# Harte Echtzeit für Anwendungsprozesse in Standard-Betriebssystemen auf Mehrkern-Prozessoren

## Workshop Echtzeit 2011

Georg Wassen, Stefan Lankes und Thomas Bemmerl

3. November 2011

LEHRSTUHL FÜR BETRIEBSSYSTEME
Univ.-Prof. Dr. habil. Thomas Bemmerl

RWTH AACHEN UNIVERSITY

# Inhalt

- Motivation
- Realisierung
- Anwendungen
- Zusammenfassung

- Motivation
- Realisierung
- Anwendungen
- Zusammenfassung

## Echtzeit-Programmierung

verbreitete Ansätze:

- POSIX.4 – weiche Echtzeit
- Kernel-Modifikationen (meist Linux)
  - Änderung am Kernel selber
  - Subkernel (Linux + Xenomai/RTAI/Adeos)
- RTOS (QNX Neutrino, Wind River VxWorks, usw.)

# Motivation

unsere Vorgaben:

- „general-purpose“ Betriebssystem
- Portabilität
- Konfigurierbarkeit
- harte Echtzeit

# Motivation

unsere Vorgaben:

- „general-purpose“ Betriebssystem
  - Nutzung beliebiger Bibliotheken und Anwendungen neben Echtzeit-Tasks
- Portabilität
- Konfigurierbarkeit
- harte Echtzeit

# Motivation

unsere Vorgaben:

- „general-purpose“ Betriebssystem
  - Nutzung beliebiger Bibliotheken und Anwendungen neben Echtzeit-Tasks
- Portabilität
  - kein/minimaler Eingriff in den Kernel
- Konfigurierbarkeit

- harte Echtzeit

# Motivation

unsere Vorgaben:

- „general-purpose“ Betriebssystem
  - Nutzung beliebiger Bibliotheken und Anwendungen neben Echtzeit-Tasks
- Portabilität
  - kein/minimaler Eingriff in den Kernel
- Konfigurierbarkeit
  - Echtzeitdomänen (weich – hart)
  - flexibler Start weiterer Tasks
- harte Echtzeit

# Motivation

unsere Vorgaben:

- „general-purpose“ Betriebssystem
  - Nutzung beliebiger Bibliotheken und Anwendungen neben Echtzeit-Tasks
- Portabilität
  - kein/minimaler Eingriff in den Kernel
- Konfigurierbarkeit
  - Echtzeitdomänen (weich – hart)
  - flexibler Start weiterer Tasks
- harte Echtzeit
  - formeller Beweis mit vertretbarem Aufwand möglich

# Motivation

unsere Vorgaben:

- „general-purpose“ Betriebssystem
  - Nutzung beliebiger Bibliotheken und Anwendungen neben Echtzeit-Tasks
- Portabilität
  - kein/minimaler Eingriff in den Kernel
- Konfigurierbarkeit
  - Echtzeitdomänen (weich – hart)
  - flexibler Start weiterer Tasks
- harte Echtzeit
  - formeller Beweis mit vertretbarem Aufwand möglich

Einsatzgebiete:

- Forschung, Messungen
- Rapid Prototyping
- mobile Anwendungen (Laptop, Roboter)
- Portierung von eingebettetem RTOS in isolierten Prozess

## Umgebung

- x86 (32 und 64 Bit)
- Mehrprozessor-System
- Hyperthreading deaktiviert

## Umgebung

- x86 (32 und 64 Bit)
  - flexibel
  - weit verbreitet, vertraut
  - auch für eingebettete Systeme
- Mehrprozessor-System

- Hyperthreading deaktiviert

## Umgebung

- x86 (32 und 64 Bit)
  - flexibel
  - weit verbreitet, vertraut
  - auch für eingebettete Systeme
- Mehrprozessor-System
  - mittlerweile Standard
  - weiter steigende Prozessorzahlen erwartet
- Hyperthreading deaktiviert

## Umgebung

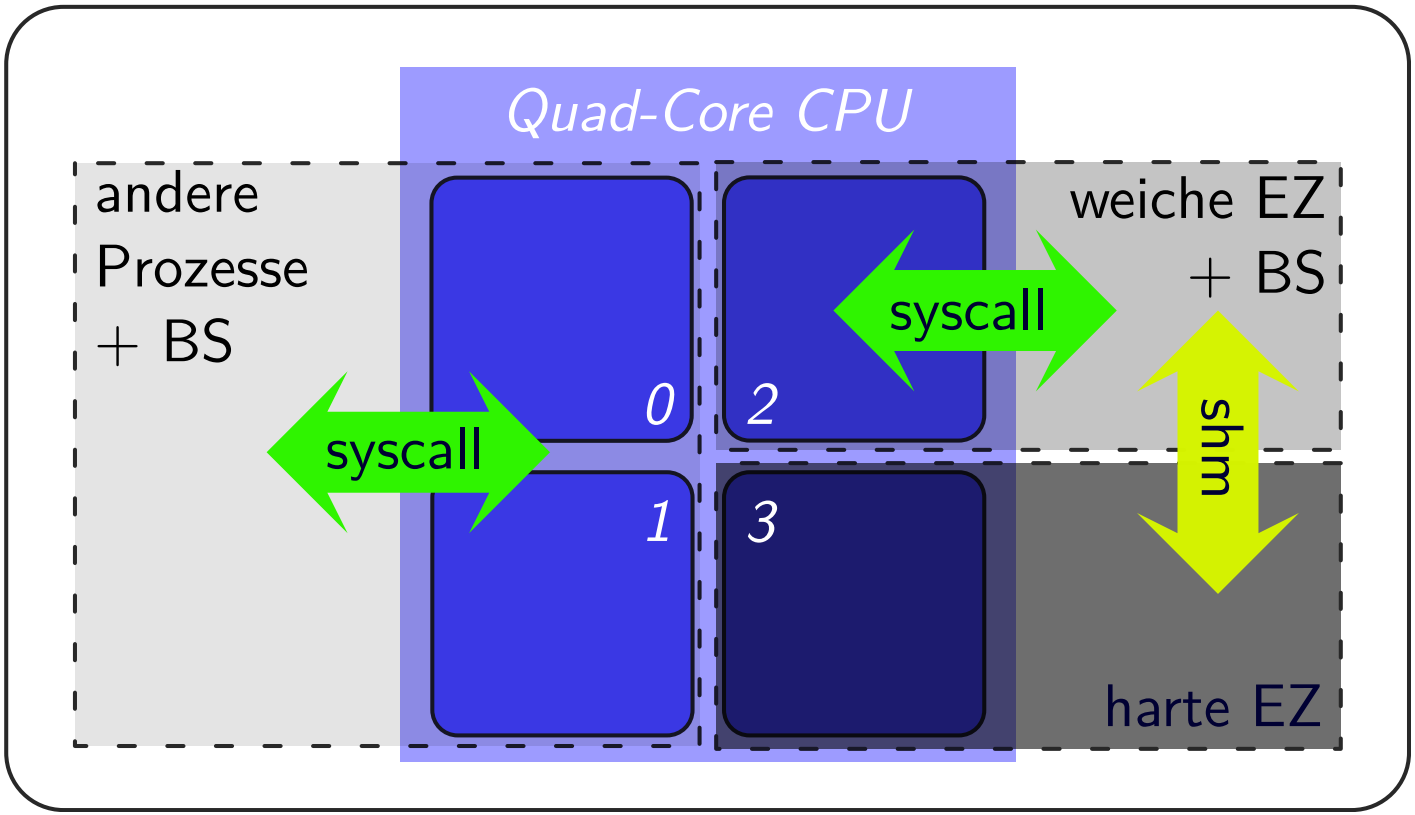
- x86 (32 und 64 Bit)
  - flexibel
  - weit verbreitet, vertraut
  - auch für eingebettete Systeme
- Mehrprozessor-System
  - mittlerweile Standard
  - weiter steigende Prozessorzahlen erwartet
- Hyperthreading deaktiviert
  - gegenseitige Beeinflussung nicht abschätzbar

# Realisierung

- Motivation
- Realisierung
  - Messungen
  - schrittweise Isolation
  - weitere Maßnahmen
  - Seiteneffekte
- Anwendungen
- Zusammenfassung

# Realisierung

- Isolation einer CPU aus laufendem Betriebssystem
- Kontrolle über jede Instruktion, die dort ausgeführt wird.
- übrige CPUs: Betriebssystem, Interrupts, Prozesse



## Hourglass-Benchmark

Regehr, 2002

- kurze Schleife
- Vergleich der Timestamps (`rdtsc`)
- kürzeste/durchschnittliche Dauer: 20–60 Ticks
- Schwelle zur Erkennung von Unterbrechungen

```
t1 = rdtsc();
while (t1 < t_end) {
    t2 = rdtsc();
    if ( (t2-t1) > threshold ) {
        record_gap(t1, t2);
    }
    t1 = t2;
}
```

## Größenordnungen

| Prozessorzyklen (Ticks) | |
|---|---|
| Instruktion, Registerzugriff | 1 |
| Cache-Zugriff | 5–10 |
| Hauptspeicherzugriff | 10–30 |
| Kontextwechsel (Userspace–Kernel) | 300 |
| kürzester Interrupt | 1000 |
| durchschnittlicher Interrupt | 5000 |

## Größenordnungen

| Prozessorzyklen (Ticks) | |
|---|---|
| Instruktion, Registerzugriff | 1 |
| Cache-Zugriff | 5–10 |
| Hauptspeicherzugriff | 10–30 |
| Kontextwechsel (Userspace–Kernel) | 300 |
| kürzester Interrupt | 1000 |
| durchschnittlicher Interrupt | 5000 |
| | |
| Schwelle in Hourglass | 500 |

## CPU-Frequenz

- moderne CPUs passen Frequenz der Systemauslastung an.
- Frequenzwechsel: unbekannte Latenz

## CPU-Frequenz

- moderne CPUs passen Frequenz der Systemauslastung an.
- Frequenzwechsel: unbekannte Latenz

→ Energieschema auf „Performance“ stellen.

## Seitenauslagerung

- Speicherseitenverwaltung (Paging) kann Hauptspeicher (RAM) auf Hintergrundspeicher (Festplatte) auslagern.
- Page-Fault

## Seitenauslagerung

- Speicherseitenverwaltung (Paging) kann Hauptspeicher (RAM) auf Hintergrundspeicher (Festplatte) auslagern.
- Page-Fault

→ `mlockall()`, Arrays vorher initialisieren

## Scheduling, CPU-Isolation

- POSIX: `sched_setscheduler()`
- CPU-Affinität: `sched_setaffinity()`
- CPU-Set (Linux)

## Scheduling, CPU-Isolation

- POSIX: `sched_setscheduler()`
  - weiche Echtzeit
  - nicht nötig, wenn einzelner Prozess auf CPU isoliert wird.
- CPU-Affinität: `sched_setaffinity()`

- CPU-Set (Linux)

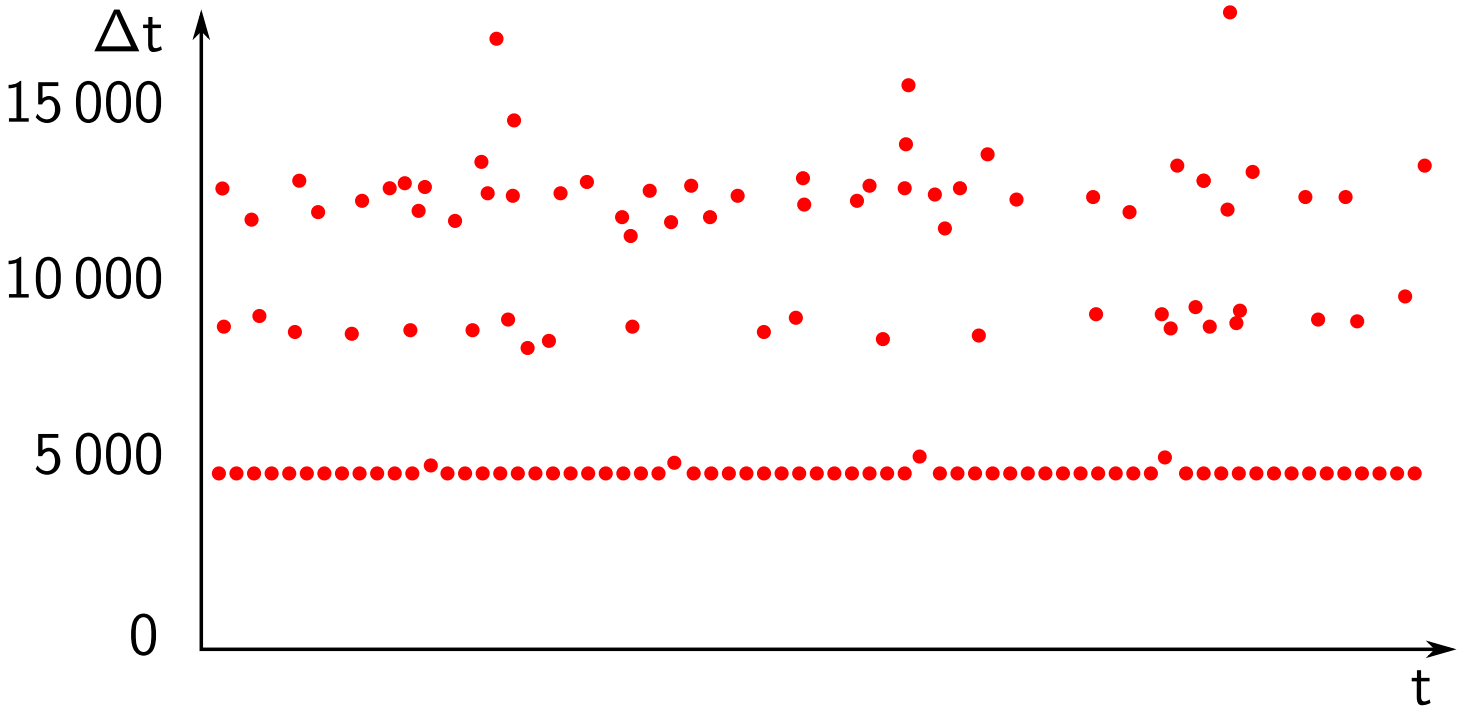
## Scheduling, CPU-Isolation

- POSIX: `sched_setscheduler()`
  - weiche Echtzeit
  - nicht nötig, wenn einzelner Prozess auf CPU isoliert wird.
- CPU-Affinität: `sched_setaffinity()`
  - Task auf eine CPU
  - alle anderen auf übrige CPUs
- CPU-Set (Linux)

## Scheduling, CPU-Isolation

- POSIX: `sched_setscheduler()`
  - weiche Echtzeit
  - nicht nötig, wenn einzelner Prozess auf CPU isoliert wird.
- CPU-Affinität: `sched_setaffinity()`
  - Task auf eine CPU
  - alle anderen auf übrige CPUs
- CPU-Set (Linux)
  - Prozesse in CPU-Partitionen
  - komfortabler

## Analyse

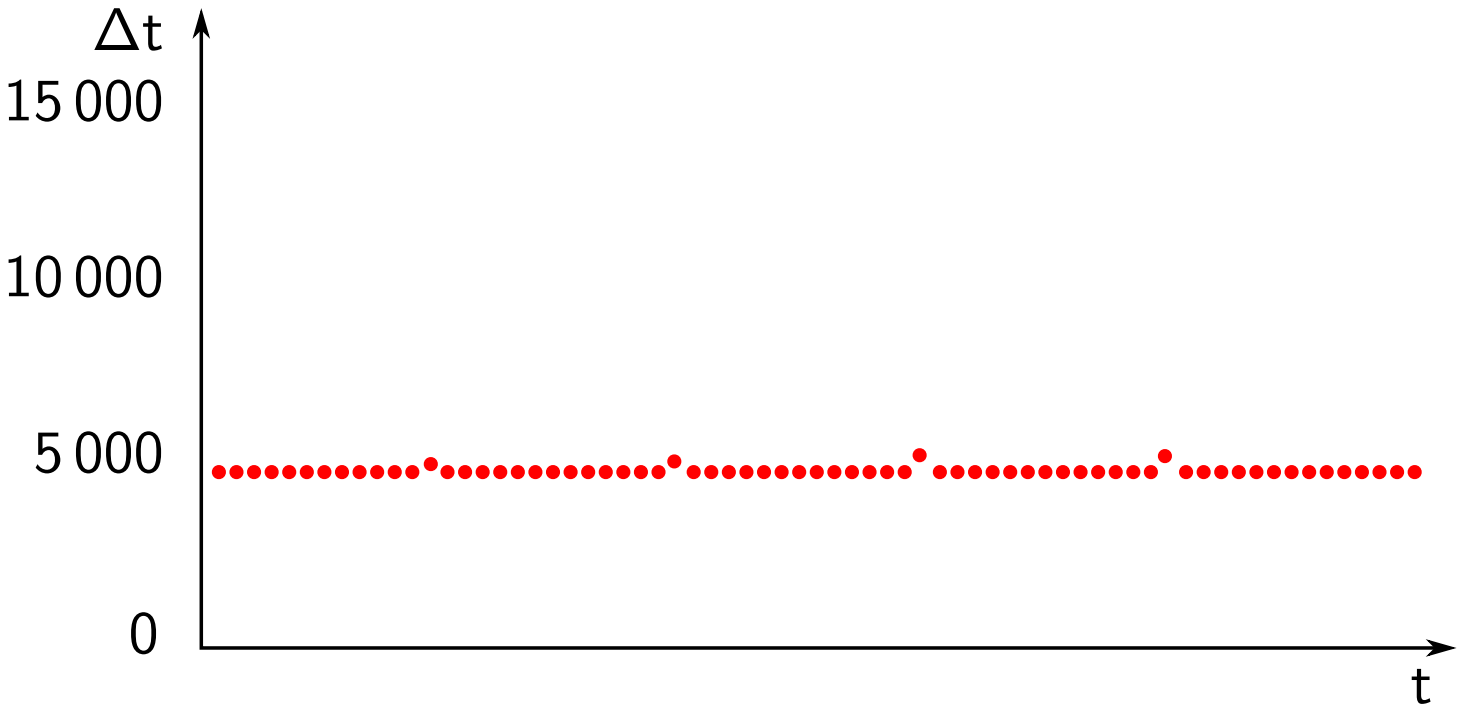



- ein Prozess; keine Unterbrechung durch Zeitscheiben
- aber: viele Interrupts

# Realisierung – Isolation

- Interrupt-Affinität
  - Linux: `/proc/irq/*/cpu_affinity`
  - Linux: *irqbalance* beenden
  - allgemein: Routing im I/O-APIC
  - IRQs auf anderen CPUs behandeln
  - betrifft Hardware-Interrupts (IRQ)

## Analyse




- weniger Unterbrechungen,
- immer noch Timer-Interrupt (local APIC)

## Interrupts sperren

- local APIC: konfigurierbar über Speicher
- Interrupt-Flag (IF)

## Interrupts sperren

- local APIC: konfigurierbar über Speicher
  - Deaktivieren möglich
- Interrupt-Flag (IF)

## Interrupts sperren

- local APIC: konfigurierbar über Speicher
  - Deaktivieren möglich
- Interrupt-Flag (IF)
  - mit I/O-Privilegienlevel 3 änderbar
  - sperrt alle Interrupts (außer Exceptions)

## Interrupts sperren

- local APIC: konfigurierbar über Speicher
  - Deaktivieren möglich
- Interrupt-Flag (IF)
  - mit I/O-Privilegienlevel 3 änderbar
  - sperrt alle Interrupts (außer Exceptions)
- IRQs wurden auf andere CPUs umgeleitet, werden also weiter behandelt.

## Analyse


- keine Unterbrechungen mehr
- freiwilliger Verzicht auf Systemaufrufe

## NMI-Watchdog

- ausgelöst durch Timer/Performance-Measurement-Counter
- nutzt nicht-maskierbaren Interrupt (NMI)

## NMI-Watchdog

- ausgelöst durch Timer/Performance-Measurement-Counter
- nutzt nicht-maskierbaren Interrupt (NMI)

→ über BS-Schnittstelle deaktivierbar

## System-Management Mode

- SMI – Interrupt höchster Priorität
- benötigt für ACPI und legacy-Support
- Bedeutung für Echtzeitanwendungen:

## System-Management Mode

- SMI – Interrupt höchster Priorität
- benötigt für ACPI und legacy-Support
- Bedeutung für Echtzeitanwendungen:
  - sehr lange Unterbrechungen ($> 500\,000$ Ticks)
  - Hardware-/BIOS-abhängig (bis 20 Hz)

## System-Management Mode

- SMI – Interrupt höchster Priorität
- benötigt für ACPI und legacy-Support
- Bedeutung für Echtzeitanwendungen:
  - sehr lange Unterbrechungen ($> 500\,000$ Ticks)
  - Hardware-/BIOS-abhängig (bis 20 Hz)

$\rightarrow$ deaktivierbar

  - nicht mit jedem BIOS möglich
  - evtl. Gefahr des Überhitzens

## Einfluss anderer CPUs

- Durchschnitt nahe bei Minimum
- Varianz der Ausführungszeit (Jitter) einer kurzen Schleife

## Einfluss anderer CPUs

- Durchschnitt nahe bei Minimum
  - → längere Unterbrechung selten
- Varianz der Ausführungszeit (Jitter) einer kurzen Schleife

## Einfluss anderer CPUs

- Durchschnitt nahe bei Minimum
  → längere Unterbrechung selten
- Varianz der Ausführungszeit (Jitter) einer kurzen Schleife (Minimum: 25 Ticks, Größenordnung)

| | | |
|---|---|---|
| andere CPUs deaktiviert | 3 | 1 ns |

## Einfluss anderer CPUs

- Durchschnitt nahe bei Minimum
  → längere Unterbrechung selten
- Varianz der Ausführungszeit (Jitter) einer kurzen Schleife (Minimum: 25 Ticks, Größenordnung)

| | | |
|---|---|---|
| andere CPUs deaktiviert | 3 | 1 ns |
| andere CPUs im Leerlauf | 10 | 3 ns |

## Einfluss anderer CPUs

- Durchschnitt nahe bei Minimum
  → längere Unterbrechung selten
- Varianz der Ausführungszeit (Jitter) einer kurzen Schleife (Minimum: 25 Ticks, Größenordnung)

| | | |
|---|---|---|
| andere CPUs deaktiviert | 3 | 1 ns |
| andere CPUs im Leerlauf | 10 | 3 ns |
| andere CPUs arbeiten im Cache | 250 | 100 ns |

## Einfluss anderer CPUs

- Durchschnitt nahe bei Minimum
  - → längere Unterbrechung selten
- Varianz der Ausführungszeit (Jitter) einer kurzen Schleife (Minimum: 25 Ticks, Größenordnung)

| | | |
|---|---|---|
| andere CPUs deaktiviert | 3 | 1 ns |
| andere CPUs im Leerlauf | 10 | 3 ns |
| andere CPUs arbeiten im Cache | 250 | 100 ns |
| andere CPUs auf Hauptspeicher | 1000 | 300 ns |

gemessen auf: Core i7 (Nehalem)

## Reaktion des Linux-Kernels

mögliche Beeinträchtigungen (abhängig von Konfiguration)

- Inter-Prozessor Interrupts
- Uhrzeit

## Reaktion des Linux-Kernels

mögliche Beeinträchtigungen (abhängig von Konfiguration)

- Inter-Prozessor Interrupts
  - werden durch Interrupt-Flag blockiert
  - Sender könnte blockiert werden
- Uhrzeit

## Reaktion des Linux-Kernels

mögliche Beeinträchtigungen (abhängig von Konfiguration)

- Inter-Prozessor Interrupts
  - werden durch Interrupt-Flag blockiert
  - Sender könnte blockiert werden
- Uhrzeit
  - Uhrzeit wird durch Timer-Interrupt gepflegt
  - Sperre: Uhrzeit wird nicht weiter gesetzt
  - CPUs driften auseinander

## Reaktion des Linux-Kernels

mögliche Beeinträchtigungen (abhängig von Konfiguration)

- Inter-Prozessor Interrupts
  - werden durch Interrupt-Flag blockiert
  - Sender könnte blockiert werden
- Uhrzeit
  - Uhrzeit wird durch Timer-Interrupt gepflegt
  - Sperre: Uhrzeit wird nicht weiter gesetzt
  - CPUs driften auseinander

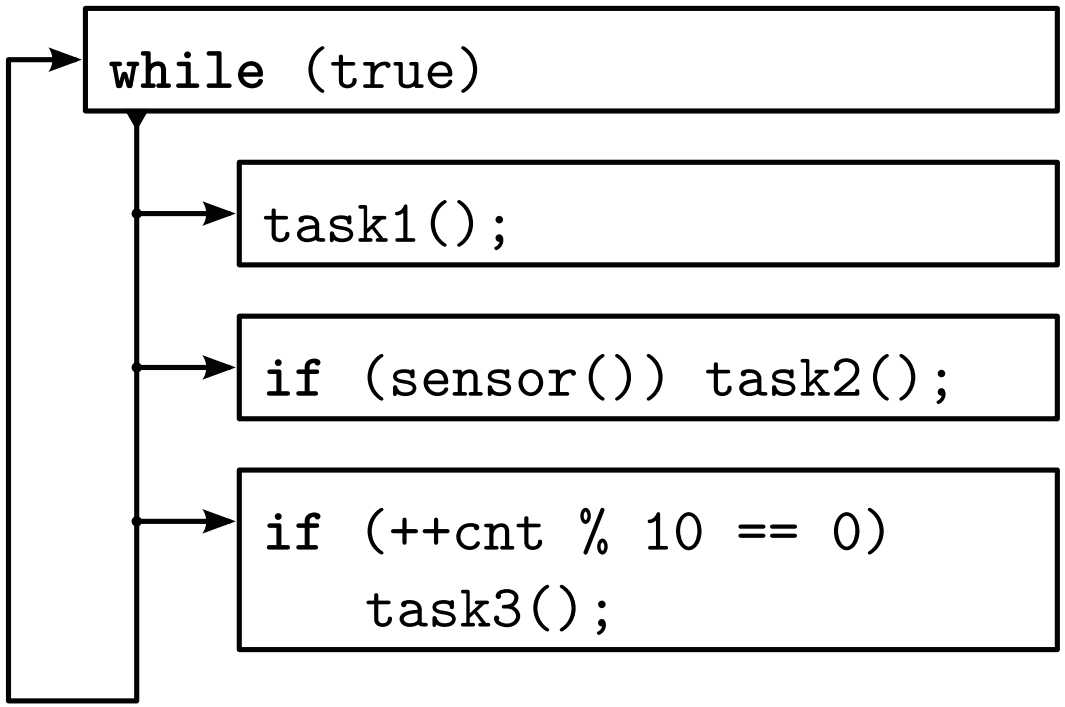
Zur Laufzeit bisher keine Störungen beobachtet, evtl. hinterher Neustart nötig.

# Anwendungen

- Motivation
- Realisierung
- Anwendungen
  - Ausblick
- Zusammenfassung

## Nutzen isolierter CPUs

- keine Interrupts: nur zeitgesteuerte Programmierung



## Nutzen isolierter CPUs

- keine Interrupts: nur zeitgesteuerte Programmierung
- keine Systemaufrufe (nach Initialisierung)
- Kommunikation ohne Betriebssystem

- Ein-/Ausgabe

## Nutzen isolierter CPUs

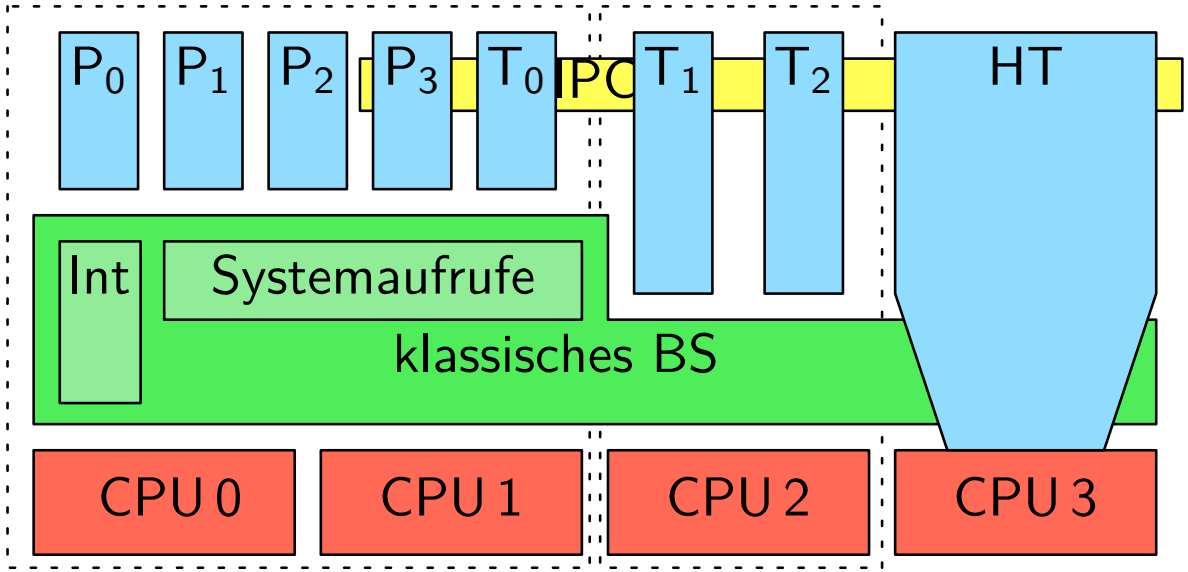
- keine Interrupts: nur zeitgesteuerte Programmierung
- keine Systemaufrufe (nach Initialisierung)
- Kommunikation ohne Betriebssystem
  - atomare Operationen auf gemeinsamem Speicher
  - Kommunikationsbibliothek: Flag, Mutex, Barrier, Queue, usw.
  - auch: lock-free, wait-free
- Ein-/Ausgabe

## Nutzen isolierter CPUs

- keine Interrupts: nur zeitgesteuerte Programmierung
- keine Systemaufrufe (nach Initialisierung)
- Kommunikation ohne Betriebssystem
  - atomare Operationen auf gemeinsamem Speicher
  - Kommunikationsbibliothek: Flag, Mutex, Barrier, Queue, usw.
  - auch: lock-free, wait-free
- Ein-/Ausgabe
  - Userspace-Treiber für I/O-Interface
  - evtl. Netzwerkkarte, Parport, RS-232 etc.

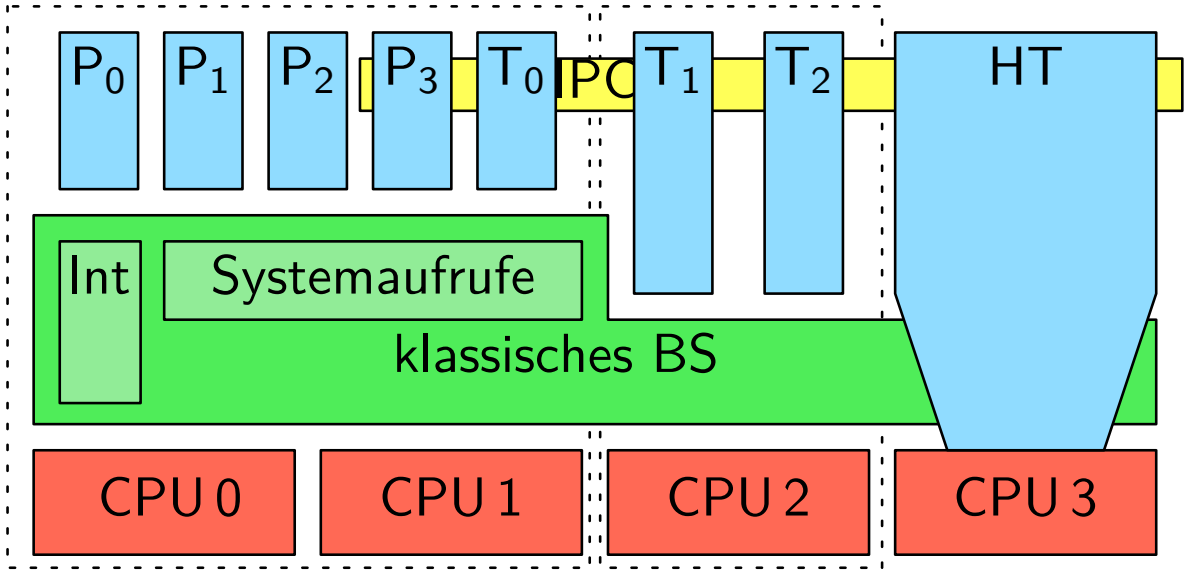
## Anwendungsbeispiel: Regelkreis

- isolierte CPU
- weitere CPU
- übriges System



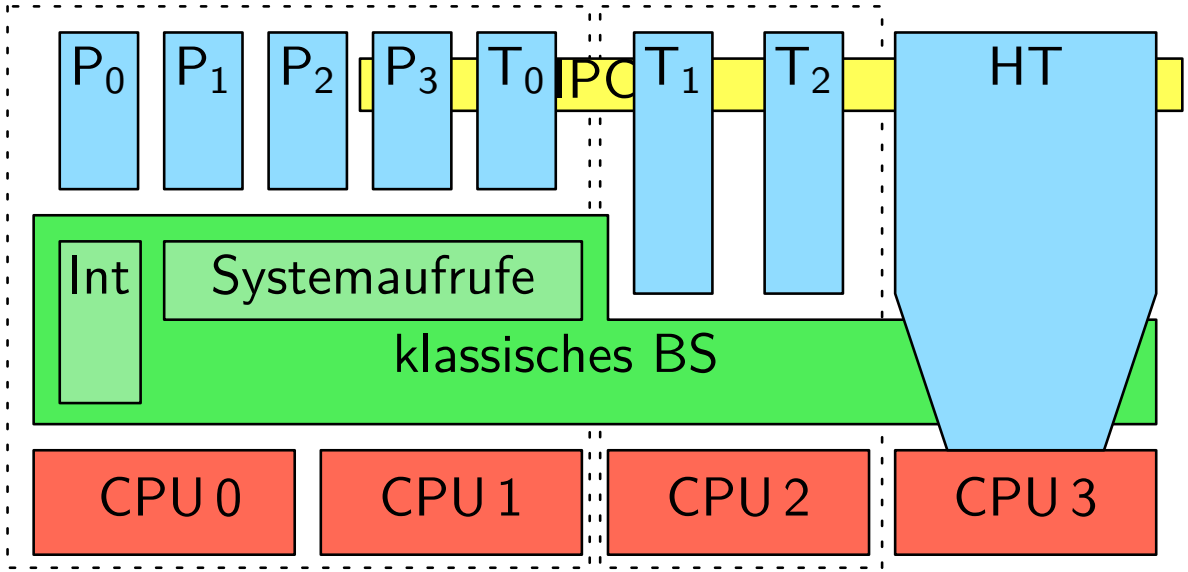
## Anwendungsbeispiel: Regelkreis

- isolierte CPU – harte Echtzeit
  - lese Eingangswerte aus Analog-Digitalwander
  - berechne Stellwerte
  - steuere Ausgabe über Digital-Analogwandler
  - Kommunikation über gemeinsamen Speicher
- weitere CPU
- übriges System



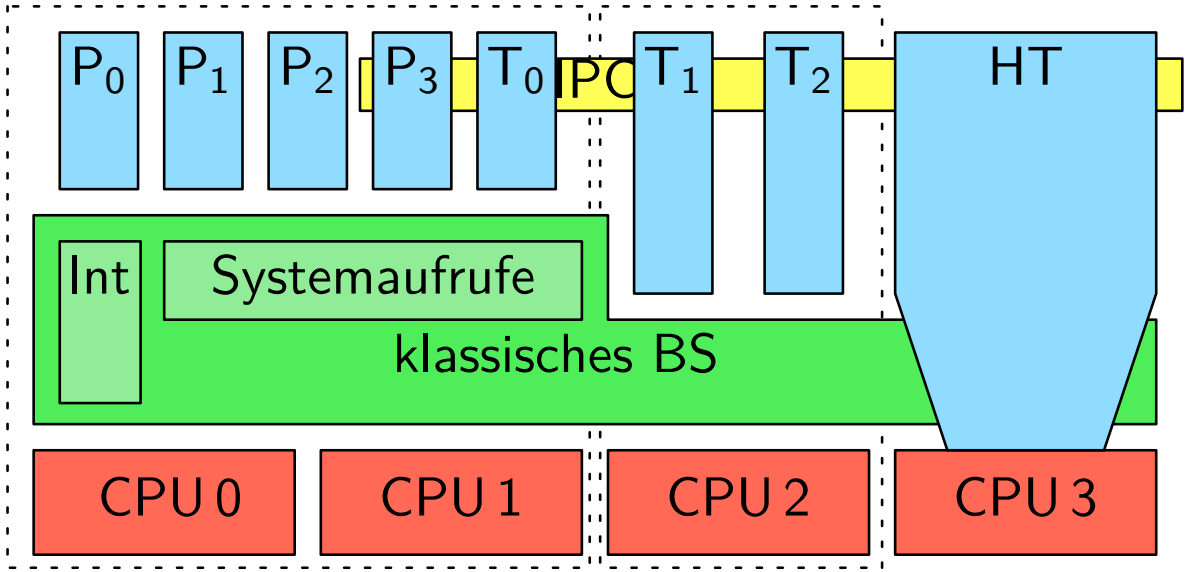
## Anwendungsbeispiel: Regelkreis

- isolierte CPU – harte Echtzeit
- weitere CPU – weiche Echtzeit
  - Übertragung von Logdaten des Regelprozesses
  - Berechnung von Regelparametern
  - Übermittlung über gemeinsamen Speicher
- übriges System



## Anwendungsbeispiel: Regelkreis

- isolierte CPU – harte Echtzeit
- weitere CPU – weiche Echtzeit
- übriges System
  - Schreiben der Logdaten in Datei
  - grafische Benutzeroberfläche zur Darstellung und Eingabe
  - alle übrigen Dienste



## Anwendungsbeispiel: Roboter

- 1 CPU: Bildaufnahme über zwei Webcams
- 2 CPUs: Tiefenbilderzeugung (rechenintensiv)
- 1 CPU: Verhaltenssteuerung

## Anwendungsbeispiel: Roboter

- 1 CPU: Bildaufnahme über zwei Webcams
  - Betriebssystemschnittstelle
  - weiche Echtzeit
  - Bilder in geteilten Puffer
- 2 CPUs: Tiefenbilderzeugung (rechenintensiv)

- 1 CPU: Verhaltenssteuerung

## Anwendungsbeispiel: Roboter

- 1 CPU: Bildaufnahme über zwei Webcams
  - Betriebssystemschnittstelle
  - weiche Echtzeit
  - Bilder in geteilten Puffer
- 2 CPUs: Tiefenbilderzeugung (rechenintensiv)
  - abwechselnde Bearbeitung des neuesten Bildpaares
  - harte Echtzeit nicht nötig
  - jedoch „best-efford“
- 1 CPU: Verhaltenssteuerung

## Anwendungsbeispiel: Roboter

- 1 CPU: Bildaufnahme über zwei Webcams
  - Betriebssystemschnittstelle
  - weiche Echtzeit
  - Bilder in geteilten Puffer
- 2 CPUs: Tiefenbilderzeugung (rechenintensiv)
  - abwechselnde Bearbeitung des neuesten Bildpaares
  - harte Echtzeit nicht nötig
  - jedoch „best-efford“
- 1 CPU: Verhaltenssteuerung
  - Nutzung der Tiefeninformation als Sensor

## Userspace-Interrupts

- Interrupt-Handler immer im Kernel-Space
- minimaler Handler (Kernel-Modul)

- Userspace-Manger

## Userspace-Interrupts

- Interrupt-Handler immer im Kernel-Space
- minimaler Handler (Kernel-Modul)
  - direkt in IDT eingetragen
  - Stackmanipulation: schiebe Funktion in Rücksprung ein
- Userspace-Manger

## Userspace-Interrupts

- Interrupt-Handler immer im Kernel-Space
- minimaler Handler (Kernel-Modul)
  - direkt in IDT eingetragen
  - Stackmanipulation: schiebe Funktion in Rücksprung ein
- Userspace-Manger
  - Kontext sichern
  - Handler-Funktion ausführen
  - Kontext wieder herstellen und fortfahren

## Userspace-Interrupts

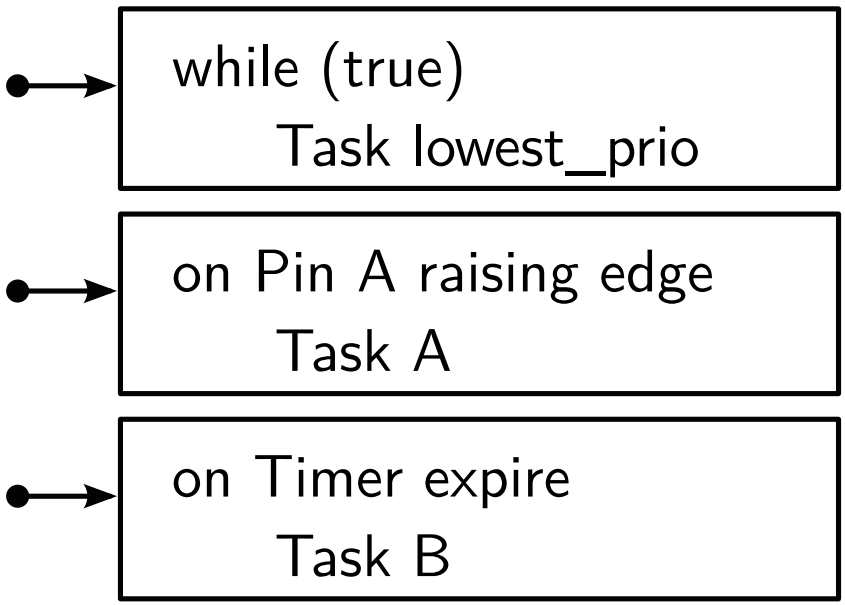
- Interrupt-Handler immer im Kernel-Space
- minimaler Handler (Kernel-Modul)
  - direkt in IDT eingetragen
  - Stackmanipulation: schiebe Funktion in Rücksprung ein
- Userspace-Manger
  - Kontext sichern
  - Handler-Funktion ausführen
  - Kontext wieder herstellen und fortfahren

→ Kontextwechsel nicht vermeidbar

→ jedoch keine „unbekannten“ Instruktionen

## Userspace-Interrupts

- ereignisgesteuertes Paradigma realisierbar
  - z. B. Timer-Interrupt, präemptives Multitasking
  - ohne Interrupt-Sperre: ungewünschte Interrupts wieder möglich



## die nächsten Schritte

- weitere Interrupt-Kontrollmöglichkeiten
  - bisher Interprozessor-Interrupts nur über IF blockierbar
- Verwirrung des Kernels durch nicht reagierende CPU:
  - Abmeldung der CPU (vgl. Hot-Plugging)
  - evtl. durch Kernel-Modul realisierbar
- Untersuchung des Einflusses von CPUs aufeinander
  - auch für RTOS von Bedeutung
- RTOS in isolierten Prozess portieren
- Übertragung auf nicht Cache-kohärent gekoppelte Systeme Cluster-on-a-Chip (Single-Chip Cloud Computer, SCC)

# Zusammenfassung

- Motivation
- Realisierung
- Anwendungen
- Zusammenfassung

## Portierbarkeit

- andere Betriebssysteme auf x86
- andere Architekturen

## Portierbarkeit

- andere Betriebssysteme auf x86
  - vieles sollte ähnlich möglich sein
  - alle genutzten Schnittstellen nah an Hardware
  - evtl. besondere Rechte zum Hardwarezugriff nötig
- andere Architekturen

## Portierbarkeit

- andere Betriebssysteme auf x86
  - vieles sollte ähnlich möglich sein
  - alle genutzten Schnittstellen nah an Hardware
  - evtl. besondere Rechte zum Hardwarezugriff nötig
- andere Architekturen
  - Interrupt-Routing und -Unterdrückung
  - atomare Operationen für Kommunikation

## Portierbarkeit

- andere Betriebssysteme auf x86
  - vieles sollte ähnlich möglich sein
  - alle genutzten Schnittstellen nah an Hardware
  - evtl. besondere Rechte zum Hardwarezugriff nötig
- andere Architekturen
  - Interrupt-Routing und -Unterdrückung
  - atomare Operationen für Kommunikation

  Beispiel ARM:
  - keine Interrupt-Deaktivierung ohne BS-Eingriff
  - keine flexiblen atomaren Operationen

# Zusammenfassung

## erreichte Ziele

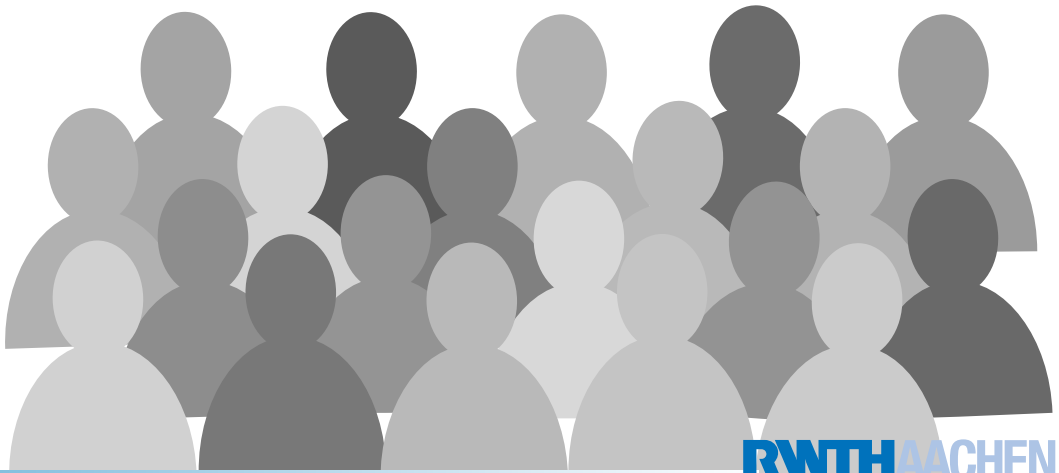
- keine Änderung am Betriebssystem nötig
- zur Laufzeit konfigurierbar, dynamischer Start
- vollständige Kontrolle über einzelne CPUs, harte Echtzeit
- Bibliothek zur Kommunikation über gemeinsamen Speicher

# Zusammenfassung

## erreichte Ziele

- keine Änderung am Betriebssystem nötig
- zur Laufzeit konfigurierbar, dynamischer Start
- vollständige Kontrolle über einzelne CPUs, harte Echtzeit
- Bibliothek zur Kommunikation über gemeinsamen Speicher

Vielen Dank!



# Zusammenfassung

## erreichte Ziele

- keine Änderung am Betriebssystem nötig
- zur Laufzeit konfigurierbar, dynamischer Start
- vollständige Kontrolle über einzelne CPUs, harte Echtzeit
- Bibliothek zur Kommunikation über gemeinsamen Speicher

Vielen Dank!
Haben Sie Fragen?