

Echtzeitsysteme

Zustellerkonzepte & Übernahmeprüfung

Peter Ulbrich

Lehrstuhl Informatik 4

02. Dezember 2014

Gliederung

- 1 Überblick
- 2 Bandweite-bewahrende Zusteller
- 3 Übernahmeprüfung
- 4 Zusammenfassung

Fragestellungen

- Wie behandelt man am besten nicht-periodische Jobs in vorranggesteuerten Systemen?
 - Das **Stehlen von Schlupf** scheidet ja leider aus - zu komplex.
 - **Hintergrundbetrieb** und **abfragende Zusteller** sind zu langsam.
 - **Vordegrundbetrieb** beeinflusst periodische Aufgaben zu sehr.
- Der Schlüssel liegt in der **Bewahrung des Budgets**
 - Wie und wann wird das Budget **verbraucht**?
 - Wie und wann wird das Budget **wieder aufgefüllt**?
 - Wie stark ist der **Einfluss auf periodische Aufgaben**?
- Wir können nun die Antwortzeit für aperiodische Arbeitsaufträge optimieren, aber was machen wir mit sporadischen Jobs?
 - Wie sehen **Akzeptanztests** eigentlich aus?

☞ heute geht es um **vorranggesteuerte Systeme!**

Gliederung

- 1 Überblick
- 2 Bandweite-bewahrende Zusteller
- 3 Übernahmeprüfung
- 4 Zusammenfassung

Abfragender Zusteller (siehe Folie V-1/16)

Vor- und Nachteile, Fluch und Segen

Verhält sich im schlimmsten Fall genau wie eine periodischen Aufgabe

- Das vereinfacht die Analyse abfragender Zusteller ungemein. . .

👉 Beschränkter, exakt quantifizierbarer Einfluss auf andere Aufgaben!

- . . . hat aber auch Nachteile

👉 Abfragende Zusteller liefern unbefriedigende Antwortzeiten!

- Grund ist der Verlust des Ausführungsbudgets
 - sobald der Zusteller untätig wird, verfällt sein Budget
 - später eintreffende Jobs müssen aufgeschoben werden

↪ Restbudget auch in Phasen der Untätigkeit bewahren

- sich aber schlimmstenfalls so verhalten, wie eine periodische Aufgabe

👉 bandweite-bewahrende Zusteller (engl. *bandwidth-preserving servers*)

Verbesserung des Abfragebetriebs

Restbudget untätig gewordener Zusteller in der Abfrageperiode nicht verfallen lassen

Zusteller, die ihr Ausführungsbudget (d.h., ihre Bandweite) innerhalb ihrer Abfrageperiode bewahren \mapsto engl. *bandwidth-preserving server*

- definieren sich durch bestimmte **Regeln** zum. . .

Verbrauch (engl. *consumption rules*) des Budgets

- Bedingungen, unter denen das Budget bewahrt/verbraucht wird

Auffüllen (engl. *replenishment rules*) des Budgets

- Festlegungen, *wann* und *wie* das Budget aufgefüllt wird

- werden nach folgendem Schema vom System verarbeitet:

- der Planer (Betriebssystem) führt Buch über den Budgetverbrauch
 - suspendiert den Zusteller, wenn das Budget verbraucht wurde
 - stellt den Zusteller bereit, wenn das Budget aufgefüllt wurde
- der Zusteller setzt sich selbst aus, wenn er untätig wird
 - Restbudget zum Zeitpunkt des Untätigwerdens bleibt ihm erhalten
 - wird ausführungsbereit, falls er *zurückgestellt* (s. V-1/15) wird

Aufschiebbarer Zusteller (engl. *deferrable server*)

Bewahrung des Restbudgets zum Zeitpunkt des Untätigwerdens – vgl. [3, S. 195]

Deferrable Server $\mapsto T_D = (p_s, e_s)$

- periodisches Auffüllen von Budget e_s mit Periode p_s (s. V-1/16)
- Bewahrung des (Rest-) Budgets von T_D in p_s bei Untätigkeit
- keine Akkumulation des Restbudgets von Periode zu Periode
 - am Ende der Abfrageperiode verfällt ggf. vorhandenes Restbudget

Verbrauchsregel Wann immer der Zusteller ausgeführt wird, verbraucht er sein Ausführungsbudget mit einer Rate $1/\text{Zeiteinheit}$.

Auffüllregel Das Ausführungsbudget des Zustellers wird zu den Zeitpunkten $k \cdot p_s$ auf e_s gesetzt, für $k = 0, 1, 2, \dots$

Beispiel: Aufschiebbarer Zusteller (siehe Folie. V-1/18)

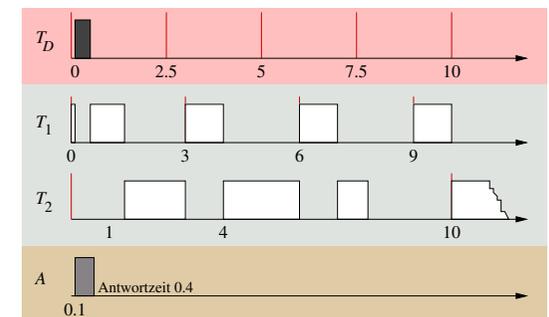
Optimiertes Antwortverhalten

periodische Tasks:

- $T_D = (2.5, 0.5)$
- $T_1 = (3, 1)$
- $T_2 = (10, 4)$
- RM

aperiodischer Job:

- $A \mapsto 0.4 [0.1, \infty)$



- das Budget von 0.5 Zeiteinheiten bleibt T_D erhalten
 - obwohl T_D zum Zeitpunkt t_0 untätig ist
 - unmittelbare Abarbeitung des Jobs A zum Zeitpunkt $t_{0,1}$
- eine Akkumulation des Budgets ist nicht vorgesehen
 - eine Übertragung des Restbudgets von 0.1 Zeiteinheiten in die Abfrageperiode $t_{2,5}$ ist nicht vorgesehen

Beispiel: Aufschiebbarer Zusteller

Budgetverbrauch und -auffüllung

periodische Tasks:

- $T_D = (3, 1)$
- $T_1 = (3.5, 1.5, 3.5, 2)$
- $T_2 = (6.5, 0.5)$
- RM

aperiodischer Job:

- $A \mapsto 1.7 [2.8, \infty)$

Verlauf:

t_0 T_D startet & wartet

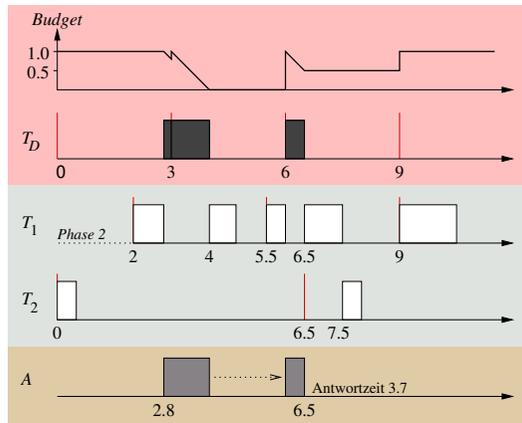
$t_{2.8}$ A wird zugestellt,
 T_D verbraucht

t_3 T_D kommt weiter in Frage

t_6 T_D kommt erneut in Frage

t_4 T_D wird vom Planer gestoppt

$t_{6.5}$ A ist beendet, T_D untätig



Beispiel: Aufschiebbarer Zusteller – EDF

Budgetverbrauch und -auffüllung — alternatives Einplanungsverfahren

periodische Tasks:

- $T_{D,1,2}$ vgl. Folie 9
- Termine = Perioden
- EDF

aperiodischer Job:

- $A \mapsto 1.7 [2.8, \infty)$

Verlauf:

t_0 T_D startet & wartet

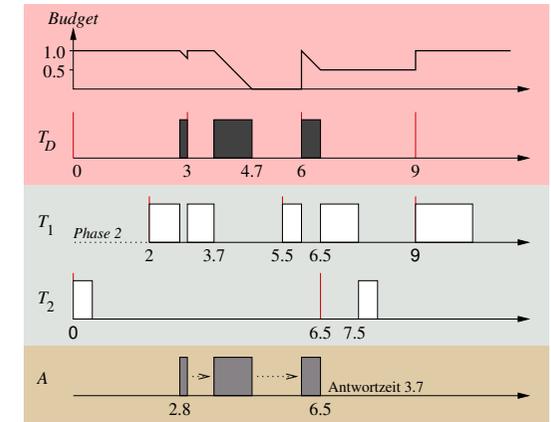
$t_{2.8}$ A wird zugestellt,
 T_D verbraucht

t_3 T_1 hat früheren Termin (5.5)

t_6 $d_1 = d_D$, T_D wird bevorzugt

$t_{3.7}$ T_D wird weiter ausgeführt

$t_{6.5}$ A ist beendet, T_D untätig



Aufschiebbarer Zusteller \cup Hintergrundzusteller

Budgetverbrauch und -auffüllung — Antwortzeitverbesserung

Beispiel (vgl. Folie 10):

Background Server

- verarbeitet die AJQ
 - unterstützt T_D
 - Abfragervariante
- niedrigste Priorität

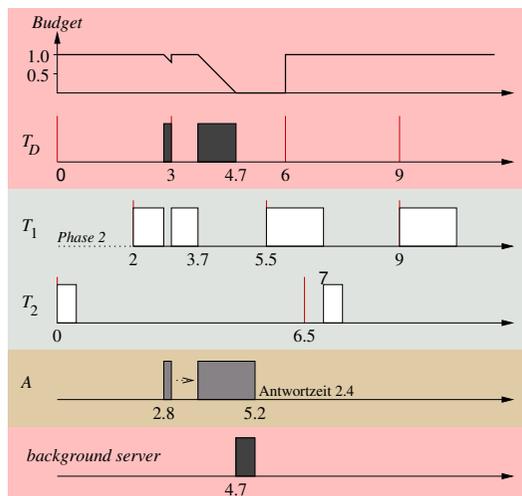
Verlauf:

$t_{4.7}$ keine periodische Task
ist ausführbar

Hintergrundbetr.

$t_{5.2}$ A ist beendet

- T_D bleibt untätig



Aufschiebbarer Zusteller — Größenbeschränkung

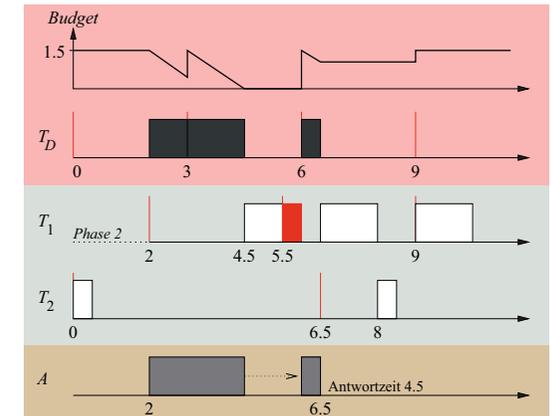
Einfluss auf die Planbarkeit periodischer Aufgaben

Verbesserung der Ansprechempfindlichkeit durch eine Vergrößerung des Budgets, anstatt Einsatz eines Hintergrundzusteller, ist problematisch:

Beispiel (vgl. Folie 9):

- $T_D = (3, 1.5)$
- $A \mapsto 3 [2, \infty)$
- T_1 verpasst Termin

Das Budget ist unter Berücksichtigung **aller möglichen Kombinationen** von Auslösezeiten aller (periodischen) Tasks zu bestimmen.



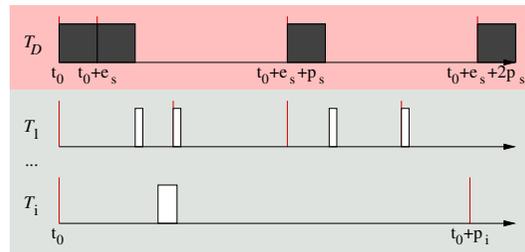
Aufschiebbarer Zusteller != periodische Aufgabe

Aufschiebbare Zusteller erfordern daher eine spezielle Planbarkeitsanalyse

Beispiel: Antwortzeitanalyse für statische Prioritäten (s. Folie IV-2/40)

- aufschiebbarer Zusteller $T_D = (p_s, e_s)$ sei gegeben

- mit der **höchsten Priorität** aller periodischen Aufgaben



- ☞ ein **kritischer Zeitpunkt** einer periodischen Aufgabe T_i tritt zum Zeitpunkt t_0 unter folgenden Bedingungen ein (s. Folie IV-2/44)
 - ein Arbeitsauftrag $J_{i,j}$ dieser Aufgabe wird ausgelöst
 - Jobs aller Aufgaben höherer Priorität T_1, \dots, T_{i-1} werden ausgelöst
 - das Budget von T_D ist e_s und T_D ist zurückgestellt
 - der nächste Auffüllzeitpunkt von T_D ist $t_0 + e_s$ (engl. *double hit*)

Aufschiebbarer Zusteller != periodische Aufgabe (Forts.)

Dies hat direkte Auswirkung auf die Berechnung der Antwortzeit

- erweiterte, iterative Bestimmung der Antwortzeit (s. Folie IV-2/41)

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

- $e_D(t)$ ist die vom aufschiebbaren Zusteller T_D verursachte Störung

$$e_D(t) = \begin{cases} e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s & \text{Priorität } P_i \text{ von } T_i \text{ ist kleiner als } P_D \\ 0 & \text{sonst} \end{cases}$$

- die Störung ist damit bis zu e_s Zeiteinheiten größer als bei einer periodischen Aufgabe mit identischen Parametern

☞ Zusteller, der sich so verhält, wie eine periodische Aufgabe
 ~> **Sporadischer Zusteller** (engl. *sporadic server*)

Sporadischer Zusteller (engl. *sporadic server*)

Sporadic Server $\mapsto T_S = (p_s, e_s)$

- beansprucht niemals mehr Prozessorzeit als die periodische Aufgabe $T = (p_s, e_s)$ in jedem Zeitintervall
- kann daher auch genau wie die periodische Aufgabe T behandelt werden, wenn auf Planbarkeit des Tasksystems geprüft wird
- ermöglicht Planbarkeit eines Systems periodischer Aufgaben, das bei Verwendung eines aufschiebbaren Zustellers nicht planbar wäre
- kommt in verschiedenen Ausführungen vor, die sich vor allem in ihren Verbrauchs- und Auffüllregeln unterscheiden:

einfach (engl. *simple*)

kumulativ (engl. *cumulative*) längere Budgetbewahrung

SpSL (Sprunt, Sha & Lehoczky) aggressiveres Auffüllen ✓

termingesteuert (engl. *deadline-driven*) läuft mit höherer Priorität

SpSL Sporadic Server [4]

Definitionen

P_{cur} ist die aktuelle Prioritätsebene des Systems.

P_s ist die Prioritätsebene des Zustellers T_s .

P_i ist eine beliebige Prioritätsebene.

- Prioritäten sind absteigend nummeriert: $P_1 \succ P_2 \succ \dots \succ P_n$

tätig Eine Zeitspanne, in der eine Prioritätsebene P_i tätig ist.

- Die Prioritätsebene P_i ist tätig, solange die aktuelle Systempriorität mind. so hoch ist wie P_i : $P_{cur} \succeq P_i$.

untätig Die Prioritätsebene P_i ist in diesem Zeitraum nicht tätig.

- Die aktuelle Systempriorität ist niedriger als die Prioritätsebene P_i : $P_{cur} \prec P_i$.

rt_i Der Auffüllzeitpunkt für die Prioritätsebene P_i , **verbrauchtes Ausführungsbudget** wird zu diesem Zeitpunkt wiederhergestellt.

SpSL Sporadic Server (Forts.)

Verbrauchs- und Auffüllregeln

Verbrauchsregeln

Wann immer der Zusteller ausgeführt wird verbraucht er sein Ausführungsbudget mit einer Rate 1/Zeiteinheit.

Auffüllregeln

R1 Intial wird das Ausführungsbudget auf e_s gesetzt.

R2 Der nächste Auffüllzeitpunkt rt_s für T_s wird jeweils auf $t_b + p_s$ gesetzt. t_b ist dabei der Zeitpunkt, an dem

- T_s besitzt Budget $\leadsto P_s$ wird tätig
- T_s besitzt kein Budget $\leadsto T_s$ Budget wird > 0 & P_s ist tätig

R3 Die nächste Auffüllung wird zum Zeitpunkt rt_s eingeplant.

- Wenn P_s untätig wird oder T_s sein Budget erschöpft.
- So viel Budget wird aufgefüllt, wie T_s seit t_b verbraucht hat.

SpSL Sporadic Server (Forts.)

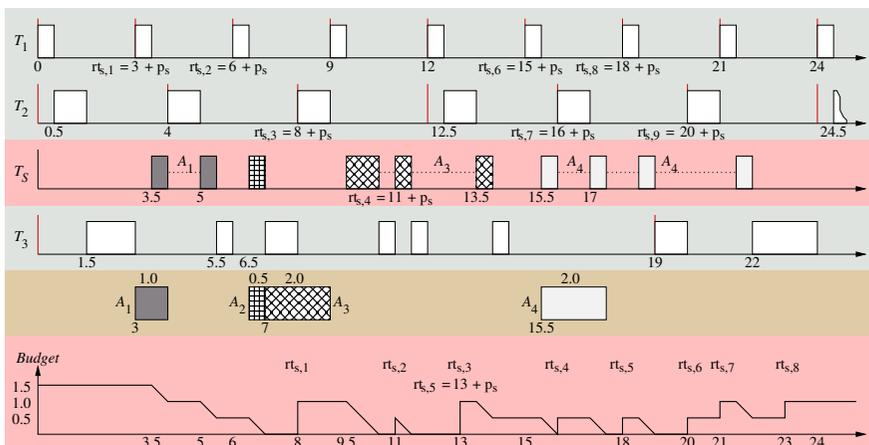
Aktionen des Planers

- 1 Überwache Tätigkeit/Untätigkeit der einzelnen Prioritätsebenen P_i
 - aktualisiere insbesondere rt_s , den nächsten Auffüllzeitpunkt von T_s
- 2 Überwache und protokolliere den Verbrauch des Budgets von T_s
 - suspendiere T_s , falls sein Budget erschöpft wird
 - stelle T_s bereit, falls sein Budget aufgefüllt wird
 - bestimme den Betrag, um den das Budget aufgefüllt wird
- 3 Verwalte **anstehende Auffüllungen** (engl. *pending replenishments*)
 - T_s muss sein Budget nicht auf einmal komplett aufbrauchen
 - dies kann auf mehrere Etappen geschehen
 - \leadsto jede Etappe wird aber einzeln aufgefüllt (s. Folie 17, Regel R3)
 - das Budget von T_s wird in **Scheiben** (engl. *chunks*) zerschnitten
 - \leadsto Planer muss eine Liste anstehender Auffüllungen verwalten
 - unter bestimmten Umständen können Scheiben auch vereinigt werden

SpSL Sporadic Server \approx Menge periodischer Aufgaben $\{T_i\}$, hierbei gilt: Periode $p_i = p_s$, Ausführungszeit $\sum_i e_i = e_s$

Beispiel: SpSL Sporadic Server

$T_1 = (3, 0.5)$, $T_2 = (4, 1)$, $T_3 = (19, 4.5)$ und $T_s = (5, 1.5)$; RM-Ablaufplanung



- jeder Auffüllzeitpunkt $rt_{s,i}$ entspricht einer Scheibe des Budgets e_s

Beispiel: SpSL Sporadic Server (Forts.)

Budgetverbrauch und -auffüllung

- $t_{3,5}$ T_s startet: $t_b = 3 \leadsto rt_{s,1} = 8$ (R2)
- T_1 startet zum Zeitpunkt $t_3 \leadsto P_s$ wurde tätig
- $t_{5,5}$ T_s wird untätig, an $rt_{s,1}$ wird 1 Zeiteinheit aufgefüllt (R3)
- $t_{6,5}$ T_s startet: $t_b = 6 \leadsto rt_{s,2} = 11$ (R2)
- T_1 startet zum Zeitpunkt $t_6 \leadsto P_s$ wurde tätig
- t_7 Budget erschöpft, an $rt_{s,2}$ werden 0.5 Einheiten aufgefüllt (R3)
- $rt_{s,1} = t_8$ Budgetauffüllung, T_s wird ausführungsbereit
- T_1 und T_2 mit höherer Priorität $\leadsto T_s$ wird noch nicht ausgeführt
- $t_{9,5}$ T_s startet: $t_b = 8 \leadsto rt_{s,3} = 13$ (R2)
- T_2 startet zum Zeitpunkt $t_8 \leadsto P_s$ wurde tätig
- $t_{10,5}$ Budget erschöpft, an $rt_{s,3}$ wird 1 Einheit aufgefüllt (R3)
- $rt_{s,2} = t_{11}$ Budgetauffüllung, T_s wird ausführungsbereit: $t_b = 11 \leadsto rt_{s,4} = 16$ (R2)
- T_1 und T_2 nicht ausführungsbereit $\leadsto T_s$ startet
 - $\leadsto T_s$ wird zum Zeitpunkt t_{11} tätig
- $t_{11,5}$ Budget erschöpft, an $rt_{s,4}$ werden 0.5 Einheiten aufgefüllt (R3)

Beispiel: SpSL Sporadic Server (Forts.)

Budgetverbrauch und -auffüllung

- $rt_{s,3} = t_{13}$ **Budgetauffüllung**, T_s wird ausführungsbereit
- $t_{13.5}$ T_s startet: $t_b = 13 \rightsquigarrow rt_{s,5} = 18$ (R2)
- zwar ist P_s bereits seit t_{12} tätig, aber T_s besitzt kein Budget
 - \rightsquigarrow Auffüllzeitpunkt $rt_{s,3}$ dient als Basis für $rt_{s,5}$
- t_{14} T_s wird untätig, an $rt_{s,5}$ werden 0.5 Einheiten aufgefüllt (R3)
- $t_{15.5}$ T_s startet: $t_b = 15 \rightsquigarrow rt_{s,6} = 20$ (R2)
- T_1 startet zum Zeitpunkt $t_{15} \rightsquigarrow P_s$ wurde tätig
- t_{16} **Budget erschöpft**, an $rt_{s,6}$ werden 0.5 Einheiten aufgefüllt (R3)
- $rt_{s,4} = t_{16}$ **Budgetauffüllung**, T_s wird ausführungsbereit
- t_{17} T_s startet: $t_b = 16 \rightsquigarrow rt_{s,7} = 21$ (R2)
- T_2 startet zum Zeitpunkt $t_{16} \rightsquigarrow P_s$ wurde tätig
- $t_{17.5}$ **Budget erschöpft**, an $rt_{s,7}$ werden 0.5 Einheiten aufgefüllt (R3)
- $rt_{s,5} = t_{18}$ **Budgetauffüllung**, T_s wird ausführungsbereit
- $t_{18.5}$ T_s startet: $t_b = 18 \rightsquigarrow rt_{s,8} = 23$ (R2)
- T_1 startet zum Zeitpunkt $t_{18} \rightsquigarrow P_s$ wurde tätig
- t_{19} **Budget erschöpft**, an $rt_{s,8}$ werden 0.5 Einheiten aufgefüllt (R3) ...

POSIX Sporadic Server

Emanzipation des SpSL Sporadic Servers in der realen Welt

In Anlehnung an den SpSL Sporadic Server wurde im Standard POSIX 1003.1d [2] der **POSIX Sporadic Server (PSS)** spezifiziert. Der Standard bietet den PSS als Einplanungsvariante **SCHED_SPORADIC** an. Bekannte Echtzeitbetriebssysteme implementieren diesen Standard:

- Wind River – VxWorks
- QNX Software Systems – QNX Neutrino RTOS
- Xenomai – eine Echtzeiterweiterung für Linux

☹ Dummerweise ist der PSS-Algorithmus fehlerhaft [5]

☹ ein PSS verhält sich nicht immer wie eine periodische Aufgabe

☹ Fehlersymptome im PSS-Algorithmus

- Anhäufung des Ausführungsbudgets (engl. *budget amplification*)
- Verfrühte Auffüllung des Budgets (engl. *premature replenishment*)
- Unzureichende zeitliche Isolation (engl. *unreliable temporal isolation*)

Anhäufung des Ausführungsbudgets

- eine exakte Überwachung des Ausführungsbudgets ist sehr aufwendig
- \rightsquigarrow POSIX beschränkt die Ausführungszeit eines PSS

to at most its available execution capacity, plus the resolution of the execution time clock used for this scheduling policy

- vereinfachte Überwachung als Preis einer effizienten Implementierung
- \rightsquigarrow **kleine Überläufe** werden in Kauf genommen
- zusammen mit weiteren, irrtümlichen Annahmen...

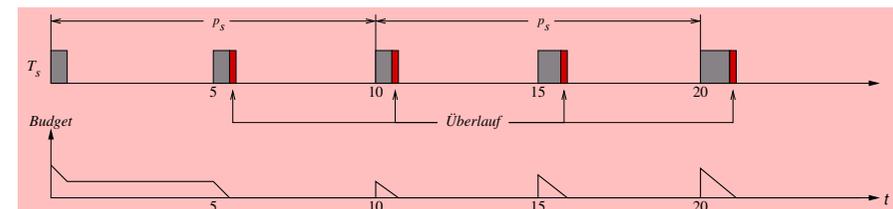
... reaches the limit imposed on its execution time ... the execution time consumed is subtracted from the available execution capacity (which becomes zero).

... If the available execution capacity would become negative by this operation ... it shall be set to zero.

... the execution capacity would become larger than ... initial budget, it shall be rounded down to a value equal to ... initial budget.

Anhäufung des Ausführungsbudgets

- ... führt dies zu einer **Ausweitung des Ausführungsbudgets**
- Bei Überläufen kann das Ausführungsbudget negativ werden!
- Das nominelle Budget muss nicht unbedingt überschritten werden!



- oben dargestelltes Fehlerbild tritt beim PSS tatsächlich auf
 - Überläufe resultieren z.B. aus der vereinfachten Budget-Überwachung
 - Anhäufung resultiert aus Auffüllung des verbrauchten Budgets
 - das Budget wurde überzogen, also wird auch zu viel aufgefüllt
- **Lösungsansatz:** Überläufe von der nächsten Auffüllung borgen
 - den Überlauf beim nächsten Auffüllzeitpunkt verrechnen
 - \rightsquigarrow erfordert ein negatives Ausführungsbudget

Verfrühte Auffüllung des Budgets

- ein fragmentiertes Ausführungsbudget bedeutet enormen Aufwand
 - viele Scheiben und anstehende Auffüllungen müssen verwaltet werden
 - anstehende Auffüllungen speichern → **Speicheraufwand**
 - viele Auffüllungen abarbeiten → **viele Unterbrechungen**
- POSIX vereinfacht die Verwaltung von Scheiben

a replenishment operation consists of adding the corresponding replenishment_amount to the available execution capacity at the scheduled time

- für das komplette, verfügbare Budget wird derselbe Aktivierungszeitpunkt t_b des Zustellers angenommen (s. Folie 17)
- das ist insbesondere dann problematisch, wenn eine Auffüllung des Budgets erfolgt, während der Zusteller gerade tätig ist

→ Folge ist eine **verfrühte Auffüllung** von Teilen des Budgets

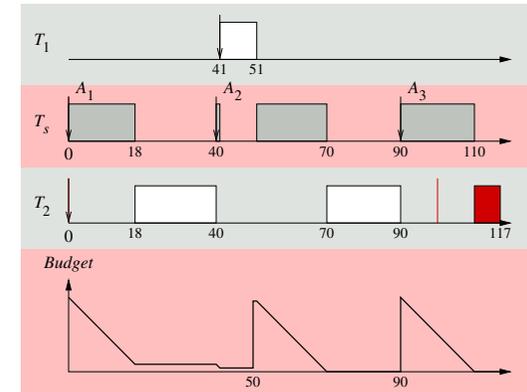
Verfrühte Auffüllung des Budgets (Forts.)

periodische Aufgaben:

- $T_1 = (200, 10, 20, 41)$
- $T_2 = (200, 49, 100, 0)$
- $T_s = (50, 20)$ (PSS)
- DM: $T_1 \succ T_s \succ T_2$

aperiodische Jobs:

- $A_1 = 18 [0, \infty[$
- $A_2 = 20 [40, \infty[$
- $A_3 = 20 [90, \infty[$



- eigentlich ist eine rechtzeitige Fertigstellung von $J_{2,1}$ möglich
 - laut Antwortzeitanalyse (s. Folie IV-2/40) gilt: $\omega_{2,1} = 99$
- Problem ist die Auffüllung des Budgets zum Zeitpunkt t_{50}
 - dieser erhält ebenso wie das Restbudget den Startzeitpunkt t_{40}
 - hier wurden zwei Scheiben unerlaubterweise vereinigt

Unzureichende zeitliche Isolation

- POSIX unterstützt folgende Planungsverfahren für terminbehaftete Arbeitsaufträge:
 - SCHED_FIFO** MLQ-Planer, stat. Prioritäten (s. Folie IV-1/20)
 - SCHED_RR** Reihum-Verfahren (engl. *round robin*)
 - SCHED_SPORADIC** $SCHED_FIFO \cap$ POSIX Sporadic Server
 - SCHED_OTHER** standardmäßiger Zeitmultiplexbetrieb
- Problem ist eine **ungünstige Verteilung** der globalen Prioritätsebenen:
 - $\geq 1 \mapsto SCHED_FIFO, SCHED_RR$ und $SCHED_SPORADIC$
 - $0 \mapsto SCHED_OTHER$
- der PSS arbeitet auch im **Hintergrundbetrieb**
 - Bei Budgeterschöpfung sinkt der PSS auf eine **Hintergrundpriorität**
 - diese wird aber gegenüber $SCHED_OTHER$ bevorzugt

→ der PSS kann Jobs in $SCHED_OTHER$ **beliebig verzögern**

Bandbreite-bewahrende Zusteller – Wrap Up

- der **POSIX Sporadic Server** ist wichtig für POSIX
 - die einzige Möglichkeit in POSIX, um Rechenzeit einzuschränken
 - eine Behebung der Fehler ist daher wünschenswert
- sporadische Zusteller** sind wichtig
 - sie stellen eine **allgemeine Rechenzeitressource** dar
 - mit $T_s = (20, 4)$ kann man z.B. 20% der Rechenzeit reservieren
 - innerhalb von T_s kann man frei über diese Rechenzeit verfügen
 - Grundlage zahlreicher Ansätze für **hierarchische Ablaufplanung**
 - sind leider **sehr, sehr komplex**
 - Achtung:** auch der SpSL Sporadic Server ist **fehlerhaft!**
 - es gibt aber diverse korrigierte Varianten, siehe z.B. [3, S. 212 ff.]
- Aufschiebbare Zusteller vs. sporadische Zusteller
 - lange Zeit galt: sporadische sind **besser** als aufschiebbare Zusteller
 - für statische Prioritäten wurde dies aber größtenteils widerlegt [1]
 - meistens sind aufschiebbare und sporadische Zusteller ebenbürtig
 - aufschiebbare Zusteller liefern oft bessere Antwortzeiten (double hit)
 - sie sind einfacher zu implementieren und erzeugen weniger Overhead

Gliederung

- 1 Überblick
- 2 Bandweite-bewahrende Zusteller
- 3 Übernahmeprüfung
- 4 Zusammenfassung

Übernahmeprüfung

Bisher: Abfertigung aperiodischer Arbeitsaufträge

- Minimierung der **durchschnittlichen Antwortzeit**
- **kontrollierbare Interferenz** mit periodischen Arbeitsaufträgen

Sporadische Arbeitsaufträge erfordern eine **Übernahmeprüfung**:

- Kann der Termin des Arbeitsauftrags eingehalten werden?
 - diese Überprüfung läuft gekoppelt zur Laufzeit ab
 - ↳ ihr **Aufwand** ist daher ein entscheidendes Kriterium

↳ Je nach Ergebnis der Übernahmeprüfung

- positiv** Zulassung und Einplanung des sporadischen Jobs
- negativ** Abweisung des Jobs und Anzeige einer Ausnahmesituation

Im folgenden: einfache Akzeptanztests für die Übernahmeprüfung

Folie 31 ff. in nach EDF geplanten Systemen

Folie 35 ff. in System mit statischen Prioritäten

Einplanung sporadischer Jobs mit EDF [3, S. 251]

Grundidee für die Einplanung sporadischer Jobs mit Hilfe von EDF:

- keine Unterscheidung periodischer und sporadischer Jobs

↳ alle Arbeitsaufträge sind sporadisch!

Akzeptanztest fußt auf dem EDF-Planbarkeitskriterium (s. Folie IV-2/33)

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

verwendet aber die **Dichte** (engl. **density**) anstelle der Auslastung

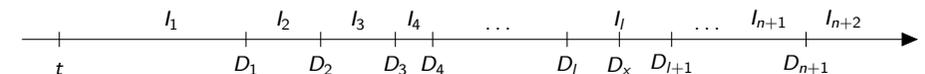
- Dichte Δ_i eines sporadischen Jobs S_i ist $e_i / (D_i - r_i)$
 - mit dem absoluten Termin D_i

$$\Delta = \sum_{i=1}^n \Delta_i = \sum_{i=1}^n \frac{e_i}{D_i - r_i} \leq 1$$

- bezieht sich auf alle derzeit aktiven, sporadischen Jobs
- ist **hinreichend**, aber **nicht notwendig** ☹

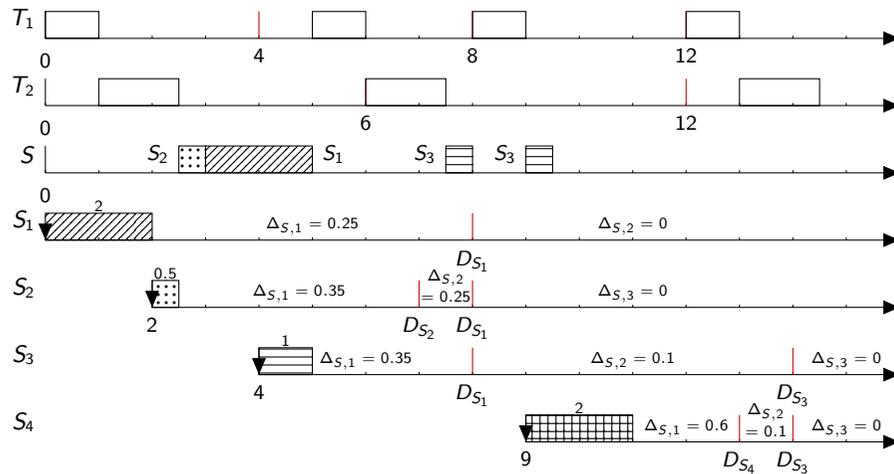
Dichte-basierter Akzeptanztest

- Dichte Δ_s der sporadischen Jobs darf $1 - \Delta$ nie überschreiten
 - Δ ist hier die durch periodische Aufgaben verursachte Dichte
- ↳ Wie bestimmt man die Dichte Δ_s der aktuellen sporadischen Jobs?



- zum Zeitpunkt t trifft der sporadische Job $S_1 = (e_1, D_1)$ ein
 - D_1 teilt die Zeitachse in Intervalle $I_1 =]t; D_1]$ und $I_2 =]D_1; \infty[$
 - die Dichte $\Delta_{s,1}$ im Intervall I_1 ist $e_1 / (D_1 - t)$ und $\Delta_{s,2} = 0$ in I_2
- Verallgemeinerung auf n sporadische Jobs und $n + 1$ Intervalle
 - in Intervall I_k ist die Dichte $\Delta_{s,k} = \sum_{j>k} \Delta_j = \sum_{j>k} e_j / (D_j - t)$
- zum Zeitpunkt t trifft nun ein Job $S_x = (e_x, D_x)$ ein ↳ **Test!**
 - der Termin D_x liege dabei im Intervall I_j
 - Zulassung ist möglich falls: $\forall k=1, \dots, j : \Delta_x + \Delta_{s,k} \leq 1 - \Delta$
 - die Gesamtdichte überschreitet also in keinem Intervall den Wert 1
 - anschließend gibt des $n + 2$ Intervalle ...

Beispiel: Dichte-basierter Akzeptanztest [3, S. 252]

 $T_1 = (4, 1)$, $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$, EDF-Ablaufplanung


Beispiel: Dichte-basierter Akzeptanztest [3, S. 252] (Forts.)

- t_0 $T_1 = (4, 1)$ und $T_2 = (6, 1.5)$ werden periodisch ausgeführt
- der relative Termin ist identisch zur jeweiligen Periode
 - $S_1 = 2(0, 8]$ trifft ein $\rightsquigarrow D_{S_1} = 8$
 $\rightsquigarrow I_1 = (0, 8] : \Delta_{S,1} = 0.25, I_2 = (8, \infty] : \Delta_{S,2} = 0$
- t_2 $S_2 = 0.5(2, 7]$ trifft ein $\rightsquigarrow D_{S_2} = 7$
- $\rightsquigarrow I_1 = (0, 7] : \Delta_{S,1} = 0.35, I_2 = (7, 8] : \Delta_{S,2} = 0.25, I_3 = (8, \infty] : \Delta_{S,3} = 0$
- $t_{2.5}$ S_2 startet und beendet sich bei t_3
- t_3 S_1 startet und beendet sich bei t_5
- t_4 $S_3 = 1(4, 14]$ trifft ein $\rightsquigarrow D_{S_3} = 14$
- S_2 hat die Ausführung bereits beendet
 - $\rightsquigarrow I_1 = (4, 8] : \Delta_{S,1} = 0.35, I_2 = (8, 14] : \Delta_{S,2} = 0.1, I_3 = (14, \infty] : \Delta_{S,3} = 0$
- $t_{7.5}$ S_3 startet und wird bei t_8 von $J_{1,3}$ unterbrochen
- t_9 S_3 wird fortgesetzt und endet bei $t_{9.5}$
- $S_4 = 2(9, 13]$ trifft ein $\rightsquigarrow D_{S_4} = 13$
 - $\rightsquigarrow I_1 = (9, 13] : \Delta_{S,1} = 0.6, I_2 = (13, 14] : \Delta_{S,2} = 0.1, I_3 = (14, \infty] : \Delta_{S,3} = 0$
 - $\rightsquigarrow \Delta_{S,1} = 0.6 \geq 1 - \Delta$: **Abweisung von S_4**

Schlupf-basierter Akzeptanztest [3, S. 258]

Sporadische Zusteller ermöglichen einfache Akzeptanztest für statische Prioritäten

Sporadischer Zusteller $T_s = (p_s, e_s)$ fertigt sporadische Jobs ab

- in T_s werden sporadische Jobs nach EDF sortiert
- \rightsquigarrow das Budget von T_s steht für sporadische Jobs zur Verfügung
 - das sind e_s Zeiteinheiten pro Auffüllperiode p_s
- \rightsquigarrow die Berechnung des Schlupfes wird hierdurch stark vereinfacht
 - ein sporadischer Job $S_1 = (e_1, D_1)$ trifft zum Zeitpunkt t ein
 - auch hier ist D_i der absolute Termin
 - T_s verfügt bis D_1 über mindestens $\lfloor (D_1 - t) / p_s \rfloor e_s$ Zeiteinheiten
 - der Schlupf $\sigma_1(t)$ von S_1 ist also $\sigma_1(t) = \lfloor (D_1 - t) / p_s \rfloor e_s - e_1$
 - um S_1 zuzulassen, muss der Schlupf $\sigma_1(t) \geq 0$ sein
- \rightsquigarrow vor S_i können bereits n sporadische Jobs zugelassen worden sein
 - diese müssen bei der Berechnung des Schlupfes berücksichtigt werden

$$\sigma_i(t) = \lfloor (D_i - t) / p_s \rfloor e_s - e_i - \sum_{D_k < D_i} (e_k - \xi_k)$$

- ξ_k beschreibt den bereits abgearbeiteten Teil von S_k
- Jobs S_k mit einem späteren Termin $D_k \geq D_i$ werden explizit geprüft

Gliederung

- 1 Überblick
- 2 Bandweite-bewahrende Zusteller
- 3 Übernahmepfprüfung
- 4 Zusammenfassung

Resümee

Bandweite bewahrende Zusteller \leadsto Verbrauchs-/Auffüllregeln

- aufschiebbar: ohne/mit Hintergrundzusteller, Planbarkeit
- sporadische: einfach; kumulativ, SpSL, POSIX

POSIX Sporadic Server: die Emanzipation des SpSL Sporadic Server

- Bedeutung innerhalb des POSIX-Standard
- Ausweitung des Budgets, verfrühte Auffüllung
- unzureichende Zeitliche Isolation

Übernahmeprüfungen für dynamische und statische Prioritäten

- dichte-basierter Akzeptanztest für die EDF-Ablaufplanung
- schlupf-basierter Akzeptanztest für sporadische Zusteller

Literaturverzeichnis

- [1] BERNAT, G. ; BURNS, A. :
New results on fixed priority aperiodic servers.
In: *Proceedings of the 20th IEEE Real-Time Systems Symposium, (RTSS '99)*.
IEEE, New York : IEEE, Dez. 1999, S. 68–78
- [2] IEEE:
1003.1d-1999 Information Technology — Portable Operating System Interface (POSIX®)
— Part 1: System Application Program Interface (API) — Amendment x: Additional
Realtime Extensions [C Language].
IEEE, New York : IEEE, 1999
- [3] LIU, J. W. S.:
Real-Time Systems.
Prentice-Hall, Inc., 2000. –
ISBN 0–13–099651–3
- [4] SPRUNT, B. ; SHA, L. ; LEHOCZKY, J. P.:
Aperiodic Task Scheduling for Hard Real-Time Systems.
In: *Real-Time Systems Journal* 1 (1989), Nr. 1, S. 27–60. –
ISSN 0922–6443

Literaturverzeichnis (Forts.)

- [5] STANOVICH, M. ; BAKER, T. ; WANG, A.-I. ; HARBOUR, M. :
Defects of the POSIX Sporadic Server and How to Correct Them.
In: *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '10)*.
IEEE, New York : IEEE, april 2010. –
ISSN 1080–1812, S. 35–45