

# Echtzeitsysteme

## Abfertigung periodischer Echtzeitsysteme

Peter Wägemann

Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

<https://sys.cs.fau.de/lehre/ss22/ezs/>

24. Mai 2022



# Fragestellungen

- Was zeichnet **periodische Echtzeitsysteme** aus?
  - Welches **Vorabwissen** ist in solchen Systemen verfügbar?
  - Reicht dies aus, um **sinnvolle Anwendungen** umzusetzen?
  - Welchen **Restriktionen** unterliegen solche Echtzeitsysteme?
- Basismechanismen für die Abarbeitung periodischer Aufträge
  - **Zeitgesteuerte Ausführung**
    - Offline-Einplanung (vgl. III-2/17)
    - Getaktete Abfertigung von Arbeitsaufträgen
    - „Busy Loop“ vs. Ablauftabellen
  - **Ereignisgesteuerte Ausführung**
    - Online-Einplanung (vgl. III-2/17)
    - Unterschied zwischen festen und dynamischen Prioritäten
    - Berechnungskomplexität



# Gliederung

- 1 Periodische Aufgaben
  - Zeitparameter periodischer Aufgaben
  - Periodische Echtzeitanwendungen
  - Restriktionen
- 2 Zeitgesteuerte Ausführung
  - Naive Implementierung
  - Ablauftabellen
  - Einlastung und Laufzeitkontrolle
  - Stapelbasierte Ablaufplanung
- 3 Ereignisgesteuerte Ausführung
  - Feste und dynamische Prioritäten
  - Verdrängbarkeit
  - Ereignisorientierter Planer
  - Berechnungskomplexität
- 4 Zusammenfassung



# Periodische Aufgabe (engl. *periodic task*)

Vorabwissen

## Periodische Aufgaben

Aufgaben die in **regelmäßigen Zeitintervallen**<sup>1</sup> kontinuierlich eine vorgegebene Systemfunktion erbringen.

Eine periodische Aufgabe ( $T_i$ ) ist eine Abfolge von Arbeitsaufträgen ( $J_{i,j}$ ) mit vorgegebenen zeitlichen Eigenschaften.



$$T_i = (p_i, e_i, D_i, \phi_i)$$

$p_i$  Periode (engl. *period*)

$e_i$  maximale Ausführungszeit (WCET)

$D_i$  relativer Termin (engl. *deadline*)

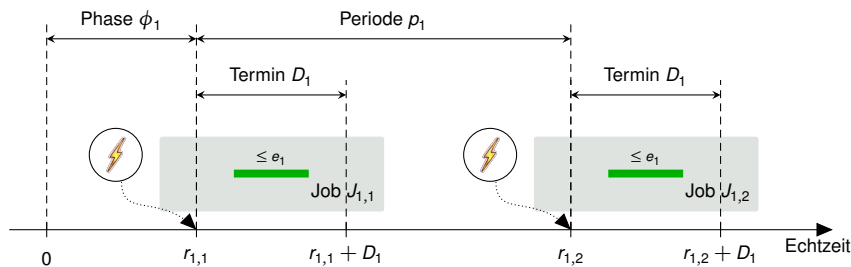
$\phi_i$  Phase (engl. *phase*)

$$J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$$

<sup>1</sup>Nach [1, S. 40 ff] ist eine periodische Aufgabe nicht wirklich periodisch, da die Abstände zwischen den **Auslösezeiten** (engl. *interrelease time*) eines Arbeitsauftrags einer periodischen Aufgabe nicht der Periode selbst entsprechen müssen. Anderswo werden solche Aufgaben verschiedentlich als sporadische Aufgaben bezeichnet.



## Periodische Aufgaben auf der Echtzeitachse

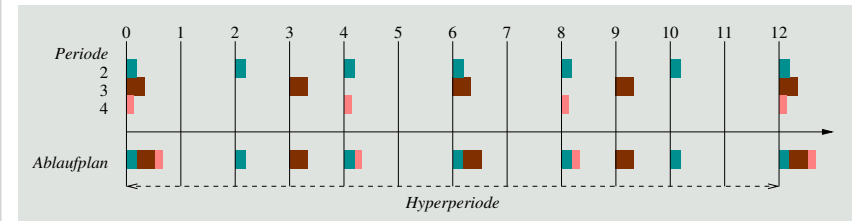


- WCET  $e_i$**  maximale Ausführungszeit aller Aufträge  $J_{i,j}$  in  $T_i$
- relativer Termin  $D_i$**  maximale Spanne zwischen Auslösezeit  $r_{i,j}$  und Fertigstellung  $\rightarrow$  absoluter Termin  $d_{i,j}$  von  $J_{i,j}$
- Periode  $p_i$**  Länge aller Zeitintervalle  $[r_{i,j}, r_{i,j+1}]$  zwischen den Auslösezeiten der Aufträge in  $T_i$
- Phase  $\phi_i$**  Auslösezeit  $r_{i,1}$  des ersten Auftrags  $J_{i,1}$  in  $T_i$  (Abstand von Beginn der Hyperperiode)



## Hyperperiode

Wiederholung eines periodischen Aufgabensystems



### Die Hyperperiode $H$

- Kleinstes gemeinsame Vielfache aller Perioden:  $kgV(p_1 \dots p_i)$
- Startpunkt für Phasenversatz und Berechnung der Auslastung
- Maximale Anzahl aller Arbeitsaufträge in  $H$  ist  $\sum_{i=1}^n H/p_i$ 
  - Hier:  $(12/2) + (12/3) + (12/4) = 13$



Phasenversatz  $\sim$  **Schwankungen** in den Einlastungszeiten

$\rightarrow$  Falls mehrere Arbeitsaufträge zum selben Auslösezeitpunkt anstehen



## Periodische Echtzeitsysteme in der Praxis

Lassen sich Echtzeitsysteme ausschließlich aus periodischen Aufgaben aufbauen?

### Rückgekoppelte Regelschleife (engl. *feedback control loop*)

initialisiere Stellwert;

initialisiere Zeitgeber und Unterbrecher;

bei Zeitgeberunterbrechung **erledige** /\* *abtasten, regeln, steuern* \*/

A/D-Wandlung der Echtzeitinstanz, Echtzeitabbild ziehen;

Echtzeitdatenbasis aktualisieren, neuen Stellwert berechnen;

D/A-Wandlung des Stellwerts, Echtzeitinstanz verändern;

**basta.**



Die Berechnung von Stellwerten für Aktoren ist eine typische Aufgabe von Echtzeitsystemen

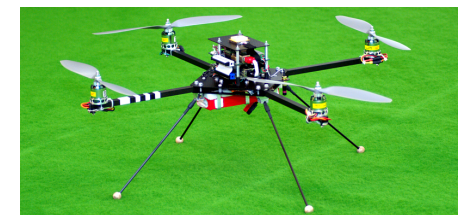
- Das kontrollierte Objekt erfährt eine **direkte digitale Regelung**
- Regelungsanwendungen zeigen dabei eine hohe **Regelmäßigkeit**

$\rightarrow$  Meist endlose Sequenz von Regelzyklen



## Periodische Echtzeitsysteme in der Praxis (Forts.)

Lassen sich Echtzeitsysteme ausschließlich aus periodischen Aufgaben aufbauen?



- Periodische Regelungsaufgaben im **I4Copter**:

alle 3 ms Sensorabtastung, Sensordatenfusion

alle 9 ms Fluglageregelung

alle 21 ms Höhenregelung



Die **zeitliche Auflösung** der Regelung richtet sich nach der **Objektdynamik** (vgl. Folie III-1/7)



## Restriktionen des periodischen Modells

Verzicht auf Entwicklungskomfort zugunsten einer realistischeren Analyse

⚠ Mathematische Ansätze zur zeitlichen Analyse periodischer Echtzeitsysteme bedingen häufig **starke Einschränkungen**:

- 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<sup>2</sup>
- A6** Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar
- A7** Alle Aufgaben verhalten sich voll-präemptiv

<sup>2</sup>D.h. die einzige gemeinsame Ressource ist die CPU und es existieren keine Einschränkungen hinsichtlich der Auslösezeiten der Arbeitsaufträge voneinander.



## Implikationen

Einschränkungen, die Einfluss auf Anwendungen ausüben

- **Betriebsmittel**: gemeinsame Betriebsmittel sind **nicht möglich**
  - ⚠ Implizieren Synchronisation
    - Aufgaben sind nicht mehr unabhängig
    - ⚡ **I4Copter**: Sensoren teilen sich den SPI-Bus
- **Rangordnung**: Komplexe Aufgaben können **nicht geteilt werden**
  - ⚠ Kooperative Dienstleistung  $\rightsquigarrow$  Koordinierung mehrerer Aufgaben
    - Aufgaben sind nicht mehr unabhängig
    - ⚡ **I4Copter**: Sensorik, Fusion und Regelung sind aufgeteilt
- **Kommunikation**: Aufgaben können **nicht synchron kommunizieren**
  - ⚠ Fortschritt hängt von Nachrichtenhandhabung ab
    - Aufgaben sind nicht mehr unabhängig
    - ⚡ **I4Copter**: Synchroner Telemetriedatenübertragung



## Gliederung

- 1 Periodische Aufgaben
  - Zeitparameter periodischer Aufgaben
  - Periodische Echtzeitanwendungen
  - Restriktionen
- 2 Zeitgesteuerte Ausführung
  - Naive Implementierung
  - Ablauftabellen
  - Einlastung und Laufzeitkontrolle
  - Stapelbasierte Ablaufplanung
- 3 Ereignisgesteuerte Ausführung
  - Feste und dynamische Prioritäten
  - Verdrängbarkeit
  - Ereignisorientierter Planer
  - Berechnungskomplexität
- 4 Zusammenfassung



## Busy Loop (Arduino-artige Implementierung eines EZS)

Die wirklich einfachste Variante für die Implementierung zyklischer Systeme?



Periodische Aufgaben wiederholt in einer Schleife ausführen

```
int main(void) {
    unsigned long cnt = 0;
    while(1) {
        warte_durchlauf();
        kontrolle_start();
        aufgabe1();
        kontrolle_stop();

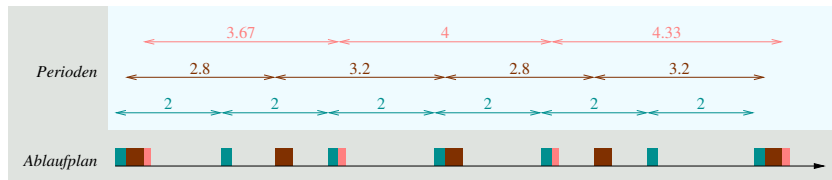
        if(cnt % 2 == 0) {
            aufgabe2_1();
        }
        10ms_nach_aufgabe1();
        if(cnt % 2 == 0) {
            aufgabe2_2();
        }
        ++cnt;
    }
    return 0;
}
```

- Längere Perioden lassen sich durch einen **Rundenzähler** ableiten
  - die Schleife definiert einen **Rahmen**
  - Ausrichtendes Raster für **alle Aktivitäten**
- Explizite Überwachung der **Rahmendauer**
  - Ausführungszeit ist i.d.R. **nicht konstant**
- Schwierige Spezifikation **zeitlichen Versatzes**
  - Abhängigkeit von der **tats. Ausführungszeit**
- Konflikte durch **lange andauernde Aufträge**
  - Evtl. ist eine **manuelle Aufteilung** nötig
- **Überwachung** der Ausführungszeit
  - Schwieriger **Abbruch** des betroffenen Auftrags

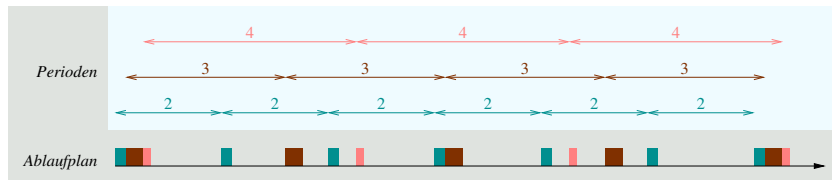


## Genauigkeit periodischer Aufgaben

Einfluss der Einplanung auf Schwankungen in der Einlastung



- Bis auf Periode 2 sind alle anderen Aufträge nicht wirklich periodisch



- Alle Aufträge laufen wirklich periodisch ab: Auftragsabstand = Periode



## Ein Ablaufplan gibt den Takt vor

Falls alle Parameter der Arbeitsaufträge im Voraus bekannt sind ...

- ☞ **Vorabwissen** ermöglicht Ablaufpläne *off-line* zu erstellen (vgl. III-2/17)

- Alle Programme und das System verhalten sich **deterministisch**
  - Oder noch besser **vorhersagbar** (vgl. Folien II/16 ff)

- **Statischer Ablaufplan**  $\mapsto$  exakter Fahrplan

- Feste Angaben wann welche Arbeitsaufträge auszuführen sind
- Zugeteilte Prozessorzeit  $\mapsto$  maximalen Ausführungszeit (WCET)
- Einlastung der Arbeitsaufträge geschieht streng nach Fahrplan
  - Alle Termine werden im Normalfall sicher eingehalten
  - ⚠ Unvorhergesehene Ausnahmen<sup>3</sup> führen zu Terminüberschreitungen

- ☞ Durch *off-line* Einplanung können Algorithmen mit **hoher Berechnungskomplexität** zum Einsatz kommen

<sup>3</sup>Gemeint sind hier die synchronen Programmunterbrechungen (d.h., *Traps*), z.B. aufgrund von Berechnungs- und/oder Adressierungsfehlern



## Abarbeitung statischer Ablaufpläne

Tabellengesteuerte Einlastung von Arbeitsaufträgen

- ☞ Vorberechneter (statischer) Ablaufplan  $\mapsto$  **Ablaufabelle**

- Jeder Tabelleneintrag entspricht einer Einplanungsentscheidung zu einem (vorab) bestimmten Zeitpunkt auf der Echtzeitachse
- Bei Einlastung wird ein **Zeitgeber** (engl. *timer*) programmiert und der Arbeitsauftrag wird gestartet
  - Kurzzeitwecker auf nächsten Entscheidungszeitpunkt stellen
  - Einzustellender Wert ist im aktuellen Tabelleneintrag zu finden
- $\rightarrow$  Ein **Zeitgebersignal** schaltet zum nächsten Tabelleneintrag weiter

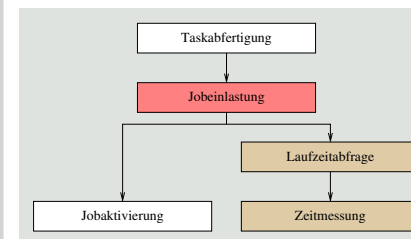
- ⚠ Am Tabellenende wird wieder zum -anfang gesprungen

- **Zyklischer Ablaufplan** (engl. *cyclic schedule*) periodischer Aufgaben
- $\rightarrow$  Die **Hyperperiode** (siehe Folie 6) gibt die Tabellengröße vor



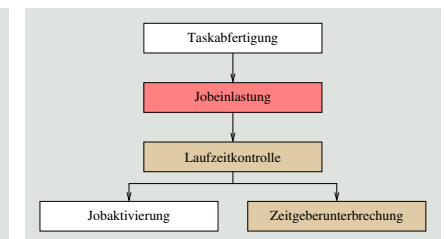
## Abfertigung von Arbeitsaufträgen

Abfragebetrieb (engl. *polling mode*) vs. Unterbrecherbetrieb (engl. *interrupt mode*)



**Abfragebetrieb**

(Folie 18 bis 19)



**Unterbrecherbetrieb**

(Folie 20 bis 22)



## Tabellengesteuerte Einlastung zyklischer Arbeitsaufträge

Taskabfertigung: Grundsätzliche Verfahrensweise

erledige *Dispatcher* (Ablaufabelle, Tabellenlänge):

setze Laufzähler auf ersten Eintrag der Ablaufabelle;

solange der Betrieb läuft tue

erledige

laste Ablaufabelle[Laufzähler].Arbeitsauftrag ein;

wenn Laufzähler < Tabellenlänge dann erhöhe Laufzähler um 1

sonst setze Laufzähler auf ersten Eintrag der Ablaufabelle;

basta;

basta.

Einlastung der Arbeitsaufträge verläuft in drei grundsätzlichen Schritten:

- 1 Laufzeitüberwachung des anstehenden Arbeitsauftrags aufsetzen
- 2 Anstehenden Arbeitsauftrag starten und ausführen
- 3 Sich auf den nächsten Entscheidungszeitpunkt **synchronisieren**



## Synchronisation durch Abfrage eines Taktzählers

Auftragseinlastung, Laufzeitabfrage und Zeitmessung

erledige laste ein (Arbeitsauftrag):

interpretiere Arbeitsauftrag. Entscheidungszeitpunkt als Taktzahl;

aktiviere Arbeitsauftrag;

solange Taktzähler < Taktzahl tue nichts;

basta.



Grundlage bildet **Taktzähler** (engl. *clock counter*) der Hardware

- Entscheidungszeitpunkt muss als Taktzahl vorliegen oder in eine Taktzahl umgerechnet werden können
  - Taktzahl wird nach Beendigung des Arbeitsauftrags abgewartet
- Gezählt werden z.B. die CPU-Takte bei Befehlsausführung



**Verzögerung von Arbeitsaufträgen kann Spätfolgen nach sich ziehen**



## Abfragebetrieb im Rückblick

Verzögerungsproblematik bei Taktzähler und Zeitkontrolle



Abtastung des Zeitgebers durch das **im Vordergrund** laufende Programm

→ Nachdem ein aktivierter Arbeitsauftrag komplett durchgelaufen ist

- Arbeitsaufträge erhalten einen gewissen Vertrauensvorschuss
- Evtl. Terminüberschreitungen werden erst im Nachhinein erkannt



Schwache/strikte Echtzeitfähigkeit liegt ganz in Anwendungshand

**Schwach:** Bei Terminüberschreitung, Ergebnis findet Verwendung

- Der nachfolgende Arbeitsauftrag startet verspätet
- Als Folge kann das System komplett aus den Takt geraten

**Strikt:** Termineinhaltung ist jederzeit garantiert

→ Die **WCET muss die Behandlung evtl. Fehlersituationen einschließen**

Alternative: **Zeitgeberunterbrechung** (engl. *timer interrupt*)



## Synchronisation durch unterbrechenden Zeitgeber

Auftragseinlastung: Einseitige Synchronisation mit Zeitgeberunterbrechung

erledige laste ein (Arbeitsauftrag):

stelle Zeitgeber ein auf Arbeitsauftrag. Entscheidungszeitpunkt;

kontrolliere Arbeitsauftrag;

solange Zeitgebersignalmarke ungesetzt ist tue nichts;

setze Zeitgebersignalmarke zurück;

basta.



Anzeige des Zeitgebersignals durch ein **im Hintergrund** arbeitendes Gerät

- Ausführungsfreigabe durch **Softwaresignal** der Behandlungsroutine
  - Zeitgebersignalmarke, die beim Konsumieren gelöscht wird
  - *Dispatcher* synchronisiert sich mit dem Zeitgeber
- Abbruch des Arbeitsauftrags als Folge einer Zeitgeberunterbrechung
  - Sofern der Arbeitsauftrag dann noch in Ausführung befindlich
  - Ist in Bezug auf die WCET des Arbeitsauftrags ein Ausnahmefall



## Synchronisation durch unterbrechenden Zeitgeber (Forts.)

Laufzeitkontrolle, Zeitgeberunterbrechung: bedingter Auftragabbruch

erledige Behandlungsroutine zum *Timer Interrupt*:

wenn Arbeitsauftrag.Zustand = laufend dann breche Arbeitsauftrag ab;  
setze Zeitgebersignalmarke;  
basta.

- Erfüllung der Wartebedingung für den *Dispatcher*
  - Ggf. Abbruch eines seinen Termin überschreitenden Arbeitsauftrags

erledige kontrolliere (Arbeitsauftrag):  
setze Arbeitsauftrag.Zustand auf laufend;  
aktiviere Arbeitsauftrag;  
setze Arbeitsauftrag.Zustand auf beendet;  
basta.

Schönheitsfehler:

- Zustand
  - Signalmarke
  - unnötige periodische Interrupts
- 🔊 *Ticking System*



## Synchronisation durch unterbrechende Zeitkontrolle: *Tickless System*

Auftragseinlastung, Laufzeitkontrolle, Zeitgeberunterbrechung: unbedingter Auftragabbruch

erledige Behandlungsroutine zum *Timer Interrupt*:

breche Arbeitsauftrag ab;  
basta.

erledige kontrolliere (Arbeitsauftrag):  
lasse Unterbrechung durch Zeitkontrolle zu;  
aktiviere Arbeitsauftrag;  
wehre Unterbrechung durch Zeitkontrolle ab;  
basta.

Ausnahmefall:

- Zeitkontrolle läuft bei Überschreitung der WCET des Arbeitsauftrags ab

erledige laste ein (Arbeitsauftrag):  
richte Zeitkontrolle aus auf Arbeitsauftrag. Entscheidungszeitpunkt;  
kontrolliere Arbeitsauftrag;  
solange Zeitkontrolle  $\neq 0$  tue nichts;  
basta.



## Tickless Systeme in der Praxis

Schlafzustände bei Untätigkeit verlängern

- Unterbrechungen der Zeitkontrolle nur wenn unabdingbar (bei Terminüberschreitung)
- Periodische Unterbrechungen führen zu (unnötigen) Overheads/CRPDs
- Tickless System  $\leadsto$  geringere Overheads
- Weitere Vorteile auch aus Sicht von *Energiesparmaßnahmen*
- Schlafzuständen (engl. *sleep modes*) in Phasen der Untätigkeit (engl. *idle*)

Idle-Task (angelehnt an AVR-Systeme)

```
void idle(void){ // enter a sleep mode when idling
  sleep_enable(); // activate sleep mode
  sei(); // set enable interrupts
  sleep_cpu(); // sleep until next interrupt
}
```

- Tickless  $\leadsto$  Verlängerung von möglichen Schlafphasen
- Auswirkungen auf Mehrkernprozessoren: schlafende Prozessoren werden im Tickless-Betrieb nicht aufgeweckt



## Stapelbasierte Abarbeitung von Ablauf Tabellen

Mischung aus lang andauernden und häufig wiederkehrenden Aufträge unterstützen



**Batch Processing** führt einen Auftrag nach dem anderen aus

- Lang andauernde Aufträge verzögern kurze, häufig wiederkehrende Aufträge
- Diese Aufträge verpassen u.U. deshalb ihre Termine
- Alternativ müssen lange Aufträge aufgeteilt werden



**Stapelbasierte Abarbeitung** von Ablauf Tabellen

- Eingelastete Auftrag verdrängt den aktuell ausgeführten Auftrag
- ⚠ Der ausgeführte Auftrag wird unterbrochen, aber nicht abgebrochen
- Mehrere *kurze* Aufträge *während* eines *langen* Auftrags ausführen



Kontrolle des ausgeführten Auftrags wird schwieriger

- Entscheidungszeitpunkte ermöglichen Einlastung *oder* Kontrolle eines Auftrags, *beides zugleich ist i.A. nicht möglich*
- Ausführungszeit eines Auftrags muss explizit protokolliert werden
- **Terminüberwachung** statt Laufzeitkontrolle notwendig (z.B. OSEKtime [2])

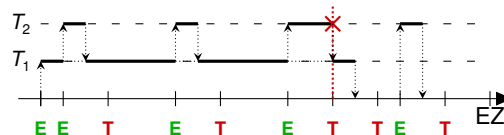


## Stapelbasierte Abarbeitung von Ablauftabellen

Beispiel –  $T_1 = (p : 20, e : 10, D : 15, \phi : 0), T_2 = (5, 1, 2, 1)$

mögliche Ablauftabelle:

Aktion	Aufgabe	Zeit
E	$T_1$	0
E	$T_2$	1
T	$T_2$	3
E	$T_2$	6
T	$T_2$	8
E	$T_2$	11
T	$T_2$	13
T	$T_1$	15
E	$T_2$	16
T	$T_2$	18



- $t = 0$   $T_1$  einlasten
- $t = 1, 6, 11$   $T_2$  einlasten,  $T_1$  verdrängen
- $t = 2, 7$   $T_2$  terminiert,  $T_1$  fortsetzen
- $t = 13$   $T_2$  verfehlt seinen Termin  
 $\leadsto$  Ausnahme auslösen,  $T_2$  abbrechen
- $t = 14$   $T_1$  terminiert
- $t = 16$   $T_2$  einlasten
- $t = 17$   $T_2$  terminiert



## Gliederung

- 1 Periodische Aufgaben
  - Zeitparameter periodischer Aufgaben
  - Periodische Echtzeitanwendungen
  - Restriktionen
- 2 Zeitgesteuerte Ausführung
  - Naive Implementierung
  - Ablauftabellen
  - Einlastung und Laufzeitkontrolle
  - Stapelbasierte Ablaufplanung
- 3 Ereignisgesteuerte Ausführung
  - Feste und dynamische Prioritäten
  - Verdrängbarkeit
  - Ereignisorientierter Planer
  - Berechnungskomplexität
- 4 Zusammenfassung



## Ereignisorientierte Einplanung

(engl. *event-driven scheduling*)

⚠ Einplanung von Arbeitsaufträgen erfolgt zu **Ereigniszeitpunkten**

- Ihr Auftreten ist nicht (exakt) vorhersehbar
  - Ereignisauslöser sind kontrollierte Objekte/andere Arbeitsaufträge
- Die Ereignisverarbeitung unterliegt einer gewissen **Dringlichkeit**

👉 **Ereignisse haben Prioritäten** die dem Ereignisauslöser und/oder der Ereignisverarbeitung zugeordnet sind

**Feste Zuordnung** → Ereignisverarbeitung/-auslöser

- Arbeitsaufträge erhalten **absolute Priorität**

**Variable Zuordnung** → Ereignisverarbeitung

- Arbeitsaufträge erhalten **relative Priorität**

Auch **prioritätsorientierte Einplanung** (engl. *priority-driven scheduling*)



## Prioritätsorientierte Algorithmen

Klassifikation

👉 Verfahren zur **prioritätsorientierten Einplanung** periodischer Arbeitsaufträge werden folglich in zwei Gruppen eingeteilt:

**Feste Priorität** (engl. *fixed priority* oder *static priority*)

- Priorität der Aufträge einer Aufgabe sind **unveränderlich**
  - Die Aufgabenpriorität steht unabhängig von der Auslösung bzw. Beendigung von Arbeitsaufträgen fest
- Prioritäten werden **statisch zum Entwurfszeitpunkt** vergeben

**Dynamische Priorität** (engl. *dynamic priority*)

- Priorität der Aufträge einer Aufgabe sind **veränderlich**
  - Aufgabenpriorität variiert relativ zu anderen Aufgaben, wenn Arbeitsaufträge ausgelöst bzw. beendet werden
- Prioritäten werden **dynamisch zur Laufzeit** vergeben



## Verdrängbarkeit

Verschränkung (engl. *interleaving*) von Arbeitsaufträgen

- ⚠ Arbeitsaufträge könn(t)en verschränkt ausgeführt werden, wenn:
  - Diese verdrängbar sind (typischerweise durch den Planer)
  - Die Zeitbedingungen (engl. *time constraints*) es erlauben
- 🗨 **Präemptivität** (engl. *preemptivity*) ist eine Eigenschaft des jeweiligen Arbeitsauftrags:
  - **Verdrängbar** (engl. *preemptable*) ist ein Arbeitsauftrag, wenn seine Ausführung suspendiert werden darf
    - An beliebigen Stellen (engl. *fully preemptive*)
    - An ausgewiesenen Stellen (engl. *preemption points*)
  - **Unverdrängbar** (engl. *non-preemptable*), sonst
    - Der Arbeitsauftrag läuft durch (engl. *run-to-completion*)

🗨 Mischbetrieb  $\leadsto$  Präemptivität als **Auftragsattribut** implementiert



## Ereignisorientierter Planer

(engl. *event-driven scheduler*)

- Einplanung ereignisbedingt ausgelöster Arbeitsaufträge resultiert in einer **dynamischen Datenstruktur**  $\rightarrow$  sortierte Liste
- ⚠ Kritisch ist die **Berechnungskomplexität** und wann sie anfällt
  - Gekoppelt mit der Einlastung: *online scheduling* (siehe III-2/15 ff)
  - Konstant oder variabel, dann jedoch mit oberer Schranke  $\rightarrow$  WCET  
 $\rightarrow$  Zum **Auslöse-** oder **Auswahlzeitpunkt** von Arbeitsaufträgen
- 🗨 **Priorität** bildet den **Sortierschlüssel** (engl. *sort key*)
  - Ergibt sich ggf. erst zum Ereigniszeitpunkt aus der Priorität der von ihm zu verarbeitenden **Ereignissen**
  - Ist eindeutig abzubilden auf einen endlichen Wertebereich

🗨 Auch **prioritätsorientierter Planer** (engl. *priority-driven scheduler*)



## Aufbau und Berechnungskomplexität

Feste/Dynamische Prioritäten und Ablauf Tabellen/-listen

- Ablaufliste  $\rightarrow$  **dynamische** Datenstruktur
  - Prioritäten entsprechen der Position innerhalb der Ablaufliste
  - Das (relative) Prioritätsgefüge passt sich zur Laufzeit an  
 $\rightarrow$  Eignung für die Implementierung **dynamischer Prioritäten**
  - Linearer Berechnungsaufwand zum Auslösezeitpunkt
    - Vorabwissen zur **WCET des Sortiervorgangs** ist gefordert
  - Konstanter Berechnungsaufwand zum Auswahlzeitpunkt
    - Aufträge vom Kopf her der (ggf. einfach verketteten) Liste entnehmen
- Ablauftabelle  $\rightarrow$  **statische** Datenstruktur
  - Prioritäten werden fest auf Tabellenindizes abgebildet
  - Zur Laufzeit unveränderliches Gefüge absoluter Prioritäten  
 $\rightarrow$  Eignung für die Implementierung **fester Prioritäten**
  - Konstanter Berechnungsaufwand zum Auslösezeitpunkt
    - Aufträge durch indizierte Adressierung in die Tabelle aufnehmen
    - Ggf. ist ein Tabelleneintrag eine Auftragsliste (FIFO) gleicher Priorität
  - Linearer Berechnungsaufwand zum Auswahlzeitpunkt
    - Vorabwissen zur **WCET des Suchvorgangs** ist gefordert
    - Tabelleneinträge können leer sein und sind zu überspringen



## Berechnungskomplexität (Forts.)

Ablaufliste vs. Ablauftabelle

### Ablaufliste

```
Job *list = 0;

void release(Job *item) {
    Job* last = 0, tail = list;

    while(tail && outrank(tail,item)) {
        last = tail;
        tail = last->next;
    }

    if(!last) {
        item->next = list; list = item;
    } else {
        item->next = tail;
        last->next = item;
    }
}

Job* extract() {
    Job* item = list;
    if(item) list = item->next;
    return item;
}
```

`release()`  $\rightarrow \mathcal{O}(n)$  Suchschritte

`extract()`  $\rightarrow \mathcal{O}(1)$  Entnahme

### Ablauftabelle

```
Job* table[Jobs];

void release(Job *item) {
    assert((priority(item) >= 0)
        && (priority(item) <= Jobs - 1));
    item->state = Ready;
}

Job* extract() {
    for(uint slot = 0; slot < Jobs; slot++)
        if(table[slot]->state == Ready) {
            table[slot]->state = Selected;
            return table[slot];
        }
    return 0;
}
```

⚠ **Feste** Anzahl an Aufträgen

`release`  $\rightarrow \mathcal{O}(1)$  Einsetzung

`extract`  $\rightarrow \mathcal{O}(n)$  Suchschritte





## Multi-Level-Queue-Scheduler, MLQ-Scheduler

Häufig anzutreffende Sonderform der Ablaufabelle

- Eine Ablaufliste je Priorität, organisiert als FIFO
- Ablauflisten werden in einer Ablaufabelle verwaltet

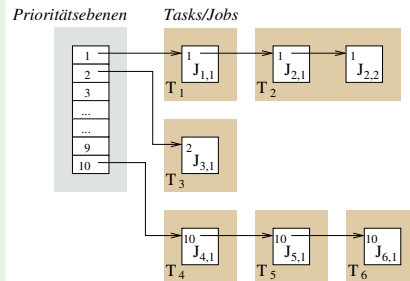
### Multi-Level-Queue

```
Job* table[Jobs];

void release(Job *item) {
    assert((prio(item) >= 0)
        && (prio(item) <= Jobs - 1));
    item->state = Ready;
    append(table[prio(item)],item);
}

Job* extract() {
    for(uint slot = 0; slot < prios; slot++)
        if(!empty(table[slot])) {
            Job *item = head(table[slot]);
            item->state = Selected;
            return item;
        }
    return 0;
}
```

- Mehrere Tasks pro Priorität
- Mehrere Aufträge pro Task
- Reihenfolge der Auslösung



## Gliederung

- 1 Periodische Aufgaben
  - Zeitparameter periodischer Aufgaben
  - Periodische Echtzeitanwendungen
  - Restriktionen
- 2 Zeitgesteuerte Ausführung
  - Naive Implementierung
  - Ablaufabellen
  - Einlastung und Laufzeitkontrolle
  - Stapelbasierte Ablaufplanung
- 3 Ereignisgesteuerte Ausführung
  - Feste und dynamische Prioritäten
  - Verdrängbarkeit
  - Ereignisorientierter Planer
  - Berechnungskomplexität
- 4 Zusammenfassung



## Zusammenfassung

Periodische Aufgaben haben in Echtzeitsystemen eine weite Verbreitung

- Periode, Phase, Hyperperiode, digitale Kontrollschleife
- Restriktionen periodischer Aufgaben und ihre Einschränkungen

Zeitgesteuerte Ausführung periodischer Aufgaben

- naive „Busy Loop“-Implementierung und Ablaufabellen
- Laufzeitkontrolle im Abfrage- und Unterbrecherbetrieb
- Tickless Systeme
- stapelbasierte Ablaufplanung

Ereignisgesteuerte Ausführung periodischer Aufgaben

- Ereignis- bzw. prioritätsorientierte Einplanung
- Feste und dynamische Prioritäten auf Task- bzw. Auftrag-Ebene
- Auslösung vs. Auswahl, Ablaufliste vs. Ablaufabelle
- *Multi-Level-Queue-Scheduler*



## Literaturverzeichnis

- [1] Liu, J. W. S.:  
*Real-Time Systems*.  
Englewood Cliffs, NJ, USA : Prentice Hall PTR, 2000. –  
ISBN 0-13-099651-3
- [2] 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>



# EZS – Cheat Sheet

## Typographische Konvention

Der erste Index gibt die Aufgabe an (z. B.  $D_i$ ), der Zweite (optional) bezieht sich auf den Arbeitsauftrag (z. B.  $d_{i,j}$ ). Exponenten zeigen verschiedene Varianten einer Eigenschaft an (z. B.  $T^{HW}$ ,  $T^{MED}$ ,  $T^{CD}$ ). Funktionen beschreiben zeitlich variierende Eigenschaften (z. B.  $P(t)$ ).

## Eigenschaften

$t$  (Real-)Zeit  
 $d$  Zeitverzögerung (engl. delay)

## Strukturelemente

$E_i$  Ereignis (engl. event)  
 $R_i$  Ergebnis (engl. result)  
 $T_i$  Aufgabe (engl. task)  
 $J_{i,j}$  Arbeitsauftrag (engl. job) der Aufgabe  $T_i$

## Temporale Eigenschaften

Allgemein

$r_i$  Auslösezeitpunkt (engl. release time)  
 $e_i$  Maximale Ausführungszeit (WCET)  
 $D_i$  Relativer Termin (engl. deadline)  
 $d_i$  Absoluter Termin  
 $\omega_i$  Antwortzeit (engl. response time)  
 $\sigma_i$  Schlupf (engl. slack)  
Periodische Aufgaben  
 $p_i$  Periode (engl. period)  
 $\phi_i$  Phase (engl. phase)

## Aufgaben – Tupel

$T_p = (p, e, D, \phi)$  Periodische Aufgabe ohne Priorität (zeitgesteuert oder dynamische Taskpriorität)

