

# Echtzeitsysteme

Ereignisgesteuerte Ablaufplanung periodischer Echtzeitsysteme

Lehrstuhl Informatik 4

17. November 2011

# Gliederung

- 1 Überblick
- 2 Einplanung
  - Gebräuchliche Verfahren
  - Statische Prioritäten
  - Prioritätsabbildung
  - Dynamische Prioritäten
- 3 Optimalität
  - RM, DM & EDF
  - Ereignisgesteuerte Ablaufplanung
- 4 Planbarkeitsanalyse
  - CPU-Auslastung
  - Zeitbedarfsanalyse
  - Antwortzeitanalyse
  - Simulation
  - Prioritätsabbildung
- 5 Zusammenfassung

# Fragestellungen

- Was sind die **Prioritäten** der ereignisorientierten Einplanung?
  - Welche Kriterien werden auf Prioritäten abgebildet?
  - **Statische** und **dynamische Verfahren** zur Bestimmung von Prioritäten
  - Wie geht man mit einer knappen Anzahl von Systemprioritäten um?
- Optimalität und Nichtoptimalität ereignisgesteuerter Ablaufplanung
  - Wie schlagen sich die vorgestellten Verfahren?
  - Wo liegen die Grenzen ereignisgesteuerter Ablaufplanung?
- Wie beurteilt man die **Planbarkeit ereignisgesteuerter Systeme**?
  - Beurteilung mit Hilfe der **maximalen, kumulativen CPU-Auslastung**
  - Wann werden die einzelnen Jobs fertig gestellt?  $\leadsto$  **Antwortzeitanalyse**
  - Wie wirken sich zu wenige Systemprioritäten aus?

# Gliederung

- 1 Überblick
- 2 **Einplanung**
  - Gebräuchliche Verfahren
  - Statische Prioritäten
  - Prioritätsabbildung
  - Dynamische Prioritäten
- 3 Optimalität
  - RM, DM & EDF
  - Ereignisgesteuerte Ablaufplanung
- 4 Planbarkeitsanalyse
  - CPU-Auslastung
  - Zeitbedarfsanalyse
  - Antwortzeitanalyse
  - Simulation
  - Prioritätsabbildung
- 5 Zusammenfassung

# Kriterien der Prioritätsvergabe

## Statische Prioritäten

RM (engl. *rate monotonic*)

- je kürzer die **Periode**, desto höher die Priorität

DM (engl. *deadline monotonic*)

- je kürzer der **relative Termin**, desto höher die Priorität

## Dynamische Prioritäten

EDF (engl. *earliest deadline first*)

- je früher der **Termin**, desto höher die Priorität

LRT (engl. *latest release-time first*), EDF umgekehrt  $\rightsquigarrow$  **Eigenstudium**

- je später die **Auslösezeit**, desto höher die Priorität

LST (engl. *least slack-time first*)  $\rightsquigarrow$  **Eigenstudium**

- je kürzer die **Schlupfzeit**, desto höher die Priorität

# RM — *Rate Monotonic*

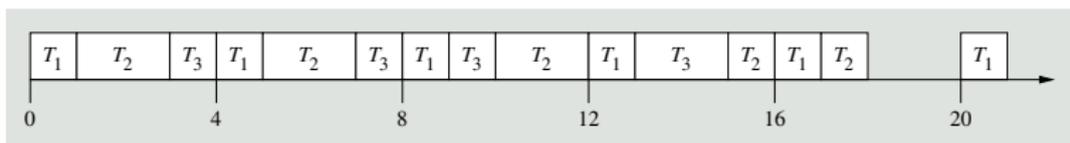
**Rate** einer Aufgabe  $T_i$  ist die Inverse der Periode von  $T_i$

- bezieht sich auf die Auslösung von Arbeitsaufträgen in  $T_i$
- je kürzer die Periode von  $T_i$ , desto höher die Rate von  $T_i$ 
  - desto höher die Priorität von  $T_i$

**Aufgaben**  $T_1 = (4, 1)$ ,  $T_2 = (5, 2)$ ,  $T_3 = (20, 5)$

- bei  $D_i = p_i$  und  $\phi_i = 0$  gibt man  $D_i$  und  $\phi_i$  nicht an
- Perioden  $p_i = \{4, 5, 20\}$
- Ausführungszeiten  $e_i = \{1, 2, 5\}$

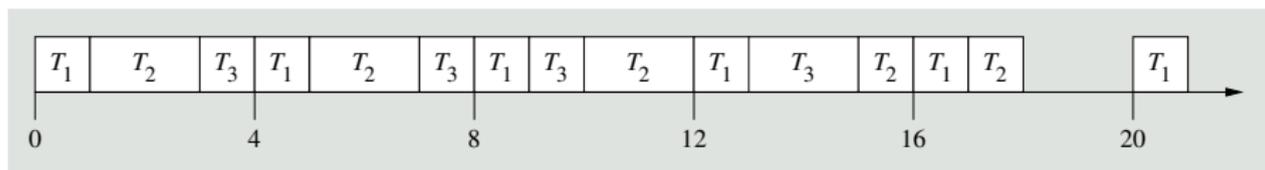
## Ablaufplan



- Arbeitsaufträge werden in ihren Aufgabenperioden ausgeführt
  - lässt den Prozessor nicht untätig, wenn ausführbereite Jobs anstehen

# RM — *Rate Monotonic* (Forts.)

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



$T_1$  hat die höchste Rate (kürzeste Periode) und startet zuerst

- alle Jobs von  $T_1$  werden ausgelöst

$T_2$  hat die zweithöchste Priorität und folgt  $T_1$

- die Jobs von  $T_2$  werden im Hintergrund von  $T_1$  ausgeführt
- der erste Job von  $T_2$  startet nach dem ersten Job von  $T_1$
- $T_2$  wird zum Zeitpunkt  $t = 16$  von  $T_1$  verdrängt

$T_3$  hat die dritthöchste Priorität und folgt  $T_2$

- die Jobs von  $T_3$  laufen im Hintergrund von  $T_1$  und  $T_2$
- $T_3$  läuft nur, wenn kein Job von  $T_1$  und  $T_2$  ausführbereit ist

untätig für Zeitintervall  $[18, 19]$  gibt es keine ausführbaren Jobs mehr

# DM — *Deadline Monotonic*

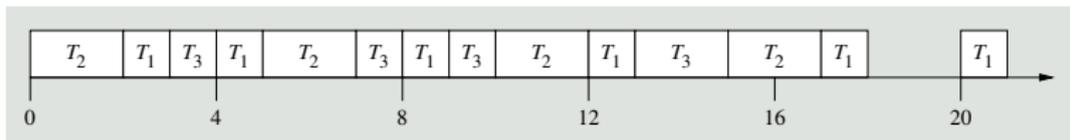
DM = RM wenn gilt:  $D_i = p_i$

- z.B.  $T_1 = (4, 1)$ ,  $T_2 = (5, 2)$ ,  $T_3 = (20, 5)$ 
  - entspricht  $T_1 = (4, 1, 4)$ ,  $T_2 = (5, 2, 5)$ ,  $T_3 = (20, 5, 20)$
  - relativer Termin und Periode jeder Aufgabe sind identisch

**Aufgaben**  $T_1 = (4, 1)$ ,  $T_2 = (5, 2, 3)$ ,  $T_3 = (20, 5)$

- Perioden  $p_i = \{4, 5, 20\}$
- Ausführungszeiten  $e_i = \{1, 2, 5\}$
- relative Termine  $D_i = \{4, 3, 20\}$

## Ablaufplan



- bei beliebigen relativen Terminen arbeitet DM besser als RM
  - d.h., DM liefert zulässige Abläufe in Fällen, wo RM scheitert

# Mehrdeutigkeit von Prioritäten

## Anwendungsebene vs. Systemebene

Echtzeitrechensysteme unterstützen typischerweise nur eine begrenzte Anzahl von Prioritätsebenen:

- 8 im IEEE 802.5 *token ring* [7]
- 32 im alten QNX, im neuen („Neutrino“) 256 [6]
- 140 in Linux 2.5 (mit Ebenen 1–100 reserviert für Echtzeitprozesse)
- 256 in VxWorks [14] und vielen anderen Echtzeitbetriebssystemen
  - **implementierungsbedingter begrenzter Wertebereich**: Bitfeld, char

Echtzeitanwendungen können jedoch mehr Prioritätsebenen benötigen, als die gegebene Systemplattform unterstützt

**uneindeutige Prioritäten** (engl. *nondistinct priorities*) sind die Folge

- die Anzahl unterschiedlicher (eindeutiger) Task-/Jobprioritäten übersteigt die Anzahl unterschiedlicher Prioritäten im System
- die Task-/Jobprioritäten lassen sich nicht eindeutig abbilden

# Prinzip der Prioritätsabbildung

Prioritätsraster (engl. *priority grid*)

- $\omega_n$  Anzahl (an eine Task/einen Job) zugewiesener Prioritäten
- $1, 2, \dots, \omega_n$  mit 1 als höchste und  $\omega_n$  als niedrigste Priorität
- $\omega_s$  Anzahl der Prioritäten des Systems
- $\pi_1, \pi_2, \dots, \pi_{\omega_s}$  mit  $\pi_k$  ( $1 \leq k \leq \omega_s$ ) im Bereich  $[1, \omega_n]$
  - zusätzlich gilt:  $\pi_j < \pi_k$  wenn  $j < k$

Menge  $\{\pi_1, \pi_2, \dots, \pi_{\omega_s}\}$  ist **Prioritätsraster**  $\Pi$ , auf das die zugewiesenen Prioritäten wie folgt abgebildet werden:

- zugewiesene Prioritäten größer gleich  $\pi_1$  auf  $\pi_1$
- zugewiesene Prioritäten im Bereich  $] \pi_{k-1}, \pi_k ]$  auf  $\pi_k$  für  $1 < k \leq \omega_s$

 die Abbildung kann **gleichmäßig** oder **ungleichmäßig** definiert sein

# Abbildung durch gleichmäßige Verteilung

(engl. *uniform mapping*)

Prioritätsraster  $\Pi$  uniform auf den Bereich zugewiesener Prioritäten legen

- sei  $Q$  definiert als Ganzzahl  $\lfloor \omega_n / \omega_s \rfloor$ , dann ist die Systempriorität  $\pi_k = kQ$  für  $k = 1, 2, \dots, \omega_s - 1$  und  $\pi_{\omega_s} = \omega_n$
- das bedeutet für einen Block von max.  $Q$  zugewiesenen Prioritäten:
  - die ersten  $Q$  höchsten  $1, 2, \dots, Q$  werden abgebildet auf  $\pi_1 = Q$
  - die nächsten  $Q$  höchsten werden abgebildet auf  $\pi_2 = 2Q$
  - usw., bis alle zugewiesenen Prioritäten „gerastert“ worden sind
- Jobs werden dann gemäß ihrer Systempriorität  $\pi_k$  abgearbeitet

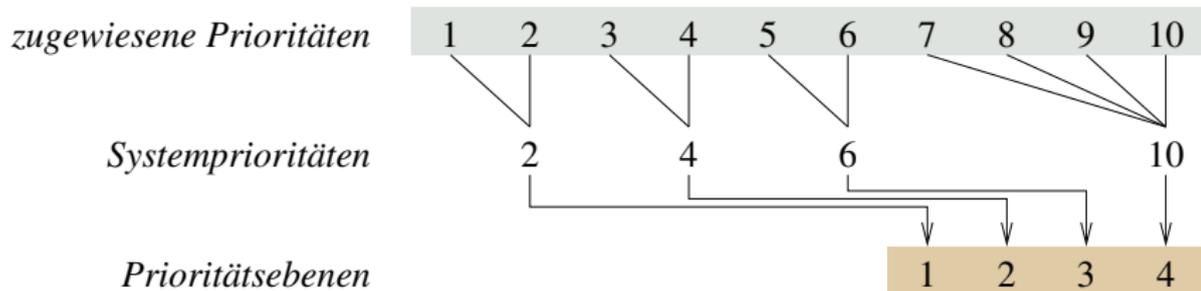
Tasks verschiedener logischer (d.h., zugewiesener) Prioritäten erhalten dieselbe physische Systempriorität, liegen auf einer Prioritätsebene

- die Jobs dieser Tasks sind einer linearen Abbildung unterworfen
- Wichtung erhalten sie durch ihre Position in der linearen Ordnung

# Abbildung durch gleichmäßige Verteilung (Forts.)

Querschneidender Belang von Anwendung und System

**Beispiel:** 10 Tasks mit zugewiesenen Prioritäten  $1, 2, \dots, 10$  und ein System, das nur die Prioritätsebenen  $1, 2, 3, 4$  unterstützt



- $[1, 2] \mapsto \pi_1 = 2$
- $[3, 4] \mapsto \pi_2 = 4$
- $[5, 6] \mapsto \pi_3 = 6$
- $[7, 10] \mapsto \pi_4 = 10$

**Problem „Fairness“:** Tasks höherer zugewiesener Prioritäten werden gleich behandelt wie Tasks mit niedrigeren zugewiesenen Prioritäten.

# Abbildung durch ungleichmäßige Verteilung

(engl. *non-uniform mapping*)

Prioritätsraster  $\Pi$  derart auf den Bereich zugewiesener Prioritäten legen, so dass das Verhältnis  $(\pi_{i-1} + 1)/\pi_i$  für  $i = 2, 3, \dots, \omega_s$  gleich bleibt

- die Methode wird auch als *constant ratio mapping* [8] bezeichnet
- für höhere zugewiesene Prioritäten werden mehr Prioritätsebenen reserviert als für niedrigere zugewiesene Prioritäten
- resultiert in eine bessere Feinabstufung höher priorisierter Tasks

**Beispiel** (wie gehabt, s. Folie IV-2/12):  $\omega_n = 10$ ,  $\omega_s = 4$

- |                                |                                   |                            |
|--------------------------------|-----------------------------------|----------------------------|
| • $[1, 1] \mapsto \pi_1 = 1$   |                                   | • $1:1 \mapsto$ Ebene $_1$ |
| • $[2, 3] \mapsto \pi_2 = 3$   | • $(\pi_1 + 1)/\pi_2 = 2/3$       | • $2:1 \mapsto$ Ebene $_2$ |
| • $[4, 6] \mapsto \pi_3 = 6$   | • $(\pi_2 + 1)/\pi_3 = 2/3$       | • $3:1 \mapsto$ Ebene $_3$ |
| • $[7, 10] \mapsto \pi_4 = 10$ | • $(\pi_3 + 1)/\pi_4 \approx 2/3$ | • $4:1 \mapsto$ Ebene $_4$ |

# EDF — *Earliest Deadline First*

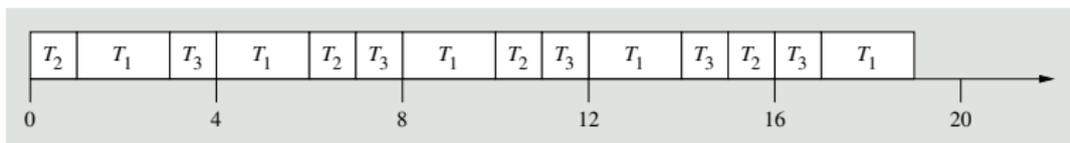
Ordnet Arbeitsaufträge nach ihrem **absoluten Termin**

- je näher der absolute Termin, umso höher die Priorität
- verschiedene Jobs derselben Aufgabe mit unterschiedlicher Priorität

**Aufgaben**  $T_1 = (4, 2)$ ,  $T_2 = (5, 1, 3)$ ,  $T_3 = (20, 5)$

- Perioden  $p_i = \{4, 5, 20\}$
- Ausführungszeiten  $e_i = \{2, 1, 5\}$
- relative Termine  $D_i = \{4, 3, 20\}$

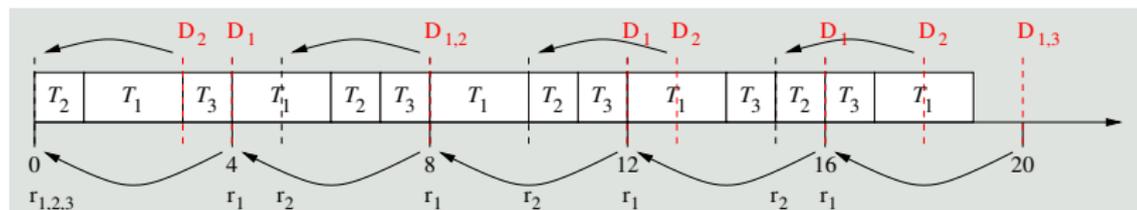
**Ablaufplan**



- Arbeitsaufträge werden möglichst auslösezeitnah gestartet
  - lässt den Prozessor nicht untätig, wenn ausführbare Jobs anstehen

# EDF — Earliest Deadline First (Forts.)

Beispiel:  $T_1 = (4, 2)$ ,  $T_2 = (5, 1, 3)$ ,  $T_3 = (20, 5)$



$T_1 = (4, 2)$

$t_0$  Auslösung,  $D_1 = 4$

$t_1$  Start –  $t_3$  Ende

$t_4$  Auslösung,  $D_1 = 8$ , Start

$t_6$  Ende

$t_8$  Auslösung,  $D_1 = 12$ , Start

$t_{10}$  Ende

$t_{12}$  Auslösung,  $D_1 = 16$ , Start

$t_{14}$  Ende

$t_{16}$  Auslösung,  $D_1 = 20$

$t_{17}$  Start –  $t_{19}$  Ende

$T_2 = (5, 1, 3)$

$t_0$  Auslösung,  $D_2 = 3$ , Start

$t_1$  Ende

$t_5$  Auslösung,  $D_2 = 8$

$t_6$  Start –  $t_7$  Ende

$t_{10}$  Auslösung,  $D_2 = 13$ , Start

$t_{11}$  Ende

$t_{15}$  Auslösung,  $D_2 = 18$ , Start

$t_{16}$  Ende

$T_3 = (20, 5)$

$t_0$  Auslösung,  $D_3 = 20$

$t_3$  Start –  $t_4$  Verdrängung

$t_7$  Fortsetzung

$t_8$  Verdrängung

$t_{11}$  Fortsetzung

$t_{12}$  Verdrängung

$t_{14}$  Fortsetzung

$t_{15}$  Verdrängung

$t_{16}$  Fortsetzung

$t_{17}$  Ende

# LRT — *Latest Release-Time First*

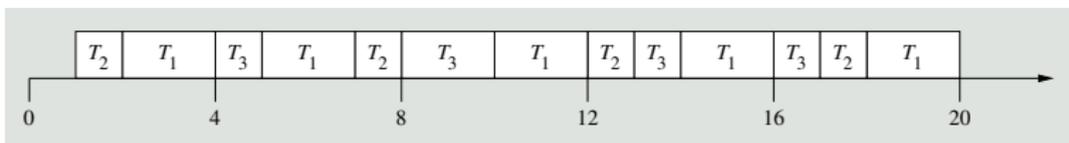
EDF umgekehrt  $\leadsto$  Arbeitsaufträge werden „rückwärts“ eingeplant

- Auslösezeiten sind Termine bzw. Termine sind Auslösezeiten

**Aufgaben**  $T_1 = (4, 2)$ ,  $T_2 = (5, 1, 3)$ ,  $T_3 = (20, 5)$

- Perioden  $p_i = \{4, 5, 20\}$
- Ausführungszeiten  $e_i = \{2, 1, 5\}$
- relative Termine  $D_i = \{4, 3, 20\}$

## Ablaufplan



- Arbeitsaufträge werden möglichst terminnah erfüllt
  - lässt den Prozessor ggf. untätig trotz ausführbereiter Jobs
    - schiebt Jobs mit harten Echtzeitbedingungen nach hinten
    - schafft vorne Spiel für Jobs mit weichen/festen Echtzeitbedingungen

# LST — *Least Slack-Time First*

auch: *Minimum Laxity First*, MLF

Schlupfzeit zum Zeitpunkt  $t$

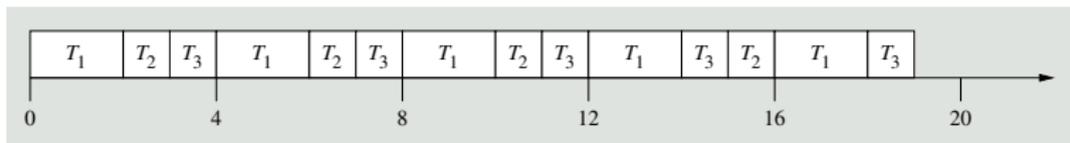
$$\text{slack}(J_i, t) = r_i + D_i - t - \text{maturity}(J_i, t)$$

$$\text{maturity}(J_i, t) = e_i - \text{elapsed time}(J_i, t)$$

**Aufgaben**  $T_1 = (4, 2)$ ,  $T_2 = (5, 1, 3)$ ,  $T_3 = (20, 5)$

- Perioden  $p_i = \{4, 5, 20\}$
- Ausführungszeiten  $e_i = \{2, 1, 5\}$
- relative Termine  $D_i = \{4, 3, 20\}$

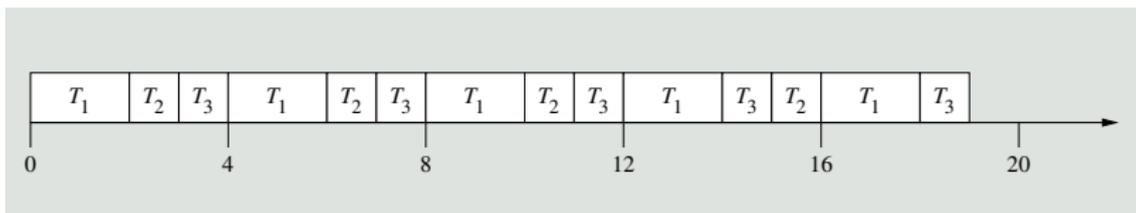
**Ablaufplan**



- benötigt Ausführungszeiten und Termine der Arbeitsaufträge
- Arbeitsaufträge werden möglichst auslösezeitnah gestartet
  - lässt den Prozessor nicht untätig, wenn ausführbare Jobs anstehen

# LST — *Least Slack-Time First* (Forts.)

Beispiel:  $T_1 = (4, 2)$ ,  $T_2 = (5, 1, 3)$ ,  $T_3 = (20, 5)$



	$J_{1,x}$			$J_{2,x}$			$J_{3,x}$		
	Job	Slack	maturity	Job	slack	maturity	Job	slack	maturity
$t_0$	$J_{1,1}$	2	0	$J_{2,1}$	2	0	$J_{3,1}$	15	0
$t_4$	$J_{1,2}$	2	0	-	-	-		12	1
$t_5$		1	1	$J_{2,2}$	2	0		11	1
$t_8$	$J_{1,3}$	2	0	-	-	-		9	2
$t_{10}$	-	-	-	$J_{2,3}$	2	0		7	2
$t_{12}$	$J_{1,4}$	2	0	-	-	-		6	3
$t_{15}$	-	-	-	$J_{2,4}$	2	0		4	4
$t_{16}$	$J_{1,5}$	2	0	-	-	-		3	4
$t_{18}$	-	-	-	-	-	-		1	4

# Gliederung

- 1 Überblick
- 2 Einplanung
  - Gebräuchliche Verfahren
  - Statische Prioritäten
  - Prioritätsabbildung
  - Dynamische Prioritäten
- 3 Optimalität**
  - RM, DM & EDF
  - Ereignisgesteuerte Ablaufplanung
- 4 Planbarkeitsanalyse
  - CPU-Auslastung
  - Zeitbedarfsanalyse
  - Antwortzeitanalyse
  - Simulation
  - Prioritätsabbildung
- 5 Zusammenfassung

# Optimalität des RM-Algorithmus

Der RM-Algorithmus ist optimal für Systeme, deren Aufgaben

- **synchron** sind (d.h.  $\phi_i = 0$ ) und
- die Voraussetzungen **A1 - A7** erfüllen (siehe IV-1/10).

**Beweisidee** (Baruah [1])

- gegeben sei ein System mit den Aufgaben  $\{T_1, T_2, T_3, \dots, T_n\}$
- mit Prioritäten  $T_1 \succ T_2 \succ \dots \succ T_n$  (nicht RM-konform)
- erzeuge einen zulässigen Ablaufplan
- Prioritäten können hinsichtlich RM umgeformt werden<sup>1</sup>
- ohne die Zulässigkeit des Ablaufplans zu zerstören

---

<sup>1</sup>Man kann die Prioritäten zweier Aufgaben  $T_1$  und  $T_2$ , die das RM-Schema verletzen (für die also  $T_1 \succ T_2$  gilt, obwohl  $p_1 > p_2$ ), tauschen, ohne dabei die Zulässigkeit des Systems zu zerstören.

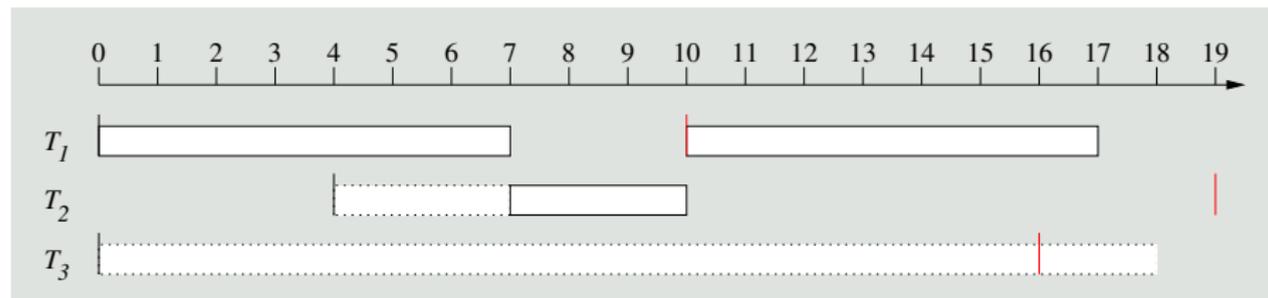
# Nichtoptimalität des RM-Algorithmus

Der RM-Algorithmus ist nicht optimal für Systeme, deren Aufgaben

- **asynchron** sind (d.h.  $\exists \phi_i > 0$ ) und
- die Voraussetzungen **A1 - A7** erfüllen.

**Beweis** (Baruah [1])

- Betrachte  $T_1 = (10, 7, 10, 0)$ ,  $T_2 = (15, 3, 15, 4)$ ,  $T_3 = (16, 1, 16, 0)$
- RM:  $T_1 \succ T_2 \succ T_3$



- $T_3$  verpasst bei  $t_{16}$  seinen Termin

- $T_1 \succ T_3 \succ T_2$  würde funktionieren

# Optimalität des DM-Algorithmus

Der DM-Algorithmus ist optimal für System, deren Aufgaben

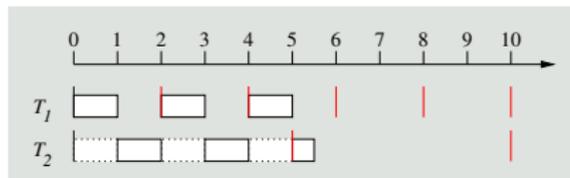
- **synchron** sind,
- die Voraussetzungen **A1**, **A2**, sowie **A4** - **A7** einhalten und
- für deren Termine  $D_i \leq p_i$  gilt.

**Beweisidee** (Baruah [1])

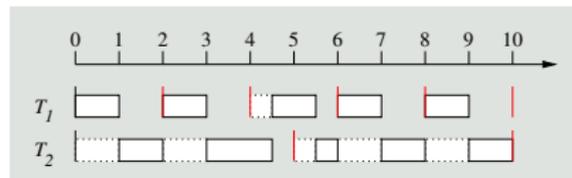
- Analog zum RM-Algorithmus

# Nichtoptimalität statischer Prioritäten

- betrachte  $T_1 = (2, 1)$  und  $T_2 = (5, 2.5)$
- sei  $T_1 \succ T_2$



$t_5$   $T_2$  verpasst Termin



$t_4$   $T_2 \succ T_1$

$t_{10}$  Hyperperiode

- vor dem Zeitpunkt  $t_4$  müsste gelten  $T_1 \succ T_2$
- zum Zeitpunkt  $t_4$  müsste gelten  $T_2 \succ T_1$

☞ Widerspruch zur statischen Vergabe von Prioritäten

# Optimalität des EDF-Algorithmus

Der EDF-Algorithmus ist optimal für Systeme, deren Aufgaben

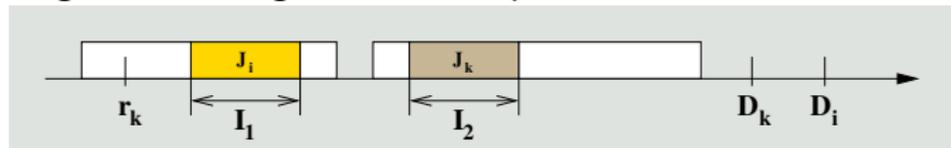
- beliebige Auslösezeiten
    - sporadisch/periodisch
    - synchron/asynchron
  - und
  - beliebige Deadlines
    - länger oder
    - kürzer als die entsprechende Periode
- besitzen, sowie
- die Voraussetzungen **A2** und **A4 - A7** erfüllen.

**Beweis** (Liu [10, S. 67])

- Jeder **zulässige** Ablaufplan für solche Systeme
- lässt sich in einen EDF-Ablaufplan umformen.

# EDF: Ablaufplanherleitung durch Umformung

Gegeben sei folgender Ablaufplan:



- betrachte alle Paare von Arbeitsaufträgen  $J_i$  und  $J_k$
- Arbeitsauftrag  $J_i$  wird im Intervall  $I_1$ ,  $J_k$  im Intervall  $I_2$  eingeplant
- der Termin von  $J_k$  sei vor dem Termin von  $J_i$ :  $D_k < D_i$
- das Intervall  $I_1$  liegt komplett vor  $I_2$ :  $I_1 < I_2$

Fall 1:  $r_k > I_1$

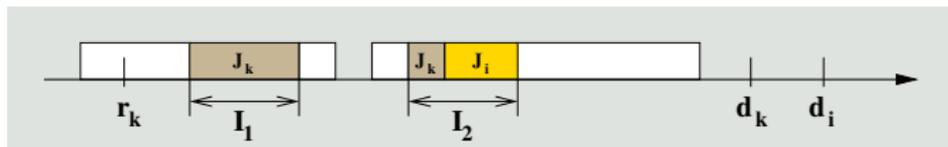
- $J_k$  kann nicht in  $I_1$  eingeplant werden
- der Ablaufplan hat bereits EDF-Form

## EDF: Ablaufplanherleitung durch Umformung (Forts.)

Fall 2:  $r_k < l_1$  ohne Beschränkung der Allgemeinheit

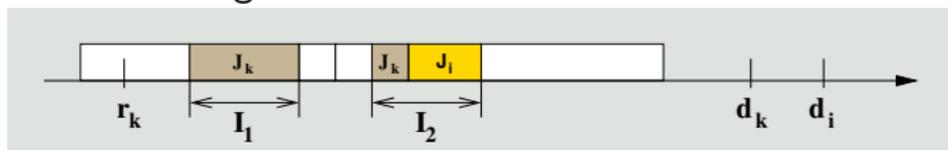
- ① tausche  $J_i$  und  $J_k$

Fall 2a:  $d(l_1) < d(l_2)$   $J_k$  passend stückeln (Verdrängung!)



Fall 2b:  $d(l_1) \geq d(l_2)$  trivial

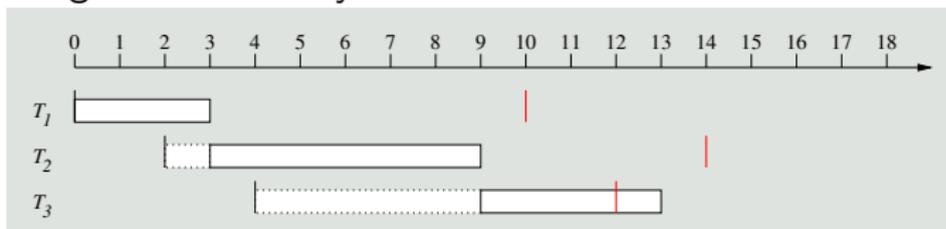
- ② verbliebene Ruheintervalle durch Verschiebung von Arbeitsaufträgen auffüllen



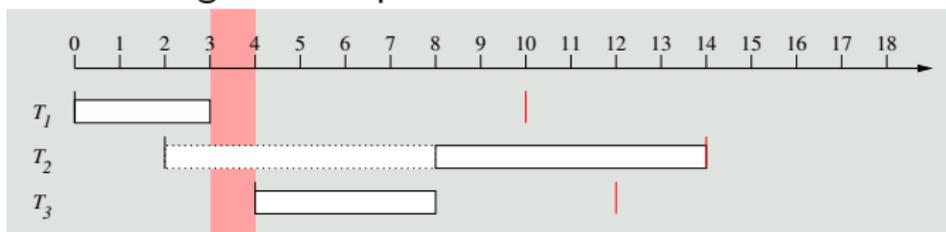
# Nichtoptimalität ereignisgesteuerter Ablaufplanung

Beliebige (in diesem Fall nicht-verdrängbare) Aufgaben

- betrachte  $T_1 = (10, 3, 10, 0)$ ,  $T_2 = (14, 6, 14, 2)$ ,  $T_3 = (12, 4, 12, 4)$
- EDF versagt bei diesem System



- obwohl ein zulässiger Ablaufplan existiert



- dieser lässt allerdings den Prozessor kurz untätig

 Der Plan wird von keinem vorranggesteuerten Algorithmus gefunden!

# Gliederung

- 1 Überblick
- 2 Einplanung
  - Gebräuchliche Verfahren
  - Statische Prioritäten
  - Prioritätsabbildung
  - Dynamische Prioritäten
- 3 Optimalität
  - RM, DM & EDF
  - Ereignisgesteuerte Ablaufplanung
- 4 **Planbarkeitsanalyse**
  - CPU-Auslastung
  - Zeitbedarfsanalyse
  - Antwortzeitanalyse
  - Simulation
  - Prioritätsabbildung
- 5 Zusammenfassung

# Aufgabenstellung

Gegeben sei eine Menge periodischer Aufgaben  $T_i = (p_i, e_i, D_i, \phi_i)$  mit

- $p_i$  der Periode
- $e_i$  der maximalen Ausführungszeit
- $D_i$  dem relativen Termin
- $\phi_i$  der Phase

der jeweiligen Aufgabe.

## Fragestellung:

Ist diese Menge von Aufgaben **zulässig**?

# Planbarkeitsanalyse

Verschiedene Analysemethoden stehen zur Auswahl

## CPU-Auslastung (engl. *loading factor*)

- Zu welchem Prozentsatz wird der Prozessor **maximal** beansprucht?
- bevorzugte Methode für **dynamische Prioritäten**

## Zeitbedarfsanalyse (engl. *processor demand*) $\leadsto$ Eigenstudium

- Wieviel Rechenzeit wird innerhalb eines Zeitintervalls benötigt?
- neuere Methode für **dynamische Prioritäten**

## Antwortzeitanalyse (engl. *response time analysis*)

- Wie lange benötigt eine Aufgabe **maximal** bis zur Fertigstellung?
- präzise Methode für **statische Prioritäten**

## Simulation (engl. *simulation*) $\leadsto$ Eigenstudium

- Wird in einem bestimmten Intervall eine Deadline verfehlt?
- bevorzugte Methode für **statische Prioritäten**

## CPU-Auslastung (engl. *loading factor*)

Die **CPU-Auslastung**  $u_{[t_1, t_2[}$  einer Menge von Arbeitsaufträgen während eines Intervalls  $[t_1, t_2[$ , ist der Anteil des **Rechenzeitbedarfs**  $h_{[t_1, t_2[}$ , der nötig ist, um diese Arbeitsaufträge auszuführen:

$$u_{[t_1, t_2[} = \frac{h_{[t_1, t_2[}}{t_2 - t_1}$$

Für eine Aussage über die Zulässigkeit einer Menge von Aufgaben  $T$  ist die **absolute CPU-Auslastung** (engl. *absolute loading factor*) von Bedeutung.

Dies ist die

- maximale CPU-Auslastung
- über alle Intervalle  $[t_1, t_2[$

$$u = \max_{0 \leq t_1 < t_2} u_{[t_1, t_2[}$$

## Rechenzeitbedarf (engl. *processor demand*)

Der **Rechenzeitbedarf** einer Menge von Aufgaben  $T$  in einem Zeitintervall  $[t_1, t_2[$ :

$$h_{[t_1, t_2[} = \sum_{t_1 \leq r_k, D_k \leq t_2} e_k$$

Das ist die maximale Ausführungszeit aller Arbeitsaufträge, deren

- Auslösezeitpunkt und
- absoluter Termin

innerhalb dieses Intervalls liegt.

# Zulässigkeitstest für EDF

## Liu und Layland [9]

Für jede Menge von  $n$  synchronen, periodischen Aufgaben, die den Kriterien **A1** - **A7** entsprechen, findet der EDF Algorithmus einen zulässigen Ablaufplan, **gdw** für die CPU-Auslastung gilt:

$$U = \sum_{i=1}^n u_i = \sum_{i=1}^n \frac{e_i}{p_i} \leq 1$$

- Coffman zeigt dies auch für asynchrone Aufgaben [5]
- schließlich zeigt Spuri [13] die „Optimalität“ des EDF-Algorithmus
  - Aufgaben wie oben
  - synchron oder asynchron
  - Kriterium:  $U \leq 1$
- analoge Tests existieren auch für RMA und DMA [10, S. 146]

# Ablaufplanungsprobleme und ihre Berechnungskomplexität

Systeme, die den Bedingungen **A1 - A7** genügen, sind in polynomieller Zeit analysierbar. Die Lockerung dieser Einschränkungen haben jedoch tiefgreifende Konsequenzen:

- verzichtet man auf **A3**  $\rightsquigarrow$  **stark  $\mathcal{NP}$ -hart** (Baruah [2])
  - Termine sind **kürzer** als die Perioden der Aufgaben.
- verzichtet man auf **A4**  $\rightsquigarrow$  **stark  $\mathcal{NP}$ -hart** (Richard [12])
  - Aufgaben **legen sich schlafen** (engl. *self-suspension*).
- verzichtet man auf **A5**  $\rightsquigarrow$  **stark  $\mathcal{NP}$ -hart** (Mok [11])
  - Der **gegenseitige Ausschluss** wird durch Semaphore gesichert.
- verzichtet man auf **A7**  $\rightsquigarrow$  **stark  $\mathcal{NP}$ -hart** (Cai [4])
  - Harmonische, periodische Aufgaben sind **nicht verdrängbar**.

 Dies hat auch Auswirkungen auf die Zulässigkeitstests!

# Beliebige Termine und Perioden

Bedingung A3 (S. IV-1/10) soll gelockert werden

$$D_i \geq p_i$$

- die Kriterien von Layland/Liu und Coffman gelten nach wie vor [3]
- diese Kriterien sind **notwendig** und **hinreichend**

$$D_i < p_i$$

## Baruah [3]

Für eine hybride Menge von  $n$  Aufgaben  $T$ , findet der EDF-Algorithmus einen zulässigen Ablaufplan, wenn gilt:

$$U = \sum_{i=1}^n \frac{e_i}{\min\{D_i, p_i\}} \leq 1$$

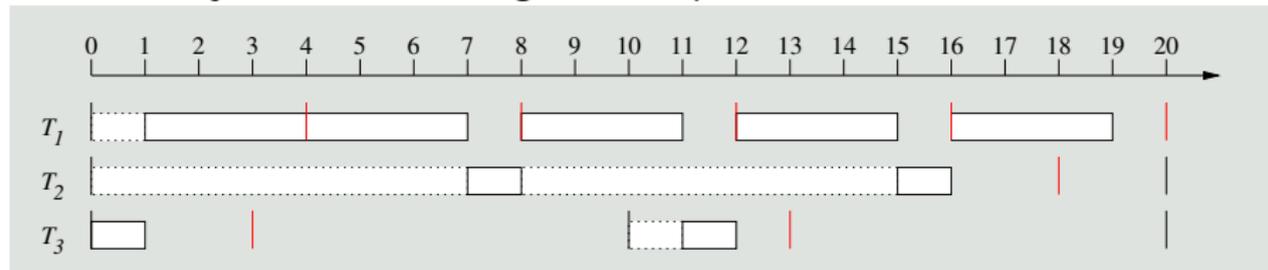
- **hybride** Menge von Aufgaben: periodische und sporadische Aufgaben
- diese Kriterium ist **nur hinreichend!**

## Dieser Test ist pessimistisch ...

Betrachte folgende Aufgaben:  $T_1 = (4, 3, 4, 0)$ ,  $T_2 = (20, 2, 18, 0)$ ,  
 $T_3 = (10, 1, 3, 0)$

- $\sum_i \frac{e_i}{\min\{D_i, p_i\}} = \frac{3}{4} + \frac{2}{18} + \frac{1}{3} = \frac{43}{36} > 1$
- das System ist laut des Tests (s. Folie IV-2/35) **nicht zulässig!**

Es existiert jedoch ein zulässiger Ablaufplan:



# Maximierung des Rechenzeitbedarfs

- hybrides System  $\leadsto$  entsprechendes synchrones, periodisches System
  - alle sporadischen Aufgaben
    - haben Phase  $\phi_i = 0$
    - treten mit ihrer maximalen Frequenz auf
- der Rechenzeitbedarf solcher Systeme ist im Intervall  $[0, t[$  maximal
  - man kann zeigen:  $\forall t_1, t_2 : h_{[t_1, t_2[} \leq h_{[0, t_2 - t_1[}$
- der Rechenzeitbedarf im Intervall  $[0, t[$  ist:

$$h(t) = \sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t - D_i}{p_i} \right\rfloor\right) e_i$$

- alle Arbeitsaufträge, die vor  $t$  beendet sein müssen
- multipliziert mit der maximalen Anzahl ihrer Aktivierungen

# Zulässigkeitstest

Der EDF-Algorithmus, erzeugt für jede hybride Menge von Aufgaben einen zulässigen Ablaufplan, **gdw**:

$$\forall t : h(t) \leq t$$

- entspricht direkt dem Satz von Spuri (S. Folie IV-2/33)
- ist als Kriterium aber so nicht brauchbar
  - schließlich gibt es unendlich viele Intervalle  $[0, t[$
  - alle zu überprüfen ist einfach unmöglich

 Einschränkung der zu überprüfenden Intervalle

# Tätigkeitsintervalle

## Liu und Layland [9]

Kann der EDF-Algorithmus für eine Menge periodischer Aufgaben keinen zulässigen Ablaufplan finden, so wird ein Termin im ersten Tätigkeitsintervall verpasst.

- innerhalb eines Tätigkeitsintervall ist der Prozessor nie untätig
  - eine Phase kontinuierlicher Prozessorauslastung
- diese Eigenschaft wurde später auch gezeigt für
  - Mengen synchroner, periodischer Aufgaben mit  $D_i \leq p_i$  und
  - generische Mengen synchroner, periodischer Aufgaben
- sei  $L$  nun die Länge des ersten Tätigkeitsintervalls
  - die maximale Länge des zu prüfenden Intervalls ist nun beschränkt
- $h(t) \leq t$  muss nicht für alle Zeitpunkte in  $[0, t[$  geprüft werden
  - $\{e_1, e_2, \dots\} = mp_i + D_i; i = 1 \dots n, m = 0, 1, \dots$
  - wobei alle  $e_i < L$  genügen
  - Zeitbedarf erhöht sich nur bei Auslösung eines Arbeitsauftrags

# Ansatz

**Antwortzeit  $\omega_j$**  ● Zeitdauer zwischen Auslösezeit und Terminationszeitpunkt (s. Folie III-2/28)

**Idee** ● Terminationszeitpunkt vor dem **absoluten Termin**  
● Antwortzeit  $\omega_j$  kürzer als der **relative Termin  $D_j$**   
● für jeden Job  $J_j$  in der Aufgabe  $T_j : \omega_j \leq D_j$

**Voraussetzungen** ● Bedingungen **A1 - A7** müssen eingehalten werden  
● Konzept ist jedoch erweiterbar

## Probleme

- Wie berechnet man die Antwortzeit?
- Wann wird die maximale Antwortzeit erreicht?

# Berechnung der Antwortzeit

- die Antwortzeit  $\omega_i$  der Aufgabe  $T_i$  berechnet sich zu

$$\omega_i(t) = e_j + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k; 0 < t \leq p_i$$

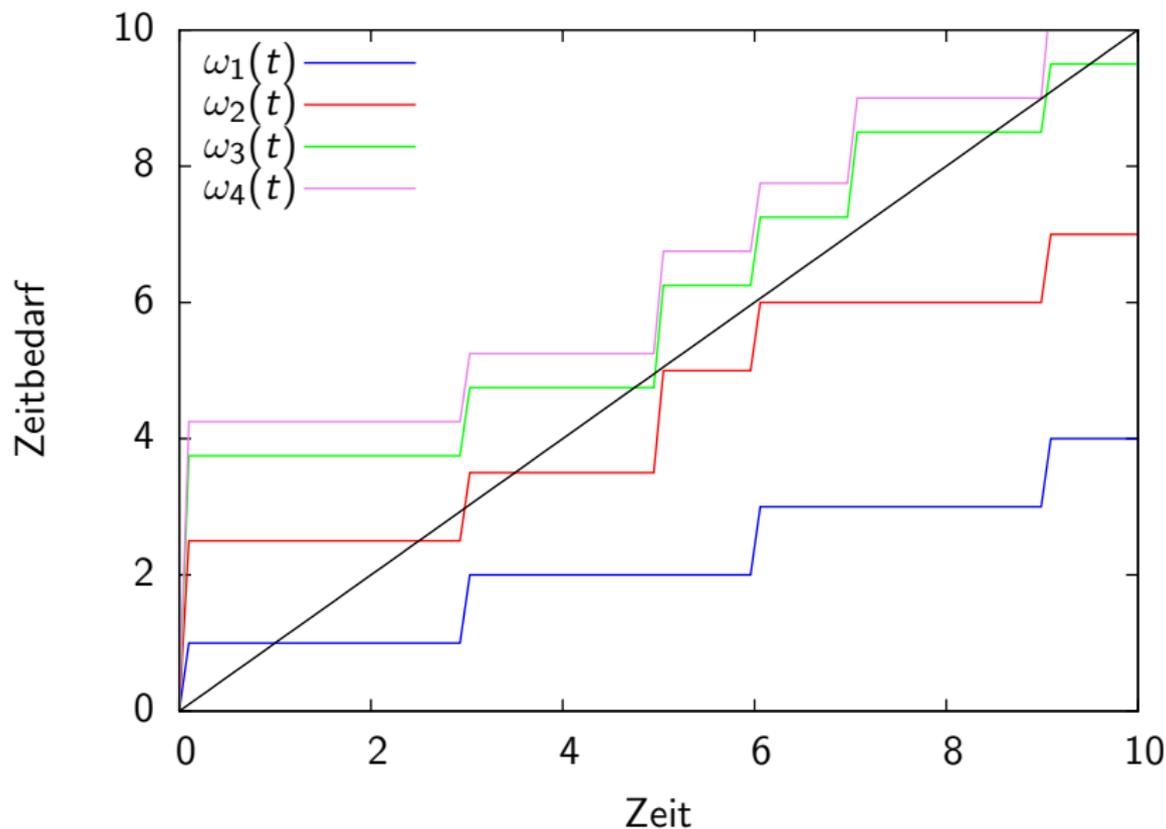
- die Aufgabe endet, bevor das Ereignis erneut eintritt
- setzt sich zusammen, aus
  - der WCET  $e_i$  von  $T_i$  selbst und
  - den WCETs  $e_1, \dots, e_{i-1}$  der Aufgaben  $T_1, \dots, T_{i-1}$  höherer Priorität $\rightsquigarrow T_i$  wird wiederholt von  $T_1, \dots, T_{i-1}$  verdrängt
- zu prüfen ist nun  $\omega_i(t) \leq t$ 

$$t = jp_k; \quad k = 1, 2, \dots, i; \quad j = 1, 2, \dots, \lfloor \min(p_i, D_i) / p_k \rfloor$$
  - Zeitbedarf erhöht sich nur bei Auslösung dringlicherer Aufgaben
  - bis das Ereignis erneut eintritt/der Termin der Aufgabe erreicht ist
  - ist die Ungleichung für **einen** Zeitpunkt  $t$  erfüllt, ist  $T_i$  **zulässig**

# Beispiel: Berechnung der maximalen Antwortzeit

Aufgaben:  $T_1 = (3, 1, 3, \phi_1)$ ,  $T_2 = (5, 1.5, 5, \phi_2)$ ,  $T_3 = (7, 1.25, 7, \phi_3)$ ,  $T_4 = (9, 0.5, 9, \phi_4)$

- Antwortzeit  $\omega_1$  von  $T_1$ 
  - $\omega_1(3) = 1 \leq 3 \rightsquigarrow$  zulässig
- Antwortzeit  $\omega_2$  von  $T_2$ 
  - $\omega_2(3) = 1.5 + \lceil \frac{3}{3} \rceil 1 = 2.5 \leq 3 \rightsquigarrow$  zulässig
- Antwortzeit  $\omega_3$  von  $T_3$ 
  - $\omega_3(3) = 1.25 + \lceil \frac{3}{3} \rceil 1 + \lceil \frac{3}{5} \rceil 1.5 = 3.75 > 3$
  - $\omega_3(5) = 1.25 + \lceil \frac{5}{3} \rceil 1 + \lceil \frac{5}{5} \rceil 1.5 = 4.75 \leq 5 \rightsquigarrow$  zulässig
- Antwortzeit  $\omega_4$  von  $T_4$ 
  - $\omega_4(3) = 0.5 + \lceil \frac{3}{3} \rceil 1 + \lceil \frac{3}{5} \rceil 1.5 + \lceil \frac{3}{7} \rceil 1.25 = 4.25 > 3$
  - $\omega_4(5) = 0.5 + \lceil \frac{5}{3} \rceil 1 + \lceil \frac{5}{5} \rceil 1.5 + \lceil \frac{5}{7} \rceil 1.25 = 5.25 > 5$
  - $\omega_4(6) = 0.5 + \lceil \frac{6}{3} \rceil 1 + \lceil \frac{6}{5} \rceil 1.5 + \lceil \frac{6}{7} \rceil 1.25 = 6.75 > 6$
  - $\omega_4(7) = 0.5 + \lceil \frac{7}{3} \rceil 1 + \lceil \frac{7}{5} \rceil 1.5 + \lceil \frac{7}{7} \rceil 1.25 = 7.75 > 7$
  - $\omega_4(9) = 0.5 + \lceil \frac{9}{3} \rceil 1 + \lceil \frac{9}{5} \rceil 1.5 + \lceil \frac{9}{7} \rceil 1.25 = 9.00 \leq 9 \rightsquigarrow$  zulässig

Zeitbedarfsfunktionen der Aufgaben  $T_1$ ,  $T_2$ ,  $T_3$  und  $T_4$ 

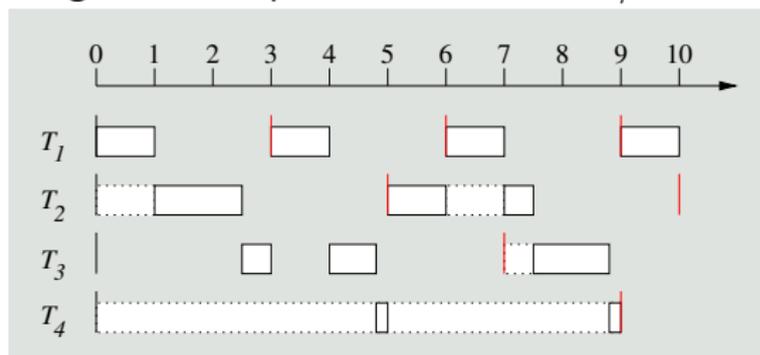
# Wann wird die Antwortzeit maximal?

- **kritischer Zeitpunkt** (engl. *critical instant*)  $\leadsto$  **maximale Antwortzeit**
  - Auslösung eines Arbeitsauftrags an seinem kritischen Zeitpunkt
- der an seinem kritischen Zeitpunkt ausgelöste Job  $J_i$  einer Task  $T_i$ 
  - $\leadsto$  hat die **maximale Antwortzeit** aller Jobs in  $T_i$ 
    - falls diese ihre Termine einhalten
  - $\leadsto$  **verpasst seinen Termin**
    - falls irgendein Arbeitsauftrag in  $T_i$  seinen Termin verpasst
- Systeme mit **statischen Prioritäten**
  - Liu und Layland [9]: ein kritischer Zeitpunkt liegt vor,
  - $\leadsto$  falls **zusammen** mit einem Arbeitsauftrag der Aufgabe  $T_i$
  - $\leadsto$  Jobs aller Aufgaben **höherer Priorität**  $T_1, \dots, T_{i-1}$  ausgelöst werden
- In Systemen mit **dynamischen Prioritäten**
  - lässt sich ein solcher kritischer Zeitpunkt **nicht** identifizieren,
  - weshalb die Antwortzeitanalyse hier **ungeeignet** ist.

# Simulation

- Vorteil**
- Analysemethoden: **komplex** und schwer verständlich
  - Planungsalgorithmen: relativ **einfach**
  - Konstruktion eines Ablaufplans!
- Voraussetzung**
- Simulation muss den *worst case* treffen
- Lösung**
- Simulation muss am kritischen Zeitpunkt beginnen

Vergleiche Beispiel auf s. Folie IV-2/42



☞ Methode, die in vielen industriellen Werkzeugen vorzufinden ist

# Relative Planbarkeit

Einfluss der Anzahl von Systemprioritäten auf die Planbarkeit eines Systems

Verschlechterung der Planbarkeit ist zu erwarten, wenn insgesamt zu wenig Systemprioritäten zur Verfügung stehen, d.h., wenn gilt:  $\omega_n > \omega_s$

- sei  $g$  das Minimum von Verhältniswerten des Prioritätsrasters
  - d.h.,  $g = \min_{2 \leq i \leq \omega_s} (\pi_{i-1} + 1) / \pi_i$  (s. Folie IV-2/13)
- im Falle von RM für große  $n$  und  $D_i = p_i$  für alle  $i$  wurde gezeigt [8], dass für die **planbare Auslastung** (engl. *schedulable utilization*) gilt:
 
$$\ln(2g) + 1 - g \text{ falls } g > 1/2$$

$$g \text{ falls } g \leq 1/2$$
- das Verhältnis dieser Auslastung zu  $\ln(2)$  ist ein Maß für die **relative Planbarkeit** des gegebenen Systems

**Beispiel:** 100 000 Tasks (evtl. noch vielmehr Jobs), d.h.,  $\omega_n = 100\,000$

- die relative Planbarkeit bei  $\omega_s = 256$  ist gleich 0.9986

 **RM:** für komplexeste Taskssysteme reichen schon **256** Prioritätsebenen

# Gliederung

- 1 Überblick
- 2 Einplanung
  - Gebräuchliche Verfahren
  - Statische Prioritäten
  - Prioritätsabbildung
  - Dynamische Prioritäten
- 3 Optimalität
  - RM, DM & EDF
  - Ereignisgesteuerte Ablaufplanung
- 4 Planbarkeitsanalyse
  - CPU-Auslastung
  - Zeitbedarfsanalyse
  - Antwortzeitanalyse
  - Simulation
  - Prioritätsabbildung
- 5 Zusammenfassung

# Resümee

**Ablaufplanung** gebräuchliche, ereignisgesteuerte Verfahren

- **statische Prioritäten**  $\leadsto$  RM, DM
  - Prioritätsabbildung im Falle nicht ausreichender Systemprioritäten
- **dynamische Prioritäten**  $\leadsto$  EDF

**Optimalität und Nichtoptimalität** von RM, DM und EDF

- Hängt von den Eigenschaften der betrachteten Aufgaben ab
- Nichtoptimalität von statischen Prioritäten und Ereignissteuerung

**Planbarkeitsanalyse** ereignisgesteuerter Ablaufplanungsverfahren

- maximalen, kumulativen CPU-Auslastung und Antwortzeitanalyse
- relative Planbarkeit im Falle nicht ausreichender Systemprioritäten

# Literaturverzeichnis

- [1] *Kapitel 28.*  
In: BARUAH, S. ; GOOSSENS, J. :  
*Scheduling Real-time Tasks: Algorithms and Complexity.*  
Chapman & Hall/CRC, 2004 (Computer and Information Science series)
- [2] BARUAH, S. K. ; ROSIER, L. E. ; HOWELL, R. R.:  
Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor.  
In: *Real-Time Systems Journal* 2 (1990), Nr. 4, S. 301–324.  
<http://dx.doi.org/http://dx.doi.org/10.1007/BF01995675>. –  
DOI <http://dx.doi.org/10.1007/BF01995675>. –  
ISSN 0922–6443
- [3] BARUAH, S. ; MOK, A. ; ROSIER, L. :  
Preemptively scheduling hard-real-time sporadic tasks on one processor.  
(1990), Dez., S. 182–190.  
<http://dx.doi.org/10.1109/REAL.1990.128746>. –  
DOI 10.1109/REAL.1990.128746
- [4] CAI, Y. ; KONG, M. C.:  
Nonpreemptive Scheduling of Periodic Tasks in Uni- and Multiprocessor Systems.  
In: *Algorithmica* 15 (1996), Nr. 6, S. 572–599

# Literaturverzeichnis (Forts.)

- [5] COFFMAN, E. G.:  
*Computer and Job-shop Scheduling Theory*.  
John Wiley & Sons Inc, 1976. –  
ISBN 978-0471163190
  
- [6] HILDEBRAND, D. :  
An Architectural Overview of QNX.  
In: *Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architectures*.  
Seattle, WA, USA, Apr. 27–28, 1992, S. 113–126
  
- [7] IEEE STANDARD 802.5:  
*Token Ring Access Method and Physical Layer Specification*.  
IEEE, New York, 1989
  
- [8] LEHOCZKY, J. P. ; SHA, L. :  
Performance of Real-Time Bus Scheduling Algorithms.  
In: *ACM Performance Evaluation Review* 14 (1986), Mai, Nr. 1, S. 44–55

# Literaturverzeichnis (Forts.)

- [9] LIU, C. L. ; LAYLAND, J. W.:  
Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.  
In: *Journal of the ACM* 20 (1973), Nr. 1, S. 46–61.  
<http://dx.doi.org/http://doi.acm.org/10.1145/321738.321743>. –  
DOI <http://doi.acm.org/10.1145/321738.321743>. –  
ISSN 0004–5411
- [10] LIU, J. W. S.:  
*Real-Time Systems*.  
Prentice-Hall, Inc., 2000. –  
ISBN 0–13–099651–3
- [11] MOK, A. K.:  
*Fundamental design problems of distributed systems for the hard real-time environment*,  
MIT, Diss., 1983
- [12] RICHARD, P. :  
On the complexity of scheduling real-time tasks with self-suspensions on one processor.  
In: *Proceedings. 15th Euromicro Conference on Real-Time Systems (ECRTS 2003)*  
(2003), Jul., S. 187–194

# Literaturverzeichnis (Forts.)

- [13] SPURI, M. :  
*Earliest Deadline Scheduling in Real-Time Systems*, Scuola Superiore S. Anna, Pisa,  
Dissertation, 1996
- [14] WIND RIVER SYSTEMS, INC.:  
*Wind River Homepage*.  
<http://www.windriver.com>,