Real-Time Linux

Jens Peter Lindemann
Lehrstuhl Neurobiologie

13. Januar 2009

- Was ist ein RTOS?
- Linux-basierte RT-Lösungen
- RT-Erweiterungen des Mainline-Kernels
- *“What's good for RT is good for the Kernel“*

Real-Time oder Real-Fast?

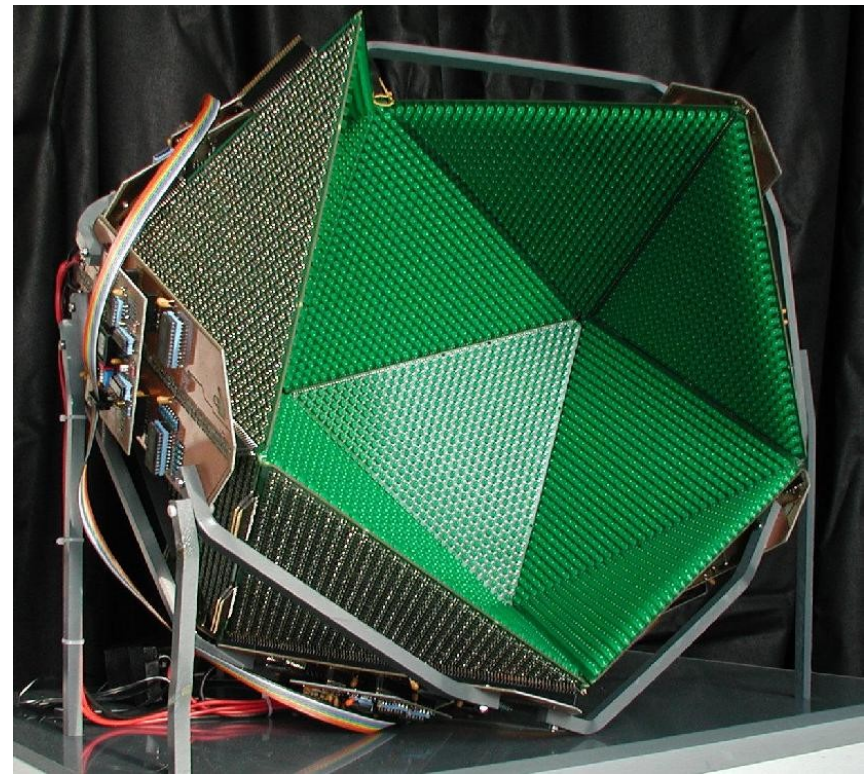
Ein Echtzeit-System arbeitet korrekt, wenn die Reaktion auf ein Event innerhalb einer spezifizierten Zeitspanne erfolgt.

- **Echtzeitsysteme sind *nicht* besonders schnell**
- **Trade-off: Vorhersagbarkeit gegen Performance**

Anwendungsbeispiel: Flimax

Visuelle Stimulation von Insekten (Fliegen, Bienen) mit natürlichen Bildfolgen

- **7168 grüne LEDs**
- **Grafik-Karte für D/A**
- **370 Updates pro Sekunde**
- **vorberechnete „Bilder“**
  - 128 x 224 x 24Bit



Nanokernel-Ansatz

- **Nanokernel für Realtime-Tasks**
- **Linux als Low-Priority Task unter Kontrolle des Nanokernels**

Beispiele:

- **RTLinux**
- **RTAI**

Echtzeit-Threads in Kernel-Modulen.

Kernel-Erweiterung

CONFIG_PREEMPT_RT

- Patch-Set von Ingo Molnar, Thomas Gleixner u. a.
- schrittweise Integration in den Mainline-Kernel

Komponenten:

- Threaded Interrupts
- Verbessertes Locking
- Priority Inheritance
- High resolution timers

Interrupt-Handling in Linux

- **Kernel-Code (Treiber) registriert Interupt-Handler**
- **Interrupt handling aufgespalten:**
  - „Top Halves“ erledigen die zeitkritischen Dinge sofort im Interrupt-Context
  - „Bottom Halves“ erledigen den Rest in „SoftIRQs“ oder „Taskslets“
- **Interrupt-Handling erzeugt Latenzen für alle Tasks**
  - „Interrupt Inversion“: Hoch priorisierte Tasks warten auf Interrupts niedrig priorisierter Tasks

Threaded Interrupts

- **Treiber registriert Interrupt Handler...**
  - Erzeugung eines Kernel-Threads für die Interruptbehandlung
  - Ein Thread je IRQ (shared Interrupts)
  - Prozesse können höhere Priorität haben, als IRQ-Threads

- **verbliebene Latenz durch Interrupt Inversion:**
  - Start der Interrupt-Service-Routine
  - Maskierung des Interrupts
  - Wecken des Interrupt-Threads
  - Rückkehr zum unterbrochenen Code

Threaded Interrupts: Ausnahmen

- **Timer Interrupt wird immer sofort behandelt**
  - Timed events
  - Scheduling
- **Treiber kann explizit `IRQ_NODELAY` erzwingen**

Threaded Interrupts: `ps`

```
  PID PRI RTPRIO CMD
    5  90     50 [softirq-high/0]
    6  90     50 [softirq-timer/0]
    7  90     50 [softirq-net-tx/]
    8  90     50 [softirq-net-rx/]
    9  90     50 [softirq-block/0]
   10  90     50 [softirq-tasklet]
   11  90     50 [softirq-sched/0]
   12  90     50 [softirq-hrtimer]
   13  90     50 [softirq-rcu/0]
   57  90     50 [IRQ-9]
  748  90     50 [IRQ-8]
  786  90     50 [IRQ-12]
  787  90     50 [IRQ-1]
 1374  90     50 [IRQ-14]
(...)
```

Kernel-Locking

Spin Locks

- Eingeführt zur Synchronisierung auf SMP-Systemen
- Für Kleine kritische Code-Bereiche
- Busy-Waiting: Performance Vorteil (kein Scheduler)
- Interrupts disabled (vermeidet busy waiting in ISR)
- CONFIG_PREEMPT: Durch Spin Locks gesicherter Code wird nicht unterbrochen

Problem für RT: Verschachtelte Spin Locks und große kritische Bereiche bewirken lange Latenzen

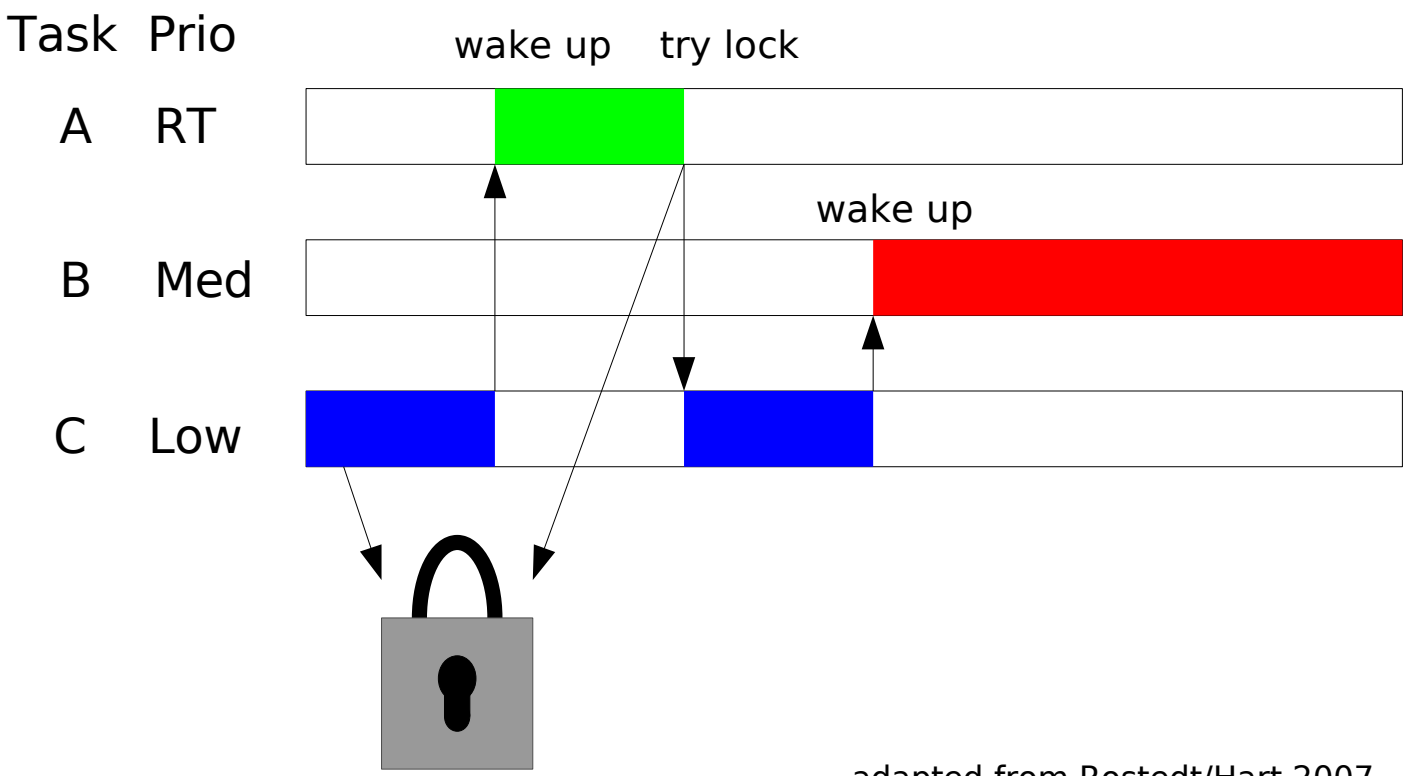
Sleeping Spin Locks (aka Mutex)

- **Konvertierung von Spin Locks in Sleeping Mutexes**
- **Neuer Typ: `raw_spinlock_t`**
  - markiert ein Lock als „echtes“ Spin Lock
  - Zur Compile-Zeit wird abhängig vom Argumenttyp zwischen Mutex und Busy-Wait entschieden.

„Looking into the header files of spinlock.h will drive a normal person mad.“

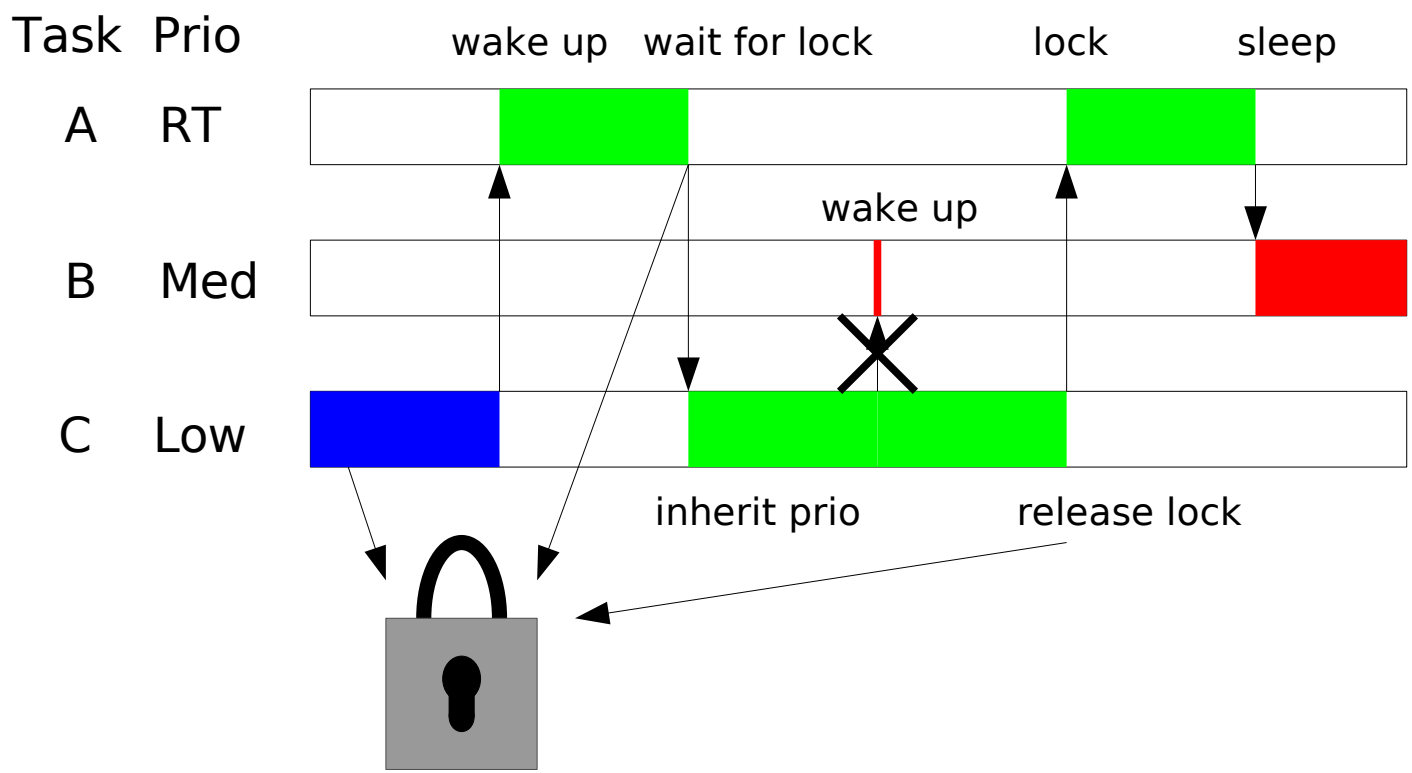
Problem: Binär verteilter 3rd -Party Treibercode...

Priority Inversion



adapted from Rostedt/Hart 2007

Gegenmitel: Priority Inheritance



adapted from Rostedt/Hart 2007

Priority Inheritance mit CONFIG_PREEMPT_RT

- **Priority Inheritance im Kernel (Mutex)**
- **Infrastruktur für Userspace Mutex mit PI-Support:**

  linux-2.6.2?/Documentation/rt-mutex-design.txt

High resolution timers

Linux Timer Architektur bisher: HZ, Ticks, Jiffies, CTW

Reorganisation:

- **Trennung von Timeouts und Timers**
  - Timer Wheel für Timeouts (softIRQ Thread)
  - hrtimers für Nanosekunden basierte Timer
- **Next Event Interrupt**
- **Reduktion von Redundanzen in Architektur-Code**

RT-Patches im Mainline Kernel

- RT-Patch deckt zahlreiche SMP Bugs (Race Conditions) auf
- `gettimeofday` redesign (2.6.18)
- IRQ Code cleanup (2.6.18)
- Trennung von Timer und Timeout Events (2.6.16)
- High Resolution Timers und DynTicks (2.6.12)
- Userspace Priority Inheritance (2.6.18)

RT-Anwendung

- **Hardware-Auswahl und -Konfiguration**
  - System Management Interrupts & CPU scaling
  - DMA bus mastering
  - VGA Console
- **Setzen der Echtzeit-Priorität**
- **Verhindern von Memory Page-faults**
  - mlockall(), Prozess-Stack!
  - File handling
- **Verwendung von PI-Mutex im User-Space**
- **Priorisierung der IRQ-Threads**

RT is good for the Kernel

„His (Ingo Molnar's) approach was not to force the Linux kernel into the RT world, but to bring beneficial parts of the RT world to Linux.“ (Roestdt & Hart 2007)

- **Big Kernel Lock (BKL)**
  - Spin Lock für große Teile des Kernels
  - Sonderbehandlung im Scheduler: Voluntary sleep
  - BKL als (RT-)Mutex: Code kann unterbrochen werden
- **Mutex/Completion ersetzen Semaphore**

DynTicks

Klassischer Linux Timer-Code: regelmäßiger Timer-Interrupt (z.B. 1000Hz linux 2.6 i386)

- **Problem: CPU „erwacht“ aus Power-saving states**
- **Lastproblem bei vielen virtualisierten Linux-Kernen**

Danymic ticks:

- **Programmierung des Timer-Interrupt auf den nächsten erwarteten Timer-Event**
- **basiert auf High Resolution Timer**

Links & Literatur

http://rt.wiki.kernel.org/

https://wiki.ubuntu.com/RealTime

Steven Rostedt & Darren V. Hart
Internals of the RT Patch
In: Proceedings of the 2007 Linux Symposium

Robert Love
Linux Kernel Development
Novell Press, Indiana, USA