Vorlesung Real-Time Systems

2. Real-Time Scheduling

Übersicht

1. Ziel diese Kapitels
2. Einführung
3. Rate-Monotonic Scheduling (RMS)
4. Analyse von RT-Scheduling-Methoden
5. Deadline-Monotonic Scheduling (DMS)
6. Response-Time Analysis (RTA)
7. Optimale Prioritätsvergabe mit Hilfe der RTA
8. Dynamisches Scheduling
9. Optimalitätsbegriff

1. Ziel dieses Kapitels

Scheduling ist die umfangreichste und wesentliche Technologie zur Umsetzung von Echtzeit-Problemen in einen konkreten Ablauf und somit das zentrale Mittel zur garantierten Einhaltung von Zeitschranken.

Diese Kapitel beschreibt alle elementaren

Scheduling-Algorithmen und -Analyse-Techniken

aus dem Bereich Echtzeitsystem.

Dafür werden zunächst alle notwendigen Begriffe und Modelle einführt.

2. Einführung

2.1 Definition „Task“

Eine **Task** ist ein

- sequentiell ablaufendes Programm,
- das ggf. mit anderen Task kommuniziert.

In einem Echtzeitsystem wird eine Task normalerweise mit einer bestimmten **Priorität** versehen. Der **Scheduler** sorgt dafür, dass immer die Task mit der höchsten Priorität die CPU bekommt.

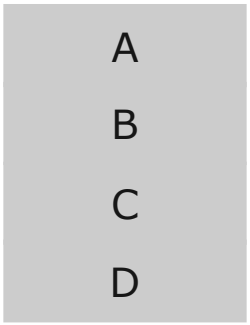
Es werden bewusst nicht die Betriebssystem-Begriffe Prozess oder Thread verwendet, weil die nachfolgende Theorie davon unabhängig ist. Eine Task kann also beides sein. Somit ist eine Task auch als elementares Stück Arbeit einer übergeordneten Problemlösung zu verstehen, deren Granularität variieren mag.

2. Einführung

Beim Mehrprogrammbetrieb mit 4 Programmen (A,B,C,D) entsteht der Eindruck, als wenn diese gleichzeitig abliefen. Tatsächlich bekommen aber die Tasks den Prozessor nach einer bestimmten Reihenfolge zugeteilt:

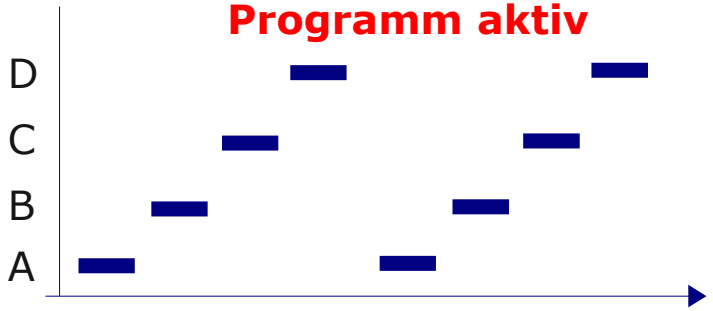
ein HW Programm-zähler

Arbeitsspeicher mit 4 Programmen



Konzeptionell
4 Programmzähler:
4 unabhängige Prozesse

real ist zu jedem Zeitpunkt nur **ein** sequentielles **Programm aktiv**



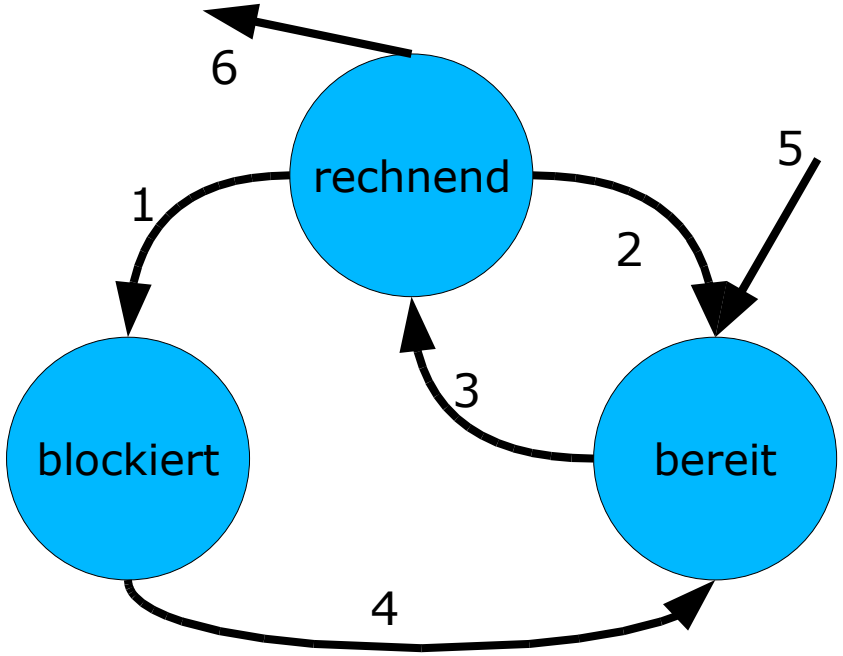
Ein **Scheduling-Algorithmus** entscheidet, wann die Arbeit einer Task unterbrochen und eine andere Task bedient wird.

2.2 Zustände einer Task

Tasks können sich in verschiedenen **Zuständen** befinden:

1. **rechnend** (running): der Prozessor ist der Task zugeteilt
2. **bereit** (ready) die Task ist ausführbar, aber eine andere Task ist gerade rechnend
3. **blockiert** (waiting): die Task kann nicht ausgeführt werden, da er auf ein externes Ereignis wartet (z.B. auf Eingabedaten).

Übergange:



1: Task wartet auf externes Ereignis
2: Scheduler wählt andere Task aus (Preemption)
3: Scheduler wählt diese Task aus
4: externes Ereignis ist eingetreten
5: eine neue Task wird erzeugt
6: die Task terminiert

2.3 Scheduling

Der Teil des Betriebssystems, der entscheidet, welche der Tasks, die im Zustand „bereit“ sind, ausgewählt wird und dann den Prozessor zugeteilt bekommt, heißt **Scheduler**. Das Verfahren, wie ausgewählt wird, nennt man **Scheduling-Algorithmus**.

Speziell für Echtzeitsysteme sind dabei die wichtigsten Kriterien:

- **Vorhersagbarkeit**
- Garantierte Einhaltung von Zeitschranken

Je nachdem, wie der Scheduler die Auswahl der Prozesse vornimmt, werden Scheduling-Algorithmen unterschieden in:

- **preemptive** Scheduling
  rechnende Prozesse können unterbrochen werden.
- **run to completion** (non-preemptive) Scheduling
  der rechnende Prozess wird nicht unterbrochen.

2. Einführung

In Echtzeitsystemen wird (fast) immer **prioritätsbasiertes Scheduling** betrieben. Dabei wird immer die rechenbereite Task mit der höchsten Priorität ausgewählt.

Man kann also klassifizieren in:

- preemptive und nicht-preemptive Verfahren (s.o.)
- (a) **statische** und (b) **dynamische** Verfahren:
  - (a) die einmal vergebene Priorität bleibt immer gleich
  - (b) die Priorität kann sich zur Laufzeit verändern
- (c) **online**- und (d) **offline**-Scheduling:
  - (c) das Scheduling wird zur Laufzeit durchgeführt und somit alle Entscheidungen spontan getroffen
  - (d) das Scheduling wird bereits vorher (pre-runtime) festgelegt und in eine Tabelle geschrieben.

Die Varianten (b) und (c) haben jeweils größeren Overhead, sind aber flexibler.

2.4 Scheduling-Modell

Im Folgenden werden einige RT-Scheduling-Verfahren vorgestellt und analysiert. In den seltensten Fällen wird nur eine Task allein den Prozessor benutzen. Deshalb ist es notwendig, das Scheduling so zu gestalten, dass potenziell alle vorhandenen Tasks rechtzeitig fertig werden.

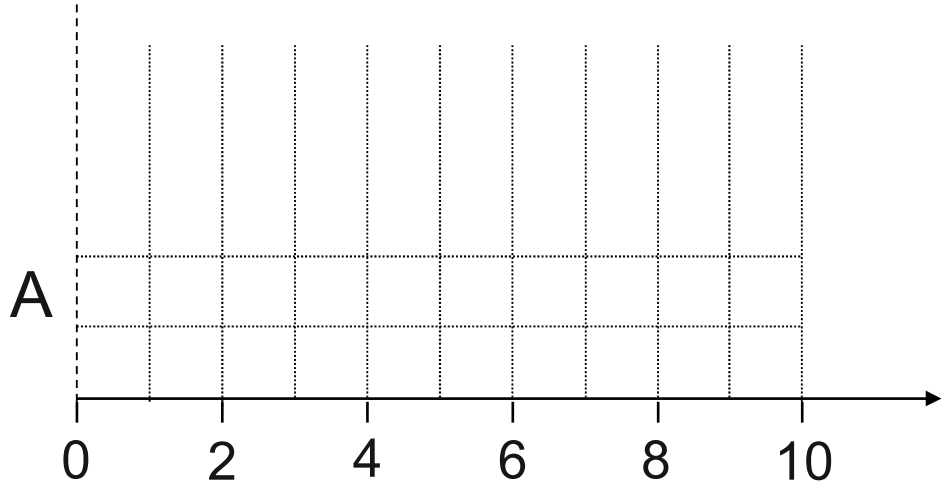
Alle Verfahren basieren auf der Kenntnis der voraussichtlichen maximalen Laufzeit einer Task (**Worst-Case Execution Time – WCET**). Diese zu bestimmen ist zwar schwierig aber möglich.

2. Einführung

Die wesentlichen Parameter des Schedulings (und dessen Analyse) umfassen somit:

- *period* T*:* Länge des Kreislaufs, z.B. $T_A=3$
- *computation time C:* längste zu erwartende Berechnungszeit, z.B. $C_A=1$
- *deadline D:* Zeitpunkt (relativ zum Beginn), bis zu dem die Berechnung jeweils erfolgt sein muss

Beispiel: Tragen Sie die Zeiträume ein, in denen Task A die CPU bekommt (es existieren keine weiteren Tasks):

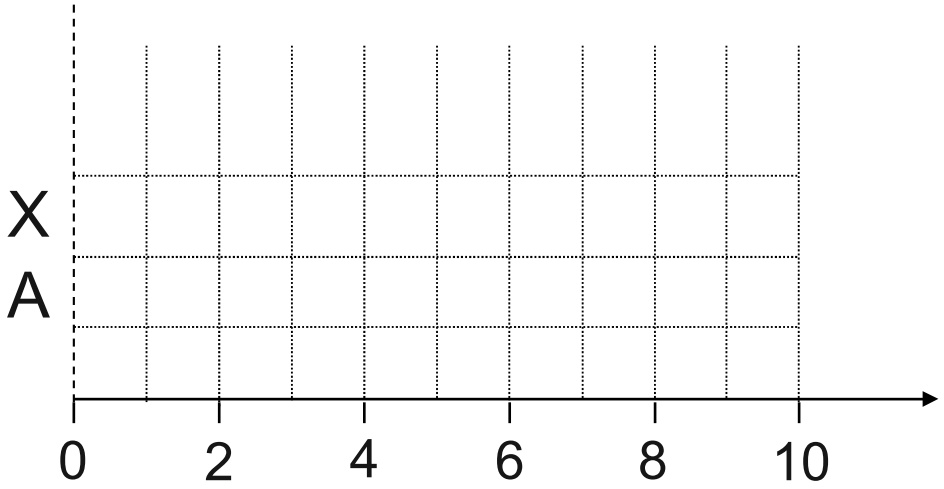


3. Rate-Monotonic Scheduling (RMS)

RMS ist das bekannteste (weil älteste) RT-Scheduling-Verfahren [Lui/Layland73]. Hierbei gilt:

- $D = T$; je kleiner D, desto höher die Priorität
- *Statisch:* die Priorität ändert sich nicht mehr
- *Preemptiv*: höher-priorisierte Tasks verdrängen niedrigere

Beispiel: Tragen Sie die Zeiträume ein, in denen die Tasks A und X jeweils die CPU bekommen (beide sind zum Zeitpunkt 0 bereit):



Task	T	C
A	3	1
X	10	3

4. Analyse von RT-Scheduling-Methoden

Um ganz sicher zu sein, dass niemals Deadlines überschritten werden, muss dies für den kompletten Zeitraum, in dem alle Task aktiv sein können, sicher gestellt werden. Dieser Zeitraum kann allerdings recht lang sein, denn er ergibt sich aus dem Kleinsten-gemeinsamen-Vielfachen (KgV) aller Perioden-Längen, beginnend mit dem **Critical Instant** (CI), d.h. 30 im letzten Beispiel – genannt: **Major Cycle**. Nachdem dieser abgearbeitet wurde, wiederholt sich das Verhalten des Systems.

Der CI repräsentiert den schlimmsten anzunehmenden Fall, nämlich dass alle Tasks gleichzeitig bereit sind. Wird diese Situation schadlos überstanden (d.h. ohne Deadline-Überschreitung), so kann auch jede andere Situation gemeistert werden.

Wird Offline-Scheduling betrieben, muss eine Tabelle für das komplette KgV erzeugt werden. Daher ist es oft günstiger, ein Online-Verfahren mit wenig Overhead (d.h. statisches Scheduling) einzusetzen.

4. Analyse von RT-Scheduling-Methoden

Bei der Analyse des RT-Schedulings geht es darum, zu bestimmen, ob alle Tasks unter dem gegebenen Scheduling-Verfahren ihre Deadlines einhalten werden. Je nach benutztem Verfahren existieren unterschiedliche Techniken, die i.A. ohne eine komplette Betrachtung des KgVs auskommen und somit effizient arbeiten. Solche Analyse-Techniken werden Schedulability-Tests genannt.
Sei ***n*** die Anzahl der Tasks. Ein Schedulability-Test für Rate-Monotonic Scheduling ist gegeben durch [Lui/Layland73]:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$$

$$U \leq 0.69 \quad \text{wenn} \quad n \to \infty$$

Der Test sagt also im Prinzip aus, dass man auf der sicheren Seite ist, wenn die Auslastung der CPU nicht 69% Prozent überschreitet.

4. Analyse von RT-Scheduling-Methoden

Tabelle:

# Tasks	U
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
∞	0.690

Beispiel: Wenden Sie den Lui-Layland Test auf folgendes Beispiel an und zeichnen Sie zur Kontrolle die aktiven Zeiträume über der Zeitachse ein:

Task	T	C
A	6	4
B	12	4

4. Analyse von RT-Scheduling-Methoden

Neben exakten Tests gibt es immer die Möglichkeit aus Gründen der Effizienz und zu Lasten der Genauigkeit eine Heuristik zu formulieren.

Diese Heuristik muss sich aber im Zweifel immer pessimistisch verhalten, um auszuschließen, dass zur Laufzeit eine Deadline verletzt wird. Solche Tests werden als *sufficient but not necessary* klassifiziert.

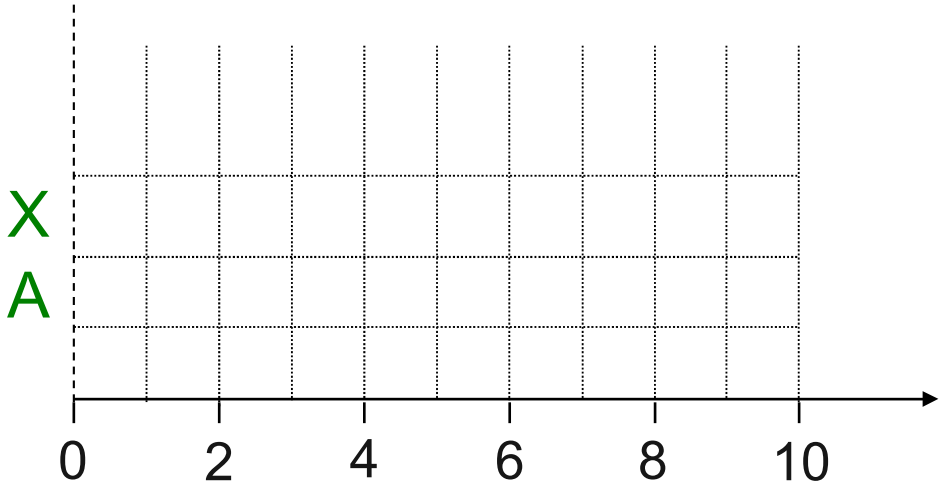
Im Folgenden werden weitere Scheduling-Verfahren und Tests diskutiert.

5. Deadline-Monotonic Scheduling (DMS)

DMS ist ähnlich zu aber etwas flexibler als Rate-Monotonic Scheduling.
Hierbei gilt:

- $D \leq T$; je kleiner D, desto höher die Priorität
- *Statisch:* die Priorität ändert sich nicht mehr
- *Preemptiv*: höher-priorisierte Tasks verdrängen niedrigere

Beispiel: Tragen Sie die Zeiträume ein, in denen die Tasks A und X jeweils die CPU bekommen (beide sind zum Zeitpunkt 0 bereit):

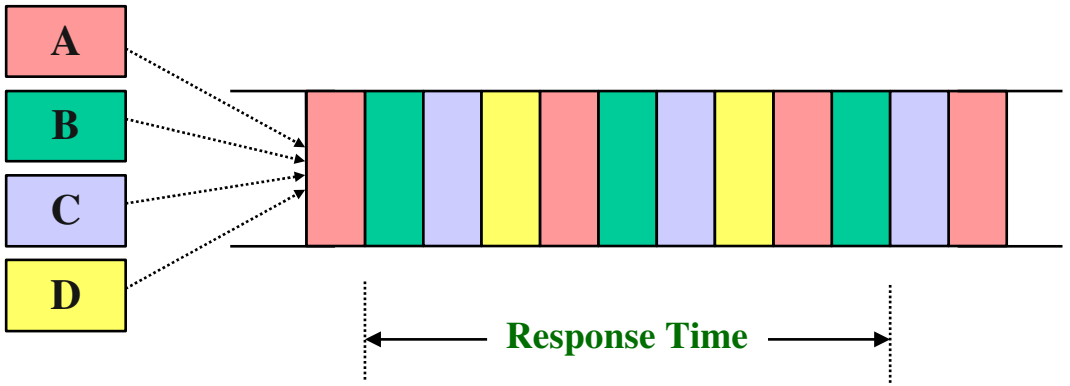


Task	T	C	D
A	4	1	4
X	10	3	3

Der Lui-Layland-Test ist hier nicht anwendbar.

6. Response-Time Analysis (RTA)

Anders als beim lastbasierten Test von Lui-Layland für RMS wird bei der RTA die maximale Antwortzeit jeder Task ermittelt und mit der dazu gehörigen Deadline verglichen:



Die RTA hat ihren Ursprung an den Englischen Universitäten in York und Warwick (1994). Die RTA kann für **alle preemptiven und statischen** RT-Scheduling-Techniken verwendet (also inklusive RMS und DMS) werden.

6. Response-Time Analysis (RTA)

6.1 Herleitung der Response-Time Analysis

- Response Time für Task ***i***: eigener Aufwand + Störungen durch höher-priorisierte Tasks

  $R_i = C_i$ + „*Störungen*“

- Störung durch eine höher-priorisierten Task ***j***:

  C_j

- Häufigkeit der Störung durch eine h.p. Task ***j***:

$$\left\lceil \frac{R_i}{T_j} \right\rceil$$

6. Response-Time Analysis (RTA)

- Insgesamt gilt:

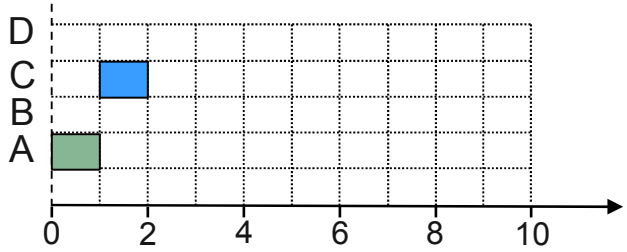
$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

  mit ***hp(i):*** Menge aller Tasks mit höherer Priorität als ***i***

- Die Rekurenz (***R_i*** steht auf beiden Seiten) wird auf gelöst durch:
  - Wahl eines Startwerts: ***$R_i^0 = C_i$***
  - Iteration bis ***R_i*** nicht mehr wächst

6.2 Beispiel RTA [2]

Schedule:



Task	T	C
A	3	1
B	6	1
C	5	1
D	10	2

Prioritätsvergabe nach RMS

Task A:

$$\text{hp(A)} = \{\} \Rightarrow R_A = C_A = 1$$

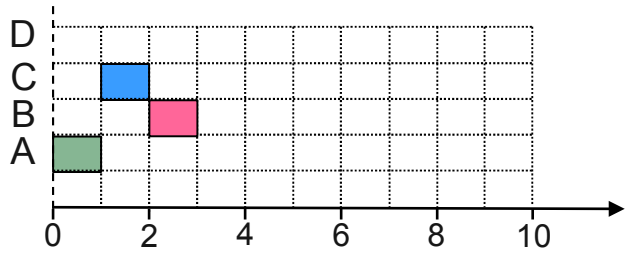
Task C:

$$\text{hp(C)} = \{A\} \Rightarrow R_C^1 = C_C + \lceil \frac{R_C^0}{T_A} \rceil C_A = 1 + \lceil \frac{1}{3} \rceil 1 = 1 + 1 = 2$$

$$R_C^2 = C_C + \lceil \frac{R_C^1}{T_A} \rceil C_A = 1 + \lceil \frac{2}{3} \rceil 1 = 1 + 1 = 2$$

6. Response-Time Analysis (RTA)

Schedule:



Task	T	C
A	3	1
B	6	1
C	5	1
D	10	2

Task B:

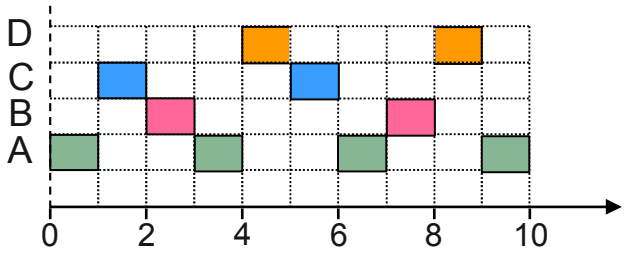
hp(B) = { A,C} $\Rightarrow$ $R_B^0 = C_B = 1$

$$R_B^1 = C_B + \lceil \frac{R_B^0}{T_A} \rceil C_A + \lceil \frac{R_B^0}{T_C} \rceil C_C = 1 + \lceil \frac{1}{3} \rceil 1 + \lceil \frac{1}{5} \rceil 1 = 1 + 1 + 1 = 3$$

$$R_B^2 = C_B + \lceil \frac{R_B^1}{T_A} \rceil C_A + \lceil \frac{R_B^1}{T_C} \rceil C_C = 1 + \lceil \frac{3}{3} \rceil 1 + \lceil \frac{3}{5} \rceil 1 = 1 + 1 + 1 = 3$$

6. Response-Time Analysis (RTA)

Schedule:



Task	T	C
A	3	1
B	6	1
C	5	1
D	10	2

Task D:

hp(D) = { A,B,C} $\Rightarrow \quad R_D^0 = C_D = 2$

$$R_D^1 = C_D + \lceil \frac{R_D^0}{T_A} \rceil C_A + \lceil \frac{R_D^0}{T_B} \rceil C_B + \lceil \frac{R_D^0}{T_C} \rceil C_C = 2 + \lceil \frac{2}{3} \rceil 1 + \lceil \frac{2}{6} \rceil 1 + \lceil \frac{2}{5} \rceil 1 = 2+1+1+1 = 5$$

$$R_D^2 = C_D + \lceil \frac{R_D^1}{T_A} \rceil C_A + \lceil \frac{R_D^1}{T_B} \rceil C_B + \lceil \frac{R_D^1}{T_C} \rceil C_C = 2 + \lceil \frac{5}{3} \rceil 1 + \lceil \frac{5}{6} \rceil 1 + \lceil \frac{5}{5} \rceil 1 = 2+2+1+1 = 6$$

$$R_D^3 = C_D + \lceil \frac{R_D^2}{T_A} \rceil C_A + \lceil \frac{R_D^2}{T_B} \rceil C_B + \lceil \frac{R_D^2}{T_C} \rceil C_C = 2 + \lceil \frac{6}{3} \rceil 1 + \lceil \frac{6}{6} \rceil 1 + \lceil \frac{6}{5} \rceil 1 = 2+2+1+2 = 7$$

$$R_D^4 = C_D + \lceil \frac{R_D^3}{T_A} \rceil C_A + \lceil \frac{R_D^3}{T_B} \rceil C_B + \lceil \frac{R_D^3}{T_C} \rceil C_C = 2 + \lceil \frac{7}{3} \rceil 1 + \lceil \frac{7}{6} \rceil 1 + \lceil \frac{7}{5} \rceil 1 = 2+3+2+2 = 9$$

$$R_D^5 = C_D + \lceil \frac{R_D^4}{T_A} \rceil C_A + \lceil \frac{R_D^4}{T_B} \rceil C_B + \lceil \frac{R_D^4}{T_C} \rceil C_C = 2 + \lceil \frac{9}{3} \rceil 1 + \lceil \frac{9}{6} \rceil 1 + \lceil \frac{9}{5} \rceil 1 = 2+3+2+2 = 9$$

6. Response-Time Analysis (RTA)

Beispiel: Wenden Sie die RTA auf folgendes Beispiel an und zeichnen Sie zur Kontrolle die aktiven Zeiträume über der Zeitachse ein:

Task	T	C	D	Prio
A	6	3	5	2 (hoch)
B	12	3	6	1 (niedrig)

Was passiert, wenn die Prioritäten vertauscht werden?

6. Response-Time Analysis (RTA)

Eigenschaften der RTA:

- Exaktes Verfahren ***(sufficient and necessary)***
- Im schlechtesten Fall wird das komplette KgV durchsucht. Im Allgemeinen ist dies jedoch nicht der Fall. Gewissermaßen wird ein „Fenster“ beginnend beim Critical Instant schrittweise vergrößert, bis es nicht mehr wächst.
- Die RTA stoppt wenn $R^n > D$, denn dann ist die Taskmenge nicht lauffähig.
- Die Prioritäten können beliebig vergeben werden, müssen aber statisch sein. Für die Deadlines gilt: $D \leq T$
- Die RTA lässt sich sehr gut erweitern.

6.3 Vereinfachte (RTA) für Deadline-Monotonic Scheduling

Ein *sufficient but not necessary* Tests für DMS ist gegeben durch [Audsley-etal91]:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

Verglichen mit der RTA verzichtet dieser Test auf die Iterationen und ist somit schneller. Es kann allerdings passieren, dass einzelne Task-Sets den Test nicht überstehen, obwohl sie eigentlich lauffähig (*feasible*) sind, weil die Response Time einzelner Tasks **überschätzt** wird.

6. Response-Time Analysis (RTA)

Beispiel: Wenden Sie den vereinfachten Test auf das Beispiel von eben an:

Task	T	C	D	Prio
A	6	3	5	
B	12	3	6	

7. Optimale Prioritätsvergabe mit Hilfe der RTA

Die RTA kann auch dafür benutzt werden, festzustellen, wie die Prioritäten so vergeben werden können, damit alle Tasks die Deadlines einhalten (optimale Prioritätsvergabe).

Dies folgt dem folgenden Theorem:

> Falls es überhaupt eine lauffähige Prioritätsvergabe für alle Tasks gibt, gilt: Wenn eine Task ***i*** die niedrigste Priorität zugewiesen bekommt und dabei seine Deadline einhält, dann gibt es eine Prioritätsvergabe mit Task ***i*** auf der niedrigsten Priorität.

Daraus kann folgender Algorithmus abgeleitet werden:

7. Optimale Prioritätsvergabe mit Hilfe der RTA

```
Assign_Pri (Set, N)
begin
    for K=1 to N            // für alle Tasks 1 bis N
        for Next=K to N
            Swap(Set, K, Next); // Tausch der Prioriät von K und Next
            Ok = RTA(K);
            exit when Ok;       // Schleife verlassen
        end for;
        exit when not Ok;     // keine Lösung gefunden
    end for;
end
```

- Die innere Schleife sucht eine Task aus der Teilmenge (K…N), die an der Stelle K ihre Deadline einhält (RTA (K) = OK). Diese RTA wird nur für Task K berechnet, denn Tasks 1…K-1 sind bereits okay. Die Reihenfolge der höher priorisierten Tasks (K+1…N) ist für die RTA (K) irrelevant.
- Sobald eine solche Task gefunden wurde, wird die innere Schleife abgebrochen, und K (in der äußeren Schleife) inkrementiert. Die Tasks 1…K-1 müssen dann nicht mehr betrachtet werden, da wir ja wissen, dass sie ihre Deadlines einhalten.

7. Optimale Prioritätsvergabe mit Hilfe der RTA

Beispiel: Wenden Sie den Prioritätsalgorithmus auf folgendes Beispiel an:

Task	T	C	D
A	3	1	3
B	6	1	6
C	5	1	5
D	10	2	10

8. Dynamisches Scheduling

Bislang wurden nur Scheduling-Algorithmen mit statischer Prioritätsvergabe betrachtet. Nun wenden wir uns dynamischen Verfahren zu, bei denen zur Laufzeit immer wieder neu die Prioritäten vergeben werden.

8.1 Earliest-Deadline First (EDF) Scheduling

Bei EDF hat (wie bei Deadline Monotonic) immer die Task mit der nächsten Deadline die höchste Priorität. Wir betrachten einen preemptiven Ansatz.

Ein Schedulability Test für EDF ist sehr einfach und effizient, sofern **T=D** gilt:

$$\sum_i \frac{C_i}{T_i} \leq 1.0$$

Für alle Tasks muss zusätzlich gelten: **C ≤ T**

Beispiel: Wenden Sie den EDF-Test auf folgendes Beispiel an und zeichnen Sie zur Kontrolle die aktiven Zeiträume über der Zeitachse ein:

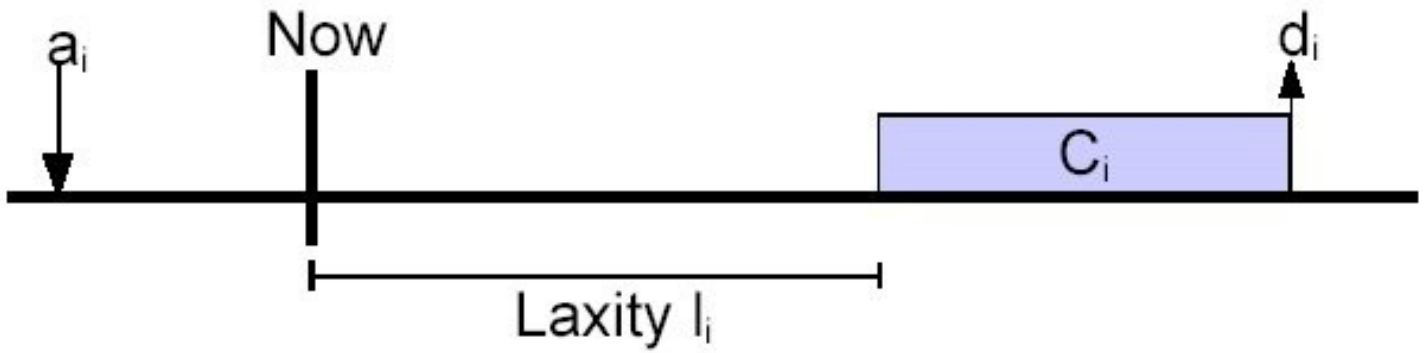
Task	T	C	D
A	3	1	3
B	6	1	6
C	5	1	5
D	10	2	10

Bewertung von Earliest-Deadline First:

- Der Test ist einfach und effizient und es ist eine hohe Prozessor-Auslastung möglich (bis 100%).
- Bei Überlast (d.h. eine oder mehrere Tasks überschreiten ihre Deadline) verhält sich EDF "gutmütiger" als RMS; d.h. es bekommen immer noch alle Tasks Rechenzeit.
- Allerdings ist das Verhalten bei Überlast kaum vorhersagbar; es entsteht ein "Domino-Effekt", d.h. immer mehr Tasks verpassen ihre Deadline
- Bei Online-Scheduling entsteht zur Laufzeit ein erheblich größerer und im Aufwand schwankender Overhead als beim statischen Verfahren. EDF wird daher selten von RTOS unterstützt.
- Erweiterungen hinsichtlich komplexerer Problemstellungen sind äußerst schwierig.

8.2 Least-Laxity First (LLF) Scheduling

LLF ist ebenfalls ein dynamischer Ansatz. Die Task mit dem jeweils kürzesten Abstand zwischen der eigenen Verarbeitungsdauer ***C*** und der Deadline ***D*** bekommt die höchste Priorität. Es wird also jeweils bestimmt, wie viel „Luft“ (**laxity**) eine Task noch hat.



Es gelten dieselben Annahmen und derselbe Schedulability Test wie für EDF (mit `C ≤ D ≤ T`). Ebenso hat LLF i.W. ähnliche Stärken und Schwächen.

Beispiel: Wenden Sie LLF Scheduling auf folgendes Beispiel an:

Task	T	C	D	Laxity
A	3	1	3	
B	6	1	6	
C	5	1	5	
D	10	2	10	

9. Optimalitätsbegriff

Ein RT-Scheduling-Algorithmus heißt ***optimal***, wenn der Algorithmus zu einer gegebenen Menge an Tasks einen brauchbaren Schedule (d.h. alle Tasks erfüllen ihren Deadlines) erzeugt, sofern dieser existiert.

RMS, DMS, LLF und EDF sind optimal (unter *einfachen* Randbedingungen).

9. Optimalitätsbegriff

Beispiel: RMS ist eine optimale Prioritätenverteilung.

Das bedeutet: *Wenn eine Anwendung mit irgendeiner festen Prioritätenverteilung ausführbar ist, dann ist sie es auch unter Verwendung von RMS.*

Beweis:
Gegeben seien 2 Tasks mit den Perioden T_1, T_2, mit $T_1 < T_2$, und den Ausführungszeiten C_1, C_2.

Dann gibt es 2 Möglichkeiten für die Prioritätenverteilung:

Variante 1:

- Task 2 erhält höhere Priorität (entgegen RMS).
- Das System ist genau dann ausführbar, wenn gilt: $C_1 + C_2 \leq T_1$ da sonst zum Critical Instant Task 1 ihre Deadline (= T_1) nicht einhalten könnte.

9. Optimalitätsbegriff

Variante 2:

Task 1 erhält höhere Priorität (gemäß RMS). Wir betrachten wieder den Critical Instant.

Wenn das System unter Variante 1 ausführbar ist, d.h. es gilt $C_1 + C_2 \leq T_1$, dann ist es unter Variante 2 ebenfalls ausführbar, denn die erste Ausführung von Task 1 und Task 2 – egal in welcher Reihenfolge – terminiert spätestens zum Zeitpunkt T_1.

$\Rightarrow$ Task 1 hält ihre Deadline (= T_1)

Da $T_2 > T_1 \Rightarrow$ Task 2 hält ihre Deadline (= T_2).

Die Beweisführung lässt sich leicht auf eine beliebige Anzahl Tasks erweitern. (Sortiere die Tasks nach Prioritäten und betrachte jeweils 2 "benachbarte" Tasks...)