

Echtzeitsysteme

Zustellerkonzepte & Übernahmeprüfung

Peter Ulbrich

Lehrstuhl für Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

<https://www4.cs.fau.de>

09. Dezember 2015



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.
 - **Vorgrundbetrieb** 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

- Aufschiebbarer Zusteller
- Sporadischer Zusteller
- SpSL Sporadic Server
- POSIX Sporadic Server
- Hierarchische Ablaufplanung

3 Übernahmeprüfung

- Dynamische Prioritäten
- Statische Prioritäten



Rückblick: Abfragender Zusteller

Vergleiche Folie V-1/23 ff



Verhalten einer periodischen Aufgabe im **schlimmsten Fall**

- Beschränkter, exakt quantifizierbarer Einfluss auf andere Aufgaben!
→ **Vereinfachung der** (zeitlichen) **Analyse**



Abfragende Zusteller liefern **unbefriedigende Antwortzeiten**

- Grund: **Verlust des Ausführungsbudgets** bei Untätigkeit
→ Verspätete Aufträge werden auf nächste Periode verschoben



Restbudget auch in Phasen der Untätigkeit bewahren?

- Unter Beibehaltung des periodischen



Bandweite-bewahrende Zusteller
(engl. *bandwidth-preserving servers*)



Überblick: Bandbreite-bewahrende Zusteller

Erhalt des Restbudgets untätig gewordener Zusteller in der Abfrageperiode

Bandbreite-bewahrende Zusteller

- Bewahren Budget (d.h., Bandbreite) innerhalb ihrer Abfrageperiode
→ **Verbesserung** des Abfragebetriebs und der **Antwortzeiten**
- Erweiterung des **Regelwerks**:
 - **Verbrauchsregeln** (engl. *consumption rules*)
 - Bedingungen unter denen das Budget bewahrt/verbraucht wird
 - **Auffüllregeln** (engl. *replenishment rules*)
 - Festlegungen *wann* und *wie* das Budget aufgefüllt wird
- Erweiterung des **Verarbeitungsschemas**:
 - **Planer** (Betriebssystem) führt Buch über den Budgetverbrauch
 - Suspendiert den Zusteller, wenn das Budget verbraucht wurde
 - Stellt den Zusteller bereit, wenn das Budget aufgefüllt wurde
 - **Zusteller** setzt sich selbst aus, wenn er untätig wird
 - Restbudget zum Zeitpunkt des Untätigwerdens bleibt erhalten
 - Wird ausführungsbereit, falls *zurückgestellt* (vgl. V-1/22)



Aufschiebbarer Zusteller

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



Aufschiebbarer Zusteller (engl. *Deferrable Server*) $\mapsto T_D = (p_D, e_D)$

- Periodisches Auffüllen von Budget e_D mit Periode p_D (vgl. V-1/23)
- Bewahrung des (Rest-)Budgets von T_D in p_D bei Untätigkeit

Aufschiebbarer Zusteller

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 $k = 0, 1, 2, \dots$



Keine Akkumulation des Restbudgets von Periode zu Periode

→ Restbudget verfällt ggf. am Ende der Abfrageperiode

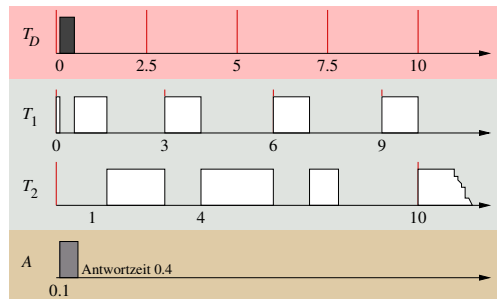


Beispiel: Aufschiebbarer Zusteller (1)

Optimiertes Antwortverhalten im Vergleich zu abfragendem Zusteller (V-1/25)

Aufgabensystem:

- **periodische Tasks**
 - $T_D = (2.5, 0.5)$
 - $T_1 = (3, 1)$
 - $T_2 = (10, 4)$
 - RM
- **aperiodischer Job**
 - $A \mapsto (0.4, [0.1, \infty))$



Budget von 0.5 Zeiteinheiten bleibt T_D erhalten

- Obwohl T_D zum Zeitpunkt t_0 untätig ist
- Unmittelbare Abarbeitung von A zum Zeitpunkt $t_{0.1}$



Keine Übertragung des Restbudgets

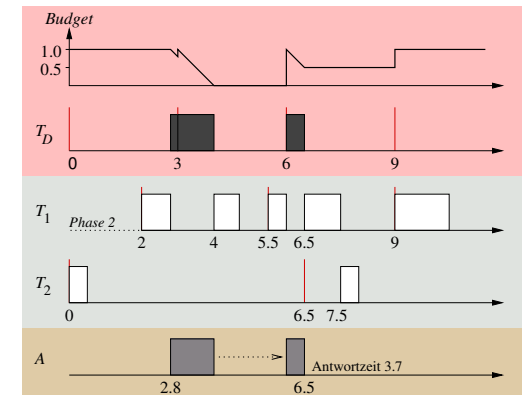
- Budget von 0,1 Zeiteinheiten verfällt mit $t_{2.5}$



Beispiel: Aufschiebbarer Zusteller (2)

Aufgabensystem:

- **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_4 T_D wird vom Planer gestoppt
- t_6 T_D kommt erneut in Frage
- $t_{6.5}$ A ist beendet, T_D untätig



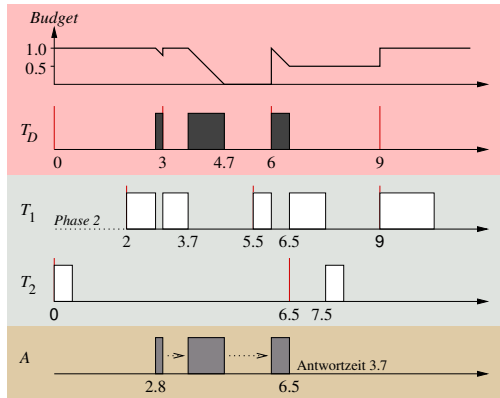
Beispiel: Aufschiebbarer Zusteller (3)

Alternatives Einplanungsverfahren mittels EDF

- Aufgabensystem:
 - periodische Tasks
 - $T_D = (3, 1)$
 - $T_1 = (3.5, 1.5, 3.5, 2)$
 - $T_2 = (6.5, 0.5)$
 - 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)
- $t_{3.7}$ T_D wird weiter ausgeführt
- t_6 $d_1 = d_D$, T_D wird bevorzugt
- $t_{6.5}$ A ist beendet, T_D untätig



Aufschiebbarer Zusteller ∪ Hintergrundzusteller

Budgetverbrauch und -auffüllung — Antwortzeitverbesserung



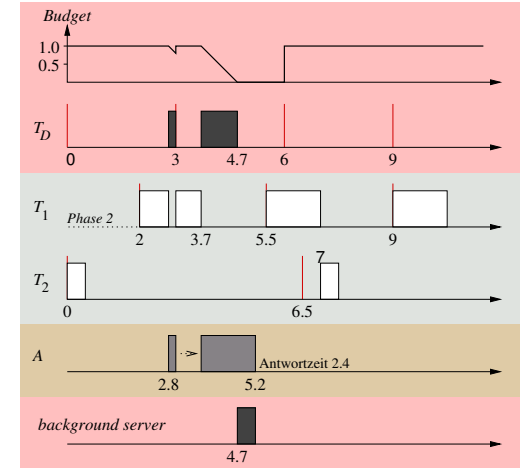
Hintergrundzusteller

- E.: *background server*
- Verarbeitet die AJQ
- Unterstützt T_D
- Niedrigste Priorität



Verlauf:

- Beispiel von Folie 9
- $t_{4.7}$ Keine periodischen Aufträge ~ Hintergrundbetrieb
- $t_{5.2}$ A ist beendet, T_D bleibt untätig



Bessere Antwortzeit, geringeres Zittern (T_1, A), weniger Kontextwechsel



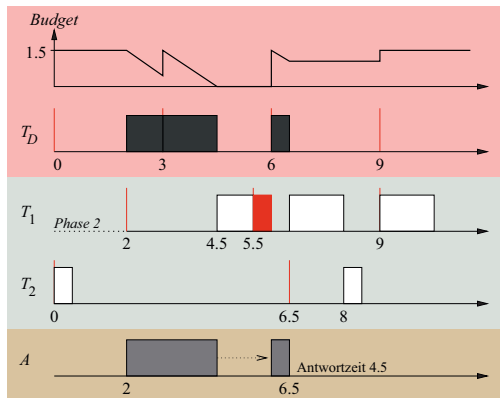
Aufschiebbarer Zusteller — Größenbeschränkung

Einfluss auf die Planbarkeit periodischer Aufgaben

- Aufgabensystem:
 - periodische Tasks
 - $T_D = (3, 1.5)$
 - $T_1 = (3.5, 1.5, 3.5, 2)$
 - $T_2 = (6.5, 0.5)$
 - RM
 - aperiodischer Job
 - $A \mapsto (3, [2, \infty))$

Verlauf:

- $t_{5.5}$ T_1 verpasst Termin
- $t_{5.2}$ A ist beendet, T_D bleibt untätig



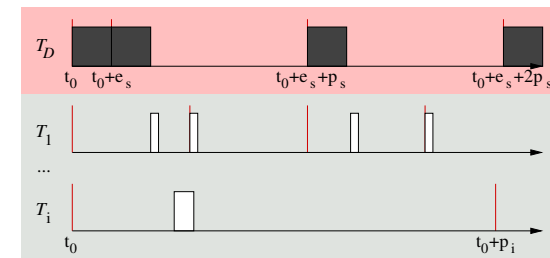
Antwortzeitverbesserung durch Vergrößerung des Budgets (anstatt Hintergrundzusteller) ggf. **problematisch**

→ Budget ist unter Berücksichtigung aller möglichen Kombinationen von Auslösezeiten aller (periodischen) Tasks zu bestimmen



Aufschiebbarer Zusteller != periodische Aufgabe

Das Problem des Doppeltreffers



Beispiel: Antwortzeitanalyse für statische Prioritäten (vgl. IV-2/32)

- $T_D \mapsto$ höchste Priorität aller periodischen Aufgaben
- Kritischer Zeitpunkt (vgl. IV-2/36) von T_i tritt zum Zeitpunkt t_0 ein:
 - Arbeitsauftrag $J_{i,j}$ dieser Aufgabe wird ausgelöst
 - Aufträge aller Aufgaben höherer Priorität T_1, \dots, T_{i-1} werden ausgelöst
 - Budget von T_D ist e_s und T_D ist zurückgestellt
 - Nächste Auffüllzeitpunkt von T_D ist $t_0 + e_s$



Doppeltreffer (engl. *double hit*) ~ Kein periodisches Verhalten



Aufschiebbarer Zusteller != periodische Aufgabe (Forts.)

Negative Auswirkungen auf die Berechnung der Antwortzeit

- ⚠ Erweiterte, iterative Bestimmung der Antwortzeit (vgl. IV-2/33)

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

- Durch den aufschiebbaren Zusteller T_D verursachte **Störung**

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

→ Störung bis zu e_D Zeiteinheiten größer als bei einer periodischen Aufgabe mit identischen Parametern

- 🔊 Sporadischer Zusteller (engl. *sporadic server*)



Sporadischer Zusteller



Sporadischer Zusteller (engl. *sporadic server*) $\mapsto T_S = (p_S, e_S)$

- Bedarf entspricht in jedem Zeitintervall dem einer periodischen Aufgabe $T = (p_S, e_S)$ mit gleichen Parametern
→ Planbarkeitsanalyse reduziert sich erneut auf periodische Aufgaben

- Verschiedene Ausführungen mit unterschiedlichen Verbrauchs- und Auffüllregeln:

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

- Prioritätsebenen ($P_1 > P_2 > \dots > P_n$):

P_i (Beliebige) Prioritätsebene einer Aufgabe T_i

P_{cur} Aktuelle Prioritätsebene des Systems

P_s Prioritätsebene des Zustellers T_s

- Tätigkeitsintervalle:

tätig $P_i: P_{cur} \geq P_i$

- Systempriorität entspricht mindestens der Prioritätsebene P_i

untätig $P_i: P_{cur} < P_i$

- Systempriorität ist niedriger als die Prioritätsebene P_i

- Auffüllzeitpunkt rt_i 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 $1/\text{Zeiteinheit}$.

Auffüllregeln

R1 Initial 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

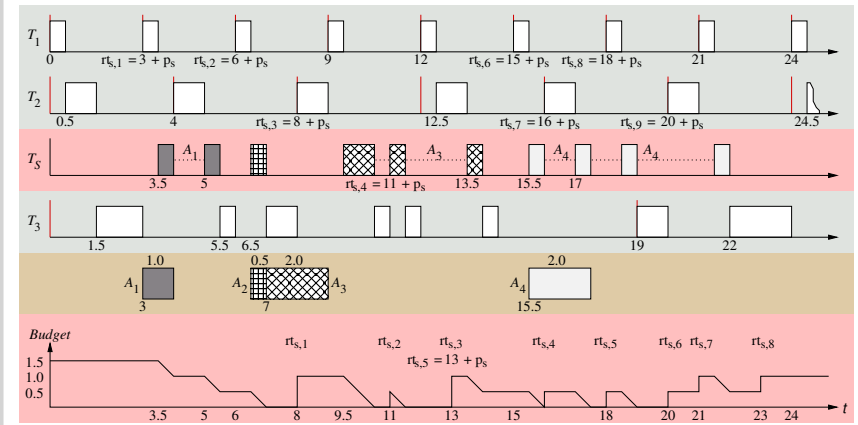
- Wenn P_s untätig wird oder T_s sein Budget erschöpft
- So viel Budget wird aufgefüllt, wie T_s seit t_b verbraucht hat



- 1 **Überwache** Tätigkeit/Untätigkeit der Prioritätsebenen P_i
 - Bestimme den nächsten Auffüllzeitpunkt rt_s von T_s
- 2 **Überwache** und protokolliere den **Verbrauch** des Budgets von T_s
 - Suspendiere T_s , falls Budget erschöpft
 - Stelle T_s bereit, falls Budget aufgefüllt wird
 - Protokollierung liefert den wieder aufzufüllenden Betrag des Budgets
- 3 **Verwalte** anstehende **Auffüllungen** (engl. *pending replenishments*)
 - T_s muss sein Budget nicht auf einmal komplett aufbrauchen
 - Jede Etappe wird einzeln aufgefüllt (vgl. Folie 16, Regel R3)
 - Das Budget von T_s wird in **Scheiben** (engl. *chunks*) zerschnitten
 - Planer muss eine Liste anstehender Auffüllungen verwalten

SpSL Sporadic Server

SpSL Sporadic Server \approx Menge periodischer Aufgaben $\{T_k\}$, hierbei gilt: Periode $p_k = p_s$, Ausführungszeit $\sum_k e_k = e_s$



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



- $t_{3.5}$ T_s startet: $t_b = 3 \rightsquigarrow rt_{s,1} = 8$ (R2)
 - T_1 startet zum Zeitpunkt $t_3 \rightsquigarrow P_s$ wurde tätig
- $t_{5.5}$ T_s wird untätig, an $rt_{s,1}$ wird 1 Zeiteinheit aufgefüllt (R3)
- $t_{6.5}$ T_s startet: $t_b = 6 \rightsquigarrow rt_{s,2} = 11$ (R2)
 - T_1 startet zum Zeitpunkt $t_6 \rightsquigarrow P_s$ 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 $\rightsquigarrow T_s$ wird noch nicht ausgeführt
- $t_{9.5}$ T_s startet: $t_b = 8 \rightsquigarrow rt_{s,3} = 13$ (R2)
 - T_2 startet zum Zeitpunkt $t_8 \rightsquigarrow P_s$ 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: $t_b = 11 \rightsquigarrow rt_{s,4} = 16$ (R2)
 - T_1 und T_2 nicht ausführungsbereit $\rightsquigarrow T_s$ startet
 - $\rightsquigarrow T_s$ wird zum Zeitpunkt t_{11} tätig
- $t_{11.5}$ **Budget erschöpft**, an $rt_{s,4}$ werden 0.5 Einheiten aufgefüllt (R3)



- $rt_{s,3} = t_{13}$ **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

SpSL Sporadic Server in der realen Welt

In Anlehnung an den SpSL Sporadic Server wurde im Standard POSIX 1003.1d [2] der **POSIX Sporadic Server (PSS)** spezifiziert
→ Einplanungsvariante **SCHED_SPORADIC**

■ Bekannte Echtzeitbetriebssysteme implementieren den PSS

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

⚠ Dummerweise ist der PSS-Algorithmus **fehlerhaft** [5]

→ PSS verhält sich nicht immer wie eine periodische Aufgabe

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



Anhäufung des Ausführungsbudgets



Exakte Überwachung des Ausführungsbudgets ist sehr aufwendig

- 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*

- Effiziente Implementierung auf Kosten der Überwachungsgenauigkeit
→ **Kleine Überläufe** werden in Kauf genommen
- Weitere irrtümlichen Annahmen:

... reaches the limit imposed on its execution time ... the execution time consumed is subtracted from the available execution capacity. 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.

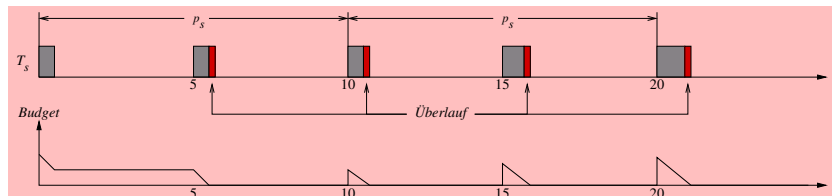


Anhäufung des Ausführungsbudgets



Konsequenz ist **Ausweitung des Ausführungsbudgets**

- Bei Überläufen kann das Ausführungsbudget negativ werden
- Das nominelle Budget muss nicht unbedingt überschritten werden



■ Realistisches Fehlerbild des PSS

- Überläufe z.B. aus der vereinfachten Budget-Überwachung
- Anhäufung resultiert aus Auffüllung des verbrauchten Budgets
 - Budget wurde überzogen, also wird auch zu viel aufgefüllt



Lösungsansatz: Überläufe von der nächsten Auffüllung borgen

- Überlauf beim nächsten Auffüllzeitpunkt verrechnen
→ Erfordert ein negatives Ausführungsbudget



Verfrühte Auffüllung des Budgets



Fragmentiertes Ausführungsbudget bedeutet **enormen Aufwand**

→ Verwaltung vieler Scheiben und anstehender Auffüllzeitpunkte

- 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 gesamte Budget wird derselbe Aktivierungszeitpunkt t_b des Zustellers angenommen (vgl. 16)



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



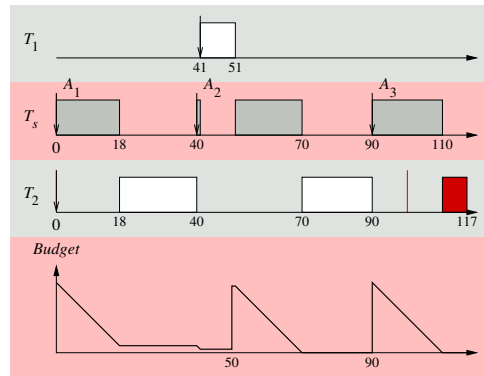
Verfrühte Auffüllung des Budgets (Forts.)

■ Periodische Aufgaben:

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

■ Aperiodische Aufgaben:

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



■ Eigentlich ist eine rechtzeitige Fertigstellung von $J_{2,1}$ möglich

- Antwortzeitanalyse (vgl. IV-2/32) liefert: $\omega_{2,1} = 99$

⚠ Problem ist die Auffüllung des Budgets zum Zeitpunkt t_{90}

- Dieser erhält ebenso wie das Restbudget den Startzeitpunkt t_{40}
- Ungültige Vereinigung zweier Scheiben



Unzureichende zeitliche Isolation

☞ POSIX unterstützt folgende Planungsverfahren für terminbehaftete Arbeitsaufträge:

- **SCHED_FIFO** MLQ-Planer, stat. Prioritäten (vgl. IV-1/33)
- **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:

- ≥ 1 → SCHED_FIFO, SCHED_RR und SCHED_SPORADIC
- 0 → SCHED_OTHER

■ Der PSS arbeitet auch im **Hintergrundbetrieb**

- Bei Budgeterschöpfung sinkt der PSS auf eine **Hintergrundpriorität**
- Diese wird aber gegenüber SCHED_OTHER bevorzugt
- PSS kann Jobs in SCHED_OTHER **beliebig verzögern**



Hierarchische Ablaufplanung

■ Wiederholung: Ablaufplanungsverfahren (vgl. IV-1/28)

- Einplanungsalgorithmen: RM, DM, EDF, ...
- Statische oder dynamische Prioritäten auf Task bzw. Job-Ebene

■ Zusteller stellen eine **allgemeine Rechenzeitressource** dar

- Durch $T_s = (20, 4)$ werden z.B. 20% Rechenzeit **reserviert**
- Innerhalb des Zustellers freie Verfügung über Rechenzeit

☞ Ermöglicht **Hierarchische Ablaufplanung** (engl. *hierarchical scheduling*)

- Beispielsweise global RM und EDF innerhalb des Zustellers
- Implementierung des Planers im Nutzerland (engl. *user land*)
- **Auswahl des jeweils optimalen Verfahrens**

⚠ Planbarkeitsanalyse wird ebenfalls hierarchisch

- Antwortzeit hängt ggf. von mehreren Ebenen ab
- Insgesamt steigen die Gemeinkosten an



Bandbreite-bewahrende Zusteller – Wrap Up

■ **Sporadische Zusteller** sind wichtig

- Stellen eine **allgemeine Rechenzeitressource** dar
- $T_s = (20, 4)$ werden z.B. 20% Rechenzeit **reserviert**
 - Freie Verfügung über diese Rechenzeit
 - Grundlage zahlreicher Ansätze für **hierarchische Ablaufplanung**
- Implementierung ist **sehr, sehr komplex**
 - **Achtung:** Auch der SpSL Sporadic Server ist **fehlerhaft!**
 - Es existieren 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

■ **POSIX Sporadic Server** ist wichtig für POSIX

- Einzige Möglichkeit in POSIX, um Rechenzeit einzuschränken
- Behebung der Fehler ist daher wünschenswert



Gliederung

- 1 Überblick
- 2 Bandweite-bewahrende Zusteller
 - Aufschiebbarer Zusteller
 - Sporadischer Zusteller
 - SpSL Sporadic Server
 - POSIX Sporadic Server
 - Hierarchische Ablaufplanung
- 3 Übernahmeprüfung
 - Dynamische Prioritäten
 - Statische Prioritäten



Übernahmeprüfung

Für nicht-periodische Aufgaben mit harten Terminen (vgl. V-1/9)

- ☞ Bisher: Abfertigung aperiodischer Arbeitsaufträge
 - Minimierung der **durchschnittlichen Antwortzeit**
 - **Kontrollierbare Interferenz** mit periodischen Arbeitsaufträgen
- ⚠ **Übernahmeprüfung** für sporadische Arbeitsaufträge erforderlich
 - Kann der Termin des Arbeitsauftrags eingehalten werden?
 - Überprüfung läuft gekoppelt zur Laufzeit ab
 - **Aufwand** ist daher ein entscheidendes Kriterium
- Je nach Ergebnis der Übernahmeprüfung
 - positiv** Zulassung und Einplanung des sporadischen Jobs
 - negativ** Abweisung und Anzeige einer Ausnahmesituation
- ☞ Im Folgenden: Akzeptanztests für die Übernahmeprüfung
 - Folie 31 ff.** in Systemen mit dynamischen Prioritäten
 - Folie 35 ff.** in Systemen mit statischen Prioritäten



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

- ☞ Einplanung sporadischer Aufträge mittels EDF (vgl. IV-2/13)
 - Idee: Keine Unterscheidung periodischer und sporadischer Jobs
 - Alle Arbeitsaufträge sind sporadisch

- Akzeptanztest basiert auf dem EDF-Planbarkeitskriterium (vgl. IV-2/28)

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

- ⚠ Betrachtung der **Dichte** (engl. *density*) anstatt der Auslastung
 - Dichte Δ_i eines sporadischen Auftrags S_i ist $e_i / (D_i - r_i)$

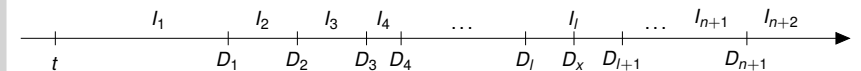
$$\Delta = \sum_{i=1}^n \Delta_i = \sum_{i=1}^n \frac{e_i}{D_i - r_i} \leq 1$$

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



Dichte-basierter Akzeptanztest

- ☞ Bestimmung der Dichte Δ_s der sporadischen Aufträge
 - Δ ist die durch periodische Aufgaben verursacht (Grund-)Dichte
 - Δ_s darf $1 - \Delta$ nie überschreiten

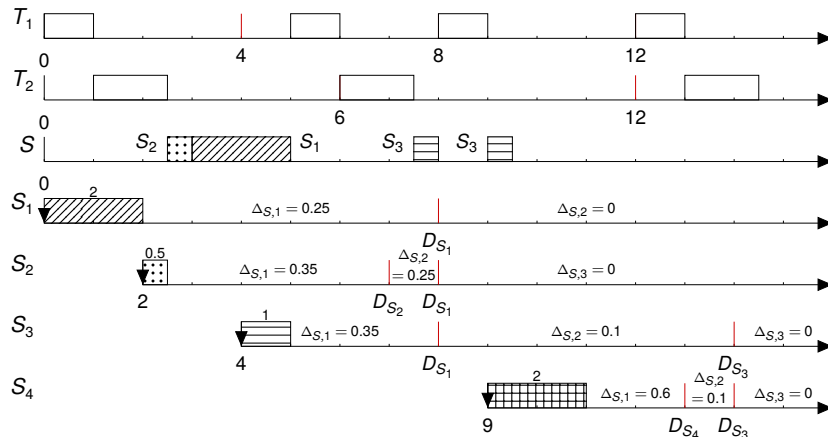


- Beispiel: Bei t trifft der sporadische Auftrag $S_1 = (e_1, D_1)$ ein
 - D_1 teilt die Zeitachse in Intervalle $I_1 = [t; D_1]$ und $I_2 = [D_1; \infty]$
 - Dichte $\Delta_{s,1}$ im Intervall I_1 ist $e_1 / (D_1 - t)$ und $\Delta_{s,2} = 0$ in I_2
 - Verallgemeinerung auf n sporadische Jobs und $n + 1$ Intervalle
 - In Intervall I_k ist die Dichte $\Delta_{s,k} = \sum_{j \geq k} \Delta_j = \sum_{j \geq k} e_j / (D_j - t)$
- Zum Zeitpunkt t trifft nun ein Auftrag $S_x = (e_x, D_x)$ ein \rightsquigarrow **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



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

- $S_1 = 2(0, 8]$ trifft ein $\rightsquigarrow D_{S_1} = 8$
- $I_1 = (0, 8] : \Delta_{S,1} = 0.25, I_2 = (8, \infty] : \Delta_{S,2} = 0$

t_2 $S_2 = 0.5(2, 7]$ trifft ein $\rightsquigarrow D_{S_2} = 7$

- $I_1 = (0, 7] : \Delta_{S,1} = 0.35, I_2 = (7, 8] : \Delta_{S,2} = 0.25, I_3 = (8, \infty] : \Delta_{S,3} = 0$

$t_{2.5}$ 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 $\rightsquigarrow I_2(\Delta_{S,2})$ kann entfallen
- $I_1 = (4, 8] : \Delta_{S,1} = 0.35, I_2 = (8, 14] : \Delta_{S,2} = 0.1, I_3 = (14, \infty] : \Delta_{S,3} = 0$

$t_{7.5}$ 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 $\rightsquigarrow D_{S_4} = 13$
- $I_1 = (9, 13] : \Delta_{S,1} = 0.6, I_2 = (13, 14] : \Delta_{S,2} = 0.1, I_3 = (14, \infty] : \Delta_{S,3} = 0$
- $\Delta_{S,1} = 0.6 \geq 1 - \Delta$: **Abweisung von S_4**



Schlupf-basierter Akzeptanztest [3, S. 258]

Sporadische Zusteller ermöglichen einfache Akzeptanztest für statische Prioritäten

- Abfertigung sporadischer Aufträge durch Sporadischer Zusteller
 - Innerhalb von $T_s = (p_s, e_s)$ Einplanung nach EDF

☞ Berechnung des Schlupfs wird hierdurch stark vereinfacht

- Beispiel: Auftrag $S_1 = (e_1, d_1)$ trifft zum Zeitpunkt t ein

- T_s verfügt bis d_1 über mindestens $\lfloor (d_1 - t) / p_s \rfloor e_s$ Budget
- Schlupf $\sigma_1(t)$ von S_1 ist also $\sigma_1(t) = \lfloor (D_1 - t) / p_s \rfloor e_s - e_1$
- Zulassung von S_1 erfordert Schlupf $\sigma_1(t) \geq 0$

⚠ Vor S_i können n sporadische Aufträge zugelassen worden sein
→ Berücksichtigung bei der Berechnung des Schlupfs

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

- $c_k(t)$ beschreibt den bereits abgearbeiteten Teil von S_k
- Aufträge S_k mit späterem Termin $D_k \geq D_i$ werden explizit geprüft



Resümee

- **Bandbreite bewahrende Zusteller** \rightsquigarrow Verbrauchs-/Auffüllregeln
 - Aufschiebbar: ohne/mit Hintergrundzusteller, **Doppeltreffer**
 - Sporadisch: **SpSL Sporadic Server**, **Komplexität**
- **POSIX Sporadic Server**: Umsetzung 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



- [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.
Englewood Cliffs, NJ, USA : Prentice Hall PTR, 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



- [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

