3.3 Dynamische Ablaufplanung

Zur Vorlesung

Embedded Systems

WS 14/15

Reiner Kolla

3.3.1 Problemstellung

Dynamische Ablaufplanung kommt dann zum Einsatz, wenn zum Zeitpunkt des Entwurfs das Planungsproblem nicht vollständig bekannt ist. Ursache dafür kann sein

- unbekannte oder datenabhängig stark variierende Länge der Bearbeitungszeit einzelner Aufgaben
- Zahl und Art der einzelnen Aufgaben nicht vorab bekannt
- Datenabhängigkeiten sind nicht vollständig bekannt oder variieren zur Laufzeit (Kontrollflussorientierte Probleme)

Dynamische Ablaufplanung kommt demnach meist dann zur Anwendung, wenn man eingebettete Systeme in Software auf der Basis sogenannter Echtzeit Systeme implementiert.
Im Jargon der Echtzeit Systeme unterscheidet man

- Offline - Probleme (= Planungsproblem ist zur Compilezeit vollständig bekannt, zumindest in Form von worst case Werten) und
- Online - Probleme (= Planungsproblem ist zur Compilezeit nicht vollständig bekannt).

Je nach Anwendungsfall kann es jedoch auch Sinn machen, ein Offline-Problem mit dynamischer Ablaufplanung anzugehen.

Deadlines, Releasetimes und Dispatchlatenz

Wir betrachten zunächst einfache Planungprobleme mit Problemgraphen $G = (V,\{ \})$, d.h. ohne Datenabhängigkeiten, aber mit Ausführungszeiten $w(v)$ für jeden Knoten v. Wir betrachten ferner

- Absolute Zeitbeschränkungen
  - Deadlines: $t_d(v)$ späteste Endzeitpunkte von Operationen
  - Releasezeiten: $t_r(v)$ früheste Startzeitpunkte von Operationen

Im Allgemeinen benötigt man bei dynamischer Ablaufplanung auch eine gewisse Rechenzeit, um die nächste, auszuführende Aufgabe zu bestimmen.

Definition -- Dispatchlatenz
Die **Dispatchlatenz** L_D bezeichnet die maximale Zeitspanne zwischen dem Stoppen einer Aufgabe u auf der CPU und dem Start der nächsten Aufgabe v.

Wir nehmen zunächst der Einfachheit halber an, dass $L_D = 0$.

Ressourcenauslastung, Abarbeitungszeit

Wir wollen zunächst noch einige weitere Begriffe formal fassen.

Definition -- Ressourcenauslastung
Die **Ressourcenauslastung** U bei einer einzigen CPU als Ressource und einem Ablaufplan der Latenz L ist in Prozent gegeben durch

$$U := \frac{\sum_{v \in V} w(v)}{L} \cdot 100$$

Bei präemptiver Ablaufplanung kann die Bearbeitung einer Aufgabe mehrfach unterbrochen werden. In diesem Fall ist neben der reinen Berechnungszeit $w(v)$ der Aufgabe v auch folgende Zeit interessant.

Definition -- Abarbeitungszeit
Sei in einem dynamischen Ablauf $\tau_b(v)$ der Zeitpunkt, zu dem die Aufgabe v zum ersten Mal zur Ausführung kommt, und $\tau_e(v)$ der Zeitpunkt, zu dem v endgültig endet. Dann ist die **Abarbeitungszeit** der Aufgabe v gegeben durch $t_A(v) := \tau_e(v) - \tau_b(v)$

Wartezeit und Flusszeit

Wir können nun auch feststellen, wie lange eine Aufgabe insgesamt warten musste.

Definition -- Wartezeit

Die **Wartezeit** einer Aufgabe *v* in einem dynamischen Ablauf ist gegeben durch $t_W(v) := \tau_e(v) - t_r(v) - w(v)$.

Die Wartezeit ist also die gesamte Zeitspanne, die eine Aufgabe von ihrem frühest möglichen Startzeitpunkt bis zum effektiven Ende nicht aktiv war.

Nimmt man die eigene Aktivität nicht von dieser Zeitspanne weg kommt man auf die sogenannte Flusszeit:

Definition -- Flusszeit

Die **Flusszeit** einer Aufgabe *v* in einem dynamischen Ablauf ist gegeben durch $t_F(v) := \tau_e(v) - t_r(v)$

Lateness und Tardiness

Die Verletzung von Zeitbedingungen in eingebetteten Systemen kann unerwünschte bis katastrophale Folgen haben. Daher sind zur Bewertung folgende Größen interessant.

Definition -- Lateness

Die **Lateness** einer Aufgabe *v* in einem dynamischen Ablauf ist gegeben durch $t_L(v) := \tau_e(v) - t_d(v)$.

Die Lateness kann sowohl positiv als auch negativ sein. Ist man nur an echten Verletzungen der Zeitbedingungen interessiert, betrachtet man

Definition -- Tardiness

Die **Tardiness** einer Aufgabe *v* in einem dynamischen Ablauf ist gegeben durch $t_T(v) := max\{t_L(v), 0\}$.

Optimierungsziele bei dynamischer Ablaufplanung

Die Optimierungsziele hängen stark davon ab, wie hart die Zeitanforderungen an das System sind. Extreme sind

time sharing systems:

Dies sind Mehrbenutzersysteme, wie man sie in der klassischen Betriebssystemwelt kennt. Primäre Ziele sind hier

- Maximierung der Ressourcenauslastung (minimiere Latenz!)
- Minimierung von mittleren Fluss- bzw. Wartezeiten (ggf. gewichtet mit Kosten *c(v)*)

$$\text{minimiere } F := \frac{\sum_{v \in V} c(v) \cdot t_F(v)}{\sum_{v \in V} c(v)}$$

$$\text{oder } W := \frac{\sum_{v \in V} c(v) \cdot t_W(v)}{\sum_{v \in V} c(v)}$$

Optimierungsziele bei dynamischer Ablaufplanung

real time operating systems (RTOS):

Hier herrschen harte Zeitbedingungen an die einzelnen Aufgaben vor. Primäre Ziele sind hier

- Minimierung der maximalen Lateness
- Minimierung der mittleren Tardiness
- Minimierung der Zahl der Aufgaben mit Tardiness > 0

$$\text{Minimiere } max\{t_L(v) \mid v \in V\}$$

$$\text{oder } \frac{1}{|V|} \sum_{v \in V} t_T(v)$$

$$\text{oder } \sum_{v \in V} (t_T(v) > 0)$$

3.3.2 Nichtpräemptive dynamische Planung

Wir wollen zunächst dynamische Planungsverfahren für Aufgaben betrachten, die nicht präemptiv sind, d.h. alle Aufgaben müssen ohne Unterbrechung auf der CPU bearbeitet werden. Wir nehmen zudem auch noch an, dass keine Datenabhängigkeiten (Präzedenzbedingungen) bestehen und keine Echtzeitbedingungen (Releasezeiten und Deadlines) vorliegen.

Da Ablaufplanungsprobleme so alt wie unsere Zivilisation sind, gibt es auch ebenso alte Heuristiken:

First come first served Planung (FCFS)

Diese Heuristik plant die Aufgaben in der Reihenfolge, in der sie entstehen. („Wer zuerst kommt, mahlt zuerst"). Die mittlere Wartezeit *W* hängt stark von der Ankunftsreihenfolge ab:

Beispiel: Betrachte *w(a) = 24, w(b) = 3, w(c) = 30*

Reihenfolge a,b,c: $t_W(a) = 0, t_W(b) = 24, t_W(c) = 27, W = 51/3 = 17$

Reihenfolge b,a,c: $t_W(a) = 3, t_W(b) = 0, \quad t_W(c) = 27, W = 30/3 = 10$

SJF - Planung: die Smith-Regel

Shortest job first Planung (SJF)

Diese Heuristik plant die Aufgaben in der Reihenfolge ihrer gewichteten Bearbeitungszeit. D.h. gegeben seien Aufgaben *v* mit Bearbeitungszeiten *w(v)* und Gewicht *c(v)*. Dann plane die Aufgaben aufsteigend nach *w(v)/c(v).*

Satz -- Smith Regel

Gegeben sei eine Menge von Jobs *V* ohne Datenabhängigkeiten mit Releasezeit $t_r(v) = 0, \forall v \in V$. Dann hat ein Ablaufplan $v_1, \ldots, v_n$ mit $w(v_i)/c(v_i) \le w(v_{i+1})/c(v_{i+1})$ für *i=1,…,n-1* minimale mittlere Wartezeit (Flusszeit).

Beweis: -- indirekt Wir nehmen an, es gäbe ein Paar *i,i+1* mit

$w(v_i)/c(v_i) > w(v_{i+1})/c(v_{i+1})$

Es ist $W := \frac{\sum_i c(v_i) \cdot t_W(v_i)}{\sum_i c(v_i)}$ und $t_W(v_i) = \sum_{j=1}^{i-1} w(v_j)$

Die Smith-Regel -- ff

Also kann man mit $C := \sum_i c(v_i)$ den Wert *C W* auch schreiben als

$$C \cdot W = K + c(v_i) \cdot \underbrace{\sum_{j=1}^{i-1} w(v_j)}_{=:T} + c(v_{i+1}) \cdot \sum_{j=1}^{i} w(v_j)$$

also $C \cdot W = K + c(v_i) \cdot T + c(v_{i+1}) \cdot (T + w(v_i))$

Tauscht man nun in der Reihenfolge *i* mit *i+1*, so erhält man als Änderung $C \cdot W_{i \leftrightarrow i+1} = K + c(v_{i+1}) \cdot T + c(v_i) \cdot (T + w(v_{i+1}))$

Nun gilt aber

$W_{i \leftrightarrow i+1} < W \quad \Leftrightarrow C \cdot W_{i \leftrightarrow i+1} < C \cdot W$

$\Leftrightarrow K + c(v_{i+1}) \cdot T + c(v_i) \cdot (T + w(v_{i+1})) < K + c(v_i) \cdot T + c(v_{i+1}) \cdot (T + w(v_i))$

$\Leftrightarrow c(v_i) \cdot w(v_{i+1}) < c(v_{i+1}) \cdot w(v_i) \quad \Leftrightarrow w(v_{i+1}) / c(v_{i+1}) < w(v_i) / c(v_i)$

Also gibt es bei minimaler mittlerer Wartezeit kein solches Paar *i,i+1*.

Anmerkungen zur Smith - Regel

Die Smith-Regel liefert uns mit SJF auch ein erstes Beispiel eines **prioritätsbasierten**, dynamischen Planungsverfahrens: Wenn man die Quotienten *-w(v)/c(v)* als Priorität interpretiert, werden die Aufgaben mit fallender Priorität bearbeitet.

Diese Priorität ist allerdings **statisch**, d.h. sie ändert sich während der Berechnung der Aufgaben nicht.

Das Problem der Minimierung der mittleren Wartezeit ist allerdings nur unter unseren Annahmen so einfach zu lösen:

Führt man von 0 verschiedene Releasezeiten ein, wird das Problem der Minimierung der mittleren Wartezeit NP-hart.

Ebenso wird das Problem NP-hart, wenn man Datenabhängigkeiten einführt.

! Es ist aussichtslos, dynamische Ablaufplanungsprobleme exakt zur Laufzeit lösen zu wollen, wenn diese algorithmisch schwierig werden, weil sonst ggf. mehr Zeit vom Dispatcher verbraucht wird, als von den zu bearbeitenden Aufgaben!

EDF - Planung: die Jackson-Regel

Earliest deadline first Planung (EDF)

Diese Heuristik plant die Aufgaben in der Reihenfolge ihrer frühesten Deadline. Sie liefert unter folgenden Bedingungen einen Ablaufplan, der die maximale Lateness minimiert:

Satz -- Jackson Regel

Gegeben sei eine Menge von Jobs V ohne Datenabhängigkeiten mit Releasezeit $t_r(v) = 0, \forall\, v \in V$. Dann hat ein Ablaufplan $v_1, \ldots, v_n$ mit $t_d(v_i) \leq t_d(v_{i+1})$ für $i=1,\ldots,n-1$ minimale maximale Lateness.

Beweis: -- indirekt

Wir nehmen wieder an, es gäbe ein Paar $i,i+1$ mit $t_d(v_i) > t_d(v_{i+1})$

Die Lateness von v_i, v_{i+1} ist jeweils

$$t_L(v_i) = \underbrace{\sum_{j=1}^{i-1} w(v_j)}_{=:T} + w(v_i) - t_d(v_i) \qquad \text{sowie } t_L(v_{i+1}) = T + w(v_i) + w(v_{i+1}) - t_d(v_{i+1})$$

Die Jackson-Regel -- ff

Unter der Annahme, dass $t_d(v_i) > t_d(v_{i+1})$ liefert demnach v_{i+1} die größere Lateness, denn

$$t_L(v_i) = T + w(v_i) - t_d(v_i) < T + w(v_i) + w(v_{i+1}) - t_d(v_{i+1}) = t_L(v_{i+1})$$

$$\Leftrightarrow t_d(v_{i+1}) < w(v_{i+1}) + t_d(v_i) \qquad \Leftarrow t_d(v_{i+1}) < t_d(v_i)$$

Tauscht man in der Reihenfolge $i \leftrightarrow i+1$, erhält man als Lateness

$$t'_L(v_i) = T + w(v_i) + w(v_{i+1}) - t_d(v_i)$$

$$t'_L(v_{i+1}) = T + w(v_{i+1}) - t_d(v_{i+1})$$

Die maximale Lateness kann durch diesen Tausch aber nur größer werden, wenn

$$t_L(v_{i+1}) < max\{T + w(v_i) + w(v_{i+1}) - t_d(v_i), T + w(v_{i+1}) - t_d(v_{i+1})\}$$

Die Jackson-Regel -- ff

$$t_L(v_{i+1}) < max\{T + w(v_i) + w(v_{i+1}) - t_d(v_i), T + w(v_{i+1}) - t_d(v_{i+1})\}$$

kann, da $t_L(v_{i+1}) > T + w(v_{i+1}) - t_d(v_{i+1})$ nur gelten, wenn

$$t_L(v_{i+1}) = T + w(v_{i+1}) + w(v_i) - t_d(v_{i+1}) < T + w(v_i) + w(v_{i+1}) - t_d(v_i)$$

$$\Leftrightarrow t_d(v_i) < t_d(v_{i+1}) \;\lightning$$

Demnach kann man jedes Paar $i,i+1$, mit $t_d(v_i) > t_d(v_{i+1})$ in der Reihenfolge tauschen, ohne die maximale Lateness zu erhöhen.

Also kann man einen Plan mit minimaler maximaler Lateness so umformen, dass er die Bedingung $t_d(v_i) \leq t_d(v_{i+1})$ für $i=1,\ldots,n-1$ erfüllt.

Anmerkungen zur Jackson - Regel

Auch die Jackson-Regel liefert uns mit EDF auch ein Beispiel eines **prioritätsbasierten**, dynamischen Planungsverfahrens.

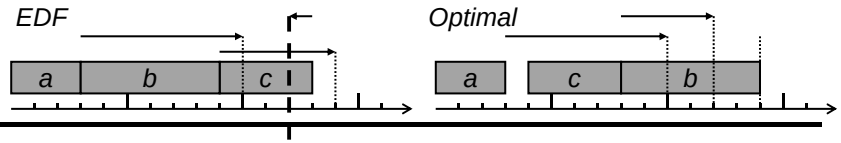
Diese Priorität ist ebenfalls **statisch**, d.h. sie ändert sich während der Berechnung der Aufgaben nicht.

Das Problem der Minimierung der maximalen Lateness ist allerdings auch nur unter unseren Annahmen so einfach zu lösen:

Führt man von 0 verschiedene Releasezeiten oder Datenabhängigkeiten ein, ist das Problem in traktabler Laufzeit nicht mehr exakt zu lösen. Ein Beispiel, dass EDF schon bei von 0 verschiedenen Ankunftzeiten nicht mehr optimal ist, liefert etwa:

Beispiel: Betrachte $w(a) = 3, w(b) = 6, w(c) = 4$ mit

$$t_r(a) = 0, t_r(b) = 2, t_r(c) = 4, t_d(a) = 10, t_d(b) = 14, t_d(c) = 12,$$



3.3.3 Präemptive dynamische Planung

Wir wollen nun dynamische Planungsverfahren für präemptive Aufgaben betrachten, d.h. alle Aufgaben können in ihrer Bearbeitung unterbrochen werden. Datenabhängigkeiten (Präzedenzbedingungen) und unterschiedliche Releasezeiten seien dahingehend berücksichtigt, dass zu einem bestimmten Zeitpunkt stets nur die Teilmenge der Aufgaben im Zustand „bereit“ (ready) betrachten.

Wir können nun die präemptiven Pendants zu den uns schon bekannten Heuristiken betrachten:

Weighted Round Robin Planung (WRR)

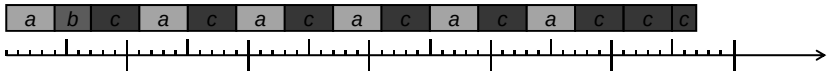
Diese Heuristik sortiert die Aufgaben in der Reihenfolge, in der sie entstehen in eine FIFO Warteschlange ein. Jede Aufgabe *v* erhält bis zu *c(v)* viele Zeitschlitze der Länge *Q*, wird dann unterbrochen, und an das Ende der Warteschlange angehängt.

WRR -- Pro‘s und Con‘s

- Kein Aushungern von Aufgaben. Jede Aufgabe *v* erhält die Ressource anteilsmäßig ihres Gewichts *c(v)*.
- Sehr effizient und einfach zu implementieren (Ringpuffer aus |*V*| Plätzen)
- Hohe mittlere Wartezeit,
- Keine Garantie von Echtzeitbedingungen (Deadlines)

Beispiel: Betrachte wieder *w(a) = 24, w(b) = 3, w(c) = 30*

Reihenfolge a,b,c Zeitquantum pro Zeitschlitz Q = 4



$t_W(a) = 47-24 = 23$, $t_W(b) = 7-3 = 4$, $t_W(c) = 27$, $W = 54/3 = 18$

Reihenfolge a,b,c Zeitquantum pro Zeitschlitz Q = 10

$t_W(a) = 23$, $t_W(b) = 10$, $t_W(c) = 27$, $W = 60/3 = 20$

WRR -- Anmerkungen

Man kann zum Weighted Round Robin Verfahren zusätzlich noch folgendes anmerken:

- Die mittlere Wartezeit ist hoch, man kann aber bei kontrollorientierten Aufgaben sicherstellen, dass jede Aufgabe *v* höchstens

$$\left(\sum_{u \in V} c(u) - c(v)\right) \cdot Q$$

Zeiteinheiten suspendiert ist.

- Lässt man das Zeitquantum *Q* sehr groß werden, verhält sich das Verfahren wie sein nicht präemptives Pendant FCFS.
- Lässt man das Zeitquantum sehr klein werden, verhält sich das System als hätte man |*V*| Ressourcen mit einer Performanz von *1/|V|* (ohne Datenabhängigkeiten). Allerdings ist hier der Aufwand für das Umschalten zwischen den Aufgaben vernachlässigt. Dieser sollte, gemessen am Quantum *Q*, gering sein!

EDF - Planung

Earliest deadline first Planung (EDF)

Die EDF - Heuristik lässt sich sehr schön auf den präemptiven Fall erweitern: Man verwalte alle Aufgaben *v* im Zustand „bereit“ in einer Prioritätswarteschlange mit Priorität $1/t_d(v)$ oder $-t_d(v)$, und bearbeite stets die Aufgabe höchster Priorität. Die Bearbeitung wird unterbrochen, sobald die Releasezeit einer Aufgabe erreicht oder eine Aufgabe in den Zustand „bereit“ übergeht, die höhere Priorität hat.

Man kann nun folgendes aussagen:

Satz

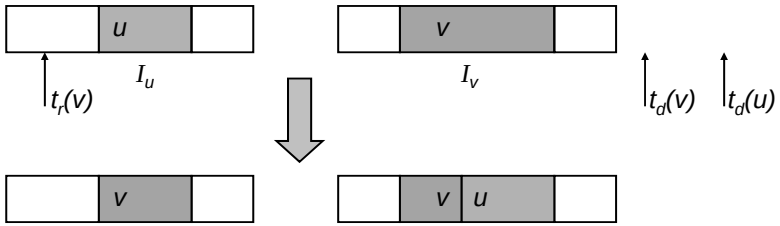
Gegeben sei eine Menge von Aufgaben *V* ohne Datenabhängigkeiten. Wenn ein präemptiver Ablaufplan existiert, der alle Deadlines einhält, dann erfüllt auch ein nach EDF erstellter Ablaufplan alle Deadlines.

Diese Aussage ist stärker als die Jackson-Regel, weil wir keine Voraussetzung über die Releasezeiten machen!

EDF - Planung -- Beweis

Wir betrachten einen Ablaufplan, der alle Deadlines erfüllt, und nehmen an, es sei kein EDF-Plan. Dann gibt es zwei aufeinander folgende Ausführungsintervalle I_u und I_v zu Aufgaben *u* und *v*, so dass $t_d(u) > t_d(v)$, aber *v* zum Beginn von *u* hätte ausgeführt werden können, d.h. $t_r(v) \leq \tau_b(I_u)$

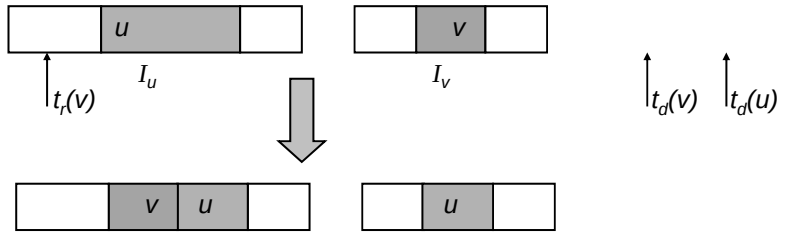
Fall 1: I_u ist kürzer als I_v



Nach Tausch halten alle Deadlines und die neuen Intervalle sind EDF-konform.

EDF - Planung -- Beweis ff

Fall 2: I_u ist länger als I_v



Nach Tausch halten alle Deadlines und die neuen Intervalle sind EDF-konform.

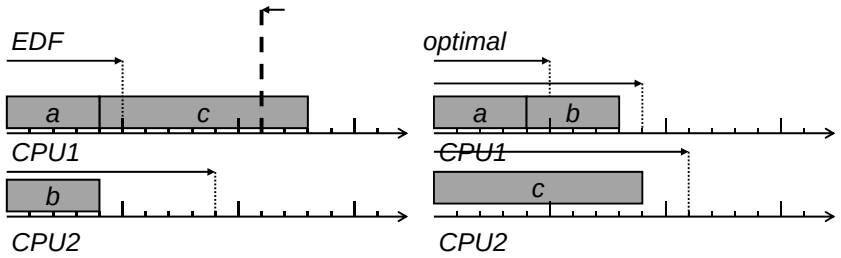
Man kann also den Plan so transformieren, bis alle Ausführungs-intervalle nach EDF geplant sind. Es kann nun nur noch Lücken geben, in denen keine Aufgabe bearbeitet wird, obwohl die Releasezeit nachfolgender Aufgaben kleiner ist. Diese Lücken kann man aber als Aufgabe mit Deadline ∞ auffassen und mit dem gleichen Argument entfernen.

Anmerkungen zur EDF - Planung

Man erhält mit EDF Planung zwar einen gültigen Ablaufplan, wenn einer existiert, aber selbst wenn man nur die 1 Prozessor Voraus-setzung wegfallen lässt, ist EDF schon nicht mehr optimal:

Beispiel: Betrachte *w(a) = 4, w(b) = 4, w(c) = 9* mit

$t_r(a) = 0, t_r(b) = 0, t_r(c) = 0, t_d(a) = 5, t_d(b) = 9, t_d(c) = 11,$



Anomalien bei prioritätsgesteuerter Planung

EDF Planung ist ein typisches Beispiel für ein Planungsverfahren nach dynamischen Prioritäten.

Problem: Das Hauptproblem dynamischer, prioritätsbasierter Planungsverfahren ist die **Vorhersagbarkeit des Verhaltens**. Daher werden dynamische Verfahren bis heute kaum in Echtzeitsystemen mit sicherheitskritischen Zeitbedingungen eingesetzt.

Dass es nicht einmal so ohne weiteres möglich ist, best und worst case Verhalten abzuschätzen, soll folgendes Beispiel verdeutlichen:

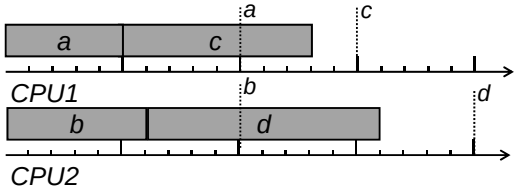
Beispiel: Gegeben seien 4 Aufgaben *a, b, c, d,* mit Releasezeiten, Deadlines und best/worst case Laufzeiten nach folgender Tabelle:

	t_r	t_d	$[w_-, w_+]$
a	0	10	[5 , 5]
b	0	10	[2 , 6]
c	4	15	[8 , 8]
d	0	20	[10,10]

Frage:
Liefert eine EDF Planung auf 2 Prozessoren, wobei eine Aufgabe nur auf dem Prozessor laufen darf, auf dem sie angefangen wurde, für alle möglichen Laufzeiten von *b* stets einen Ablauf, der alle Deadlines einhält?

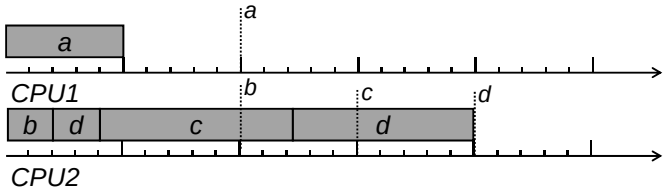
Anomalien ff

Wir betrachten zunächst den scheinbar schlechtesten Fall, *w(b) = 6*:



	t_r	t_d	[w_- , w_+]
a	0	10	[5 , 5]
b	0	10	[2 , 6]
c	4	15	[8 , 8]
d	0	20	[10,10]

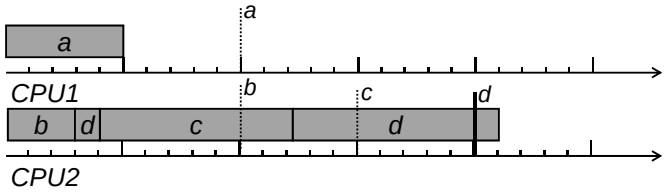
Auch im scheinbar besten Fall, *w(b) = 2*, ist alles im grünen Bereich:



Anomalien ff

	t_r	t_d	[w_- , w_+]
a	0	10	[5 , 5]
b	0	10	[2 , 6]
c	4	15	[8 , 8]
d	0	20	[10,10]

Aber wenn *w(b) = 3* ist, passiert ein Unglück:



3.3.4 Präemptive periodische Planung

Wir wollen nun dynamische Planungsverfahren für präemptive **periodische** Aufgaben betrachten, d.h. alle Aufgaben können in ihrer Bearbeitung unterbrochen werden. Zunächst wollen wir ein System periodischer Aufgaben ohne Datenabhängigkeiten betrachten:

Periodische Tasks:

Gegeben sei eine endliche Menge *V* von Aufgaben mit einer **relativen Releasezeit** (Phase) $t_r^*(v)$, einer **relativen Deadline** $t_d^*(v)$, einer **Laufzeit** $w(v) \le t_d^*(v)$ und einer **Periode** $P(v) \ge t_d^*(v)$.

Dann wird dadurch ein System periodischer Tasks definiert, wobei für jedes $v \in V$ eine unendliche Menge periodisch auftretetender Tasks (v,n), $n \in N_0$ mit folgenden Releasezeiten und Deadlines assoziiert wird:

$t_r(v,n) := t_r^*(v) + n*P(v)$

$t_d(v,n) := t_r^*(v) + t_d^*(v) + n*P(v)$

Ratenmonotone Planung (RM-scheduling)

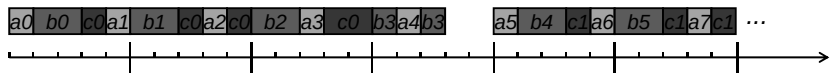
Ein einfaches dynamisches Planungsverfahren für solche periodischen Tasks ist das sogenannte ratenmonotone Planungsverfahren. Es ist ein prioritätsbasiertes Verfahren, das die Tasks *v* gemäß ihrer **Rate** *1/P(v)* bzw. *-P(v)* als Priorität plant.

Ratenmonotone Planung (RM-Scheduling)

Halte stets alle bereiten Tasks *v* in einer Prioritätswarteschlange *Q* geordnet nach ihrer Rate *1/P(v).* Ordne stets Tasks höchster Priorität zuerst zu.

Beispiel: Tasks *a,b,c* mit $(t_r^*,w,t_d^*,P) = (0,1,4,4), (0,2,5,5), (0,5,20,20)$

RM Schedule bei einer CPU:



Auslastung $\sum_v \frac{w(v)}{P(v)}$ im Beispiel: 0.25 + 0.4 + 0.25 = 0.9

Deadlinemonotone Planung (DM-scheduling)

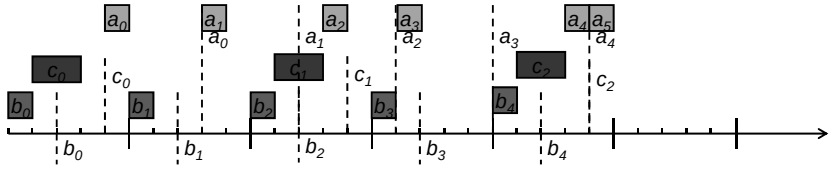
Deadlinemonotone Planung (DM-Scheduling)

Halte stets alle bereiten Tasks *v* in einer Prioritätswarteschlange *Q* geordnet nach ihrer relativen Deadline - $t_d^*(v)$. Ordne stets Tasks höchster Priorität, d.h. kleinster relativer Deadline zuerst zu.

Im Falle, dass wie im letzten Beispiel $t_d^*(v)$ proportional zu $P(v)$ ist, sind RM und DM Planung äquivalent. DM Planung kann aber bei kürzeren Deadlines als Perioden noch gültige Pläne finden, wohingegen RM versagt.

Beispiel: Betrachte wieder Tasks *a,b,c* mit (t_r^*,w,t_d^*,P)

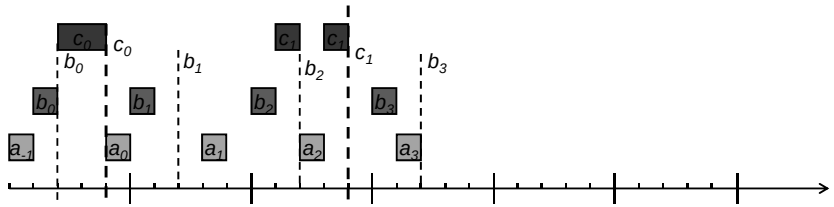
(4,1,4,4), (0,1,2,5), (0,2,3.99,10), d.h. zuerst *b* dann *c* dann *a*



Deadlinemonoton versus Ratenmonoton

Beispiel: Wir planen nun die Tasks *a,b,c* mit (t_r^*,w,t_d^*,P)

(4,1,4,4), (0,1,2,5), (0,2,3.99,10), ratenmonoton, d.h. höchste Priorität hat nun *a*, dann *b*, dann erst *c*:



!! c_i verpassen ihre Deadlines !!

Planung mit statischer Priorität

Beobachtung:

Sowohl ratenmonotone, als auch deadlinemonotone Planung sind prioritätsgesteuerte Verfahren, die die Priorität vorab den Aufgabentypen *v* fest zuordnen, d.h. alle Instanzen *(v,j)* haben gleiche Priorität. Wir nennen eine solche Prioritätsvergabe bei periodischen Tasks auch **statische Priorität**.

Wir wollen nun untersuchen, unter welchen hinreichenden Bedingungen eine statische Prioritätsvergabe zu korrekten Ablaufplänen führt. Dies liefert uns sowohl einen einfachen Sicherheitstest, als auch Optimalitätsaussagen.

Wir nehmen von nun an an, dass alle Tasks verschiedene Priorität haben und nach absteigender Priorität $\pi(v_i)$ nummeriert sind, d.h. $\pi(v_1)>...>\pi(v_n)$.

Definition: Wir nennen $R(v_i):= max\{\tau_e(v_i,j) - (t_r^*(v_i)+j*P(v_i)) \mid j \in N\}$ die maximale Flusszeit oder auch maximale Response einer Instanz der Aufgabe v_i. Instanzen (v_i,j) mit Flusszeit $\geq min\{t_d^*(v_i),R(v_i)\}$ heißen kritisch.

Schranke zur maximalen Response

Lemma:

Gegeben sei ein System periodischer Tasks unter statischer Prioritätszuordnung $\pi(v_1) >...> \pi(v_n)$. Dann liefert ein nach π erstellter Ablaufplan für jedes t_k,das die Ungleichung

$$t_k \geq w(v_k) + \sum_{i=1}^{k-1} \left\lceil \frac{t_k}{P(v_i)} \right\rceil \cdot w(v_i)$$

erfüllt, einen Ablaufplan mit maximaler Response $R(v_k) \leq t_k$.

Beweis:
Betrachte für eine kritische Instanz (v_k,j) das Bearbeitungszeitintervall

$$[\tau_e(v_k,j) - t_k\,,\ \tau_e(v_k,j)]$$

Dann wurde von den Tasks v_i, $i<k$, höherer Priorität höchstens die Zeit

$$\sum_{i=1}^{k-1} \left\lceil \frac{t_k}{P(v_i)} \right\rceil \cdot w(v_i)$$

in diesem Intervall verbraucht.
Es verbleibt also mindestens $w(v_k)$ Zeit für (v_k,j).

Schranke zur maximalen Response --ff

Beweis- ff

Da (v_k,j) am Ende des Intervalls endet, können Aufgaben niedrigerer Priorität nur dann in diesem Intervall Zeit verbraucht haben, wenn der Releasezeitpunkt von (v_k,j) echt in diesem Intervall liegt, d.h $R(v_k) < t_k$ gilt.

Wenn keine Aufgabe niedrigerer Priorität in diesem Intervall gelaufen ist, muss, da mindestens $w(v_k)$ Zeit für (v_k,j) bleibt, (v_k,j) vollständig in diesem Intervall und insbesondere in einem Abschnitt der Länge t' vor Ende des Intervalls bearbeitet worden sein.

Wir nehmen nun an, dass der Releasezeitpunkt von (v_k,j) t'' vor $\tau_e(v_k,j) - t_k$ liegt. Dann gilt für das Zeitintervall

$$[\tau_e(v_k,j) - t_k - min\{t',t''\},\ \tau_e(v_k,j) - min\{t',t''\}]$$

der Länge t_k wieder, dass mindestens $w(v_k)$ Zeit für (v_k,j) bleibt.

Würde eine Task niedrigerer Priorität ausgeführt, wäre dies nun ein Widerspruch zur Lage des Releasezeitpunktes, also muss t' das Minimum bilden. Dann würde aber (v_k,j) komplett bis zum Zeitpunkt $\tau_e(v_k,j)-t'$ bearbeitet und schon zu diesem Zeitpunkt enden.

Also liegt der Releasezeitpunkt stets im Intervall, d.h $R(v_k) \le t_k$.

Zeitanforderungsanalyse

Das Lemma liefert uns bei vorgegebener Priorität ein hinreichendes Kriterium dafür, ob die Response für jede Aufgabe v_i kürzer ist als deren Deadline:

➔ Überprüfe mit aufsteigendem $w(v_k) \le t_k \le t_d^*(v_k)$ für jede Task v_k, ob

$$t_k \ge w(v_k) + \sum_{i=1}^{k-1} \left\lceil \frac{t_k}{P(v_i)} \right\rceil \cdot w(v_i)$$

Findet man ein solches t_k für jede Task v_k , dann halten alle Tasks ihre deadlines, im anderen Falle ist es nicht sicher.

Algorithmus -- Zeitanforderungsanalyse
Für jede Task v_k $k = 1,\ldots,n$
do for $i = 1,\ldots,k-1$
  for $j = 1,\ldots, \lfloor t_d^*(v_k)/P(v_i) \rfloor$
    do Prüfe die Ungleichung für $t_k = j*P(v_i)$ und exit falls ja.
  Prüfe die Ungleichung für $t_d^*(v_k)$
  -- s. Anmerkung

Einfach periodische Systeme

Anmerkung:

Der Algorithmus nutzt als Optimierung noch aus, dass für einen Wert $t_k < t_d^*(v_k)$, der die Ungleichung erfüllt, auch ein Vielfaches $t_k \le t'_k < t_d^*(v_k)$ mindestens einer der Perioden der Aufgaben höherer Priorität die Ungleichung erfüllt, falls es existiert. Man muss bis auf den Fall $t_k = t_d^*(v_k)$, also nur Vielfache von Perioden $P(v_i)$, $i < k$, probieren.

Die Laufzeit reduziert sich damit auf den pseudopolynomiellen Faktor $q := max\{P(v_i) | i \le n\} / min\{P(v_i) | i \le n\}$ zu $O(n^2q)$.

Dieser Faktor ist im allgemeinen kleiner als der Faktor

$$max\{ t_d^*(v_i) - w(v_i) \mid i \le n\},$$

den man bei einfacher sequientieller Suche nach t_k hätte.

Das Lemma liefert uns auch noch eine hübsche Folgerung:

Definition

Ein System periodischer Tasks mit $P(v_1) \le \ldots \le P(v_n)$ heißt **einfach periodisch**, wenn für alle Tasks v_i gilt: $P(v_{i+1})$ ist ein Vielfaches von $P(v_i)$ und $t_d^*(v_i) = P(v_i)$.

DM/RM Planung einfach periodischer Systeme

Satz

Ein System einfach periodischer Tasks hält unter DM oder RM Planung alle Deadlines solange die Auslastung U nicht über 100% ist.

Beweis: Sei $U \le 1$ d.h.

$$\forall k : \sum_{i=1}^{k} \frac{w(v_i)}{P(v_i)} \le U \le 1 \quad \Leftrightarrow \forall k : \frac{w(v_k)}{P(v_k)} + \sum_{i=1}^{k-1} \frac{w(v_i)}{P(v_i)} \le 1$$

$$\Leftrightarrow \forall k : w(v_k) + \sum_{i=1}^{k-1} \frac{P(v_k)}{P(v_i)} \cdot w(v_i) \le P(v_k)$$

$$\Leftrightarrow \forall k : w(v_k) + \sum_{i=1}^{k-1} \left\lceil \frac{P(v_k)}{P(v_i)} \right\rceil \cdot w(v_i) \le P(v_k)$$

da einfach periodisch.

Die Ungleichung des Lemmas gilt also spätestens für alle Deadlines (= Perioden) solange die Auslastung höchstens 100% ist.

Kritische Tasksysteme

Wir wollen eine Aussage über DM/RM-Planbarkeit allein von der Auslastung des Tasksystems her herleiten. Dazu wollen wir zunächst stets annehmen, dass $t_d^*(v) = P(v)$, und die Tasks nach aufsteigenden Perioden nummeriert sind. Wir können also ein System periodischer Tasks auffassen als ein Tupel *(p,w)* von *n*-stelligen Vektoren mit $p_i = P(v_i)$, $w_i = w(v_i)$ und $p_1 < \ldots < p_n$.

Definition:

Ein System periodischer Tasks *(p,w)* heißt ***p*-kritisch**, genau dann, wenn es ein Zeitintervall $[t - p_n, t]$ gibt, mit $t = t_d^*(v_n) + m*p_n$, in dem zu jedem Zeitpunkt unter DM-Planung ein Task auf dem Prozessor läuft.

Offenbar kann man bei einem *p*-kritischen System die Laufzeit keiner Task mehr erhöhen, ohne dass v_n seine Deadline verpasst:
Das Intervall endet zu einer Deadline von v_n und enthält zu jedem $i < n$ mindestens eine aktive Task. Würde man ein w_i erhöhen, würde die Intervalllänge nicht mehr ausreichen, um v_n zu beenden, da schon zu jedem Zeitpunkt eine Task läuft.

Kritische Tasksysteme -- Eigenschaften

Lemma -- krT1

Gegeben sei ein *p*-kritisches System periodischer Tasks *(p,w)* minimaler Auslastung mit $2p_1 \geq p_n$. Dann gilt:

$$p_1 = \sum_{i=1}^{n} w_i$$

Beweis:
Betrachte ein beliebiges Intervall $[t - p_1, t]$ das auf eine Deadline t von v_n hin endet. Wäre

$$p_1 > \sum_{i=1}^{n} w_i$$

dann wäre, da zu jedem $i \leq n$ höchstens eine Task in diesem Intervall läuft, der Prozessor für ein $\varepsilon > 0$ idle.
Das Intervall liegt aber stets innerhalb eines Intervalls $[t - p_n, t]$ wo t Deadline zu v_n ist.
Also gäbe es kein Intervall $[t - p_n, t]$, in dem zu jedem Zeitpunkt der Prozessor belegt ist. Damit gilt ≤.

Kritische Tasksysteme -- Eigenschaften

Beweis ff:
Wir nehmen nun an, es wäre $p_1 < \sum_{i=1}^{n} w_i$

dann gibt es ein $\varepsilon > 0$, mit

$$\varepsilon < \min\{\textstyle\sum_{i=1}^{n} w_i - p_1, \frac{w_n}{2}\}$$

Wir konstruieren nun ein neues Tasksystem indem wir setzen:

$$w_1 := w_1 + \varepsilon \text{ sowie } w_n := w_n - 2\varepsilon$$

Alle anderen w_i bleiben erhalten. Dann bleibt das kritische Intervall $[t - p_n, t]$ kritisch, da die Zeit, die wir v_n weggenommen haben zweimal von v_1 verbraucht wird.
Für die Auslastung gilt aber

$$U_{neu} - U = \frac{\varepsilon}{p_1} - \frac{2\varepsilon}{p_n} \leq 0$$

$$\Leftarrow \quad \frac{\varepsilon}{p_1} \leq \frac{2\varepsilon}{p_n} \quad \Leftarrow \quad p_n \leq 2p_1$$

Dies ist aber ein Widerspruch zur minimalen Auslastung, also gilt "="

Kritische Tasksysteme -- Eigenschaften ff

Lemma -- krT2

Gegeben sei ein *p*-kritisches System periodischer Tasks *(p,w)* mit $2p_1 \geq p_n$. Dann gilt:

$$w_n \geq p_n - 2\sum_{i=1}^{n-1} w_i$$

Beweis:
Es gibt ein Intervall $[t - p_n, t]$ das auf eine Deadline t von v_n hin endet, so dass der Prozessor stets durch eine Task belegt ist.
Da v_n in diesem Intervall wegen $2p_1 \geq p_n$ höchstens 2 mal von jeder anderen Task unterbrochen wird, und diese im schlimmsten Fall vollständig ablaufen können, bleibt für v_n mindestens

$$w_n \geq p_n - 2\sum_{i=1}^{n-1} w_i$$

Kritische Tasksysteme -- Eigenschaften ff

Lemma -- krT3

Gegeben sei ein *p*-kritisches System periodischer Tasks *(p,w)* mit $2p_1 \geq p_n$. Dann gilt:

Wenn das System niedrigstmögliche Auslastung hat, ist für $i < n$

$$w_i = p_{i+1} - p_i$$

Beweis: Induktion nach *i*

i = 1:

a) Wäre $w_1 = p_2 - p_1 - \varepsilon$ für $\varepsilon > 0$, dann wäre in jedem Intervall zwischen einem Start *t* von v_1 und $t+p_2$ zwei Starts von v_1 und höchstens ein Start aller anderen v_i.
Der Prozessor wäre also für mindestens

$$p_2 - (2w_1 + \sum_{i=2}^{n} w_i) = p_2 - (w_1 + p_1) \quad \text{-- Lemma krT1}$$
$$= p_2 - (p_2 - p_1 - \varepsilon + p_1) = \varepsilon$$

idle, im Widerspruch zu *p*-kritisch!

Kritische Tasksysteme -- Eigenschaften ff

Beweis -- ff

b) Wäre $w_1 = p_2 - p_1 + \varepsilon$ für ein $\varepsilon > 0$, könnte man ein Tasksystem mit $w'_1 = w_1 - \varepsilon$, $w'_n = w_n + \varepsilon$, und $w'_i = w_i$ sonst, konstruieren, das planbar bleibt, weil v_n die freie Rechenzeit bekommt.

Die Auslastung ändert sich um $U'-U = -\varepsilon/p_1 + \varepsilon/p_n < 0$, sinkt also. Damit hatte das System nicht minimale Auslastung unter der Nebenbedingung, *p*-kritisch zu sein.

$i \rightarrow i+1$:

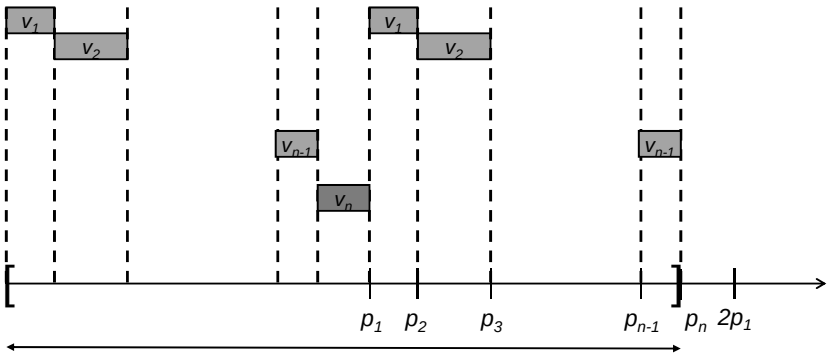
a) Wäre $w_{i+1} = p_{i+2} - p_{i+1} - \varepsilon$ für $\varepsilon > 0$, dann wäre wieder in jedem Intervall zwischen einem Start *t* von v_{i+1} und $t+p_{i+2}$ zwei Starts von v_i $i \leq i+1$ und höchstens ein Start aller v_j, $j > i+1$
Der Prozessor wäre mit I.A. für mindestens

$$p_{i+2} - (2\sum_{j=1}^{i+1} w_j + \sum_{j=i+2}^{n} w_j) \overset{krT1}{=} p_{i+2} - (\sum_{j=1}^{i} w_j + w_{i+1} + p_1)$$
$$\overset{I.A.}{=} p_{i+2} - (p_{i+1} - p_1 + p_{i+2} - p_{i+1} - \varepsilon + p_1) = \varepsilon$$

idle, im Widerspruch zu *p*-kritisch! Fall b) analog zur Verankerung!

Kritische Tasksysteme -- Beispiel

Beispiel: Ein kritisches Tasksystem minimaler Auslastung für $2p_1 \geq p_n$



kritisches Intervall, Prozessor voll ausgelastet

$$w_n = p_n - 2\sum_{i=1}^{n-1} w_i$$

Kritische Tasksysteme -- der Satz

Satz -- Layland & Liu

Ein System *n* periodischer Tasks mit $t_d^*(v) = P(v)$ ist stets korrekt DM- bzw. RM-planbar, solange für die Auslastung gilt

$$U \leq n(2^{\frac{1}{n}} - 1)$$

Beweis:
Wir zeigen den Satz zunächst für Tasksysteme mit $2p_1 \geq p_n$, wo p_1 die minimale Periode und p_n die maximale Periode ist.
Dazu betrachten wir ein *p*-kritisches Tasksystem mit minimaler Auslastung. Für dieses gilt mit den letzten Lemmata

$$w_i = p_{i+1} - p_i \quad \text{sowie} \quad w_n = p_n - 2\sum_{i=1}^{n-1} w_i = p_n - 2\sum_{i=1}^{n-1}(p_{i+1} - p_i)$$
$$= p_n - 2(p_n - p_1) = 2p_1 - p_n$$

Also ist die Auslastung

$$U_{min}(p) = \sum_{i=1}^{n-1} \frac{p_{i+1} - p_i}{p_i} + \frac{2p_1 - p_n}{p_n} = \sum_{i=1}^{n-1} \frac{p_{i+1}}{p_i} + \frac{2p_1}{p_n} - n$$

Kritische Tasksysteme -- der Satz ff

Beweis -- ff:
Nun müssen wir die Auslastung noch über alle Periodenverteilungen *p* minimieren.
Da U_{min} konvex ist, finden wir die minimale Periodenverteilung *q* mittels

$$\frac{\partial U_{min}}{\partial p_i}(q) = 0 \text{ für } 1 \le i \le n$$

für $1 < i < n$ ist

$$\frac{\partial U_{min}}{\partial p_i} = \frac{\partial \sum_{i=1}^{n-1} \frac{p_{i+1}}{p_i} + \frac{2p_1}{p_n} - n}{\partial p_i} = -\frac{p_{i+1}}{p_i^2} + \frac{1}{p_{i-1}}$$

woraus für *q* folgt:

$$\boxed{\frac{q_{i+1}}{q_i} = \frac{q_i}{q_{i-1}} =: \delta}$$

Kritische Tasksysteme -- der Satz ff

Beweis -- ff:
Für $i = 1$ ist

$$\frac{\partial U_{min}}{\partial p_1} = \frac{\partial \sum_{i=1}^{n-1} \frac{p_{i+1}}{p_i} + \frac{2p_1}{p_n} - n}{\partial p_1} = -\frac{p_2}{p_1^2} + \frac{2}{p_n}$$

woraus für *q* folgt:

$$\boxed{\delta = \frac{q_2}{q_1} = 2\frac{q_1}{q_n}}$$

d.h.

$$\boxed{\delta q_n = 2q_1}$$

und für $i = n$ ist schliesslich

$$\frac{\partial U_{min}}{\partial p_n} = \frac{\partial \sum_{i=1}^{n-1} \frac{p_{i+1}}{p_i} + \frac{2p_1}{p_n} - n}{\partial p_n} = \frac{1}{p_{n-1}} - \frac{2p_1}{p_n^2}$$

woraus für *q* wiederum folgt:

$$\boxed{\delta = \frac{q_n}{q_{n-1}} = 2\frac{q_1}{q_n}}$$

Kritische Tasksysteme -- der Satz ff

Beweis -- ff:
Setzt man nun all dies zusammen, erhält man für $1 < i \le n$

$$q_i = \delta \cdot q_{i-1} = \delta^2 \cdot q_{i-2} = \cdots = \delta^{i-1} \cdot q_1$$

und mit $\delta q_n = 2q_1$: $\delta^n q_1 = 2q_1 \quad \Leftrightarrow \boxed{\delta = 2^{\frac{1}{n}}}$

Damit haben wir aber für die minimale Auslastung

$$U_{min} := U_{min}(q) = \sum_{i=1}^{n-1} \frac{q_{i+1}}{q_i} + \frac{2q_1}{q_n} - n \qquad -- \quad \delta = \frac{q_{i+1}}{q_i} = 2\frac{q_1}{q_n}$$

$$= \sum_{i=1}^{n-1} \delta + \delta - n = n \cdot \delta - n \quad \boxed{= n \cdot (2^{\frac{1}{n}} - 1)}$$

Damit gilt der Satz für Tasksysteme mit $p_n \le 2p_1$

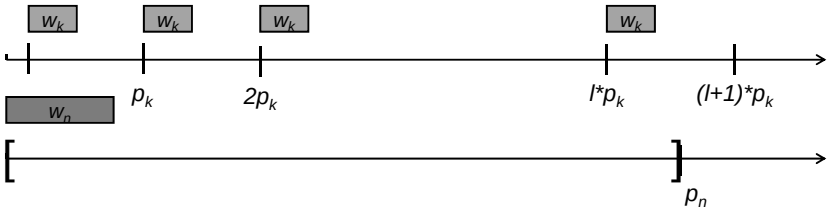
Kritische Tasksysteme -- der Satz ff

Für Tasksysteme mit einem maximalen Periodenverhältnis > 2 zeigen wir, dass ein beliebiges solches Tasksystem planbar ist, wenn ein Tasksystem mit niedrigerem Periodenverhältnis und niedrigerer Auslastung planbar ist. Damit folgt dann der Satz.

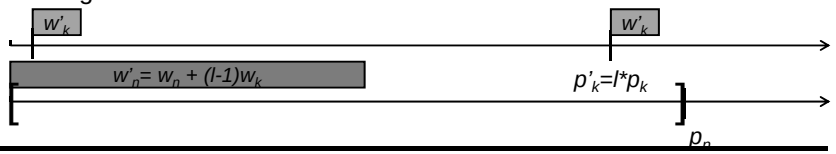
Kritische Tasksysteme -- der Satz ff

Wir betrachten ein Tasksystem mit Periodenverhältnis > *2* und p_k sei die größte Periode mit $l*p_k < p_n \leq (l+1)p_k$, $l \geq 2$

Wir betrachten ein Intervall $[t,t+p_n]$ an dessen Ende eine Deadline für p_n liegt:



Wir erhöhen nun die Periode p_k auf $l*p_k$ und transformieren das System wie folgt:



Kritische Tasksysteme -- der Satz ff

Wenn das neue Tasksystem planbar ist, ist sicher auch das alte planbar, da die $(l-1)*w_k$ Zeitanteile von v_n dort mit höherer Priorität laufen.

Für die Auslastung des neuen Systems gegenüber vorher gilt aber

$$U' - U = \frac{w_k}{l \cdot p_k} - \frac{w_k}{p_k} + \frac{(l-1)w_k}{p_n}$$

$$= \frac{w_k}{l \cdot p_k} - \frac{l \cdot w_k}{l \cdot p_k} + \frac{(l-1)w_k}{p_n} \quad = -\frac{(l-1)w_k}{l \cdot p_k} + \frac{(l-1)w_k}{p_n}$$

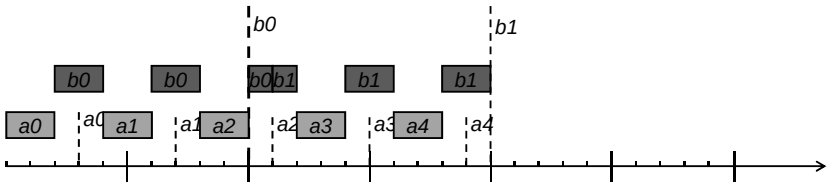
$$= \underbrace{(\frac{1}{p_n} - \frac{1}{l \cdot p_k})}_{< 0,\ \text{da } l*p_k < p_n}(l-1)w_k$$

Also sinkt mit $l \geq 2$ die Auslastung für die neue Task, aus deren Planbarkeit die Planbarkeit der alten Task folgt. Wir iterieren nun das Argument solange bis $2p_1 \geq p_n$ gilt. Damit gilt dann der Satz.

Dynamische versus statische Priorität

Beispiel: Wir betrachten die Tasks *a,b* mit (t_r^*, w, t_d^*, P)
(0,2,3,4), (0,5,10,10),

sowohl ratenmonotone, als auch deadlinemonotone Planung würden zuerst *a*, dann *b* planen:



!! *(b,2i)* verpasst seine Deadline !!

Grund: Die Vergabe der Priorität erfolgt statisch, ohne Rücksicht auf die absoluten Deadlines. *b* sollte zwischen Zeitpunkt 6 und 9 eine höhere Priorität als *a* haben, sonst eine niedrigere.

Dynamische versus statische Priorität -- ff

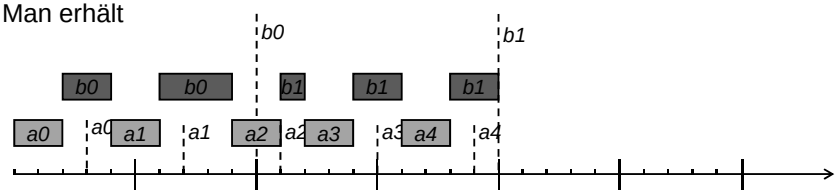
EDF-Planung lässt sich auch für den periodischen Fall einsetzen. Man vergibt als Priorität einer Task *(v,n)* einfach deren absolute Deadline - $t_d(v,n) = -(t_d^*(v) + t_r^*(v) + n*P(v))$.

Beispiel: Wir betrachten wieder die Tasks *a,b* mit (t_r^*, w, t_d^*, P)
(0,2,3,4), (0,5,10,10),

Nun ergeben sich die Prioritäten von

- 3, -7, -11, -15, -19 für a0, a1, a2, a3, a4 und -10 für b0, -20 für b1.

Man erhält



Alle Deadlines sind eingehalten. Man beachte auch, dass diese Art dynamischer Priorität relativ effizient zu implementieren ist, da die Priorität eines Jobs nach Eintritt in die Warteschlange fest bleibt.

Dynamische versus statische Priorität -- ff

In unserem Beispiel gelang also mit statischer Priorität kein gültiger Ablaufplan. Eine Ursache dafür ist, dass die Auslastung des Systems sehr hoch ist (100%).

Frage1: Liefert EDF Planung stets einen Ablaufplan, der alle Deadlines hält, solange die Auslastung kleiner oder gleich 1 ist?

Ja, aber andererseits haben dynamische Verfahren, wie EDF den Nachteil der Unvorhersagbarkeit, wenn das System überlastet ist. Eine Ursache liegt darin, dass Aufgaben, die ihre Deadline sowieso verpassen müssen, nun mit höherer Priorität laufen, als Aufgaben, die ihre Deadline einhalten könnten!

Frage2: Kann man Bedingungen angeben, unter denen statische Prioritäten, wie DM bzw. RM Pläne, garantiert ihre Deadlines halten?

Ja! s. letzter Satz.

Ein einfaches Belegungslemma

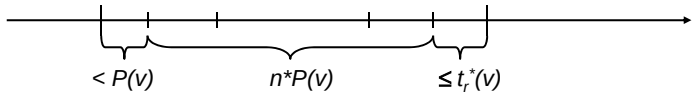
Lemma

Gegeben sei eine Menge von periodischen Tasks *V* ohne Datenabhängigkeiten mit $t_d^*(v)=P(v)$. Dann belegt jede Aufgabe *v*, die in einem Intervall der Länge *t* mindestens eine Deadline hat, die CPU in einem korrekten Ablaufplan für mindestens

$$\left\lfloor \frac{t - t_r^*(v)}{P(v)} \right\rfloor \cdot w(v)$$

Zeiteinheiten.

Beweis:
Ein Intervall der Länge *t*, das mindestens eine Deadline zu *v* enthält, umfasst mindestens *n* Perioden von *v* komplett, mit



Also ist $n*P(v) \leq t - t_r^*(v) < (n+1)P(v)$

Ein einfaches Belegungslemma -- ff

Beweis ff
Damit gilt aber

$$n \leq \frac{t - t_r^*(v)}{P(v)} < n+1$$

Nun muss aber jede der echt enthaltenen Perioden komplett auf der CPU abgearbeitet werden, und belegt sie für *w(v)* Zeiteinheiten.

Also belegt *v* die CPU für mindestens

$$\left\lfloor \frac{t - t_r^*(v)}{P(v)} \right\rfloor \cdot w(v)$$

Zeiteinheiten.

Optimalität von EDF falls $P(v) = t_d^*(v)$

Satz -- Layland und Liu

Gegeben sei eine Menge von periodischen Tasks *V* ohne Datenabhängigkeiten mit $P(v) = t_d^*(v)$, $\forall\, v \in V$. Dann liefert dynamisches EDF Scheduling einen korrekten Ablaufplan, solange für die Auslastung gilt

$$1 \geq U := \sum_{v \in V} \frac{w(v)}{P(v)}$$

Beweis:
Wir zeigen, dass die Auslastung größer als 1 sein muss, wenn dynamisches EDF Scheduling eine Deadline verpasst.
Sei *(v,n)* die erste Aufgabe, die ihre Deadline zu einem Zeitpunkt $t=t_r^*(v)+(n+1)*P(v)$ verpasst.
Sei ferner *V'* die Teilmenge der Tasks, die eine Deadline innerhalb des Zeitraums $[t_r^*(v)+n*P(v), t_r^*(v)+(n+1)*P(v)]$ hatten.
Dann belegen diese Aufgaben nach dem letzten Lemma, da sie ihre Deadlines alle halten, in diesem Intervall die CPU für mindestens

$$\sum_{u \in V'} \frac{P(v)}{P(u)} \cdot w(u) \geq X \geq \sum_{u \in V'} \left\lfloor \frac{P(v) - t_r^*(u)}{P(u)} \right\rfloor \cdot w(u)$$ Zeiteinheiten.

Optimalität von EDF -- ff

Beweis ff:
Da aber *(v,n)* selbst in diesem Intervall nicht komplett ablaufen kann, gilt

$$P(v) < w(v) + X$$

und erst recht

$$P(v) < w(v) + \sum_{u \in V'} \frac{P(v)}{P(u)} \cdot w(u)$$

$$\Leftrightarrow 1 < \underbrace{\frac{w(v)}{P(v)} + \sum_{u \in V'} \frac{w(u)}{P(u)}}_{\leq U}$$

Also ist schon die Auslastung $U > 1$.

Test auf EDF-Planbarkeit

Man kann aus dem letzten Satz auch leicht ein hinreichendes Kriterium für die korrekte EDF-Planbarkeit ableiten:

Satz -- Liu

Gegeben sei eine Menge von periodischen Tasks *V* ohne Datenabhängigkeiten. Dann liefert dynamisches EDF Scheduling einen Ablaufplan, der alle Deadlines einhält, falls

$$1 \geq \sum_{v \in V} \frac{w(v)}{t_d^*(v)}$$

Beweis:
Wir zeigen, dass die Bedingung verletzt sein muss, wenn dynamisches EDF Scheduling eine Deadline verpasst.
Sei *(v,n)* die erste Aufgabe, die ihre Deadline zu einem Zeitpunkt $t=t_d^*(v)+t_r^*(v)+n*P(v)$ verpasst, und sei wieder *V'* die Teilmenge der Tasks, die eine Deadline innerhalb des Zeitraums $[t_r^*(v)+n*P(v), t_d^*(v)+t_r^*(v)+n*P(v)]$ hatten.
Dann belegen diese Aufgaben, da sie ihre Deadlines alle halten, in diesem Intervall die CPU für mindestens $\sum_{u \in V'} \left\lfloor \frac{t_d^*(v)-t_r^*(u)}{P(u)} \right\rfloor \cdot w(u)$

Test auf EDF-Planbarkeit -- ff

Beweis ff:
Da aber *(v,n)* selbst in diesem Intervall nicht komplett ablaufen kann, gilt

$$t_d^*(v) < w(v) + \sum_{u \in V'} \left\lfloor \frac{t_d^*(v)-t_r^*(u)}{P(u)} \right\rfloor \cdot w(u)$$

und wegen $P(u) \geq t_d^*(u)$ stets, erst recht

$$t_d^*(v) < w(v) + \sum_{u \in V'} \frac{t_d^*(v)}{t_d^*(u)} \cdot w(u)$$

$$\Rightarrow 1 < \frac{w(v)}{t_d^*(v)} + \sum_{u \in V'} \frac{w(u)}{t_d^*(u)}$$

$$\Rightarrow 1 < \sum_{u \in V} \frac{w(u)}{t_d^*(u)}$$

Bemerkung: Das Kriterium ist allerdings nur hinreichend für eine korrekten Ablauf. Ist es verletzt, kann ein EDF Plan dennoch ggf. alle Deadlines einhalten. Es eignet sich aber als einfache Richtlinie oder als effizienter **On-line Test** (Sicherheit).