Echtzeitsysteme

Zeitgesteuerte Ablaufplanung periodischer Echtzeitsysteme

Lehrstuhl Informatik 4

24. November 2011

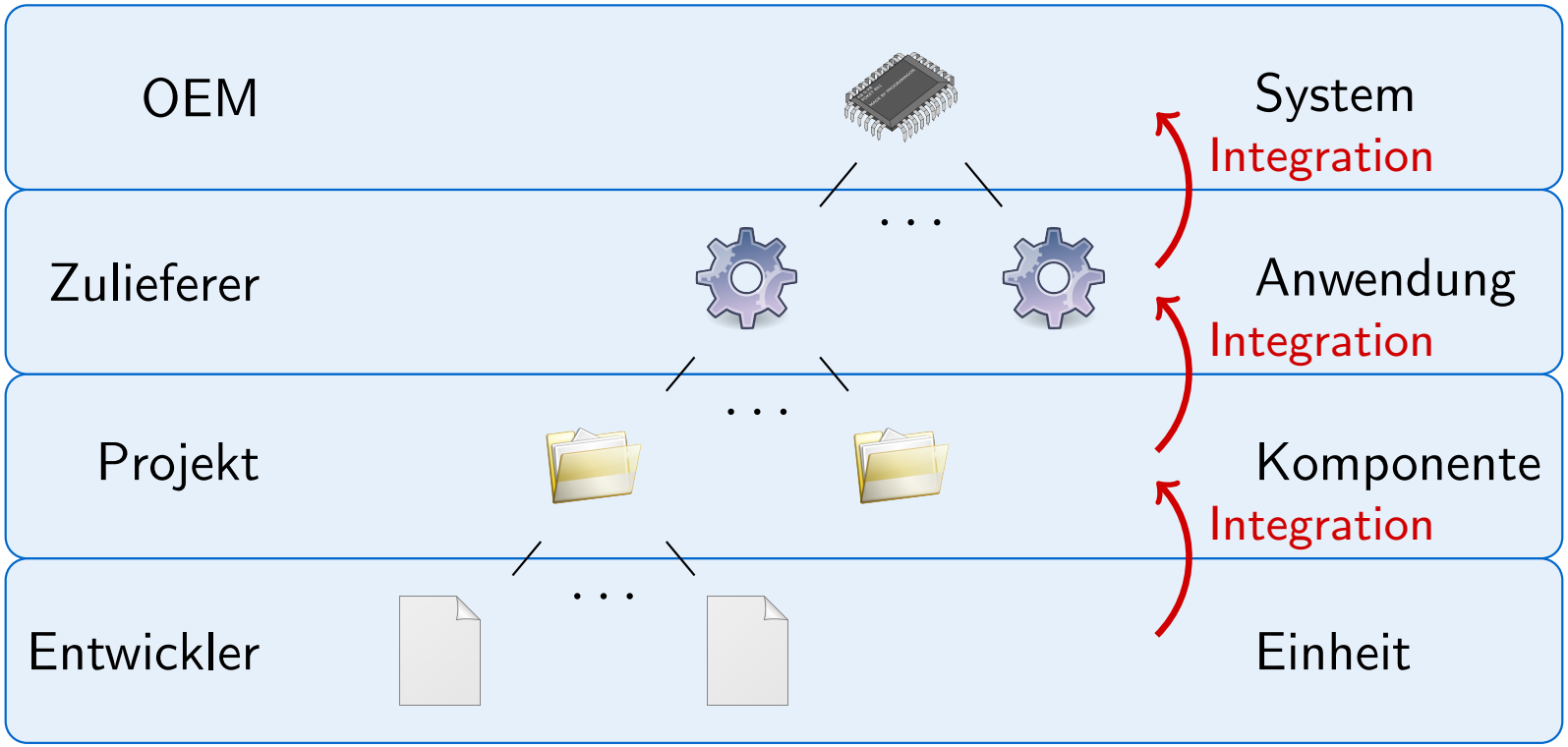
Gliederung

Fragestellungen

- Wie bestimmt man eine geeignete Ablauftabelle für eine gegebene Menge von Aufgaben?
- Manuelle Bestimmung zyklischer Ablaufpläne
  - Warum bestimmt man Ablaufpläne manuell?
  - Gibt es Leitlinien, um die manuelle Erstellung zu unterstützen?
- Algorithmische Bestimmung zyklischer Ablaufpläne
  - Heuristische Verfahren
  - Optimale Verfahren
- Wie flexibel sind zyklische Ablaufpläne?

Gliederung

Ablaufplanung – eine Bottom-Up-Perspektive



Der Integrationsprozess verläuft *Bottom-Up*:

1. Bündelung von Softwareeinheiten (engl. *unit*) zu Komponenten
2. Komponenten implementieren Arbeitsaufträge in Anwendungen
3. Einplanung der Arbeitsaufträge in einer statischen Ablauftabelle

Herausforderung: Integration

Die Ablaufplanung ist der finale Schritt bei der Systemerstellung, der jedoch enorm von den bereitgestellten Edukten abhängt:

Software-Einheiten und -Komponenten Implementierungs- und Entwurfsentscheidungen beeinflussen die maximalen Ausführungszeiten dieser Artefakte.

Anwendung Die Abbildungen Komponenten $\mapsto$ Arbeitsaufträge und Jobs $\mapsto$ Aktivitätsträger beschneiden den Spielraum der Ablaufplanung.

Die Erstellung von Software-Einheit, -Komponente, Anwendung und System fällt meist in verschiedene Zuständigkeitsbereiche:

- Softwarekomponenten werden von Softwarehäusern zugekauft (z.B. Betriebssystem, Mathematik- oder Kryptographiebibliothek)
- ein Zulieferer fügt diese Komponenten zu einer Anwendung zusammen (z.B. ABS, Fahrspurassistent)
- ein OEM fertig schließlich das endgültige Produkt (z.B. ein Auto)

Herausforderung: Integration (Forts.)

Wenn funktionale Schnittstellenbeschreibungen nicht ausreichen

Nachträgliche Änderungen an den Softwareeinheiten, -komponenten und Anwendungen bedeuten für den OEM beträchtlichen Aufwand:

Fehlerbehebungen beeinflussen das Laufzeitverhalten

- die maximale Laufzeit von Softwarekomponenten ändert sich
- das Laufzeitverhalten wandelt sich u.U. vollständig, wenn die Abbildung auf die Aktivitätsträger angepasst werden muss
  - z.B. wenn Abhängigkeiten missachtet wurden

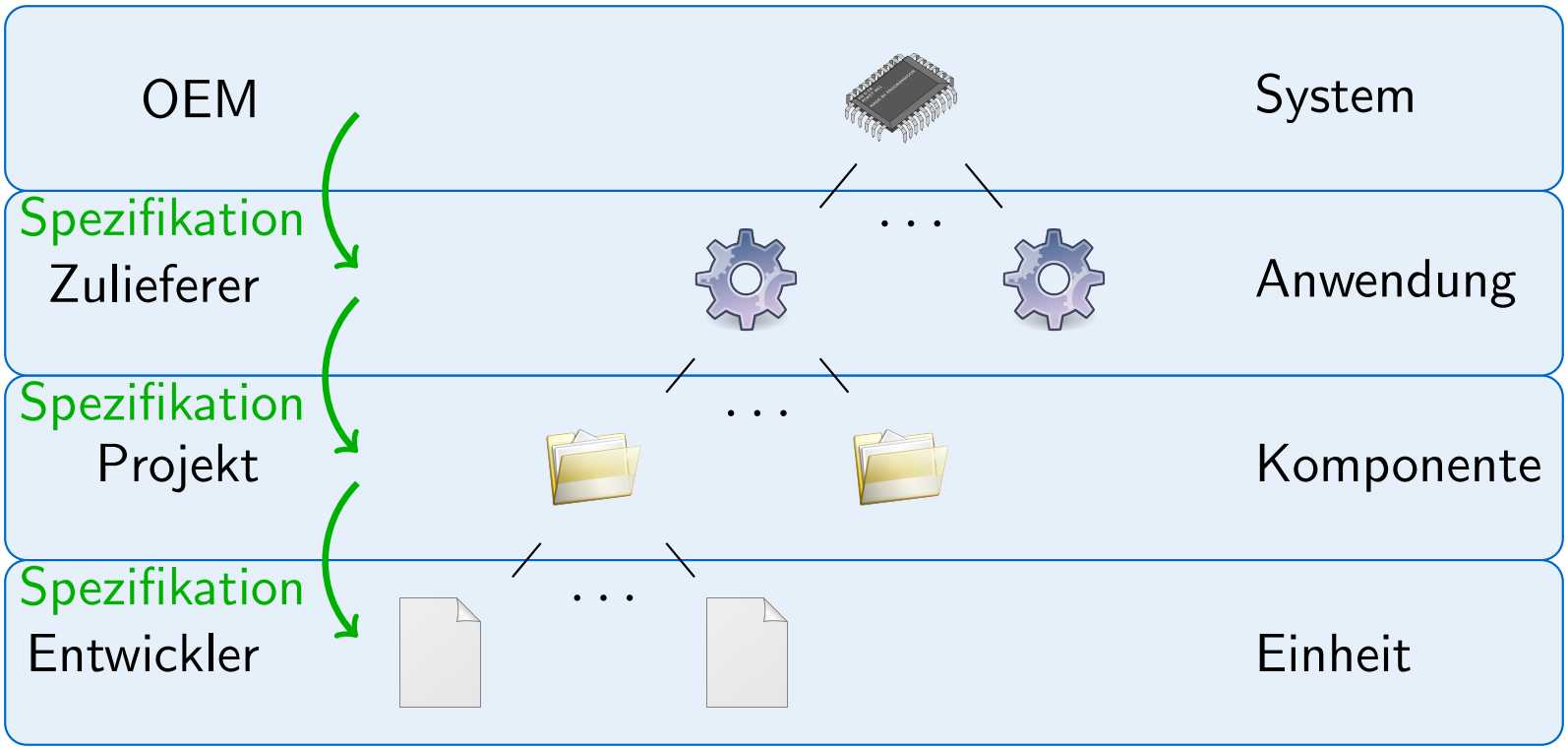
Nachbesserungen werden notwendig, wenn eine Komponenten z.B. einfach zu viel Rechenzeit benötigt, und die Ablaufplanung fehlschlägt

- aufgrund ineffizienter Codierung oder
- wegen einer ungeschickten Strukturierung der Anwendung

☞ Spezifiziere des zeitlichen Verhaltens der Softwarekomponenten!

- z.B. durch eine manuelle, vorgezogene Ablaufplanung

Spezifikation zeitlichen Verhaltens



Die Spezifikation erfolgt *Top-Down*:

- der OEM weist den Anwendungen Zeitschlitze im Ablaufplan zu
- Anwendungen verteilen die Rechenzeit auf Softwarekomponenten
- Komponenten und Einheiten müssen mit ihrer Rechenzeit haushalten

Spezifikation zeitlichen Verhaltens (Forts.)

Eine globale Planung der zeitlichen Abläufe ist wünschenswert:

- Einhaltung der Zeitschlitze etc. werden zu lokalen Belangen

$\rightsquigarrow$ Problemlösung innerhalb desselben Zuständigkeitsbereichs ist möglich

Gute Verbindung zur Idee der Rahmenkonstruktion (engl. *framework*)

- *Hollywood*-Prinzip: „Don't call us, we'll call you"

$\rightsquigarrow$ der OEM gibt auch die Anwendungsstruktur vor

Problem: Für die Erstellung eines geeigneten, globalen Ablaufplans ist Vorabwissen notwendig!

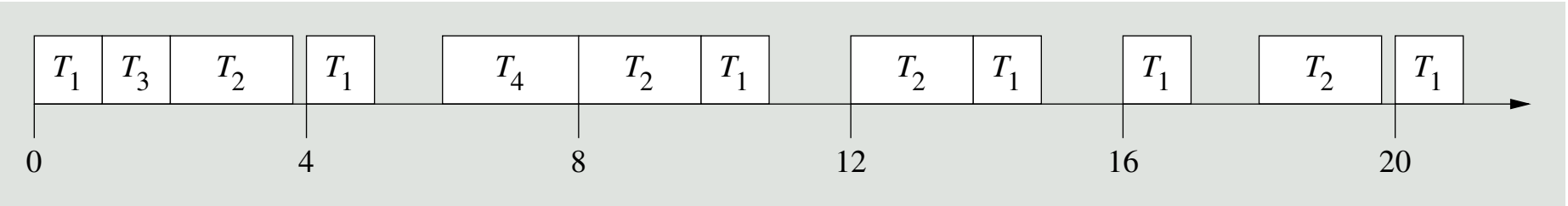
- Rückgriff auf zurückliegende Entwicklungsprojekte
- Erkenntnis aus der Entwicklung von Prototypen

☞ Leitlinien für die manuelle Erstellung gut strukturierter, zyklischer Ablaufpläne sind wünschenswert und sinnvoll!

Gliederung

Regelmäßigkeit zyklischer Abläufe

Einplanungsentscheidungen können trotz periodischer Aufgaben *ad hoc*, d.h., in unregelmäßigen Abständen wirksam werden:



- Entscheidungszeitpunkte sind 0, 1, 2, 4, 6, 8, 10, 12, 14, 16, 18

Regularität bei der Umsetzung und Überprüfung solcher Entscheidungen zur Laufzeit trägt wesentlich zum Determinismus bei

„gute Anordnung“ (engl. *good structure*) eines zyklischen Ablaufplans

- Einplanungsentscheidungen *nicht* zu beliebigen Zeitpunkten treffen

Rahmen (engl. *frames*)

Strukturelemente von zyklischen Ablaufplänen

Zeitpunkte von Einplanungsentscheidungen unterteilten die Echtzeitachse in Intervalle fester Länge f (engl. *frame size*)

- Entscheidungen werden nur am Rahmenanfang getroffen/wirksam
  - innerhalb eines Rahmens ist Verdrängung ausgeschlossen

$\rightsquigarrow$ die Phase einer periodischen Aufgabe ist Vielfaches von f

- jeder Job einer Task wird am Anfang eines Rahmens ausgelöst

Aufgaben, die der *Dispatcher* zusätzlich zur Einlastung eines Jobs am Anfang eines Rahmens durchführen kann...

- sind Überwachung/Durchsetzung von Einplanungsentscheidungen:
  - Wurde ein für den Rahmen eingeplanter Job bereits ausgelöst?
  - Ist dieser Job auch zur Ausführung bereit?
  - Gab es einen „Überlauf“ eines Termins, steht Fehlerbehandlung an?
- beinflussen im großen Maße die Bestimmung eines Wertes für f

Randbedingungen für die Rahmenlänge

Lang genug und so kurz wie möglich halten...

f hinreichend lang $\rightsquigarrow$ Jobverdrängung vermeiden

1. ist erfüllt, wenn gilt: $f \geq \max(e_i)$, für $1 \leq i \leq n$
   - jeder Job läuft in der durch f gegebenen Zeitspanne komplett durch
2. f teilt die Hyperperiode H so, dass gilt: $\lfloor p_i/f \rfloor - p_i/f = 0$
   - ermöglicht die zyklische Ausführung des Ablaufplans

- das Intervall H heißt großer Durchlauf (engl. *major cycle*,
  - Intervall der Länge f heißt kleinster Durchlauf (engl. *minor cycle*)

f hinreichend kurz $\rightsquigarrow$ Terminüberwachung unterstützen

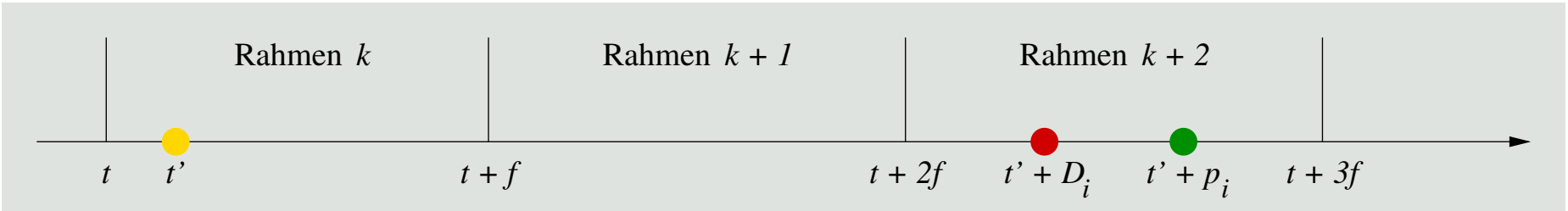
3. erfordert eine rechtzeitige Auslösung: $f \leq p_i$, für $1 \leq i \leq n$
4. ist möglich unter der Bedingung: $2f - gcd(p_i, f) \leq D_i$

- Rahmen „passend“ auf die anstehenden Aufgaben verteilen
  - zwischen der Auslösezeit und dem Termin jedes Jobs

Randbedingungen für die Rahmenlänge (Forts.)

Platzierung einer Task auf der Echtzeitachse

Feststellung eines passenden Bereichs für f von $T = (p_i, e_i, D_i)$:[1]



- t ist der Anfang eines Rahmens, in dem ein Job in T_i ausgelöst wird
- t' ist der Zeitpunkt der Auslösung des betreffenden Jobs
- Rahmen $k + 1$ erlaubt die Kontrolle des bei t' ausgelösten Jobs
  - der Rahmen sollte daher zwischen t' und $t' + D_i$ des Jobs liegen
- dies ist erfüllt, wenn gilt: $t + 2f \leq t' + D_i$ bzw. $2f - (t' - t) \leq D_i$
  - $t' - t$ ist mindestens größter gemeinsamer Teiler von p_i und f [3]

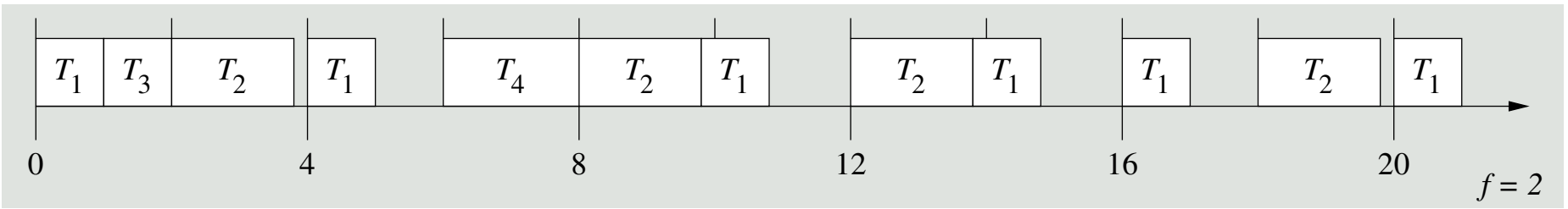
[1] Befindet sich f in diesem Bereich, gibt es wenigstens einen Rahmen zwischen der Auslösungszeit und dem Termin jedes Arbeitsauftrags der betreffenden Aufgabe.

Randbedingungen für die Rahmenlänge (Forts.)

$T_i = (p_i, e_i, D_i, \phi_i)$; gilt $D_i = p_i$ und $\phi_i = 0$, werden D_i und ϕ_i nicht geschrieben

Beispiel: $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$, $T_4 = (20, 2)$

- $f \geq 2$ muss gelten, um jeden Job komplett durchlaufen zu lassen
- mögliche Rahmenlängen in H sind 2, 4, 5, 10 und 20 ($H = 20$)
- nur $f = 2$ erfüllt jedoch alle drei Bedingungen (S. IV-3/13) zugleich



Beispiel: $T_x = (15, 1, 14)$, $T_y = (20, 2, 26)$, $T_z = (22, 3)$

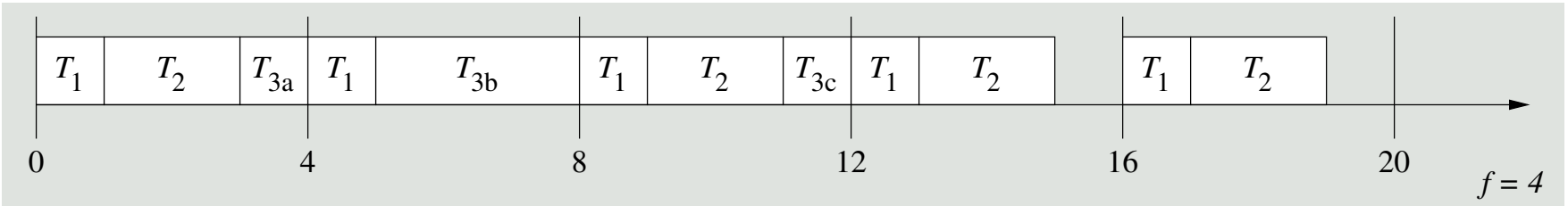
- $f \geq 3$ muss gelten, um jeden Job komplett durchlaufen zu lassen
- mögliche Rahmenlängen in H: 3, 4, 5, 10, 11, 15, 20, 22 ($H = 660$)
- jedoch nur $f = 3, 4$ oder 5 erfüllt alle drei Bedingungen (S. IV-3/13)

Konflikte und deren Auflösung

Taskparameter zugunsten einer guten Ablaufplananordnung korrigieren

Arbeitsaufträge sind in Scheiben zu schneiden, falls die Parameter der Aufgaben nicht alle Randbedinungen (S. IV-3/13) erfüllen können

- gegeben sei folgendes Tasksystem $\mathbf{T} = \{(4,1),(5,2,7),(20,5)\}$:
  - $f \geq \max(e_i)$ gilt für $f \geq 5$ und $2f - gcd(p_i, f) \leq D_i$ gilt für $f \leq 4$ **!?**
- $T_3 = (20,5)$ ist aufzuteilen in $T_3' = \{(20,1),(20,3),(20,1)\}$
  - d.h., in drei Teilaufgaben $T_{3a} = (20,1)$, $T_{3b} = (20,3)$, $T_{3c} = (20,1)$
  - das resultierende System hat fünf Tasks und die Rahmenlänge $f = 4$



- $T_3 = (20,5)$ in zwei Teilaufgaben aufzuteilen, bleibt erfolglos:
  - $\{(20,4),(20,1)\}$ geht nicht, wegen $T_1 = (4,1)$
  - $\{(20,3),(20,2)\}$ geht nicht, da für $T_{3b} = (20,2)$ kein Platz bleibt

Entstehungsprozess eines zyklischer Ablaufplans

Gegenseitige Abhängigkeit von Entwurfsentscheidungen

1. eine Rahmenlänge festlegen (S. IV-3/13)
   - durch Taskparameter ggf. gegebene Konflikte erkennen und auflösen
2. Arbeitsaufträge in Scheiben aufteilen (S. IV-3/16)
   - insbesondere kann dies zur Folge haben, andere Programm- bzw. Modulstrukturen herleiten zu müssen
   - die erforderlichen Programmtransformationen geschehen bestenfalls (semi-) automatisch durch spezielle Kompilatoren
   - schlimmstenfalls sind die Programme manuell nachzuarbeiten
3. die Arbeitsauftragsscheiben in die Rahmen platzieren

☞ Rahmenlängen sind querschneidende nicht-funktionale Eigenschaften

☞ Besondere Eignung für die Planung von Kommunikationssystemen

- Nachrichten lassen sich sehr gut und gezielt aufteilen

Gliederung

Handarbeit ist mühsam!

Statische Ablaufpläne werden sehr schnell umfangreich

- statische Ablauftabellen zur Hyperperiode „aufgeblasen“
- Beispiel: $T_1 = (20, 3),\ T_2 = (15, 2),\ T_3 = (2, 0.25)$
  - resultiert in einer Ablauftabelle mit 37 Einträgen
  - fügt man $T_4 = (40, 3)$ hinzu, werden daraus 77 Einträge

Ablaufplanung ist algorithmisch schwierig (s. Folie IV-2/34)

- bis auf einfach gelagerte Fälle, ist Ablaufplanung stark $\mathcal{NP}$-hart

☞ Automatisierte Berechnung von Ablauftabellen

- Computer sind dafür da, große Datenmengen schnell zu verarbeiten
- exponentielles Wachstum der Laufzeit ist auch für Computer fatal
  - Entwicklung heuristischer und optimaler Verfahren
- Verfahren haben nur eine sehr geringe Praxisrelevanz
  - pessimistische Annahmen über die WCET erweisen sich als hinderlich

Überblick - Heuristiken und optimale Verfahren

Grundlegende Aufgabenstellung: Berechnung einer statischen Ablauftabelle für eine Menge periodischer Aufgaben (s. Folie IV-2/29)
Existierende Verfahren gehen deutlich über diese Anforderungen hinaus:

- Berücksichtigung gerichteter und ungerichteter Abhängigkeiten
- verteilte Systeme und Mehrkern- sowie Mehrprozessorsysteme
  - Allokation von Rechenknoten für Arbeitsaufträge
  - integrierte Ablaufplanung für Kommunikationssysteme
- Beschleunigung von Arbeitsaufträgen durch Duplizierung
- ...

Heuristiken finden u.U. keine Lösung, obwohl das Problem lösbar ist.

- z.B. genetische Algorithmen oder List-Scheduling [4].

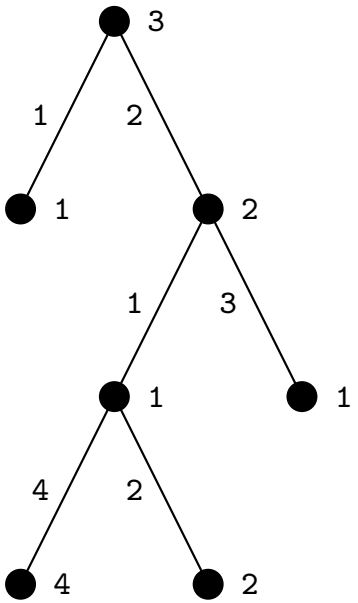
Optimale Verfahren finden eine Lösung, sofern eine solche existiert.

$\rightsquigarrow$ exponentiell wachsende Laufzeit im schlimmsten Fall

- z.B. lineare Programmierung [7] oder Branch&Bound [1]

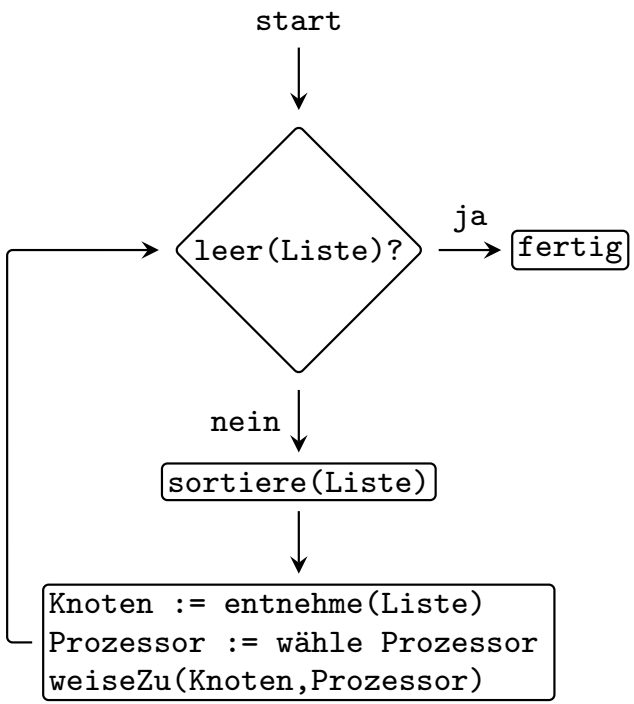
List-Scheduling

List-Scheduling als Basis heuristischer Verfahren, die gerichtete, azyklische Graphen (engl. *directed acyclic graph*, DAG) ordnen [6].



Knotengewichten $\rightsquigarrow$ Berechnung

Kantengewichten $\rightsquigarrow$ Kommunikation

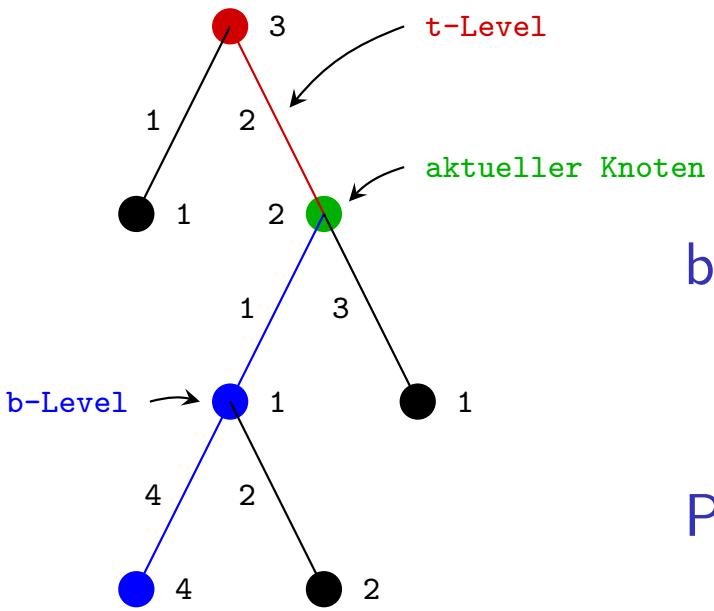


verwaltet Knoten in einer sortierten Liste

b-Level und t-Level

Grundlage der Sortierkriterien beim List-Scheduling

Die Position eines Knotens im DAG bestimmt die Position in der Liste:



t-Level der längste Pfad von einer Wurzel zum aktuellen Knoten

- variiert während der Ablaufplanung
- Kommunikation *verschwindet* u.U.

b-Level der längste Pfad vom aktuellen Knoten zu einem Endknoten

- während der Planung i.d.R. konstant

Pfadlänge Summe der Knoten- und Kantengewichte ohne aktuellen Knoten

verschiedene Algorithmen gewichten t-Level und b-Level unterschiedlich

b-Level $\rightsquigarrow$ Knoten auf dem kritischen Pfad werden bevorzugt

t-Level $\rightsquigarrow$ plant Knoten entlang der topologischen Ordnung

Beispiele

HLFET (**H**ighest **L**evel **F**irst with **E**stimated **T**imes) [2]

- vernachlässigt Kommunikation $\rightsquigarrow$ Kantengewichte $= 0$
- verwendet b-Level als alleiniges Sortierkriterium

ISH (**I**nsertion **S**cheduling **H**euristic) [5]

- Sortierkriterium $\rightsquigarrow$ wie HLFET
- evtl. entstehen Intervalle der Untätigkeit (engl. *idle time slots*)
  - Auffüllung durch Knoten aus der Bereitliste falls möglich

DLS (**D**ynamic **L**evel **S**cheduling) [8]

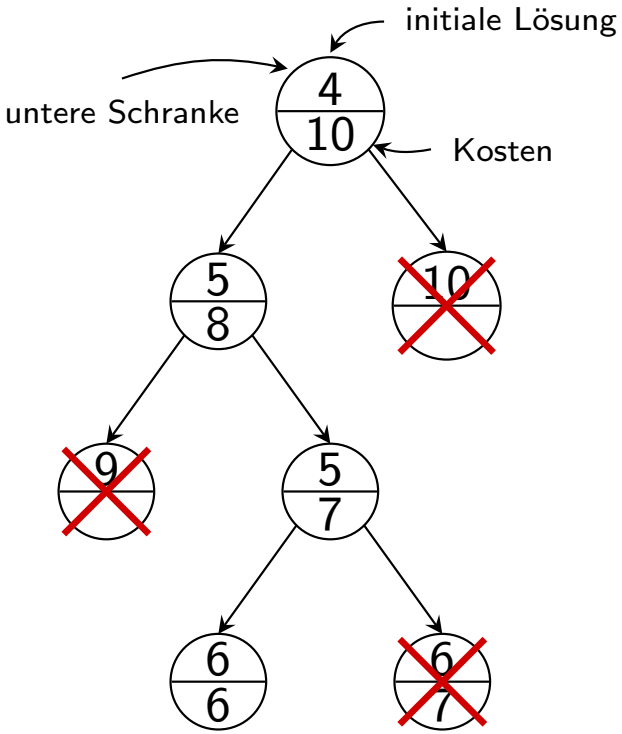
- Sortierkriterium: *Dynamic Level*
  - Differenz: b-Level – frühestem Startzeitpunkt für einen Prozessor
  - wird nach jedem Durchlauf neu berechnet
- Bereitliste enthält zunächst nur die Wurzelknoten des DAG
  - Hinzufügen von Knoten, sobald deren Vorgänger eingeplant wurden

Branch&Bound-Prinzip

Planungsproblem $\mapsto$ Suchproblem in einem Suchbaum

- u.U. muss der komplette Suchraum betrachtet werden

$\rightsquigarrow$ eine potentiell exponentielle Laufzeit ist die Folge



1. finde eine initiale Lösung
   - bestimme ihre tatsächlichen Kosten und
   - eine untere Schranke für die Kosten
2. leite verbesserte initiale Lösungen ab
   - das ist der Branch-Schritt
   - verwirf ungeeignete initiale Lösungen

   $\rightsquigarrow$ Reduktion des Suchraums: Bound-Schritt
3. Wiederhole diese Schritte ...
   bis die optimale Lösung gefunden wurde
   - oder klar ist, dass keine Lösung existiert

☞ Um Optimalität zu erreichen, müssen im Branch-Schritt alle Möglichkeiten ausgeschöpft werden, eine Lösung zu verbessern

Branch&Bound-Algorithmen und statische Ablaufplanung

Wo kommen die initiale Lösung, die Kosten und die untere Schranke her?

Initiale Lösungen sind bereits vollständige gültige Ablaufpläne

- diese können aber noch Termine verletzen, sind also nicht zulässig

$\rightsquigarrow$ ein Verfahren für deren Bestimmung wird benötigt

Kosten einer Lösung sind daher ihre Verspätung

- das ist die maximale Terminüberschreitung aller Jobs

Untere Schranken liefert die Vereinfachung des Planungsproblems

- so dass oben gewähltes Verfahren bereits optimal ist

Verbesserung erzielt man durch Manipulation des Planungsproblems

- den Job mit der größten Verzögerung früher einplanen

$\rightsquigarrow$ die Verspätung des Systems reduzieren

- ohne die ursprünglichen Vorgaben zu verletzen

☞ Ziel: Eine Lösung finden, deren Kosten kleiner oder gleich 0 sind. Der zugehörige Ablaufplan ist daher zulässig.

Der Algorithmus von Abdelzaher und Shin [1]

Ein Beispiel für einen Branch&Bound-Algorithmus für die statische Ablaufplanung

Was kann der Algorithmus? – Eine Merkmalsliste...

- Auslösezeiten, Ausführungszeiten, Termine
- Gerichtete und ungerichtete Abhängigkeiten
- Einkern-, Mehrkern- und Mehrprozessorsysteme
  - Der Algorithmus führt jedoch keine Allokation durch!
- verteilte Systemen und nachrichtenbasierter Kommunikation

Initiale Lösung $\rightsquigarrow$ globaler EDF-Algorithmus

- erweitert um die Behandlung ungerichteter Abhängigkeiten

$\rightsquigarrow$ für obiges Planungsproblem nicht optimal

Kosten Ablaufplan mit dem globalen EDF-Algorithmus bestimmen

Untere Schranke Problem so vereinfachen, dass EDF optimal ist!

- keine ungerichteten oder kernübergreifenden Abhängigkeiten

Verbesserung durch das gezielte hinzufügen von Abhängigkeiten

Gliederung

Betriebswechsel: Flexibilität in zeitgesteuerten Systemen

Eine „*one fits all*“-Lösung für statische Ablauftabellen ist nicht sinnvoll

- Negativbeispiel: Neuprogrammierung von Steuergeräten im Auto
  - Software wird per CAN-Bus verteilt
  - Daten werden in das normale Kommunikationsverhalten eingebettet
  - niedrige Nutzlast (engl. *payload)* ist die Folge

- statische Ablauftabellen orientieren sich am schlimmsten Fall
  - Jobs beanspruchen immer die ihnen zugewiesene Ausführungszeit
  - auch wenn sie zwar periodisch, aber nur selten ausgelöst werden
- eine Entflechtung der Arbeitsaufträge ist das Ziel
  - Arbeitsaufträge befinden sich nur in einer gemeinsamen Ablauftabelle, wenn sie auch zusammen ausgelöst werden können
- Gruppierungen solcher Arbeitsaufträge definieren Betriebszustände
  - repräsentiert durch eine eigene Ablauftabelle
  - der Wechsel des Betriebszustands impliziert auch ihren Wechsel
- ein Betriebswechsel erfordert ein systemweit koordiniertes Vorgehen
  - insbesondere in verteilten Systemen ist dies nicht trivial

Rekonfiguration des Tasksystems

Änderung von Taskanzahl und -parameter

Umstellen auf einen neuen statischen Ablaufplan bedeutet mehr als nur einen Tabellenwechsel zu vollziehen:

1. Zerstörung und Erzeugung von periodischen Tasks
   - einige periodische Tasks werden aus dem System gelöscht, wenn ihre Funktion nicht mehr erforderlich ist $\rightsquigarrow$ Betriebsmittelfreigabe
   - andere müssen dem System neu hinzugefügt werden, ggf. sind Programme nachzuladen $\rightsquigarrow$ Betriebsmittelanforderung
   - manche Tasks überdaueren den Betriebswechsel, da sie im alten und neuen Tasksystem benötigt werden
2. Einlagerung und Aktivierung der neuen Ablauftabelle
   - neue Taskparameter und neuer Ablaufplan wurden *à priori* bestimmt

Betriebswechsel vom speziellen Arbeitsauftrag (engl. *mode-change job*) durchführen lassen $\mapsto$ nichtperiodischer Job

- ausgelöst durch ein (interaktives) Kommando zum Betriebswechsel
- verbunden mit einem weichen oder harten Termin

Arten von Betriebswechsel

Aperiodischer oder sporadischer Job

aperiodisch $\mapsto$ Betriebswechsel mit weichem Termin

- mit höchster Dringlichkeit ausgeführt als aperiodischer Job
  - der vor allen anderen aperiodischen Jobs zum Zuge kommt
- Zerstörung aperiodischer/sporadischer Jobs ist problematisch
  - die Ausführung aperiodischer Jobs wird hinausgezögert, bis der Betriebswechsel vollendet worden ist
  - im Falle sporadischer Jobs stehen zwei Optionen zur Verfügung:
    (a) der Betriebswechsel wird unterbrochen und später fortgesetzt
    (b) die Übernahmeprüfung berücksichtigt den neuen Ablaufplan
- Ziel ist es, die Antwortzeit für den Betriebswechsel zu minimieren

sporadisch $\mapsto$ Betriebswechsel mit hartem Termin

- die Anwendung muss die evtl. Abweisung des Jobs behandeln
  - sie wird den Betriebswechsel ggf. hinausschieben

Gliederung

Resümee

Entwicklungsprozesse führen verschiedenste Akteure zusammen

- Firmen/Arbeitsgruppen sind u.U. über den ganzen Globus verstreut

$\rightsquigarrow$ eine zeitliche Spezifikation der Abläufe ist wünschenswert

- sie ermöglicht die Entwicklung top-down zu strukturieren
- wird durch eine manuelle, statische Ablaufplanung unterstützt

Struktur zyklischer Ablaufpläne $\rightsquigarrow$ „gute Anordnung“, Determinismus

- Rahmen, Rahmenlänge, Scheiben; *major/minor cycle*

Algorithmische Einplanung ordnet gerichtete, azyklische Graphen

- und berücksichtigt diverse Nebenbedingungen

$\rightsquigarrow$ Entlastung bei der Lösung eines komplexen Problems

- List-Scheduling- und Branch&Bound-Algorithmen

Betriebswechsel bewerkstelligen aperiodische oder sporadische Jobs

- Tabellenwechsel, Betriebsmittelfreigabe/-anforderung, Nachladen

Literaturverzeichnis

[1] Abdelzaher, T. F. ; Shin, K. G.:
Combined Task and Message Scheduling in Distributed Real-Time Systems.
In: *IEEE Transactions on Parallel and Distributed Systems* 10 (1999), Nr. 11, S. 1179–1191.
http://dx.doi.org/10.1109/71.809575. –
DOI 10.1109/71.809575. –
ISSN 1045–9219

[2] Adam, T. L. ; Chandy, K. M. ; Dickson, J. R.:
A comparison of list schedules for parallel processing systems.
In: *Communications of the ACM* 17 (1974), Nr. 12, S. 685–690.
http://dx.doi.org/10.1145/361604.361619. –
DOI 10.1145/361604.361619. –
ISSN 0001–0782

[3] Baker, T. P. ; Shaw, A. C.:
The Cyclic Executive Model and Ada.
In: *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*.
Huntsville, Alabama, USA, Dez. 6–8, 1988, S. 120–129

Literaturverzeichnis (Forts.)

[4] Casavant, T. L. ; Kuhl, J. G.:
A taxonomy of scheduling in general-purpose distributed computing systems.
In: *IEEE Transactions on Software Engineering* 14 (1988), Nr. 2, S. 141–154.
http://dx.doi.org/10.1109/32.4634. –
DOI 10.1109/32.4634. –
ISSN 0098–5589

[5] Kruatrachue, B. ; Lewis, T. G.:
Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems / Oregon State University.
Corvallis, OR, USA, 1976. –
Forschungsbericht

[6] Kwok, Y.-K. ; Ahmad, I. :
Static scheduling algorithms for allocating directed task graphs to multiprocessors.
In: *ACM Computing Surveys* 31 (1999), Nr. 4, S. 406–471.
http://dx.doi.org/10.1145/344588.344618. –
DOI 10.1145/344588.344618. –
ISSN 0360–0300

Literaturverzeichnis (Forts.)

[7] Schild, K. ; Würtz, J. :
Off-Line Scheduling of a Real-Time System.
In: *Proceedings of the 13th ACM Symposium on Applied Computing (SAC '98)*.
New York, NY, USA : ACM, 1998. –
ISBN 0–89791–969–6, S. 29–38

[8] Sih, G. C. ; Lee, E. A.:
A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures.
In: *IEEE Transactions on Parallel and Distributed Systems* 4 (1993), Nr. 2, S. 175–187.
http://dx.doi.org/10.1109/71.207593. –
DOI 10.1109/71.207593. –
ISSN 1045–9219