Evaluation von Echtzeitbetriebssystemen für den Einsatz in Integrated Modular Avionics

Autor: Martin Christian
Betreuer: Dr. Martin Bogdan, Universität Leipzig
Thomas Schanne, EADS Deutschland GmbH

Einführung
Gliederung

Gliederung:

1.Einführung

Motivation, Problemstellung

2.Anforderungen

Welche Anforderungen stellt die Integrated Modular Avionic an einen Kernel?

3.Analyse

- Linux: Freies Unix für den PC
- Xen: Hypervisor für Para-Virtualisierte Gast-OS
- Minix 3: µKernel + OS nach dem „TV model“
- L4: µKernel abstrahiert Raum, Aktivität und Kommunikation

4.Implementierung

- L4 Implementierungen + Linux Ports für die µKernel
- Arbeitsschritte

5.Evaluation

Was wurde erreicht? Werden die Anforderungen erfüllt?

Einführung
Problemstellung

Motivation:

- Linux setzt sich auf dem Markt eingebetteter Systeme durch [heise, 2003]
- Linux wird bei Boeing in Flugzeugen eingesetzt [heise, 2006]
- Eine Evaluation aller verfügbaren Echtzeitbetriebssysteme ist in einer Diplomarbeit nicht durchführbar
- EADS Deutschland GmbH stellt das Entwicklungsboard des Laser Range Radar Projekts *Hellas* zur Verfügung

Problemstellung:

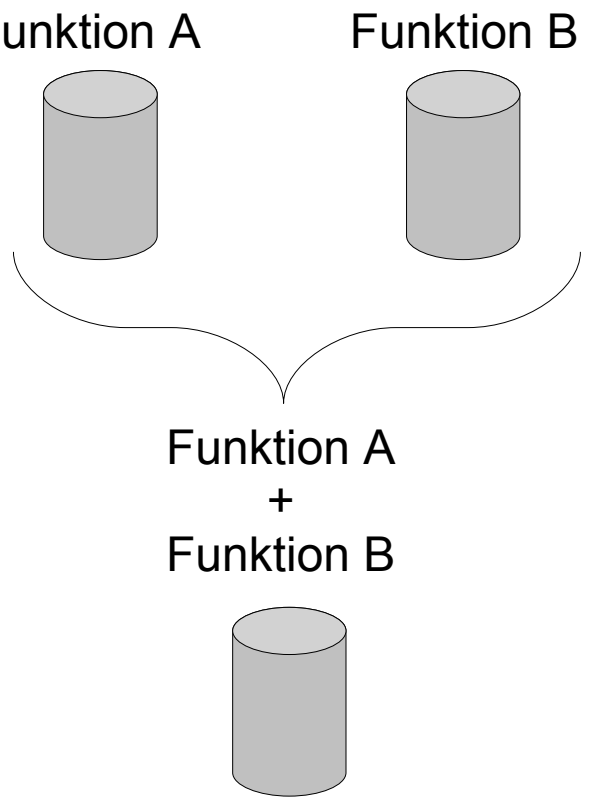
1. Den vernünftigsten Weg finden, Linux in der Luftfahrt einzusetzen
2. Linux auf diese Weise auf das EADS Entwicklungsboard portieren

Einführung
Begriffe

- ***Kernel***: „*[...] is used to denote the part of the operating system that is mandatory and common to all other software.*“ [Liedke, 1995]
  - ➔ Monolithischer Kernel: Scheduling, Interrupt-Handling, Speichermanagement und Gerätetreiber sind Teile des Kernels.
  - ➔ Mikrokernel: „[...] a concept is tolerated inside the µ-kernel only if moving it outside the kernel [...] would prevent the implementation of the system's required functionality“ [Liedke, 1995]
- **Echtzeit**: „*A real-time system responds in a (timely) predictable way to all individual unpredictable external stimuli arrivals.*“ [Timmerman+, 2005]
  - ➔ Weiche Echtzeit: Zeitvorgaben müssen im Durchschnitt eingehalten werden
  - ➔ Harte Echtzeit: Zeitvorgaben müssen immer eingehalten werden

Anforderungen
IMA

Avionik: „*[...] Kunstw. aus Aviatik u. Elektronik*“ . Sie umfasst die „*Gesamtheit der elektronischen Geräte, die in der Luftfahrt verwendet werden.*“ [Duden, 2001]



<u>Ziele der Integrated Modular Avionics (IMA):</u>

- *Funktionalität*: Mehr Funktionalität auf weniger Raum
- *Sicherheit*: Einfache Bedienung, Rekonfiguration bei Hardwarefehlern
- *Kosten*: Modularer Aufbau senkt Kosten bei Entwicklung und Wartung

Anforderungen

IMA-Kriterien

Anforderungen an einen IMA-Kernel:

- ***Echtzeit:***

  Der Kernel muss mindestens die Echtzeitanforderungen der anspruchsvollsten Anwendung erfüllen. → Harte Echtzeit

- ***Partitionierung:***

  "The behaviour and performance of software in one partition must be unaffected by the software in other partitions." [Rushby, 1999]

  - Räumlich: Keine Partition darf Daten anderer Partitionen manipulieren können → Weder im Hauptspeicher noch auf Geräten
  - Zeitlich: Keine Partition darf einer anderen Zeit stehlen

Anforderungen
IMA-Kriterien

Anforderungen (Fortsetzung):

- **Trusted Computing Base (TCB):**
  - Minimale TCB → einfachere Zertifizierung
  - Weniger Code → weniger Bugs [Herder+, 2006]

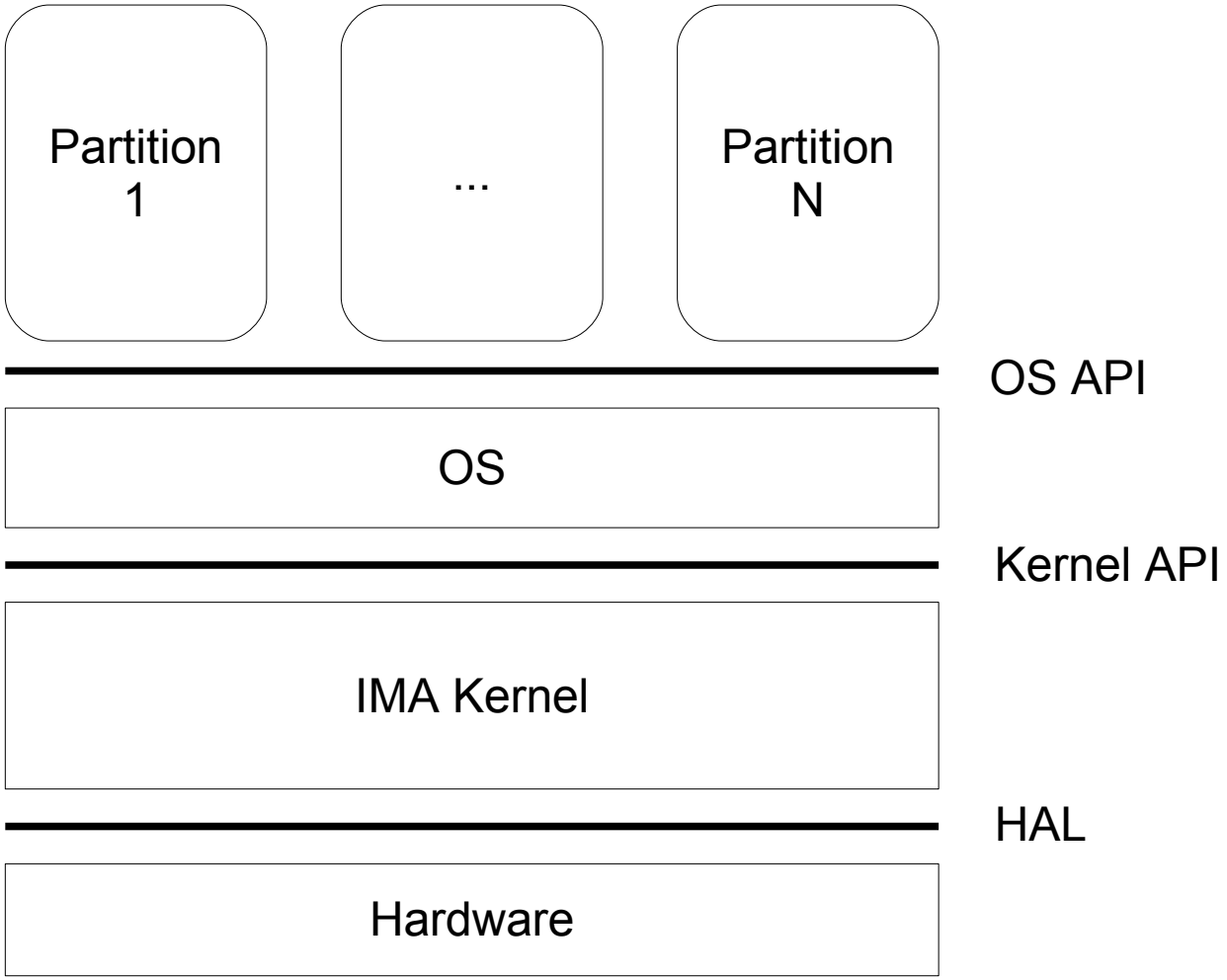
- **Offene Standards:**

  Unabhänigkeit von Herstellern sichert Verfügbarkeit von Komponenten

- **Modularität:**
  - Wiederverwendung von Komponenten → Entwicklungskosten
  - Austausch von Komponenten → Lagerkosten (Produktzyklus > 10 Jahre)

Anforderungen

IMA-Modell



Generisches IMA Modell nach [Bennett, 2003]

Analyse
Ausgrenzung

Einschränkungen:

- Nur Open Source Lösungen: Beschaffung, Veröffentlichung
- Ziel-orientierte Auswahl: Nur Kernel für die es ein Linux gibt
- Vorauswahl von [Bennett, 2003]

Ausgeschlossene Kernel (Auswahl):

- Mach, C5: Veraltet, da µKernel der 1. Generation
- RTEMS: OS mit nur einem Adressraum (keine Partitionierung)
- MicroC/OS-II: Nicht Open Source im engeren Sinn, kein Linux
- xBSD: Multi-User OS, keine Echtzeit
- VxWorks: Nicht Open Source
- QNX: Nicht Open Source
- PikeOS: Nicht Open Source
- Coyotos: Frühes Stadium der Entwicklung

Analyse
Eingrenzung

Kernel in der engeren Wahl:

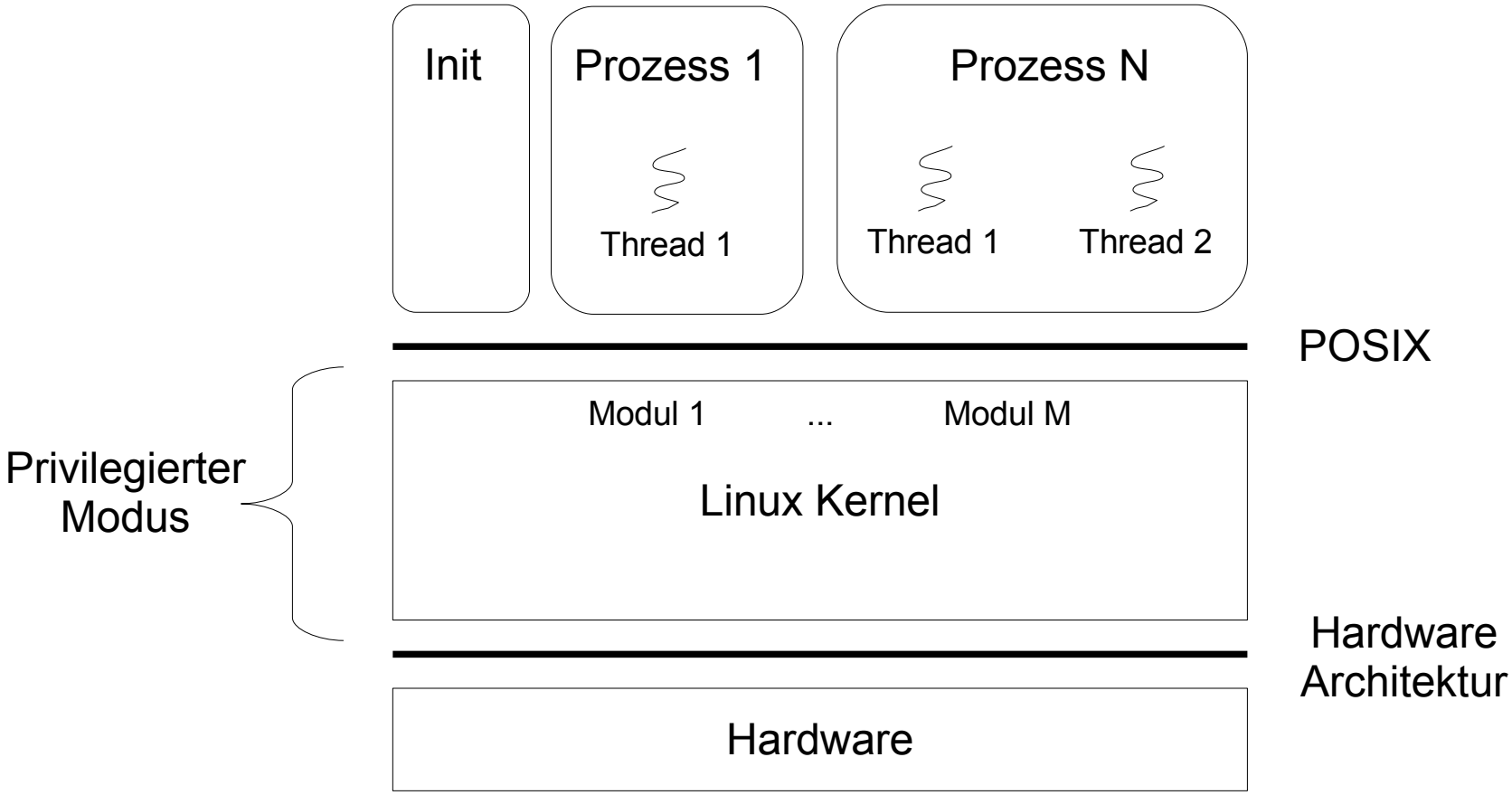
- Linux: Monolithischer Kernel mit und ohne Real-Time Patches
- Xen: Para-Virtualisierung von der University of Cambridge (UK)
- Minix 3: Kernel Betriebssystem von der Vrije Universiteit Amsterdam
- L4: Generische µKernel API von Jochen Liedtke

Methodik:

- Qualitative oder quantitative Analyse? → Qualitative Analyse!
- LoC Metrik: `find . -regex '.*\.\(c\|cc\)' -print0 | xargs -0 cat | wc -l`

Analyse
Linux

Ein freies Mehrbenutzer UNIX System für den PC



Analyse
Linux

Echtzeit:

- POSIX RT-Erweiterungen: RT-Scheduler, Seiten im phys. Speicher fixieren
- Jeder Geräte-Treiber kann das System blockieren

Partitionierung:

- Räumliche Partitionierung durch MMU oberhalb des Kernels
- User-Mode Scheduling innerhalb eines Prozesses möglich

Offene Standards:

- POSIX ist “quasi” frei

Modularität:

- Definierte Schnittstelle für Gerätetreiber
- Quellcode separiert: Architektur / Generisch

TCB:

- Monolithischer Kernel
- Kernel 2.6.9. für IA32 ohne Treiber < 150.000 LoC

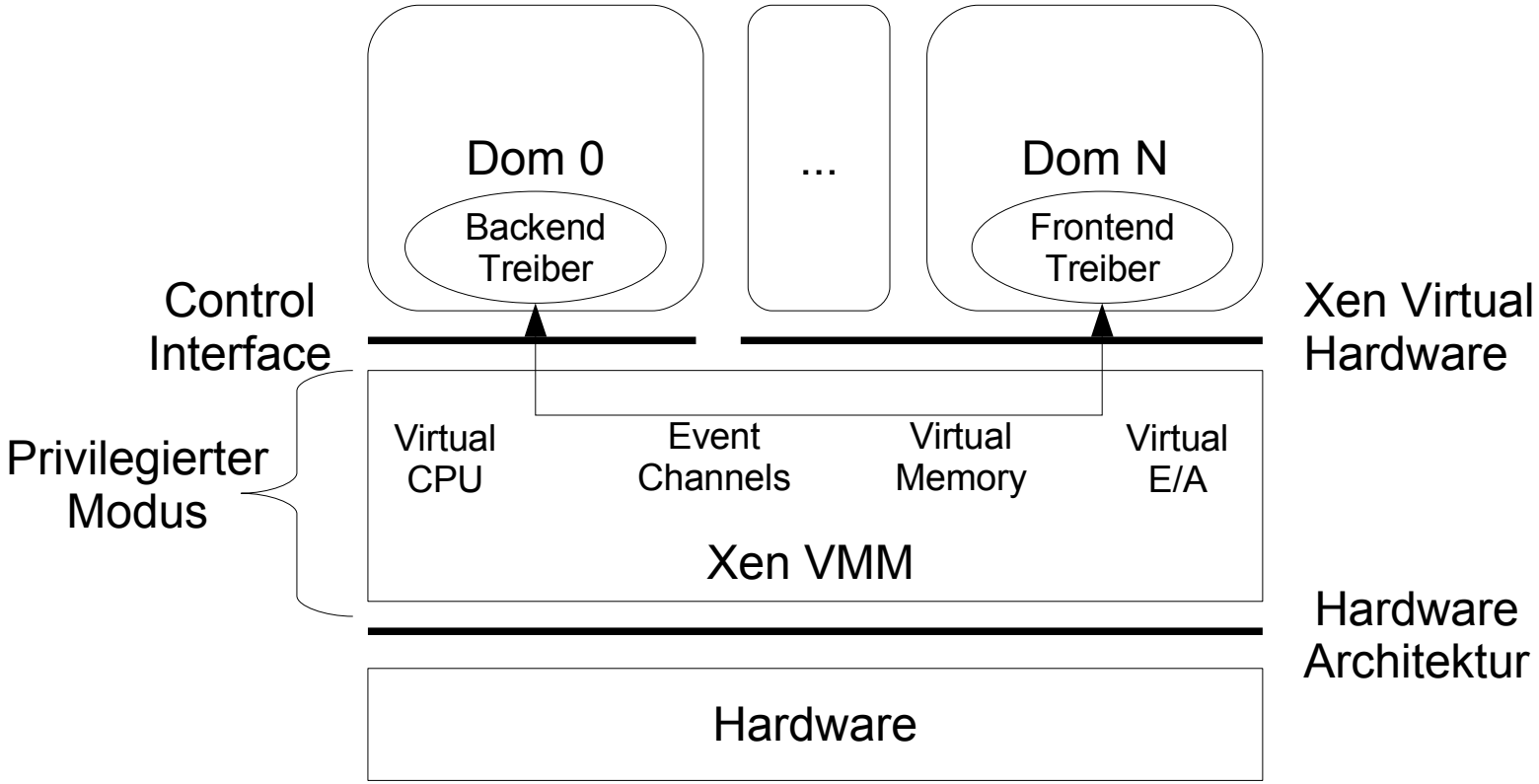
Analyse
Linux

Echtzeit Erweiterungen für Linux:

- **RTLinux**
  - Linux als Idle Thread des µKernel
  - µKernel und Linux im Kernel Mode → keine Partitionierung
  - Windriver hat am 20.02.07 bekannt gegeben, alle Rechte an RTLinux (inklusive Patent) erworben zu haben → nicht Open Source im engeren Sinn
- **RTAI/Adeos**:
  - ➔ Adeos I-Pipe wird als Kernel Modul geladen → keine Partitionierung
  - ➔ RT-Tasks im User- oder Kernel-Mode, Co-Scheduler für RT-Tasks
- **TimeSys**: Linux als Abonnement mit RT-Patches
- **MontaVista Linux**: Vanilla Kernel mit RT-Patches
  - ➔ Verringerte Interrupt-Latenz
  - ➔ Verringerter Anteil nicht-preemptiblen Kernel Codes

Analyse
Xen

Virtualisierte Hardware für bis zu 100 Gast-OS [Barham+, 2003]



Analyse
Xen

Echtzeit:

- EDF-Scheduler
- Split-Driver: Backend in Dom0, Frontend in Gast-Os

Partitionierung:

- Performance Isolation durch Virtualisierung von Speicher, CPU, E/A, Interrupts
- VMM Scheduler auf Domain Ebene und Scheduler des Gast-OS

Offene Standards:

- Virtuelle Hardware: Teilmenge der echten Hardware
- Xen Management API

Modularität:

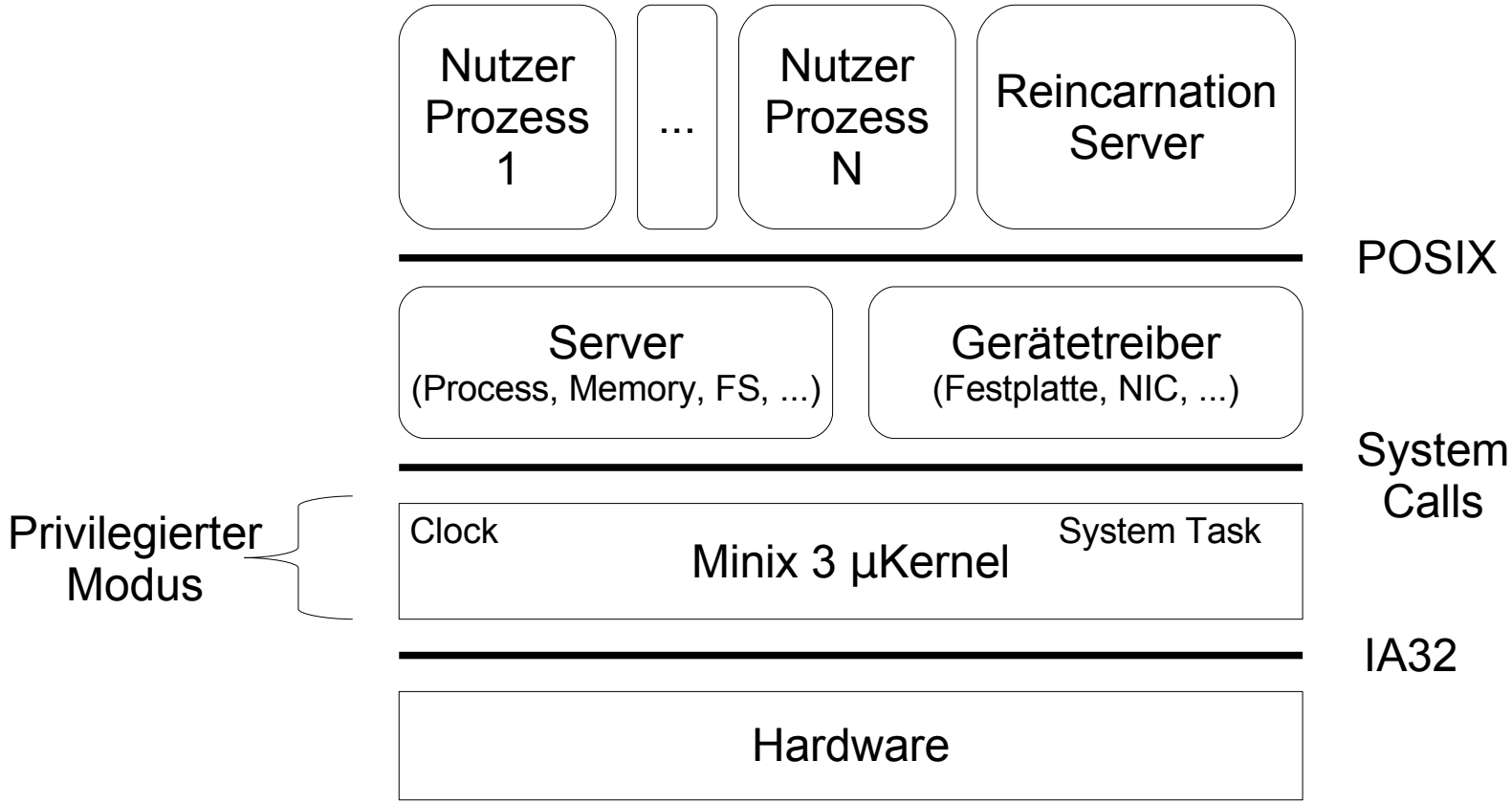
- VMM, Dom0 und Gast-OS sind logisch getrennt

TCB:

- Hypervisor + Dom0
- `xen-3.0.4_1-src/xen/arch/x86` < 60.000 LoC

Analyse
Minix 3

Ein Computersystem nach dem „TV model“ [Herder+, 2006]



Analyse
Minix 3

Echtzeit:

- Kein RT-Scheduler
- Speichermanager verwendet Swapping

Partitionierung:

- Adressräume werden durch MMU geschützt
- Scheduling auf Prozess-Ebene

Offene Standards:

- POSIX

Modularität:

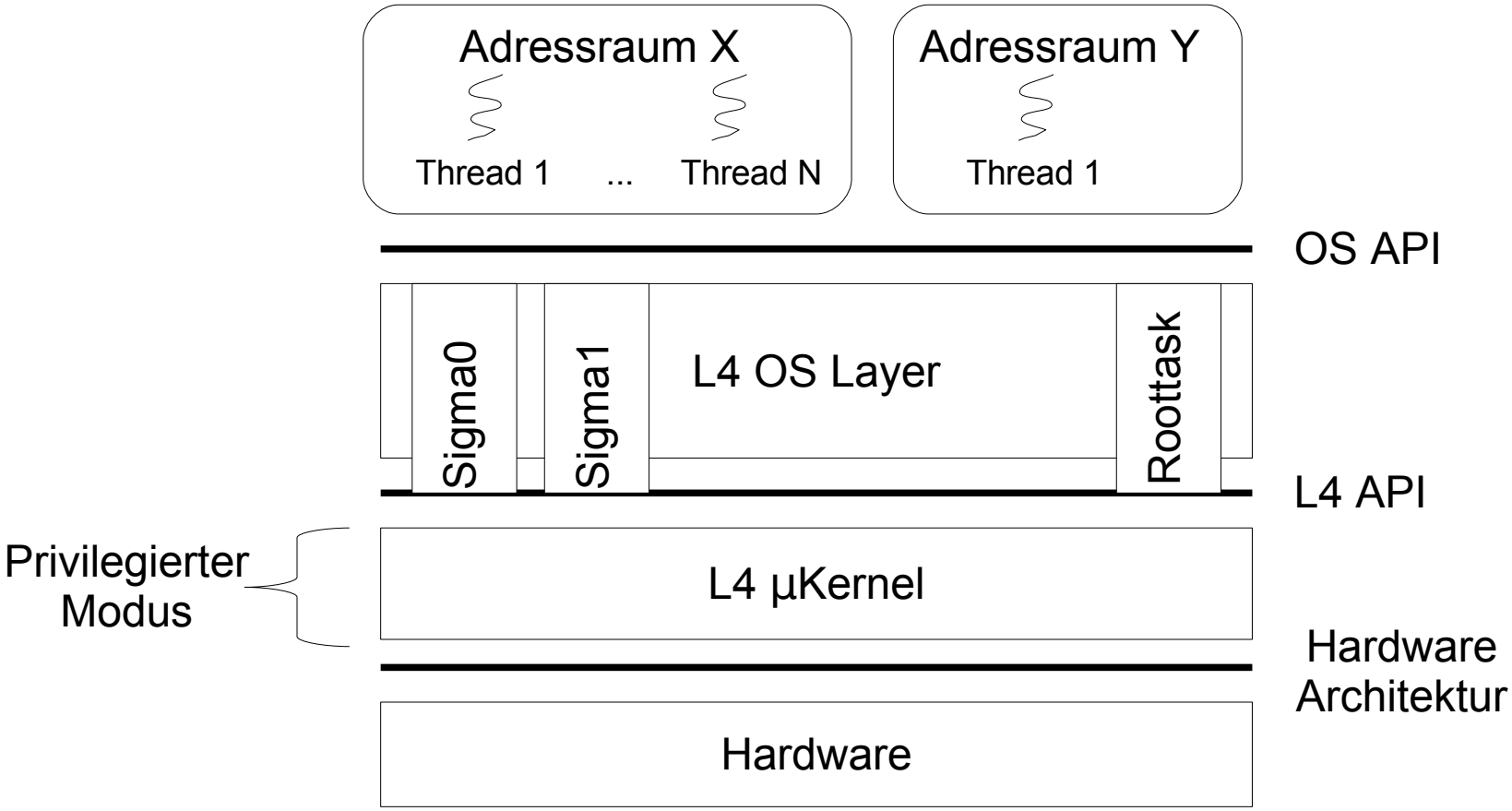
- Kernel und OS-Dienste eng verknüpft
- Isolation von µKernel, Servern und Geräte-Treibern

TCB:

- µKernel + Clock Treiber + System Task + Process- and Memory-Manager
- `src/kernel/*.c` < 5.000 LoC (nur IA32)

Analyse
L4

Abstraktion der Hardware durch Adressräume, Threads, IPC



Analyse
L4

Echtzeit:

- RR-Scheduler mit 256 Prioritäten
- Interrupts als IPC, Handling im User Mode

Partitionierung:

- Adressräume werden durch MMU geschützt
- Timeslice Donation ermöglicht Scheduler innerhalb von Adressräumen

Offene Standards:

- Kernel API in verschiedenen Versionen: V2, X.0, X.2, N1
- OS API hängt von System ab

Modularität:

- Isolation von µKernel, Servern und Geräte-Treibern
- L4Ka::Pistachio und Nachfolger trennen API / Architektur / Plattform

TCB:

- µKernel, Sigma0, root-Task
- L4Ka::Pistachio für IA32 < 15.000 LoC

Analyse
Entscheidung

And the winner is: **L4** (again [Benett, 2003])

Gründe:

- [Ruocco, 2006] → Harte Echtzeit
- Scheduling im User-Mode → Partitionierung
- Abstraktion von L4 ist generisch → Modularität
- L4 APIs sind frei verfügbar → Offener Standard
- µKernel → TCB
- Roadmap:
  - NICTA: seL4, L4.verified (Mathematisch korrekter µKernel)
  - TU Dresden: Verified Fiasco

Implementierung
Alternativen

L4 µKernel:

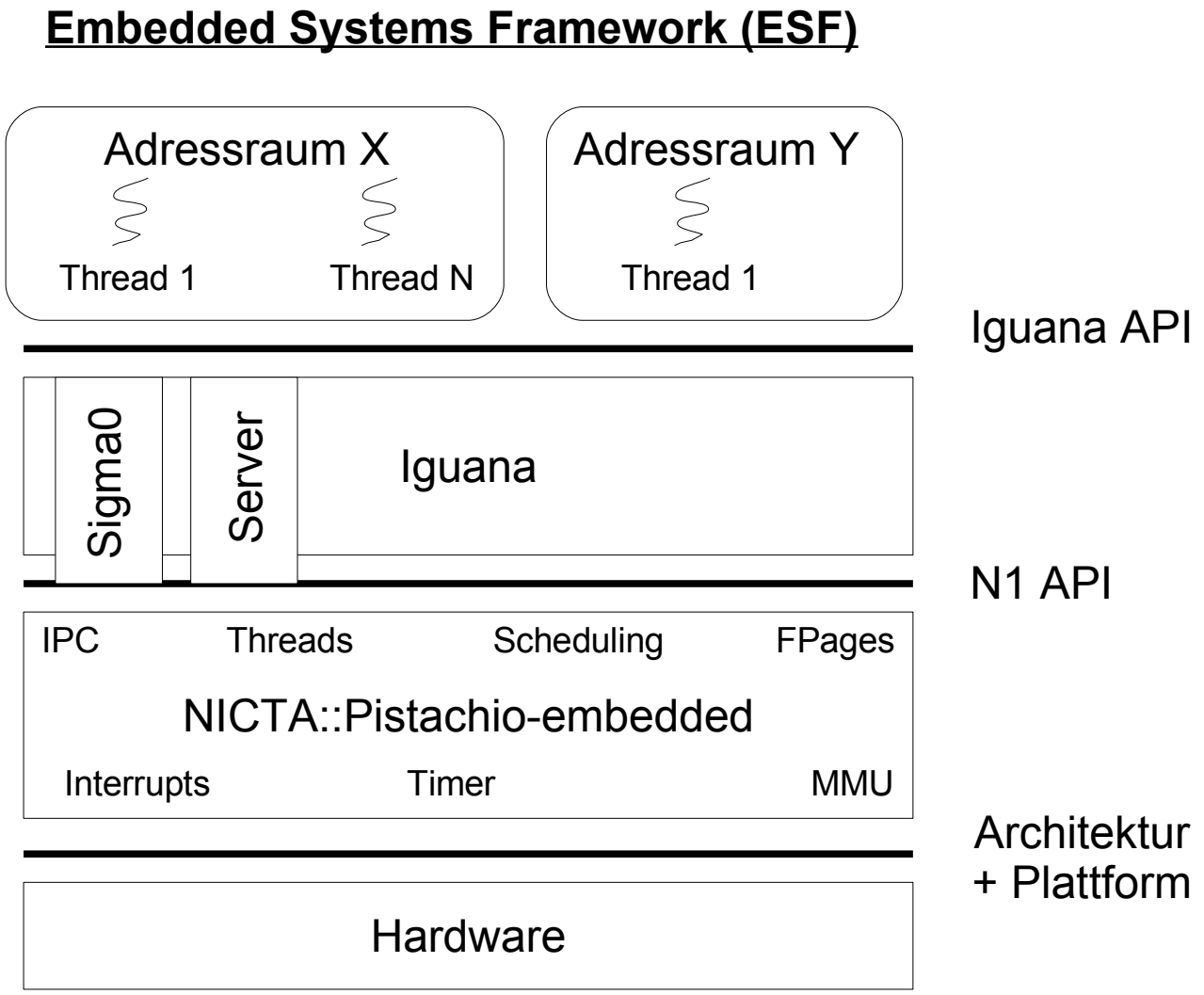
Name	*API*	*Projekt*	*Architekturen*
Pistachio	X.2	L4Ka	IA32/64, ARM, PowerPC32/64, …
Pistachio-embedded	N1	NICTA	IA32, ARM, MIPS
OKL4	OKL4	OKL	IA32, ARM
Fiasco	V2/X.0	DROPS	IA32

Linux auf L4:

Name	*Voraussetzung*
Afterburner	Pistachio + GCC-Afterburner
Wombat	Pistachio-embedded + ESF ◄—
L4Linux	Fiasco + L4Env
User Moder Linux	Linux

Implementierung

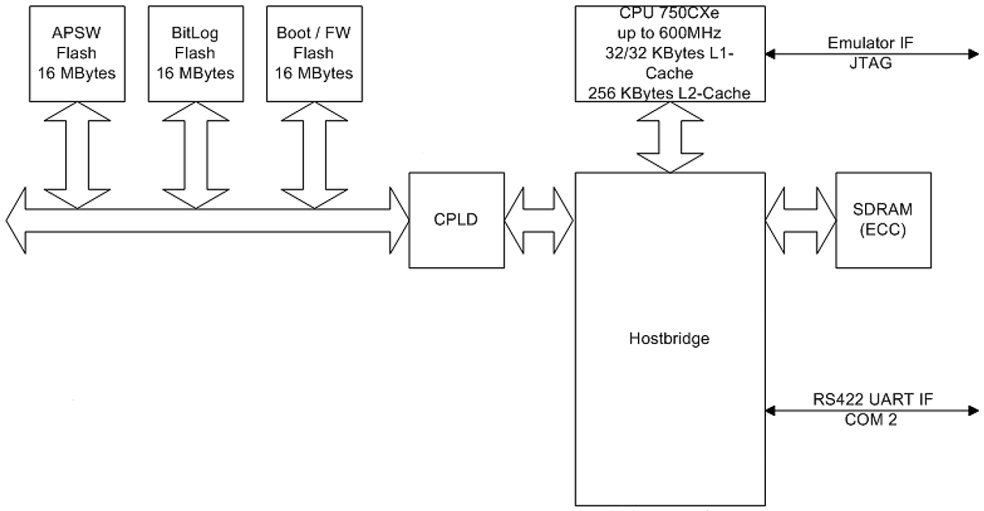
ESF-Modell



Implementierung
Hardware

Board Details (relevante Teile):

- PowerPC 750 Cxe @ 600 Mhz
- 128 MB SD-RAM
- 16 MB Boot-Flash
- 100 Mhz Bustakt
- Hostbridge Marvell GT-6426x



CPU Details:

- Decremeter arbeitet mit Geschwindigkeit Bustakt/4
- Getrennte Memory Mangement Unit (MMU) für Daten und Befehle
- 4 Block Adress Translations pro MMU
- Segmentiertes Paging mit 4 KB Seiten
- TLB mit 128 Einträgen pro MMU

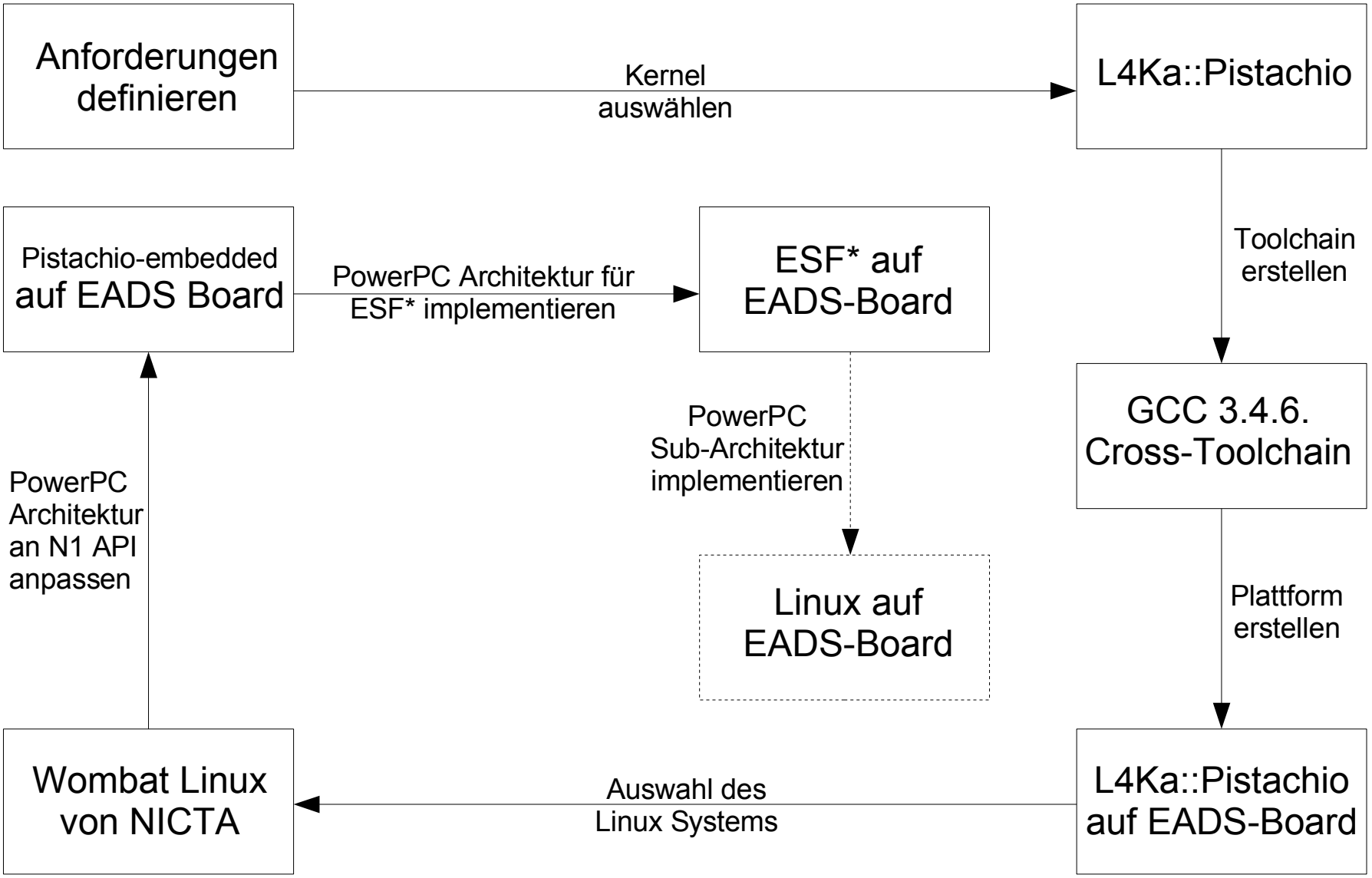
Implementierung
Hardware + IMA

Hardware Unterstützung der IMA Kriterien:

- Echtzeit:
  - → Decrementer Interrupt mit einer Auflösung von 0.04 µs (4/Bustakt)
- Partitionierung:
  - → Zeitlich:
    - Decremeter ermöglicht Scheduling mit variablen Zeitscheiben
    - Interrupts durch Geräte von inaktiven Partitionen lassen sich auf der Hostbridge maskieren
  - → Räumlich:
    - 2 CPU Modi: User Model und Supervisor Model
    - MMU für Daten und Befehle / mit BAT oder Paging
    - MMU schützt durch Memory Mapped I/O auch Geräte
- Modularität: 3 Ebenen Architektur des Prozessors
- TCB: n/a

Implementierung

Vorgehen



*) ESF = Embedded Systems Framework von National ICT Australia

Implementierung

Arbeitsschritte

1. Cross-Toolchain:
   - Host Plattform: IA32/Linux → Target Platform: PowerPC32/Elf
2. Build System:
   - PowerPC Architektur und Board in `SCons` integrieren
   - PowerPC Unterstützung zu `Dite` hinzufügen (Elf-Image Tool)
3. Architektur:
   - Startcode des Kernels überarbeiten: OpenFirmware, KMem, Reihenfolge
   - N1: keine Local Thread ID mehr / neuer User Thread Control Block (UTCB)
   - Eintrittsroutinen für 4 System Calls müssen an N1 angepasst werden
   - Neue Funktionen im Kernel Debugger implementieren
4. Plattform:
   - Boot-Code für Elfloader und Kernel
   - Interrupt Handling
   - Linker Skript
   - `getc()` und `putc()` für Marvell Debug I/O

Evaluation
Ergebnis

5.ESF:

- `libc`: Datentypen, Sprungfunktion, CRT0, E/A für Board
- `libcycles`: Zykluszähler der CPU auslesen
- `libiguana`: CRT0 für Threads
- `libl4`: Anpassen an N1 ABI/UTCB, neue KDebug Funktionen
- `libmutex`: `try_lock` Implementierung
- `drv_powerpc_timer`: User Mode Timer basierend auf Decrementer

Ergebnis:

- PowerPC Implementierung des µKernel an N1 API angepasst
- PowerPC Architektur zu ESF hinzugefügt
- ESF läuft auf dem Entwicklungsboard

Evaluation
IMA

Echtzeit:

- L4Ka::Pistachio(-embedded) nutzt statische Zeitscheiben der Länge: 1953 µs
- Für Kernel-Aktivitäten sind Interrupts ausgeschalten

Partitionierung:

- System Call `MemoryControl` wurde nicht implementiert → Caching von Seiten kann nicht verboten werden → BAT-Eintrag für Marvell-Hostbridge ist im User Model aktiv → World-writeable
- Inkrementelle Pagefaults funktionieren nicht (Fehler beim TLB-Update?) → Pager vergibt volle Rechte

Evaluation
IMA

Modularität: keine Veränderung zur Analyse

Offene Standards: keine Veränderung zur Analyse

TCB:

- Bekannte Bugs im µKernel:
  - `MemoryControl` System Call läuft „amok“
  - Smash-Thread Unit-Test schlägt fehl, bei etwa 20 Threads ist kein Kernel Memory mehr vorhanden → Speicherleck?
  - Fehler im TLB-Update → Bug in der Seitenverwaltung des PowerPC
- Iguana läuft nur nach diversen Bugfixes und Workarounds → weitere Bugs zu vermuten

Evaluation

Fazit

Fazit:

- Entwicklungsstadium Alpha. Noch nicht zum Einsatz in IMA geeignet!
- PowerPC Implementierung des µKernels muss überarbeitet werden.
- Der Weg ist richtig, aber zu lang für diese Arbeit.

Danke für die Aufmerksamkeit!

Bibliographie

- **[Barham+, 2003]**: P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield: „Xen and the Art of Virtualization“. New York. 2003
- **[Benett, 2003]**: M. D. Benett: „A Kernel For IMA Systems“. York. 2003
- **[Duden, 2001]**: Bibliographisches Institut: „Duden – Fremdwörterbuch“. Mannheim. 2001
- **[heise, 2003]**: heise online: „Embedded-Spezialist Wind River tritt Open-Source-Labs bei“. 2003
- **[heise, 2006]**: heise online: „Der Pinguin geht auf U-Boot-Jagd“. 2006
- **[Herder+, 2006]**: J. N. Herder, H. Bos, B. Gras, P. Homburg, A. S. Tanenbaum: „MINIX 3: A Highly Reliable, Self-Repairing Operating System. Amsterdam. 2006
- **[Liedtke, 1995]**: J. Liedtke: „On µ-Kernel Construction“. New York. 1995

Bibliographie

- **[Ruocco, 2006]**: S. Ruocco: „Real-Time Programming and L4 Microkernels“. Sydney. 2006
- **[Rushby, 1999]**: J. Rushby: „Partitioning in Avionics Architectures: Requirements, Mechansims, and Assurance“. Menlo Park. 1999
- **[Timmerman+, 2005]:** M. Timmerman, L. Pernell: „RTOS State Of The Art - Understanding RTOS Technology And Markets?”. 2005