Echtzeitsysteme

Taktsteuerung

24. November 2008

Überblick

Arbeitsaufträge mit strikten Terminen

Alle Parameter der Arbeitsaufträge sind im Voraus bekannt

Vorabwissen bahnt den Weg, um Ablaufpläne *off-line* erstellen zu können

- alle Programme sind determiniert, das System ist deterministisch

statischer Ablaufplan $\mapsto$ exakter Jobfahrplan; enthält feste Angaben darüber, wann welche Arbeitsaufträge auszuführen sind

- die jedem Arbeitsauftrag zugeteilte Prozessorzeit ist gleich seiner maximalen Ausführungszeit $\rightsquigarrow$ WCET
- Einlastung der Arbeitsaufträge geschieht streng nach Fahrplan
  - alle Termine werden im Normalfall sicher eingehalten
  - unvorhergesehene Ausnahmen[1] führen zu Terminüberschreitungen
- da die Einplanung *off-line* geschieht, können Algorithmen mit hoher Berechnungskomplexität zum Einsatz kommen

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

Abarbeitung statischer Ablaufpläne

Tabellengesteuerte Einlastung von Arbeitsaufträgen

Repräsentation vorberechneter (statischer) Ablaufpläne $\rightsquigarrow$ Tabelle

- jeder Tabelleneintrag entspricht einer Einplanungsentscheidung zu einem (vorab) bestimmten Zeitpunkt auf der Echtzeitachse
- dabei werden zwei Arten von Tabelleneinträgen unterschieden:
  1. Adresse bzw. Identifikation eines Arbeitsauftrags
  2. Ruheintervall (engl. *idle interval*) einer Aufgabe
- bei Einlastung wird ein Zeitgeber (engl. *timer*) programmiert und der Arbeitsauftrag/das Ruheintervall wird gestartet
  - „Kurzzeitwecker“ auf nächsten Entscheidungszeitpunkt stellen
  - einzustellender Wert ist im aktuellen Tabelleneintrag zu finden
- ein Zeitgebersignal schaltet zum nächsten Tabelleneintrag weiter

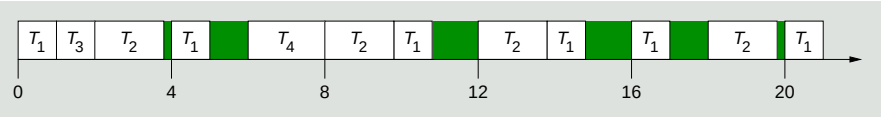
Reihumverfahren: am Tabellenende wird wieder zum -anfang gesprungen

- zyklischer Ablaufplan (engl. *cyclic schedule*) periodischer Aufgaben

Ruheintervalle periodischer Aufgaben

Arbeitsaufträge, um überschüssige Zeit zu verbrauchen...

Phasen von beabsichtigter „Untätigkeit“ zwischen den Arbeitsaufträgen:



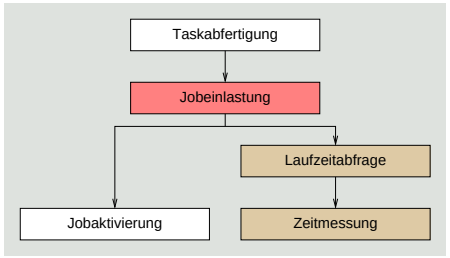
- nicht beanspruchte, freie/verfügbare Zeitintervalle in den Perioden
  - der mit periodischen Arbeitsaufträgen ggf. entstehende „Verschnitt“
- spezielle Arbeitsaufträge an die CPU, untätig (engl. *idle*) zu sein

Nutzung der Ruheintervalle für andere Zwecke kann möglich sein:

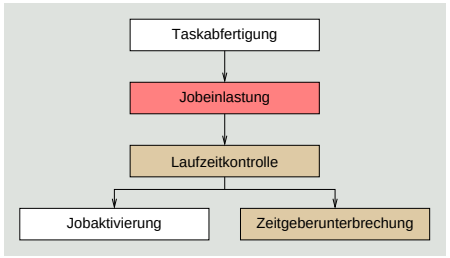
- z.B. zur Ausführung aperiodischer/sporadischer Arbeitsaufträge
- bzw. zur Hintergrundausführung sonstiger (nicht Echtzeit) Jobs

Abfertigung von Arbeitsaufträgen

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



Abfragebetrieb
(S. 5-7 bis 5-9, 5-13)



Unterbrecherbetrieb
(S. 5-10 bis 5-12, 5-13)

Benutzthierarchie

Die Benutztbeziehung [1] in einer funktionalen Hierarchie drückt Abhängigkeiten von der Verfügbarkeit korrekter Implementierungen von Funktionen aus. *A* benutzt *B*, wenn die korrekte Ausführung von *B* zwingend ist für die Korrektheit von *A*: d.h., die Korrektheit von *A* hängt ab von der Korrektheit von *B* (*A* liegt über *B*).

Tabellengesteuerte Einlastung zyklischer Arbeitsaufträge

Taskabfertigung: Grundsätzliche Verfahrensweise

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

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

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

☞ Verzögerung von Arbeitsaufträgen kann Spätfolgen nach sich ziehen

Synchronisation durch Abfrage einer Zeitkontrolle

Jobeinlastung, Laufzeitabfrage und Zeitmessung

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

Zeitkontrolle im Sinne von „zurück zählen“ (engl. *count down*)

- der Entscheidungszeitpunkt muss als relativer Zeitwert vorliegen oder in einen solchen umgerechnet werden können
  - auf diesen Wert wird ein Zeitmesser (engl. *timer*) eingestellt
- für den Zeitwert t gilt: $t \geq WCET(Arbeitsauftrag)$

☞ 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 sonst, d.h., wenn Termineinhaltung jederzeit garantiert ist
- die WCET muss die Behandlung evtl. Fehlersituationen einschließen

☞ Alternative: Zeitgeberunterbrechnung (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

- Ausführungsfreigabe durch Softwaresignal der Behandlungsroutine
  - hier: die Zeitgebersignalmarke, die beim Konsumieren gelöscht wird
  - der *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

Synchronisation durch unterbrechenden Zeitgeber (Forts.)

Laufzeitkontrolle, Zeitgeberunterbrechung: Bedingter Jobabbruch

erledige Behandlungsroutine zum *Timer Interrupt*:
  wenn Arbeitsauftrag.Zustand = laufend dann breche Arbeitsauftrag ab;
  setze Zeitgebersignalmarke;
basta.

Erfüllung der Wartebedingung für den (aktiv wartenden) *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.
```

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

Ausnahmefall die Zeitkontrolle läuft bei Überschreitung der WCET des Arbeitsauftrags ab

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


Aktivierung eines Arbeitsauftrags

Frage der technischen Repräsentation: Routine vs. Koroutine

```
erledige aktiviere (Arbeitsauftrag):
  rufe Arbeitsauftrag.Routine auf;
basta.
```

Arbeitsauftrag ↦ Routine

- ggf. auch als Makro
  - C/C++ *inline function*

```
erledige aktiviere (Arbeitsauftrag):
  setze Arbeitsauftrag.Koroutine fort;
basta.

erledige Koroutine (Arbeitsauftrag):
  solange der Betrieb läuft tue
    erledige
      rufe Arbeitsauftrag.Routine auf;
      setze Dispatcher.Koroutine fort;
    basta;
basta.
```

Arbeitsauftrag ↦ Koroutine

- autonomer Kontrollfluss
- eigener Laufzeitkontext

Aktivitätsträger einer Routine

- Wiederverwendung der prozeduralen Ausprägung des Arbeitsauftrags
- kooperative Verarbeitung mehrfädiger Programme

Abbruch von Arbeitsaufträgen

Ausnahmebehandlung (engl. *exception handling*)

Arbeitsauftragsabbrüche unterscheiden sich je nach Aktivierungsmodell

- Aufruf/Aktivierung einer Routine oder Koroutine
  - prozedur- oder prozessorientierter Ansatz

Prozeduraufruf ↦ Aktivierungsblöcke zerstören

- *Dispatcher* und Arbeitsauftrag laufen im selben Programmfaden ab
- der *Timer Interrupt* terminiert Prozedurinkarnationen
- in der nächsten Periode beginnt der Arbeitsauftrag von vorne

Koroutinenaufruf ↦ *Dispatcher*-Kontext reaktivieren

- *Dispatcher*/Arbeitsaufträge laufen in eigenen Programmfäden ab
- der *Timer Interrupt* verdrängt den Arbeitsauftragsfaden
- in der nächsten Periode fährt der Arbeitsauftrag an der Stelle fort
  - ist je nach Arbeitsauftrag sinnvoll oder nicht zu tolerieren
  - ggf. ist wie beim Prozeduraufrufmodell zu verfahren. . .

Variantenvielfalt von Arbeitsaufträgen

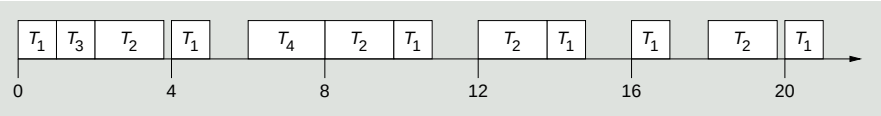
Programmiersprachliche Formulierung

Qual der Wahl. . .

- jede der behandelten Optionen ist bedeutsam für eine bestimmte Anwendungsklasse und folglich auch sinnvoll
  - bei weiterer Konkretisierung werden sich zusätzliche Optionen ergeben
- den Arbeitsauftrag als parametrischen Datentypen formulieren
  - d.h., als „Programmgerüst“ bzw. Schablone (engl. *template*)
- der Datentypparameter bestimmt sodann die technische Ausprägung
  - Laufzeitüberwachung:
    - Taktzähler
    - Zeitkontrolle oder Zeitgeber, abfragend bzw. unterbrechend
  - Laufzeitkontext:
    - gemeinsamer Kontext mit der aufrufenden Instanz ⇝ Routine
    - ein von der aufrufenden Instanz getrennter Kontext ⇝ Koroutine
  - Prozedurart: konventionell, *inline*, *virtual*, *pure virtual*
- Wunsch: linguistische Unterstützung für generische Programmierung

Regelmäßigkeit zyklischer Abläufe

Einplanungsentscheidungen können trotz periodischer Aufgaben *ad hoc*, d.h., in unregelmäßigen Abständen wirksam werden:



- Entscheidungszeitpunkte sind 0, 1, 2, 4, 6, 8, 10, 12, 14, 16, 18
- zusätzlich die Zeitpunkte für Ruheintervalle: 3.8, 5, 11, 15, 17, 19.8

Regularität bei der Umsetzung und Überprüfung solcher Entscheidungen zur Laufzeit trägt wesentlich zum Determinismus bei

„gute Anordnung“ (engl. *good structure*) eines zyklischen Ablaufplans

- Einplanungsentscheidungen *nicht* zu beliebigen Zeitpunkten treffen

Rahmen (engl. *frames*)

Strukturelemente von zyklischen Ablaufplänen

Zeitpunkte von Einplanungsentscheidungen unterteilen die Echtzeitachse in Intervalle fester Länge f (engl. *frame size*)

- Entscheidungen werden nur am Rahmenanfang getroffen/wirksam
  - innerhalb eines Rahmens ist Verdrängung ausgeschlossen
- Folge: die Phase einer periodischen Aufgabe ist Vielfaches von f
  - der erste Job jeder Task wird am Anfang eines Rahmens ausgelöst

Aufgaben, die der *Dispatcher* zusätzlich zur Einlastung eines Jobs am Anfang eines Rahmens durchführen kann...

- sind Überwachung/Durchsetzung von Einplanungsentscheidungen:
  - wurde ein für den Rahmen eingeplanter Job bereits ausgelöst?
  - ist dieser Job auch zur Ausführung bereit?
  - gab es einen „Überlauf“ eines Termins, steht Fehlerbehandlung an?
- beinflussen im großen Maße die Bestimmung eines Wertes für f

Randbedingungen für die Rahmenlänge

Lang genug und so kurz wie möglich halten...

f hinreichend lang $\rightsquigarrow$ Jobverdrängung vermeiden

1. ist erfüllt, wenn gilt: $f \geq \max(e_i)$, für $1 \leq i \leq n$
   - jeder Job läuft in der durch f gegebenen Zeitspanne komplett durch
2. f teilt die Hyperperiode H so, dass gilt: $\lfloor p_i/f \rfloor - p_i/f = 0$
   - die Periode einer beliebigen Task in H kann hergenommen werden

- das Intervall in H von F Rahmen heißt größter Durchlauf
  - engl. *major cycle*, beginnt mit Rahmen $kF + 1$, für $k = 0, 1, 2, \ldots$
  - Intervall der Länge f heißt kleinster Durchlauf (engl. *minor cycle*)
- im Regelfall verlängert sich der Ablaufplan: Vielfaches von f

f hinreichend kurz $\rightsquigarrow$ Terminüberwachung unterstützen

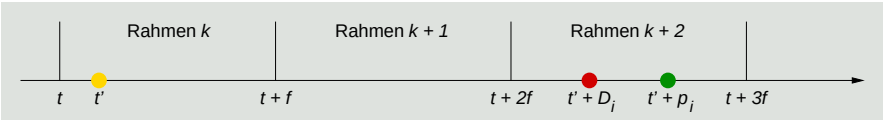
3. ist möglich unter der Bedingung: $2f - gcd(p_i, f) \leq D_i$

- Rahmen „passend“ auf die anstehenden Aufgaben verteilen
  - zwischen der Auslösezeit und dem Termin jedes Jobs (S. 5-19)

Randbedingungen für die Rahmenlänge (Forts.)

Platzierung einer Task auf der Echtzeitachse

Feststellung eines passenden Bereichs für f von $T = (p_i, e_i, D_i)$:[2]



- t ist der Anfang eines Rahmens, in dem ein Job in T_i ausgelöst wird
- t' ist der Zeitpunkt der Auslösung des betreffenden Jobs
- Rahmen $k + 1$ erlaubt die Kontrolle des bei t' ausgelösten Jobs
  - der Rahmen sollte daher zwischen t' und $t' + D_i$ des Jobs liegen
- dies ist erfüllt wenn gilt: $t + 2f \leq t' + D_i$ bzw. $2f - (t' - t) \leq D_i$
  - $t' - t$ ist mindestens größter gemeinsamer Teiler von p_i und f [2]

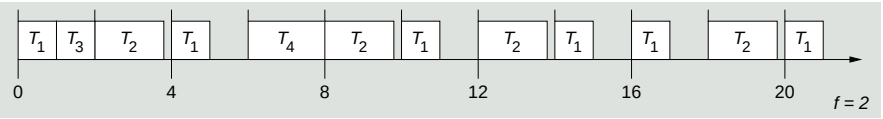
[2] Befindet sich f in diesem Bereich, gibt es wenigstens einen Rahmen zwischen der Auslösungszeit und dem Termin jedes Arbeitsauftrags der betreffenden Aufgabe.

Randbedingungen für die Rahmenlänge (Forts.)

$T_i = (p_i, e_i, D_i)$; gilt $D_i = p_i$, wird D_i nicht geschrieben

Beispiel S. 5-16: $T_1 = (4,1)$, $T_2 = (5,1.8)$, $T_3 = (20,1)$, $T_4 = (20,2)$

- $f \geq 2$ muss gelten, um jeden Job komplett durchlaufen zu lassen
- mögliche Rahmenlängen in H sind 2, 4, 5, 10 und 20 ($H = 20$)
- nur $f = 2$ erfüllt jedoch alle drei Bedingungen (S. 5-18) zugleich



Beispiel: $T_x = (15,1,14)$, $T_y = (20,2,26)$, $T_z = (22,3)$

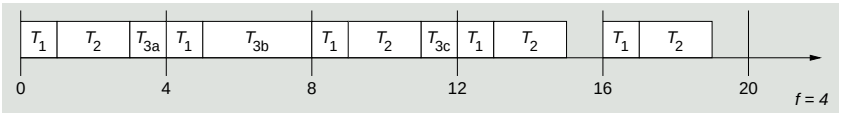
- $f \geq 3$ muss gelten, um jeden Job komplett durchlaufen zu lassen
- mögliche Rahmenlängen in H: 3, 4, 5, 10, 11, 15, 20, 22 ($H = 660$)
- jedoch nur $f = 3, 4$ oder 5 erfüllt alle drei Bedingungen (S. 5-18)

Konflikte und deren Auflösung

Taskparameter zugunsten einer guten Ablaufplananordnung korrigieren

Arbeitsaufträge sind in Scheiben zu schneiden, falls die Parameter der Aufgaben nicht alle Randbedinungen (S. 5-18) erfüllen können

- gegeben sei z.B. folgendes Tasksystem $\mathbf{T} = \{(4,1), (5,2,7), (20,5)\}$:
  - $f \geq \max(e_i)$ gilt für $f \geq 5$ und $2f - gcd(p_i, f) \leq D_i$ gilt für $f \leq 4$ **!?**
- $T_3 = (20,5)$ ist aufzuteilen in $T_3' = \{(20,1), (20,3), (20,1)\}$
  - d.h., in drei Teilaufgaben $T_{3a} = (20,1)$, $T_{3b} = (20,3)$, $T_{3c} = (20,1)$
  - das resultierende System hat fünf Tasks und die Rahmenlänge $f = 4$



- $T_3 = (20,5)$ in zwei Teilaufgaben aufzuteilen, bleibt erfolglos:
  - $\{(20,4), (20,1)\}$ geht nicht, wegen $T_1 = (4,1)$
  - $\{(20,3), (20,2)\}$ geht nicht, da für $T_{3b} = (20,2)$ kein Platz bleibt

Entstehungsprozess eines zyklischer Ablaufplans

Gegenseitige Abhängigkeit von Entwurfsentscheidungen

1. eine Rahmenlänge festlegen (S. 5-18)
   - durch Taskparameter ggf. gegebene Konflikte erkennen und auflösen
2. Arbeitsaufträge in Scheiben aufteilen (S. 5-21)
   - insbesondere kann dies zur Folge haben, andere Programm- bzw. Modulstrukturen herleiten zu müssen
   - die erforderlichen Programmtransformationen geschehen bestenfalls (semi-) automatisch durch spezielle Kompilatoren
   - schlimmstenfalls sind die Programme manuell nachzuarbeiten
3. die Arbeitsauftragsscheiben in die Rahmen platzieren

☞ Rahmenlängen sind querschneidende nicht-funktionale Eigenschaften

Aperiodische Arbeitsaufträge: Einplanung

Schlupf nutzen

Ausführung aperiodischer Jobs erfolgt im Hintergrund periodischer Jobs

- d.h., nachdem alle Jobs mit harten Terminen durchgelaufen sind
  - genauer: alle Jobscheiben mit harten Terminen in ihren Rahmen
  - zur Erinnerung: aperiodische Jobs haben weiche oder feste Termine
- jeder Schlupf auf der gesamten Echtzeitachse kann genutzt werden
  - Rahmen werden bei Bedarf „aufgefüllt“ mit aperiodischen Jobs
  - am Rahmenende wird ein unvollendeter aperiodischer Job verdrängt
  - der evtl. Jobrest füllt einen späteren Rahmen (mit) auf...
- die Einplanungsentscheidung wird zur Laufzeit getroffen (*online*)
  - aperiodische Jobs werden ereignisbedingt und damit zufällig ausgelöst

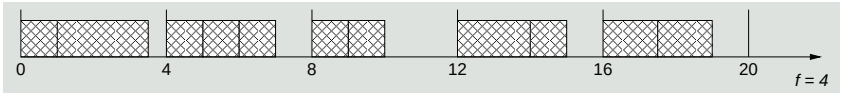
Folge: aperiodische Jobs werden zugunsten periodischer Jobs verzögert

- ihre Antwortzeit verschlechtert sich
- die Ansprechempfindlichkeit des Systems lässt nach

Aperiodische Arbeitsaufträge: Einplanung (Forts.)

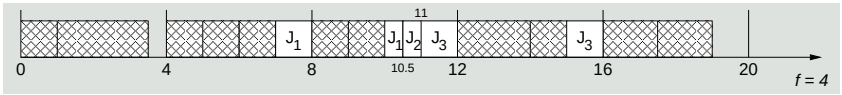
Beispiel: *Major Cycle* eines zyklischen Ablaufplans periodischer Jobs

- der erste große Durchlauf weist fünf Schlupfbereiche auf



  - schraffierte Bereiche bedeuten statisch eingeplante periodische Jobs

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



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

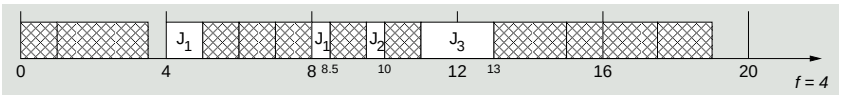
- mittlere Antwortzeit: $((10.5 - 4) + (11 - 9.5) + (16 - 10.5))/3 = 4.5$

Aperiodische Arbeitsaufträge: Antwortzeitverbesserung

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

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

- Ansatz ist, periodischen Jobs Zeitpuffer am Rahmenende entziehen
  - periodische Jobs werden ans Ende ihres Rahmens „geschoben“
  - vorne im Rahmen wird Platz für aperiodische Jobs geschaffen
- aperiodische Jobs J_1, J_2 und J_3, wie im Beispiel vorher (S. 5-24):



  J_1 wird sofort eingelastet, muss jedoch verdrängt werden
  J_2 wird ebenso behandelt, kann aber komplett durchlaufen
  J_3 wird verzögert bis der laufende periodische Job fertig ist

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

Aperiodische Arbeitsaufträge: Einlastung

Dispatcher aperiodischer Jobs als spezialisierten periodischen Job auffassen

Ausführung aperiodischer Jobs übernimmt ein Anbieter (engl. *server*) mit einem eigenen, autonomen Kontrollfluss $\rightsquigarrow$ (engl. *aperiodic server*)

- Anbieter wurde als erster Arbeitsauftrag eines Rahmens gestartet
  - für die Dauer der am Rahmenanfang verfügbaren Schlupfzeit
- er ruft die in einer Warteschlange stehenden aperiodischen Jobs auf

```
erledige aperiodischer Dispatcher (Warteschlange): /* Koroutine */
  solange der Betrieb läuft tue
    wenn die Warteschlange nicht leer ist dann
      erledige
        entnehme der Warteschlange einen Arbeitsauftrag;
        rufe Arbeitsauftrag.Routine auf;
      basta;
basta.
```


Aperiodische Arbeitsaufträge: Einlastung (Forts.)

Zusammenspiel zwischen periodischen und aperiodischen *Dispatcher*

Idee ist, dass der Anbieter periodischer Jobs (engl. *periodic server*) den „aperiodischen *Dispatcher*“ als Koroutine ausführt (S. 5-13):

```
erledige aktiviere (Arbeitsauftrag): /* aperiodischer Dispatcher */
  setze Arbeitsauftrag.Koroutine fort;
basta.
```

Zeitkontrolle sorgt für die Unterbrechung des laufenden aperiodischen Jobs, der dann im nächsten Schlupf fortgeführt wird $\rightsquigarrow$ Verdrängung

```
erledige Behandlungsroutine zum Timer Interrupt:
  wenn Arbeitsauftrag.Art = aperiodisch
    dann setze periodischen Dispatcher.Koroutine fort;
    sonst breche Arbeitsauftrag ab;
basta.
```

Sporadische Arbeitsaufträge

Zufällig ausgelöste Jobs mit harten Terminen

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

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

☞ „gleichzeitige“ sporadische Jobs werden oft nach EDF getestet

Rekonfiguration des Tasksystems

Änderung von Taskanzahl und -parameter

Umstellen auf einen neuen statischen Ablaufplan bedeutet mehr als nur einen Tabellenwechsel zu vollziehen:

1. Zerstörung und Erzeugung von periodischen Tasks
   - einige periodische Tasks werden aus dem System gelöscht, wenn ihre Funktion nicht mehr erforderlich ist ⇝ Betriebsmittelfreigabe
   - andere müssen dem System neu hinzugefügt werden, ggf. sind Programme nachzuladen ⇝ Betriebsmittelanforderung
   - manche Tasks überdaueren den Betriebswechsel, da sie im alten und neuen Tasksystem benötigt werden
2. Einlagerung und Aktivierung der neuen Ablauftabelle
   - neue Taskparameter und neuer Ablaufplan wurden *à priori* bestimmt

Betriebswechsel vom speziellen Arbeitsauftrag (engl. *mode-change job*) durchführen lassen ↦ nichtperiodischer Job

- ausgelöst durch ein (interaktives) Kommando zum Betriebswechsel
- verbunden mit einem weichen oder harten Termin

Arten von Betriebswechsel

Aperiodischer oder sporadischer Job

aperiodisch ↦ Betriebswechsel mit weichem Termin

- mit höchster Dringlichkeit ausgeführt als aperiodischer Job
  - der vor allen anderen aperiodischen Jobs zum Zuge kommt
- Zerstörung aperiodischer/sporadischer Jobs ist problematisch
  - die Ausführung aperiodischer Jobs wird hinausgezögert, bis der Betriebswechsel vollendet worden ist
  - im Falle sporadischer Jobs stehen zwei Optionen zur Verfügung:
    (a) der Betriebswechsel wird unterbrochen und später fortgesetzt
    (b) die Übernahmeprüfung berücksichtigt den neuen Ablaufplan
- Ziel ist es, die Antwortzeit für den Betriebswechsel zu minimieren

sporadisch ↦ Betriebswechsel mit hartem Termin

- die Anwendung muss die evtl. Abweisung des Jobs behandeln
  - sie wird den Betriebswechsel ggf. hinausschieben

Resümee

Ablauftabellen ↦ vorberechnete (statische) Ablaufpläne

- Tabelleneinträge sind Jobs und deren Einlastungszeitpunkte

Einlastung und Laufzeitkontrolle im Abfrage- oder Unterbrecherbetrieb

- Taktzähler, Zeitgeber, Zeitkontrolle; Job als Routine/Koroutine

Struktur zyklischer Ablaufpläne ⇝ „gute Anordnung“, Determinismus

- Rahmen, Rahmenlänge, Scheiben; *major/minor cycle*

nichtperiodische Arbeitsaufträge ↦ periodische/sporadische Jobs

- Schlupf (stehlen); Einplanung, Anwortzeitverbesserung, Einlastung
- Übernahmeprüfung (engl. *acceptance test*), Ausnahmebehandlung

Betriebswechsel bewerkstelligen aperiodische oder sporadische Jobs

- Tabellenwechsel, Betriebsmittelfreigabe/-anforderung, Nachladen

Literaturverzeichnis

[1] David Lorge Parnas.
Some hypotheses about the "Uses" hierarchy for operating systems.
Technical Report BS I 75/2, TH Darmstadt, 1975.

[2] Theodore P. Baker and Alan C. Shaw.
The cyclic executive model and Ada.
In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, pages 120–129, Huntsville, Alabama, USA, December 6–8, 1988.