

Echtzeitsysteme

Abfertigung periodischer Echtzeitsysteme

Peter Ulbrich

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

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

11. November 2015



- 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 Jobs
 - **Zeitgesteuerte Ausführung**
 - „*Busy Loop*“ vs. Ablauftabellen
 - zeitgesteuerte Abfertigung von Arbeitsaufträgen
 - **Ereignisgesteuerte Ausführung**
 - ereignisorientierte Einplanung
 - feste und dynamische Prioritäten
 - Berechnungskomplexität: Ablauftabelle vs. Ablaufliste



- 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 Aufgaben

Aufgaben die in **regelmäßigen Zeitintervallen**¹ kontinuierlich eine vorgegebene Systemfunktion erbringen.

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

¹Nach [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

Aufgaben die in **regelmäßigen Zeitintervallen**¹ 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*)

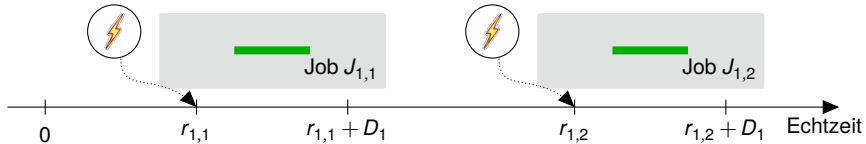
ϕ_i Phase (engl. *phase*)

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

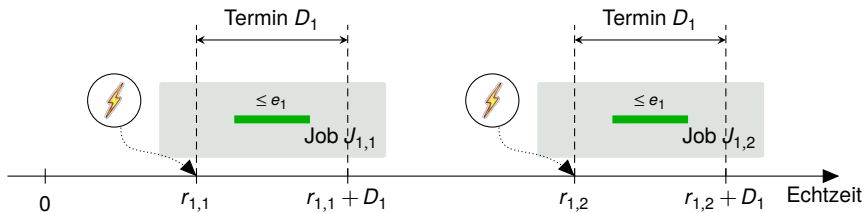
¹Nach [1, S. 40 ff] ist eine periodische Aufgabe nicht wirklich periodisch, da die Abstände zwischen den **Auslösezeiten** (engl. *interrelease time*) eines Arbeitsauftrags einer periodischen Aufgabe nicht der Periode selbst entsprechen müssen. Anderswo werden solche Aufgaben verschiedentlich als sporadische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



Periodische Aufgaben auf der Echtzeitachse



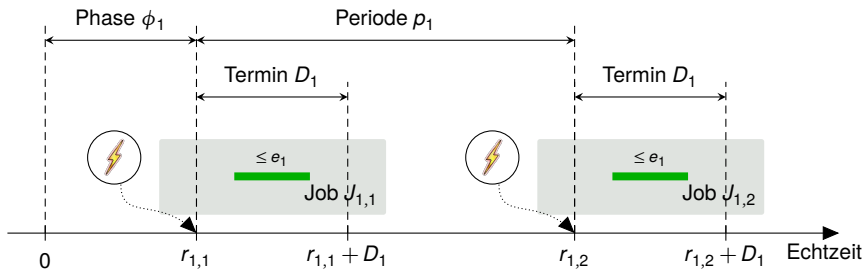
Periodische Aufgaben auf der Echtzeitachse



WCET e_i Maximale Ausführungszeit aller Jobs $J_{i,j}$ in T_i
relativer Termin D_i Maximale Spanne zwischen Auslösezeit $r_{i,j}$ und
Fertigstellung $\rightarrow$ absoluter Termin $d_{i,j}$ von $J_{i,j}$



Periodische Aufgaben auf der Echtzeitachse

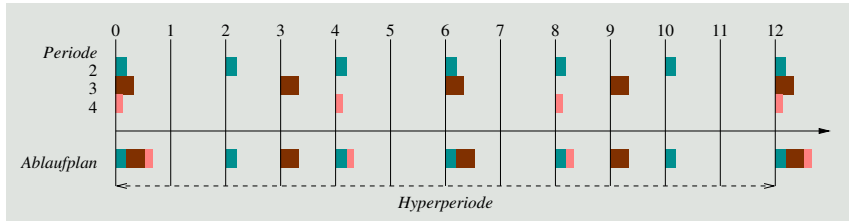


- WCET e_i** Maximale Ausführungszeit aller Jobs $J_{i,j}$ in T_i
- relativer Termin D_i** Maximale Spanne zwischen Auslösezeit $r_{i,j}$ und Fertigstellung $\mapsto$ absoluter Termin $d_{i,j}$ von $J_{i,j}$
- Periode p_i** Minimale Länge aller Zeitintervalle $[r_{i,j}, r_{i,j+1}]$ zwischen den Auslösezeiten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



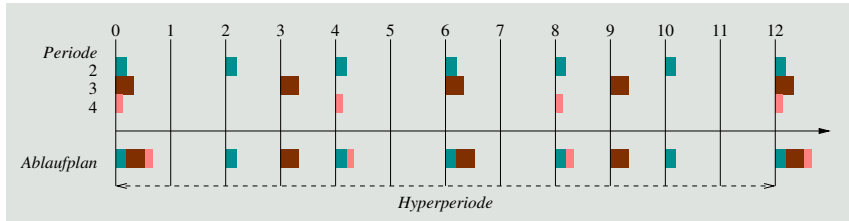
Hyperperiode

Wiederholung eines periodischen Aufgabensystems



Hyperperiode

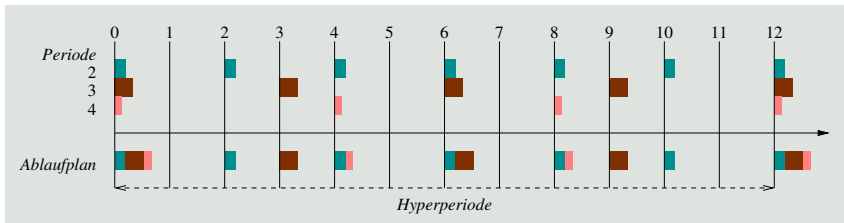
Wiederholung eines periodischen Aufgabensystems



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: $(12/2) + (12/3) + (12/4) = 13$





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 $\leadsto$ **Schwankungen** in den Einlastungszeiten

→ 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



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

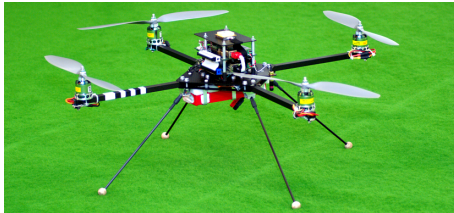


Die Berechnung von Stellwerten für Aktoren ist eine typische Aufgabe von Echtzeitsystemen

- Das kontrollierte Objekt erfährt eine direkte digitale Regelung
 - Regelungsanwendungen zeigen dabei eine hohe Regelmäßigkeit
- Meist endlose Sequenz von Regelzyklen



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

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 unabhängig²
- A6** Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar
- A7** Alle Aufgaben verhalten sich voll-präemptiv

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



- **Betriebsmittel:** Gemeinsame Betriebsmittel sind **nicht möglich**
 - ⚠ Implizieren Synchronisation
 - Aufgaben sind nicht mehr unabhängig



- **Betriebsmittel:** Gemeinsame Betriebsmittel sind **nicht möglich**
 - ⚠ Implizieren Synchronisation
 - Aufgaben sind nicht mehr unabhängig
- **Rangordnung:** Komplexe Aufgaben können **nicht geteilt werden**
 - ⚠ Kooperative Dienstleistung $\leadsto$ Koordinierung mehrerer Aufgaben
 - Aufgaben sind nicht mehr unabhängig



- **Betriebsmittel:** Gemeinsame Betriebsmittel sind **nicht möglich**
 - ⚠ Implizieren Synchronisation
 - Aufgaben sind nicht mehr unabhängig
- **Rangordnung:** Komplexe Aufgaben können **nicht geteilt werden**
 - ⚠ Kooperative Dienstleistung $\leadsto$ Koordinierung mehrerer Aufgaben
 - Aufgaben sind nicht mehr unabhängig
- **Kommunikation:** Aufgaben können **nicht synchron kommunizieren**
 - ⚠ Fortschritt hängt von Nachrichtenhandhabung ab
 - Aufgaben sind nicht mehr unabhängig



- **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 $\leadsto$ 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:** Synchrone Telemetriedatenübertragung



- 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



Die *Busy Loop*

Die wirklich einfachste Variante für die Implementierung zyklischer Systeme?



Periodische Aufgaben wiederholt in einer Schleife ausführen

```
int main(void) {  
  
    while(1) {  
  
        aufgabe1();  
  
    }  
    return 0;  
}
```



Die *Busy Loop*

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) {  
  
        aufgabe1();  
  
        if(cnt % 2 == 0) {  
            aufgabe2();  
        }  
  
        ++cnt;  
    }  
    return 0;  
}
```

- Längere Perioden lassen sich durch einen **Rundenzähler** ableiten
 - die Schleife definiert einen **Rahmen**
 - Ausrichtendes Raster für **alle Aktivitäten**



Die *Busy Loop*

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();  
  
        aufgabe1();  
  
        if(cnt % 2 == 0) {  
            aufgabe2();  
        }  
  
        ++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**



Die *Busy Loop*

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();

        aufgabe1();

        if(cnt % 2 == 0) {
            aufgabe2();
        }

        10ms_nach_aufgabe1();

        ++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**



Die *Busy Loop*

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();

        aufgabe1();

        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 Jobs**
 - Evtl. ist eine **manuelle Aufteilung** nötig



Die *Busy Loop*

Die wirklich einfachste Variante für die Implementierung zyklischer Systeme?



Periodische Aufgaben wiederholt in einer Schleife ausführen

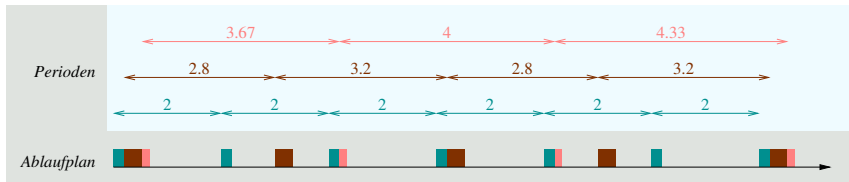
```
int main(void) {
    unsigned long cnt = 0;
    while(1) {
        warte_durchlauf();
        kontrolle_start();
        aufgabe1();
        kontrolle_stop();
        if(cnt % 2 == 0) {
            aufgabe2_1();
        }
        10ms_nach_aufgabe1();
        if(cnt % 2 == 0) {
            aufgabe2_2();
        }
        ++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 Jobs**
 - Evtl. ist eine **manuelle Aufteilung** nötig
- **Überwachung** der Ausführungszeit
 - Schwieriger **Abbruch** des betroffenen Jobs



Genauigkeit periodischer Aufgaben

Einfluss der Einplanung auf Schwankungen in der Einlastung

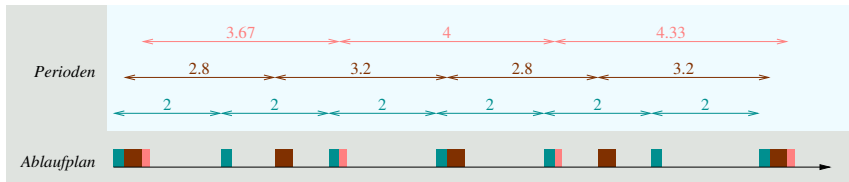


- Bis auf Periode 2 sind alle anderen Jobs nicht wirklich periodisch

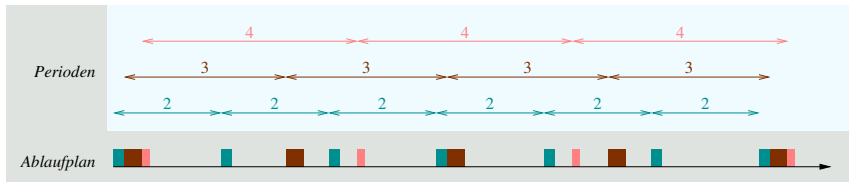


Genauigkeit periodischer Aufgaben

Einfluss der Einplanung auf Schwankungen in der Einlastung



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



- Alle Jobs laufen wirklich periodisch ab: Jobabstand = Periode



Ein Ablaufplan gibt den Takt vor

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



Vorabwissen ermöglicht Ablaufpläne *off-line* zu erstellen

- Alle Programme und das System verhalten sich **deterministisch**
 - Oder noch besser **vorhersagbar** (vgl. Folien II/15 ff)

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



Ein Ablaufplan gibt den Takt vor

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



Vorabwissen ermöglicht Ablaufpläne *off-line* zu erstellen

- Alle Programme und das System verhalten sich **deterministisch**
 - Oder noch besser **vorhersagbar** (vgl. Folien II/15 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 Ausnahmen³ führen zu Terminüberschreitungen

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

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





Vorberechneter (statischer) Ablaufplan $\mapsto$ **Ablauftabelle**

- Jeder Tabelleneintrag entspricht einer Einplanungsentscheidung zu einem (vorab) bestimmten Zeitpunkt auf der Echtzeitachse





Vorberechneter (statischer) Ablaufplan $\mapsto$ **Ablauftabelle**

- 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





Vorberechneter (statischer) Ablaufplan $\mapsto$ **Ablauftabelle**

- 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
- Ein **Zeitgebersignal** schaltet zum nächsten Tabelleneintrag weiter





Vorberechneter (statischer) Ablaufplan $\mapsto$ **Ablauftabelle**

- 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
- Ein **Zeitgebersignal** schaltet zum nächsten Tabelleneintrag weiter



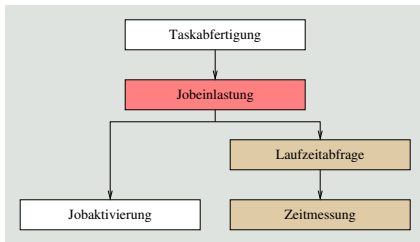
Am Tabellenende wird wieder zum -anfang gesprungen

- **Zyklischer Ablaufplan** (engl. *cyclic schedule*) periodischer Aufgaben
- Die **Hyperperiode** (siehe IV-1/6) gibt die Tabellengröße vor



Abfertigung von Arbeitsaufträgen

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



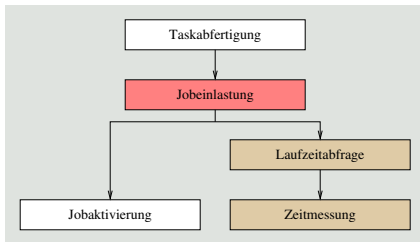
Abfragebetrieb

(S. IV-1/18 bis IV-1/19)



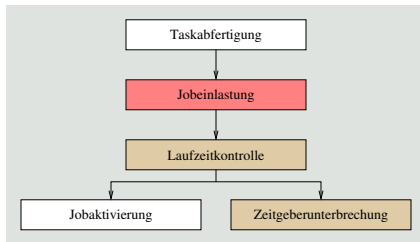
Abfertigung von Arbeitsaufträgen

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



Abfragebetrieb

(S. IV-1/18 bis IV-1/19)



Unterbrecherbetrieb

(S. IV-1/20 bis IV-1/22)



```
erledige Dispatcher (Ablauf-tabelle, Tabellenlänge):  
  setze Laufzähler auf ersten Eintrag der Ablauf-tabelle;  
  solange der Betrieb läuft tue  
    erledige  
      laste Ablauf-tabelle[Laufzähler].Arbeitsauftrag ein;  
      wenn Laufzähler < Tabellenlänge dann erhöhe Laufzähler um 1  
      sonst setze Laufzähler auf ersten Eintrag der Ablauf-tabelle;  
  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

Jobeinlastung, Laufzeitabfrage und Zeitmessung

erledige laste ein (Arbeitsauftrag):

interpretiere Arbeitsauftrag. Entscheidungszeitpunkt als Taktzahl;

aktiviere Arbeitsauftrag;

solange Taktzähler < Taktzahl tue nichts;

basta.



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



Synchronisation durch Abfrage eines Taktzählers

Jobeinlastung, 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





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





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





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**





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

Jobeinlastung: 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



Synchronisation durch unterbrechenden Zeitgeber

Jobeinlastung: 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



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 war
 - Ist in Bezug auf die WCET des Arbeitsauftrags ein Ausnahmefall



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 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.



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
- unnötiger *Interrupt*



Synchronisation durch unterbrechende Zeitkontrolle

Jobeinlastung, Laufzeitkontrolle, Zeitgeberunterbrechung: Unbedingter Jobabbruch

erledige Behandlungsroutine zum *Timer Interrupt*:
breche Arbeitsauftrag ab;
basta.



Synchronisation durch unterbrechende Zeitkontrolle

Jobeinlastung, Laufzeitkontrolle, Zeitgeberunterbrechung: Unbedingter Jobabbruch

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.



Synchronisation durch unterbrechende Zeitkontrolle

Jobeinlastung, Laufzeitkontrolle, Zeitgeberunterbrechung: Unbedingter Jobabbruch

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.

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



Synchronisation durch unterbrechende Zeitkontrolle

Jobeinlastung, Laufzeitkontrolle, Zeitgeberunterbrechung: Unbedingter Jobabbruch

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):
 riche Zeitkontrolle aus auf Arbeitsauftrag. Entscheidungszeitpunkt;
 kontrolliere Arbeitsauftrag;
 solange Zeitkontrolle $\neq 0$ **tue** nichts;
basta.



Stapelbasierte Abarbeitung von Ablauftabellen

Mischung aus lang andauernden und häufig wiederkehrenden Jobs unterstützen



Batch Processing führt einen Job nach dem anderen aus

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



Stapelbasierte Abarbeitung von Ablauftabellen

Mischung aus lang andauernden und häufig wiederkehrenden Jobs unterstützen



Batch Processing führt einen Job nach dem anderen aus

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



Stapelbasierte Abarbeitung von Ablauftabellen

- Eingelastete Job verdrängt den aktuell ausgeführten Job
 - ⚠ Der ausgeführte Job wird nicht abgebrochen
- Mehrere *kurze* Jobs **während** eines *langen* Jobs ausführen



Stapelbasierte Abarbeitung von Ablauftabellen

Mischung aus lang andauernden und häufig wiederkehrenden Jobs unterstützen



Batch Processing führt einen Job nach dem anderen aus

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



Stapelbasierte Abarbeitung von Ablauftabellen

- Eingelastete Job verdrängt den aktuell ausgeführten Job
 - ⚠ Der ausgeführte Job wird nicht abgebrochen
- Mehrere *kurze* Jobs **während** eines *langen* Jobs ausführen



Kontrolle des ausgeführten Jobs wird schwieriger

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

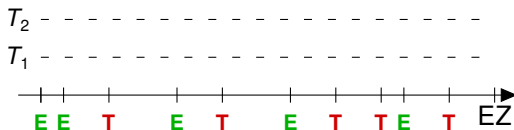


Stapelbasierte Abarbeitung von Ablauftabellen

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

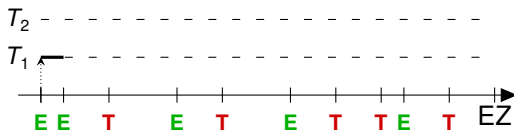


Stapelbasierte Abarbeitung von Ablauftabellen

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



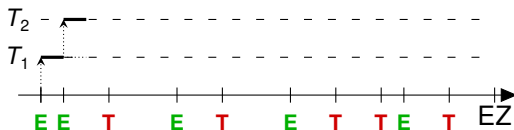
$t = 0$ T_1 einlasten

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

$t = 1$ T_2 einlasten, T_1 verdrängen

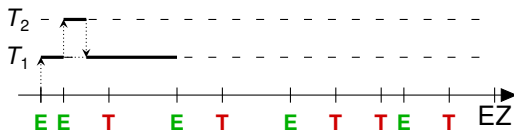


Stapelbasierte Abarbeitung von Ablauftabellen

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

$t = 1$ T_2 einlasten, T_1 verdrängen

$t = 2$ T_2 terminiert, T_1 fortsetzen

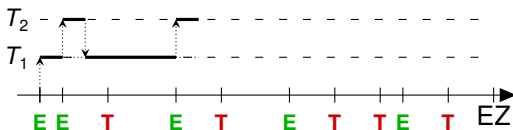


Stapelbasierte Abarbeitung von Ablauftabellen

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

$t = 1,6$ T_2 einlasten, T_1 verdrängen

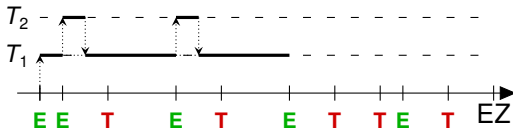
$t = 2$ T_2 terminiert, T_1 fortsetzen

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

$t = 1,6$ T_2 einlasten, T_1 verdrängen

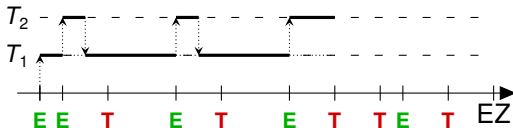
$t = 2,7$ T_2 terminiert, T_1 fortsetzen

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

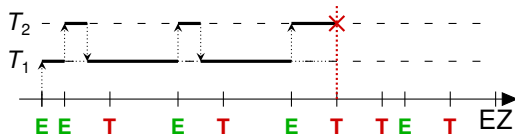


Stapelbasierte Abarbeitung von Ablauftabellen

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

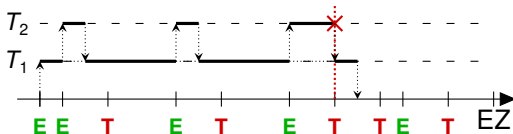


Stapelbasierte Abarbeitung von Ablauftabellen

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

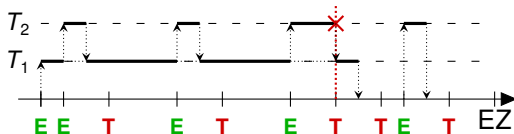


Stapelbasierte Abarbeitung von Ablauftabellen

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



- 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





Einplanung von Arbeitsaufträgen erfolgt zu Ereigniszeitpunkten

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





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

- Arbeitsaufträgen erhalten **absolute Priorität**

Variable Zuordnung → Ereignisverarbeitung

- Arbeitsaufträgen erhalten **relative Priorität**





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

- Arbeitsaufträgen erhalten **absolute Priorität**

Variable Zuordnung → Ereignisverarbeitung

- Arbeitsaufträgen erhalten **relative Priorität**

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





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





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





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





Verschiedene Kategorien von Einplanungsalgorithmen

- **Feste Priorität** wie gehabt (siehe IV-1/27)
- **Dynamische Priorität** (engl. *task-level dynamic-priority*)
 - **Feste Priorität auf Jobebene** (engl. *job-level fixed-priority*)
 - **Dynamische Priorität auf Jobebene** (engl. *job-level dynamic-priority*)





Verschiedene Kategorien von Einplanungsalgorithmen

- **Feste Priorität** wie gehabt (siehe IV-1/27)
- **Dynamische Priorität** (engl. *task-level dynamic-priority*)
 - **Feste Priorität auf Jobebene** (engl. *job-level fixed-priority*)
 - **Dynamische Priorität auf Jobebene** (engl. *job-level dynamic-priority*)



Praxisrelevanz haben Verfahren, die Arbeitsaufträgen feste Prioritäten zuweisen

- Zuweisung erfolgt jedoch zum Auslösezeitpunkt eines Auftrags
 - Wenn er ereignisbedingt auf die **Bereitliste** (engl. *ready list*) kommt
 - Die Priorität eines ausgelösten Auftrags bleibt gleich
- Auf Jobebene sind die Prioritäten fest, auf Taskebene aber variabel





Verschiedene Kategorien von Einplanungsalgorithmen

- **Feste Priorität** wie gehabt (siehe IV-1/27)
- **Dynamische Priorität** (engl. *task-level dynamic-priority*)
 - **Feste Priorität auf Jobebene** (engl. *job-level fixed-priority*)
 - **Dynamische Priorität auf Jobebene** (engl. *job-level dynamic-priority*)



Praxisrelevanz haben Verfahren, die Arbeitsaufträgen feste Prioritäten zuweisen

- Zuweisung erfolgt jedoch zum Auslösezeitpunkt eines Auftrags
 - Wenn er ereignisbedingt auf die **Bereitliste** (engl. *ready list*) kommt
 - Die Priorität eines ausgelösten Auftrags bleibt gleich
- Auf Jobebene sind die Prioritäten fest, auf Taskebene aber variabel



dynamische Priorität → dynamisch auf Task- und fest auf Jobebene





Arbeitsaufträge könn(t)en verschränkt ausgeführt werden, wenn:

- Diese verdrängbar sind (typischerweise durch den Planer)





Arbeitsaufträge könn(t)en verschränkt ausgeführt werden, wenn:

- Diese verdrängbar sind (typischerweise durch den Planer)
- Die Zeitbedingungen (engl. *time constraints*) es erlauben





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:





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*)





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*)





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*)





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 **Jobattribut** implementiert



- Einplanung ereignisbedingt ausgelöster Arbeitsaufträge resultiert in einer **dynamischen Datenstruktur** → sortierte Liste



- Einplanung ereignisbedingt ausgelöster Arbeitsaufträge resultiert in einer **dynamischen Datenstruktur** → 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 → WCET
- Zum **Auslöse-** oder **Auswahlzeitpunkt** von Arbeitsaufträgen



- Einplanung ereignisbedingt ausgelöster Arbeitsaufträge resultiert in einer **dynamischen Datenstruktur** → 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 → WCET
- 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



- Einplanung ereignisbedingt ausgelöster Arbeitsaufträge resultiert in einer **dynamischen Datenstruktur** → 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 → WCET
→ 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*)



- Ablaufliste $\mapsto$ **Dynamische** Datenstruktur
 - Prioritäten entsprechen der Position innerhalb der Ablaufliste
 - Das (relative) Prioritätsgefüge passt sich zur Laufzeit an
 - $\rightarrow$ Eignung für die Implementierung **dynamischer Prioritäten**



- Ablaufliste $\mapsto$ **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



- Ablaufliste $\mapsto$ **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
 - Nahezu konstanter Berechnungsaufwand zum Auswahlzeitpunkt
 - Jobs vom Kopf her der (ggf. einfach verketteten) Liste entnehmen



- Ablaufliste $\mapsto$ **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
 - Nahezu konstanter Berechnungsaufwand zum Auswahlzeitpunkt
 - Jobs vom Kopf her der (ggf. einfach verketteten) Liste entnehmen
- Ablauftabelle $\mapsto$ **Statische** Datenstruktur
 - Prioritäten werden fest auf Tabellenindizes abgebildet
 - Zur Laufzeit unveränderliches Gefüge absoluter Prioritäten
 - $\rightarrow$ Eignung für die Implementierung **fester Prioritäten**



- **Ablaufliste** → **Dynamische** Datenstruktur
 - Prioritäten entsprechen der Position innerhalb der Ablaufliste
 - Das (relative) Prioritätsgefüge passt sich zur Laufzeit an
 - Eignung für die Implementierung **dynamischer Prioritäten**
 - Linearer Berechnungsaufwand zum Auslösezeitpunkt
 - Vorabwissen zur **WCET des Sortiervorgangs** ist gefordert
 - Nahezu konstanter Berechnungsaufwand zum Auswahlzeitpunkt
 - Jobs vom Kopf her der (ggf. einfach verketteten) Liste entnehmen
- **Ablaufabelle** → **Statische** Datenstruktur
 - Prioritäten werden fest auf Tabellenindizes abgebildet
 - Zur Laufzeit unveränderliches Gefüge absoluter Prioritäten
 - Eignung für die Implementierung **fester Prioritäten**
 - Konstanter Berechnungsaufwand zum Auslösezeitpunkt
 - Jobs durch indizierte Adressierung in die Tabelle aufnehmen
 - Ggf. ist ein Tabelleneintrag eine Jobliste (FIFO) gleicher Priorität



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



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;
}
```



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 $O(n)$

extract nahezu $O(1)$



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 $O(n)$

extract nahezu $O(1)$

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;
}
```



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 $O(n)$

extract nahezu $O(1)$

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;
}
```

release $O(1)$

extract $O(n)$, obere Schranke Jobs



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



Multi-Level-Queue-Scheduler, MLQ-Scheduler

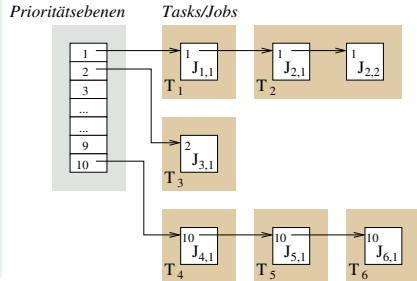
Häufig anzutreffende Sonderform der Ablaufabelle

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

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





Jobauslösung mit konstantem Aufwand $O(1)$ erfordert:





Jobauslösung mit konstantem Aufwand $O(1)$ erfordert:

- 1 Ablaufplan ist dynamische Datenstruktur (Tabelle) aus mehrere Prioritätsebenen

Wartelisten $\mapsto$ LIFO

Warteschlangen $\leadsto$ FIFO





Jobauslösung mit konstantem Aufwand $O(1)$ erfordert:

- 1 Ablaufplan ist dynamische Datenstruktur (Tabelle) aus mehrere Prioritätsebenen
 - Wartelisten $\mapsto$ LIFO
 - Warteschlangen $\leadsto$ FIFO
- 2 Jobs die über denselben Tabelleneintrag erfasst werden besitzen dieselbe Priorität $\leadsto$ Prioritätsschlange
 - Sonst könnte LIFO/FIFO Prioritätsverletzung zur Folge haben





Jobauslösung mit konstantem Aufwand $O(1)$ erfordert:

- 1 Ablaufplan ist dynamische Datenstruktur (Tabelle) aus mehrere Prioritätsebenen
 - Wartelisten $\mapsto$ LIFO
 - Warteschlangen $\leadsto$ FIFO
- 2 Jobs die über denselben Tabelleneintrag erfasst werden besitzen dieselbe Priorität $\leadsto$ Prioritätsschlange
 - Sonst könnte LIFO/FIFO Prioritätsverletzung zur Folge haben
- 3 Anzahl der Tabelleneinträge entspricht mindestens der Anzahl statisch zugewiesener Prioritäten
 - Ggf. werden dann nahezu alle Tabelleneinträge nur einen Job erfassen
 - Abhängig von der Echtzeitanwendung und dem Einplanungsverfahren





Jobauslösung mit konstantem Aufwand $O(1)$ erfordert:

- 1 Ablaufplan ist dynamische Datenstruktur (Tabelle) aus mehrere Prioritätsebenen
 - Wartelisten $\mapsto$ LIFO
 - Warteschlangen $\leadsto$ FIFO
- 2 Jobs die über denselben Tabelleneintrag erfasst werden besitzen dieselbe Priorität $\leadsto$ Prioritätsschlange
 - Sonst könnte LIFO/FIFO Prioritätsverletzung zur Folge haben
- 3 Anzahl der Tabelleneinträge entspricht mindestens der Anzahl statisch zugewiesener Prioritäten
 - Ggf. werden dann nahezu alle Tabelleneinträge nur einen Job erfassen
 - Abhängig von der Echtzeitanwendung und dem Einplanungsverfahren

Jobauswahl ist unter diesen Bedingungen nicht in $O(1)$ möglich:

- Leere Tabelleneinträge sind ggf. zu überspringen





Vorrangsteuerung ist mit grundsätzlichen Konflikt konfrontiert:

- Entweder Jobauslösung oder Jobauswahl mit $O(1)$ zu versehen
 - Beides zugleich geht nicht





Vorrangsteuerung ist mit grundsätzlichen Konflikt konfrontiert:

- Entweder Jobauslösung oder Jobauswahl mit $O(1)$ zu versehen
 - Beides zugleich geht nicht



Für Jobauslösung in $O(1)$ spricht:

- Ereignisgesteuerte Auslösung benötigen konstante Zeit
 - Z.B. als Folge eines *Interrupts* oder der Zustellung eines *Zeitsignals*
 - Bedeutsam für voll-verdrängbare Systeme
- Ereignisbedingte Jobverzögerungen lassen sich exakt bestimmen





Vorrangsteuerung ist mit grundsätzlichen Konflikt konfrontiert:

- Entweder Jobauslösung oder Jobauswahl mit $O(1)$ zu versehen
 - Beides zugleich geht nicht



Für Jobauslösung in $O(1)$ spricht:

- Ereignisgesteuerte Auslösung benötigen konstante Zeit
 - Z.B. als Folge eines *Interrupts* oder der Zustellung eines *Zeitsignals*
 - Bedeutsam für voll-verdrängbare Systeme
- Ereignisbedingte Jobverzögerungen lassen sich exakt bestimmen



Für Jobauswahl in $O(1)$ spricht:

- Übergang zum nachfolgenden Job benötigt konstante Zeit
 - Z.B. wenn der aktuelle Job durchgelaufen ist oder blockiert





Vorrangsteuerung ist mit grundsätzlichen Konflikt konfrontiert:

- Entweder Jobauslösung oder Jobauswahl mit $O(1)$ zu versehen
 - Beides zugleich geht nicht



Für Jobauslösung in $O(1)$ spricht:

- Ereignisgesteuerte Auslösung benötigen konstante Zeit
 - Z.B. als Folge eines *Interrupts* oder der Zustellung eines *Zeitsignals*
 - Bedeutsam für voll-verdrängbare Systeme
- Ereignisbedingte Jobverzögerungen lassen sich exakt bestimmen



Für Jobauswahl in $O(1)$ spricht:

- Übergang zum nachfolgenden Job benötigt konstante Zeit
 - Z.B. wenn der aktuelle Job durchgelaufen ist oder blockiert

Linux (bis 2.6), Mach, QNX, ..., VxWorks verhelfen Jobauslösung zu $O(1)$



- 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 Aufgaben haben in Echtzeitsystemen eine weite Verbreitung

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



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 Ablauftabellen
- Laufzeitkontrolle im Abfrage- und Unterbrecherbetrieb
- stapelbasierte Ablaufplanung

Ereignisgesteuerte Ausführung periodischer Aufgaben

- Ereignis- bzw. prioritätsorientierte Einplanung
- Feste und dynamische Prioritäten auf Task- bzw. Job-Ebene
- Auslösung vs. Auswahl, Ablaufliste vs. Ablauftabelle
- *Multi-Level-Queue-Scheduler*, Prioritätsorientierter *O(1)-Scheduler*



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 Ablauftabellen
- Laufzeitkontrolle im Abfrage- und Unterbrecherbetrieb
- stapelbasierte Ablaufplanung

Ereignisgesteuerte Ausführung periodischer Aufgaben

- Ereignis- bzw. prioritätsorientierte Einplanung
- Feste und dynamische Prioritäten auf Task- bzw. Job-Ebene
- Auslösung vs. Auswahl, Ablaufliste vs. Ablauftabelle
- *Multi-Level-Queue-Scheduler*, Prioritätsorientierter *O(1)-Scheduler*



- [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>



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^{HI}, T^{MED}, T^{LO}). Funktionen beschreiben zeitlich variierende Eigenschaften (z. B. $P(t)$).

Temporale Eigenschaften

Allgemein

- r_i Auslösezeitpunkt (engl. release time)
- e_i Maximale Ausführungszeit (WCET)
- D_i Relativer Termin (engl. deadline)
- d_i Absoluter Termin
- ω_i Antwortzeit (engl. response time)

Periodische Aufgaben

- p_i Periode (engl. period)
- ϕ_i Phase (engl. phase)

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

