfalls muss Betriebswechsel hinausgezögert werden

# 5 Nicht-periodische Echtzeitsysteme

## 5.1 Grundlegende Abfertigung nicht-periodischer Echtzeitsysteme

Lockerung der ersten Regel der item 1: "Alle Aufgaben sind periodisch"

- Mögliche Zustandsänderungen für Trigger von nicht-periodischen Aufgaben können beispielsweise sein:
  Kommunikation
  Fehlerbehandlung
- Im Folgenden wird versucht die mathematischen Restriktionen (item 1) zu entfernen und trotzdem die zugehörigen Beweise beibehalten zu können
- Entfernen von: "Alle Aufgaben sind periodisch"

Nicht-periodische Aufgabe $T_i = (i_i, e_i, D_i)$ - Abkürzungen: table 1:

- $i_i$ Minimale Zwischenankunftszeit (Minimales Intervall zwischen Auslösezeiten der Aufträge $T_i$) - (Ersatz für die Phase)
- $e_i$ WCET
- $D_i$ Relativer Termin
- Ist eine Abfolge von Arbeitsaufträgen ($J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$) mit vorgegebenen zeitlichen Eigenschaften
- Weiche/feste Termine $\mapsto$ Aperiodische Aufgabe
- Harte Termine $\mapsto$ Sporadische Aufgaben (deutlich seltener als aperiodische)
- Einflüsse nicht-periodischer Arbeitsaufträge ist zu begrenzen, um statische Garantien für periodische Arbeitsaufträge beibehalten zu können
- Aperiodische Arbeitsaufträge $\mapsto$ Antwortzeitminimierung
- Sporadische Arbeitsaufträge $\mapsto$ Termingarantie, Erfordert eine Entscheidung bzgl. Zulassung bzw. Abweisung des sporadischen Arbeitsauftrags. Für die Zulassung muss gelten Schlupf $\sigma >$ WCET

Behandlung nicht-periodischer Aufgaben:

- Unterbrecherbetrieb $\rightsquigarrow$ Nicht-periodische Aufgaben haben Vorfahrt (Unterbrechungsbehandlung, Verdrängung periodischer)
  Es kann sinnvoll sein nach Verbrauch der verfügbaren Slack-Time (der periodischen Aufgaben) die nicht-periodischen Aufgaben durch die entsprechenden periodischen zu verdrängen. Diese Funktionalität will jedoch erst einmal bereitgestellt werden.

  Pros: * Gute Antwortzeit für nicht-periodische Arbeitsaufträge

  Cons: * Unterbrechungsbehandlung notwendig
  * Verdrängung und Verzögerung periodischer Arbeitsaufträge

- Hintergrundbetrieb $\rightsquigarrow$ Periodische Aufgaben haben Vorfahrt (Verdrängung)

  Pros: * Liefert korrekte Ablaufpläne

  Cons: * Verdrängung erforderlich
  * Lange Antwortzeiten für nicht-periodische Arbeitsaufträge

- Periodischer Zusteller $\rightsquigarrow$ Alles ist eine periodische Aufgabe (periodisches Abfragen nicht-periodischer Ereignisse).
  Ist für die Einlastung von aperiodischen und sporadischen Aufgaben verantwortlich. Verfügt periodisch über ein gewisses Zeit-Budget. (Überbleibendes verfällt und kann für folgende nicht verwendet werden)

  Die einfachste Form eines periodischen Zustellers ist ein *Abfragender Zusteller*, der zu Beginn der Perioden Ereignisse abfragt. Dies muss hochfrequentiert erfolgen, um auch sporadische Aufgaben in der erforderlichen Zeit bearbeiten zu können.

  Pros: * Einfache Implementierung
  * Liefert korrekte Ablaufpläne

  Cons: * Lange Antwortzeiten durch Ausführungsbudget
  * Hoher Overhead durch Abfragebetrieb

- Slack-Stealing
  Verschiebung periodischer Aufträge um ihre Schlupfzeit, um für nicht-periodische Aufträge Zeit-Slots zu schaffen. Der Schlupfzeit-Dieb darf gierig sein.

  Schlupfzeiten korrekt zu berechnen ist jedoch aufwändig. Betrachtung kompletter Hyperperioden (statisch) und aktuellen Tätigkeitsintervallen (dynamisch).

  Pros: * Einfache Umsetzung

  Cons: * In vorranggesteuerten Systemen eher unpraktikabel

Wichtig ist stets die Berücksichtigung von Unterbrechungen, die durchaus selbst erhebliche Last erzeugen können. Daher sollten diese ein nicht zu präsentes Auftreten vorweisen. Indem beispielsweise Interrupts nach Bursts für einige Zeit blockiert werden.

## 5.2 Zustellerkonzepte und Übernahmeprüfung

Nicht-periodische Jobs in vorranggesteuerten Systemen:

- Bei den meisten Zustellern erfolgt die Einplanung nach Prioritäten, wobei der Zusteller die höchste hat.
- Slack-Stealing ist zu komplex
- Hintergrundbetrieb und abfragende Zusteller sind zu langsam (Verspätete Aufgaben landen in nächster Periode)
- Vordergrundbetrieb hat zu starken Einfluss auf periodische Aufgaben

⇒ Budget bewahren

Bandweite-bewahrender Zusteller:

- Planer innerhalb des OS logt Budgetverbrauch und suspendiert den Zusteller bei verbrauchtem Budget und stellt ihn bei Auffüllung bereit.
- Zusteller suspendet sich selbst bei Untätigkeit und sichert dadurch Restbudget
- Verbessert den Abfragebetrieb

### 5.2.1 Aufschiebbarer Zusteller (Bandweite-bewahrender Zusteller)

Ist kein periodisches Verhalten. Es sind *double hits, Doppeltreffer* möglich. Dh. während der Zusteller läuft wird sein Budget aufgefüllt.

Aufschiebbarer Zusteller - Deferrable Server $T_D = (p_D, e_D)$ - Abkürzungen: table 1:

- Periodische Auffüllen von Budget ($e_D$) und Bewahrung des Rest-Budgets ($T_D$) bei Untätigkeit
- Alternativ kann dies auch mit EDF (section 4.2) erfolgen, wobei auch periodische Arbeitsaufträge als sporadisch betrachtet werden. Betrachtung der Dichte $\Delta_i = \frac{e_i}{D_i - r_i}$, eines sporadischen Auftrags $S_i$. $\Delta = \sum_{i=1}^{n} \Delta_i = \sum_{i=1}^{n} \frac{e_i}{D_i - r_i} \leq 1$
- Restbudget verfällt am Ende der Abfrageperiode
- Verbraucherrate $\frac{1}{\text{Zeiteinheit}}$
- Setzen des Budgets zu Zeitpunkt $k \cdot p_s, \quad k \in [0, \infty[$, auf $e_s$

Aufschiebbarer Zusteller ∪ Hintergrundzusteller:

- Ist das Budget aufgebraucht (aufschiebbarer Zusteller) und liegen keine weiteren periodischen Aufgaben mehr an, so können weitere aperiodische/sporadische Aufgaben mittels des Hintergrundzustellers bearbeitet werden.
- (Bessere Antwortzeit), wenn periodische Aufgaben relativ kurz
- geringeres Zittern
- weniger Kontextwechsel
- Tradeoff kurze Antwortzeit $\mapsto$ großes Budget $\leftrightarrow$ Deadlines periodischer Tasks einhalten $\mapsto$ kleines Budget.
- Bei statischen Prioritäten meist besser als sporadische Zusteller

### 5.2.2 SpSL Sporadischer Zusteller, sporadic server $T_S = (p_S, e_S)$

Abkürzungen: table 1

- Ist wieder periodisches Verhalten, in Kombination von RM verfügt dieser nicht zwingend über die höchste Priorität
- Meist besser als aufschiebbare Zusteller
- Auffüllung zu Tätigkeitsbeginn
- Bestimmung des aufzufüllenden Budgets bei Tätigkeitsende
- Planbarkeitsanalyse anhand periodischer Aufgaben
  - Index $s$ signalisiert Server

- $P_s$ ist die Prioritätsebene des Zustellers $T_s$ oder eine höhere, also mindestens die Priorität des Zustellers
- $P_{cur}$ aktuelle Systempriorität
- Serverpriorität $P_s$ **tätig**: $P_{cur} \succeq P_s$
- Serverpriorität $P_s$ **untätig**: $P_{cur} \prec P_s$

1. $e_s$ initiales Ausführungsbudget
2. Auffüllzeitpunkt $rt_s = t_b + p_s$ für $T_s$, wobei $t_b$ der Zeitpunkt an dem:
   - $T_s$ Budget besitzt und $P_s$ tätig wird (eine aperiodische/sporadische Aufgabe kommt rein)
   - $T_s$ kein Budget besitzt, $P_s$ tätig ist und Budget aufgefüllt wird
3. Einplanung der nächsten Auffüllung zum Zeitpunkt $rt_s$
   - Wenn $P_s$ untätig wird oder Budget von $T_s$ erschöpft wird
   - So viel Budget wird aufgefüllt, wie $T_s$ seit $t_b$ bis zum ersten untätig werden verbrauchte

1. Überwache Un-/Tätigkeit der Prioritätsebene $P_i$ und bestimme den nächsten Auffüllzeitpunkt $rt_s$ von $T_s$
2. Überwache Verbrauch des Budgets von $T_s$
   - Suspendiere $T_s$, falls Budget erschöpft
   - Stelle $T_s$ bereit, falls Budget aufgefüllt wird
3. Verwalte anstehende Auffüllungen, Budget von $T_s$ wird in chunks zerschnitten

Das Äquivalente Posix-Konstrukt PSS nimmt einige Fehler für eine performante Implementierung in kauf (überziehen von Budget, verfrühte Auffüllung, ungünstige Verteilung globaler Prioritätsebenen, da PSS vom Scheduler geplante Jobs beliebig verzögern kann). Budget kann überzogen werden, wodurch auch zu viel aufgefüllt wird. Hier könnte man Überläufe von der nächsten Auffüllung borgen. (negatives Ausführungsbudget notwendig).

# 6 Rangfolge

Lockerung der zweiten Regel der item 1: “Alle Arbeitsaufträge können an ihren Auslösezeitpunkten eingeplant und ausgeführt werden” zugunsten mehrseitiger Synchronisation Lockerung der fünften Regel der item 1: “Alle Aufgaben sind unabhängig” zugunsten mehrseitiger Synchronisation

Ist im Grunde die Beschreibung von Abhängigkeiten. (Kausalordnung)

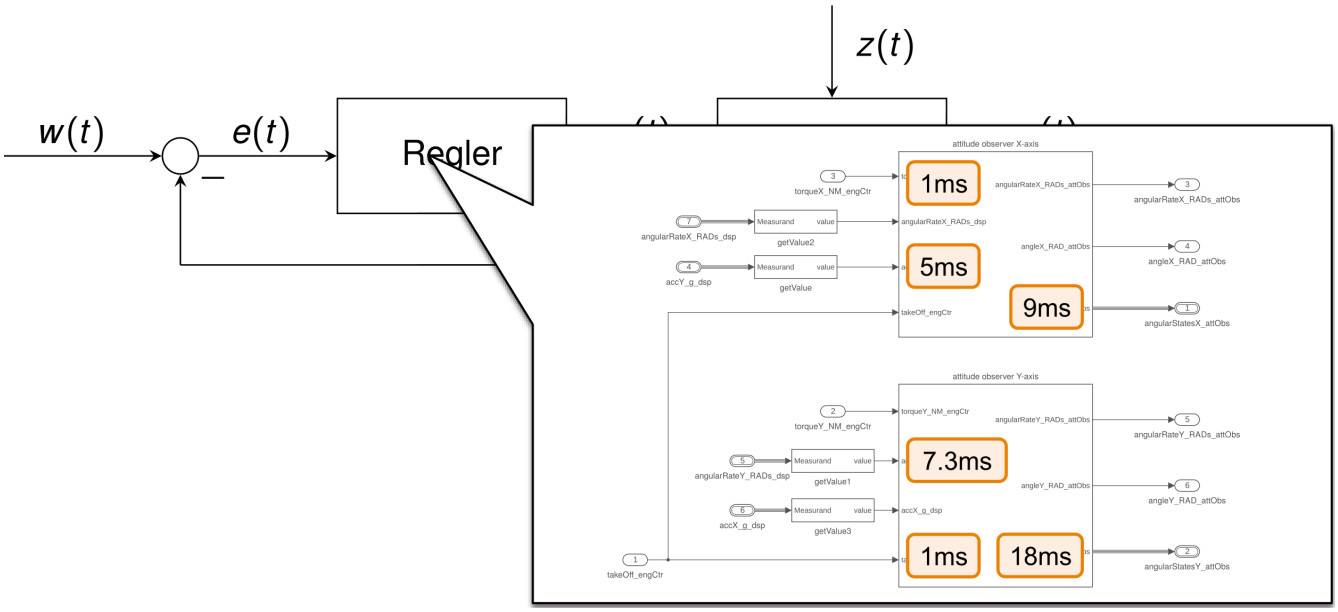



Abbildung 12: zusammengefasste Regler (Die Kästchen) sind ein schlechter Ansatz, weil alles pessimistisch mit 1ms gescheduled wird (da nicht entkoppelt), obwohl das gar nicht notwendig wäre.
Alternativ “least is best” Letzter Wert genügt erschwert Rausch-Unterdrückung, bspw. keine Mittelwertbestimmung mehr möglich.
Optimierung: Verschmelzung zeitlich identischer Domänen in einem Task

- Gegenteil von run-to-completion Semantik (Ereignisorientiertes Ausführungsmodell) ist Prozessorientiertes Ausführungsmodell

# 7 Zugriffskontrolle

Aufhebung der vierten Regel der item 1: “Kein Arbeitsauftrag gibt die Kontrolle über den Prozessor ab” zugunsten der blockierenden Synchronisation Lockerung der siebten Regel der item 1: “Alle Aufgaben verhalten sich voll-präemptiv” zugunsten mehrseitiger Synchronisation

Nicht-blockierende Synchronisationsmechanismen werden in EZS in der Praxis selten verwendet, sind aber theoretisch interessant.

Vergleich zur Rangfolge:

- Rangfolge $\mapsto$ zeitlich geordnete Ereignisse
- Zugriffskontrolle $\mapsto$ nebenläufige Ereignisse

Zugriffskontrolle:

Koordinierung nebenläufiger Ereignisse

- Synchronisation gleichzeitiger Zugriffe auf gemeinsame Betriebsmittel
- Erzeugung einer Rangfolge nebenläufiger Ereignisse
- Sequentialisierung von Arbeitsaufträgen entlang einer Kausalordnung
- Synchronisation ist immer eine nicht-funktionale Eigenschaft (der Systemaufrufe)

Bus ist unteilbar:

Der Bus lässt sich nicht trivial, falls überhaupt virtualisieren und entziehen. Ein Bus-Entzug während des Schreibens ist fatal.

- Begrenzte/unteilbare Betriebsmittel implizieren Kooperation
- Schutz vor Überlappung $\mapsto$ Semaphore
- Schutz vor Verdrängung $\mapsto$ Dispatcher deaktivieren

normale Prioritätenumkehr:

Ist die Folge der Blockierung eines höher priorisierten durch einen niedriger priorisierten Auftrag. Diese Form ist nicht vermeidbar, da Unteilbarkeit kritischer Abschnitte/Betriebsmittel vorliegt.

unkontrollierte Prioritätenumkehr:

Aufträge niedrigerer Priorität, die Locks reserviert haben, werden von unbeteiligten Aufträgen mittlerer Priorität verdrängt, wodurch ein etwaig höher priorer

Task, der das Lock belegen möchte die Deadline reißen könnte, da die Freigabe des Betriebsmittels zu lange benötigt.

Meiner Meinung nach könnte es hier interessant sein eingeschränkt nicht-blockierende Synchronisation zu verwenden. Also die nieder prioren Task durchlaufen den kritischen Abschnitt mit der Möglichkeit eines Rollbacks. Ein höher priorer Thread darf direkt das Lock "belegen" und den kritischen Abschnitt durchlaufen. Inwiefern hier Protokolle aussehen müssten wäre doch mal eine Vorlesung wert. ;)

Lösungsansätze:

- Verdrängungssteuerung - non preemptive critical sections (NPCS)
  - Jedwede Verdrängung wird für die gesamte Belegungszeit (kritischer Ressourcen) unterbunden.
  - Ist verklemmungsfrei
  - Vereinfacht Bestimmung der maximalen Verzögerungszeit
    Jeder höher priorisierte Auftrag wird maximal einmal durch nieder priore blockiert
  - Ist gut wenn die meisten Aufträge im Konflikt zueinander stehen und die Belegungszeit kurz ist
  - Für Systeme mit festen und dynamischen Prioritäten geeignet
- Prioritätsvererbung (priority inheritance)
  - Benachteiligt unbeteiligte Arbeitsaufträge
  - Vererbt Prioritäten beim Anfordern an Lock-haltende Aufträge bis zur Freigabe
  - Erschwert WCET-Berechnung, da nieder priore unbeteiligte höherer Priorität verzögern können.
  - Insbesondere bei geschachtelter Vererbung schwierig
  - Prioritätsumkehr wird nicht vermieden, aber entschärft.
  - Verklemmung möglich
- Prioritätsobergrenzen (priority ceiling) ($\hat{\Pi}$ eines Betriebsmittels), $J$ Auftrag mit Priorität $P(t)$ und Betriebsmittel $R$
  - Benachteiligt unbeteiligte Arbeitsaufträge
  - Verklemmungsvorbeugend
  - Die Blockierung aufgrund von Prioritätsobergrenzen nennt man auch *Aufhebungssperre - avoidance blocking*
    Jeder Arbeitsauftrag blockiert höchstens einmal. Keine Transitivität der Blockierung. Ein blockierender Auftrag wird nicht selbst blockiert.

- Spezialfall der Prioritätsvererbung
  Ist aber weniger greedy, da Anforderung zurückgewiesen werden kann, gleichwohl das zugehörige Betriebsmittel frei ist
  $J_l$ erbt aktuelle Priorität $P_h(t)$ von $J_h$
  $J_l$ behält diese Priorität bis er alle Betriebsmittel, deren Prioritätsobergrenze $\geq P_h(t)$ ist, freigab
- Aktuelle Prioritätsobergrenze des Systems $\hat{\Pi}(t)$
  1. $J$ fordert $R$ an
     * $R$ ist belegt $\mapsto$ $R$ ist gesperrt, $J$ blockiert
     * $R$ ist frei $\mapsto$ $R$ wird $J$ zugeteilt und gesperrt, falls
       · $J$'s Priorität ist größer als die Systemobergrenze: $P(t) > \hat{\Pi}(t)$
       · $J$ hält zum Zeitpunkt $t$ mindestens ein Betriebsmittel mit Prioritätsobergrenze: $P(t) \leq \hat{\Pi}(t)$
       · Sonst bleibt $R$ frei und $J$ blockiert
- Lässt sich durch Stapelorientierung vereinfachen.
  * Bei Vergabe wird die Priorität des Betriebsmittels vererbt
  * Wird ein Stack mehreren Prozessen zugeordnet, so darf kein Auftrag bei Anforderung eines gemeinsamen Betriebsmittels blockieren, da sonst Stackbereiche anderer Aufträge überschrieben werden können
  * Ausführungsabgabe lediglich durch preemption! Oben auf dem Stack läuft stets der höchst-priore
  * Aufträge blockieren niemals nach Ausführungsbeginn, da keine Einlastung ohne die Betriebsmittelzuteilung erfolgt
  * Verklemmungsfrei durch Verklemmungsvorbeugung (kein zirkulares Warten)
  * Geerbt wird die Priorität des Betriebsmittels
  * Betriebsmittel-Zuteilung erfolgt sofort mit Anforderung.
  * Mit Ausführungsbeginn einer Aufgabe sind alle weiteren benötigten Betriebsmittel frei, da sonst die Grenzpriorität größer-gleich ihrer Priorität wäre, wodurch die Einlastung verzögert worden wäre
  * Bei Verdrängung eines Auftrags sind alle von diesem benötigten Betriebsmittel frei, da die Verdrängung sonst von der Grenzpriorität unterbunden worden wäre.
  * Auf ein benötigtes Betriebsmittel kann direkt zugegriffen werden
  * Nicht immer kann jeder Aufgabe ein eigener Faden bzw. Stapel zugeteilt werden.

- Prioritätsobergrenzen mit dynamischer Priorität.
  Hier variieren die Prioritäten der Aufgaben ⇒ Die Grenzprioritäten der Betriebsmittel ändern sich ebenso
  Entsprechend müssen die Obergrenzen bei jeder Auslösung angepasst werden
  1. Weise Auftrag Priorität zu
  2. Aktualisiere Grenzprioritäten aller Betriebsmittel
  3. Aktualisiere Grenzpriorität des Systems
- à priori Wissen notwendig, um sinnvolle Prioritätsvergabe der Betriebsmittel vorzunehmen, aufwendig

Es fehlt irgendwo zwischen 29 und 32 eine Folie

Blockierungsarten:

- Direkte Blockierung
- Blockierung durch Vererbung
- Blockierung durch Aufhebungssperre

# 8 Mehrkern-Echtzeitsysteme

- Planbarkeit bei RMA ist bei einer Auslastung von etwa maximal 70% gegeben, wohingegen die Auslastung bei EDF durchaus 100% betragen darf.

Rate Monotonic Algorithm?

- Sehr viele valide Annahmen verlieren bei Verwendung von Mehrkernsystemen ihre Gültigkeit

Dhall Effekt:

Die garantierte Auslastung kann beliebig schlecht werden und konvergiert im schlimmsten Fall hin zu einem Einkernsystem.

Probleme/Herausvorderungen:

- Prioritätsproblem - Wann, in welcher Reihenfolge erfolgt Einlastung
  - statisch für Aufgaben - RMA
  - dynamisch für Aufgaben - EDF
  - dynamisch für Aufträge - PFAIR
- Allokationsproblem - Wo, auf welchem Kern
  - Keine Migration - no migration
  - Migration von Aufgaben - task-level migration
  - Migration von Aufträgen - job-level migration (Das sinnvollste, da wenig Datenmengen verschoben werden müssen)

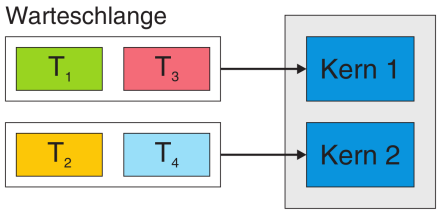



Abbildung 13: Partitionierte Ablaufplanung
Pinning von Aufgaben an Kerne (vor Laufzeit) mit Kern-lokalen Warteschlangen.
Auslastung im schlimmsten Fall ca. 50%

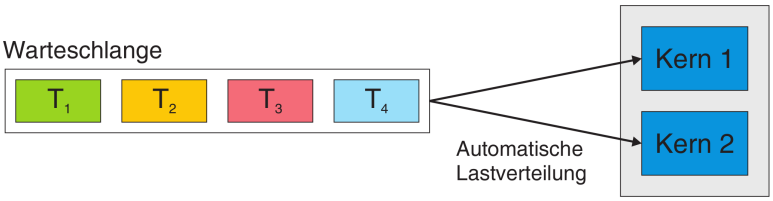



Abbildung 14: Globale Ablaufplanung
Verteilung zur Laufzeit, wobei Migration möglich, hierbei fallen allerdings unter Umständen hohe Kosten an und Laufzeit-Analysen werden erschwert.

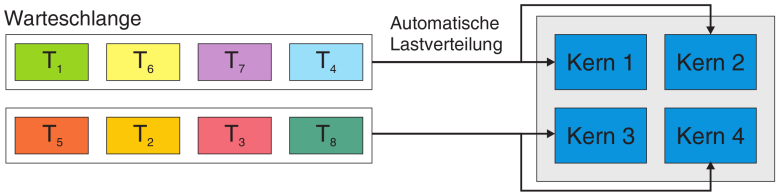



Abbildung 15: Hybride Ablaufplanung
Minimierung der Migration bei Maximierung der Auslastung
Mittels Zusammenfassen mehrerer Kerne zu Clustern und Begrenzung der Migrations-Möglichkeit (z.B. Migration nur zwischen bestimmten Kernen möglich)

# 9 Abkürzungsverzeichnis

Tabelle 1: Abkürzungen

| Typographische Konvention | | |
|---|---|---|
| Erster Index bezeichnet die Aufgabe | | |
| Zweiter Index bezieht sich auf den Arbeitsauftrag | | |
| Exponenten definieren die Eigenschaften | | |
| Funktionen spezifizieren zeitliche Eigenschaften | | |
| **Temporale Eigenschaften** | | |
| $r_i$ | Auslösezeitpunkt | |
| $e_i$ | Maximale Ausführungszeit (WCET) | |
| $D_i$ | Relativer Termin | |
| $d_i$ | Absoluter Termin | |
| $\omega_i$ | Antwortzeit | |
| $p_i$ | Periode | |
| $\phi_i$ | Phase | |
| $i_i$ | nicht-periodisch - Minimale Zwischenankunftszeit | |
| | (Zeit zw. Auslösung der Aufträge) | |
| **Strukturelemente** | | |
| $E_i$ | Ereignis | |
| $R_i$ | Ergebnis | |
| $T_i$ | Aufgabe | $T_i(p_i, e_i, D_i, \phi_i)$ |
| | nicht-periodisch | $T_i = (i_i, e_i, D_i)$ |
| $J_{i,j}$ | Arbeitsauftrag der Aufgabe $T_i$ | $J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$ |
| **Zusteller** | | |
| $p_z d$ | Periode | |
| $e_z$ | Budget | |
| $T_Z$ | Rest-Budget | $T_Z = (p_Z, e_Z)$ |
| Wobei $z, Z$ ersetzt wird durch: | | |
| $d, D$ | Deferrable Server, Aufschiebbarer Zusteller | |
| $s, S$ | Sporadic Server, Sporadischer Zusteller | |

## Section 7 - fehlende Folie

Mathematisches Modell (item 1) am Ende der Vorlesung.

1. “Alle Aufgaben sind periodisch”
   Gelockert
2. “Alle Arbeitsaufträge können an ihren Auslösezeitpunkten eingeplant und ausgeführt werden”
   Gelockert
3. “Termine und Perioden sind identisch”
4. “Kein Arbeitsauftrag gibt die Kontrolle über den Prozessor ab”
   Aufhebung zugunsten der blockierenden Synchronisation
5. “Alle Aufgaben sind unabhängig”
   Gelockert
6. “Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar”
7. “Alle Aufgaben verhalten sich voll-präemptiv”
   Lockerung zugunsten mehrseitiger Synchronisation

# 10 Abbildungsverzeichnis

## Abbildungsverzeichnis



# 11 Tabellenverzeichnis

## Tabellenverzeichnis

## Liste der noch zu erledigenden Punkte