

# Echtzeitsysteme

## Ablaufsteuerung

2./9. November 2009

# Überblick

## Ablaufsteuerung

Trennung von Belangen

Arbeitsweise

Zeitparameter

Taskmodelle

Gebräuchliche Verfahren

Zusammenfassung

Bibliographie

# Planung des zeitlichen Ablaufs und Abfertigung

## Einplanung (engl. *scheduling*) $\mapsto$ Strategie

- ▶ Erstellung des Ablaufplans von Arbeitsaufträgen
  - ▶ Festlegung einer Einlastungsreihenfolge
- ▶ in Bezug auf die Aufgabenbearbeitung geschieht dies:
  - entkoppelt (engl. *off-line*)  $\leadsto$  statisch, vor Laufzeit
  - gekoppelt (engl. *on-line*)  $\leadsto$  dynamisch, zur Laufzeit<sup>1</sup>

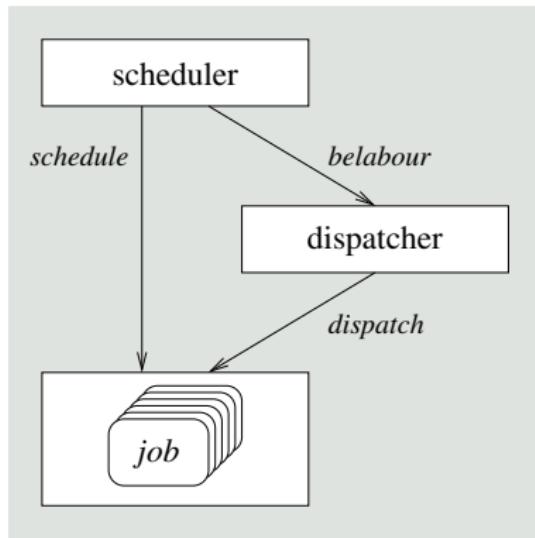
## Einlastung (engl. *dispatching*) $\mapsto$ Mechanismus

- ▶ Abarbeitung des Ablaufplans von Arbeitsaufträgen
  - ▶ Umsetzung der Einplanungsentscheidungen
- ▶ ist immer gekoppelt mit der Aufgabenbearbeitung
  - ▶ Ablaufpläne können nur *online* befolgt werden

---

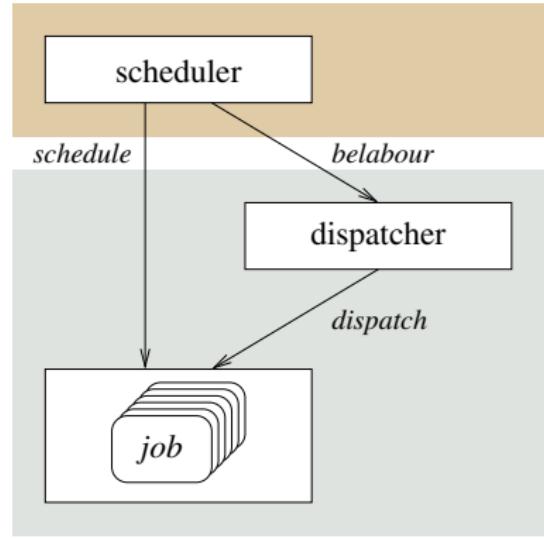
<sup>1</sup>Vorlage ist ggf. ein vor Beginn der Aufgabenbearbeitung statisch erstellter Ablaufplan, der während der Aufgabenbearbeitung dynamisch fortgeschrieben wird.

# Einplanung und Einlastung



gekoppeltes System

- ▶ zeit- und örtlich gekoppelt
  - ▶ zur Laufzeit
  - ▶ integriert in einem System
    - ▶ auf einem Rechner



entkoppeltes System

- ▶ zeit- oder örtlich entkoppelt
  - ▶ vor und zur Laufzeit
  - ▶ separiert in zwei Systeme
    - ▶ ggf. auf zwei Rechner

# Einplanungszeitpunkte

Adaptierbarkeit (engl. *adaptability*) vs. Vorhersagbarkeit (engl. *predictability*)

*on-line scheduling* (zur Laufzeit), kommt ohne *à priori* Wissen aus

- ▶ einzige Option bei unbekannter zukünftiger Auslastung
  - ▶ Lastparameter sind erst zur Joblaufzeit bekannt
- ▶ die getroffenen Entscheidungen sind häufig nur suboptimal
  - ▶ eingeschränkte Fähigkeit, Betriebsmittel maximal zu nutzen
- ▶ ermöglicht/unterstützt jedoch ein *flexibles System*

*off-line scheduling* (vor Laufzeit), benötigt *à priori* Wissen

- ▶ Voraussetzung ist ein *deterministisches System*, d.h.:
  - ▶ alle Lastparameter sind vor Joblaufzeit bekannt
  - ▶ ein fester Satz von Systemfunktionen ist gegeben
- ▶ zur Laufzeit ist kein **NP-schweres Problem** mehr zu lösen
  - ▶ d.h., einen Ablaufplan zu finden, der alle Task/Job-Fristen einhält
- ▶ Änderungen am System bedeuten Neuberechnung des Ablaufplans
  - ▶ dies gilt für alle Änderungen an Software *und* Hardware

# Grundsätzliche Verfahren

Vorangetrieben durch interne oder externe Ereignisse

taktgesteuert (engl. *clock-driven*, auch *time-driven*)

- ▶ Einlastung nur zu festen Zeitpunkten
  - ▶ vorgegeben durch das Echtzeitrechensystem
- ▶ statische (entkoppelte) Einplanung

reihum gewichtet (engl. *weighted round-robin*)

- ▶ Echtzeitverkehr in Hochgeschwindigkeitsnetzen
  - ▶ im Koppelnetz (engl. *switched network*)
- ▶ untypisch für die Einplanung von CPU-Jobs

vorranggesteuert (engl. *priority-driven*, auch *event-driven*)

- ▶ Einlastung zu Ereigniszeitpunkten
  - ▶ vorgegeben durch das kontrollierte Objekt
- ▶ dynamische (gekoppelte) Einplanung



# Taktsteuerung

Zeitgesteuertes (engl. *time-triggered*) System

Einlastungszeitpunkte von Arbeitsaufträgen wurden *à priori* bestimmt

- ▶ alle Parameter aller Arbeitsaufträge sind *off-line* bekannt
  - ▶ WCET, Betriebsmittelbedarf (z.B. Speicher, Fäden, Energie), ...
- ▶ zur Laufzeit anfallende Verwaltungsgemeinkosten sind minimal

Einlastung der Arbeitsaufträge erfolgt in variablen oder festen Intervallen

- ▶ im variablen Fall wird ein Zeitgeber (engl. *timer*) mit der Länge des jeweils einzulastenden Arbeitsauftrags programmiert  $\mapsto$  WCET
  - ▶ jeder Zeitablauf bewirkt eine asynchrone Programmunterbrechung
  - ▶ als Folge findet die Einlastung des nächsten Arbeitsauftrags statt
- ▶ im festen Fall liefert der Zeitgeber regelmäßige Unterbrechungen
  - ▶ ein festes Zeitraster liegt über die Ausführung der Arbeitsaufträge
  - ▶ dient z.B. dem Abfragen (engl. *polling*) von Sensoren/Geräten

# Vorrangsteuerung

Ereignisgesteuertes (engl. *event-triggered*) System

Einplanung und Einlastung laufen gekoppelt ab  $\mapsto$  Ereigniszeitpunkte

- ▶ asynchrone Programmunterbrechungen: Hardwareereignisse
  - ▶ Zeitsignal, Bereitstellung von Sensordaten, Beendigung von E/A
- ▶ Synchronisationspunkte: ein-/mehrseitige Synchronisation
  - ▶ Schlossvariable, Semaphor, Monitor

Ereignisse haben Prioritäten, die Dringlichkeiten zum Ausdruck bringen

- ▶ Prioritäten werden *off-line* vergeben und ggf. *on-line* fortgeschrieben
  - ▶ Arbeitsaufträge haben eine statische oder dynamische Priorität
- ▶ die Zuteilung von Betriebsmitteln erfolgt prioritätsorientiert
  - ▶ Arbeitsaufträge höherer Priorität haben Vorrang
- ▶ Betriebsmittel (insb. CPU) bleiben niemals absichtlich ungenutzt
  - ▶ im Gegensatz zur Taktsteuerung, die Betriebsmittel brach liegen lässt

# Punkte auf der Echtzeitachse

## Bereitstellung und Erfüllung

**Auslösezeit** (engl. *release time*) Zeitpunkt, zu dem ein Arbeitsauftrag zur Ausführung bereitgestellt wird

- ▶ ab dem Zeitpunkt ist Einlastung des betreffenden Jobs möglich
  - ▶ vorausgesetzt, gewisse Abhängigkeitsbedingungen<sup>2</sup> sind erfüllt
- ▶ ggf. verzögern Einplanung/Koordinierung die Einlastung des Jobs

**Termin** (engl. *deadline*) Zeitpunkt, zu dem ein Arbeitsauftrag seine Ausführung beendet haben soll bzw. muss

- ▶ wird differenziert nach der Art seines Bezugszeitpunktes:
  - ▶ **absolut** (engl. *absolute deadline*) als Echtzeit oder
  - ▶ **relativ** (engl. *relative deadline*) zur Auslösezeit
- ▶ ist je nach Anforderung, weich, fest oder hart
- ▶ gibt mit dem Wert  $\infty$  keine Frist für den betreffenden Job vor

---

<sup>2</sup>Daten-/Kontrollfluss bzgl. kontrolliertes Objekt bzw. anderer Arbeitsaufträge.

# Intervalle auf der Echtzeitachse

## Ausführung und Freiraum

**Antwortzeit** (engl. *response time*) Zeitdauer zwischen Auslösezeit und dem Terminationszeitpunkt eines Arbeitsauftrags

- ▶ die **maximal erlaubte Antwortzeit** wird durch einen **relativen Termin** beschränkt
- ▶ absoluter Termin  $\geq$  Auslösezeit + relativer Termin

**Schlupfzeit** (engl. *slack time*) Zeitdauer zwischen Terminationszeitpunkt und Fristablauf eines sich in Bearbeitung befindlichen Arbeitsauftrags

- ▶ unter der Annahme, dass der Arbeitsauftrag nicht mehr blockiert oder unterbrochen wird

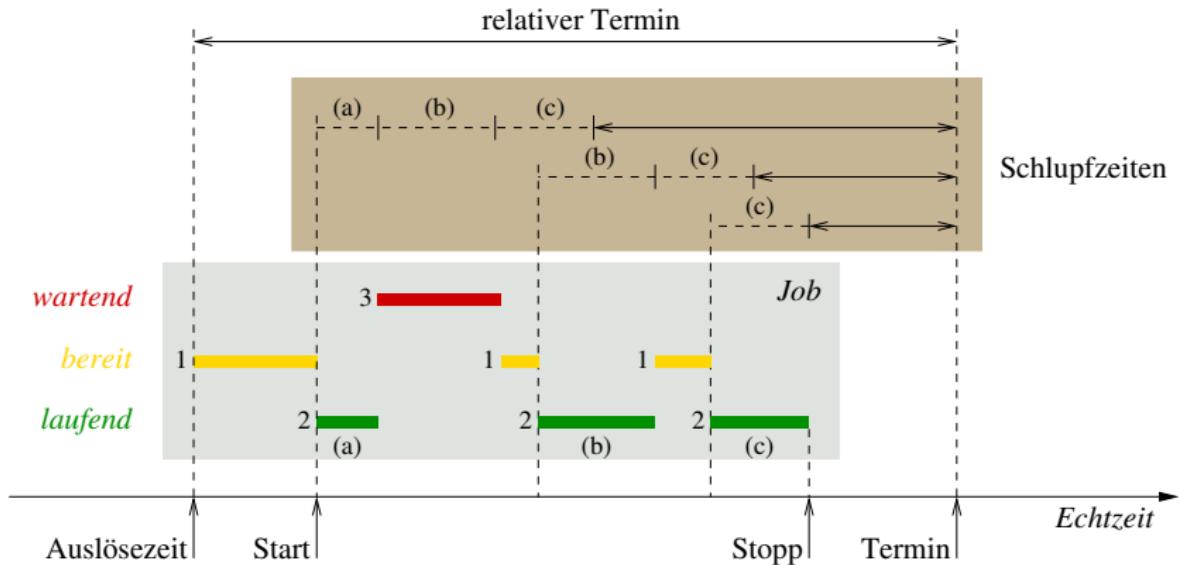
$$\text{slack}(J_i, t) = \text{deadline}(J_i) - t - \text{maturity}(J_i, t)$$

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

- ▶ gibt der Einplanung Spielraum zur Einlastung eines Jobs

# Jobphasen auf der Echtzeitachse

Ablaufzustände eines Fadens



## Arbeitsauftrag einer komplexen Aufgabe

- (1) Einplanung, (2) Einlastung, (3) Synchronisation

# Zufälle auf der Echtzeitachse

## Asynchrone Ereignisse

**sporadische Auslösezeit** (engl. *sporadic release-time*) Auslösezeit eines Arbeitsauftrags **zu einem externen Ereignis**

- ▶ Eintrittszeitpunkte externer Ereignisse ergeben sich zufällig
  - ▶ Auslösezeiten ereignisbehandelnder Jobs sind im Voraus unbekannt
- ▶ Folge: Schwankungen der Auslösezeit (engl. *release-time jitter*)
  - ▶ die Auslösezeit  $r_i$  liegt im Bereich  $[r_i^-, r_i^+]$
  - ▶  $r_i^-$  ist die früheste,  $r_i^+$  ist die späteste Auslösezeit
- ▶ die Arbeitsaufträge laufen sporadisch oder aperiodisch ab

*Beispielsweise kann ein Flugzeugführer das Autopilotensystem jederzeit abschalten. Wenn dies geschieht, wechselt das Autopilotensystem vom Reiseflug- in den Bereitschaftsbetrieb. Die Arbeitsaufträge, die diesen Betriebswechsel ausführen, sind sporadische Arbeitsaufträge. [1, S. 38]*

# Periodische Aufgabe (engl. *periodic task*)

## Vorabwissen

Aufgaben, die in (halb-) regelmäßigen Zeitintervallen kontinuierlich eine vorgegebene Systemfunktion erbringen<sup>3</sup>

- ▶ jede periodische Aufgabe  $T_i$  ist eine Abfolge von Arbeitsaufträgen:
  - Periode  $p_i$  von  $T_i$  ist die minimale Länge aller Zeitintervalle zwischen den Auslösezeiten der Jobs in  $T_i$
  - Ausführungszeit  $e_i$  von  $T_i$  ist die maximale Ausführungszeit aller Jobs in  $T_i$
  - Phase  $\phi_i$  von  $T_i$  ist Auslösezeit des ersten Jobs in  $T_i$
- ▶ zu jeder Zeit sind  $p_i$  und  $e_i$  aller periodischen Aufgaben  $T_i$  bekannt
  - ▶ gegeben durch *à priori* Wissen bzw. der WCET jedes einzelnen Jobs

---

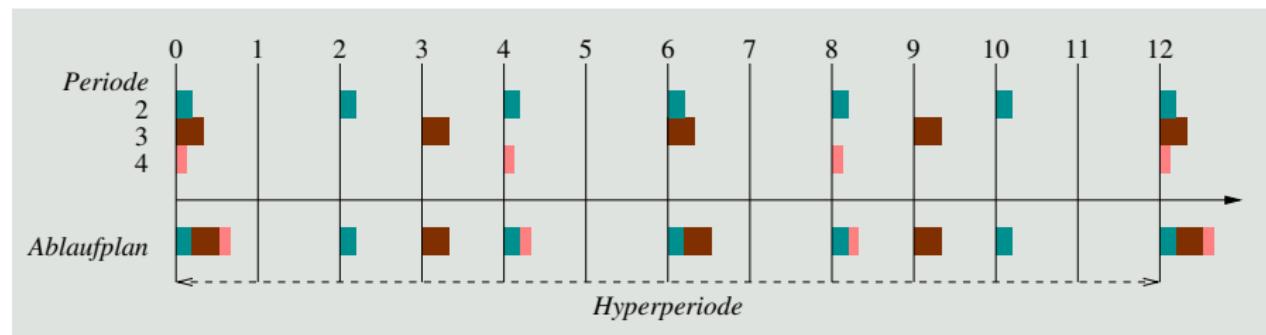
<sup>3</sup>Nach [1] ist eine periodische Aufgabe nicht wirklich periodisch, da die Abstände zwischen den Auslösezeiten (engl. *interrelease time*) eines Arbeitsauftrags einer periodischen Aufgabe nicht der Periode selbst entsprechen müssen. Anderswo werden solche Aufgaben verschiedentlich als sporadische Aufgaben bezeichnet.

# Hyperperiode

Mix verschiedener Aufgaben mit unterschiedlichen Perioden

Zeitintervall, in dem alle periodischen Aufgaben (mindestens einmal) durchgelaufen sind und erneut zusammen zur Ausführung anstehen:

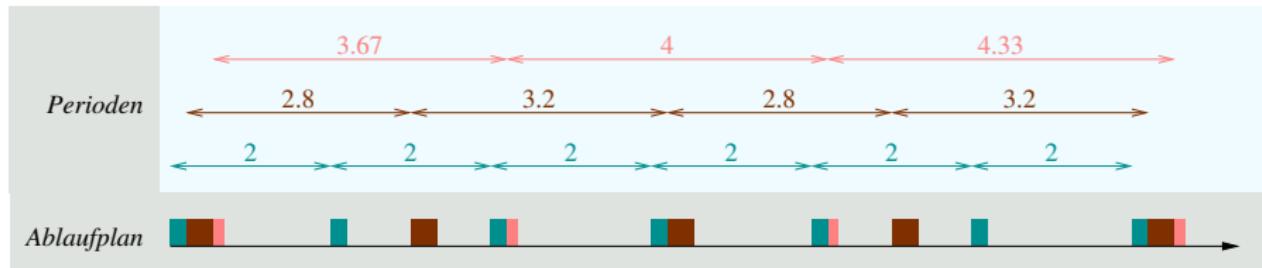
- ▶ Hyperperiode  $H$ , das kleinste gemeinsame Vielfache aller Perioden
- ▶ führt ggf. zu **Schwankungen in den Einlastungszeiten**
  - ▶ ein Problem bei Monoprozessorsystemen  $\leadsto$  geschickte Einplanung



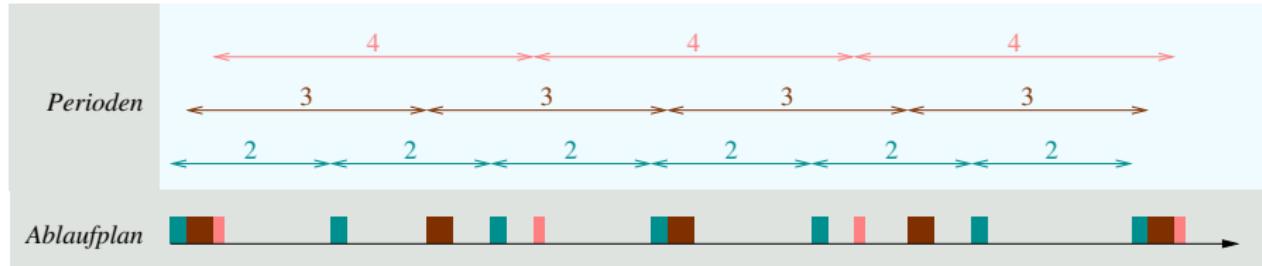
- ▶ maximale Anzahl aller Arbeitsaufträge in  $H$  ist  $\sum_{i=1}^n H/p_i$ 
  - ▶ im vorliegenden Beispiel:  $(12/2) + (12/3) + (12/4) = 13$

# Genauigkeit periodischer Aufgaben

Einfluss der Einplanung auf Schwankungen in der Einlastung



- ▶ bis auf Periode 2 sind alle anderen Jobs nicht wirklich periodisch



- ▶ alle Jobs laufen wirklich periodisch ab: Jobabstand = Periode

# Belegungszeit (engl. *busy time*)

Zeitspanne, für die der Prozessor beansprucht wird

Auslastung  $u_i$  (engl. *utilization*) einer Aufgabe  $T_i$ ,  $u_i = e_i/p_i$

- ▶ dabei steht  $T_i$  für eine wirklich periodische Aufgabe (S. 4-12)
  - ▶ sie stellt eine Referenz für die **Auslastungsobergrenze** dar
  - ▶ d.h., die Referenz einer beliebigen, durch  $T_i$  modellierten Aufgabe
- ▶ alle Aufgaben im System definieren die **totale Auslastung**  $U$ 
  - ▶ die Summe der Auslastungen jeder einzelnen Aufgabe

Beispiel: drei periodische Aufgaben ...

- ▶ mit den Ausführungszeiten 1, 1, 3 und den Perioden 3, 4, 10
- ▶ führen zu den Einzelauslastungen 0.33, 0.25 und 0.3
- ▶ resultieren in eine totale Auslastung von 0.88
  - ▶ d.h., sie belegen den Prozessor zu 88 % seiner Zeit

# Aperiodische/Sporadische Aufgabe

Strom aperiodischer/sporadischer Arbeitsaufträge

Abfolge von Arbeitsaufträgen, deren Auslösezeiten im Voraus unbekannt sind, d.h., die Bereitstellung der Jobs geschieht ereignisbedingt:

aperiodisch (engl. *aperiodic task*)

- ▶ Termine der Arbeitsaufträge sind weich oder fest
- ▶ ggf. haben die Arbeitsaufträge auch keine Termine

sporadisch (engl. *sporadic task*)

- ▶ Termine (einiger) der Arbeitsaufträge sind hart

Varianz in der **Zwischenankunftszeit** (engl. *interarrival time*)<sup>4</sup> ggf. nur einiger der Arbeitsaufträge solcher Aufgaben ist typisch

- ▶ sie kann stark schwanken
- ▶ oft sind nur Minimalwert oder statistische Verteilung bekannt

---

<sup>4</sup> auch: (engl. *interrelease time*).

# Aperiodische/Sporadische Aufgabe (Forts.)

## Beispiele

### Aperiodische Arbeitsaufträge

- ▶ für Empfindlichkeitseinstellung eines Radars sollte das System **reagierend** (engl. *responsive*) arbeiten, damit Maßnahmen zwecks Abgleich/Korrektur zeitnah abgeschlossen werden können
- ▶ Reaktionen, die verzögert oder zu spät erfolgen, sind zwar unerfreulich aber tolerierbar
- ▶ beim Mix darf die **Ansprechempfindlichkeit** (engl. *responsiveness*) aperiodischer Jobs nie auf Kosten der sporadischen optimiert werden

### Sporadische Arbeitsaufträge

- ▶ ein Selbststeuerungssystem muss Kommandos nicht nur innerhalb einer bestimmten Zeit annehmen sondern auch darauf reagieren (S. 4-11)
- ▶ ggf. muss es **transiente Fehler** rechtzeitig erkennen und behandeln und die Erholung (engl. *recovery*) in einer Frist abschließen ↳ Fehlertoleranz

# Rangfolge (engl. *precedence*)

Abhängigkeit von Kontrollflüssen

Arbeitsaufträge können gezwungen sein, in einer ganz bestimmten Reihenfolge ausgeführt werden zu müssen

- ▶ Beispiel Radarüberwachungsanlage ...
  - ▶ Signalaufbereitungsauftrag muss vor Nachführerauftrag gelaufen sein
- ▶ Beispiel Kommunikationssystem ...
  - ▶ Sendeauftrag muss vor Empfangsauftrag gelaufen sein
  - ▶ Empfangsauftrag muss vor Bestätigungsauftrag gelaufen sein
- ▶ Beispiel Anfragesystem ...
  - ▶ Eingabeauftrag muss vor Authentifizierungsauftrag gelaufen sein
  - ▶ Authentifizierungsauftrag muss vor Suchauftrag gelaufen sein
  - ▶ Suchauftrag muss vor Ausgabeauftrag gelaufen sein

☞ die Rangfolge ist oft in Datenabhängigkeiten begründet

# Datenabhängigkeit (engl. *data dependency*)

Abhängigkeit von konsumierbaren Betriebsmitteln

Arbeitsaufträge brauchen zum Ablauf ggf. **konsumierbare Betriebsmittel**

- ▶ ihre Anzahl ist (log.) unbegrenzt: Nachrichten, Signale, Interrupts  
Produzent kann beliebig viele davon erzeugen  
Konsument zerstört sie wieder bei Inanspruchnahme

Produzent und Konsument sind voneinander abhängige Entitäten

- ▶ der Konsument vom Produzenten ...
  - ▶ weil ein konsumierbares Betriebsmittel erst bereitgestellt werden muss, um es in Anspruch nehmen zu können
- ▶ der Produzent vom Konsumenten ...
  - ▶ weil konsumierbare Betriebsmittel auf endlich viele wiederverwendbare Betriebsmittel abgebildet werden
  - ▶ weil der Produzent dazu erst ein wiederverwendbares Betriebsmittel anfordern muss, das vom Konsumenten später wieder freizugeben ist
  - ▶ Beispiel: **begrenzter Puffer** (engl. *bounded buffer*)

# Koordinierung (engl. *coordination*)

Abhängigkeiten analytisch/konstruktiv behandeln

durch Einplanung  $\leadsto$  analytische Verfahren

- ▶ Ablaufpläne berücksichtigen Rangfolgen und Datenabhängigkeiten
  - ▶ *à priori Wissen*  $\mapsto$  periodische Aufgaben
- ▶ Arbeitsaufträge laufen komplett durch (engl. *run to completion*)
  - ▶ sie warten weder ex- noch implizit, dürfen jedoch verdrängt werden
  - ▶ ggf. sind nicht-blockierende Betriebssystemschnittstellen gefordert
- ▶ Ergebnis ist ein System von ausschließlich einfachen Aufgaben

durch Kooperation  $\leadsto$  konstruktive Verfahren

- ▶ Synchronisationspunkte in den Programmen explizit machen
  - ▶ d.h., *Zeitsignale austauschen*  $\mapsto$  Semaphor
- ▶ Arbeitsaufträge sind Produzenten/Konsumenten von Ereignissen
  - ▶ interne Ereignisse von anderen Arbeitsaufträgen
  - ▶ externe Ereignisse von den kontrollierten Objekten
- ▶ Ergebnis ist ein System von (ggf. vielen) komplexen Aufgaben

# Verdrängbarkeit

Verschränkung (engl. *interleaving*) von Arbeitsaufträgen

Arbeitsaufträge könn(t)en verschränkt ausgeführt werden, wenn ...

- ▶ der Planer (engl. *scheduler*) dynamisch, ereignisgesteuert arbeitet
- ▶ die Zeitbedingungen (engl. *time constraints*) es erlauben

**Präemptivität** (engl. *preemptivity*), eine Eigenschaft, die in Abhängigkeit von jedem einzelnen Arbeitsauftrag gesehen werden muss

**verdrängbar** (engl. *preemptable*) ist ein Arbeitsauftrag, wenn seine Ausführung suspendiert werden darf

an beliebigen Stellen (engl. *full preemptive*)

an ausgewiesenen Stellen (engl. *preemption points*)

**unverdrängbar** (engl. *non-preemptable*), sonst

- ▶ der Job muss durchlaufen (engl. *run to completion*)

☞ ggf. Mischbetrieb → Präemptivität als **Jobattribut** implementiert

# Kriterien der Prioritätsvergabe

## Überblick

### Dynamische Prioritäten

#### EDF (engl. *earliest deadline first*)

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

#### LRT (engl. *latest release-time first*), EDF umgekehrt

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

#### LST (engl. *least slack-time first*)

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

### Statische Prioritäten (periodische Verfahren)

#### 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

# EDF — Earliest Deadline First

- ▶ benötigt kein Wissen über Ausführungszeiten von Arbeitsaufträgen
- Arbeitsaufträge**  $J_1 \mapsto 3(0, 6]$ ,  $J_2 \mapsto 2(5, 8]$ ,  $J_3 \mapsto 2(2, 7]$
- ▶ Ausführungszeiten 3, 2, 2 — verzichtbar
  - ▶ zulässige Ausführungsintervalle (*earliest, latest*)
- Ablaufplan**  $J_1[1, 3] \rightarrow J_3[4, 5] \rightarrow J_2[6, 7]$
- ▶ resultierende Ausführungsintervalle [*start, stop*]
  - ▶ Ausführungsintervall (7, 8] gibt Verzögerungsspiel
- ▶ Arbeitsaufträge werden möglichst auslösezeitnah gestartet
    - ▶ lässt den Prozessor nicht untätig, wenn ausführbare Jobs anstehen

# LRT — Latest Release-Time First

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

- ▶ Auslösezeiten sind Termine bzw. Termine sind Auslösezeiten
- ▶ benötigt Wissen über Ausführungszeiten von Arbeitsaufträgen
- Arbeitsaufträge  $J_1 \mapsto 3(0, 6]$ ,  $J_2 \mapsto 2(5, 8]$ ,  $J_3 \mapsto 2(2, 7]$ 
  - ▶ Ausführungszeiten 3, 2, 2
  - ▶ zulässige Ausführungsintervalle (*earliest, latest*)
- Ablaufplan  $J_1[2, 4] \rightarrow J_3[5, 6] \rightarrow J_2[7, 8]$ 
  - ▶ resultierende Ausführungsintervalle [*start, stop*]
  - ▶ Ausführungsintervall (0, 1] bleibt ungenutzt
- ▶ 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

- ▶ benötigt Wissen über die Ausführungszeiten von Arbeitsaufträgen

**Schlupfzeit**  $slack(J_i, t) = deadline(J_i) - t - maturity(J_i, t)$

- ▶ zum Zeitpunkt  $t$
- ▶  $maturity(J_i, t) = WCET(J_i) - elapsed\ time(J_i, t)$

**Arbeitsaufträge**  $J_1 \mapsto 3(0, 6]$ ,  $J_2 \mapsto 2(5, 8]$ ,  $J_3 \mapsto 2(2, 7]$

- ▶ Ausführungszeiten 3, 2, 2
- ▶ zulässige Ausführungsintervalle (*earliest, latest*)

**Ablaufplan**  $J_1[1, 2] \rightarrow J_3[3, 4] \rightarrow J_1[5, 6] \rightarrow J_2[7, 8]$

- ▶ solange  $J_1$  läuft, ist seine Schlupfzeit 3
- ▶  $J_3$  (Schlupfzeit 3) verdrängt  $J_1$  zum Zeitpunkt 2
- ▶ während  $J_3$  läuft, fällt die Schlupfzeit von  $J_1$  auf 1
- ▶  $J_2$  trifft zum Zeitpunkt 5 ein, seine Schlupfzeit ist 1
- ▶ zu dem Zeitpunkt hat  $J_1$  eine Schlupfzeit von 0

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

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

Beispiel:  $J_1 \mapsto 3(0, 6]$ ,  $J_2 \mapsto 2(5, 8]$ ,  $J_3 \mapsto 2(2, 7]$ 

$t$	$J_1$	$J_2$	$J_3$
0	$6 - 0 - (3 - 0) = 3$		
1	$6 - 1 - (3 - 1) = 3$		
2	$6 - 2 - (3 - 2) = 3$		$7 - 2 - (2 - 0) = 3$
3	$6 - 3 - (3 - 2) = 2$		$7 - 3 - (2 - 1) = 3$
4	$6 - 4 - (3 - 2) = 1$		$7 - 4 - (2 - 2) = 3$
5	$6 - 5 - (3 - 2) = 0$	$8 - 5 - (2 - 0) = 1$	
6	$6 - 6 - (3 - 3) = 0$	$8 - 6 - (2 - 0) = 0$	
7		$8 - 7 - (2 - 1) = 0$	
8		$8 - 8 - (2 - 2) = 0$	

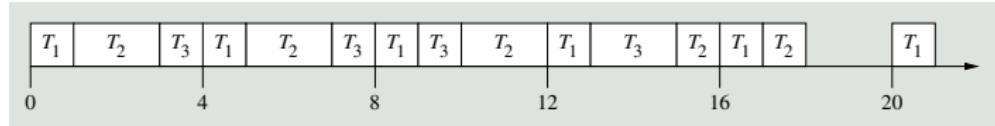
$$J_1[1, 2] \rightarrow J_3[3, 4] \rightarrow J_1[5, 6] \rightarrow J_2[7, 8]$$

# 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  $T_i$
  - ▶ benötigt Wissen über Ausführungszeiten von Arbeitsaufträgen
- Aufgaben  $T_1 = (4, 1)$ ,  $T_2 = (5, 2)$ ,  $T_3 = (20, 5)$
- ▶ 3-Tupel  $(p_i, e_i, D_i)$ ; bei  $D_i = p_i$  wird  $D_i$  nicht angegeben
  - ▶ Perioden  $p_i = 4, 5, 20$
  - ▶ Ausführungszeiten  $e_i = 1, 2, 5$
  - ▶ relative Termine  $D_i = p_i$

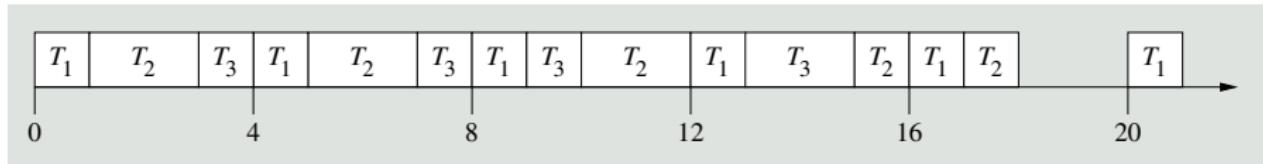
## Ablaufplan



- ▶ 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 mit Ende des ersten Jobs von  $T_1$
- ▶  $T_2$  wird zum Zeitpunkt  $t = 16$  von  $T_1$  verdrängt

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

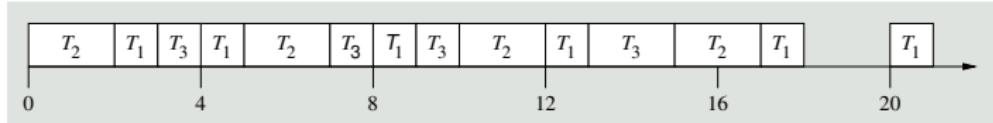
- ▶ die Jobs von  $T_3$  laufen im Hintergrund von  $T_1$  und  $T_2$
- ▶  $T_3$  läuft nur, wenn kein Job von  $T_1$  und  $T_2$  ausführbereit ist
- ▶ für Zeitintervall  $[18, 19]$  gibt es keine ausführbereiten Jobs mehr

# DM — *Deadline Monotonic*

DM = RM wenn gilt:  $D_i$  ist proportional zu  $p_i$

- ▶ 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: identisch/proportional
  - ▶ benötigt Wissen über Ausführungszeiten von Arbeitsaufträgen
- 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

# Resümee

Trennung unterschiedlicher Belange  $\mapsto$  Strategie & Mechanismus

- ▶ Einplanung ist die Strategie, Einlastung ist der Mechanismus

Arbeitsweise ist zeit- oder ereignisgesteuert: Einplanung & Einlastung

- ▶ gekoppelt im zeitgesteuerten System (Taktsteuerung)
- ▶ entkoppelt im ereignisgesteuerten System (Vorrangsteuerung)

Zeitparameter sind Punkte und Intervalle auf der Echtzeitachse

- ▶ (sporadische) Auslösezeit, (absoluter) Termin
- ▶ Antwortzeit bzw. relativer Termin, Schlupfzeit

Taskmodelle für periodische Aufgaben

- ▶ aperiodische oder sporadische Aufgaben bzw. Arbeitsaufträge
  - ▶ je nach dem, ob Jobtermine weich/fest oder hart sind
- ▶ Rangfolgen, Abhängigkeiten, Koordinierung, Verdrängung

Verfahren EDF, LRT, LST, RM und DM

# Literaturverzeichnis

- [1] Jane W. S. Liu.  
*Real-Time Systems.*  
Prentice-Hall, Inc., 2000.