Harte Echtzeit-Anforderungen mit Linux umsetzen

Mit Linux können Systeme mit harten Echtzeit-Anforderungen einfach umgesetzt werden. Doch welcher Ansatz ist der richtige? Und welche Latenzzeiten können damit erreicht werden?

JAN ALTENBERG*

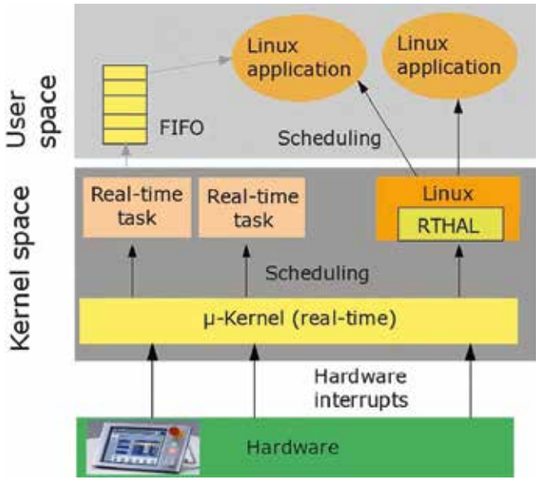

Bild 1: *Struktur von RTAI.*

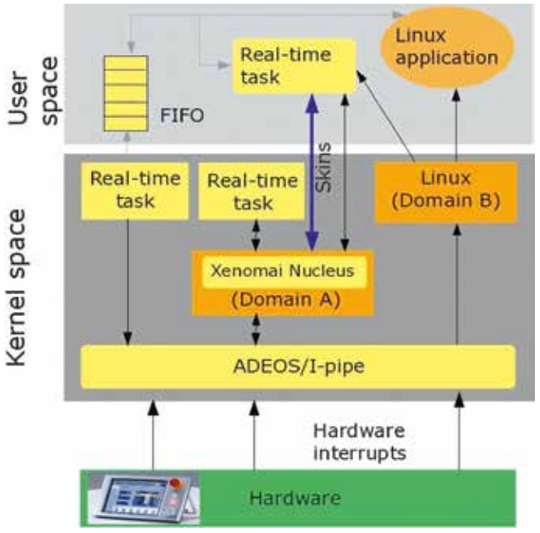

Bild 2: *Struktur von Xenomai.*

Bilder: linutronix GmbH

Linux ist aufgrund der hohen Anzahl unterstützter CPU-Architekturen, der nahezu unendlichen Anzahl von Treibern und der guten Portierbarkeit und Skalierbarkeit eines der derzeit leistungsfähigsten Embedded-Betriebssysteme. Auch Systeme mit Anforderungen an harte Echtzeit können mit Linux einfach umgesetzt werden.

Verfügbare Technologien für Echtzeit-Linux

Für Echtzeit mit Linux gibt es unterschiedliche Varianten und Ansätze. Doch welcher Ansatz ist der richtige? Und welche Latenzzeiten können damit erreicht werden? Dieser Artikel stellt unterschiedliche Technologien vor, mit denen harte Echtzeitfähigkeit unter Linux erreicht werden kann. Weiterhin wird aufgezeigt, welcher Jitter und welche Latenzzeiten mit diesen Technologien erreicht werden können.

Grundsätzlich gibt es zwei Ansätze, um Linux echtzeitfähig zu machen: Mikrokernel-Ansatz und In-Kernel Ansatz. Im Mikrokernel Ansatz werden alle Echtzeitaufgaben in ei-

Jan Altenberg
... ist Consultant, Trainer und Projektleiter bei der linutronix GmbH.

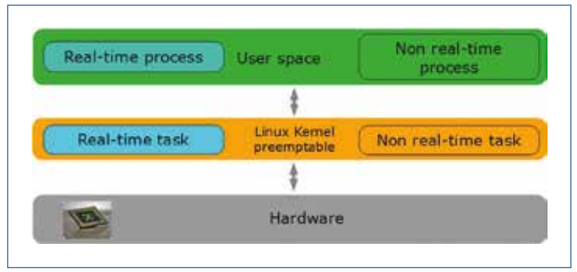

Bild 3: *Struktur von PREEMPT_RT.*

nem eigenen RTOS gehandhabt, Linux wird innerhalb dieses RTOS als niederpriore Task geschedult. Genau genommen ist hier also nicht von Echtzeit mit Linux die Rede. Wir sollten an dieser Stelle vielmehr von Echtzeit neben Linux sprechen. Der sogenannte In-Kernel Ansatz verfolgt das Ziel, Linux an sich echtzeitfähig zu machen (ohne darunterliegenden Mikrokernel). Im Folgenden wollen wit verschiedene Vertreter dieser Ansätze vorgestellen und deren Nutzen in der Praxis aufzeigen.

RTAI: Realtime Application Interface

Das Realtime Application Interface (RTAI) ist eine Entwicklung der Technischen Universität Mailand und entstand unter der Schirmherrschaft von Professor Paolo Mantegazza. RTAI ist ein klassischer Vertreter des Mikrokernel Ansatzes. Oberstes Designziel von RTAI ist und war es, die kleinstmöglichen Latenzzeiten auf einer gegebenen Hardwareplattform zu erzielen. Dieses Designziel bedingt diverse Einschränkungen für RTAI Applikationen. Weiterhin wird nur eine recht kleine Anzahl an Zielplattformen unterstützt (derzeit x86, x86_64 und diverse ARM Plattformen). In der Praxis finden sich kaum noch neue Projekte, die auf RTAI aufsetzen. Bild 1 in der Bildergalerie zeigt den prinzipiellen Aufbau von RTAI.

Xenomai: Echtzeit im Userpace einfach nutzen

Das Xenomai Projekt wurde im Jahre 2001 gegründet. Im Gegensatz zu RTAI erlaubt es Xenomai Echtzeit im Userpace relativ einfach zu nutzen (RTAI erlaubt dies nur sehr eingeschränkt). Die Besonderheit von Xenomai sind die sogenannten Skins, die es vereinfachen sollen, Applikationen von anderen Echtzeitsystemen (z. Bsp. VxWORKS, ...) nach Xenomai zu portieren.

Xenomai Skins bilden die API dieser Systeme ab. Xenomai unterstützt derzeit folgende Architekturen: PowerPC32, PowerPC64, x86, x86_64, Blackfin, ARM und ia64). Die zentralen Begriffe im Designkonzept von Xenomai stellen Xenomai Nucleus, die Interrupt Pipeline (IPIPE), die Hardware Abstraction Layer (HAL) und die System Abstraction Layer (SAL) dar. Die IPIPE kann bildlich als virtueller Interruptcontroller betrachtet werden. Sie organisiert das System in verschiedene Domains. Interrupts werden von IPIPE entgegengenommen und an die einzelnen Domains verteilt.

Nucleus beinhaltet die Xenomai-Core-Funktionalität. Diese ist zuständig dafür, alle notwendigen Ressourcen bereitzustellen, die Skins benötigen um die Funktionalität von Echtzeitbetriebssystemen nachbilden zu können. Der Hardware Abstraction Layer beinhaltet den Plattform und CPU abhängigen Code. Alle darüber liegenden Layer (darunter auch Nucleus) bauen darauf auf. Sowohl im Kernel als auch in der Applikation muss mit eigenen Bibliotheken und einer eigenen API gearbeitet werden. Es können also nicht die Standard Linux Bibliotheken verwendet werden! (dies gilt auch für RTAI). Bild 2 in der Bildergalerie zeigt das Konzept von Xenomai.

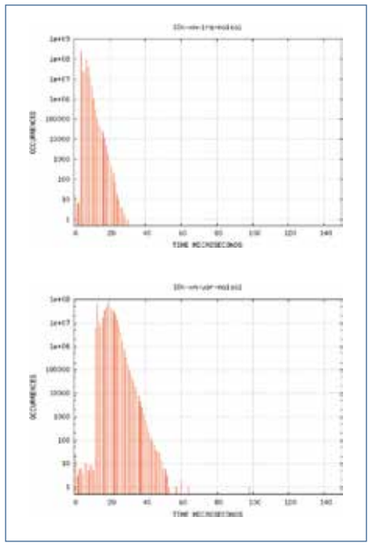

Bild 4+5: *Reaktionszeit Xenomai im Kernel (oben) und für die Applikation (unten).*

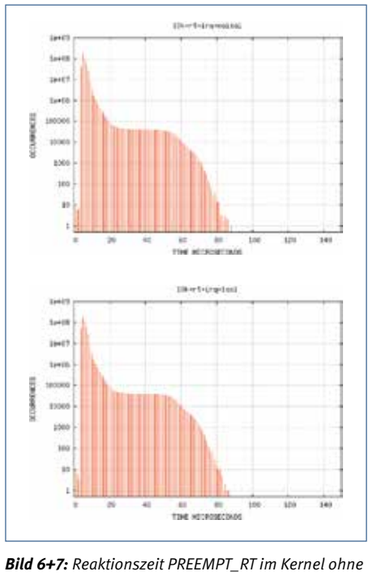

Bild 6+7: *Reaktionszeit PREEMPT_RT im Kernel ohne (oben) und mit isoliertem Core (unten).*

Der Linux Echtzeit-Preemption-Patch PREEMPT_RT

Der Realtime Preemption Patch entstand ursprünglich aus Arbeiten von Ingo Molnar und Thomas Gleixner. Vor allem Thomas Gleixner ist heute die treibende Kraft bei der Entwicklung von PREEMPT_RT. Im Gegensatz zu RTAI und Xenomai macht PREEMPT_RT den Linux Kernel an sich echtzeitfähig. Dies wird im Besonderen durch folgende Mechanismen erreicht:

- Sleeping Spinlocks: Spinlocks werden durch RT Mutexe ersetzt. Raw Spinlocks ersetzen die Eigenschaft der ursprünglichen Spinlocks
- Threaded Interrupt Handlers: Interrupt Handler laufen per Default nicht im harten Interruptkontext, sondern als Kernelthread.

Viele Mechanismen, die ursprünglich in PREEMPT_RT entwickelt wurden, haben schon lange Ihren Weg in den Mainline-Linuxzweig gefunden: High Resolution Timer (Hochauflösende Timer unabhängig vom Scheduler Tick), Priority Inheritance, generisches Interrupthandling für alle Architekturen und bereits in 2.6.30 die Threaded Interrupt Handler.

Weiterhin hat sich die Linux-Entwicklergemeinde schon im Jahre 2006 darauf geeinigt, dass Preempt RT in den Linux Kernel integriert wird. Mit den Arbeiten des OSADL und einer neuen Working Group der Linux Foundation (RTL, gegründet Oktober 2015; Gründungsmitglieder sind u.a. Google, OSADL, Intel, ARM, Texas Instruments, National Instruments, Altera (heute Intel FPGA) und andere) wurde erst kürzlich ein weiterer großer Schritt in diese Richtung getan. Aufgrund der vielen Vorteile und der großen Akzeptanz in der Linux Community hat sich PREEMPT_RT in den letzten Jahren als de-facto Standard für Echtzeitlinux durchgesetzt.

Neben den vielen technologie-bedingten Vorteilen ist noch die Tatsache erwähnenswert, dass sich das OSADL in umfangreichem Maße mit der Qualitätssicherung für PREEMPT_RT basierte Systeme befasst. Hierfür werden in einer Testfarm unzählige Benchmarks und Latenzzeitmessungen auf einer Vielzahl von Hardware durchgeführt (https://www.osadl.org/QA-Farm-Realtime.qa-farm-about.0.html).

Weiterhin bietet der Realtime Preemption Patch den großen Vorteil, dass Echtzeitapplikationen als POSIX Realtime Applikationen geschrieben werden können. Es wird keine spezielle API verwendet. PREEMPT_RT unterstützt eine Vielzahl von Architekturen (PowerPc, x86, x86_64, MIPS, ARM, ...).

Wie Bild 3 zeigt, integriert PREEMPT_RT die Echtzeitfunktionalität "nahtlos" in den Linux Kernel. Auch die Entwickler anderer Projekte haben die Vorzüge von PREEMPT_RT bereits erkannt. Xenomai 3 bietet Unterstützung für PREEMPT_RT. Dies ermöglicht den Einsatz von Xenomai Skins auf PREEMPT_RT-Kerneln.

Evaluierung der verschiedenen Ansätze für Echtzeit-Linux

Zur Gegenüberstellung von Mikro-Kernel und In-Kernel Ansätzen wurden vergleichende Messungen auf einer ARM Cortex A9 CPU durchgeführt. Xenomai wurde als Vertreter der Mikro-Kernel Technologie gewählt, PREEMPT_RT als Vertreter der In-Kernel Technologie. Gemessen wurde die Reaktionszeit auf ein externes Signal, welches mit einer Frequenz von 10kHz generiert wurde.

Zur Durchführung der Messungen wurde die OSADL Latency Box verwendet (https://www.osadl.org/uploads/media/OSADL-Latency-Box.pdf). Gemessen wurde jeweils die Reaktionszeit für Kernel und Applikation. Alle Messungen wurden unter denselben Bedingungen und 100% CPU Last durchgeführt (erzeugt mit dem Programm „hackbench“).

Ergebnisse Xenomai: Bild 4 zeigt die Reaktionszeit auf das Event im Kernel. Bild 5 stellt die Latenzzeiten für eine Applikation dar, die auf das Event wartet. Der Worst-Case liegt bei knapp unter 100 Mikrosekunden.

Ergebnisse PREEMPT_RT: Bild 6 + 7 in der Bildergalerie zeigen die Reaktionszeiten im Kernel unter PREEMPT_RT. Durch Isolieren eines Cores lässt sich der Worst-Case noch etwas verbessern.

Bei den Messungen für die Applikation schneidet PREEMPT_RT etwas besser ab als

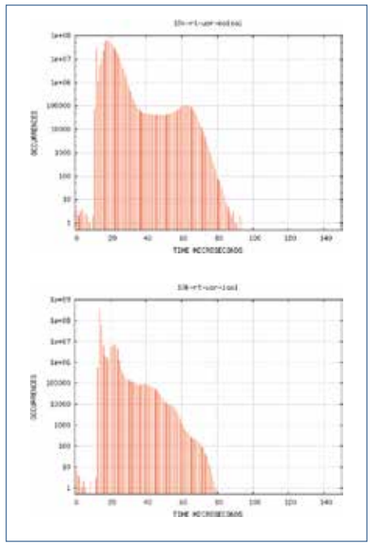

Bild 8+9: *Reaktionszeit PREEMPT_RT in der Applikation ohne (oben) und mit isoliertem Core (unten).*

Bild 10: *FIQ basierte Lösung auf PREEMPT_RT.*

Xenomai. Der Worst-Case liegt bei etwas über 90 Mikrosekunden.

Durch Isolieren eines Cores lässt sich das Ergebnis auf 80 Mikrosekunden verbessern: Nun ist der Vergleich der Reaktionszeiten im Kernel nicht ganz fair, denn bei Xenomai sprechen wir hier ja von einem Mikrokernel und nicht vom Linux Kernel. Denn genaugenommen arbeiten wir den Code ja nicht im Linux Kontext ab (wie es bei Xenomai der Fall ist). Daher wurde für PREEMPT_RT noch eine weitere Messung durchgeführt: Das Abarbeiten des kritischen Codepfades im FIQ Kontext (den viele ARM basierte SOCs bieten). Der FIQ lebt in seiner „eigenen Welt“, es ist aber möglich einen FIQ Handler aus Linux heraus zu registrieren!

Bild 8 zeigt die Ergebnisse einer FIQ-basierten Lösung (basierend auf einem PREEMPT_RT Kernel). Der Worst-Case ließ sich hier auf 30 Mikrosekunden verbessern! Dieser einzelne „Ausreißer“ lässt sich mit großer Wahrscheinlichkeit auf ein Hardwareproblem zurückführen. Auf anderen Plattformen konnten mit diesem Ansatz Reaktionszeiten von < 10us erreicht werden!

Fazit: Hervorragende Echtzeit-Eigenschaften mit Linux

Linux besitzt mit der passenden Erweiterung hervorragende Echtzeiteigenschaften. Aufgrund der hohen Akzeptanz in der Entwickergemeinde und der einfachen Handhabbarkeit hat sich hierfür der sogenannte PREEMPT_RT Ansatz als Standard etabliert. Die Latenzzeiten dieses In-Kernel Ansatzes sind auf Applikationsebene vergleichbar mit denen von Mikrokernel (wie Xenomai).

Die Mikrokernel können lediglich im Kernel bessere Latenzen erreichen, wobei hier zu berücksichtigen ist, dass hier nicht im Linux Kontext gearbeitet wird, sondern im Mikrokernel (und somit auch mit dessen Restriktionen und API). Wer bereit ist, für bessere Latenzzeiten Restriktionen in Kauf zu nehmen, kann auf ARM basierten Systemen auch mit einer FIQ-Lösung und PREEMPT_RT arbeiten. Hiermit sind teilweise Latenzzeiten unter 10 Mikrosekunden zu möglich.

PREEMPT_RT bietet in Summe den besten Tradeoff aus geringen Latenzzeiten und einfacher Handhabbarkeit. Weiterhin ist zu beachten, dass Organisationen wie das OSADL die Qualität des PREEMPT_RT Ansatzes kontinuierlich prüfen.

Eine weitere gute Nachricht ist die Tatsache, dass die Linux Foundation das RTL Projekt ins Leben gerufen hat, das die Integration in den Linux-Mainlinekernel über die nächsten Jahre finanziert und vorantreibt. Diese Tatsachen geben dem Anwender zusätzliche Planungssicherheit für zukünftige Projekte und die Pflege bestehender Produkte! // **SG**

linutronix

ELEKTRONIKPRAXIS Nr. 4 19.2.2019