Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

RealTime Linux

Paul Seidel

Seminar Prozessteuerung und Robotik WS 08/09

Lehrstuhl BS und Middleware Prof. Polze

Hasso-Plattner-Institut Potsdam

Übersicht

- Standard-Kernel
- Dual-Kernel
  - RTAI/LXRT
- In-Kernel
  - KURT/LiberRTOS
- Programmbeispiele

Standard-Kernel(1)

- Vanilla: Standard-Kernel ohne Veränderungen (kernel.org)
- Seit 1991 in Entwicklung
- Aktuelle Version 2.6.27.9[1]
- Monolithischer Kernel mit Modulkonzept
  - Kernel erweiterbar durch Kernelmodule
  - Gerätetreiber als Kernelmodule implementiert
  - mittlerweile auch Mikrokernel-Konzepte implementiert
    - FUSE – Filesystem in User Space

1: Stand 14.12.2008

Standard-Kernel(2)

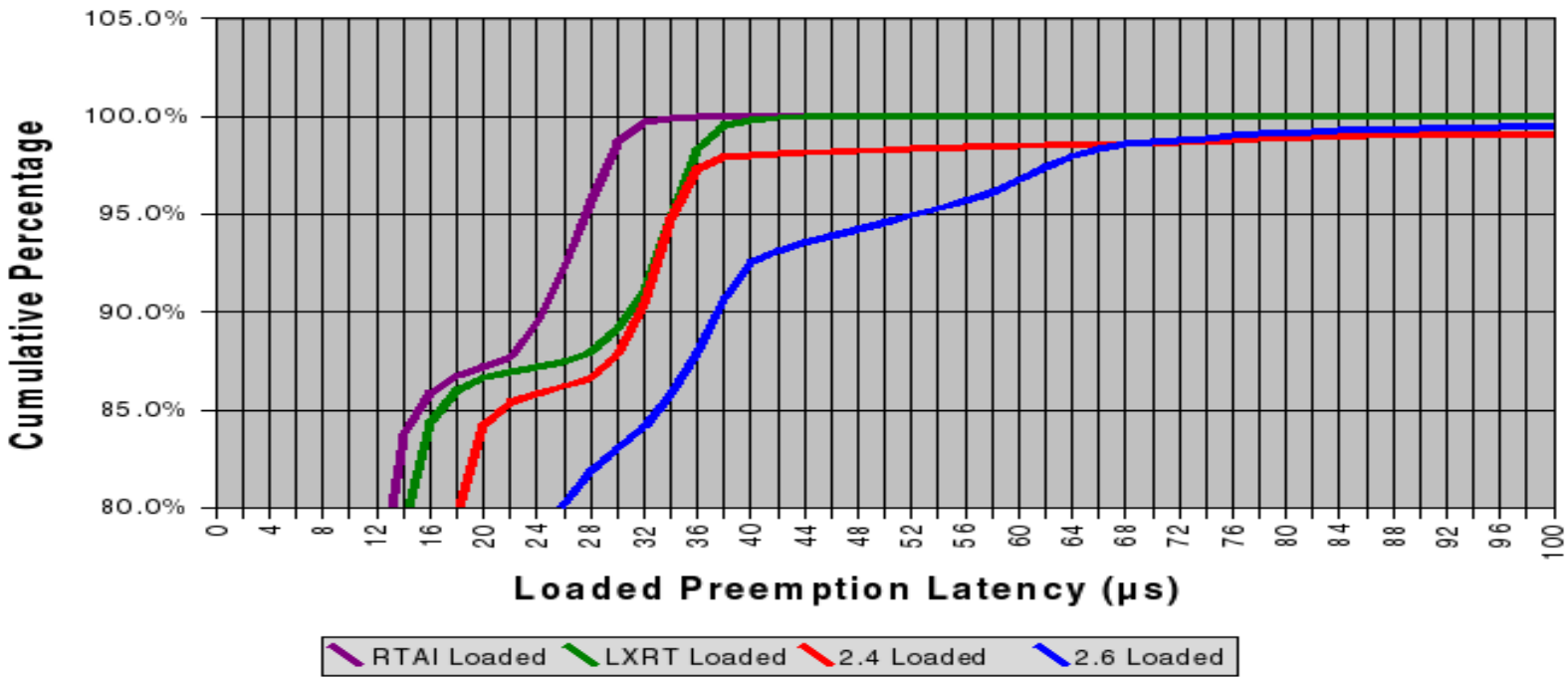
- Scheduler
  - alt: O(1)-Scheduler
  - neu: CFS
    - Completely Fair Scheduler
    - O(log n)
    - modelliert ideale Multitasking-CPU, welche jeden Task parallel mit derselben Geschwindigkeit ausführt
    - Task mit der höchsten Differenz zwischen erwarteter und tatsächlicher CPU-Zeit wird ausgewählt
    - Sortierung der Tasks über Rot-Schwarz-Bäume
    - Hintergrundprozessen blockieren i.A. interaktive Prozesse nicht

Standard-Kernel(3)

- seit Version 2.6 zum größten Teil präemptiv
- User-Mode-Prozesse können Kernel unterbrechen
  - Mehr Laufzeit für nicht-Kernel-Prozesse
- aber: Kernel nicht echtzeitfähig durch hohen Jitter
  - Atomare Kernelfunktionen müssen vollständig ablaufen
  - z.B. Spinlocks, Mutexes, Semaphoren
  - rechenintensiver Hardwarezugriff blockiert andere Tasks

Standard-Kernel(4)

Preemption-Latency-Vergleich (2.4, 2.6, LXRT, RTAI)



OS	Max. Loaded(µs)	Threshold 99.999% Loaded (µs)
Linux2.4	4446	760
Linux2.6	578	440
RTAI	42	40
LXRT	50	48

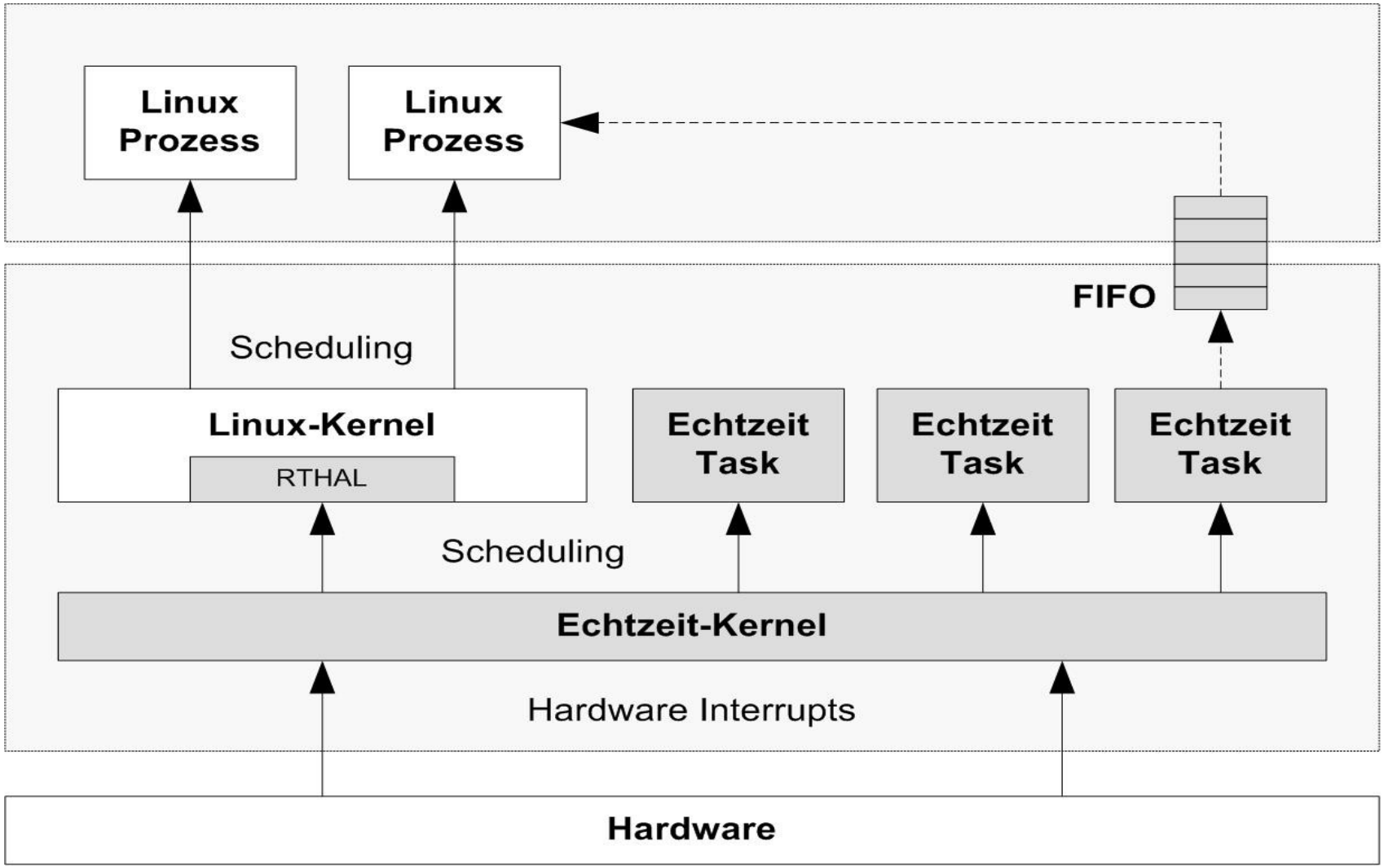
Standard-Kernel(5)

- Standard-Kernel ist nicht echtzeitfähig
  - hohe Unterbrechungslatenzen bei Interrupts
  - hoher Jitter bei Unterbrechungslatenzen unter Last
  - daher: Zeitauflösung zu gering

Dual-Kernel(1)

- Benutzung eines Mikrokernels unterhalb des eigentlichen Kernels
  - Abfangen aller Interrupts
  - Abarbeitung der meisten Interrupts im Mikrokernel
  - Nur einige Interrupts werden an Linux-Kernel weitergeleitet
- Beispiele: RTLinux, RTAI

Dual-Kernel(2)



RTAI(1)

- RTAI – Real Time Application Interface
- Dual-Kernel-Ansatz
- Kern ist ein Kernelmodul
- Echtzeit-Tasks als Kernelmodule
  - Starten mit /sbin/insmod
  - Beenden mit /sbin/rmmod
  - Auflisten mit /sbin/lsmod
- Mikrokernel schedult Linux-Kernel und Echtzeit-Tasks
- Linux-Kernel läuft mit niedrigster Priorität

RTAI(2)

RTHAL (real time hardware abstraction layer)

- Patch für den Kernel
- verändert Funktionspointer, welche auf Interrupt-Handling-Funktionen des Kernels zeigen
- wird RTAI-Kernelmodul geladen, zeigen Funktionspointer auf Interrupt-Handling-Routinen von RTAI
- Vorteile
  - da RTAI Kernelmodul, lässt sich RTAI je nach Bedarf an- und ausschalten
  - Fehlersuche vereinfacht

RTAI(3)

- Interrupt-Handling
  - Kernel benutzt Funktionspaar sti() und cli() zum Flaggen von Interrupts
  - cli(): Interrupt deaktivieren, sti(): Interrupt aktivieren
  - RTAI übeschreibt mittels RTHAL diese Funktionen
    - ruft Kernel cli(), so wird Interrupt in RTAI geflagt
    - RTAI arbeitet solche Interupts dann selbst ab
    - aber nur wenn Echtzeitanwendung einen Handler dafür registriert hat
    - sonst springt RTAI aus dem Interrupt-Kontext in das laufende Programm zurück

RTAI(4)

Kommunikationsmöglichkeiten

- zwischen Echtzeit-Tasks
  - Mailboxen
- zwischen Echtzeit-Tasks und Linux-Prozessen
  - Shared Memory
  - FIFOs

Timer

- Auslösung veranlasst Rescheduling
- einmalig
  - unterschiedlich lange Intervalllängen möglich
- oder: periodisch

LXRT

- ermöglicht Entwicklung von RTAI-Echtzeitanwendungen im Userspace
- API ähnlich zu Kernelspace-API (Ausnahme: POSIX-Threads)
- Funktionsweise
  - zu jedem LXRT-Prozess gehört ein Echtzeit-Task im Kernel-Space
  - LXRT-Prozess ruft Funktion aus API auf
  - Daten werden in Echtzeit-Task kopiert
  - Echtzeit-Task führt Arbeit durch
  - Daten werden zurück kopiert

LXRT

Vorteile:

- fehlerhafte Programmierung führt nicht zu Systemabsturz
- einfacher Zugriff auf Userspace-Bibliotheken
- schnellere Entwicklung möglich

Nachteile:

- LXRT-Prozesse sind normale Linux-Prozesse
  - von Linux-Scheduler teilweise abhängig
  - Overhead bei Systemaufrufen
  - im Gegensatz zu Kernelspace wird Speicher ausgelagert
    - locken des Speichers nötig (mlockall)

In-Kernel

Veränderung des Kernels selbst

- anderer Scheduler
- Erweiterung der Kernel-API um Echtzeitfunktionalität
- veränderte Interruptbehandlung

KURT/LibeRTOS

KURT-Linux

- Kansas University Real-time
- LibeRTOS: industrieller Ableger von KURT
- In-Kernel-Ansatz
- erstes System dieser Art
- verändertes Interrupt Handling und Thread Scheduling gegenüber Standard-Kernel
- Kommunikation mit Echtzeiterweiterung über speziellen Gerätetreiber (/dev/kurt)

KURT: Timer Interrupt Handling (1)

- Standard-Kernel nutzt Timer-Interrupt zum Rescheduling
- Timer-Interrupt wird in regelmäßigen Abstanden ausgelöst (~10ms)
  - minimale Timerauflösung daher ebenfalls 10ms
  - zu groß für Echtzeitanwendungen (häufig <10µs benötigt)

KURT: Timer Interrupt Handling (2)

- KURT nutzt Timer-Interrupt, um das nächste bevorstehende Event zu Schedulen --> Timer-Interrupt wird nicht periodisch ausgelöst
- KURT nutzt Time Stamp Counter zur Ermittlung der Systemzeit um einen hochauflösenden Timer bereitzustellen
  - Anzahl Prozessorticks seit Start der Maschine
  - Instruktion RDTSC auf x86-Prozessoren
  - Genauigkeit dennoch fragwürdig
    - unterschiedliche Werte je Prozessor auf Multicore-Maschinen
    - Energiesparmaßnahmen (Heruntertakten) verändern Tickrate
    - aber: TSC mit konstanter Rate auf neueren Intelprozessoren

KURT-Scheduling(1)

- vier verschiedene Modi für Scheduler
  - normal: Standard-Linux-Scheduler wird benutzt
  - SCHED_KURT_FOCUSSED
    - nur RT-Prozesse werden gewählt
    - Standard: Idle-Prozess
  - SCHED_KURT_PREFERRED
    - KURT_EXPLICIT und KURT_ANYTIME werden bevorzugt gewählt
    - ist kein RT-Prozess verfügbar, wird Linux-Scheduler gerufen
  - SCHED_KURT_MIXED (wie KURT_PREFERRED, aber:)
    - KURT_EXPLICIT wird bevorzugt gewählt
    - KURT_ANYTIME wird wie Nicht-RT-Prozess behandelt

KURT-Scheduling(2)

Prozess-Modi

- KURT_EXPLICIT
  - werden nur vom KURT-Scheduler geschedult
  - Submodi
    - KURT_EPISODIC (worst case execution time)
    - KURT_CONTINOUS (Unterbrechung durch Watchdogs)
- KURT_ANYTIME
  - wie KURT_EXPLICIT | KURT_CONTINOUS
  - aber: können auch vom Linux-Scheduler aufgerufen werden
- KURT_PERIODIC
  - nur ein Prozess darf KURT_PERIODIC sein
  - wird periodisch gerufen (Angabe in Mikrosekunden)

Interrupt Handling in KURT

- einige Interrupt belegen exklusiv CPU
- werden durch Semaphore in Software ersetzt
  - Interrupts somit durch KURT schedulbar
  - Real-Time-Tasks werden nicht durch Interrupts behindert

In-Kernel vs. Dual-Kernel

Vorteile von In-Kernel gegenüber Dual-Kernel

- Echtzeitanwendung immer im Userspace
- „normale Anwendung“ - kein Kernelmodul
- normaler Zugriff auf Kernel-Userspace-API
- externe Bibliotheken einfacher anwendbar

Vorteile von Dual-Kernel gegenüber In-Kernel

- einfachere Implementierung
- einfacher auf andere Betriebssysteme zu portieren
- durch Kernel-Module transparent und komplett abschaltbar

Programm-Beispiele

Quellen

- Comparative Analysis of RTLinux and RTAI, Ripoll, 2002
  - http://www.linuxdevices.com/files/misc/ripoll-rtl-v-rtai.html
- A comparison of hard real-time Linux alternatives, Laurich, 2004
  - http://linuxdevices.com/articles/AT3479098230.html
- http://en.wikipedia.org/wiki/Completely_Fair_Scheduler
- http://kernel.org
- http://people.redhat.com/mingo/cfs-scheduler/sched-design-CFS.txt
- KURT-Linux User Manual
  - http://www.ittc.ku.edu/kurt/papers/user-manual-DRAFT.pdf
- LibertOS: A Flexibly Configurable and Highly Capable Platform for Industrial Automation, Niehaus, 2005
  - http://www.ittc.ku.edu/techreview2005/presentations/Niehaus_L
- RTAI user manual