

# Contents

<b>1</b>	<b>placeholder</b>	<b>2</b>
<b>2</b>	<b>Zentrale Beschreibegrößen</b>	<b>2</b>
2.1	placeholder . . . . .	2
2.1.1	Technischer Prozess . . . . .	2
2.1.2	Rechenprozesse . . . . .	3
2.1.3	System-software . . . . .	3
2.2	Realzeitbedingungen . . . . .	3
2.2.1	Auslastungsbedingung . . . . .	3
2.2.2	Rechtzeitigkeitsbedingung . . . . .	4
2.2.3	Harte und weiche Realzeit . . . . .	5
2.3	Systemaspekte . . . . .	5
2.3.1	Unterbrechbarkeit . . . . .	5
2.3.2	Prioritäten . . . . .	6
2.3.3	Ressourcenmanagment . . . . .	7
<b>3</b>	<b>System-software</b>	<b>7</b>
3.1	Firmware . . . . .	7
3.2	RT-OS . . . . .	7
3.2.1	Systemcall-Interface . . . . .	8
3.2.2	Taskmanagment . . . . .	9
3.2.3	Scheduling . . . . .	13
3.2.4	Singlecore Scheduling . . . . .	14
3.2.5	Multicore Scheduling . . . . .	17
3.2.6	Memory Managment . . . . .	19
3.2.7	I/O Managment . . . . .	20
<b>4</b>	<b>placeholder4</b>	<b>23</b>
<b>5</b>	<b>placeholder5</b>	<b>23</b>
<b>6</b>	<b>Realzeitnachweis</b>	<b>23</b>
6.1	Zentrale Beschreibungsgrößen (Wiederholung) . . . . .	23

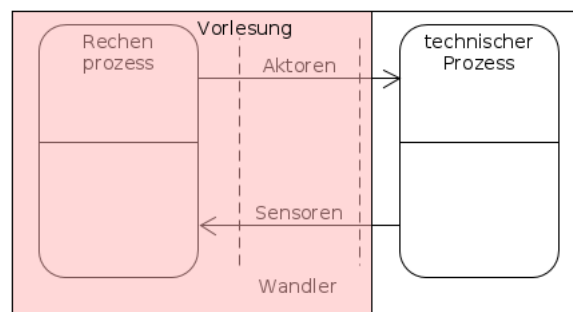
# 1 placeholder

## 2 Zentrale Beschreibegrößen

### 2.1 placeholder

**Defintion:** Realzeitsystem haben neben funktionalen Anforderungen auch **zeitliche** Anforderungen.

Ein Realzeitsystem besteht softwaretechnisch aus einer Reihe von Tasks und aus der System-Software.

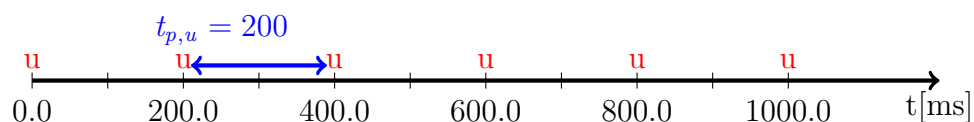


#### 2.1.1 Technischer Prozess

Rechenzeitanforderung = Ereignis von technischen Prozess Releasetime = Zeitpunkt des Auftretens der RZ-Anforderung (RZ/RT = Realzeit)

**Beispiel:** periodisches Signal **u** alle 200ms

$t_{Release,u,1}$	=	0ms	$t_{Release,u,2}$	=	200ms
$t_{Release,u,3}$	=	400ms	$t_{Release,u,4}$	=	600ms



**Prozesszeit** = zeitlicher Abstand zwischen zwei **RZ-Anforderungen** gleichen Typs.

$$t_{Pmin,i} = \text{minimal} \Rightarrow t_{max,i} = \frac{1}{t_{Pmin,i}}$$

$$t_{Pmax,i} = \text{maximal} \leq \text{uninteressant}$$

$t_{Dmin,i}$  = minimal zulässige Reaktionszeit

$t_{Dmax,i}$  = maximal zulässige Reaktionszeit

Airbag:

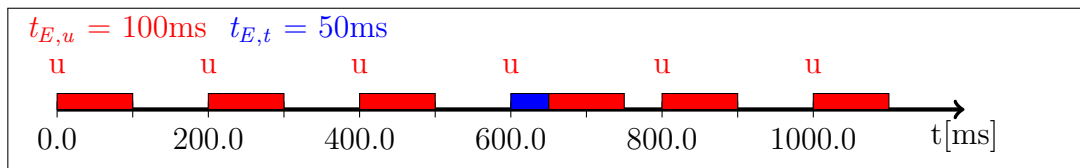
$t_{Dmax} = 50\text{ms}$ (Zeit bis zum Aufschlag) -  $30\text{ms}$ (Zeit zum aufblasen) =  $20\text{ms}$

$t_{Dmin} = 0\text{ms}$

Phase = minimal Zeitlicher Abstand zwischen zwei **unterschiedlicher** RZ-Anforderungen  $t_{Ph,i,j}$

### 2.1.2 Rechenprozesse

- Ausführungszeit (Executiontime) = Rechenzeit für eine RZ-Anforderung (ohne Warte oder Schlafzeiten)
- WCET  $t_{Emax,i}$  -> Erfahrung oder Messen Worstcase
- BCET  $t_{Emin,i} = 0$  Bestcase



- Reaktionszeit  $t_{R,i}$  = Zeit zwischen dem Auftreten der RZ-Anforderungen **i** und dem Ende der Bearbeitung.

$T_{Rmax,i}$  = maximale Reaktionszeit

$T_{Rmin,i}$  = minimale Reaktionszeit

$T_{R,i} = t_{W,i} + t_{E,i}$  wobei  $t_{W,i}$  Summe aller Wartezeiten

### 2.1.3 System-software

- Latenzzeit  $t_{L,i}$  = Zeit zwischen dem Auftreten einer RZ-Anforderung und dem Start der Bearbeitung - **Interrupt Latenzzeit** - **Tasklatenzzeit**

## 2.2 Realzeitbedingungen

### 2.2.1 Auslastungsbedingung

$\rho_i = \frac{t_{E,i}}{t_{P,i}}$  Auslastung der RZ-Anforderung i

$\rho_{max,i} = \frac{t_{Emax,i}}{t_{Pmin,i}}$  Worstcase, max. Auslastung

### 1. RT Bedingung

$$\rho_{max,ges} = \sum_{j=1}^n \frac{t_{Emax,j}}{t_{Pmin,j}} \leq c$$

j = für alle RZ-Anforderungen, c = Anzahl der Rechnerkerne

Beispiel: 2 RZ-Anforderungen A und B

$$\left. \begin{array}{l} t_{Pmin,A} = 2ms \\ t_{Emax,A} = 0.8ms \end{array} \right\} \rho_{max,A} = \frac{0.8ms}{2ms} = 0.4ms$$

$$\left. \begin{array}{l} t_{Pmin,B} = 1ms \\ t_{Emax,B} = 0.3ms \end{array} \right\} \rho_{max,B} = \frac{0.3ms}{1ms} = 0.3ms$$

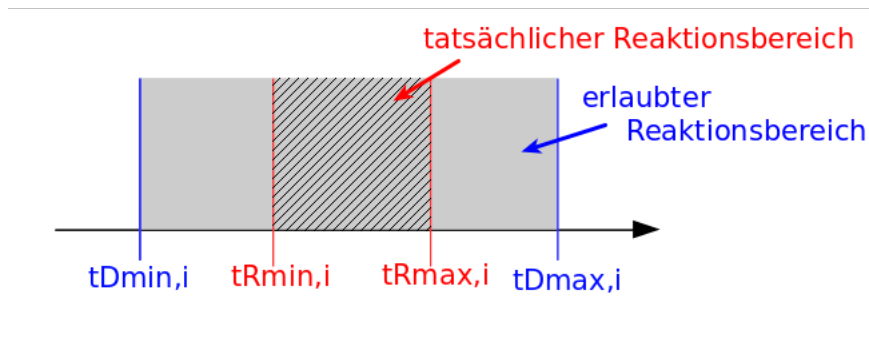
$$\rho_{max,ges} = \rho_{max,A} + \rho_{max,B} = 0.7 = 70\%$$

Annahme Singlecore c = 1  $\rho_{max,ges} \leq c \Rightarrow 0.7 \leq 1$

Auslastungsbedingung erfüllt

## 2.2.2 Rechtzeitigkeitsbedingung

Für den Realzeitbetrieb muss die tatsächliche Reaktion innerhalb des Zulässigen Reaktionsbereiches erfolgt sein.

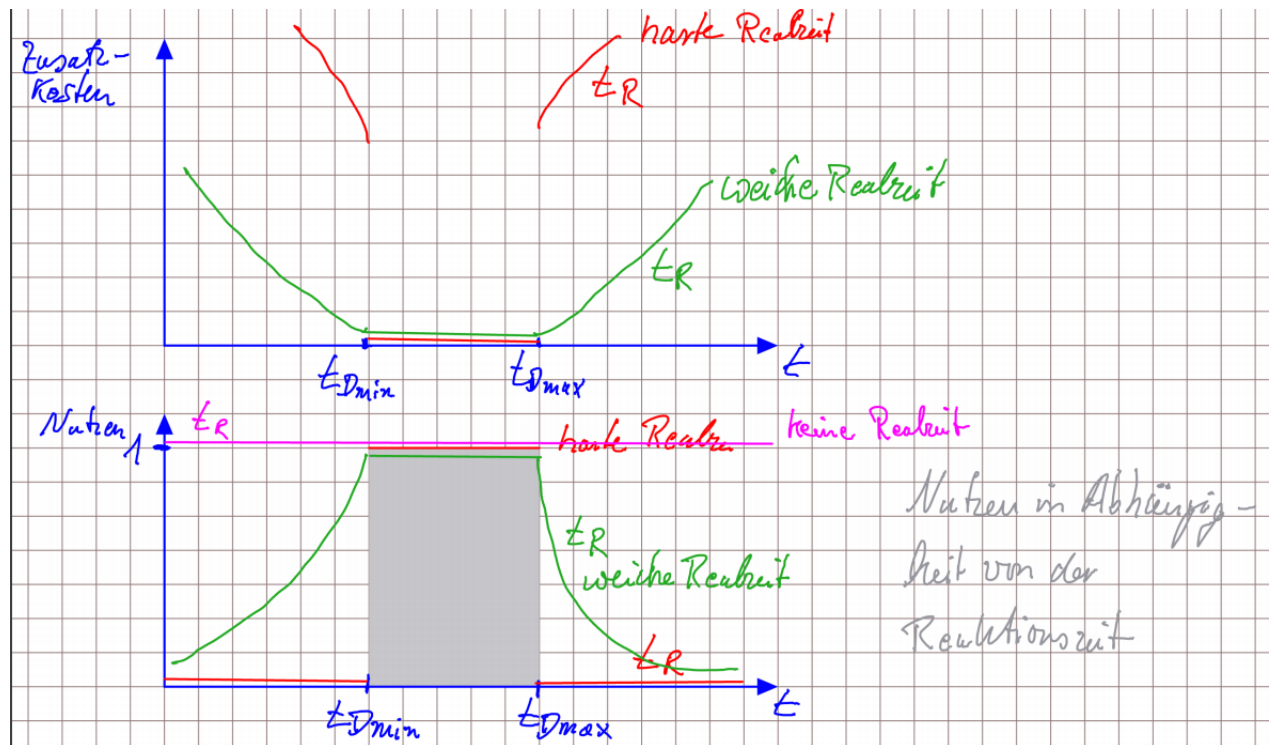


### 2. RT Bedingung

Für alle RZ-Anforderungen j muss gelten:

$$t_{Dmin,j} \leq t_{Rmin,j} \leq t_{Rmax,j} \leq t_{Dmax,j}$$

### 2.2.3 Harte und weiche Realzeit



## 2.3 Systemaspekte

### 2.3.1 Unterbrechbarkeit

**Forderung:** Codesequenzen lassen sich in Teilsequenzen unterteilen, die in korrekter Reihenfolge aber unabhängig voneinander abgearbeitet werden können.

=> notwendig für den Realzeitbetrieb

**Begründung:** Ein Messwert soll kontinuierlich erfasst werden.

$$t_{Emin,u} = t_{Emax,E} = 0.5ms$$

Jeweils 100 Messwerte (alle 100ms) sollen weiterverarbeitet werden

$$t_{Pmin,w} = 100ms$$

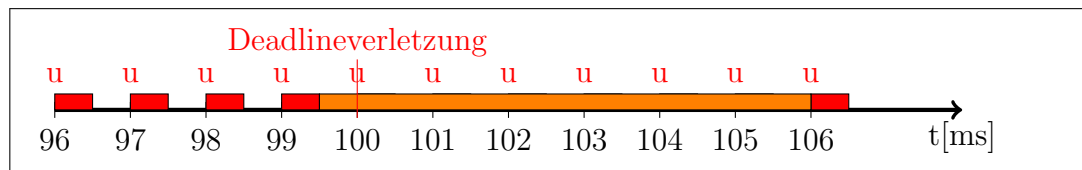
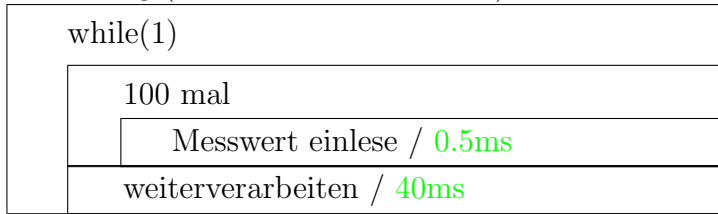
$$t_{Dmin,w} = 0ms$$

$$t_{Dmax,w} = 100ms$$

$$t_{Dmax,w} = 100ms$$

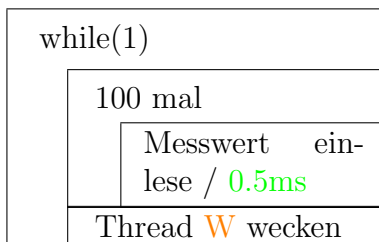
$$t_{Emin,w} = t_{Emax,w} = 40ms$$

Lösung (ohne Unterbrechbarkeit):

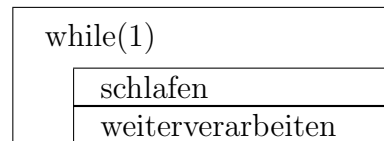


$$\rho_{max,u} = \frac{0.5ms}{1ms} = 0.5; \rho_{max} = \frac{40ms}{100ms} = 0.4 \Rightarrow \rho_{max,ges} = 0.9$$

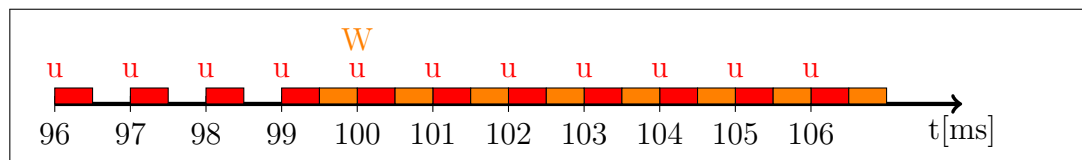
Lösung mit Unterbrechbarkeit:



Thread U



Thread W



### Konsequenzen:

- a) Inter-prozess-Kommunikation (IPC) (Sync, Datenaustausch)
- b) Multithreading/Multitasking

### 2.3.2 Prioritäten

**Forderung:** Der Systemarchitekt muss einfluss auf die Abarbeitungsreihenfolge mehrerer Tasks nehmen können z.B. über Prioritäten.

### 2.3.3 Ressourcenmanagement

→ später

## 3 System-software

### 3.1 Firmware

**Aufgabe:**

- Basisinitialisierung der Hardware
- Diagnose
- Betriebinitialisierung
- Laden + Aktivieren von Codes
- Runtime Services

**Ausprägungen:**

- BIOS
- UEFI
- Bootloader ("Das U-Boot")
- Monitor Software

### 3.2 RT-OS

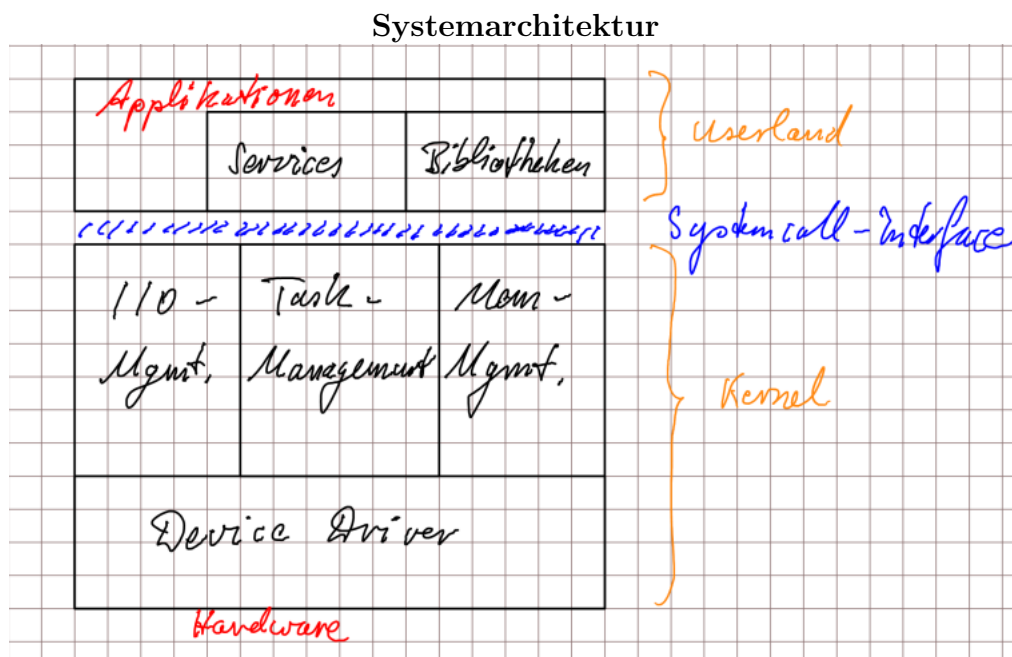
**Definition.:** Bezeichnung für alle Software-Komponenten, die

- die Ausführung der Applikationen und
- die Verteilung der Betriebsmittel (Memory, Files, CPU, Drucker, ...) ermöglichen, steuern und überwachen.

### Anforderungen:

- Zeitverhalten
- Ressourcenverbrauch
- Zuverlässigkeit und Stabilität
- Sicherheit
- Flexibilität und Kompatibilität
- Portierbarkeit
- Skalierbarkeit

**Beispiele:** Sämtliche Betriebssysteme



### 3.2.1 Systemcall-Interface

Systemcall = Dienst des Kernels -> 300-400 Dienste

**Beispiele:** `open()`, `close()`, `read()`, `write()`, `exit()`, `fork()`, `clone()`, `clock_nanosleep()`, `kill()`, `adjtime()`,...

Technische Realisierung: SW-Interrupt



**Ablauf:** `ret = write(fd, "Hello World", 13);`

↓ Systemcall "write" per SW-Interrupt

"int 0x80", "trap", "sysenter"

Systemcall mit `EAX = 4` ← x86 Register

ISR (SW-Interrupt 0x80)

↓ `EAX = 4` → bedeutet write

`vfs.write()`

↓ wertet die übrigen CPU-Register aus

↓ `fd` → entscheidet über den zu nutzenden Gerätetreiber

`driver.write()` → gibt Hardwaretechnisch die Daten aus

### 3.2.2 Taskmanagment

**Aufgabe:** Verwaltung der Ressource CPU

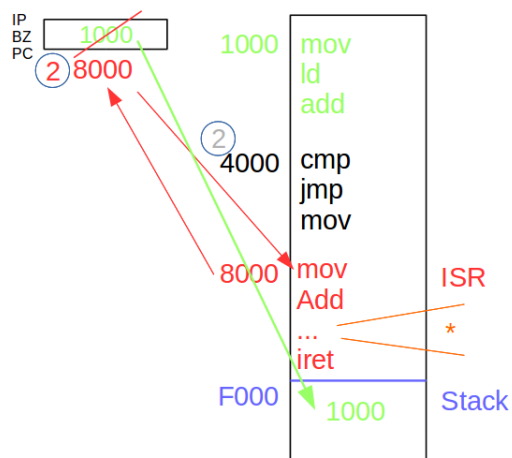
→ quasi parallele Verarbeitung auf einzelnen CPU-Kernen

→ real parallele Verarbeitung auf Multicore-Rechnern

Scheduling = Auswahl des als nächsten zu bearbeiten Jobs

Content Switch = Aktivierung eines Jobs

**Singelcore-Scheduling** Realisierung: Modifikation der Rücksprungadresse auf dem Stack beim Interrupt.



1. Code an der im IP stehenden Address wird abgearbeitete

2. IR tritt auf

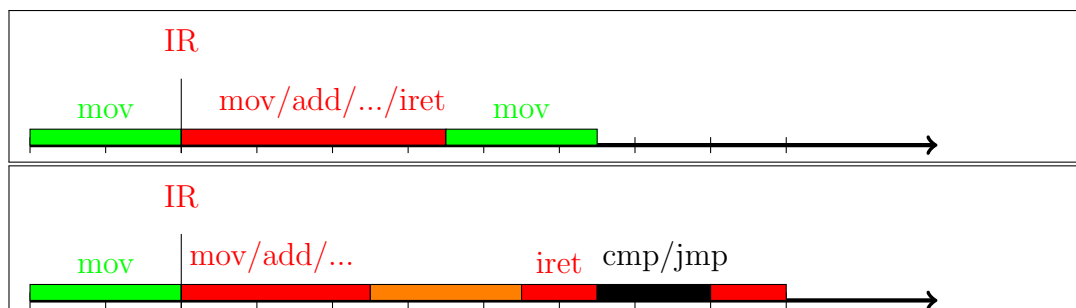
- Inhalt vom IP wird auf den Stack gelegt

- IP wird auf die Adresse der ISR gelegt (CPU arbeitet die ISR ab)

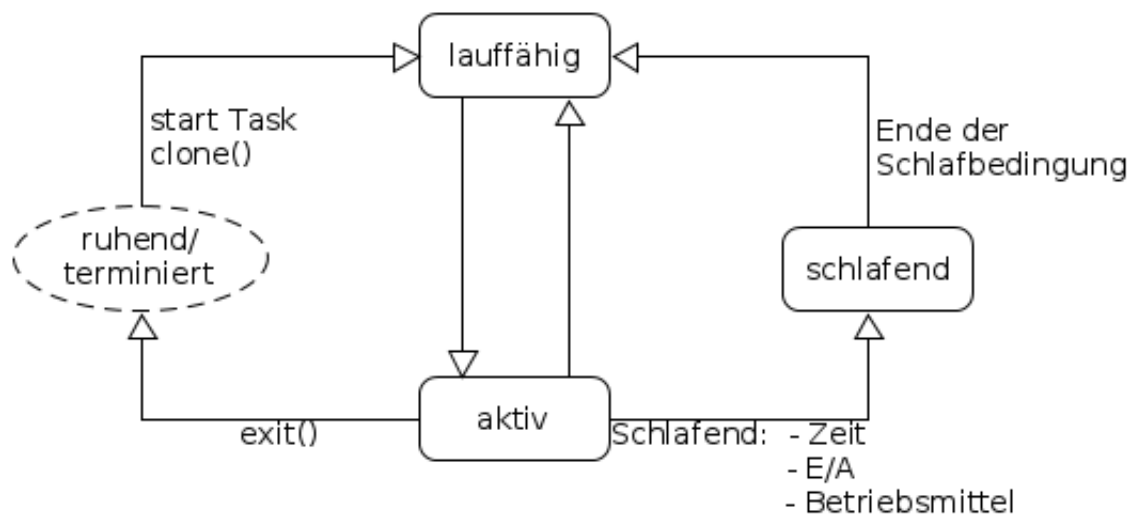
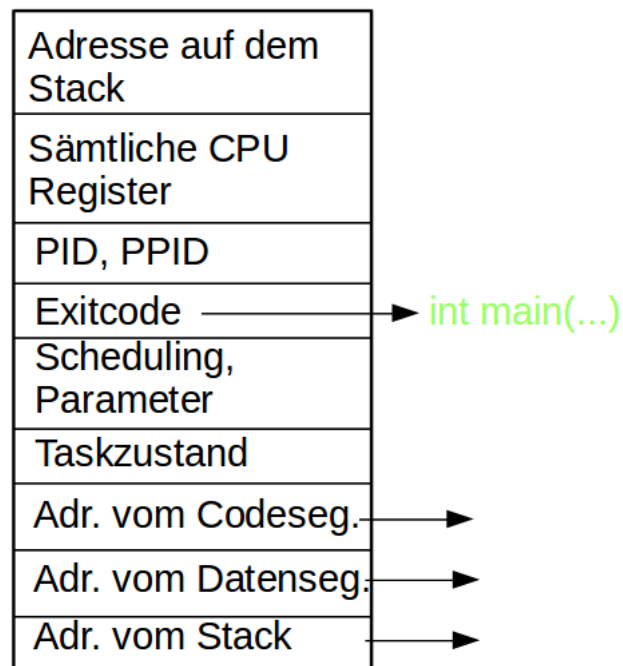
3. Bei ired wird die auf dem stack hinterlegte Adresse zurück auf den IP geladen

-> normale Verarbeitung wird fortgesetzt

\* zusätzlicher Code der die auf dem Stack liegende Adresse ändert  
z.B. die 1000 wird mit 4000 überschrieben



Eine Datenstruktur wird benötigt, um alle Informationen zu einer Code-sequenzen speichern(Job, Rechenprozess): [Task Control Block \(TCB\)](#)

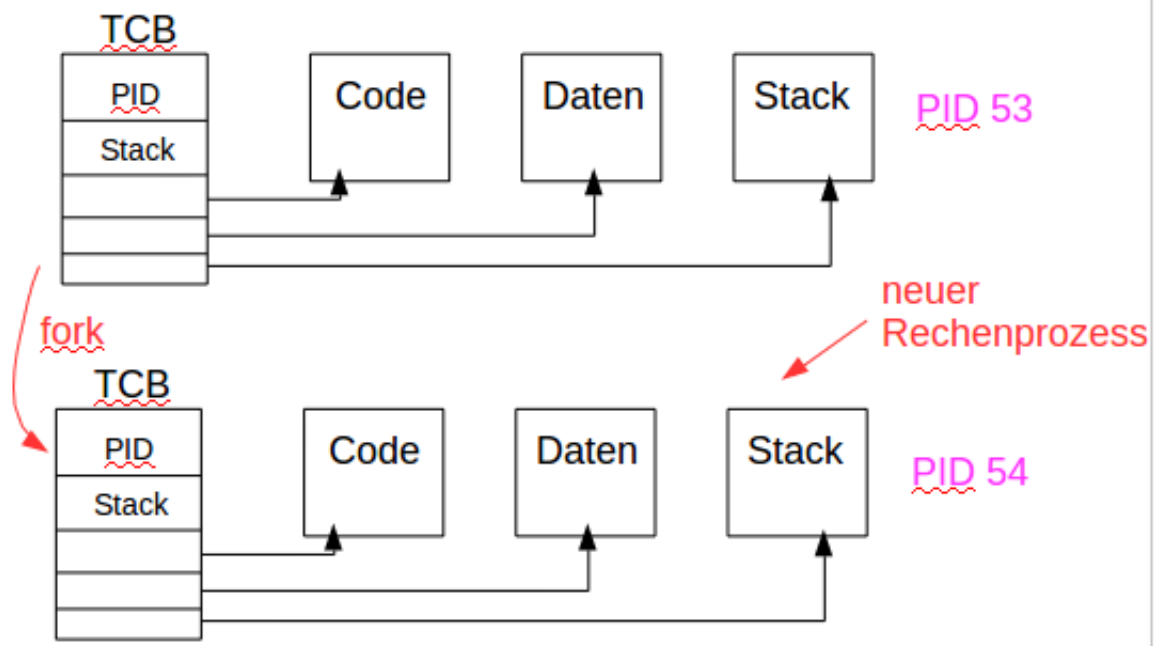


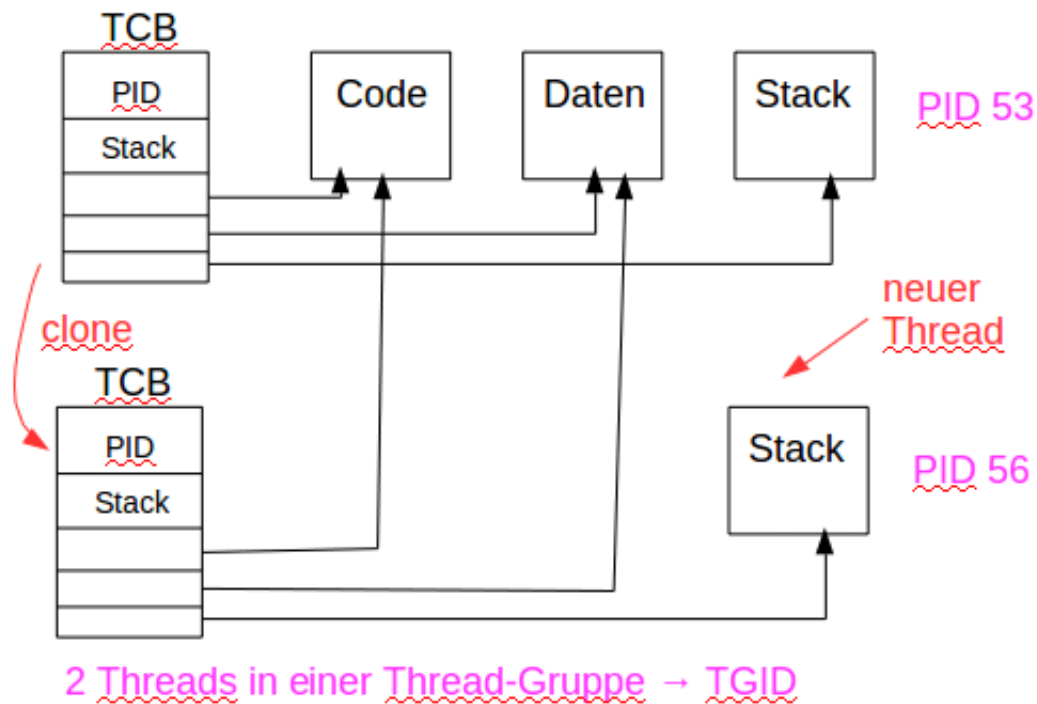
**Erzeugen von Rechenprozessen** Neue Jobs werden durch Kopieren von

- TCB

- Stack-Segment
- Code-Segment
- Daten Segment
- Neue PID vergeben

erzeugt





Threads einer Threadgruppe teilen sich Code- und Datensegment Vorteil:

- Speicherplatzersparnis
- schnell erzeugt
- einfache IPC (Inter Process Communication)

Nachteil:

- Safety: Ein amok laufender Thread bringt die gesamte Threadgruppe in einen inkonsistenten Zustand.

### 3.2.3 Scheduling

**Def.:** Auswahl der Task, der arbeiten/rechnen darf

Content Switch: Aktivierung der ausgewählten Task

Statisches Scheduling: Bedarfsituation ist bekannt

↳ Reihenfolge (Plan) kann im vorhinein festgelegt werden (SPS)

Dynamisches Scheduling: Die Auswahl erfolgt auf Basis der aktuellen Bedarfsituation (z.B. PC, Smartphone)

Singlecore Scheduling: Quasiparallele Verarbeitung auf einem CPU-Kern

Multicore Scheduling: Realparallele Verarbeitung auf mehreren CPU-Kernen

↳ Verteilungsaufgabe Preemption Point: Auftreten einer RF-Anforderung -> Interrup Service Routine(ISR) wird aktiv -> Scheduling

### 3.2.4 Singlecore Scheduling

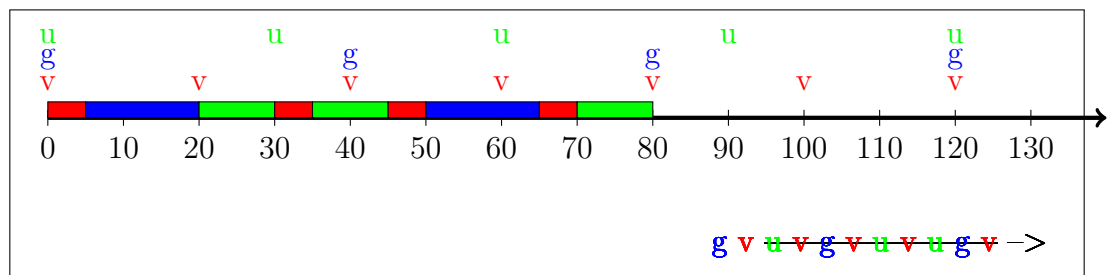
Table 1: Beispieldaten:

RZ-Anf	$t_{Pmin}$	$t_{Dmin}$	$t_{Dmax}$	$t_{Emin}$	$t_{Emax}$
v	20ms	0ms	20ms	1ms	5ms
g	40ms	0ms	40ms	1ms	15ms
u	30ms	0ms	30ms	1ms	10ms

First come First Serve (FCFS, FIFO)

- Lauffähige Jobs werden gemäß Auftrittszeitpunkt in eine Queue eingeführt
- Der erste Job in der Liste wird ausgewählt
- Er darf solange die CPU benutzen (rechnen) bis
  - a) er sich schlafen legt
  - b) er sich beendet

Ein "aufgeweckter" Job wird an den Anfang der Queue gestellt und unterbricht den laufenden Job nicht.

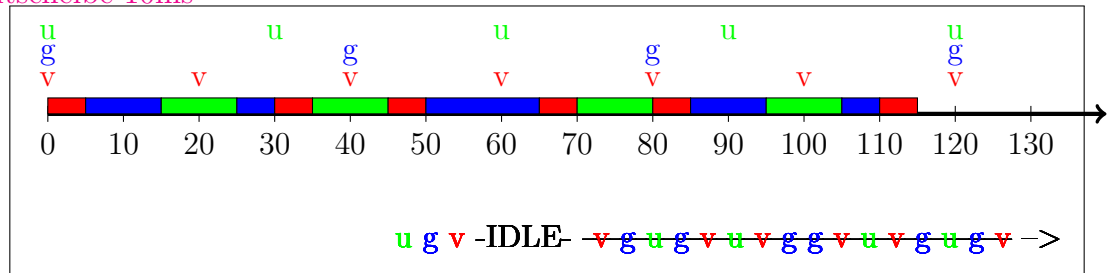


Zeitscheibenverfahren (Round Robin, Rate Monotonic)

- Lauffähige Jobs werden gemäß Auftrittszeitpunkt in eine Queue eingehängt
- Der erste Job in der Liste wird ausgewählt
- Er darf die CPU benutzen bis
  - a) er sich schlafen legt

- b) er sich beendet
- c) seine Zeitscheibe (Quantum) aufgebraucht ist
- Ein aufgeweckter Job wird ans Ende der Queue angehängt

Zeitscheibe 10ms



Prioritätengesteuertes Scheduling

- Jedem Job wird eine Priorität zugewiesen
- Der **Lauffähige** Job mit der höchsten Priorität wird ausgewählt
- Er darf rechnen bis
  - a) er sich schlafen legt
  - b) er sich beendet
  - c) ein Job mit höherer Prio Lauffähig wird

Problem: Verteilung der Prioritäten

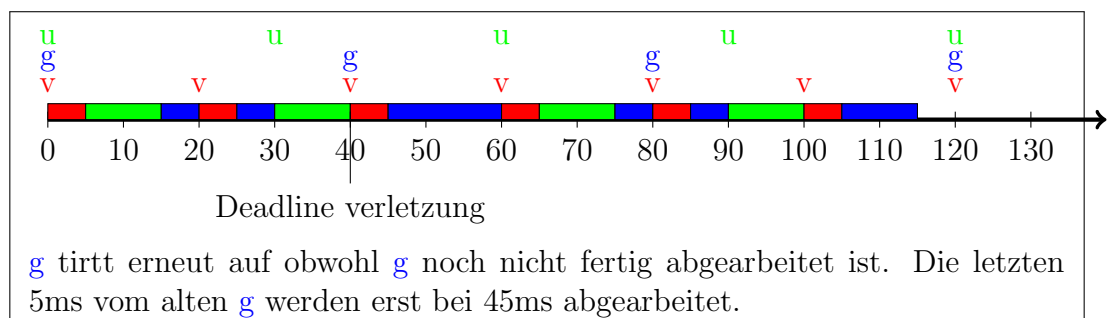
kurze  $t_{E_{max}}$  und kurze  $t_{P_{min}}$  -> hohe Prio

Gibt es zwischen  $t_{E,i}$  und  $t_{P,i}$  keine Korrelation hilft bei der Prioritätenverteilung nur ausprobieren!

**Beispiel:** V = 1 (höchste Prio)

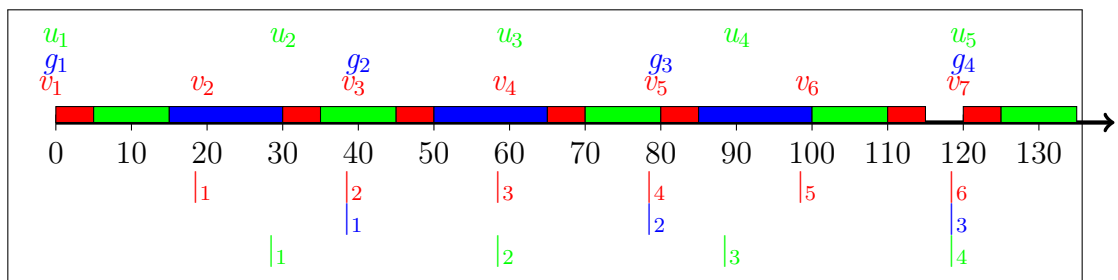
G = 3 (niedrigste Prio)

U = 2 (mittlere Prio)



### Deadline-Scheduling (Earliest Deadline First)

- Der Lauffähige Job, der als erstes fertig sein muss ( $t_{Dmax}$ ) darf rechnen.
- Er darf arbeiten bis
  - a) er sich schlafen legt
  - b) er sich beendet
  - c) ein Job Lauffähig wird, der früher abgeschlossen sein muss.



Eignung für RT-Systeme: **optimales Verfahren**

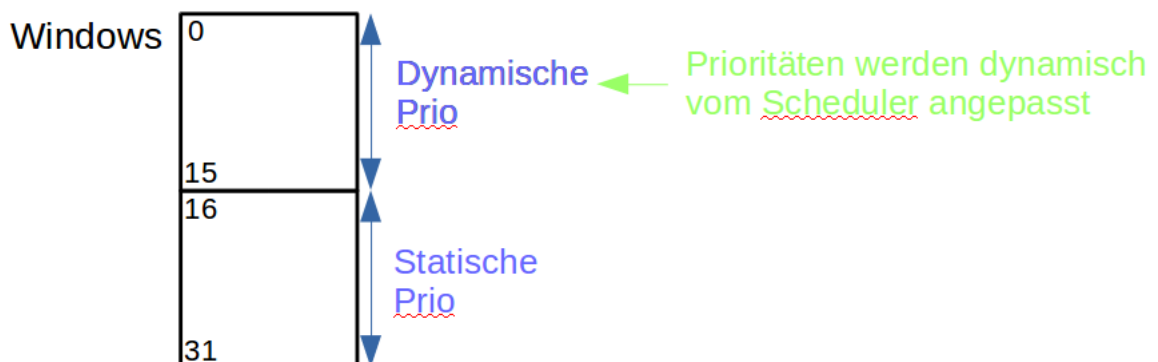
### Kombinierte Schedulingverfahren

- Prioritätengesteuertes Scheduling → Prioritätsebene
- Jobs mit gleiche Priorität (=auf gleicher Prioritätsebene)

werden gemaess  $\left. \begin{array}{l} a) RoundRobin \\ b) FCFS(FIFO) \end{array} \right\}$  Posix-Scheduling

gescheduled => Linux,

*VxWorks, Windows*  
Round Robin





**Linux:** Prioritäten per Konsole vergeben: `chrt 69 ./carrera //` werte von 0..99

Konzept der Scheduling Klassen:

- 0 Stop-Sched-Class
- 1 Prioritäten gesteuertes Scheduling
- 2 EDF (Earliest Deadline first)
- 3 CFS (Completely Fair Scheduling)
- 4 Idle Shed. Class

### 3.2.5 Multicore Scheduling

**Aufgabe:** Verteilung der Jobs auf die CPU-Kerne, so dass unsere Zeitbedingungen eingehalten werden und das System (energie-)effizient arbeitet.

**Lösungen:**

- 1. Partitioniert Scheduling  
Tasks werden auf die CPU-Kerne Statisch verteilt.(per Hand)
- 2. Semipartitioniertes Scheduling  
task werden in Gruppen auf die CPU-Kerne verteilt.
- 3. Globales Scheduling  
Scheduler verteilt die Tasks auf Basis der aktuellen Lastsituation.

Taskmigration: Verschiebung von Tasks auf andere CPU-Kerne  
Problemstellung:

- **Kosten** der Taskmigration sind abhängig von der eingesetzten Hardware
- **Nutzen** ist abhängig von der eingesetzten Hardware

Hardware-Architekturen SMP: Symmetirc Multi Processing (= alle CPU Kerne sind gleich)

Kosten: mittel

Nutzen: mittel

SMT: Symmetric Multi Threading (Hyperthreading, = Verdopplung der Instruction Pipeline)

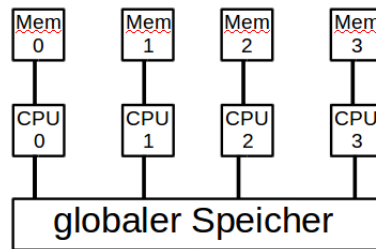
Kosten: niedrig

Nutzen: niedrig

NUMA: Non Uniform memory Architecture

Kosten: hoch

Nutzen: hoch



bigLITTLE-Architektur: (z.B. 4(starke) + 4(schwache) Kerne)

Optimierungsziel: Energieeffizienz

Kosten(in Hinblick auf Leistung) = SMP(mittel)

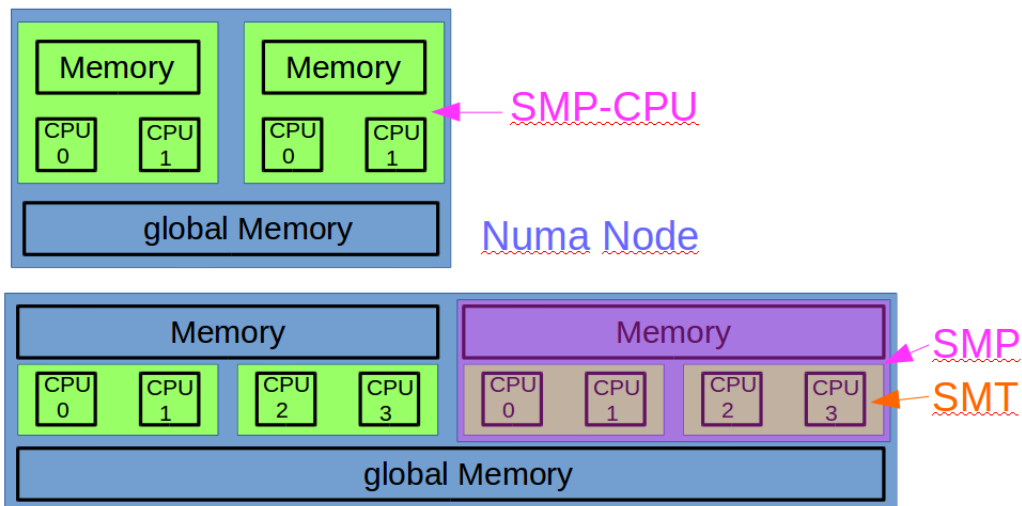
Nutzen = SMP(mittel)

AMP: Asymmetric Multiprocessing (unterschiedliche CPU-Kerne)

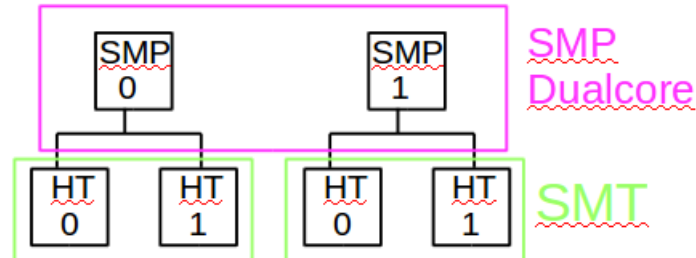
Kosten: hoch

Nutzen: je nach Anwendung

=> wird in der Praxis über Partitioniertes Scheduling genutzt.



Linux bildet beim Booten eine **Scheduling Domain**



Der Multicore Scheduler balanziert die last innerhalb einer Scheduling Domain -> sorgt für ausgeglichen Lastverhältnisse.

Die einzelnen Cores verwenden einen Singlecore Scheduler. Unter Linux ist der name der Rechenprozesse, die für Multicore-Scheduling zuständig sind "migration".

### Der Multicore Scheduler wird aktiv:

- exit()
- pthread\_create(), clone(), fork()
- clock\_nanosleep()
- zeitgesteuert

### 3.2.6 Memory Managment

#### Aufgaben:

- Speicherschutz
- Adressumsetzung
- virtuellen Speicher zur verfügung stellen
- erweiterten Speicher zur verfügung stellen

Technologien:

1. Segmentierung
2. Paging (Seitenorientierung)

## Segmentierung

GRAFIK

## Paging

GRAFIK

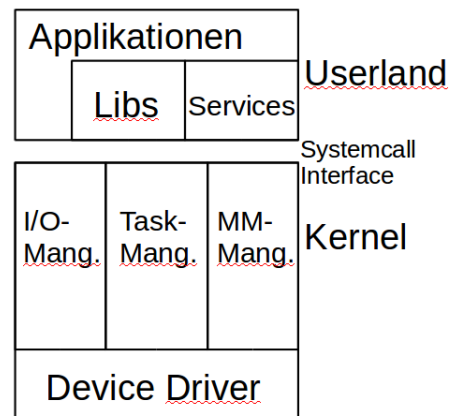
Auf 32 Bit Systemen: Two Level Paging  
Auf 64 Bit Systemen: Three Level paging

**Konsequenz:** Um eine Variable aus dem Hauptspeicher zu lesen, sind auf einem 32 Bit System 3 Hauptspeicherzugriffe notwendig.  
=> zur Optimierung TCB (Cache)  
Außerdem: In den Page Directories sind die obersten Einträge (auf 32Bit 15 Byte) für den Kernspace reserviert.

### 3.2.7 I/O Managment

Aufgabe:

- a) Einheitliche API für den HW Zugriff
- b) Systemkonforme Integration von Hardware über Gerätetreiber
- c) Strukturierter Zugriff auf Daten (Filesystem)



API: `open`, `cloes()`, `read()`, `write()`, `ioctl()`, `fcntl()`, `seek()`

Hintergrund: Direct I/O - Bufferd I/O

`printf("\n Hallo");` <- Buffered

`printf(" Hallo \n");`

Buffered I/O => Daten werden aus Performance-Gründen zwischengespeichert. Reale Ausgabe erfolgt, wenn der Zwischenspeicher (Buffer) voll ist oder wenn ein "\n" kommt.

`fopen(), fclose(), fprintf(), fwrite(), fread(), fflush()`

Direct I/O => Ein-/Ausgabe-Aufrufe werden direkt ausgeführt.

Kontrollfluss (Wann?, Unter welchen Umständen?)

### **Zugriffsarten:**

- Blockierend
- nicht Blockierend
- Asynchron

a) Blockierend

Beispiel "read": Job schläft, bis die angeforderten Daten zur Verfügung stehen -> Carrear Bahn, eigene Spur

b) Nicht Blockierend `fd = open("dev/carrera", O_RDWR|O_NONBLOCK)`

Beispiel "read": Job bekommt die angeforderten Daten oder die Information, dass zu Zeit des Aufrufes keine Daten zur Verfügung stehen.

-> Job wird nicht schlafen gelegt.

c) Asynchron

Aufträge werden dem Kernel übergeben und das Resultat zu einem späteren zeitpunkt abgeholt.

Systemkonforme Integration von Hardware über Gerätetreiber

**Idee:** open, close, read, write, ... bieten einen einheitlichen Zugriff auf unterschiedliche Peripherie

**Außerdem:** Applikationen sollen keinesfalls direkt auf Peripherie zugreifen können (Safety)

**Und:** Applikationen sollen keine Details der Hardware (z.B. Adressen, Bit-maskierungen, Gerätetreiber) kennen müssen.

GRAFIK

Filesysteme Strukturen zur Ablage von Daten auf Hintergrundspeicher in hierarchisch organisierten Dateien.

**Beispiel:** Ext4, NTFS, FAT32, exFat, JFFS2, ISO9660, ...

**Merkmale:**

- maximale FileSystem-Größe
- Anzahl Dateien
- maximale Datei Größe
- Attribute
- Zugriffszeit

4 placeholder4

5 placeholder5

## 6 Realzeitnachweis bei Prioritätengesteuerten und EDF-Scheduling

### 6.1 Zentrale Beschreibungsgrößen (Wiederholung)

Technischer prozess:

- $t_{p,i}$  = Prozesszeit, zeitlicher Abstand zwischen zwei RT-Anforderungen  $i$   
 $t_{Pmin,i}$
- $t_{Dmin,i}$  = minimal zulässige Reaktionszeit
- $t_{Dmax,i}$  = maximal zulässige Reaktionszeit
- $t_{Ph,i}$  = Phase, zeitlicher Abstand zwischen zwei unterschiedlicher Ereignisse

Rechenprozesse:

- $t_{Emin,i}$  = minimale Ausführungszeit BCET
- $t_{Emax,i}$  = maximale Ausführungszeit WCET
- $t_{Rmin,i}$  = minimale Reaktionszeit
- $t_{Rmax,i}$  = maximale Reaktionszeit
  - ↳ Zeitlicher Abstand zwischen dem Eintreffen einer RT-Anforderung  $i$  und dem Ende der Bearbeitung
- $t_{W,i}$  = Wartezeit, Summe der Zeiten, in der eine Codesequenz arbeiten könnte, aber nicht dran kommt.

Systemsoftware:

- $t_{L,i}$  = Latenzzeit, zeitlicher Abstand zwischen dem Eintreffen einer RT-Anforderung  $i$  und dem Start der Bearbeitung
- Schedulingverfahren

1. RT Bedingung

$$\rho_{max,ges} = \sum_{j=1}^n \frac{t_{Emax,i}}{t_{Pin,i}} \leq c$$

$j$  = für alle RZ-Anforderungen,  $c$  = Anzahl der Rechnerkerne

2. RT Bedingung

Für alle RZ-Anforderungen  $j$  muss gelten:

$$t_{Dmin,j} \leq t_{Rmin,j} \leq t_{Rmax,j} \leq t_{Dmax,j}$$

$$\text{Utilization } u = \sum_{j=1}^n \frac{t_{Emax,j}}{\min(t_{Dmax,j}, t_{Pmin,j})}$$

Realzeitnachweis bei prioritätengesteuerten Scheduling

**1. Schritt:** Anforderungen und zeitliche Parameter der Lösung zusammenstellen.

**2. Schritt:** Utilization überprüfen

Realzeitbedingungen werden grundsätzlich eingehalten, falls

$$u \leq n \times (2^{\frac{1}{n}} - 1) \text{ wobei } n = \text{Anzahl der Threads/Tasks, RT-Anforderungen}$$

$$n = 1 \Rightarrow u \leq 100\%$$

$$n = 2 \Rightarrow u \leq 82.8\%$$

$$n = 3 \Rightarrow u \leq 78\%$$

$$n \rightarrow \infty \Rightarrow u \leq 69.3\%$$

**3. Schritt:** 2. Realzeitbedingung überprüfen (falls  $u \leq n \times (2^{\frac{1}{n}} - 1)$ )

Problem: Bestimmung von  $t_{Rmax}$

Idee: Die  $t_{Emax}$  der höheren oder gleich Priorität Tasks werden aufsummiert über die Zeit (= Arbeit für den Rechner zum Zeitpunkt  $t$ ). Es wird der



Zeitpunkt gesucht, an dem die benötigte Zeit nicht mehr größer als die zur Verfügung gestellte Rechenzeit ist.

$$t_{c,p}(t) = \sum_{j \in J} \left\lceil \frac{t}{t_{Pmin,j}} \right\rceil \times t_{Emax,j}$$

J = alle höher oder gleich Prioren Jobs

p = Priorität

Die zur Verfügung stehende Rechenzeit ergibt sich zu

$$t_{available}(t) = t$$

Gesucht ist damit die Lösung der Gleichung

$$t_c(t) = t$$

Diese Gleichung lässt sich iterativ lösen:

$$\text{Startwert: } t_p^{(1) < -Iterationsschritt} = \sum_{j \in J} t_{Emax,j}$$

$$\text{Iteration: } t_p^{(l+1)} = t_{c,p}(t_p^{(l)}) = \sum \left\lceil \frac{t_p^{(l)}}{t_{Pmin,j}} \right\rceil \times t_{Emax,j}$$

$$\text{Abbruch: } t_p^{(l)} == t_p^{(l+1)} \quad t_{Rmax,p} = t_p^{(l)}$$

Table 2: Beispiel:

RZ-Anf	$t_{Pmin}$	$t_{Dmin}$	$t_{Dmax}$	$t_{Ph}$	$t_{Emin}$	$t_{Emax}$	$t_{Rmin}$	$t_{Rmax}$	Prio
A	30ms	0ms	20ms	0ms	2ms	10ms	2ms		1
B	45ms	0ms	45ms	0ms	3ms	15ms	3ms		2
C	60ms	0ms	60ms	0ms	4ms	15ms	4ms		3

$$t_{Emin} = t_{Rmin}$$

Berechnung der Utilization u:

$$u = \sum_{j=1}^n \frac{t_{Emax,j}}{\min(t_{Dmax,j}, t_{Pmin,j})} = \frac{10ms}{20ms} + \frac{15ms}{45ms} + \frac{15ms}{60ms} = 1.083$$

$$\rho_{max,ges} = \sum_{j=1}^n \frac{t_{Emax,j}}{t_{Pmin,j}} \leq c \Rightarrow \frac{10ms}{30ms} + \frac{15ms}{45ms} + \frac{15ms}{60ms} = 0.917 \leq 1$$

Berechnung der Schranke für n = 3

$$s = n \times (2^{\frac{1}{n}}) = 0.78$$

Bedingung  $u \leq 0.78$  ist nicht erfüllt -> weiter rechnen

Bestimmung von  $t_{Rmax,1}$  für die Jobs der Priorität 1

$$1. \text{ Aufstellen von } t_{c,p}(t) = \sum_{j \in J} \left\lceil \frac{t}{t_{Pmin,j}} \right\rceil \times t_{Emax,j} = t_{c,1}(t) = \left\lceil \frac{t}{30ms} \right\rceil \times 10ms$$

$$2. \text{ Startwert: } t_1^{(1)} = 10ms (= t_{Emax,1})$$

$$3. \text{ Iteration: } t_1^{(2)} = t_{c,1}(10ms) = \left\lceil \frac{10ms}{30ms} \right\rceil \times 10ms = 10ms$$

$$\text{Abbruch, da: } t_1^{(1)} = t_1^{(2)} \Rightarrow t_{Rmax,1} = 10ms$$

Priorität 2

$$1. \text{ Aufstellen von } t_{c,2}(t) = \underbrace{\left\lceil \frac{t}{30ms} \right\rceil \times 10ms}_A + \underbrace{\left\lceil \frac{t}{45ms} \right\rceil \times 15ms}_B$$

$$2. \text{ Startwert: } t_2^{(1)} = t_{Emax,A} + t_{Emax,B} = 10ms + 15ms = 25ms$$

$$3. \text{ Iteration: } t_2^{(2)} = t_{c,2}(25ms) = \left\lceil \frac{25ms}{30ms} \right\rceil \times 10ms + \left\lceil \frac{25ms}{45ms} \right\rceil \times 15ms = 25ms$$

$$\text{Abbruch, da: } t_2^{(1)} = t_2^{(2)} \Rightarrow t_{Rmax,2} = 25ms$$

Priorität 3

$$1. \text{ Aufstellen von } t_{c,3}(t) = \underbrace{\left\lceil \frac{t}{30ms} \right\rceil \times 10ms}_A + \underbrace{\left\lceil \frac{t}{45ms} \right\rceil \times 15ms}_B + \underbrace{\left\lceil \frac{t}{60ms} \right\rceil \times 15ms}_C$$

2. Startwert:

$$t_3^{(1)} = t_{Emax,A} + t_{Emax,B} + t_{Emax,C} = 10ms + 15ms + 15ms = 40ms$$

3. Iteration:

$$t_3^{(2)} = t_{c,3}(40ms) = \left\lceil \frac{40ms}{30ms} \right\rceil \times 10ms + \left\lceil \frac{40ms}{45ms} \right\rceil \times 15ms + \left\lceil \frac{40ms}{60ms} \right\rceil \times 15ms = 50ms$$

$$t_3^{(3)} = t_{c,3}(50ms) = \left\lceil \frac{50ms}{30ms} \right\rceil \times 10ms + \left\lceil \frac{50ms}{45ms} \right\rceil \times 15ms + \left\lceil \frac{50ms}{60ms} \right\rceil \times 15ms = 65ms$$

$$t_3^{(4)} = t_{c,3}(65ms) = \left\lceil \frac{65ms}{30ms} \right\rceil \times 10ms + \left\lceil \frac{65ms}{45ms} \right\rceil \times 15ms + \left\lceil \frac{65ms}{60ms} \right\rceil \times 15ms = 90ms$$

$$t_3^{(5)} = t_{c,3}(90ms) = \left\lceil \frac{90ms}{30ms} \right\rceil \times 10ms + \left\lceil \frac{90ms}{45ms} \right\rceil \times 15ms + \left\lceil \frac{90ms}{60ms} \right\rceil \times 15ms = 90ms$$

Abbruch, da:  $t_3^{(4)} == t_3^{(5)} \Rightarrow t_{Rmax,3} = 90ms$

### Überprüfen der 2.RT-Bedingung

	$t_{Dmin,j}$	$t_{Rmin,j}$	$t_{Rmax,j}$	$t_{Dmax,j}$
Prio 1	0ms ≤	2ms ≤	10ms ≤	20ms
Prio 2	0ms ≤	3ms ≤	25ms ≤	45ms
Prio 3	0ms ≤	4ms ≤	90ms ≤	60ms

Jobs der Priorität 3 (Task C) werden unter Umständen nicht schritthal-  
tend abgearbeitet.

### RT-Nachweis EDF(Deadline Scheduling)

**1. Schritt:** Anforderungen und Beschreibungsgrößen zusammenstellen

**2. Schritt:** Auslastungsbedingung und Utilization überprüfen

Grenze  $S = 1$  <- auf Singelcore

$$u = \sum_{j=1}^n \frac{t_{Emax,j}}{\min(t_{Dmax,j}, t_{Pmin,j})} \leq 1$$

**3. Aufstellen:**  $t_c(I)$  //Intervall I nicht t!

$$t_c(I) = \sum_{j=1}^n \left\lfloor \frac{I + t_{Pmin,j} - t_{Dmax,j} - t_{Ph,j}}{t_{Pmin,j}} \right\rfloor \times t_{Emax,j}$$

**4. Überprüfung der RT-Bedingung:**  $t_c(J) \leq J$

$$t_c(J) \leq J$$

für alle J ein Bereich  $0 \leq J \leq kgV(t_{Pmin,j}) + max(t_{Ph,j})$

Hierzu:

- a) die zu berechnenden J bestimmen
- b) für alle zu berechnenden J ist  $t_c(J)$  auszurechnen.
- c) Bedingungen überprüfen
  - $t_{Dmin,j} \leq t_{Rmin,j}$
  - $t_c(J) \leq J$

Table 3: Beispiel(EZN bei EDF):

RZ-Anf	$t_{Pmin}$	$t_{Dmin}$	$t_{Dmax}$	$t_{Ph}$	$t_{Emin}$	$t_{Emax}$	$t_{Rmin}$	$t_{Rmax}$	Prio
A	30ms	0ms	20ms	0ms	2ms	10ms	2ms		1
B	45ms	0ms	45ms	0ms	3ms	15ms	3ms		2
C	60ms	0ms	60ms	10ms	4ms	15ms	4ms		3

**2. Auslastung :**

$$\rho_{max,ges} = \sum_{j=1}^n \frac{t_{Emax,j}}{t_{Pmin,j}} \leq c \Rightarrow \frac{10ms}{30ms} + \frac{15ms}{45ms} + \frac{15ms}{60ms} = 0.917 \leq 1$$

**3. Utilization :**

$$u = \sum_{j=1}^n \frac{t_{Emax,j}}{\min(t_{Dmax,j}, t_{Pmin,j})} = \frac{10ms}{20ms} + \frac{15ms}{45ms} + \frac{15ms}{60ms} = 1.083$$

s = 1 (Bei EDF)

weitermachen da  $u \leq s$

4. RT-Bedingung prüfen :

$$\begin{aligned}
 t_c(J) &= \sum_{j=1}^n \left\lfloor \frac{J + t_{Pmin,j} - t_{Dmax,j} - t_{Ph,j}}{t_{Pmin,j}} \right\rfloor \times t_{Emax,j} \\
 &= \left\lfloor \frac{J + 30ms - 20ms - 0ms}{30ms} \right\rfloor \times 10ms \text{ A} \\
 &= \left\lfloor \frac{J + 45ms - 45ms - 0ms}{45ms} \right\rfloor \times 15ms \text{ B} \\
 &= \left\lfloor \frac{J + 60ms - 60ms - 10ms}{60ms} \right\rfloor \times 15ms \text{ C} \\
 &= \left\lfloor \frac{J + 10ms}{30ms} \right\rfloor \times 10ms + \left\lfloor \frac{J}{45ms} \right\rfloor \times 15ms + \left\lfloor \frac{J - 10ms}{60ms} \right\rfloor \times 15ms
 \end{aligned}$$

5. :

$$\begin{aligned}
 0 &\leq JkgV(30ms, 45ms, 60ms) + \max(0ms, 0ms, 10ms) \\
 0 &\leq J \leq 180ms + 10ms \\
 0 &\leq J \leq 190ms
 \end{aligned}$$