



Vorlesung  
**Moderne  
Realzeitsysteme**

**Systemsoftware**

Professor Dr. Michael Mächtel

Systemsoftware

## Lernziele

- Aufgaben der Firmware kennen lernen
- Strukturen von Realzeitsystemen unterscheiden können
- Aufgaben des Realzeitbetriebssystems kennen lernen:
  - Taskmanagement
  - Scheduling
  - E/A Management
- Zeitverwaltung der Systemsoftware kennen lernen

Professor Dr. Michael Mächtel 2

Systemsoftware


## Inhalt

- Firmware
- Realzeitbetriebssysteme
  - Systemcalls
  - Taskmanagement
  - Memory Management
  - I/O Management
  - Timekeeping
  - Sonstige Realzeitaspekte
- Linux

Professor Dr. Michael Mächtel 3

Systemsoftware

## Firmware



Professor Dr. Michael Mächtel 4

## Firmware

- Bezeichnung der hardwarenahen Systemsoftware
- Häufig im Flashspeicher
  - Zugriff (eventuell) über JTAG-Interface
- Ausprägungen
  - Bios
  - Systemsoftware intelligenter IO (z.B. WLAN-Karten)

## Firmware

### ■ Aufgaben

- Grundinitialisierung
  - CPU
  - MMU (Speicherverwaltung)
  - Timer/Uhr
- Diagnose, Funktionstest
- Betriebsinitialisierung
  - Interrupt
  - MMU
  - Timer/Uhr
  - Watchdog
  - Serielle Schnittstelle

## Firmware

- Systemsoftware – Architekturvarianten
  - Deeply embedded systems
    - Firmware und Applikation bilden eine Einheit
    - Vorteil: Kurze Bootzeit
    - Nachteil: Unflexibel, beispielsweise bei notwendigen Updates

**Firmware Applikation**

## Firmware

### ■ Systemsoftware – Architekturvarianten

- Einfache eingebettete Systeme
  - Firmware wird mit einem Bootloader kombiniert.
  - Die geladene Applikation wird häufig ebenfalls „Firmware“ genannt.
  - Vorteil: Funktionalität kann den Bedürfnissen angepasst werden.
  - Nachteil: Längere Bootzeit durch den Lademechanismus.

**Firmware Bootloader**

**Applikation**

## Firmware

- Systemsoftware – Architekturvarianten
  - Multifunktionale eingebettete Systeme
    - Firmware und Bootloader laden ein Realzeitbetriebssystem (zum Beispiel Linux).
    - Das Betriebssystem führt die eigentliche Applikation aus.
    - Vorteil: Sehr flexibel.
    - Nachteil: Ressourcenverbrauch.

Firmware Bootloader

Linux Rootfilesystem

Applikation

## Firmware

- Systemsoftware – Architekturvarianten
  - PC-basiertes Realzeitsystem
    - Firmware und Bootloader laden einen flexiblen Bootloader.
    - Der Bootloader lädt ein (Realzeit-)Betriebssystem.
    - Vorteil: Einsatz standardisierter Hardware.
    - Nachteil: Ressourcenverbrauch, lange Bootzeit.

Firmware Bootloader

Bootloader

Linux Rootfilesystem

Applikation

## Firmware

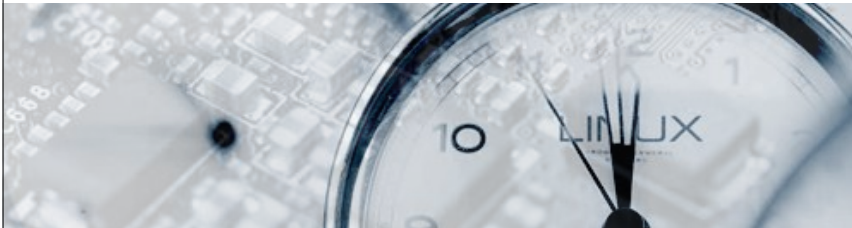
- Auf einer PC-basierten Plattform wird die Firmware „**BIOS**“ (Basic Input Output System) genannt.
- Moderne PCs verwenden **UEFI** (unified extensible firmware interface)
  - Boot-Services
  - Runtime-Services
- Als second-Stage Bootloader werden häufig **grub** oder **syslinux** eingesetzt.
- ARM-basierte Plattformen setzen als Firmware und Bootloader häufig „**Das U-Boot**“ oder „**barebox**“ ein.

## Firmware

- **Zusammenfassung**
  - Bezeichnung für hardwarenahe, eigenständige Software
  - Systemsoftware ist unter anderem
    - Firmware
    - Bootloader
    - Monitor
    - Betriebssystem
  - Unterschiedliche Architekturvarianten
    - deeply embedded systems
    - Einfache eingebettete Systeme
    - Multifunktionale Systeme
    - PC-basierte Systeme

## Realzeitbetriebssysteme (RT-OS)

1. **Systemcalls**
2. **Taskmanagement**
3. **Memory Management**
4. **I/O Management**
5. **Timekeeping**
6. **Sonstige Realzeitaspekte**



## Realzeitbetriebssysteme

### Definition

Bezeichnung für alle **Softwarekomponenten**, die

- die **Ausführung der Benutzerprogramme**
- die **Verteilung der Betriebsmittel** (z.B. Speicher, Prozessor, Dateien, Drucker)

ermöglichen, steuern und überwachen.

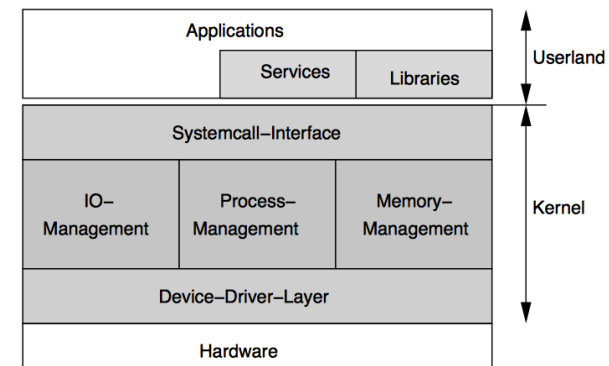
## Realzeitbetriebssysteme

### ■ Anforderungen an Realzeitbetriebssysteme

- Zeitverhalten (deterministisch, kurze Reaktionszeiten)
- Ressourcenverbrauch (Speicher, Rechenzeit)
- Zuverlässigkeit und Stabilität (Veralten bei fehlerhafter Software)
- IT-Sicherheit (Zugangsschutz, Dateischutz)
- Flexibilität und Kompatibilität (Unterstützung von Standard, POSIX)
- Portabilität (Betriebssystem ist auf unterschiedlicher Hardware lauffähig)
- Skalierbarkeit (Unterstützung von unterschiedlich leistungsfähiger Hardware)

## Realzeitbetriebssysteme

### ■ Betriebssystem-Architektur



# Realzeitbetriebssysteme (RT-OS)

1. **Systemcalls**
2. Taskmanagement
3. Memory Management
4. I/O Management
5. Timekeeping
6. Sonstige Realzeitaspekte



## RT-OS – Systemcalls

- Dienste des Betriebssystemkerns
- Beispiele:
  - open, close, read, write: Datei- und Peripheriezugriff
  - clone, exit, wait, kill: Taskmanagement
  - clock\_nanosleep, clock\_gettime: Zeitverwaltung
- Aktivierung eines Systemcalls wird über einen Softwareinterrupt realisiert (beziehungsweise spezifische Prozessorbefehle wie `sysenter`)
- Moderne Betriebssysteme implementieren mehr als 300 Systemcalls
- Bibliotheksfunktionen kapseln Systemcalls für die Anwendung

## RT-OS – Systemcalls

### ■ Systemcalls

```
.text
.globl write_hello_world
write_hello_world:
    movl $4,%eax      ; //code fuer "write" syscall
    movl $1,%ebx      ; //file descriptor fd (1=stdout)
    movl $message,%ecx ; //Adresse des Textes (buffer)
    movl $12,%edx     ; //Laenge des auszugebenden Textes
    int $0x80         ; //SW-Interrupt, Auftrag an das BS
    ret
.data
message:
    .ascii "Hello World\n"
```

## Realzeitbetriebssysteme (RT-OS)

1. Systemcalls
2. **Taskmanagement**
3. Memory Management
4. I/O Management
5. Timekeeping
6. Sonstige Realzeitaspekte



## RT-OS – Taskmanagement

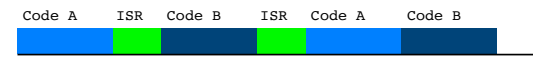
### Aufgaben:

- Verwaltung der Ressource CPUs
- Scheduling
- Quasi-parallele Verarbeitung mehrerer Tasks
- Real-parallele Verarbeitung mehrerer Tasks

## RT-OS – Taskmanagement

### ■ Realisierung

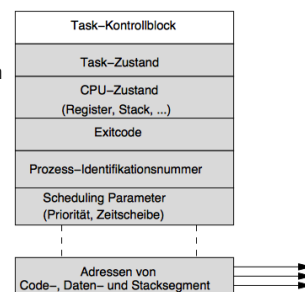
- Realisierung der quasi-parallelen Verarbeitung:
  - Innerhalb einer Interrupt Service Routine (ISR) wird die durch die Hardware auf dem Stack abgelegte Rücksprungadresse (in den normalen Code) ausgetauscht.
  - Nach Ende der ISR arbeitet die CPU damit nicht mit dem Code weiter, an dem sie vor der Unterbrechung durch die ISR gearbeitet hat, sondern an der „ausgetauschten“ Codesequenz.
  - Wird mit dem nächsten Interrupt wieder die ursprüngliche Adresse auf dem Stack abgelegt (ausgetauscht), arbeitet die CPU nach dem Ende des Interrupts die ursprüngliche Codesequenz ab.



## RT-OS – Taskmanagement

### ■ Context Switch

- Durch Interrupts werden Codesequenzen „preempted“ (unterbrochen), sie werden zu einem späteren Zeitpunkt fortgesetzt.
- Hierzu ist es erforderlich, den genauen Zustand zum Zeitpunkt der Unterbrechung zu speichern.
- Die hierfür verwendete Datenstruktur nennt sich „Task Control Block“ (TCB).
- Für jede unabhängige Codesequenz (Task) wird ein TCB benötigt.
- Der Teil der ISR, die den Austausch der Rücksprungadresse vornimmt, wird „Context Switch“ genannt.



## RT-OS – Taskmanagement

### ■ Scheduler

- Die ISR wird außerdem um Code ergänzt, die aus den vorliegenden Codesequenzen diejenige auswählt, die als nächstes abgearbeitet werden soll. Dieser Code wird mit „Scheduler“ bezeichnet.
- Der Scheduler benötigt für die Auswahl Kriterien. Als Kriterien bieten sich beispielsweise an:
  - Priorität
  - Bereits verwendete Verarbeitungszeit
- Ohne Interrupts gibt es kein Scheduling.

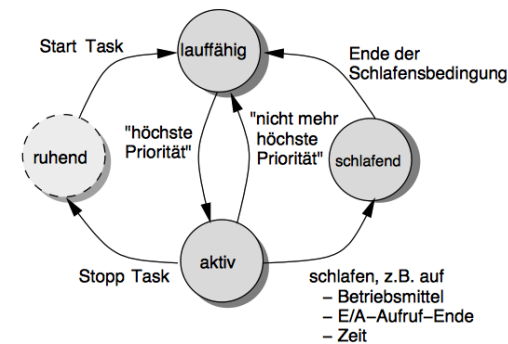
## RT-OS – Taskmanagement

### ■ Scheduler

- Klassisch: Per Timer werden regelmäßig – zum Beispiel als 10 ms – Interrupts ausgelöst
  - Timertick.
- Modern: Mit jedem Interrupt rechnet der Kernel den nächsten Zeitpunkt aus, an dem sinnvoll das nächste Scheduling stattfindet
  - Tickless-System.
  - Vorteil:
    - zeitliche Genauigkeit
    - energieeffizient
  - Nachteil:
    - Overhead

## RT-OS – Taskmanagement

### ■ Übersicht Taskzustände



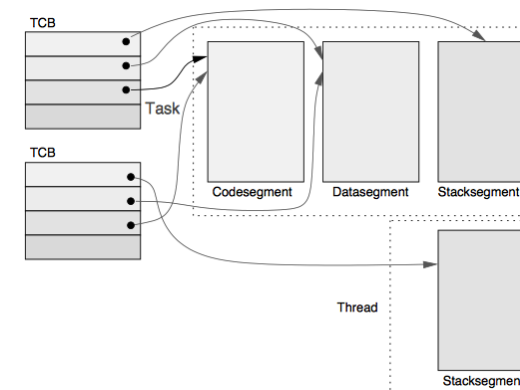
## RT-OS – Taskmanagement

### ■ Taskzustände

- Die vom Scheduler ausgewählte und vom Context Switch aktivierte Task wird als „aktiv“ bezeichnet.
- Eine Task, die für die Verarbeitung Daten (oder ähnliches) benötigt, wird als „schlafend“ bezeichnet. Es ist nicht sinnvoll, wenn der Scheduler diese Task auswählen würde.
- Eine Task, die etwas zu arbeiten hat, aber gerade nicht ausgewählt wurde, wird als „lauffähig“ oder „bereit“ bezeichnet.
- Tasks und deren Zustandsübergänge werden im Taskzustandsübergangsdiagramm dargestellt.

## RT-OS – Taskmanagement

### ■ Prozesse und Threads



## RT-OS – Taskmanagement

### ■ Vergleich zwischen Prozesse und Threads

Prozess	Thread
Eigener TCB	Eigener TCB
Eigenes oder gemeinsam genutztes Codesegment	Gemeinsam genutztes Codesegment
Eigenes Datensegment	Gemeinsam genutztes Datensegment
Eigenes Stacksegment	Eigenes Stacksegment

## RT-OS – Taskmanagement

### ■ Scheduling

- Interrupts sind also Grundvoraussetzung für quasi-parallele Verarbeitung (preemptives Scheduling).
- Schedulingplan: zeitliche Zuordnung der Tasks zu den Prozessoren und Ressourcen
- Werden von den Tasks alle maximalen Deadlines eingehalten, so arbeitet das System korrekt (Korrektter Schedulingplan)
- Ob für eine Menge von Tasks ein korrekter Schedulingplan existiert, wird mit einem Einplanbarkeitstest untersucht.

## RT-OS – Taskmanagement

### ■ Schedulingtest

- Damit die Tasks korrekt gescheduled werden können, darf die Auslastung auf jedem CPU-Kern nicht über 100% sein ( $c=1$ ):

$$\rho_{ges} = \sum_{j=1}^n \frac{t_{E_{max},j}}{t_{P_{min},j}} \leq c$$

- Der Schedulingplan definiert den Zeitbereich von 0 bis zu einem Zeitpunkt  $t$ , ab welchem sich der Plan wiederholt. Dieser Zeitbereich wird **Hyperperiode** genannt.

## RT-OS – Taskmanagement

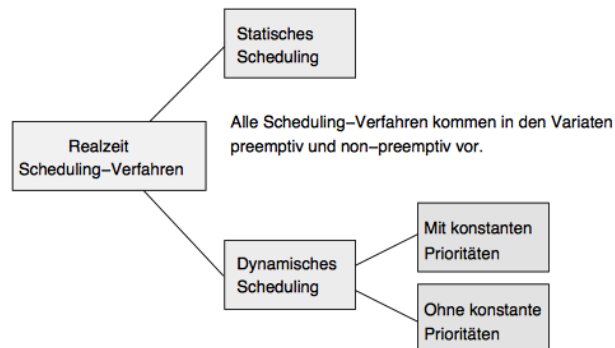
### ■ Scheduling-Verfahren

- Ein Scheduling-Verfahren wird als optimal bezeichnet, wenn es in jeder Situation einen korrekten Schedulingplan findet.
- Ein Scheduling-Verfahren ist nicht-optimal, wenn es keinen korrekten Schedulingplan findet, obwohl mindestens einer existiert.
- In einem überlasteten System existiert kein korrekter Schedulingplan.



## RT-OS – Taskmanagement

### ■ preemptive / non-preemptive Scheduling-Verfahren



## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

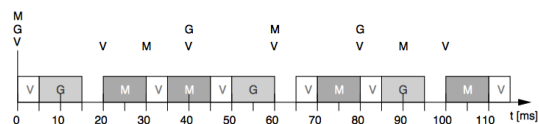
- Wird eingesetzt in Systemen mit harten Realzeitanforderungen und einem einfachen Laufzeitsystem
- Geeignet für Harte Realzeitsysteme
- Realzeitznachweis einfach
  - Schedulingplan wird offline berechnet (Zeitslots)
  - Keine Interruptverarbeitung in den Zeitslots

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan.
  - Ziel: Task ohne Unterbrechung ausführen.

Task	$t_{pmin}$	$t_{Dmin}$	$t_{Dmax}$	$t_{Emin}$	$t_{Emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
G	40 ms	0 ms	40 ms	1 ms	10 ms
M	30 ms	0 ms	30 ms	1 ms	10 ms



**Aufwendig: ständiges Programmieren des Timers**

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

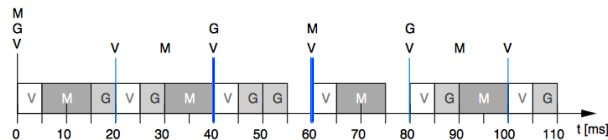
- Zeitgesteuerter Schedulingplan: nach jedem Taskende muss der Timer neu programmiert werden!

Releasetime	Task
0	V
5	G
20	M
30	V
35	M
45	V
50	G
65	V
70	M
80	V
85	G
100	M
110	V

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots
  - Vorteil: Timer muss nicht umprogrammiert werden!
- Lösung mit Framesize 20:



Wie kommt man auf diese Lösung?

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots
- Bestimmung der konstanten Framesize / Slotlänge (Frame f):

0. Hyperperiode (H) bestimmen:
  - kompletter Schedulingplan muss erstellt werden
1. untere und obere Framegrenze bestimmen
  - min. Framlänge (f) entspricht größter Ausführungszeit des Tasksets
  - max. Framlänge (f) entspricht kleinster maximale Deadline des Tasksets
2. Schedulingtabellengröße optimieren
  - f muß ganzzahliger Teiler von H sein (ganzzahliger Teiler der min. Prozesszeit)
3. Überprüfung der möglichen Framegröße für jede Task
  - zwischen Release Time und maximaler Deadline JEDER Task muss mindestens 1 Frame liegen

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots
- Bestimmung der konstanten Slotlänge (Frame f):

0. KgV aller  $p_{\min}$
1.  $f \geq \max_{1 \leq j \leq n} (t_{E_{\max,j}})$  und  $f \leq \min_{1 \leq j \leq n} (t_{D_{\max,j}})$
2.  $\exists i : \text{mod}(t_{p,i}, f) = 0$
3.  $2 * f - \text{gcd}(t_{P_{\min,i}}, f) \leq t_{D_{\max,i}}$

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots
- Bestimmung der konstanten Slotlänge (Frame f):

Task	$t_{p_{\min}}$	$t_{d_{\min}}$	$t_{d_{\max}}$	$t_{e_{\min}}$	$t_{e_{\max}}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
G	40 ms	0 ms	40 ms	1 ms	10 ms
M	30 ms	0 ms	30 ms	1 ms	10 ms

0. KgV aller  $p_{\min}$ 
  - $\Rightarrow \text{KgV}(20, 30, 40) = 120$

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots  
Bestimmung der konstanten Slotlänge (Frame f):

Task	$t_{pmin}$	$t_{dmin}$	$t_{dmax}$	$t_{emin}$	$t_{emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
G	40 ms	0 ms	40 ms	1 ms	10 ms
M	30 ms	0 ms	30 ms	1 ms	10 ms

$$1. f \geq \max_{1 \leq j \leq n} (t_{Emax,j}) \text{ und } f \leq \min_{1 \leq j \leq n} (t_{Dmax,j})$$

$\Rightarrow f$  im Intervall [10,20]

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots  
Bestimmung der konstanten Slotlänge (Frame f):

Task	$t_{pmin}$	$t_{dmin}$	$t_{dmax}$	$t_{emin}$	$t_{emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
G	40 ms	0 ms	40 ms	1 ms	10 ms
M	30 ms	0 ms	30 ms	1 ms	10 ms

$$2. \exists i : \text{mod}(t_{p,i}, f) = 0$$

V: 1,2,4,5,10,20  
G: 1,2,4,5,8,10,20,40  
M: 1,2,3,5,6,10,15

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

Task	$t_{pmin}$	$t_{dmin}$	$t_{dmax}$	$t_{emin}$	$t_{emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
G	40 ms	0 ms	40 ms	1 ms	10 ms
M	30 ms	0 ms	30 ms	1 ms	10 ms

$$3. 2 * f - \text{gcd}(t_{Pmin,i}, f) \leq t_{Dmax,i}$$

V mit  $f = 10\text{ms}$ :  $2 \cdot 10\text{ms} - \text{gcd}(20\text{ms}, 10\text{ms}) \leq 20\text{ms}$ ;  $20\text{ms} - 10\text{ms} \leq 20\text{ms}$  (OK)

V mit  $f = 15\text{ms}$ :  $2 \cdot 15\text{ms} - \text{gcd}(20\text{ms}, 15\text{ms}) \leq 20\text{ms}$ ;  $30\text{ms} - 5\text{ms} > 20\text{ms}$

V mit  $f = 20\text{ms}$ :  $2 \cdot 20\text{ms} - \text{gcd}(20\text{ms}, 20\text{ms}) \leq 20\text{ms}$ ;  $40\text{ms} - 20\text{ms} \leq 20\text{ms}$  (OK)

G mit  $f = 10\text{ms}$ :  $2 \cdot 10\text{ms} - \text{gcd}(40\text{ms}, 10\text{ms}) \leq 40\text{ms}$ ;  $20\text{ms} - 10\text{ms} \leq 40\text{ms}$  (OK)

G mit  $f = 15\text{ms}$ :  $2 \cdot 15\text{ms} - \text{gcd}(40\text{ms}, 15\text{ms}) \leq 40\text{ms}$ ;  $30\text{ms} - 5\text{ms} \leq 40\text{ms}$  (OK)

G mit  $f = 20\text{ms}$ :  $2 \cdot 20\text{ms} - \text{gcd}(40\text{ms}, 20\text{ms}) \leq 40\text{ms}$ ;  $40\text{ms} - 20\text{ms} \leq 40\text{ms}$  (OK)

M mit  $f = 10\text{ms}$ :  $2 \cdot 10\text{ms} - \text{gcd}(30\text{ms}, 10\text{ms}) \leq 30\text{ms}$ ;  $20\text{ms} - 10\text{ms} \leq 30\text{ms}$  (OK)

M mit  $f = 15\text{ms}$ :  $2 \cdot 15\text{ms} - \text{gcd}(30\text{ms}, 15\text{ms}) \leq 30\text{ms}$ ;  $30\text{ms} - 15\text{ms} \leq 30\text{ms}$  (OK)

M mit  $f = 20\text{ms}$ :  $2 \cdot 20\text{ms} - \text{gcd}(30\text{ms}, 20\text{ms}) \leq 30\text{ms}$ ;  $40\text{ms} - 10\text{ms} \leq 30\text{ms}$  (OK)

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots  
Bestimmung der konstanten Slotlänge (Frame f):

Task	$t_{pmin}$	$t_{dmin}$	$t_{dmax}$	$t_{emin}$	$t_{emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
G	40 ms	0 ms	40 ms	1 ms	10 ms
M	30 ms	0 ms	30 ms	1 ms	10 ms

- Mögliche Lösungen für  $f=10$  und  $f=20$

- Experimentelle Rechnerkernverteilung für  $f=10$  und  $f=20$  liefert jedoch keinen korrekten Schedulingplan ...

## RT-OS – Taskmanagement

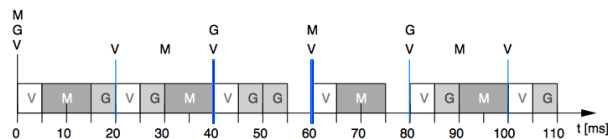
### ■ Statisches Singlecore-Scheduling

- Weiteres Unterteilen der Tasks (wenn möglich)
- Hier Task G:

Task	$t_{pmin}$	$t_{dmin}$	$t_{dmax}$	$t_{emin}$	$t_{emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
G	40 ms	0 ms	40 ms	1 ms	10 ms
M	30 ms	0 ms	30 ms	1 ms	10 ms

→  $G1 = 5ms$   
→  $G2 = 5ms$

### ■ Mögliche Lösung mit $f=20$



## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling

- Zeitgesteuerter Schedulingplan mit konstanten Zeitslots  
Vorteil: Timer muss nicht umprogrammiert werden!
- Tasks einer Frame werden vom Dispatcher nacheinander abgearbeitet (Batch-Jobs)
- $f=20$ :

Frame-Nummer	Tasks
0	V, M, G1
1	V, G2, M
2	V, G1, G2
3	V, M
4	V, G1, M
5	V, G2

## RT-OS – Taskmanagement

### ■ Statisches Singlecore-Scheduling: Bewertung

- Falls Tasks nicht unterbrochen werden müssen:
  - Keine Synchronisation zwischen den Tasks nötig, einfaches Betriebssystem(Laufzeitsystem) langt aus.
  - Einfacher Realzeitnachweis.
- Falls Tasks unterbrochen werden müssen
  - Fehleranfällig, da 'künstlich' aus einer Tasks u.U. mehrere Tasks entstehen

## RT-OS – Taskmanagement

### ■ Dynamisches Singlecore-Scheduling

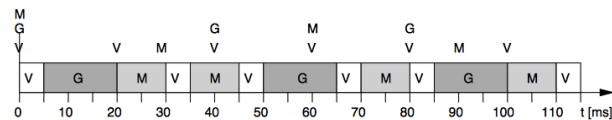
- Typischerweise in ereignisgesteuerten OS:
  - FIFO (FCFS) Scheduling
    - Tasks werden entsprechend den Auftrittszeitpunkten geschedult.
  - RoundRobin (Zeitscheiben) Scheduling
    - Tasks werden wiederholt für die Dauer einer Zeitscheibe reihum ausgeführt.
  - Prioritätenbasiertes Scheduling
    - Scheduler wählt die Task mit höchster Priorität aus.

# RT-OS – Taskmanagement

- **FIFO / FCFS (Auftrittsreihenfolge 'VGM')**

- Eine Task wird solange ausgeführt, bis diese freiwillig die Kontrolle abgibt (z.B. mit Syscall `yield()` )

Task	$t_{pmin}$	$t_{Dmin}$	$t_{Dmax}$	$t_{emin}$	$t_{Emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
GUI	40 ms	0 ms	40 ms	1 ms	15 ms
MONITORING	30 ms	0 ms	30 ms	1 ms	10 ms



# RT-OS – Taskmanagement

- **Bewertung FIFO/FCFS:**

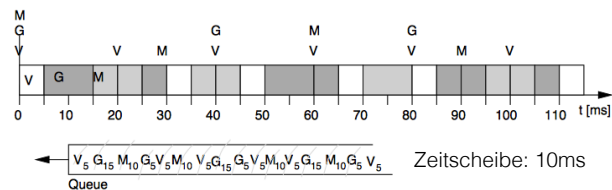
- Nicht für Realzeit geeignet
  - FIFO (FCFS) Scheduling
    - Tasks werden entsprechend den Auftrittszeitpunkten gescheduled.
  - Als eigenständiges Scheduling in Realzeitsystemen nicht geeignet
    - Tasks können beliebig lange laufen
    - Fehler in einer Task können das System dauerhaft blockieren
  - In Kombination mit prioritätenbasiertem Scheduling
    - Tasks gleicher Prioritätenebene können nach FIFO/FCFS gescheduled werden (POSIX).

# RT-OS – Taskmanagement

### ■ Zeitscheibenverfahren (Auftrittsreihenfolge 'VGM')

- Eine Task wird solange ausgeführt, bis diese freiwillig die Kontrolle abgibt (z.B. mit Syscall `yield()` )

Task	$t_{pmin}$	$t_{Dmin}$	$t_{Dmax}$	$t_{Emin}$	$t_{Emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
GUI	40 ms	0 ms	40 ms	1 ms	15 ms
MONITORING	30 ms	0 ms	30 ms	1 ms	10 ms



# RT-OS – Taskmanagement

- **Bewertung Zeitscheibenverfahren:**

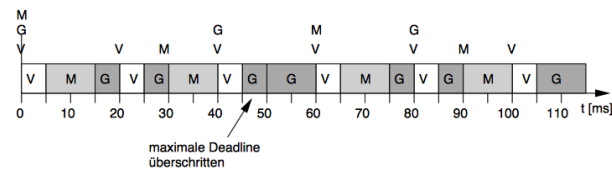
- Nicht für Realzeit geeignet
  - RoundRobin (Zeitscheiben) Scheduling
    - Tasks werden wiederholt für die Dauer einer Zeitscheibe reihum ausgeführt.
    - Wichtigere Tasks können eine längere Zeitscheibe erhalten, jedoch ist keine Garantie für ein rechtzeitiges Scheduling möglich
  - In Kombination mit Prioritätenbasierten Scheduling
    - Tasks gleicher Prioritätenebene können nach Zeitscheibenverfahren gescheduled werden (POSIX).

## RT-OS – Taskmanagement

### ■ Prioritätengesteuertes Scheduling

- Faustregel: Tasks mit kurzer max. Ausführungszeit UND kurzer maximaler Deadline bekommen hohe (feste) Priorität zugeteilt ( $V=1$ ;  $M=2$ ;  $G=3$ ).

Task	$t_{pmin}$	$t_{dmin}$	$t_{dmax}$	$t_{emin}$	$t_{emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
GUI	40 ms	0 ms	40 ms	1 ms	15 ms
MONITORING	30 ms	0 ms	30 ms	1 ms	10 ms



## RT-OS – Taskmanagement

### ■ Bewertung Prioritätenbasiertes Scheduling mit festen zugeteilten Prioritäten:

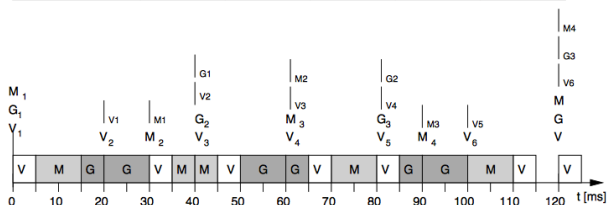
- Für Realzeit geeignet
  - Standard Scheduling in allen Realzeitbetriebssystemen
  - Arbeitet nicht optimal:
    - Korrekter Schedulingplan wäre möglich, jedoch findet ein Scheduling mit statischen Prioritäten diesen nicht.

## RT-OS – Taskmanagement

### ■ Deadline Scheduling

- Earliest Deadline First (EDF): Scheduler wählt die Task, deren maximale Deadline dem Momentan-Zeitpunkt am nächsten ist.

Task	$t_{pmin}$	$t_{dmin}$	$t_{dmax}$	$t_{emin}$	$t_{emax}$
V	20 ms	0 ms	20 ms	1 ms	5 ms
GUI	40 ms	0 ms	40 ms	1 ms	15 ms
MONITORING	30 ms	0 ms	30 ms	1 ms	10 ms



## RT-OS – Taskmanagement

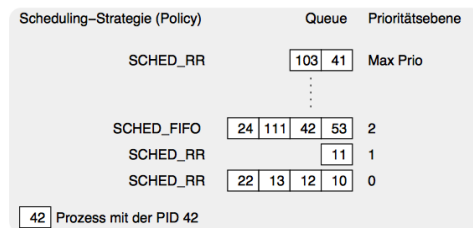
### ■ Bewertung Deadline Scheduling:

- Für Realzeit geeignet
  - Bei manchen Realzeitsystemen als Scheduler implementiert.
  - Deadline der Tasks müssen dem System bekannt gemacht werden.
  - Arbeitet (in einem Einprozessorsystem) optimal:
    - Findet immer einen korrekten Schedulingplan, sofern einer existiert (das System also nicht überlastet ist).

## RT-OS – Taskmanagement

### ■ Kombinierte Scheduling-Verfahren

- Moderne Realzeitsysteme kombinieren verschiedene Schedulingverfahren, wobei als Basis ein prioritätengesteuertes Scheduling verwendet wird.



## RT-OS – Taskmanagement

### ■ Multicore Scheduling-Verfahren

- Problem: Tasks auf mehrere Prozessoren verteilen
- Aktuelles Forschungsthema!
- Verfahren:
  - Partitioniertes Scheduling
  - Globales Scheduling
  - Semi-partitioniertes Scheduling

## RT-OS – Taskmanagement

### ■ Partitioniertes Scheduling-Verfahren

- Problem: Tasks auf mehrere Prozessoren verteilen
- Offline-Verteilung der Tasks auf die zur Verfügung stehenden CPUs
- Dafür werden Algorithmen zur Lösung von Optimierungsproblemen verwendet,
  - z.B. der »First-Fit-Decreasing-Utilization Partitioning Algorithm«.
  - Der Algorithmus verteilt die Tasks auf die Cores unter Beachtung einer von der jeweiligen Scheduling-Strategie auf dem Core abhängigen Auslastungsgrenze.
  - Eine weitere Optimierung besteht darin, möglichst alle Threads einer Threadgruppe auf einen Core zu verteilen

## RT-OS – Taskmanagement

### ■ Globales Scheduling-Verfahren

- Bei diesem Verfahren verteilt der Scheduler die Tasks auf die Kerne dynamisch,
  - weshalb Tasks auf unterschiedlichen Kernen ablaufen können.
  - Eine feste Zuordnung einer Task auf einen Kern existiert nicht.
- Theoretisch ist dadurch zwar eine 100%ige Auslastung der CPUs möglich,
  - in der Praxis verlängern die durch die Migration entstehenden Cache-Misses aber die Ausführungszeiten.

## RT-OS – Taskmanagement

### ■ Semi-partitioniertes Scheduling

- Hierbei handelt es sich um eine Mischform der beiden vorherigen Varianten.
- Gegenüber der reinen Partitionierung ist bei dieser Art die Partitionierung selbst Teil des Schedulers.
- Das bedeutet, dass der Scheduler:
  - neben der Verteilung der Tasks auf die einzelnen Prozessoren
  - auch eine Aufspaltung von Tasks zur Laufzeit vornehmen kann.

## RT-OS – Taskmanagement

### ■ Praktische Einschränkungen Multicore Scheduling

- Partitionierung ist immer dann mit hoher Wahrscheinlichkeit die beste Lösung, wenn das Taskset vorab bekannt ist
  - was bei den meisten Realzeitsystemen der Fall ist.
- Auch wenn der globale Ansatz Reserven der CPU-Auslastung durch Migration der Tasks nutzt,
  - die bei der Partitionierung unberücksichtigt bleiben,
  - ist die Auswirkung der Migration selbst auf die Ausführungszeit der Task schwierig zu bestimmen.
  - Die Modelle beziehen die Migration insgesamt zu ungenau in ihre Analysen mit ein. Daher kommt es zu sehr hohen Abweichungen der im Modell bestimmten Zeiten von den in der Realität gemessenen Zeiten.

## RT-OS – Taskmanagement

### ■ Globales Scheduling im Linux-Kernel

- Beim Booten legt Linux zunächst ein Profil der Hardware-Architektur an.
- Das ist notwendig, um die jeweiligen Taskmigrationskosten und den mit einer Taskmigration verbundenen Nutzen zu bestimmen.
- Insgesamt unterscheidet Linux hierzu drei unterschiedliche Mehrprozessorarchitekturen:
  - Simultaneous Multithreading (SMT)
  - Symmetric Multi-Processing (SMP)
  - Non-uniform Memory Architecture (NUMA)

## RT-OS – Taskmanagement

### ■ Simultaneous Multithreading (SMT)

- Diese von Intel eingeführte Architektur hat in den meisten Fällen zwei Registersätze und Pipelines, sodass sehr leicht zwischen zwei Threads umgeschaltet werden kann.
- Allerdings gibt es nur eine Verarbeitungseinheit:
  - Der Performancegewinn wird allgemein mit bis zu 10 Prozent angegeben.
  - Im strengen Sinn stellt dies allerdings kein wirkliches Multiprozessorsystem dar.



## RT-OS – Taskmanagement

### ■ Symmetric Multi-Processing (SMP)

- Mehrere Prozessorkerne nutzen gemeinsam einen physikalischen Speicher.
- Da auf einen Speicher nicht parallel zugegriffen werden kann, müssen gleichzeitige Zugriffe der Prozessorkerne sequentialisiert werden.
- Das führt zu Performanceeinbußen.

## RT-OS – Taskmanagement

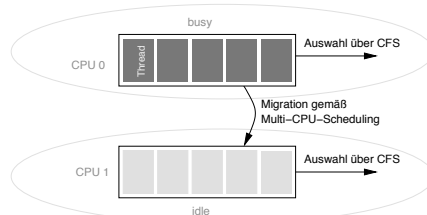
### ■ Non-uniform Memory Architecture (NUMA)

- Einzelne Prozessoren verfügen über einen eigenen, lokalen Speicher, auf den sie exklusiv zugreifen können.
- Zur Kommunikation der Prozessoren untereinander gibt es zudem meistens einen globalen, gemeinsam genutzten Speicher.
- Je nach Speicher sind die Zugriffe unterschiedlich schnell.

## RT-OS – Taskmanagement

### ■ Linux Multicore Scheduler

- Der übergeordnete Mehrprozessor-Scheduler verteilt die Tasks auf die einzelnen Prozessorkerne.
- Der Mehrprozessor-Scheduler ist häufig als eigene Task realisiert, die im Linux-Kernel beispielsweise »**Migration**« heißt.
- Das Verschieben eines Rechenprozesses von einer auf die andere CPU wird als **Taskmigration** bezeichnet.



## RT-OS – Taskmanagement

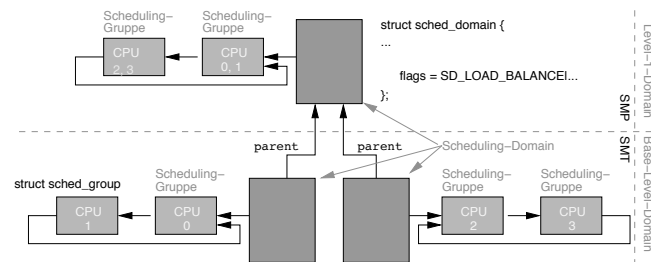
### ■ Scheduling Gruppen / Scheduling Domain

- Da in der Praxis meist gemischte Hardware-Architekturen vorkommen, führte der Linux-Kernel sogenannte Scheduling-Domains und Scheduling-Gruppen ein.
- Eine Scheduling-Domain ist der Container für (untergeordnete) Scheduling-Gruppen.
- Eine Scheduling-Gruppe wiederum steht für eine CPU oder für eine andere (untergeordnete) Scheduling-Domain.

## RT-OS – Taskmanagement

### ■ Beispiel

- Eine Architektur mit zwei Hyperthreading-Prozessoren wird mithilfe dreier Scheduling-Domains abgebildet:
- zwei Basis-Domains und die übergeordnete Level-1-Domain.



## RT-OS – Taskmanagement

### ■ Erklärung zum Beispiel (1)

- Die Level-1-Domain umspannt alle vier Prozessoren,
- die Basis-Domains jeweils einen Hyperthreading Prozessor mit seinen zwei logischen Kernen.
- Der Mehrprozessor-Scheduler sorgt für die Lastverteilung innerhalb einer Scheduling-Domain.
  - Die Entscheidung über eine Migration wird dabei abhängig von der Last innerhalb der Scheduling-Gruppen, den Kosten für die Migration und dem erwarteten Gewinn gefällt.

## RT-OS – Taskmanagement

### ■ Erklärung zum Beispiel (2)

- In Unix-Systemen kann über die Funktionen `sched_get_affinity()` und `sched_set_affinity()` die Affinität, also die Zugehörigkeit eines Threads zu einer CPU ausgelesen und festgelegt werden:
  - Damit ist also die Zuordnung eines Threads auf eine spezifische CPU möglich.
  - Durch diesen Mechanismus können Realzeitprozesse von Prozessen ohne strenge zeitliche Anforderungen separiert und damit deterministischer abgearbeitet werden.

## Realzeitbetriebssysteme (RT-OS)

1. Systemcalls
2. Taskmanagement
3. Memory Management
4. I/O Management
5. Timekeeping
6. Sonstige Realzeitaspekte



## RT-OS – Memory Management

- **Aufgaben** der Memory Management Unit (MMU) sind
  - der Speicherschutz,
  - die Adressumsetzung,
  - virtuellen Speicher und
  - erweiterten Speicher zur Verfügung stellen.

## RT-OS – Memory Management

- **Speicherschutz**
  - Applikationen (Prozesse, aber nicht Threads) werden voreinander geschützt, indem jeder Prozess seinen eigenen Adressraum bekommt.
  - Ein Zugriff ist damit nur möglich auf eigene Daten-, Stack- und Codesegmente.
  - Greift eine Applikation auf Speicherbereiche zu, die nicht zur Applikation gehören, oder versucht die Applikation, aus einem Codesegment Daten zu lesen, führt dies – dank MMU – zu einer Ausnahmebehandlung, die als Konsequenz ein Abbruch des Zugriffs zur Folge hat.

## RT-OS – Memory Management

- **Adressumsetzung**
  - Programme sollen einen einheitlichen Adressraum bekommen, um das Laden von Programmen zu beschleunigen und Shared-Libraries zu ermöglichen.
  - Aus Sicht jeder Applikation beginnt der eigene Adressraum ab der Adresse 0.
  - Mehrere Tasks können sich ein (Code-)Segment teilen.
    - Durch die Trennung von Code- und Datensegmenten und mithilfe der MMU können mehrere Prozesse, die auf dem gleichen Code beruhen, ein oder mehrere Codesegmente teilen.
    - Dadurch wird der Hauptspeicherbedarf reduziert.

## RT-OS – Memory Management

- **Virtuellen Speicher (1)**
  - Durch die MMU kann virtueller Speicher zur Verfügung gestellt werden.
  - Damit können Applikationen auf mehr Speicher zugreifen, als physikalisch vorhanden ist.
    - Als Speicherersatz wird ein Hintergrundspeicher (Festplatte) verwendet, der sogenannte Swap-Space.

## RT-OS – Memory Management

### ■ Virtuellen Speicher (2)

- Der vorhandene Hauptspeicher wird durch die Speicherverwaltung in Seiten (oder sogenannte Pages) eingeteilt. Wenn keine freien Pages mehr zur Verfügung stehen, werden die Inhalte belegter Seiten auf den Hintergrundspeicher ausgelagert und die freigeräumte Page kann genutzt werden.
  - Dieser Vorgang wird Paging oder Swapping genannt.
- In Realzeitsystemen wird Swapping nur bedingt eingesetzt, da es zu nicht-deterministischen Systemen führt.

## RT-OS – Memory Management

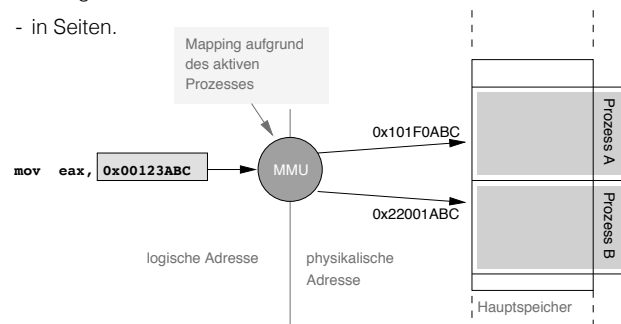
### ■ MMU (1)

- MMUs bestehen aus Hardware, die durch entsprechende Software initialisiert werden muss.
- Heutige Mikroprozessoren haben im Regelfall eine MMU integriert.
- Funktionsweise
  - Prozessorkern erzeugt eine logische Adresse
  - MMU setzt die logische Adresse in eine physikalische Adresse um.
  - Umsetzungsregeln dazu werden in die MMU geladen.
  - Eine MMU muss also durch das Betriebssystem initialisiert werden.

## RT-OS – Memory Management

### ■ MMU (2)

- Prinzipiell kann eine MMU den physikalischen Speicher auf zwei Arten aufteilen:
  - in Segmente und
  - in Seiten.



## RT-OS – Memory Management

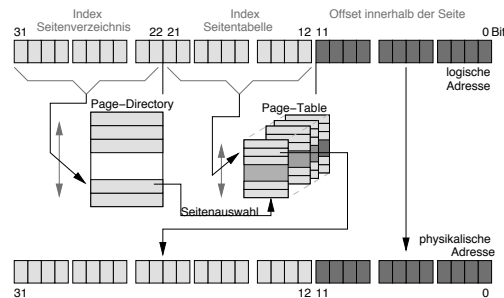
### ■ Page Table (1)

- Moderne Betriebssysteme nutzen die Segmentierung, um damit unter anderem Thread Local Storage (TLS) zu realisieren.

## RT-OS – Memory Management

### ■ Page Table (2)

- Für Speicherschutz oder auch Swapping wird dagegen vorwiegend **Paging**, also die Speicherverwaltung auf Basis von Seiten, eingesetzt.



## RT-OS – Memory Management

### ■ Realzeitaspekte der Speicherverwaltung (1)

- Applikationen, die ausgelagert sind, führen zu hohen Latenzzeiten
  - sobald der Scheduler diese auswählt, müssen erst die Speicherbereiche vom Hintergrundspeicher in den Hauptspeicher transportiert werden.
- Folglich ist das Paging am besten komplett auszustellen.
  - Alternativ muss es zumindest für die Realzeitanwendungen deaktiviert werden (Systemcall `mlock()` beziehungsweise `mlockall()`).

## RT-OS – Memory Management

### ■ Realzeitaspekte der Speicherverwaltung (2)

- Erlaubt ein Betriebssystem, auch im Kernel dynamisch Speicher zu reservieren, kann es im Laufe des Betriebs zu einer Fragmentierung des Speichers kommen.
  - In diesem Fall kann der Kernel Anfragen nach größeren, zusammenhängenden Speicherbereichen nicht mehr befriedigen. Erst nachdem andere Komponenten wieder Speicher freigegeben haben, ist das möglich.
  - Das wiederum führt zu Verzögerungen und im ungünstigsten Fall sogar zum Absturz des Systems.

## Realzeitbetriebssysteme (RT-OS)

1. Systemcalls
2. Taskmanagement
3. Memory Management
4. I/O Management
5. Timekeeping
6. Sonstige Realzeitaspekte



## RT-OS – I/O Management

### ■ Das Ein-/Ausgabe-Management erfüllt in einem Kernel die folgenden Aufgaben:

- Einheitliches Application Programming Interface (API) zur Verfügung stellen.
  - z.B. POSIX Interface
- Systemkonforme Integration von Hardware ermöglichen (Treiberinterface).
- Strukturierte Ablage von Informationen in Dateien und Verzeichnissen (Filesysteme) ermöglichen.

## RT-OS – I/O Management

### ■ Treiberinterface

- Um an der Applikationsschnittstelle ein einheitliches Interface anbieten zu können, muss ein Peripheriemodul gemäß bestimmter Konventionen (Schnittstellen) in den Betriebssystemkern eingebunden werden.
- Verwendet dann eine Applikation eine der Zugriffsfunktionen `open()`, `close()`, `read()`, `write()` oder `ioctl()`, wird im Gerätetreiber eine entsprechende Funktion aufgerufen.
- Ein Gerätetreiber ist also:
  - ein Satz von Funktionen, die den Zugriff auf eine spezifische Hardware (Gerät) steuern.
  - die gerätespezifische Implementierung der allgemeinen Schnittstellenfunktionen.

## RT-OS – I/O Management

### ■ Die Aufgabe des Betriebssystems (I/O-Management) im Kontext der Gerätetreiber ist:

- eine eindeutige Schnittstelle zur Treiberintegration zur Verfügung zu stellen,
- die Verwaltung der Ressourcen (Interrupts, Ports etc.),
- Mechanismen für die Realisierung von Zugriffsarten und Zugriffsschutz bereitzustellen,
- die Zuordnung der allgemeinen Schnittstellenfunktionen (Systemcalls) zu den gerätespezifischen Funktionen (zum Beispiel `driver_open()`) durchzuführen.

## RT-OS – I/O Management

### ■ Warum Treiber?

- Auch heute findet man noch sehr häufig Ingenieure, die Peripheriemodule nicht über einen Treiber, sondern direkt aus der Applikation heraus ansteuern. Jedoch ist ein Treiber notwendig!
- Die Gründe:
  - Um benötigte Ressourcen vom Betriebssystem zugeteilt zu bekommen.
  - Um systemkritische Teile zu kapseln.
  - Um bei Applikationsfehlern das Gerät in den sicheren Zustand überführen zu können.

## RT-OS – I/O Management

### ■ Um benötigte Ressourcen vom Betriebssystem zugeteilt zu bekommen.

- Schließlich ist das Betriebssystem für die Verwaltung der Ressourcen zuständig.
- Wird ein Gerät nicht systemkonform eingebunden, kann das Betriebssystem die zugesicherten Eigenschaften nicht mehr garantieren.
  - Es hat beispielsweise keinerlei Kontrolle darüber, ob eine Ressource (ein Interrupt, ein I/O-Bereich oder ein sonstiger Speicherbereich) bereits vergeben ist oder nicht.

## RT-OS – I/O Management

### ■ Um systemkritische Teile zu kapseln.

- Zugriffe auf Hardware sind sicherheitskritische Aktionen, die nur innerhalb eines Treibers durchgeführt werden dürfen.
- Wird aber beispielsweise die Hardware aus der Applikation heraus angesteuert (indem die Register bzw. Speicherbereiche der Hardware in den Adressraum der Applikation ein-geblendet werden), muss die Applikation erweiterte Zugriffsrechte erhalten.
- Damit wiederum gefährdet sie die Sicherheit des gesamten Systems. Programmierfehler können nicht nur zum Absturz der Task, sondern sogar zum Absturz des gesamten Systems führen.

## RT-OS – I/O Management

### ■ Um bei Applikationsfehlern das Gerät in den sicheren Zustand überführen zu können.

- Durch die eindeutige Trennung zwischen Applikation und Gerätetreiber ist es für das Betriebssystem möglich, bei Applikationsfehlern das Gerät in den sicheren Zustand zu überführen.
- Stürzt die Applikation durch einen Fehler ab, erkennt dies das Betriebssystem.
- Da es außerdem weiß, welche Applikation auf das Gerät zugegriffen hat, kann es die im Gerätetreiber befindliche Funktionalität aktivieren, um einen sicheren Gerätezustand herzustellen.

**Insbesondere dieser Grund ist bei sicherheitskritischen Realzeitsystemen entscheidend!**

## RT-OS – I/O Management

### ■ Filesystem (1)

- Filesysteme dienen zum Abspeichern bzw. Sichern von:
  - Programmen (Betriebssystemkern, Dienstprogrammen und Applikationen)
  - Konfigurationsinformationen
  - Daten (z.B. HTML-Seiten)
- auf sogenanntem Hintergrundspeicher
- Damit auf einen Hintergrundspeicher mehrere Dateien/Daten abgelegt werden können, ist eine Organisationsstruktur (Filesystem, z.B. FAT, VFAT, NTFS, Linux Ext4 Filesystem) notwendig.

## RT-OS – I/O Management

### ■ Filesystem (2)

- Diese Organisationsstruktur sollte:
  - einen schnellen Zugriff ermöglichen und
  - wenig Overhead (bezüglich Speicherplatz) für die Verwaltungsinformation benötigen.

## RT-OS – I/O Management

### ■ File Cache / Buffer

- Um einen schnellen Zugriff zu ermöglichen, arbeiten einige Systeme mit einem dazwischengeschalteten Cache, dem sogenannten Buffer- oder Pagecache.
- Möchte eine Applikation eine Datei, die auf dem Filesystem abgelegt ist, lesen, schaut der Betriebssystemkern im Buffercache nach, ob die gewünschte Information dort vorhanden ist oder nicht.
- Ist dies nicht der Fall, wird die Information in den Cache geladen und dann weiter an die Applikation gereicht.

## RT-OS – I/O Management

### ■ Realzeitaspekte bei File Cache / Buffer (1)

- Dieses Buffering ist für Realzeitsysteme oftmals nicht tolerabel:
  - Das Filesystem kann dann Inkonsistenzen enthalten, wenn das System abstürzt, der Hintergrundspeicher aber nicht mit dem Cache rechtzeitig abgeglichen worden ist. (Abhilfe: Journaling Filesystem)
  - Zugriffe auf das Filesystem sind nicht deterministisch, da nicht vorhergesagt werden kann, ob angeforderte Daten im Cache gefunden werden oder zu einem Leseauftrag an den Hintergrundspeicher führen.
  - Abhilfe: sync-Modus. Beim sync-Modus werden Dateien direkt (also unter Umgehung des Buffercache) geschrieben, ein Lesezugriff kann aber über den deutlich schnelleren Buffercache stattfinden.

## RT-OS – I/O Management

### ■ Realzeitaspekte bei File Cache / Buffer (2)

- Ebenfalls zur Performance-Steigerung wird der IO-Scheduler im Betriebssystemkern eingesetzt (Achtung: kein zeitlicher Determinismus).
  - Da Solid State Disks (SSD) keine mechanischen Elemente besitzen, sind die zurzeit eingesetzten IO-Scheduler für diese ungeeignet. Daher schaltet der Systemarchitekt das IO-Scheduling für SSDs häufig aus (unter Linux durch Auswahl des Algorithmus noop).



## Realzeitbetriebssysteme (RT-OS)

1. Systemcalls
2. Taskmanagement
3. Memory Management
4. I/O Management
5. Timekeeping
6. Sonstige Realzeitaspekte



## RT-OS – Timekeeping

### ■ Zeitverwaltung (1)

- Zeiten spielen in einem Realzeitsystem naturgemäß eine wesentliche Rolle. Ein Zeitgefühl wird für unterschiedliche Aufgaben benötigt:

#### - Zeitmessung

- Das Messen von Zeiten ist eine oft vorkommende Aufgabe in der Automatisierungstechnik. Geschwindigkeiten lassen sich beispielsweise über eine Differenzzeitmessung berechnen.

#### - Zeitsteuerung für Dienste

- Spezifische Aufgaben in einem System müssen in regelmäßigen Abständen durchgeführt werden, z.B. das Erfassen von Messwerten zu bestimmten Zeitpunkten.

## RT-OS – Timekeeping

### ■ Zeitverwaltung (2)

#### ■ Watchdog (Zeitüberwachung)

- Neben der Zeitmessung spielt die Zeitüberwachung eine sicherheitsrelevante Rolle bei Realzeitsystemen. Zum einen werden Dienste wie z.B. die Ausgabe von Daten an eine Prozessperipherie zeitüberwacht, zum anderen aber auch das gesamte System (Watchdog).

## RT-OS – Timekeeping

### ■ Realisationsformen von Zeitgebern (1)

- Das für Betriebssystem und Anwendung benötigte ‚Zeit‘ wird über Hardware- oder Software-Zähler realisiert, die periodisch getaktet inkrementieren oder dekrementieren.
- Die Software-Zähler werden typischerweise im Rahmen einer Interrupt-Service-Routine inkrementiert beziehungsweise dekrementiert.
  - Typisch 100 Hz -> 10 ms Auflösung
  - Für eine höhere Genauigkeit von Zeitoperationen, muss die Rate der Timer-Interrupts erhöht werden. Das geht allerdings zulasten der Effizienz bei leistungsschwachen Systemen (Höhere Auslastung durch ISR Verarbeitung)

## RT-OS – Timekeeping

### ■ Realisationsformen von Zeitgebern (2)

- Moderne Realzeitsysteme sind tickless. Dazu wird mit jedem Timer-Interrupt der nächste Zeitpunkt berechnet, an dem der Interrupt erneut ausgelöst werden soll. Vorteile:
  - Effizienz,
  - Energie,
  - Genauigkeit

## RT-OS – Timekeeping

### ■ Ausprägungen von Zeitgebern (1)

#### ■ Absolutzeitgeber

- Zeiten werden grundsätzlich relativ zu einem Ereignis (beispielsweise die Geburt Christi oder die sogenannte Unix-Zeit, der 1.1.1970) angegeben.
- Falls dieses Ereignis aus Sicht eines Rechnersystems außerhalb und vor allem vor der Aktivierung des Rechnersystems liegt, spricht man von einer absoluten Zeit (Uhren).

#### ■ Relativzeitgeber

- Steht das Ereignis in direkter Beziehung zum Rechnersystem, beispielsweise das Ereignis Start des Rechners, spricht man von einer relativen Zeit.
- Der zugehörige Zeitgeber ist ein Relativzeitgeber. Die Relativzeitgeber gibt es als Vorwärts- und als Rückwärtszähler (Timer).

## RT-OS – Timekeeping

### ■ Ausprägungen von Zeitgebern (2)

#### ■ Time-Stamp-Counter (TSC)

- ein Zähler, der mit der Taktfrequenz der CPU getaktet wird,
- wird genutzt, um damit das Timekeeping im OS auf eine sehr genaue Basis (Nanosekunden) zu stellen.
- Bei einem Interrupt wird die real vergangene Zeit durch Auslesen des TSC bestimmt und die internen (Software-)Zeitgeber werden entsprechend angepasst.
- Dabei gibt es allerdings das Problem der variablen Taktfrequenzen:
  - Um Energie zu sparen, werden moderne Prozessoren mit möglichst niedriger Frequenz getaktet.
  - Erst wenn mehr Leistung benötigt wird, wird auch die Taktfrequenz erhöht.
- Der Taktfrequenzwechsel muss vom Timekeeping natürlich berücksichtigt werden.

## RT-OS – Timekeeping

### ■ Problemfelder beim Umgang mit Zeiten (1)

#### ■ Zählerüberläufe

- Die verwendeten Hard- und Software-Zähler haben eine endliche Breite.
- Zur Lösung des Problems sind zum einen ausreichend große Zähler zu verwenden.
- Linux bspw. setzt ab Kernel 2.6 auf 64 Bit breite Zähler.

## RT-OS – Timekeeping

### ■ Problemfelder beim Umgang mit Zeiten (2)

#### ■ Zeitsynchronisation

- Rechner sind Teil eines Gesamtsystems, bei dem mehrere Komponenten Informationen austauschen.
- Hierbei spielt sehr häufig die Zeit eine wichtige Rolle, sodass die Systeme zeitlich synchronisiert werden.
- Ist dabei eine Zeitkorrektur notwendig, darf diese niemals sprunghaft vorgenommen werden!
- Wird die Zeit sprunghaft vorgestellt (z. B. bei der Umstellung von Winterzeit auf Sommerzeit), kommen manche Zeitpunkte nicht vor (bei der genannten Zeitumstellung bspw. die Zeit 2:30 Uhr).
- Steht zu einem ausgelassenen Zeitpunkt aber ein Zeitauftrag (Backup) an, wird dieser nicht durchgeführt und geht verloren.

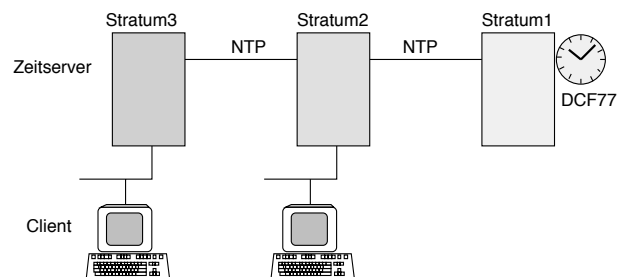
## RT-OS – Timekeeping

### ■ Zeitsynchronisation

- Zeitkorrektur niemals sprunghaft anpassen!
- Das Network Time Protocol (NTP) ist für diese Art der Zeitsynchronisation entwickelt worden.
  - Beim Network Time Protocol stellen Zeitserver die aktuelle Zeit zur Verfügung, wobei die Zeitserver klassifiziert werden.
  - Der Zeitserver, der die Uhrzeit selbst bestimmt (z.B. durch eine DCF77-Funkuhr), ist ein so- genannter Stratum-1-Server.
  - Der Server, der die Uhrzeit von einem Stratum-1-Server bezieht, ist der Stratum-2-Server usw.
  - Bei der Verteilung der Uhrzeit werden Berechnungen über die Laufzeit der Pakete zwischen den Rechnern durchgeführt und die Uhrzeit wird entsprechend korrigiert.

## RT-OS – Timekeeping

### ■ Zeitsynchronisation per NTP



## RT-OS – Timekeeping

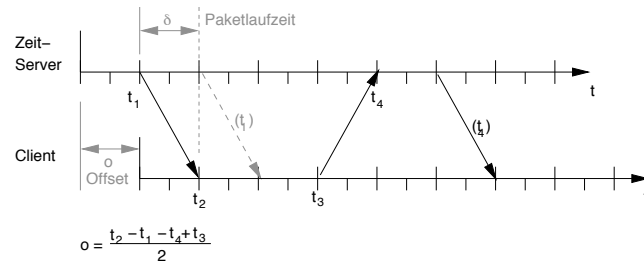
### ■ Precision Time Protocol IEEE 1588

- Das Precision Time Protocol (PTP) bietet eine Synchronisation von Zeitgebern
  - in einem lokalen Netzwerk (LAN) mit einer Genauigkeit im Mikrosekundenbereich und auch darunter (Sub-Mikrosekundenbereich)
- Eine Station im LAN ist Zeitserver, die anderen sind die Slaves.
  - Zeitserver sollte immer die Station sein, deren Uhr am genauesten ist.

## RT-OS – Timekeeping

### ■ Precision Time Protocol IEEE 1588

- Durch die umgekehrte Übertragungsrichtung der Pakete kann Offset ohne Bestimmung der Laufzeit der Pakete berechnet werden (siehe Buch).



## RT-OS – Timekeeping

### ■ Precision Time Protocol IEEE 1588

- Elimination der Laufzeit  $\delta$  zur Bestimmung des Offsets  $o$ :  
 $t_2 - t_1 = o + \delta$   
 $t_4 - t_3 = -o + \delta$   
 $\delta = t_2 - t_1 - o$   
 $\delta = t_4 - t_3 + o$   
 $t_2 - t_1 - o = t_4 - t_3 + o$   
 $o = 1/2(t_2 - t_1 - t_4 + t_3)$
- Ein Client erfasst die folgenden vier Zeitstempel:  
 $t_1 = 1000 \text{ us}$ ,  
 $t_2 = 1060 \text{ us}$ ,  
 $t_3 = 1400 \text{ us}$  und  
 $t_4 = 1456 \text{ us}$ .  
 Damit ergibt sich für den Offset  $o$  zu  $o = 1/2(1060 - 1000 - 1456 + 1400) = 4$ .

## RT-OS – Timekeeping

### ■ Precision Time Protocol IEEE 1588

- Die Zeitsynchronisation wird bis zu zehn Mal pro Sekunde zwischen den Teilnehmern durchgeführt.
- Darüber hinaus legt die Norm einen Mechanismus fest, durch den der Zeitserver, also der Rechner mit der genauesten Uhr, automatisiert identifiziert wird.

## Realzeitbetriebssysteme (RT-OS)

1. Systemcalls
2. Taskmanagement
3. Memory Management
4. I/O Management
5. Timekeeping
6. Sonstige Realzeitaspekte

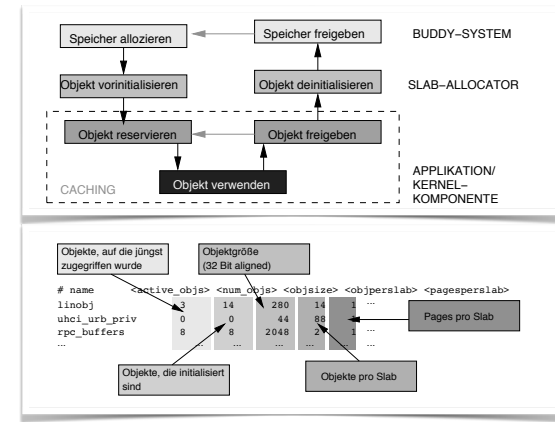


## RT-OS – Sonstige Realzeitaspekte

- Innerhalb eines Realzeitbetriebssystems wird die **Berechenbarkeit** im Wesentlichen durch zwei Aspekte beeinflusst
  - Die Berechenbarkeit der Algorithmenlaufzeit
  - Der Einsatz von Caches
- Hardware Caches eines Realzeitsystems
  - Daten- und Instruktionscache
  - Translation Lookaside Buffer (TLB)
- Software Caches der Systemsoftware
  - Page-Cache
  - Objekt-Cache (Buddy-System mit typisierten Objekten)

## RT-OS – Sonstige Realzeitaspekte

### ■ Zwischenspeicher für Kernelobjekte



## Linux



## Linux

- Linux ermöglicht ein deterministisches Zeitverhalten mit kurzen Latenzzeiten.
  - Ältere Kernel: PREEMPT-RT-Patch.
- Realtime Features:
  - Prioritätengesteuerte Scheduling, Deadline Scheduling
  - Memory-Locking,
  - Protokolle für Prioritätsinversion,
  - Threaded Interrupts,
  - CPU-Affinität.

## Linux Scheduling

- Prioritätengesteuertes (Singlecore-)Scheduling
  - 140 Prioritätsebenen in zwei Gruppen:
    - 0 bis 39 dynamischen Prioritäten,
    - 40 bis 139 statischen Prioritäten.
      - nach FCFS (First Come First Serve) oder
      - nach dem Zeitscheibenverfahren (Round Robin).
- Auf Multicore-Maschinen
  - Linux Multicore Scheduler
  - Task Migration zwischen CPU's

## Parameter Linux EDF

- EDF steht ab Linux Kernel 3.14 zur Verfügung
- Zur Parametrierung des EDF sind insgesamt vier Werte wichtig:
  - das Schedulingverfahren selbst (policy),
  - die Verarbeitungszeit im ungünstigsten Fall (worst case execution time, WCET),
  - der kürzeste Zeitraum, innerhalb dem der Rechenprozess nach der letzten Aktivierung erneut aktiviert wird (Periode)
  - die eigentliche Deadline.

## Parameter Linux EDF

- der Linux-Kernel hat das klassische EDF um eine Auslastungskontrolle (bandwidth control) ergänzt,
- Der Quotient aus WCET und Periode ergibt die Auslastung durch den jeweiligen Rechenprozess.
  - Linux stellt sicher, dass die Summe der Auslastungen der per EDF auszuwählenden Jobs in der Standardeinstellung nicht mehr als 95 Prozent der zur Verfügung stehenden Rechenzeit ist.
  - Damit bleiben mindestens fünf Prozent für Tasks übrig, die über einem anderen Schedulingalgorithmus ausgewählt werden.
  - Diese Einstellung lässt sich übrigens über die Proc-Dateien „/proc/sys/kernel/sched\_rt\_period\_us“ und „/proc/sys/kernel/sched\_rt\_runtime\_us“ ändern.
  - Um diese Auslastungskontrolle außer Betrieb zu nehmen, wird eine „-1“ in die Proc-Datei „sched\_rt\_runtime\_us“ geschrieben.

## Ausgabe der Userspace Tools

- Der Kernel dem Userland voraus.
  - Das macht sich beispielsweise bei den einschlägigen Kommandos zur Darstellung aktiver Tasks bemerkbar.
  - Diese kennen nämlich das Deadline-Scheduling noch nicht und zeigen daher für das verwendete Scheduling die kernelinterne Nummer "#6" an

```

quade 4411 2207 4412 4 TS 19 11:39 ? 00:00:00 /usr/bin/python3 /usr/share/oneconf/o
quade 4411 2207 4413 4 TS 19 11:39 ? 00:00:00 /usr/bin/python3 /usr/share/oneconf/o
quade 4411 2207 4415 4 TS 19 11:39 ? 00:00:00 /usr/bin/python3 /usr/share/oneconf/o
root 4440 2 4440 1 TS 19 11:40 ? 00:00:00 [kworker/u8:2]
root 4468 4284 4468 2 TS 19 11:41 pts/24 00:00:00 ./dl_test
root 4468 4284 4469 2 #6 140 11:41 pts/24 00:00:00 ./dl_test
quade 4470 4302 4470 1 TS 19 11:41 pts/25 00:00:00 ps -elfc

```

## Ausgabe der Userspace Tools

- Im Verzeichnis `/proc/<pid>/` die Datei „sched“
  - policy 6 = EDF; prio -1 (interne Kernel-Prio zur Verwaltung)

```

root@easentxibit7/home/quaden: Terminal Hilfe
avg_per_cpu      : 37200.352756
nr_switches      : 3720
nr_voluntary_switches : 0
nr_involuntary_switches : 3720
se.load.weight    : 1024
se.avg.runnable_avg_sum : 17618
se.avg.runnable_avg_period : 17618
se.avg.load_avg_contrib : 1023
se.avg.decay_count : 0
policy           : 6
prio             : -1
clock-delta      : 44
mm->numa_scan_seq : 0
numa_migrations, 0
numa_faults_memory, 0, 0, 1, 0, -1
numa_faults_memory, 1, 0, 0, 0, -1
root@eas-mobil:/home/quaden#

```

## Testprogramme

- Weitere Testprogramme:
  - rt-app und
  - schedtool-dl
  - Spannende Werkzeuge, um das Realzeitverhalten des Linux-Kernels auf die Probe zu stellen.
  - Beide Werkzeuge werden per git installiert.
    - git clone <https://github.com/scheduler-tools/rt-app>
    - ./autogen.sh ; ./configure --with-deadline; make
    - git clone <https://github.com/scheduler-tools/schedtool-dl.git>
    - make

## Linux Speicherverwaltung

- Basiert primär auf Paging
- Innerhalb des Kernels kann dynamisch Speicher über ein Buddy-System reserviert werden
  - Darauf aufbauend arbeitet der Slab-Allocator mit typisierten Objekten, die insbesondere bezüglich des Zeitverhaltens beim Einsatz sehr effizient sind.
- Applikationsseitig können Speicherbereiche im Hauptspeicher gelockt werden
  - Kein zeitaufwendiges Ein-/Auslagern der Seiten
  - Deterministisches Zeitverhalten
  - Ein-/Auslagern der Seiten kann auch komplett abgeschaltet werden

## Linux I/O Management

- Linux unterstützt eine Reihe von Dateisystemen
  - Beispielsweise: Dateisysteme JFFS2 oder YAFFS2 für Flash-Speicher.
- Linux hat eine sehr breite Unterstützung für unterschiedliche Hardware
  - Standardisiertes Interface zur systemkonformen Einbindung von Gerätetreibern
  - Module ermöglichen die leichte Erweiterbarkeit

## Linux I/O Management

### Unterbrechungsmodell

- Nicht nur Applikationen, sondern auch Codesequenzen des Kernels sind unterbrechbar (preemptable)
  - Kernel-Preemption bietet kurze Latenzzeiten
    - kritische Abschnitte müssen sorgfältig vor gleichzeitigem Zugriff geschützt werden
- Unterbrechungsmodell von Linux hat 4 Ebenen:
  - Applikationsebene (Userland), Kernel-Ebene, Soft-IRQ-Ebene, Interrupt-Ebene (ISR's)
  - Mithilfe Threaded Interrupts Reduktion auf drei Ebenen
    - Threads der eigenen Realzeitanwendung können vor aktivierten Interrupt-Service-Routinen ablaufen.

## Linux Zeitverwaltung

- Bei Linux handelt es sich um ein Tickless-System
  - je nach Situation wird der nächsten Unterbrechungszeitpunkt ausgerechnet
- Watchdogs werden unterstützt

## Sonstige Realzeitaspekte Linux

- Userland.
  - Je nach Anwendungsfall stehen für den Linux-Kernel verschiedene Userlands zur Verfügung.
    - Busybox
    - Sehr verbreitet im embedded Bereich
    - damit ist ein komplettes Userland möglich
    - Modul 'Systemsoftware'
- Moderne Technologien für eine hohe IT-Security

