

Vorlesung  
**Moderne  
Realzeitsysteme**


**Realzeitarchitekturen**

Professor Dr. Michael Mächtel

Realzeitarchitekturen

## Realzeitarchitekturen

- 1. Grundstrukturen**
- 2. Entwurf moderner Realzeitsysteme**



Professor Dr. Michael Mächtel 2

Realzeitarchitekturen

## Realzeitarchitekturen

- 1. Grundstrukturen**
- 2. Entwurf moderner Realzeitsysteme**



Professor Dr. Michael Mächtel 3

Realzeitarchitekturen

## Systemsoftware

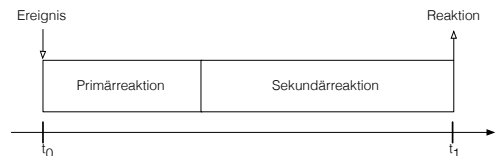
- Zentrales Element beim Aufbau eines Realzeitsystems ist die Systemsoftware, die die folgenden Grundstrukturen für den Aufbau moderner Realzeitsysteme vorgibt:
  - Realzeitsystem ohne spezielle Systemsoftware,
  - Realzeitsystem basierend auf Standard-Systemsoftware,
  - Realzeitsystem, das dediziert eingebaute Realzeiteigenschaften nutzt (Threaded Interrupts),
  - Realzeitsystem, bei dem Teile der Applikation in den Kernel verlagert werden (Userland-to-Kernel),
  - System mit einem Realzeitkernel auf Singlecore-Basis,
  - Realzeitsystem auf Multicore-Basis (Multikern-Ansatz),
  - System, das auf die Kombination von Realzeit-Betriebssystem und Standardbetriebssystem setzt (Multikernel-Ansatz).

Professor Dr. Michael Mächtel 4

## Einfluss der Systemsoftware...

### ... auf die Reaktionszeit

- Die Realzeitsteuerung verarbeitet Rechenzeitanforderungen fast immer durch eine
  - Primärreaktion und eine
  - Sekundärreaktion



## Primärreaktion

- Die Primärreaktion ist meist in Form einer Interrupt-Service-Routine ausgeprägt,
  - die aufgrund einer Rechenzeitanforderung aufgerufen wird.
- Dadurch hat sie eine kurze Latenzzeit.
- Im Rahmen der ISR wird zumindest der Interrupt quittiert und die Sekundärreaktion aktiviert.
- Zeitkritische Aufgabenteile werden ebenfalls während der Primärreaktion abgewickelt.

## Sekundärreaktion

- Die Sekundärreaktion ist typischerweise als Task ausgeprägt, je nach Realzeitarchitektur manchmal auch als 'Tasklet' (z.B. Linux).
  - Das Tasklet ist Teil der Systemsoftware.
- In der Sekundärreaktion werden die zeitlich weniger kritischen Aufgaben abgearbeitet.

## Der Systemarchitekt

- verteilt die Aufgaben, die aufgrund einer Rechenzeitanforderung anfallen, auf die beiden Teile.
- Die Erledigung von Aufgaben in der Primärreaktion (ISR) führt zu einer kurzen Reaktionszeit für die zugehörige Rechenzeitanforderung.
- Allerdings werden dadurch die allgemeinen Latenzzeiten schlechter.
- Tendenziell versucht der Architekt daher, die Primärreaktion auf das absolute Minimum zu beschränken und die eigentlichen Arbeiten in der Sekundärreaktion durchzuführen.

## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. Basierend auf Standard-Systemsoftware
3. Nutzen von dediziert eingebaute Realzeiteigenschaften
4. Applikation in den Kernel verlagern
5. Realzeitkernel auf Singlecore-Basis
6. Realzeitkernel auf Multicore-Basis
7. Multikernel-Ansatz



## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. Basierend auf Standard-Systemsoftware
3. Nutzen von dediziert eingebaute Realzeiteigenschaften
4. Applikation in den Kernel verlagern
5. Realzeitkernel auf Singlecore-Basis
6. Realzeitkernel auf Multicore-Basis
7. Multikernel-Ansatz



## Realzeitsystem...

### ... ohne spezielle Systemsoftware

- Bei dieser Architekturvariante setzt die Applikation ohne Zwischenschicht **direkt** auf der Hardware auf.
- Sie kümmert sich damit selbst um die Prozessorinitialisierung und die Bedienung sämtlicher Interrupts.
- Ohne Systemsoftware gibt es auch **kein Multitasking!**
  - Daher wird die Applikation singlethreaded entworfen.
  - Allerdings wird beim Softwareentwurf Parallelverarbeitung häufig durch Implementierung unabhängiger Verarbeitungsstränge in Interrupt-Service-Routinen eingeplant (**Poor Man's Multithreading**).

## Realzeitsystem...

### ... ohne spezielle Systemsoftware

- Das **Zeitverhalten** wird allein durch die Applikation bestimmt und ist damit in weiten Teilen berechenbar.
- **Latenzzeiten** ergeben sich im Wesentlichen durch die Länge der Interrupt-Service-Routinen (Interrupt-Latenzzeit).
- Diese sind wiederum von der Aufteilung der Reaktion in Primär- und Sekundärreaktion durch den Entwickler abhängig.

## Realzeitsystem...

### ... ohne spezielle Systemsoftware

- **Sie wählen diese Architektur**, wenn Ihr Realzeitsystem aus einem einfachen Mikrocontroller (Singlecore) aufgebaut wird.
- Dieser Mikrocontroller hat typischerweise eine Verarbeitungsbreite von 8 Bit, manchmal auch von 16 Bit.
- Die Aufgabenstellung selbst ist nicht komplex, oft handelt es sich um nicht vernetzte Systeme.
- Die zeitlichen Anforderungen können sowohl weich als auch hart sein, aufgrund der meist schwachen Prozessorleistung liegen sie im zwei- bis dreistelligen Millisekundenbereich.
- Kürzere Zeitanforderungen lassen sich auch einhalten, wenn Teile der Software in Hardware ausgelagert werden, beispielsweise über ein Field Programmable Gate Array (FPGA).

## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. **Basierend auf Standard-Systemsoftware**
3. Nutzen von dediziert eingebaute Realzeiteigenschaften
4. Applikation in den Kernel verlagern
5. Realzeitkernel auf Singlecore-Basis
6. Realzeitkernel auf Multicore-Basis
7. Multikernel-Ansatz



## Realzeitsystem...

### ... basierend auf Standardbetriebssystem

- Sind die zeitlichen Anforderungen weich oder im **dreistelligen Millisekundenbereich**, kann ein Standardbetriebssystem eingesetzt werden.
- Der Vorteil dieser Realzeitarchitektur besteht darin, dass der Entwickler bereits mit seinem Betriebssystem, mit der Entwicklungsumgebung und mit den Schnittstellen vertraut ist.
- Es stehen viele Bibliotheken mit vorgefertigten und auch komplexen Routinen zur Verfügung.
- Standardbetriebssysteme unterstützen häufig aktuelle und moderne Funktionalitäten.

## Realzeitsystem...

### ... basierend auf Standardbetriebssystem

- Unter Umständen beeinflussen lange Interrupt-Service-Routinen von Standard-Hardware-Komponenten die Reaktionszeiten der Tasks (Sekundärreaktion).
- Realzeitsysteme, die auf Standardbetriebssystemen beruhen, sind allein aufgrund von deren Verbreitung Angriffen ausgesetzt.
- **Sie wählen diese Architektur** bei weichen oder harten Realzeitanforderungen im dreistelligen Millisekundenbereich, falls eine Standard-Hardware (häufig eine PC-Plattform) zur Verfügung steht.

## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. Basierend auf Standard-Systemsoftware
3. **Nutzen von dediziert eingebaute Realzeiteigenschaften**
4. Applikation in den Kernel verlagern
5. Realzeitkernel auf Singlecore-Basis
6. Realzeitkernel auf Multicore-Basis
7. Multikernel-Ansatz



## Threaded Interrupts

### Realzeiterweiterungen für Standard-OS

- Standardbetriebssysteme bieten mehrere Möglichkeiten, das Zeitverhalten von Applikationen deterministischer zu gestalten.
  - prioritätengesteuerten Scheduling,
  - die Vergabe von Realzeitprioritäten
  - und das Ausschalten von Swapping.

## Threaded Interrupts

### Realzeiterweiterungen für Standard-OS

- Eine der sehr nützlichen Erweiterungen stellen **Threaded Interrupts** dar.
  - Bei dieser Technik werden Interrupt-Service-Routinen in zwei Teile geteilt,
    - in eine Basis-ISR und
    - einen Thread.
  - Die Basis-ISR ist für alle Interrupts identisch und versetzt den zugeordneten, hochprioritären Thread in den Zustand lauffähig.
  - Damit wird eine sehr kurze Primärreaktion erreicht. Die tatsächliche Reaktion wird mit der Sekundärreaktion abgehandelt.

## Threaded Interrupts

### Realzeiterweiterungen für Standard-OS

- Beim Entwurf müssen nicht nur die eigenen Threads, sondern eben auch die sonstigen Systemthreads (ISRs, Timer, Tasklets) berücksichtigt werden.
- Die Verteilung der Prioritäten ist so zu gestalten, dass nicht nur die Realzeitbedingungen für die erstellte Realzeitapplikation, sondern auch diejenigen für die Systemthreads eingehalten werden.
  - Das ist nicht trivial, da man die minimal und maximal zulässigen Reaktionszeiten der Systemthreads ebenso wenig kennt wie deren Laufzeiten.
  - Die Laufzeiten wiederum werden benötigt, um die sich ergebenden Verzögerungen zu bestimmen.

## Threaded Interrupts

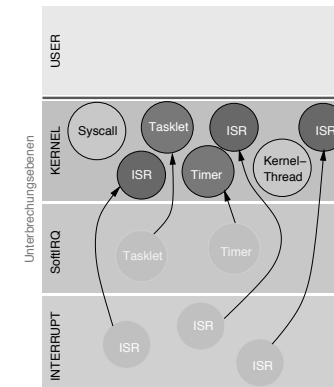
### Realzeiterweiterungen für Standard-OS

- Der eigenen Realzeitapplikation sollte demnach auch nur dann eine höhere Priorität zugeteilt werden, falls deren Verarbeitungszeit kurz ist. Diese geht dann auf Einprozessorsystemen als Verdrängungszeit (Wartezeit) in die Reaktionszeit der Systemthreads ein.

## Threaded Interrupts

### Realzeiterweiterungen für Standard-OS

#### Beispiel Linux



## Threaded Interrupts: Bewertung

### Realzeiterweiterungen für Standard-OS

#### ■ Diese Architektur ist zu wählen:

- wenn ausreichend Rechnerleistung zur Verfügung steht und
- die Anforderungen an das Zeitverhalten im unteren Millisekundenbereich, eventuell niedriger, liegen.
- Die Architektur bietet sich vor allem auch auf Singlecore-Maschinen an.

## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. Basierend auf Standard-Systemsoftware
3. Nutzen von dediziert eingebaute Realzeiteigenschaften
4. Applikation in den Kernel verlagern
5. Realzeitkernel auf Singlecore-Basis
6. Realzeitkernel auf Multicore-Basis
7. Multikernel-Ansatz





## Userland to Kernel

- Wenn die zeitlichen Anforderungen auf Applikationsebene (im Userland) nicht eingehalten werden können, kann der Systemarchitekt überlegen, zeitkritische Teile der Applikation (Sekundärreaktion) in den Kernel zu verlagern.
- Allein bedingt durch den Wegfall beziehungsweise die Verkürzung der Kontextwechselzeiten sind die Latenzzeiten kürzer und der Overhead geringer.
- Im Wesentlichen können Codesequenzen in folgende Kernel-Komponenten verlagert werden:
  - Interrupt-Service-Routinen
  - Soft-IRQs
  - Kernel-Threads
  - Gerätetreiber

## Userland to Kernel

- Die Verarbeitungszeit der ISR (Primärreaktion) darf dabei nicht zu lang werden, verlängert diese doch direkt die Latenzzeiten der übrigen Systemteile, insbesondere auch der übrigen Interrupt-Service-Routinen.
- Eine Verlagerung in Soft-IRQs führt dazu, dass die Codesequenzen direkt nach aktivierten Interrupt-Service-Routinen ablaufen.
  - Auf dieser Ebene lassen sich auch relativ leicht zyklisch abzuarbeitende Programmteile realisieren.
- Die **einfachste** Portierung von Teilen der Applikation wird über priorisierbare **Kernel-Threads** erreicht.
- Zeitkritische Teile der Applikation werden dann sinnvoll in Gerätetreiberfunktionen integriert, wenn Realzeitapplikation und Gerätetreiber ohnehin zueinander gehören.

## Userland to Kernel

- Um diese Realzeitarchitektur zu verwirklichen, ist für den Entwickler eine Einarbeitung in die Kernel-Programmierung notwendig.
- Die Dienste des Kernels (Systemcalls) können genutzt werden, sind aber anders benannt und die Parametrierung unterscheidet sich im Detail von den Pendanten des Userlands.
- Codesequenzen der Interrupt-Ebene und SoftIRQ-Ebene sind weiter eingeschränkt: Funktionen beziehungsweise Funktionalitäten wie das Schlafenlegen sind tabu.

## Userland to Kernel

- Die Verlagerung der Software in den Kernel führt zu einem **nicht portierbaren** und vor allem stark auf das System abgestimmten Design führt.
  - Es besteht keine saubere Trennung mehr zwischen Applikation und Betriebssystemkern.
  - Es besteht die Gefahr von Systemabstürzen durch Programmierfehler, die sich im Kernel ungeschützt bemerkbar machen können.
- Mithilfe dieses Ansatzes lässt sich häufig der Einsatz eines (proprietären) Realzeit-OS vermeiden.

## Userland to Kernel: Bewertung

- **Sie wählen diese Architektur**, wenn Sie
  - harte Realzeitanforderungen im unteren Milli- oder im Mikrosekundenbereich erfüllen müssen,
  - diese im Userland nicht einhalten können und
  - andere Methoden, wie beispielsweise Threaded Interrupts, nicht ausreichen.
- Die Architektur ist geeignet für Single- und auch für Multicore-Maschinen.

## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. Basierend auf Standard-Systemsoftware
3. Nutzen von dediziert eingebaute Realzeiteigenschaften
4. Applikation in den Kernel verlagern
5. **Realzeitkernel auf Singlecore-Basis**
6. Realzeitkernel auf Multicore-Basis
7. Multikernel-Ansatz



## Realzeit-OS

- ermöglicht – abhängig von der verwendeten Hardware – Zeitanforderungen im einstelligen Millisekundenbereich einzuhalten.
- Dazu bietet der Betriebssystemkern typischerweise
  - ein prioritätengesteuertes Scheduling.
  - Threads als kleinste Scheduling-Einheit
  - kleinen Footprint
  - Im Idealfall ist die Interrupt-Latenzzeit allein durch die Hardware bestimmt.
    - Keine Interruptsperr im Kernel!

## Realzeit-OS: Vor- / Nachteile

- **Vorteil:** Durch die im Vergleich zu einem Standardsystem geringe Verbreitung ist es seltener Angriffen von ‚Script-Kiddies‘ ausgesetzt.
- **Nachteil:** hohe Einarbeitungszeit



## Realzeit-OS: Komponenten

- Ein klassisches Realzeitbetriebssystem besteht aus mehreren Komponenten:
  - Userland mit Bibliotheken und wichtige Systemprogramme
  - Kernel +
  - Board Support Package (BSP).
    - Das BSP enthält alle notwendigen Anpassungen, damit das System auf einer bestimmten Hardware ablauffähig ist.
- Realzeitbetriebssysteme sind nicht immer topaktuell

## Realzeit-OS: Einsatz

- Einige der Realzeitbetriebssysteme bieten ähnliche Möglichkeiten wie Linux,
  - z.B. die Primär- und Sekundärreaktion in unterschiedlichen Systemschichten auszuführen.
- **Sie wählen diese Architektur,**
  - wenn Sie harte Realzeitanforderungen, typischerweise kombiniert mit einer proprietären, auf die Anforderungen hin optimierten Hardware einsetzen wollen.

## Realzeitbetriebssystem: VxWorks

- klassisches Realzeitbetriebssystem mit einer starken Marktdurchdringung
- ursprünglich als Realzeitbetriebssystem für einfache (16-Bit-)Mikrocontroller entwickelt, inzwischen auch 32- und 64-Bit-Prozessoren.
- skalierbares, deterministisches, zuverlässiges Realzeitbetriebssystem:
  - mit breiten Hardware-Support
  - Der Mikrokern benötigt wenige Ressourcen,
  - dank guter Portierbarkeit für unterschiedliche Plattformen verfügbar

## Realzeitbetriebssystem: VxWorks

- Die Wind River Workbench genannte Entwicklungsumgebung auf dem Host basiert auf Eclipse.
- Varianten:
  - MILS: Fokus auf Security
  - CERT: Fokus auf Safety.

## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. Basierend auf Standard-Systemsoftware
3. Nutzen von dediziert eingebaute Realzeiteigenschaften
4. Applikation in den Kernel verlagern
5. Realzeitkernel auf Singlecore-Basis
6. Realzeitkernel auf Multicore-Basis
7. Multikernel-Ansatz



## Realzeitarchitektur auf Multicore Basis

- Prinzip: eine oder mehrere CPUs werden alleine für die Realzeitaufgaben reserviert:
  - bereits ein moderner Linux-Kernel bietet diese Möglichkeit an.
- Aufgabe des Systemarchitekten ist es :
  - die abzuarbeitenden Tasks in Realzeit- und Nichtrealzeittasks zu klassifizieren
  - die zeitlichen Parameter der Prozesse und die sich ergebende Gesamtauslastung bestimmt werden.
  - die Gesamtauslastung muss gemäß erster Realzeitbedingung unterhalb von:
    - $\text{Cores} * 100\%$  liegen.
- Interrupts müssen einzelnen CPU-Kernen exklusiv zugewiesen werden.
- Das muss jedoch durch die Firmware (Bios) unterstützt werden.

## Realzeitarchitektur auf Multicore Basis

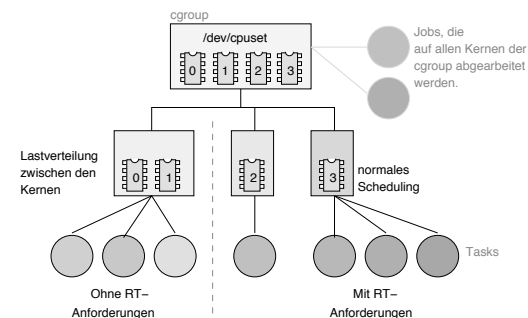
### Tasks aufteilen

- Jedem Rechenprozess kann eine Affinität mitgegeben werden.
- Verteilung der Tasks auf die unterschiedlichen Cores:
  - Alle Tasks, die keine zeitlichen Anforderungen haben, werden auf CPU 0 verlagert.
    - Die Realzeitprozesse werden auf den übrigen Prozessoren abgearbeitet, wobei der Scheduler selbst die Verteilung auf die einzelnen Prozessoren vornimmt.
- Alternativ dazu kann der Systemarchitekt die Rechenprozesse direkt den einzelnen Rechnerkernen zuordnen.
- In jedem Fall muss nach der Verteilung auf die Prozessoren für jeden Prozessor beziehungsweise Prozessorkern getrennt ein Realzeitnachweis geführt werden.

## Realzeitarchitektur auf Multicore Basis

### Linux ,cpuset'

- Der Admin gruppiert über ein virtuelles Filesystem (type cpuset) sowohl Prozessoren als auch Speicherressourcen (Memory Nodes) und verteilt auf diese die Tasks.



## Realzeitarchitektur auf Multicore Basis

### Interrupts aufteilen

- Typischerweise wird die Boot-CPU (Nummer 0) als die CPU eingeplant, die nicht Realzeitprozesse abarbeitet,
  - denn nicht bei jeder Hardware lassen sich Interrupts beliebig auf unterschiedliche Prozessorkerne verteilen.
- Kernel-Programmierer kann beim Einhängen der Interrupt-Service-Routine festlegen, dass die ISR nicht »balanced«, also auf unterschiedlichen Rechnerkernen abgearbeitet wird.
  - Vom Userland aus kann das Attribut ‚mask‘ über das Proc-Filesystem manipuliert werden.

## Realzeitarchitektur auf Multicore Basis

### Interrupts aufteilen

- Beispiel, wie auf einer Vierkern-Maschine (`smp_affinity=0x0f`) die CPU-0 von der Verarbeitung des Interrupts 1 ausgenommen (`smp_affinity=0x0e`) wird:

```
root@lab01:~# cat irq/1/smp_affinity
```

```
0f
```

```
root@lab01:~# echo "0x0e"> irq/1/smp_affinity
```

```
root@lab01:~# cat irq/1/smp_affinity
```

```
0e
```

## Realzeitarchitektur auf Multicore Basis

### SoftIRQs, Tasklet, Timer

- Typischerweise werden Softirqs und Tasklets auf der CPU abgearbeitet, auf der die Interrupt-Service-Routine angestoßen wurde.
  - Hier muss der Systemarchitekt sorgfältig die IRQ-Affinität für einen mit harten Realzeitanforderungen versehenen Prozess auf die für Realzeitaufgaben reservierte CPU legen.
- Da Interrupts heutzutage allerdings gemeinsam (unterschiedliche Hardware löst den gleichen Interrupt aus) genutzt werden, muss der Konstrukteur dafür sorgen, dass die zugehörige Interrupt-Leitung exklusiv nur von dem einen Kern genutzt wird.
  - In solch einem Fall ist ein Eingriff in das PCI-Handling erforderlich, was die Portabilität erschwert.

## Realzeitarchitektur auf Multicore Basis

### Fazit

- Diese moderne Realzeitarchitektur setzt eine Multicore-Hardware voraus.
- Sie ermöglicht im Userland, wieder abhängig von der Leistungsfähigkeit der eingesetzten Hardware, Task-Latenzzeiten im unteren Millisekunden- oder im dreistelligen Mikrosekundenbereich.
- Der Systementwurf, also die Einteilung in RT- und Nicht-RT-Prozessoren und die Verteilung der Threads auf die einzelnen Bereiche, ist komplex und zurzeit noch Handarbeit.

## Grundstrukturen

1. Ohne spezielle Systemsoftware
2. Basierend auf Standard-Systemsoftware
3. Nutzen von dediziert eingebaute Realzeiteigenschaften
4. Applikation in den Kernel verlagern
5. Realzeitkernel auf Singlecore-Basis
6. Realzeitkernel auf Multicore-Basis
7. **Multikernel-Ansatz**



## MultiKernel-Architektur

- Hierbei handelt es sich um eine Form der Virtualisierung, wobei der Realzeitkernel den Hypervisor darstellt.
- Der Realzeitkernel kontrolliert insbesondere das Sperren und Freigeben von Interrupts.
- Das Verfahren ist in der Implementierung zwar vergleichsweise aufwandsarm, nachteilig dabei ist aber, dass sämtliche Tasks, die unter dem Standardbetriebssystem laufen, nicht realzeitfähig sind.
- Insbesondere ist es nicht möglich, dass eine Task des Realzeitkerns eine Komponente des Standardbetriebssystems nutzt (z.B. TCP/IP), ohne dass diese Realzeit-Task ihre Echtzeitfähigkeit (ihren Determinismus) verliert.
- Implementierungen finden sich sowohl für Linux (RT-Linux, RTAI, Xenomai) und Varianten des Windows-Betriebssystems.

## RT-Linux

- RT-Linux ist am Department of Computer Science der Universität New Mexico entwickelt worden.
- RT-Linux selbst ist zunächst ein kleiner Realzeitkern, der Linux in einer eigenen Task ablaufen lässt.
  - Linux ist dabei die Idletask, die nur dann aufgerufen wird, wenn keine andere Realzeit-Task lauffähig ist.
- Jeder Versuch der Linux-Task, Interrupts zu sperren (um selbst nicht unterbrochen zu werden), wird vom Realzeitkern abgefangen.

## RT-Linux: Interrupt-Bearbeitung

- RT-Linux emuliert die komplette Interrupt-Hardware:
  - Versucht Linux selbst, einen Interrupt zu sperren, so merkt sich der Realzeitkern dies, sperrt den Interrupt jedoch nicht.
  - Tritt nun der Interrupt auf, wird vom Realzeitkern entweder ein Realzeit-Interrupt-Handler (Realtime Handler) aufgerufen oder der Interrupt wird als pending markiert.
  - Gibt die Linux-Task Interrupts wieder frei, emuliert RT-Linux den als pending markierten Interrupt für die Linux-Task.
  - Damit können Interrupts die Linux-Task in jedem Zustand unterbrechen und Linux als solches ist nicht in der Lage, irgendwelche Latenzzeiten zu verursachen.

## RT-Linux: Übersicht

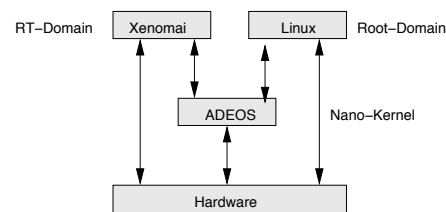
- Informationsaustausch zwischen den beiden Systemen in RT-Linux:
  - Real-Time-Fifos und
  - Shared-Memory

## RT-Linux: Bewertung

- Die Variante RT-Linux ist mit einem Patent versehen, sodass der Einsatz der Software bedenklich ist.
- Alternativ wird daher heute auf die Lösungen RTAI (Real Time Application Interface) oder Xenomai gesetzt, die dieses Problem nach derzeitigem Stand nicht mitbringen.
- Beide Varianten setzen primär auf eine Modifikation des Linux-Kernels namens Adeos.

## ADEOS

- Adeos/ipipe (Adaptive Domain Environment for Operating Systems) ermöglicht als sogenannter Nano-Kernel mehreren Betriebssystemen, die gleiche Hardware zu nutzen
- Basisbegriffe sind:
  - Domains und die
  - Interrupt- beziehungsweise Event-Pipeline (I-pipe).



## ADEOS: Domain und I-Pipeline

- Jedes Betriebssystem – im Realzeitumfeld sind dies typischerweise ein Realzeitkernel wie Xenomai und ein Standard-Linux – wird als Domain geführt:
  - Die Root-Domain ist Linux
  - Jede Domain ist priorisiert, wobei die Root-Domain nicht die höchste Priorität haben muss.

## ADEOS: Domain und I-Pipeline

- Für die Bearbeitung von Interrupts und Events stellt Adeos eine Interrupt-Pipe zur Verfügung (I-pipe):
  - In dieser Pipe reihen sich die Domains gemäß ihrer Priorität ein.
  - Ein Realzeitkernel wie RTAI oder Xenomai hat typischerweise sehr hohe Priorität und bekommt daher den Interrupt als Erstes zugestellt.
  - Ist der Interrupt irrelevant für die jeweilige Domain, wird er von Adeos an die nächste Stufe der Pipeline weitergereicht

## RTAI und XENOMAI

- Die beiden Realzeitkernel RTAI und Xenomai nutzen Adeos, um neben dem Standard-Linux das Einhalten harter Realzeitanforderungen zu garantieren.
- Dabei unterscheiden sich die beiden Lösungen bezüglich ihrer Projektziele, der Interfaces und der unterstützten Hardware-Plattformen.
  - Primäres Ziel unter RTAI ist es, möglichst kurze Latenzzeiten zu erreichen.
  - Xenomai demgegenüber steht für hohe Portabilität. Entsprechend sind auch die unterstützten Schnittstellen vielfältiger.

## RTAI und XENOMAI

- Während RTAI nur die beiden Interfaces RTDM (Realtime Driver Model) und POSIX 1003.1b unterstützt, hat Xenomai neben dem eigenen, nativen API noch sogenannte Skins für VxWorks, PSoS+ und Vrtx implementiert.
- Damit können Realzeitapplikationen, die beispielsweise für VxWorks geschrieben sind, mit geringem Aufwand unter Linux ablaufen.

## Multi-Kernel Architektur: Bewertung

- **Vorteil:**
  - harte Realzeitanforderungen im Mikrosekundenbereich können erfüllt werden
  - Auch wenn Multicore-Plattformen unterstützt werden, wird die Lösung primär für Singlecore-Anwendungen eingesetzt.
  - Die Verknüpfung mit dem Standardsystem ermöglicht es dem Anwender, bekannte Benutzerschnittstellen / Oberflächen zu präsentieren und Realzeitdaten in normale Anwendungen (beispielsweise Office) zu integrieren.



## Multi-Kernel Architektur: Bewertung

### ■ **Nachteil:**

- gleich mehrere Betriebssysteme müssen installiert, konfiguriert, aufeinander abgestimmt und vor allem gepflegt werden.
- Je nach eingesetztem Betriebssystem und Version ist das eingesetzte Standardbetriebssystem weiterhin nicht realzeitfähig.
- Der Entwickler muss sich in zwei Systeme und deren Interfaces einarbeiten.
  - Er teilt die zu lösende Aufgabe in die zeitlich kritischen und in die zeitlich unkritischen Bereiche ein.

## Realzeitarchitekturen

### 1. Grundstrukturen

### 2. Entwurf moderner Realzeitsysteme



## Entwurf moderner Realzeitsysteme

- Ähnelt in seiner Struktur zunächst dem Entwurf eines Systems ohne zeitliche Anforderungen:
  - typischerweise V-Modell oder Spiralmodell, bei denen die einzelne Entwurfsphasen immer wieder in neuer Verfeinerung durchlaufen werden.
- **Unterschiede** ergeben sich aufgrund der zusätzlichen, nicht funktionalen Anforderungen:
  - Realzeit, Safety, Security, Reliability,
  - Hardware/Software Codesign : der oftmals notwendigen Parallelentwicklung von Hard- und Software.
  - Erschwernissen des Debuggens und Testens eingebetteter, zeitkritischer Software.

## Anforderungen

- Die Anforderungen werden für die Anforderungsspezifikation (Requirement Specification) zusammengetragen.
- Neben den funktionalen Anforderungen müssen im Realzeitbereich die **zeitlichen Anforderungen** erfasst werden.
  - Für jede Rechenzeitanforderung:
    - die minimal und die maximal zulässige Reaktionszeit ( $t_{Dmin}$  und  $t_{Dmax}$ ),
    - die maximale Auftrittshäufigkeit (minimale Prozesszeit  $t_{Pmin}$ ) und
    - die zeitlichen Abhängigkeiten untereinander, die durch die  $t_A$  ausgedrückt werden.
- In der Kategorie sonstige Anforderungen sind im Realzeitumfeld vor allem **Anforderungen an Safety und an Security** zu definieren.

## Entwurf allgemein (1)

- Zur Realisierung der **funktionalen** und **zeitlichen** Anforderungen sind die benötigten Hard- und Software-Komponenten auszuwählen beziehungsweise zu entwerfen.
- Hierzu ist es sinnvoll, zunächst **Funktionsblöcke** zu definieren, die mit den funktionalen Anforderungen korrespondieren und denen dann zeitliche Anforderungen zugeordnet werden. Dabei sind bereits einige Entwurfsprinzipien zu beachten:
  - Trennung von Oberfläche und Abläufen
  - Trennung der zeitkritischen von den nicht zeitkritischen Abläufen.
  - Berücksichtigung von Sicherheitsaspekten, beispielsweise das Prinzip, einzelnen Codesequenzen nur die Privilegien zuzuordnen, die sie auch benötigen.

## Entwurf allgemein (2)

- Entwurfsschritte:
  - Entwurf der Hardware und der Software
  - Entwurf der Realzeitarchitektur
  - Realzeitnachweis durchführen
  - Realisierung
  - Test und Dokumentation

## Entwurf Hardware (1)

- Anhand der Funktionsblöcke kann die Hardware ausgewählt werden.
- Dabei schätzt der Entwickler bereits grob ab, ob die Funktionsblöcke in Realzeit verarbeitet werden können oder nicht.
- Ist das für einzelne Blöcke beispielsweise nicht möglich, kann über eine Realisierung des Blocks in Form von Hardware, beispielsweise von Field Programmable Gate Arrays (FPGA), nachgedacht werden.
  - Alternativ muss eine generell leistungsfähigere Hardware in Betracht gezogen werden.

## Entwurf Hardware (2)

- Für die Auswahl der Hardware ist darüber hinaus die Ankopplung von Peripherie relevant.
  - Gerade im Bereich der eingebetteten Systeme werden häufig Feldbusse wie beispielsweise CAN (Controller Area Network) eingesetzt.

## Entwurf Software (1)

- Hand in Hand mit der Hardware wird die Systemsoftware ausgewählt:
  - Unter Umständen ist zudem eine Portierung der Systemsoftware einzuplanen:
    - Firmware, Bootloader, Betriebssystem
  - Je nach Plattform sind ein Bootloader und das Betriebssystem auszuwählen.

## Entwurf Software (2)

- Moderne Realzeitsysteme setzen auf Multithreading:
  - Die bereits identifizierten Funktionsblöcke werden demnach auf möglichst jeweils einzelne Threads abgebildet.
  - Dadurch ergibt sich Inter-Thread-Kommunikation, die zu planen ist:
    - Welche Daten werden wann zwischen welchen Einheiten wie ausgetauscht?
    - Dem Schutz der sich dabei ergebenden kritischen Abschnitte ist besondere Aufmerksamkeit zu schenken.

## Entwurf Architektur (1)

- Auf Basis der bisherigen Ergebnisse ist die Realzeitarchitektur zu entwerfen.
- Dabei muss nicht zwangsläufig eine der erwähnten Architekturen in Reinform vorkommen, sondern es sind durchaus Kombinationen denkbar:
  - beispielsweise ein Multicore-Ansatz kombiniert mit Threaded Interrupts.
- Im nächsten Schritt werden die Threads auf Prozessorkerne verteilt und systemweit Prioritäten vergeben.

## Entwurf Architektur (2)

- Anschließend können die Verarbeitungszeiten abgeschätzt oder aber ausgemessen werden.
- Idealerweise werden die Thread-Gruppen so auf die Kerne verteilt, dass sie untereinander keine gemeinsame Ressourcen haben.
  - Dadurch treten keine Blockierzeiten zwischen Prozessorkernen auf und die Berechnung der Blockierzeiten bleibt gültig.

## Realzeithnachweis

- Mit Kenntnis der ursprünglichen Zeitanforderungen und der Verarbeitungszeiten ist der Realzeithnachweis für jede Rechenzeitanforderung und für jeden Prozessorkern getrennt durchzuführen.
  - Können dabei einzelne Realzeitanforderungen nicht eingehalten werden, ist der Entwurf anzupassen.
  - Eventuell führen andere Algorithmen zu einer kürzeren Laufzeit.
  - Das Umverteilen der Jobs zwischen den Rechnerkernen bei einem Multicore-Ansatz könnte ebenso Abhilfe bringen,
    - sowie eine leistungsstärkere Hardware.
  - Vielleicht sind aber auch die zeitlichen Anforderungen doch nicht so streng, wie sie ursprünglich definiert wurden?
- Das Anpassen ist so lange durchzuführen, bis alle funktionalen und alle zeitlichen Anforderungen erfüllt werden können.

## Test und Dokumentation (1)

- Als letzter Schritt der Entwurfsphase muss der Test geplant werden. Hierzu sind für jedes Modul Testpattern inklusive der erwarteten Ergebnisse aufzustellen. Auch für das Gesamtsystem ist ein Testplan zu entwerfen.
- Für den Komponententest wird auf die während des Entwurfs vorbereiteten Testfälle zurückgegriffen. Eventuell muss der Entwickler sich eine Testumgebung programmieren, die Funktionsblöcke mit den notwendigen Parametern aufruft und die Ergebnisse verifiziert.
- Zur Dokumentation des Entwurfes können beispielsweise Datenflussdiagramme eingesetzt werden, die einen Überblick über das Gesamtsystem geben.

## Test und Dokumentation (2)

- Die Abläufe innerhalb der einzelnen Funktionsblöcke werden per Struktogramm beschrieben.
- Die Nebenläufigkeit des Systems kann schließlich mit einem Petrinetz modelliert werden
- In vielen Fällen hat sich auch UML (Unified Modelling Language) etabliert.

## Fehlersuche

- **Debugging problematisch:**
  - Das Debuggen hat Einfluss auf das Zeitverhalten.
  - Im Embedded-Umfeld ist häufig ein Remote-Debugging notwendig.
    - Auf dem Zielsystem (Target) läuft der Debug-Server, der über z.B. die serielle Schnittstelle mit dem Debug-Host kommuniziert.
    - Der Debug-Host benötigt neben dem Debugger-Frontend,
      - die Executables, die Objektdateien und den zugehörigen Quellcode.
  - Bei einigen Realzeitararchitekturen befinden sich Teile der Realzeitanwendung im Kernel, hier ist ein Kernel-Debugging notwendig.

## Fehlersuche

### ■ Tracing:

- manchmal die einzige Möglichkeit dar, zeitrelevante Fehler zu finden.
- Dazu können
  - systemimmanente Tracing-Mechanismen (strace) oder
  - selbst implementierte (fprintf) genutzt werden.

## Selbst implementiertes Tracing

- Sinnvoll ist es, jede Ausgabe mit einem Zeitstempel zu versehen, eventuell auch den Namen der Quellcodedatei und die zugehörige Zeilennummer mit ausgeben zu lassen.
- Der Zeitstempel ermöglicht den zeitlichen Abgleich, wenn später Traces von unterschiedlichen Threads zusammengeführt werden müssen
- Sinnvollerweise stellt der Entwickler zudem direkt ein Trace-Makro zur Verfügung, das per Define zum Generierungszeitpunkt entweder ein- oder ausgeschaltet werden kann.

## Selbst implementiertes Tracing

### Beispiel

```
#include <stdio.h>
#include <sys/time.h>
int main( int argc, char **argv, char **envp ) {

    struct timeval tv;
    gettimeofday( &tv, NULL );
    fprintf(stderr, "%ld s %ld us: %s %s line %d\n",
        tv.tv_sec, tv.tv_usec, __FILE__,
        __FUNCTION__, __LINE__ );
    return 0;
}
```

## Post-Mortem Debugging

- im Fall eines Program Absturzes wird ein Speicherabzug der von der Task verwendeten Speichersegmente angelegt (core)
- Zur Freischaltung und Konfiguration wird in den Userlimits per ulimit-Kommando die maximale Größe der Speicherabzugsdatei (Core File Size) in Blöcken festgelegt.

## Post-Mortem Debugging

### Beispiel

```
user@host:/tmp>ulimit -c 100000
user@host:/tmp>./faulty_app
Speicherzugriffsfehler (Speicherabzug
geschrieben)
user@host:/tmp>gdb faulty_app core
GNU gdb (Ubuntu/Linaro 7.2-1ubuntu1)
7.2 Copyright (C) 2010 Free Software
Foundation, Inc.
...
Loaded symbols for /lib64/ld-linux-
x86-64.so.2
Core was generated by `./faulty_app'.
Program terminated with signal 11,
Segmentation fault.
#0 0x00000000004004cf in main (argc=1,
argv=0x7ffff2163e98,
envp=0x7ffff2163ea8) at faulty_app.c:7
7   c = *ptr;
(gdb)
```

## Kernel Debugging: kgdb

- Da hierbei der Kernel angehalten werden muss, steht keine Ablaufumgebung mehr zur Verfügung
- Technisch wird dieses Problem dadurch gelöst, dass komplexe Funktionen auf ein zweites System ausgelagert werden.
- Für den zu debuggenden Kernel wird nur noch ein sogenannter Debug-Server benötigt, der einfache Kommandos, wie beispielsweise Speicherzellen lesen, schreiben oder Breakpoints setzen, im zu untersuchenden System ausführen kann.
- Im Linux-Kernel stellt der kgdb einen solchen Debug-Server dar.
  - Als Frontend wird gdb eingesetzt.

## Kernel Debugging: kdb

- Linux bietet mit dem kdb alternativ ein lokales Frontend an, das zwar bereits komplexere Kommandos ausführen kann, aber kein Debugging auf Hochsprachenniveau ermöglicht:
  - root@host:# echo kms,kbd > /sys/module/kgdboc/parameters/kgdboc
- So scharf gestellt wird ein Kernel-Panic ebenso in den Debugger springen wie das Aktivieren über eine magische Tastatursequenz oder der schreibende Zugriff auf die Datei /proc/sysrq-trigger:
  - root@host:# echo g > /proc/sysrq-trigger

## Hardwaregestützte Debugging-Verfahren

- Primär ist JTAG entwickelt worden, um integrierte Schaltungen (IC) auf ihre Funktion testen zu können, wenn diese bereits auf der Zielhardware verbaut wurden.
- Integrierte Schaltungen mit JTAG-Interface besitzen zusätzliche Schaltungsteile, die im Normalbetrieb völlig abgetrennt von den anderen Funktionsblöcken des IC sind.
  - Diese Schaltungsteile können über eigene Anschlussleitungen (PIN) aktiviert werden.



## Hardwaregestützte Debugging-Verfahren

- Außer zur Überprüfung der Hardware lässt sich JTAG im Bereich der eingebetteten Systeme einsetzen für:
  - Programmierung von Flash-Speicher
  - Einzelschritt-Debugging (Breakpoints setzen/rücksetzen, run/stop, single step, watchpoints setzen)
  - Testen der digitalen Ein- und Ausgänge
    - JTAG ist damit auch eine einfache Form eines Incircuit-Emulators.

## ICE (Incircuit-Emulator)

- Bei einem ICE wird ein Baustein in der Schaltung, typischerweise die CPU, durch den Emulator ersetzt.
- Dieser ist in der Lage, die CPU mit all ihren Signalen zu emulieren.
- Um dies zu realisieren, setzt der ICE häufig eine Spezialvariante der zu debuggenden CPU ein.
- Mit einem ICE kann man sich darüber hinaus einen Überblick über CPU-interne Zustände verschaffen.
- Da es sich bei dem Incircuit-Emulator um sehr leistungsfähige Hardware handelt, ist der Debugger zum einen sehr spezifisch und zum anderen sehr teuer. Das schränkt seine Verbreitung ein.

## Profiling: gprof (1)

- Unter Profiling versteht man das Ausmessen des Zeitverhaltens einer Applikation.
- Es wird gemessen, wie häufig eine:
  - Funktion aufgerufen wird und wie lang die Abarbeitung einer Funktion dauert.
- Unter Linux wird die Instrumentalisierung durch die Option **-pg** beim Kompilieren und Linken aktiviert. Die Auswertung schließlich erfolgt mit dem Programm gprof.

## Profiling: gprof (2)

- Profiling wird typischerweise über zwei Technologien realisiert.
  - Bei der **Instrumentalisierung** wird der Code modifiziert, sodass die gesuchten Informationen beim Ablauf vom Code selbst generiert werden.
    - Damit hat das Profiling einen – allerdings geringen – Einfluss auf das Zeitverhalten.
  - Bei der **Abtastung** wird in regelmäßigen, kurzen Abständen der Program Counter (PC) der CPU ausgelesen.
    - Aus der dort abgelegten Adresse kann berechnet werden, welche Funktion gerade abgearbeitet wird.
    - Die Information lässt auch Rückschlüsse über die Laufzeit der Funktion zu.

## Profiling: callgrind / helgrind (1)

- **Callgrind** ist ein sogenanntes Profiling-Werkzeug, welches Funktionsaufrufe innerhalb einer Anwendung protokolliert und als Aufrufgraph (call-graph) ausgibt.
  - Damit lässt sich der Programmpfad genau verfolgen.
- Mit **Helgrind** lassen sich Synchronisationsprobleme in parallelen Anwendungen finden.
  - Dazu zählen Deadlocks, Race Conditions oder der fehlerhafte Einsatz der POSIX Pthread API an sich wie z.B.:
    - Freigeben eines invaliden oder zuvor nicht angeforderten Locks,
    - rekursives Anfordern eines Mutex.

## Profiling: callgrind / helgrind (2)

- Tools wie helgrind helfen dem Entwickler zwar bei der Suche nach Synchronisationsfehlern, können aber keine vollständige Testabdeckung realisieren.
  - Auch sind False-Positiv-Meldungen dieser Tools nicht selten. Für die Analyse wird also ein erfahrener Entwickler benötigt.
- Alle Valgrind-Werkzeuge unterstützen von Haus aus Threads.

## Der Systemtest (1)

- Der abschließende Systemtest soll sicherstellen, dass sowohl die funktionalen als auch die zeitlichen Anforderungen eingehalten werden.
  - Dazu werden die Testfälle, die bereits beim Entwurf festgelegt wurden, abgearbeitet.
- Um die zeitlichen Anforderungen zu überprüfen, wird das Realzeitsystem unter verschiedenen Lastsituationen, insbesondere unter hoher Last, getestet.

## Der Systemtest (2)

- Last kann auf zweierlei Arten erzeugt werden:
  - Verarbeitet das Realzeitsystem Signale aus einem technischen Prozess, so wird versucht, eine Situation herbeizuführen, bei der möglichst sämtliche Signale gleichzeitig mit höchster Frequenz auftreten.
  - Das Realzeitsystem wird mit einem (nutzlosen) Hintergrundprozess belastet. Hierbei ist zu differenzieren, ob eine CPU-Last, eine Netzwerklast, eine Last des Dateisystems oder alles zusammen erzeugt werden soll
    - Beispiele siehe Buch!



Vielen Dank

Fragen?

Professor Dr. Michael Mächtel