

Echtzeitsysteme

Grundlegende Abfertigung nicht-periodischer Echtzeitsysteme

Lehrstuhl Informatik 4

28. November 2013

Fragestellungen

- Wo unterscheiden sich periodische und **nicht-periodische Aufgaben**?
 - Wo spielen nicht-periodische Aufgaben eine Rolle?
 - Welche Herausforderungen ergeben sich für ihre Abfertigung?
- Welche **Basistechniken** stehen für die Abarbeitung zur Verfügung?
 - Sind diese Techniken auf Anwendungsebene umsetzbar?
 - Benötigt man spezielle Unterstützung des Betriebssystems?
 - Welche **Risiken**, **Vorteile** und **Nachteile** beinhalten diese Techniken?
- **Schlupfzeit** – Kann man sie für nicht-periodische Jobs verwenden?
 - Wie bestimmt man die Schlupfzeit?

Gliederung

- 1 Überblick
- 2 **Nicht-periodische Aufgaben**
 - Zeitparameter
 - Nicht-periodische Echtzeitanwendungen
 - Herausforderung: Mischbetrieb
- 3 **Basistechniken**
 - Periodischer Zusteller
 - Unterbrecherbetrieb
 - Hintergrundbetrieb
- 4 **Slack-Stealing**
 - Taktgesteuerte Systeme
 - Vorranggesteuerte Systeme
- 5 **Zusammenfassung**

Gliederung

- 1 Überblick
- 2 **Nicht-periodische Aufgaben**
 - Zeitparameter
 - Nicht-periodische Echtzeitanwendungen
 - Herausforderung: Mischbetrieb
- 3 **Basistechniken**
 - Periodischer Zusteller
 - Unterbrecherbetrieb
 - Hintergrundbetrieb
- 4 **Slack-Stealing**
 - Taktgesteuerte Systeme
 - Vorranggesteuerte Systeme
- 5 **Zusammenfassung**

Nicht-periodische Aufgabe (engl. *non-periodic task*)

Bieten im Vergleich zu periodischen Aufgaben deutlich weniger verwertbares Vorwissen

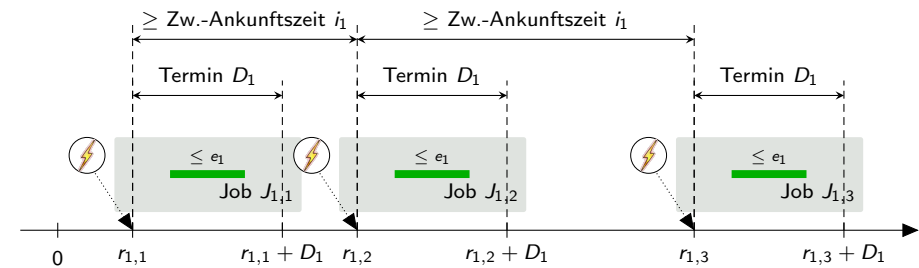
Aufgaben, die in unregelmäßigen Zeitintervallen eine vorgegebene Systemfunktion erbringen. Jede nicht-periodische Aufgabe $T_i = (i_i, e_i, D_i)$ ist eine Abfolge von Arbeitsaufträgen mit

- i_i der minimalen Zwischenankunftszeit
(engl. *minimum interarrival-time*)
- e_i der maximalen Ausführungszeit (WCET)
- D_i dem relativen Termin

Aperiodische Aufgaben (engl. *aperiodic tasks*) besitzen **weiche** oder **feste Termine**, während sporadische Aufgaben (engl. *sporadic tasks*) **harte Termine** besitzen.

- ☛ Für nicht-periodische Aufgaben steht deutlich **weniger Wissen a-priori** zur Verfügung als für periodische Aufgaben!
 - Aussagen zum **Zeitpunkt ihrer Auslösung** sind kaum möglich!

Nicht-periodische Aufgaben auf der Echtzeitachse



Ausführungszeit e_i : maximale Ausführungszeit aller Jobs $J_{i,j}$ in T_i

relativer Termin D_i : maximale Zeitspanne zwischen Auslösezeit $r_{i,j}$ und Fertigstellung eines Jobs $J_{i,j}$ in T_i

Zwischenankunftszeit i_i : minimale Länge aller Zeitintervalle $[r_{i,j}; r_{i,j+1}]$ zwischen den Auslösezeiten der Jobs in T_i

Nicht-periodische Aufgaben in der Praxis

Nicht-periodische Aufgaben behandeln entsprechende Ereignisse, die sich aus Zustandsänderungen ableiten (vgl. *event trigger*, Folie III-2/8). Mögliche Quellen für solche Zustandsänderungen sind unter anderem:

- Mensch-Maschine-Interaktion
 - menschliches Verhalten ist kaum quantifizierbar
- Fehlerbehandlung
 - sowohl innerhalb der Echtzeitanwendung, als auch im gesamten Echtzeitsystem (z.B. durch Verschleiß bedingte Ausfälle)
- Umsetzung von Steuerkommandos
 - Empfang über die Fernbedienung
~ **sporadische Aufgabe**
- Statusinformation per WLAN senden
 - falls ein interner Puffer voll läuft
~ **aperiodische Aufgabe**



Restriktionen des nicht-periodischen Modells

Lockerung der Restriktion A1 (s. Folie IV-1/10)

Mathematische Ansätze zur Analyse periodischer Echtzeitsysteme schränken solche Systeme häufig stark ein:

- ~~A1 Alle Aufgaben sind periodisch.~~
- A2 Alle Arbeitsaufträge können an ihren Auslösezeitpunkten eingeplant und ausgeführt werden.
- A3 Termine und Perioden sind identisch.
- A4 Kein Arbeitsauftrag gibt die Kontrolle über den Prozessor ab.
- A5 Alle Aufgaben sind unabhängig voneinander, d.h. die einzige gemeinsame Ressource ist die CPU und es existieren keine Einschränkungen hinsichtlich der Auslösezeiten der Arbeitsaufträge.
- A6 Der Overhead durch Unterbrechungen, Ablaufplanung oder Verdrängung ist vernachlässigbar.
- A7 Alle Aufgaben verhalten sich voll-präemptiv.

Mischbetrieb: periodisch $\leftrightarrow$ sporadisch/apperiodisch

Gekoppelte Einplanung nicht-periodischer Arbeitsaufträge

Koexistenz periodischer und nicht-periodischer Arbeitsaufträge

- **Erhaltung statischer Garantien** für periodische Arbeitsaufträge
 - Behinderung periodischer Arbeitsaufträge durch nicht-periodische Jobs muss begrenzt sein!
- dynamische Einplanung nicht-periodischer Arbeitsaufträge

Sporadische Arbeitsaufträge $\leadsto$ Übernahmeprüfung

- Entscheidung über **Zulassung** oder **Abweisung** des Arbeitsauftrags:
 - 1 Einplanung eines zugelassenen sporadischen Arbeitsauftrags, so dass sein Abschluss zum vorgegebenen Termin sichergestellt ist
 - 2 Zusicherung der Termineinhaltung aller periodischen Aufgaben und anderen zuvor bereits zugelassenen sporadischen Arbeitsaufträge

Aperiodische Arbeitsaufträge $\leadsto$ Antwortzeitminimierung

- bei Termineinhaltung aller periodischen Aufgaben sowie der bereits zugelassenen (und eingeplanten) sporadischen Arbeitsaufträge

Sporadische Arbeitsaufträge

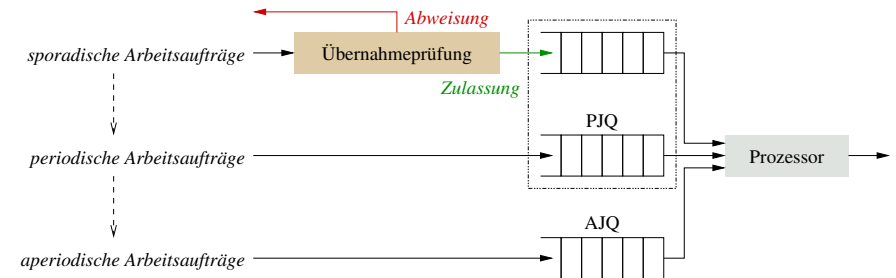
Zufällig ausgelöste Jobs mit harten Terminen

Durchführung einer **Übernahmeprüfung** (engl. *acceptance test*) für einen sporadischen Job wenn dieser (ereignisbedingt) ausgelöst wird

- der ausgelöste Job wird angenommen, wenn seine Ausführung zusammen mit allen anderen Jobs des Systems machbar ist
 - der gegenwärtige Ablaufplan muss **genügend viel Schlupf** aufweisen
 - mind. soviel, wie die max. Ausführungszeit des sporadischen Jobs
 - die Ausführungszeit wird ggf. erst zum Auslösezeitpunkt bekannt
 - nur Schlupf **vor dem Termin** des sporadischen Jobs ist von Relevanz
 - alle Rahmen, die mit dem Termin erfasst werden, finden Beachtung
 - der Test ist **gekoppelt mit der Jobeinlastung**, er läuft **online** ab
- scheitert der Test, so wird der sporadische Job abgewiesen
 - der Anwendung eine **schwerwiegende Ausnahmesituation** anzeigen
 - für die Ausnahmebehandlung wird soviel Zeit wie möglich freigestellt

„gleichzeitige“ sporadische Jobs werden oft nach EDF getestet

Prioritätswarteschlangen im Betriebssystem



- bereitgestellte periodische Arbeitsaufträge $\leadsto$ **Periodic Job Queue**
 - diese kann auch durch eine statische Ablaufabelle implementiert sein
- sporadische Arbeitsaufträge durchlaufen ein/zwei Warteschlangen:
 - 1 ausgelöste Arbeitsaufträge müssen ggf. auf Übernahmeprüfung warten
 - 2 zugelassene Arbeitsaufträge kommen in eine eigene oder die PJQ
- bereitgestellte aperiodische Arbeitsaufträge $\leadsto$ **Aperiodic Job Queue**

Gliederung

- 1 Überblick
- 2 Nicht-periodische Aufgaben
 - Zeitparameter
 - Nicht-periodische Echtzeitanwendungen
 - Herausforderung: Mischbetrieb
- 3 Basistechniken
 - Periodischer Zusteller
 - Unterbrecherbetrieb
 - Hintergrundbetrieb
- 4 Slack-Stealing
 - Taktgesteuerte Systeme
 - Vorranggesteuerte Systeme
- 5 Zusammenfassung

Überblick

Grundlegende, mit minimaler Unterstützung des Laufzeitsystems umsetzbare Behandlungsmethoden für nicht-periodische Ereignisse. Sie sind sowohl für takt- als auch für vorranggesteuerte Systeme geeignet und teilweise vollständig auf Anwendungsebene umsetzbar.

Periodischer Zusteller $\leadsto$ **Alles ist eine periodische Aufgabe!**

- Abfragen nicht-periodische Ereignisse durch periodische Aufgaben
- $\leadsto$ kommt meist ohne gesonderte Unterstützung aus

Unterbrecherbetrieb $\leadsto$ Abfertigung im **Vordergrund**

- Ereignisbehandlung direkt in der Unterbrechungsbehandlung
- $\leadsto$ Unterstützung von **Ausnahmebehandlungen** (s. Folie III-1/18 ff.)

Hintergrundbetrieb $\leadsto$ **Periodische Aufgaben haben Vorfahrt!**

- Phasen der Untätigkeit für nicht-periodische Aufgaben nutzen
- $\leadsto$ benötigt **Verdrängung** (s. Folie III-2/15 ff.)

Periodischer Zusteller (Forts.)

Arbeitsweise

Auslösung, Bereitstellung und Ausführung — der Zusteller...

- wird „zurückgestellt“ (engl. *backlogged*) wenn:
 - mindestens ein nicht-periodischer Job ausführungsbereit ist
 - bei leerer Warteschlange ist der Zusteller untätig (engl. *idle*)
 - der erste nicht-periodische Arbeitsauftrag ausgelöst wird
- kommt in Frage (engl. *is eligible*), für die Ausführung wenn er:
 - einen **Auftragsüberhang** (engl. *backlog*) aufweist *und*
 - über ein **Ausführungsbudget** verfügt
- nimmt teil am **Einplanungsverfahren** periodischer Aufgaben
 - als „normale“ periodische Aufgabe mit $T_s = (p_s, e_s)$
- verbraucht (engl. *consumes*) sein **Budget** während der Ausführung
 - bis es auf Null abgesunken d.h. aufgebraucht (engl. *exhausted*) ist

Periodischer Zusteller (engl. *periodic server*)

Periodische Abarbeitung aperiodischer Arbeitsaufträge

Spezialisierung einer **periodischen Aufgabe** $T_s = (p_s, e_s)$ (s. Folie IV-1/5)

- definiert durch **Periode** p_s und **Ausführungszeit** e_s
 - das Verhältnis $u_s = e_s/p_s$ ist die Größe (engl. *size*) des Zustellers
- mit e_s als sogenanntes **Ausführungsbudget** (engl. *execution budget*)
 - das Budget wird um bis zu e_s Einheiten aufgefüllt (engl. *replenished*)
- und der **Auffüllperiode** p_s (engl. *replenishment period*)
 - das Budget des Zustellers wird regelmäßig erneuert
 - der Zeitpunkt wird **Auffüllzeit** (engl. *replenishment time*) genannt
- innerhalb eines beliebigen Zeitintervalls der Länge p_s niemals länger als e_s Zeiten (sporadische/aperiodische Arbeitsaufträge) ausführend
- verschiedenartig ausgelegt: abfragend, aufschiebbar, sporadisch



Ein **periodischer Zusteller** ist i.d.R. für die Ausführung der Jobs **mehrerer sporadischer/aperiodischer Aufgaben** zuständig!

Abfragender Zusteller (engl. *polling server*)

Verfall des Restbudgets zum Zeitpunkt des Untätigwerdens

Abfrager (engl. *poller*) $\mapsto T_P = (p_s, e_s)$

- mit **Abfrageperiode** p_s (engl. *polling period*)
 - zyklisch bereitgestellt im Abstand von (ganzzahlig vielfachen) p_s
- schrittweise Abfertigung von Jobs innerhalb einer Abfrageperiode
 - Abarbeitung des Auftragsüberhangs nicht-periodischer Jobs
 - ggf. Unterbrechung des laufenden Arbeitsauftrags am Periodenende
- ohne Auftragsüberhang **verfällt das Budget unverzüglich**
 - d.h., sobald der Abfrager feststellt, untätig sein zu müssen
 - auch, wenn dies bereits am Anfang der Abfrageperiode erkannt wird¹
- Antwortzeiten aperiodischer Arbeitsaufträge schwanken ggf. stark
 - je nach Auslösezeitpunkt eines Auftrags bzw. Zustand des Abfragers

¹Eintreffende aperiodische Arbeitsaufträge nachdem der Abfrager seine Untätigkeit festgestellt hat, kommen frühestens in der nächsten Abfrageperiode zum Zuge.

Beispiel: Abfragender Zusteller

Grundlegende Funktionsweise abfragender Zusteller

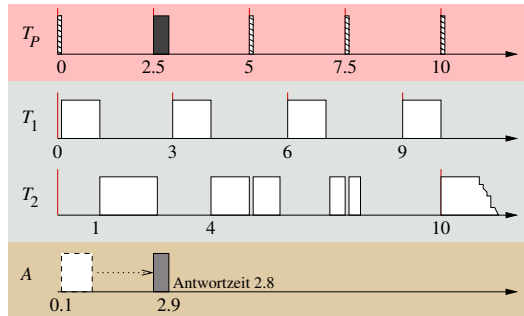
Beispiel:

- periodische Tasks

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

- aperiodischer Job

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



- T_P hat die kürzeste Periode und erhält daher höchste Priorität (RM)
- zu Beginn der Abfrageperioden t_0 , t_5 , $t_{7.5}$ und t_{10} ist die AJQ leer
- die Ausführung von A erfolgt in Abfrageperiode $t_{2.5}$
 - weil A erst kurz nach dem Abfragezeitpunkt t_0 ausgelöst wurde
- das Laufzeitsystem kommt ohne Verdrängung aus
 - der Zusteller verfügt über genügend Budget, um A innerhalb einer Abfrageperiode vollständig abzuarbeiten

Beispiel: Abfragender Zusteller

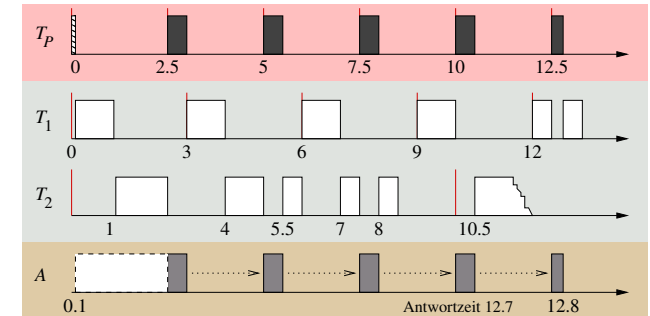
Aufteilung eines aperiodischen Arbeitsauftrags auf mehrere Auffüllperioden

Beispiel:

$$\left. \begin{matrix} T_P \\ T_1 \\ T_2 \end{matrix} \right\} \text{ wie gehabt}$$

RM

$$A \mapsto 2.3(0.1, \infty)$$

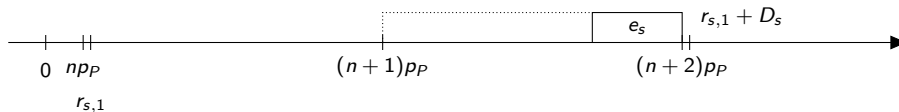


- die Ausführung von A benötigt (mindestens) fünf Abfrageperioden:
 - 4×0.5 Zeiteinheiten (dem Budget von T_P)
 - in den Abfrageperioden $t_{2.5}$, t_5 , $t_{7.5}$ und t_{10}
 - 1×0.3 Zeiteinheiten (bis A beendet ist)
 - in Abfrageperiode $t_{12.5}$; die AJQ ist leer, T_P gibt auf...
- das Laufzeitsystem muss Verdrängung unterstützen

Abfragende Zusteller und sporadische Aufgaben

Wenn der Abfragebetrieb in der Ereignissteuerung an seine Grenzen stößt...

Der periodische Zusteller $T_P = (p_P, e_P)$ behandelt die sporadische Aufgabe $T_S = (i_S, e_S, D_S)$. Zum Zeitpunkt $r_{s,1}$ wird der Job $J_{s,1}$ ausgelöst.



- schlimmstenfalls wird der Abtastzeitpunkt np_P verpasst: $r_{s,1} > np_P$
 - $J_{s,1}$ wird also erst ab der zweiten Abfrageperiode abgearbeitet
- T_P kann durch die ereignisgesteuerte Einplanung verzögert werden
 - die Fertigstellung ist erst zur 3. Abfrageperiode $(n+2)p_P$ garantiert
- der Termin von T_S muss aber gehalten werden: $(n+2)p_P \leq r_{s,1} + D_S$

☞ der Termin D_S begrenzt die Auffüllperiode p_P : $p_P \leq D_S/2$

↪ $D_S \leq i_S$, falls $J_{s,1}$ vor $r_{s,2}$ fertiggestellt sein muss

↪ Gefahr der Überlast – normalerweise gilt: $i_S \ll \overline{r_{s,i+1} - r_{s,i}}$

- Minimale Zwischenankunftszeiten sind u.U. sehr kurz!

Nachteile des Abfragebetriebs

Verfall des noch nicht vollständig ausgeschöpften Ausführungsbudgets eines untätigen Abfragers

↪ längere Antwortzeiten im Falle aperiodischer Aufgaben

↪ Überlast im Fall sporadischer Aufgaben

- sporadische Ereignisse müssten sehr hochfrequent abgefragt werden

- noch eintreffende nicht-periodische Arbeitsaufträge bleiben in der laufenden Abfrageperiode unberücksichtigt
 - Ansammlung in der Warteschlange, der Abfrager wird zurückgestellt
 - so geschehen mit $A \mapsto 0.4(0.1, \infty)$ (s. Folie V-1/17)
- die „zu spät“ eingetroffenen Aufträge werden frühestens in der nächsten Abfrageperiode (entsprechend Ablaufplan) behandelt

☞ das Restbudget eines Abfragers müsste bewahrt werden können...

↪ ... dazu mehr in der nächsten Woche ☺

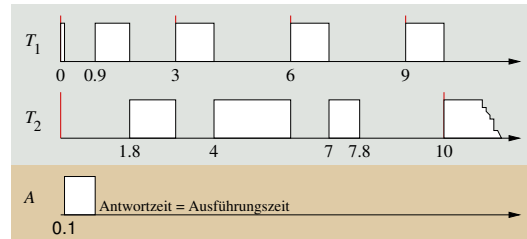
Unterbrecherbetrieb

Antwortzeitminimierung — auf Kosten eines „gut geordneten“ Ablaufplans

- ausgelöste nicht-periodische Arbeitsaufträge werden sofort ausgeführt, sie verdrängen die sich in Ausführung befindliche periodische Aufgabe

Beispiel:

- periodische Tasks
 - $T_1 = (3, 1)$
 - $T_2 = (10, 4)$
 - RM
- aperiodischer Job
 - $A \mapsto 0.8(0.1, \infty)$



- werden nicht-periodische Arbeitsaufträge immer sofort ausgeführt, erhöht sich das Risiko von **Schwankungen** im Ablauf periodischer Aufgaben

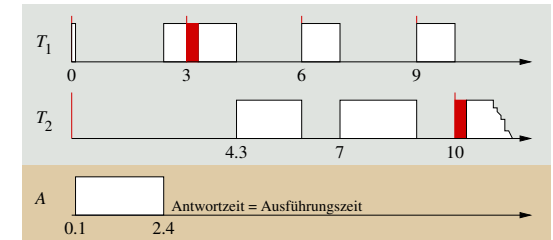
Unterbrecherbetrieb (Forts.)

Antwortzeitminimierung — auf Kosten eines korrekten Ablaufplans

- werden nicht-periodische Arbeitsaufträge bevorzugt ausgeführt, ist die **Termineinhaltung** periodischer Aufgaben **nicht gesichert**

Beispiel (vgl. V-1/21):

- aperiodischer Job
 - $A \mapsto 2.3(0.1, \infty)$
 - run to completion
- periodische Tasks
 - T_1 zu spät
 - T_2 zu spät



- ist die **Schlupfzeit** (s. Folie III-2/29) der unterbrochenen periodischen Aufgabe abgelaufen, muss diese Aufgabe fortgesetzt werden

Apollo 11, Mondlandung

Folklore zum Bordcomputer der Landefähre [2, 6]

- das Rendezvousradar² wurde vor Beginn der Landung eingeschaltet
 - falsche Vorgabe der Checkliste an die Astronauten
- dadurch beanspruchte das Radarsteuerprogramm zuviel Rechenzeit
 - Netzteile von Radar und Landeeinheit waren nicht synchronisiert
 - das Rendezvousradar erzeugte eine Flut von **Scheinunterbrechungen**
 - unerwarteterweise wurden dadurch etwa 15 % an Rechenlast erzeugt
 - Folge: Verzögerung/Ausfall von Berechnungen zur Landungskontrolle
- die Landungskontrolle hatte minimalen Treibstoffverbrauch als Ziel
 - das Kontrollprogramm setzte alle zwei Sekunden ein Kommando ab
 - zur Stabilisierung wurde der Autopilot jede 1/10 Sekunde aktiv
- die Landephase war mit einer Dauer von 11 Minuten geplant
 - fehlerbedingt fielen gut eine Minute lang alle Kontrollkommandos aus
 - dennoch klappte die Landung: Umschaltung auf manuelle Kontrolle

²Messung von Zeitintervallen zwischen bekannten Landmarken und Überprüfung von Position und Geschwindigkeit des Landemoduls relativ zum Kommandomodul.

Maxime von Echtzeitrechensystemen: Unterlast

Kapazitäten gezielt frei lassen ...

- Echtzeitrechensysteme dürfen in kritischen Situation nur bis zu einem vorgegebenen Maximum belastet werden
 - das deutlich unter 100 % liegt
- solche Situationen zu identifizieren, zu bewerten und freizuhalten Kapazitäten zu bestimmen, ist eine große Herausforderung
 - die durchgehende **Anforderungsanalyse**³ zwingend macht
- unterbrechungsbedingte Verzögerungen und Last im Voraus einzuplanen, benötigt eine ordentliche Portion von Expertenwissen
 - das auch Scheinunterbrechungen kaum vorhersagen kann

³engl. *requirements engineering*, Fundament und Teilaktivität systematischer Softwareentwicklung — hier aber nicht nur Software.

Kontrollierter Unterbrecherbetrieb

Die Beeinflussung periodischer Aufgaben durch nicht-periodischer Aufgaben einschränken

Unterbrechungen erschweren eine deterministische Ausführung periodischer Aufgaben oder machen dies gar unmöglich:

- der Zeitpunkt ihres Auftretens ist a-priori **nicht bekannt**
- sie werden gegenüber aktuell ausgeführten Jobs **bevorzugt**

Quelle	max. Frequenz
Messerschalter	333
loser Draht	500
Kippschalter	1000
Wippschalter	1300
serielle Schnittstelle (115 kbps)	11500
Ethernet (10 Mbps)	14880
CAN-Bus	15000
I2C-Bus	50000
USB	90000
Ethernet (100 Mbps)	148800
Ethernet (1 Gbps)	1488000

max. Raten verschiedener Unterbrechungsquellen [5]

- bereits unscheinbare Komponenten erzeugen signifikante Last durch Unterbrechungen
- periodische Aufgaben stehen Unterbrechungen „wehrlös“ gegenüber

☞ Gefahr der **Überlast**

Kontrollierter Unterbrecherbetrieb (Forts.)

Ein Überlastsicherung ist unumgänglich!

Ansatzpunkt unbekannter Zeitpunkt des Auftretens von Unterbrechungen

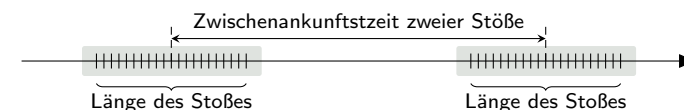
☞ Beschränke das Auftreten von Unterbrechungen [5]

① Überwachung der **minimalen Zwischenankunftszeit**

- die nächste Unterbrechung wird erst angenommen, wenn mindestens die minimale Zwischenankunftszeit verstrichen ist

② Überwachung von **Unterbrechungsstößen** (engl. *bursts*)

- nach einem Unterbrechungsstoß werden alle Unterbrechungen bis zum nächsten Unterbrechungsstoß abgeblockt



Prinzip wird in verbreiteten Echtzeitbetriebssystemen aufgegriffen:

OSEKtime [4] und **AUTOSAR OS** [1] schränken die minimale Zwischenankunftszeit ein oder überwachen sie.

Den Schlupf periodischer Arbeitsaufträge nutzen

Nicht-periodische Arbeitsaufträge in deren Hintergrund ausführen

Ausführung aperiodischer Jobs erfolgt **im Hintergrund** periodischer Jobs

- d.h., wenn keine periodischen Jobs zur Ausführung anstehen
 - in diesen Phasen ist der Prozessor ohnehin **untätig**
- jeder Schlupf auf der gesamten Echtzeitachse kann genutzt werden
 - Ruhephasen werden mit nicht-periodischen Jobs „aufgefüllt“
 - ein unvollendeter nicht-periodischer Job wird bei Bedarf verdrängt
 - der evtl. Jobrest füllt eine spätere Ruhephase (mit) auf...
- die Einplanungsentscheidung wird zur Laufzeit getroffen (**online**)
 - nicht-periodische Jobs werden ereignisbedingt ausgelöst

☞ nicht-periodische Jobs werden zugunsten periodischer Jobs verzögert

- ihre Antwortzeit verschlechtert sich
- Termineinhaltung bei sporadischen Aufgaben wird schwieriger
- die Ansprechempfindlichkeit des Systems lässt nach

Hintergrundbetrieb

Korrektur Ablaufplan — auf Kosten des Antwortverhaltens

- aperiodische Arbeitsaufträge werden nur ausgeführt, falls keine periodischen/sporadischen Arbeitsaufträge zur Ausführung anstehen

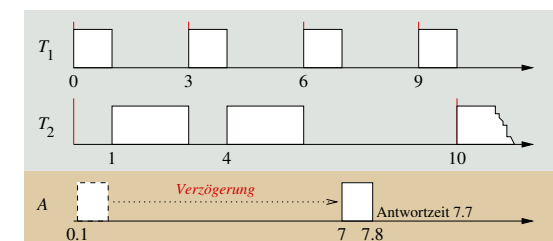
Beispiel:

• periodische Tasks

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

• aperiodischer Job

- $A \mapsto 0.8(0.1, \infty)$



- minimiert die (mittleren) Antwortzeiten nicht-periodischer Arbeitsaufträge nur suboptimal $\leadsto$ **schlechtes Antwortverhalten**

Qual der Wahl...

Abfragender Zusteller $\longleftrightarrow$ Unterbrecherbetrieb $\longleftrightarrow$ Hintergrundbetrieb

Abfragender Zusteller

- + **einfache Implementierung** auf Anwendungsebene
 - nicht-periodischer Arbeitsaufträge $\mapsto$ periodische Aufgabe
- + liefert immer **korrekte Ablaufpläne**
- **lange Antwortzeiten**, weil das Ausführungsbudget aufgegeben wird
- **hoher Overhead** durch den Abfragebetrieb

Unterbrecherbetrieb

- + **sehr gute Antwortzeiten** für nicht-periodische Arbeitsaufträge
- erfordert die **Behandlung von Unterbrechungen**
- verzögert periodische Arbeitsaufträge $\leadsto$ **Überlastgefahr**

Hintergrundbetrieb

- + liefert immer **korrekte Ablaufpläne**
- benötigt **Verdrängung**
- **lange Antwortzeiten** für nicht-periodische Arbeitsaufträge

Gliederung

- 1 Überblick
- 2 Nicht-periodische Aufgaben
 - Zeitparameter
 - Nicht-periodische Echtzeitanwendungen
 - Herausforderung: Mischbetrieb
- 3 Basistechniken
 - Periodischer Zusteller
 - Unterbrecherbetrieb
 - Hintergrundbetrieb
- 4 Slack-Stealing
 - Taktgesteuerte Systeme
 - Vorranggesteuerte Systeme
- 5 Zusammenfassung

Slack-Stealing

Die Abfertigung nicht-periodischer Jobs durch Nutzung der Schlupfzeit optimieren

Echtzeitbetrieb bedeutet **Rechtzeitigkeit** (s. Folie II/11)

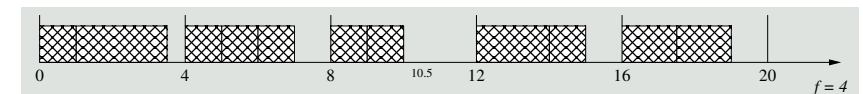
- ☞ Warum sollte man Jobs dann **vor ihrem Termin** fertigstellen?
 - kurz vor oder genau zum Termin ist schließlich ausreichend
- ☞ Verschiebe periodische Jobs nach hinten...
 - führe in der entstehenden Lücke nicht-periodische Jobs aus
- ~ Stehle den Schlupf des periodischen Jobs und nutze ihn!
 - daher der Name: engl. **slack-stealing**
- ☞ ...gefährde aber nicht ihren Termin:
 - sobald der Schlupf aufgebraucht ist,
 - muss der gerade ausgeführte, nicht-periodische Job suspendiert
 - und der verzögerte periodische Job gestartet werden

Slack-Stealing existiert für **takt-** und für **vorranggesteuerte Systeme**

Taktsteuerung und Slack-Stealing

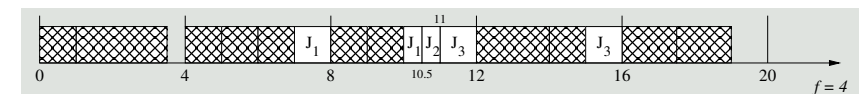
Beispiel: *Major Cycle* eines zyklischen Ablaufplans periodischer Jobs (s. Folie IV-3/11)

- der erste große Durchlauf weist fünf Schlupfbereiche auf



- schraffierte Bereiche bedeuten statisch eingeplante periodische Jobs

- aperiod. Jobs $J_1 \mapsto 1.5(4, \infty]$, $J_2 \mapsto 0.5(9.5, \infty]$, $J_3 \mapsto 2(10.5, \infty]$



- Ausführungszeiten 1.5, 0.5 und 2
- zulässige Ausführungsintervalle (*earliest*, *latest*)
- ∞ meint: der Job hat keinen, einen weichen oder festen Termin

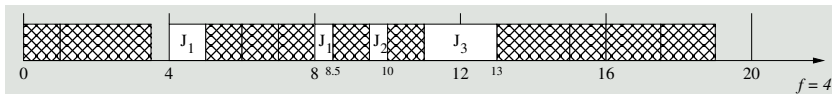
- mittlere Antwortzeit: $((10.5 - 4) + (11 - 9.5) + (16 - 10.5))/3 = 4.5$

Taktsteuerung und Slack-Stealing (Forts.)

Schlupf „stehlen“ (engl. *slack stealing*)

Schlupf in Rahmen k ist die **Zeitspanne** $f - x_k$, wobei x_k Zeiteinheiten bereits für Scheiben periodischer Jobs in k reserviert sind

- Ansatz ist, periodischen Jobs **Zeitpuffer am Rahmenende entziehen**
 - periodische Jobs werden ans Ende ihres Rahmens „geschoben“
 - die Berechnung des Schlupfes geschieht einmal vor der Laufzeit und hängt nur vom aktuellen Rahmen ab
- Jobs J_1 , J_2 und J_3 , wie im Beispiel vorher (s. Folie V-1/32):



J_1 wird sofort eingelastet, muss jedoch verdrängt werden

J_2 wird ebenso behandelt, kann aber komplett durchlaufen

J_3 wird verzögert bis der laufende periodische Job fertig ist

- mittlere Antwortzeit: $((8.5 - 4) + (10 - 9.5) + (13 - 10.5))/3 = 2.5$

Vorrangsteuerung und Slack-Stealing

- technisch ist Slack-Stealing auch hier einfach
 - ein **Schlupfzeit-Dieb** (engl. *slack-stealer*) arbeitet anstehende nicht-periodische Arbeitsaufträge ab, auf **höchster Priorität**, wenn Schlupfzeit vorhanden ist, und auf **niedrigster Priorität**, wenn keine Schlupfzeit vorhanden ist
- aufwändig ist die **Berechnung der Schlupfzeit** [3, S. 233 ff.]
 - im Falle nach EDF geplanter Systeme mit **dynamischen Prioritäten**
 - müssen für **statische vorberechnete Schlupfzeiten** mindestens alle Jobs einer **kompletten Hyperperiode** berücksichtigt werden
 - bei einer **dynamischen Berechnung** muss immer das **aktuelle Tätigkeitsintervall**⁴ betrachtet werden
 - zusätzlich ist Buchführung über **Untätigkeit**, **gestohlenen Schlupf** und **bereits verbrauchte Rechenzeit** der periodischen Jobs notwendig
 - im Fall RM-geplanter Systeme mit **statischen Prioritäten**
 - hängt die Schlupfzeit sogar von ihrem Verwendungszeitpunkt ab
 - der Schlupfzeit-Dieb darf daher nicht **gierig** (engl. *greedy*) sein

⁴dessen Länge zunächst mit Hilfe der Zeitbedarfsanalyse (s. Folie IV-2/37) bestimmt werden muss

Gliederung

- Überblick
- Nicht-periodische Aufgaben
 - Zeitparameter
 - Nicht-periodische Echtzeitanwendungen
 - Herausforderung: Mischbetrieb
- Basistechniken
 - Periodischer Zusteller
 - Unterbrecherbetrieb
 - Hintergrundbetrieb
- Slack-Stealing
 - Taktgesteuerte Systeme
 - Vorranggesteuerte Systeme
- Zusammenfassung

Resümee

Nicht-periodische Aufgaben werden ereignisgesteuert ausgelöst

- harte** o. **feste/weiche Termine** (sporadische/apperiodische Aufgaben)
- ihr **Mischbetrieb** ist eine Herausforderung

Abfragende Zusteller konvertieren sie in periodische Aufgaben

- schlechte Antwortzeiten**, Ausführungsbudget, Auffüllperiode

Unterbrecherbetrieb bevorzugt nicht-periodische Aufgaben

- sehr gut Antwortzeiten, anfällig für **Überlast**
- gefährdet statische Garantien** $\leadsto$ kontrollierter Unterbrecherbetrieb

Hintergrundbetrieb stellt nicht-periodische Aufgaben hinten an

- Antwortzeiten** hängen von der Last periodischer Aufgaben ab

Slack-Stealing ist ein guter Kompromiss

- einfache Umsetzung in gut strukturierten, zeitgesteuerten Systemen
- nicht praktikabel** in vorranggesteuerten Systemen

Literaturverzeichnis

- [1] AUTOSAR:
Specification of Operating System (Version 4.0.0) / Automotive Open System
Architecture GbR.
2009. –
Forschungsbericht
- [2] JR., S. R. M.:
My Fascinating Interview with Allan Klumpp.
[http://www.unt.edu/UNT/departments/CC/Benchmarks/benchmarks_html/sepoct95/
lunar.htm](http://www.unt.edu/UNT/departments/CC/Benchmarks/benchmarks_html/sepoct95/lunar.htm), 1995
- [3] LIU, J. W. S.:
Real-Time Systems.
Prentice-Hall, Inc., 2000. –
ISBN 0–13–099651–3
- [4] OSEK/VDX GROUP:
Time Triggered Operating System Specification 1.0 / OSEK/VDX Group.
2001. –
Forschungsbericht. –
<http://portal.osek-vdx.org/files/pdf/specs/ttos10.pdf>

Literaturverzeichnis (Forts.)

- [5] REGEHR, J. ; DUONGSAA, U. :
Preventing interrupt overload.
In: *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages,
Compilers and Tools for Embedded Systems (LCTES '05).*
New York, NY, USA : ACM Press, 2005. –
ISBN 1–59593–018–3, S. 50–58
- [6] ZÜHLSDORF, R. :
Protokoll des Funkverkehrs bei der ersten Landung auf dem Mond.
<http://members.fortunecity.de/rogerzuehlsdorf/Ap11d.htm>, 1999