h

7 Prozessorvergabe in Echtzeitsystemen

Burns. A.; Wellings, A.: Real-Time Systems and Programming Languages. Addison-Wesley, 2nd Edition, 1990, p. 399 - 440.

Stankovic, J. A.; Spuri, M.; Ramamritham, K. Buttazo, G. C.: Deadline Schedling for Real-Time Systems. Kluwer Academic Publishers, 1998.

7.1 Systemmodelle

Ein einfaches Modell

- Die Anwendung besteht aus einer festen Anzahl von Prozessen.
- Alle Prozesse arbeiten periodisch mit fester, a priori bekannter Periode.
- Die Prozesse sind voneinander vollkommen unabhängig.
- Der Aufwand für Prozeßumschaltungen ist so gering, daß er vernachlässigt werden kann.
- Jeder Prozeß muß innerhalb seiner Periode einen Durchlauf abschließen (d. h. jeder Durchlauf kann eigener Auftrag mit Zielzeit angesehen werden).
- Alle Prozesse besitzen eine a priori bekannte maximale Durchlaufzeit.

Diese Eigenschaften lassen zu, daß alle Prozesse gleichzeitig einen Durchlauf beginnen können.

Standardnotation für die wichtigsten Prozeßattribute

Notation	Beschreibung
B	Längste auftretende Blockierungszeit
C	Längste auftretende Rechenzeit
D	Zielzeit
I	Maximale Interferenz (durch höherpriore Prozesse)
J	Zulässiger Freigabejitter
N	Zahl der Prozesse im System
P	Priorität (soweit vorhanden)
R	Längste zulässige Verweilzeit
T	Periodenlänge
U	Systembelastung (C / T)

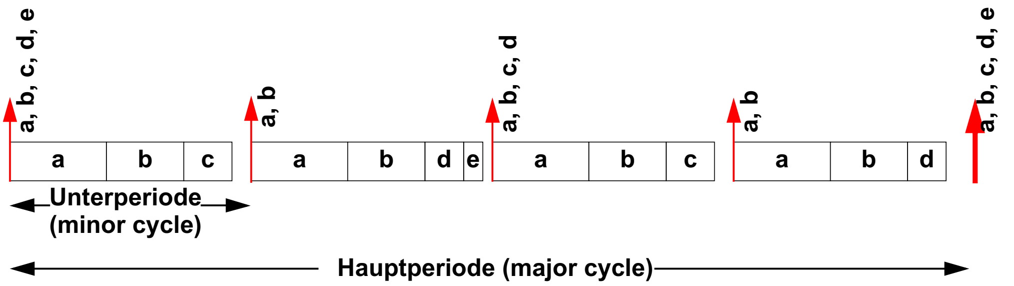
In diesem Kapitel werden Prioritäten durch ganze Zahlen ausgedrückt, wobei größere Werte höhere Priorität bezeichnen!

Vorsicht: Die Literatur ist in dieser Hinsicht uneinheitlich!

7.2 Zyklische Ausführungspläne

Beispiel zur Vorgehensweise

Prozeß	Periode	Rechenzeit
a	25	10
b	25	8
c	50	5
d	50	4
e	100	2




```
while (true) {
    Procedure_for_a;
    Procedure_for_b;
    Procedure_for_c;
    Wait_for_interrupt;

    Procedure_for_a;
    Procedure_for_b;
    Procedure_for_d;
    Procedure_for_e;
    Wait_for_interrupt;

    Procedure_for_a;
    Procedure_for_b;
    Procedure_for_c;
    Wait_for_interrupt;

    Procedure_for_a;
    Procedure_for_b;
    Procedure_for_d;
    Wait_for_interrupt;
}
```

BP 2

Prozessorvergabe - Echtzeitsysteme: Modelle

Eigenschaften

Vorteilhaft

- **Zur Laufzeit nur ein Prozeß, d. h. keine Prozeßumschaltungen**
- **Die Prozeduren alle in gleichem Adreßraum ausgeführt**

Nachteilig

- **Alle Perioden müssen Vielfache der Unterperiode sein**
- **Einordnung sporadischer Prozesse sehr schwierig**
- **Vorgänge mit sehr langen Perioden müssen evtl. in sehr viele kurzlaufende Teilstücke zerlegt werden**
- **Konstruktion des Ablaufplans schwierig, NP-vollständig**

BP 2

Prozessorvergabe - Echtzeitsysteme: RMS

7.3 **Prozeßbasierte Zuordnung**

Jeder periodische Vorgang als eigener Prozeß

Zuordnung:

- **nicht-verdrängend**
- **verdrängend**
- **verzögert verdrängend (kooperativ)**

Abarbeitung nach abnehmenden Raten (rate monotonic priority)

- $T_i < T_j \Rightarrow P_i > P_j$

S7.1 **Wenn statische Prioritäten so vergeben werden können, daß keine Verspätungen auftreten, dann treten auch bei Abarbeitung nach abnehmenden Raten keine Verspätungen auf.**

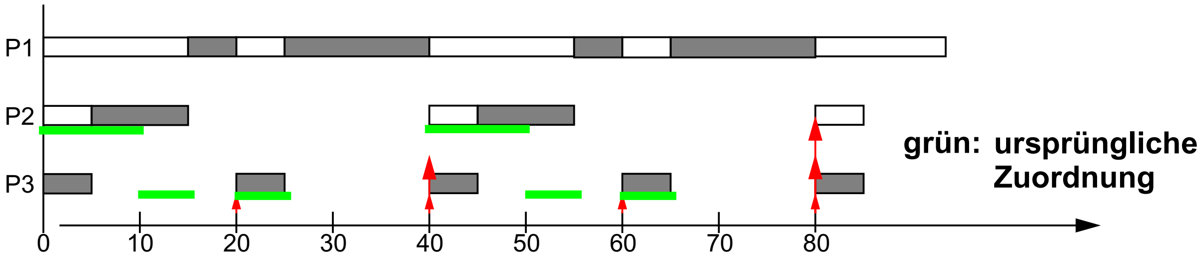
Prozessorvergabe - Echtzeitsysteme: RMS

Beweis

Gegeben seien n Prozesse A_i mit einer statischen Prioritätenzuordnung, die keine Zielzeitverletzungen zur Folge hat und es sei $P_i > P_j$ und $T_i > T_j$.

Dann führt Vertauschung von P_i und P_j auch nicht zu Zielzeitverletzungen.

Prozeß	T	C	P
P1	80	40	1
P2	40	10	2
P3	20	5	3



Prozessorvergabe - Echtzeitsysteme: RMS

Wie kann man feststellen, ob im konreten Fall bei Verwendung von RMS keine Verspätungen auftreten?

7.4 Nutzungsbasierte Tests

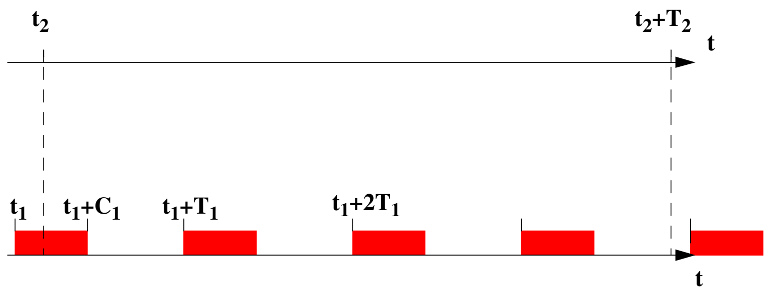
D7.1 Definition

- **Ein kritischer Zeitpunkt für einen Auftrag ist ein Zeitpunkt, zu dem die Ankunft des Auftrags die längste Verweilzeit zur Folge hätte.**
- **Ein kritisches Zeitintervall für einen Auftrag ist ein Intervall das mit einem kritischen Zeitpunkt beginnt und mit seiner Fertigstellung endet.**

S7.1 **Für jeden Auftrag ist es ein kritischer Zeitpunkt, wenn er gleichzeitig mit allen höher prioren Aufträgen in das System kommt.**

Beweis

Für zwei Prozesse unmittelbar ersichtlich durch Verschiebung des Startzeitpunktes von Prozeß 1



Allgemein durch Iteration obiger Überlegung.

S7.2 **Für zwei periodische Prozesse ist die kleinste obere Grenze für die Auslastung**

$$U = 2(2^{1/2} - 1).$$

Beweis

T_1 und T_2 seien die Perioden der beiden Prozesse,

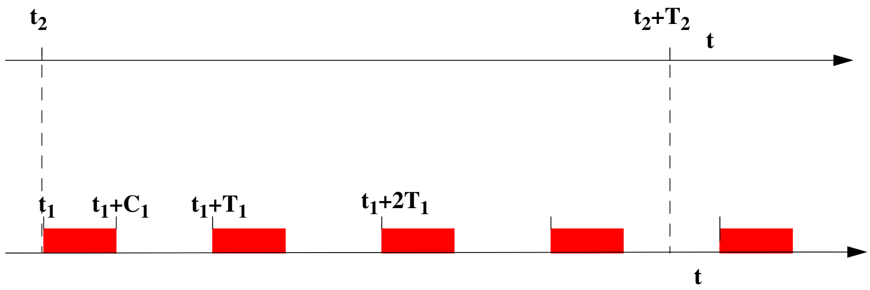
C_1 und C_2 ihr Rechenzeitbedarf pro Periode.

O. B. d. A.: $T_1 < T_2$.

Kritischer Zeitabschnitt enthält $\lceil T_2/T_1 \rceil$ **Anforderungen von T_1.**

BP 2 Prozessorvergabe - Echtzeitsysteme: RMS

a) C_1 sei so kurz, daß alle Anforderungen von T_1 während T_2 erfüllt werden können.



Es muß gelten: $C_1 \le T_2 - T_1 \lfloor T_2 / T_1 \rfloor$.

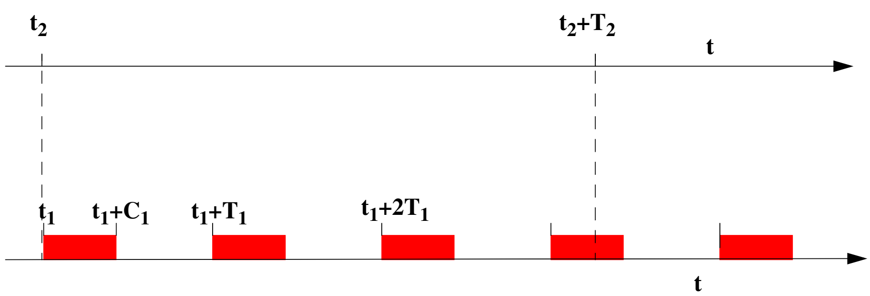
Also gilt für den maximalen Wert von C_2: $C_2 = T_2 - C_1 \lceil T_2 / T_1 \rceil$.

Daraus errechnet sich $U = 1 + C_1 [1/T_1 - (1/T_2) \lceil T_2/T_1 \rceil]$.

U ist mit steigenden C_1 monoton fallend.

BP 2 Prozessorvergabe - Echtzeitsysteme: RMS

b) Die Ausführung der $\lceil T_2 / T_1 \rceil$-ten Anforderung von Auftrag 1 erstreckt sich in die 2. Anforderung des Auftrags 2.



Es muß gelten: $C_1 \ge T_2 - T_1 \lfloor T_2 / T_1 \rfloor$.

Also gilt für den maximalen Wert von C_2: $C_2 = -C_1 \lfloor T_2 / T_1 \rfloor + T_1 \lfloor T_2 / T_1 \rfloor$.

Daraus errechnet sich $U = (T_1/T_2) \lfloor T_2/T_1 \rfloor + C_1 [1/T_1 - (1/T_2) \lfloor T_2/T_1 \rfloor]$.

U ist mit steigenden C_1 monoton steigend.

Das Minimum von U liegt offenbar an der Grenze zwischen den beiden Fällen, d. h. bei $C_1 = T_2 - T_1 \lfloor T_2/T_1 \rfloor$.

In diesem Fall ist $U = 1 - (T_1/T_2)[\lceil T_2/T_1 \rceil - (T_2/T_1)][(T_2/T_1) - \lfloor T_2/T_1 \rfloor]$.

Setzt man zur Vereinfachung der Schreibweise

$I = \lfloor T_2/T_1 \rfloor$ **und** $f = (T_2/T_1) - \lfloor T_2/T_1 \rfloor$,

so kann obige Gleichung geschrieben werden in der Form

$U = 1 - f(1-f)/(I+f)$.

Da U monoton wachsend in I ist, muß im Minimum I = 1 sein.

Minimierung bzgl. f liefert dann $f = 2^{1/2} - 1$.

Satz 7.3 läßt sich verallgmeinern zu.

S7.3 **(Liu and Layland, 1973)**

Wenn die Bedingung

$$\sum_{i=1}^{N} \frac{C_i}{T_i} < N(2^{1/N} - 1)$$

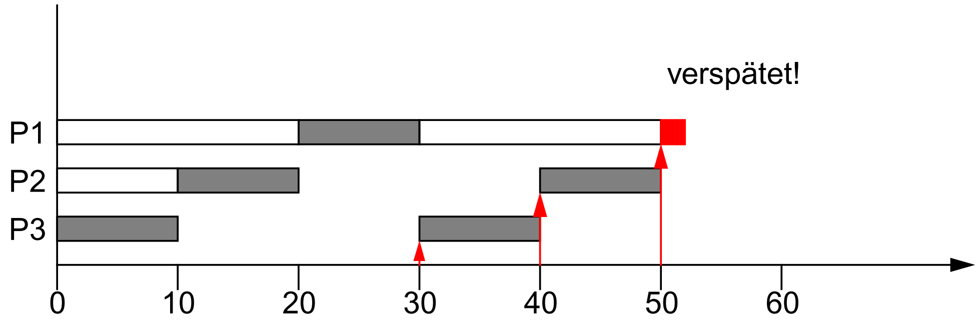
erfüllt ist, dann werden bei Zuordnung nach abnehmenden Raten die Zielzeiten eingehalten.

U(1) = 1,0	**U(6) ≈ 0,734**
U(2) ≈ 0,828	**U(7) ≈ 0,728**
U(3) ≈ 0,779	**U(8) ≈ 0,724**
U(4) ≈ 0,756	**U(9) ≈ 0,720**
U(5) ≈ 0,743	
	U(∞) = ln 2 ≈ 0,69

Beispiele

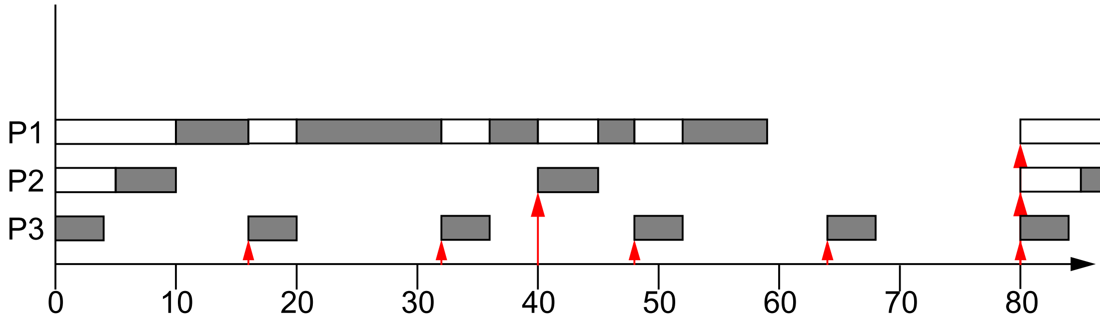
Beispiel A: Bedingung nicht erfüllt; Perioden nicht einhaltbar

Prozeß	T	C	P	U
P1	50	12	1	0.24
P2	40	10	2	0.25
P3	30	10	3	0.33



Beispiel B: Bedingung erfüllt; Perioden eingehalten

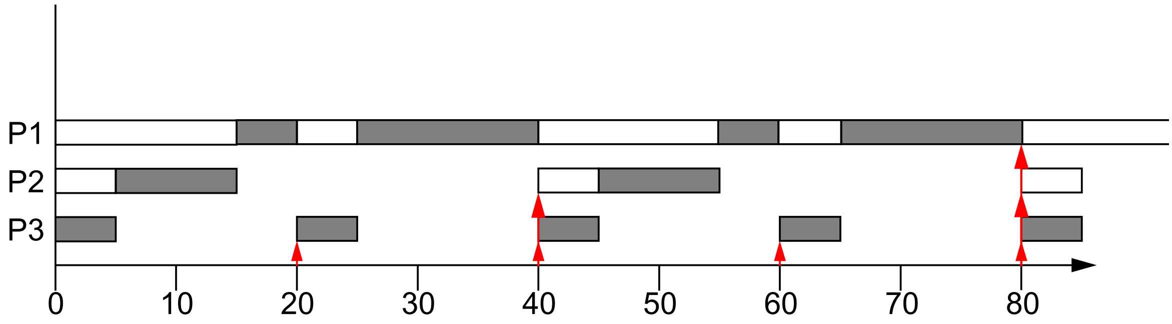
Prozeß	T	C	P	U
P1	80	32	1	0.400
P2	40	5	2	0.125
P3	16	4	3	0.250



Prozessorvergabe - Echtzeitsysteme: RMS

Beispiel C: Bedingung nicht erfüllt; Perioden einhaltbar

Prozeß	T	C	P	U
P1	80	40	1	0.50
P2	40	10	2	0.25
P3	20	5	3	0.25



Prozessorvergabe - Echtzeitsysteme: RMS

7.5

Antwortzeitanalyse

- **Prozeß höchster Priorität:** $R = C$
- **Für andere:** $R_i = C_i + I_i$
- **Maximale Verdrängungszeit tritt ein, wenn alle höherprioren Prozesse gleichzeitig mit dem betrachteten eine Periode beginnen; o. B. d. A. Beginn zum Zeitpunkt 0.**

Betrachtet wird das Intervall $[0, R_i)$ **und der Fall** $j < i$

- **Zahl der Deblockierungen von Prozeß j:** $\left\lceil \frac{R_i}{T_j} \right\rceil$
- **Maximale Interferenz:** $\left\lceil \frac{R_i}{T_j} \right\rceil C_j$

also $I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j,$

dabei bezeichnet hp(i) **die Menge der Prozesse, deren Priorität größer i ist.**

- **Durch Substitution:** $R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$

Lösung obiger impliziter Gleichung für R_i

- **Kleinster Fixpunkt liefert schlechteste Antwortzeit**
- **Iteratives Verfahren**

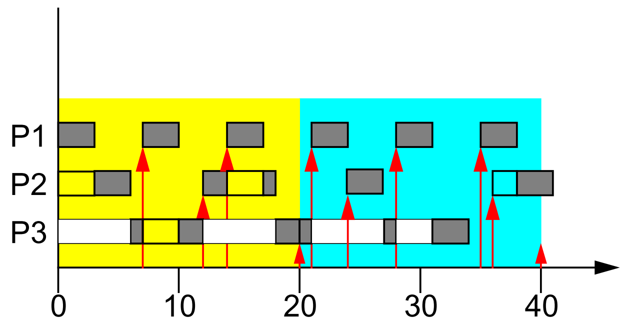
$$w_i^{(n+1)} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^{(n)}}{T_j} \right\rceil C_j$$

- **Algorithmus:**

```
for (int i = 0; i < N; i++) {
   int n =  0; w[n][i] = C[i];
   while (true) {
     w[n + 1][i] = C[i];
     for (int j = 0; j < N; j++) {
       if (P[j] > P[i])
         w[n + 1][i] += Math.ceil((float)w[n][i] / T[j]) * C[j];
     }
     if (w[n + 1][i] == w[n][i]) {
       R[i] = w[n][i];
       break;
     }
     if (w[n + 1][i] > T[i]) {
       // Not schedulable
       break;
     }
     n++;
   }
}
```

Beispiel D:

Prozeß	T	C	P	$w^{(0)}$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$	$w^{(4)}$	$w^{(5)}$
P1	7	3	3	3	3	3	3	3	3
P2	12	3	2	3	6	6	6	6	6
P3	20	5	1	5	11	14	17	20	20



S7.4 **Notwendig und hinreichend für die Einhaltbarkeit der Zielzeiten ist, daß für alle Prozesse i die Beziehung $R_i \leq T_i$ gilt.**

7.6 Worst-case- Ausführungszeit

Bislang angenommen, daß für jeden Prozeß i die maximal benötigte Rechenzeit C_i bekannt ist.

Wie wird sie bestimmt?

- **Messung; Einfluß von Pufferspeichern!**
- **Analyse (abzählen der benötigten Takte); setzt genaues Prozessormodell voraus!**
- **Behandlung von Schleifen? Evtl. semantisches Wissen einbeziehen**

```
for (int i = 1; i <= 10; i++) {
   if (condition)
      ... ; // Block, der 100 Zeiteinheiten benötigt
   else
      ... ; // Block, der 10 Zeiteinheiten  benötigt
}
```

- **Vielleicht bekannt, daß erster Block höchstens dreimal durchlaufen wird.**
- **Alle Iterationen und Rekursionen müssen a priori bekannte Schranken besitzen.**

7.7 **Sporadische und aperiodische Prozesse**

Erweiterung des Systemmodells

- **Zusätzlich sporadische Aufträge**
  - **T interpretiert als kleinstes Zeitintervall zwischen zwei Auslösungen**
- **Zielzeit kann vor Periodenende liegen: D < T**

Vorangehender Algorithmus auch hier anwendbar, wenn die Bedingung `w[n + 1][i] > T[i]` durch `w[n + 1][i] > D[i]` ersetzt wird.

Funktioniert für jede Prioritätenzuordnung!

Harte und weiche Echtzeitbedingungen (hard and soft real-time)

In manchen Fällen ist Einhalten von Zielzeiten erwünscht, aber nicht unbedingt erforderlich.

Dann sind Strategien möglich, die nicht immer mit der schlechtesten Situation arbeiten, z. B.

- **Prozesse sollen mit mittleren Ausführungszeiten und Startraten einplanbar sein**
- **Harte Echtzeitprozesse sollen einplanbar sein**

7.8 **Zuordnung nach *deadline monotonic priority***

Für D < T: In vorangehenden Überlegungen T durch D ersetzen

S7.5 **Zuordnung nach *deadline monotonic priority* ist für jede Menge Q von Prozessen, die nach einer prioritätsgeteuerten Strategie einplanbar ist, eine optimale prioritätsgeteuerte Strategie.**

7.9 Interaktion und Blockierung

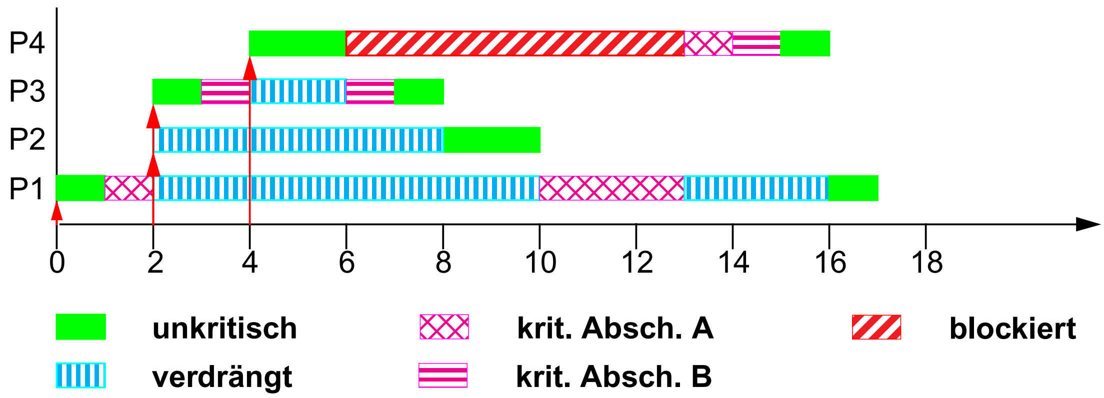
Zusätzlich zu berücksichtigen: Prozesse können sich gegenseitig koordinieren

Es kann zu Prioritätsinversionen (*priority inversion*) kommen, d. h. ein hochpriorer Prozeß muß auf Freigabe durch einen niederprioren warten.

- **Meist nicht vollkommen vermeidbar, aber nachteilige Effekte können minimiert werden.**

- **Beispiel E**

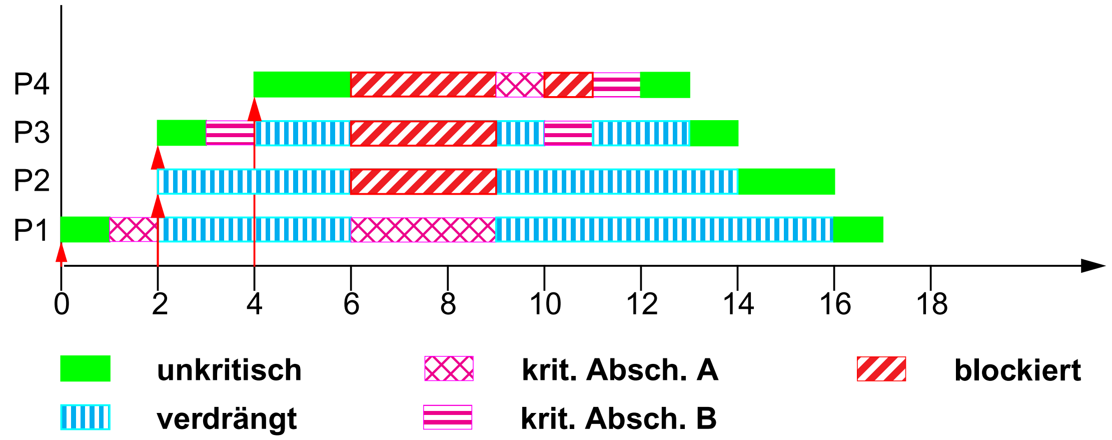
Prozeß	Priorität	Ausführungsfolge	Startzeit
P4	4	RRABR	4
P3	3	RBBR	2
P2	2	RR	2
P1	1	RAAAAR	0



Vermeidung dieses Effektes durch Prioritätsvererbung

Wenn Prozeß P blockiert wird, weil ein Prozeß Q mit niedriger Priorität im kritischen Abschnitt ist, dann bekommt Q bis zum Verlassen des kritischen Abschnitts die gleiche Priorität wie P

Beispiel E



Diese Vererbung kann sich über mehrere Prozesse erstrecken!

S7.6 **Sha e. a., 1990:**

Wenn ein Prozeß m kritische Abschnitte besitzt und n Prozesse niedrigerer Priorität existieren, dann wird er bei Prioritätsvererbung höchstens min(m, n)-mal blockiert.

Bestimmung der maximalen Blockierungszeit

Es sei $\text{nutzung}(k, i) = 1$**, wenn Betriebsmittel k von wenigstens einem Prozeß geringerer Priorität und wenigstens einem Prozeß mit gleicher oder höherer Priorität benutzt wird.**

Ansonsten gelte $\text{nutzung}(k, i) = 0$**.**

K bezeichne die Zahl verschiedener Betriebsmittel.

Dann gilt für die maximale Blockierungszeit B_i

$$B_i \le \sum_{k=1}^{K} \text{nutzung}(k, i)\text{Zahl\_k\_A}(k)$$

Antwortzeiten bei Berücksichtigung von Blockierungen

- Frühere Berechnung ergänzt zu

$$R = C + B + I$$

- also

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Daraus ergibt sich die Rekursion

$$w_i^{(n+1)} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^{(n)}}{T_j} \right\rceil C_j$$

Diese Formel ist wegen der Art, in der die Blockierungen abgeschätzt wurden, pessimistisch.

7.10 Priority ceiling

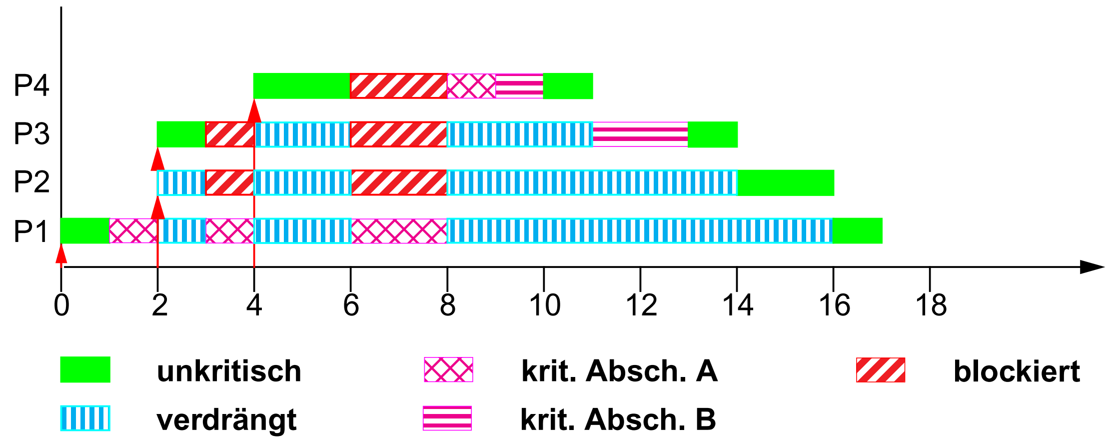
Ursprüngliches Prioritäten begrenzendes Protokoll (original ceiling priority protocol, OCPP)

- Jeder Prozeß besitzt eine statische Priorität.
- Jedes Betriebsmittel besitzt eine statische Grenzpriorität (ceiling value); es ist dies die maximale Priorität benutzender Prozesse.
- Prozesse besitzen eine dynamische Priorität. Für einen Prozeß P, der ein Betriebsmittel BM belegt hat, ist dies das Maximum aus seiner statischen Priorität und den dynamischen Prioritäten derjenigen Prozesse, die blockiert sind, weil sie auf die Freigabe des Betriebsmittels BM warten.
- Ein Prozeß kann ein Betriebsmittel nur dann belegen, wenn seine dynamische Priorität höher ist, als die Grenzpriorität aller derzeit von anderen Prozessen belegten Betriebsmittel.

Wirkung: Die Belegung eines ersten Betriebsmittels ist erlaubt. Ein zweites Betriebsmittel darf nur belegt werden, wenn kein Prozeß höherer Priorität existiert, der gelegentlich beide Betriebsmittel benutzt.

Prozessorvergabe - Echtzeitsysteme: Priority Ceiling

Beispiel E



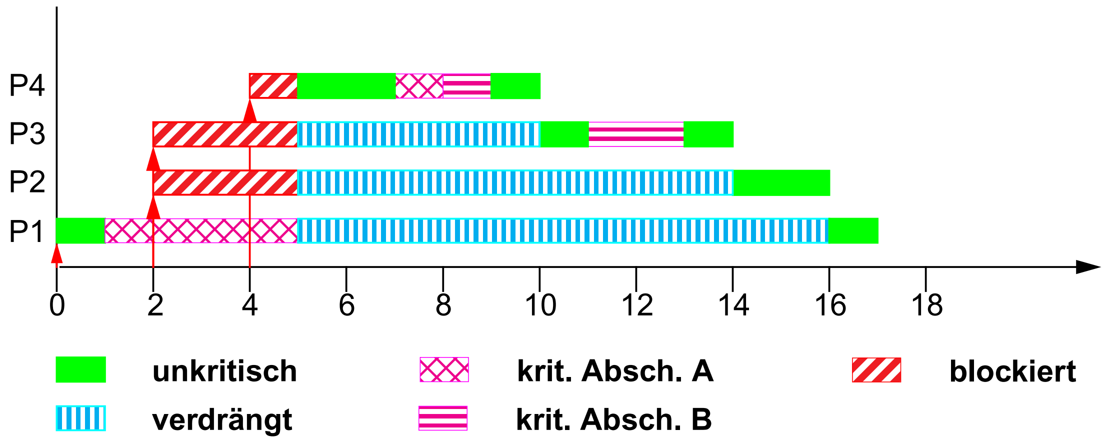
Prozessorvergabe - Echtzeitsysteme:

Unmittelbar begrenzend (immediate ceiling priority protocol, ICPP, in POSIX: Priority Protected Protocol)

- **Jeder Prozeß besitzt eine statische Priorität.**
- **Jedes Betriebsmittel besitzt eine statische Grenzpriorität (ceiling value); es ist dies die maximale Priorität benutzender Prozesse.**
- **Prozesse besitzen eine dynamische Priorität. Für einen Prozeß P ist dies das Maximum aus seiner statischen Priorität und den Grenzprioritäten aller von ihm belegter Betriebsmittel.**
- **Ein Prozeß kann ein Betriebsmittel nur dann belegen, wenn seine dynamische Priorität höher ist, als die Grenzpriorität aller derzeit von anderen Prozessen belegten Betriebsmittel.**

Prozessorvergabe - Echtzeitsysteme: Priority Ceiling

Beispiel E



Prozessorvergabe - Echtzeitsysteme: Priority Ceiling

Vergleich

- Beide Strategien haben das gleiche Worst-Case-Verhalten
- ICPP ist einfacher zu implementieren
- ICPP erzeugt weniger Kontextwechsel
- ICPP erzeugt mehr Prioritätsänderungen

7.11 Dynamisch Prioritäten (Zuordnung nach aufsteigenden Zielzeiten, EDF)

D4.2 Definition

- **Gegeben sei eine Menge von Echtzeitaufträgen und ein Zeitintervall $[t_1, t_2)$. Die Rechenlast der Menge im Intervall $[t_1, t_2)$ ist**

$$h_{[t_1, t_2)} = \sum_{t_1 \leq r_k,\, d_k \leq t_2} C_k.$$

- **Gegeben sei eine Menge von Echtzeitaufträgen. Ihr Belastungsfaktor im Intervall $[t_1, t_2)$ ist der Anteil des Intervalls der zur Abarbeitung benötigt wird, d. h.**

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

- **Der (absolute) Belastungsfaktor ist das Suprenum über alle möglichen Intervalle, d. h.**

$$u = \sup_{0 \leq t_1 \leq t_2} u_{[t_1, t_2)}$$

S7.7 Satz (Spuri, 1995)

Eine Menge von Echtzeitaufträgen ist genau dann ohne Verspätungen abarbeitbar, wenn $u \leq 1$ ist·

Beweis

a) Notwendig: trivial

b) Hinreichend:

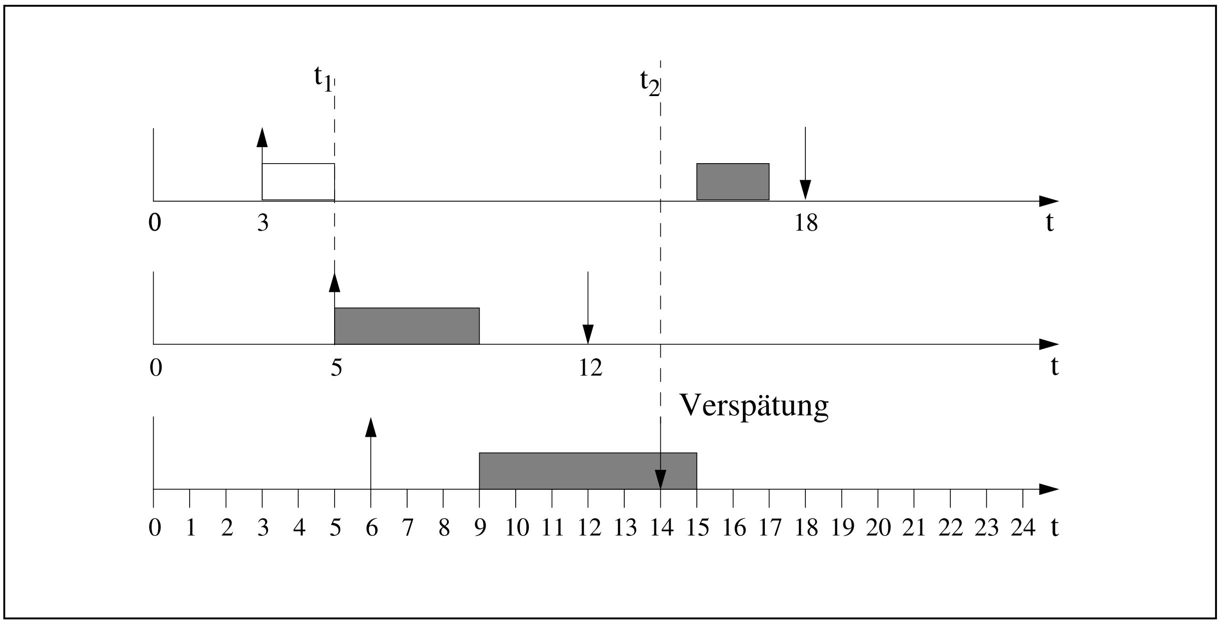
Zum Zeitpunkt t_2 beginne eine Verspätung. Dann ist davor eine Rechenperiode, in der nur Teilaufträge mit kleinerer Zielzeit bearbeitet werden.

t_1 sei letzter Zeitpunkt vor t_2 derart, daß alle vor t_1 freigegebenen Teilaufträge mit Zielzeiten kleiner oder gleich t_2 abgearbeitet sind.

Dann muß gelten

$$\sum_{t_1 \leq r_k,\, d_k \leq t_2} C_k > (t_2 - t_1)$$

Also ist $u[t_1, t_2) > 1$ und somit u > 1, was einen Widerspruch darstellt.



S7.8 **Satz (Liu and Layland, 1973)**

Jede Menge synchroner periodischer Aufträge mit einer Auslastung $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$

kann genau dann mit EDF ohne Verspätung abgearbeitet werden, wenn $U \le 1$ **ist.**

Beweis

Es genügt zu zeigen, daß der Lastfaktor u gleich U ist.

Nach Satz 7.8 folgt daraus die Aussage.

Für jedes Intervall $[t_1, t_2)$ **gilt:**

$$\sum_{t_1 \le r_k,\, t_2 \le d_k} C_k \le \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \le \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1) \sum_{i=1}^{n} \frac{C_i}{T_i}$$

also ist $u[t_1, t_2) \le U$.

Sei t_1 = 0 and t_2 = kgV(T_1,...,T_n), dann gilt

$$u[t_1, t_2) = \frac{\sum_{i=1}^{n} \frac{t_2}{T_i} C_i}{t_2} = U.$$

Somit ist u = U.

S7.9 **Satz (Coffman, 1976)**

Jede Menge asynchroner periodischer Aufträge mit einer Auslastung $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$

kann genau dann mit EDF ohne Verspätung abgearbeitet werden, wenn $U \le 1$ **ist.**

Beweis

Wiederum ist es ausreichend zu zeigen, daß u = U ist.

Für jedes Intervall gilt [t_1, t_2):

$$\sum_{t_1 \le r_k,\, t_2 \le d_k} C_k \le \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \le \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1) \sum_{i=1}^{n} \frac{C_i}{T_i},$$

also ist $u[t_1, t_2) \le U$.

BP 2

Prozessorvergabe - vert. Syst.: EDF

Man betrachte das Intervall [0, s + mH), wobei m eine ganze Zahl größer 0 ist, $s = \max\{s_1,\ldots,s_n\}$, und $H = \mathrm{kgV}(T_1,\ldots,T_n)$:

$$h[0, s + mH) \le \sum_{i=1} \frac{mH}{T_i} C_i = mH \sum_{i=1}^{n} \frac{C_i}{T_i} = mHU.$$

Damit folgt $u[0, s + mH) \le \frac{mH}{s + mH} U$

und für beliebiges m $u \ge U$,

also ist u = U.

Mit Satz 7.8 folgt daraus die Aussage.

BP 2

Prozessorvergabe - vert. Syst.: EDF

Daß schon geringfügig komplexere Modelle die Analyse stark erschweren können, zeigt der folgende Satz.

S7.10 **Satz (Leung and Merrill, 1980)**

Zu entscheiden, ob bei einer synchronen Menge periodischer Aufträge, bei denen die Zielzeiten vor den Periodenenden liegen, keine Verspätungen auftreten ist NP-hart.

7.12 Scheduling in verteilten Echtzeitsystemen

Besonderheit

Teilaufträge können mit Empfang einer Nachricht beginnen und deshalb möglicherweise nicht sofort mit Periodenbeginn rechenbereit sein.

Die Differenz aus Periodenbeginn und tatsächlicher Rechenbereitschaft wird als Jitter bezeichnet (ist immer kleiner oder gleich 0!)

Teilaufträge:

- **Ankunftszeitpunkt:** Zeitpunkt des Periodenbeginns
- **Freigabezeitpunkt:** Aufnahme in die Bereit-Warteschlange

Es gibt drei wesentliche Untersuchungsmethoden.

- **Holistische Analyse auf Basis von EDF**
  Statische Zuordnung zu Knoten und lokale Ablaufplanung
- **Spring-Algorithmus**
  Zweigeteilter statischer Algorithmus
  - Welche Aufträge sollen dem gleichen Knoten zugeteilt werden (zur Reduktion des Kommunikationsaufwands)
  - Ermittlung eines Ablaufplans für jeden Knoten
- **Gezielte Weitergabe und Angebotseinholung**
  Angewandt bei dynamischer Erzeugung von Aufträgen, die nicht an einen bestimmten Knoten gebunden sind.

Genauere Betrachtung der holistischen Analyse

Besondere Herausforderung: 'End-zu-End'-Bedingungen

Das sind Zeitbedingungen die Aktivitäten auf mehreren Knoten umfassen und den Kommunikationsaufwand einbeziehen.

Einschränkung

- Empfang höchstens einer Nachricht pro Periode und zwar am Anfang
- 'End-zu-End'-Bedingungen nur zwischen einem Sender und einem Empfänger

Fünf wesentliche Zeitabschnitte:

1. Erzeugungs-Verzögerung: Zeit zum Erzeugen einer Nachricht und zum Einreihen in die Sende-Warteschlange
2. Sende-Verzögerung: Zeit von der Einreihung in die Sende-Warteschlange bis zur tatsächlichen Einspeisung in das Verbindungsnetz
3. Übertragungs-Verzögerung: Verweilzeit im Netz
4. Annahme-Verzögerung: Zeit von der Übergabe durch das Netz bis zur Ankunft beim Empfangsprozeß

5. Bearbeitungs-Verzögerung: Zeit für die Bearbeitung der empfangenen Nachricht

Prozessorvergabe - vert. Syst.: EDF

Globale Betrachtung wird vermieden durch "Attribut-Vererbung"

- **Nachricht erbt vom Sender Periode und Freigabejitter (Schwankung der Sende-Verzögerung)**
- **Empfänger erbt beide Attribute von der empfangenen Nachricht**

Gesamtanalyse iterativ:

1. **Anfangssituation ohne Jitter**
2. **Lokale Ermittlung der größten Verzögerungen und Nachrichten sowie für das Netz**
3. **Ermittlung der Verzögerungs-Jitter**
4. **Wiederholung ab 2**

Prozessorvergabe - vert. Syst.: EDF

Bezeichnungen

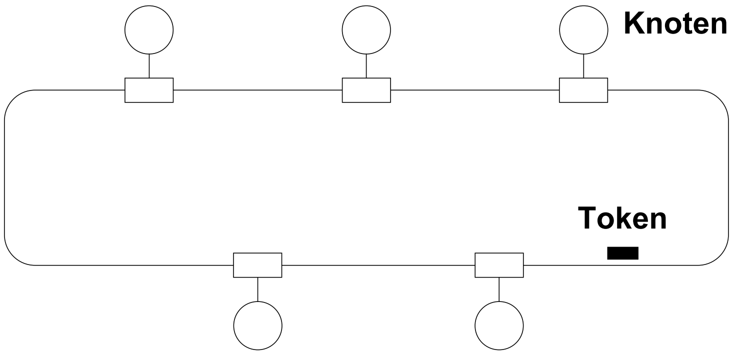
RT_{edf}:	**Menge aller Anwortzeiten der Aufträge eines Knotens unter Verwendung von EDF**
RT_{net}:	**Menge der Übertragungsverzögerungen unter Verwendung der Netzstrategie NET**
J:	**Maximaler Freigabejitter**
P:	**Netzwerkverzögerungen**

Iteratives Rechenverfahren

Teilschritt 1	Teilschritt 2
$RT_1^{(m+1)} = \mathbf{RT}_{edf}(J_1^{(m)})$	$J_1^{(m+1)} = \mathbf{P}_1(RT_{net}^{(m+1)})$
.	.
.	.
.	.
$RT_n^{(m+1)} = \mathbf{RT}_{edf}(J_n^{(m)})$	$J_n^{(m+1)} = \mathbf{P}_n(RT_{net}^{(m+1)})$
$RT_{net}^{(m+1)} = \mathbf{RT}_{net}(J_1^{(m)})$	$J_{net}^{(m+1)} = \mathbf{P}_{net}(RT_1^{(m+1)}, \ldots, RT_n^{(m+1)})$

Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme), F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg
ist ohne Genehmigung des Autors unzulässig

Beispiel: Tokenring



- Token darf maximale Umlaufzeit TTRT (target token rotation time) nicht überschreiten
- Jeder Knoten p darf das Token für maximal H_p Zeiteinheiten behalten (synchronous bandwidth)
- Der gesamte Overhead des Token-Protokolls pro Umlauf sei τ.

Es muß gelten: $\sum_{p=1}^{n} H_p + \tau \le TTRT$.

S7.11 **Wenn bei Verwendung von EDF für ein bestimmtes Ankunftszeitmuster der Teilaufträge eine Verspätung auftritt,**

dann existiert in einem Ablauf, in dem die Ankunftszeiten gleichzeitig Freigabezeiten sind,

eine Verspätung ohne vorangehenden Leerstand des Prozessors.

S7.12 **Verallgemeinerung des Satzes 7.2 von Liu und Layland**

Die längste Antwortzeit eines Auftrags i tritt in einem Intervall mit Auslastungsfaktor 1 auf, in dem jeder andere Auftrag zu Beginn des Intervalls einen Teilauftrag in die Bereitwarteschlange einfügt und seine weiteren Teilaufträge mit maximaler Rate folgen.

Dieser Satz erlaubt eine iterative Berechnung der schlechtesten Anwortzeit gemäß nachfolgender Überlegung.

Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme), F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

Es sei in einem Intervall gemäß Satz 7.13 $a \geq -J$ **Freigabezeitpunkt des Teilauftrags von Auftrag i, der maximale Antwortzeit aufweist.**

Dann besitzt in diesem Intervall der erste Teilauftrag des Auftrags i den Freigabezeitpunkt $s_i(a) = a + J_i - \left\lfloor \frac{a + J_i}{T_i} \right\rfloor T_i$**.**

Dementsprechend erzeugt in diesem Intervall Auftrag i die Last $\left(1 + \left\lfloor \frac{a + J_i}{T_i} \right\rfloor\right) C_i$**.**

Von allen anderen Teilaufträgen haben höchstens $1 + \left\lfloor \frac{a + D_i + J_j - D_j}{T_j} \right\rfloor$ **eine Zielzeit kleiner oder gleich** $d = a + D_i$**.**

Also ist die Last der bezüglich Zielzeit d höherprioren Teilaufträge , die bis zum Zeitpunkt t ankamen

$$W_i(a, t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min\left\{\left\lceil \frac{t + J_j}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i + J_j - D_j}{T_j} \right\rfloor\right\} C_j.$$

Die Länge $L_i(a)$ **der resultierenden Vollastperiode bezüglich Zielzeit d kann damit iterativ berechnet werden gemäß**

$$L_i^{(0)}(a) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} C_j$$

$$L_i^{(m+1)}(a) = W_i(a, L_i^{(m)}(a)) + \left(1 + \left\lfloor \frac{a + J_i}{T_i} \right\rfloor\right) C_i + B_{k(a + D_i)},$$

mit $k(d) = \max\{k : (D_k - J_k \leq d)\}$**, wenn die Aufträge so numeriert sind, daß** $i < j \Rightarrow D_i - J_i \leq D_j - J_j$ **für alle i und j erfüllt ist.**

Es läßt sich zeigen, daß dieses Verfahren nach endlich vielen Schritten konvergiert, wenn $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$ **ist.**

Die längste Antwortzeit bezogen auf a ist dann $rt_i(a) = \max\{J_i + C_i + B_i, L_i(a) - a\}$**.**

Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme), F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

Prozessorvergabe - vert. Syst.: EDF

Man kann zeigen, daß das erste Vollastintervall (bei obigen Annahmen) unter allen Vollastintervallen maximale Länge L besitzt.

Daher müssen nur im Intervall $-J_i \leq a \leq L - J_i - C_i - B_i$ **(also für endlich viele Argumente) die Werte von** $rt_i(a)$ **berechnet und ihr Maximum ermittelt werden.**

Prozessorvergabe - vert. Syst.: EDF

Mit der gleichen Methode lassen sich wegen der Attributvererbung die Sendeverzögerungen der Kommunikationsprozessoren berechnen.

Die auftretenden Verzögerungen beim Netzzugang können als Interferenz eines fiktiven, höherprioren Auftrags betrachtet werden.

Dabei sind drei Einflüsse zu berücksichtigen:

- **Die lokal höherpriore Nachrichtenlast mit Zielzeitpunkten kleiner** $a + D_i$**.**
- **Bisherige Nachrichtenpakete zur Übermittlung der Gesamtnachricht.**
- **Der Einfluß anderer Knoten auf den Netzzugang.**

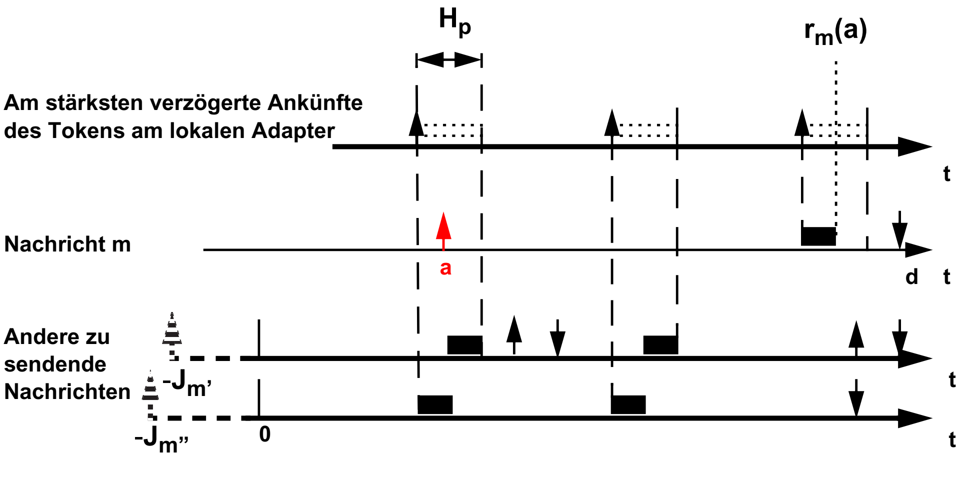
Die ersten beiden sind gut abschätzbar; der dritte bedarf einer gesonderten Überlegung.

Sendeverzögerung bei synchroner Nutzung des Tokenrings

Nachrichten werden in Pakete zerlegt

- C_m Zahl der Pakete für Nachricht m
- n_m Auftrag i versendet Nachricht m in jeder n_m-ten Periode

Mögliches Szenario



a Ankunft der Nachricht m in der Sendewarteschlange

H_p Intervall, in dem der Knoten p Packete senden darf

BP 2 Prozessorvergabe - vert. Syst.: EDF

ρ **Zeit zur Einspeisung einer Nachricht**

P **Verweilzeit im Netzwerk**

$L_{m,k}(a)$ **Zeit, die von höherprioren Paketen verbraucht wird**

$$r_m = \max_{a \geq -J_m}\{r_m(a)\}$$

Verzögerung der ersten k Pakete

$$r_{m,k}(a) = \max\{J_m + B_m + k\rho + P, L_{m,k}(a) + \rho + P - a\}$$

Die für höherpriore Nachrichten verbrauchte Zeit $L_{m,k}(a)$ wird bestimmt durch

1. **zu sendende Nachrichten mit Zielzeit kleiner oder gleich $a + D_m$**
2. **vorangehende Pakete**
3. **Zeit für Umlauf des Tokens**

Für alle drei Anteile sind die Obergrenzen gut abschätzbar

BP 2 Prozessorvergabe - vert. Syst.: EDF

Bestimmung des Wertes für a, der zur maximalen Verzögerung führt mit gleicher Methode wie bei Ermittlung der gleichen Methode, wie sie zur Ermittlung der Erzeugungsverzögerung angewandt wurde.

Annahmeverzögerung

- **Wird für jedes Zeitintervall bestimmt durch die Zahl ankommender Nachrichten**
- **Kann anhand der Modellannahmen gut abgeschätzt werden**

Bearbeitungsverzögerung

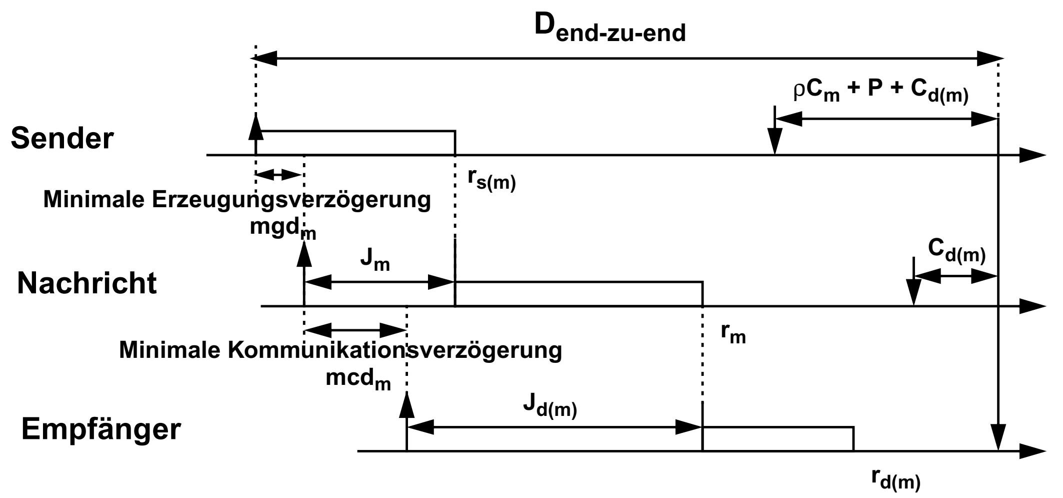
- **Berechnung wie bei Erzeugungsverzögerung, wobei die Annahmeverzögerung als Last höchster Priorität behandelt wird.**

Bestimmung der Zielzeitenvorgabe für die einzelnen Komponenten aus der End-zu-End-Verzögerung

Optimale Zielzeitenvorgabe ungelöst

BP 2 Prozessorvergabe - vert. Syst.: EDF

Approximation



$$D_{d(m)} = \min\{D_{d(m)}, D_{end-zu-end} - mgd_m - mcd_m\}$$

$$D_m = \min\{D_m, D_{d(m)} - C_{d(m)} + mcd_m\}$$

$$D_{s(m)} = \min\{D_{s(m)}, D_m - mcd_m + mgd_m\}$$