

Echtzeitsysteme

Übungen zur Vorlesung

Nicht-periodische Aufgaben: Extended Scope (Teil 1)

Florian Schmaus **Peter Wägemann**

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme)
<https://www4.cs.fau.de>

14. Dezember 2017



- 1 Wiederholung: Nicht-periodische Aufgaben
- 2 Interrupts in Echtzeitsystemen
- 3 Aufgabe 6: Extended Scope



Nicht-periodische Aufgaben

- Definiert durch $T_i = (i_j, e_i, D_i)$
- *Aperiodische* vs. *sporadische* Aufgabe
- *Mischbetrieb*: periodisch \leftrightarrow sporadisch/aperiodisch
 - *dynamische* Einplanung
 - Beeinflussung periodischer Aufgaben?
 - Übernahmeprüfung \leftrightarrow Antwortzeitminimierung

Nicht-periodische Arbeitsaufträge

- Kaum a-priori Wissen (Zeitpunkt, WCET, ...)
- Herausforderung Mischbetrieb: Erhaltung **statischer Garantien**
- Abweisung (spor. Aufg.): schwerwiegende Ausnahmesituation



Basistechniken zur Umsetzung

- **Unterbrecherbetrieb** \leadsto bevorzugt nicht-periodische Aufgaben
- **Hintergrundbetrieb** \leadsto stellt nicht-periodische Aufgaben hinten an
- **Zusteller** \leadsto konvertiert nicht-period. in periodische Aufgaben
 - Spezielle periodische Aufgabe $T_s = (p_s, e_s)$
 - Ausführungsbudget, Auffüllperiode und -regeln
 - Abbildung auf Prioritätswarteschlange (z. B. AJQ)

Slack Stealing

- Idee: Termin ist maßgeblich
 \leadsto *Verschieben* periodischer Aufgaben möglich
- *Erfordert Unterbrecherbetrieb*
- Problem: **Schlupfzeit** bestimmen
 - Zeitsteuerung (mit Rahmen): Einfach $\leadsto f - x_k$
 - Ereignissteuerung: schwierig \leadsto dynamischen Berechnung



Periodische Zusteller

- Verschiedene Ausführungen
z. B.: Polling, Deferrable, Sporadic Server
- Unterscheiden sich im **Regelwerk**
- I. d. R. für mehrere Aufgaben zuständig

Beispiel: Abfragender Zusteller (Polling Server)

- Periodische Aufgabe $T_P = (p_s, e_s)$
 - Budget e_s verfällt
 - Im Falle sporadischer Aufgaben schwierig:
 - $p_P \leq \frac{D_s}{2}$, wobei $D_s \leq j_s \rightsquigarrow$ Abtasttheorem
- hohe Abtastfrequenz, Überlastgefahr



1 Wiederholung: Nicht-periodische Aufgaben

2 Interrupts in Echtzeitsystemen

3 Aufgabe 6: Extended Scope

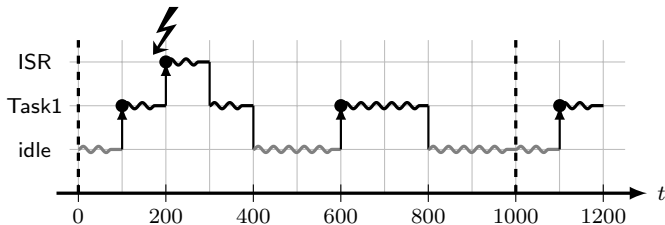




- *Interrupt, ISR*: Hardwareunterstützung für Kontrolltransfer an Interrupt-Handler
- *Interrupt-Handler*: Code der beim Auftreten des Interrupts ausgeführt wird
- *Interrupt-Vektor*: Nummer & Speicheradresse des Interrupt-Handlers
- *Interrupt-Controller*: Hardwareeinheit für Interruptbehandlung
- *Pending Interrupt*: noch nicht abgearbeiteter Interrupt
- *Interrupt-Latenz*: Zeit bis Interrupt erkannt/behandelt wird
- *Geschachtelter Interrupt*



Probleme von Interrupts [1, 2]



- *Prioritätsverletzungen*
- Prellen
 - Entprellung in Soft- oder Hardware
 - Tiefpassfilterung
- Auftrittshäufigkeit
 - Maximale Auftrittsfrequenz (= minimale Zwischenankunftszeit)
 - Soft- oder Hardware-seitige Überwachung
- Auftrittszeitpunkte
 - Zeitliche Garantien jederzeit gewährleisten
 - Zugriff auf Ressourcen



1 *Scheduling*: (WCET-)Analyse muss Interrupts (als Overheads) beachten

2 *Aufrufgraphen*

- **Identifikation der Kontexte** in denen Interrupts auftreten könnten
- ISR → DSR → `cyg_thread_resume()`

3 *Zeitanalyse*

- Bestimmung der **maximalen Auftrittsfrequenz**
- WCET-Analyse der Interrupt-Behandlung (in Isolation)

4 *Korrektheit des Stacks*

- Gemeinsamer Stack?
- Effekte von Interrupts auf Stack-Anordnungen
- Bestimmung von **Stack-Budgets** (worst-case stack usage)

5 *Korrektheit der Nebenläufigkeiten*

- Identifikation von Datenstrukturen auf die nebenläufig zugegriffen wird
- Vermeidung von **Race-Conditions**, Verwendung atomarer Operationen



- Detektion und **Behandlung falscher Interrupts** (engl. spurious interrupts)
- Externe Geräte können fehlerhaft sein \leadsto *Babbling Idiot*
- Software-Lösung (weiche Echtzeit)
 - Zählen von Interrupts über Zeitintervall
 - Verwendet in Linux ¹
 - Nur möglich wenn $WCET(ISR) < \frac{1}{\text{minimale Zwischenankunftszeit}}$
 - Detektion von spurious Interrupts
 - Deaktivierung des IRQs
 - Adaptives Pollen von Geräten
 - ☞ Zusätzlicher Laufzeit-Overhead
- Hardware-Lösung (harte Echtzeit)
 - **Zählen in Hardware** der Auftrittshäufigkeiten
 - TriCore CPU erlaubt das Zählen von externen Ereignissen (Komparatoren)
 - Überwachung implementierbar
 - Kein zusätzlicher Laufzeit-Overhead (außer Konfigurationsaufwand)

¹<https://github.com/torvalds/linux/blob/master/kernel/irq/spurious.c>

1 Wiederholung: Nicht-periodische Aufgaben

2 Interrupts in Echtzeitsystemen

3 Aufgabe 6: Extended Scope



Aufgabe 6: Extended Scope

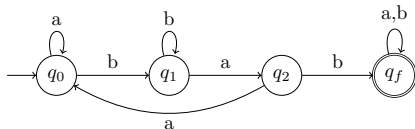


Abbildung: Beispiel für Zustandsmaschine

- Befehlsschnittstelle für Oszilloskop
- Auswertung von Benutzereingaben
 - *Was ist hier problematisch?*
 - Unterbrecher-, Hintergrundbetrieb, Periodischer Zusteller
- *Moduswechsel*
 - Dynamische Anpassung je nach Situation (Rekonfiguration der Ablauf Tabellen)
 - Siehe Vorlesung 4-3
 - Systemweite Koordination mittels *Zustandsmaschine*
- Erweiterte Übung
 - *Besprechung nächste Übung*
 - Rangfolge
 - Mailboxen, Events



- [1] John Regehr.
Safe and structured use of interrupts in real-time and embedded software.
Handbook of Real-Time and Embedded Systems, I. Lee, JY-T. Leug, and SH Son, Eds. Chapman and Hall/CRC, pages 1–13, 2007.
- [2] John Regehr and Usit Duongsaa.
Preventing interrupt overload.
In *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '05)*, pages 50–58, New York, NY, USA, 2005. ACM Press.

