linutronix

LINUX und Echtzeit

Eine Übersicht prinzipieller Lösungsansätze

"Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT."

Linus Torvald, Linux Kernel Summit 2006:

Kurzfassung:

Ausgehend von der theoretischen Vorstellung der verschiedenen Ansätze, wie ein Betriebssystem so geändert werden kann, dass es Echtzeitfähig ist, wird auf die Umsetzung dieser Ansätze in echte Produkte für Linux eingegangen. Diese werden kurz mit Ihren prominentesten Vertretern vorgestellt.
Im Abschluss gibt es einen Ausblick auf die Situation mit dem aktuellen Linux Kernel 3.x und der Weiterentwicklung des Preempt-RT Patches.

Echtzeit – Definition und Anforderungen

Leider gibt es keinen eineindeutigen Begriff „ECHTZEIT". Die folgenden Festlegungen sollen für das hier Geschriebene die Basis bilden.
Das Ziel eines Echtzeit-Betriebssystems (RTOS – ***r***eal-***t***ime ***o***perating ***s***ystem) ist es, ein vorhersagbares und deterministisches Verhalten bereitzustellen. Ein RTOS stellt daher die Infrastruktur bereit, die bei korrekter Benutzung garantiert, dass vorgegebene Reaktionszeiten im Allgemeinen (weiche Echtzeit) oder aber in jedem Falle deterministisch (harte Echtzeit) eingehalten werden. Die wichtigsten technischen Faktoren, um dieses Verhalten zu erreichen, sind daher minimale Interrupt Latenzzeiten und Thread Um-schaltzeiten.

Ein weiterer, sehr wichtiger Punkt für die Qualität eines Echtzeitsystems ist die Unterstützung periodischer Prozesse. Es muss sichergestellt sein, dass eine dezidierte Task immer innerhalb einer definierten Periode zur Ausführung gelangt. Hierzu muss ggfs. eine andere, laufende Task unterbrochen werden können (*preemption*).

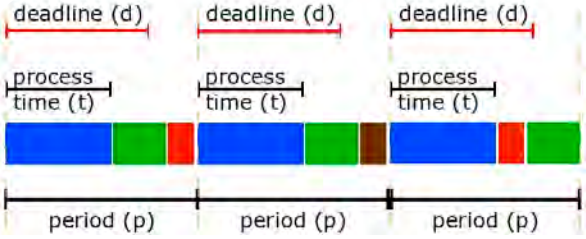

Abb. 2: Periodic task

Abschließend sei hier noch eines ganz klar gesagt. Echtzeit bedeutet nicht High Performance oder höchster Durchsatz. Im Gegenteil, Echtzeitfä-higkeit kann sogar den Durchsatz verringern, wegen der Unterbrechbarkeit von Prozessen. Ein Ver-halten, das man beispielsweise bei großen Servermaschinen beobachten kann. Echtzeit heißt auch nicht unbedingt und alleine minimale Latenzen zwischen einem Ereignis und der Reaktion darauf.
Echtzeit heißt deterministisches Verhalten gegenüber (externen) Ereignissen. Echtzeit heißt Vermeidung von „Überraschungen".
Es gibt noch weitere Herausforderungen, die gelöst werden müssen, wenn ein Betriebssystem echtzeitfähig sein soll. Auf diese wird im Rahmen dieses Textes nicht eingegangen

Wie bekommt man „Echtzeit" in ein Betriebssystem? Darauf gibt es 4 prinzipiell mehr oder weniger unterschiedliche Antworten. Und jede Antwort hat eine entsprechende Inkarnation als frei oder kommerziell erhältliches Produkt gefunden.

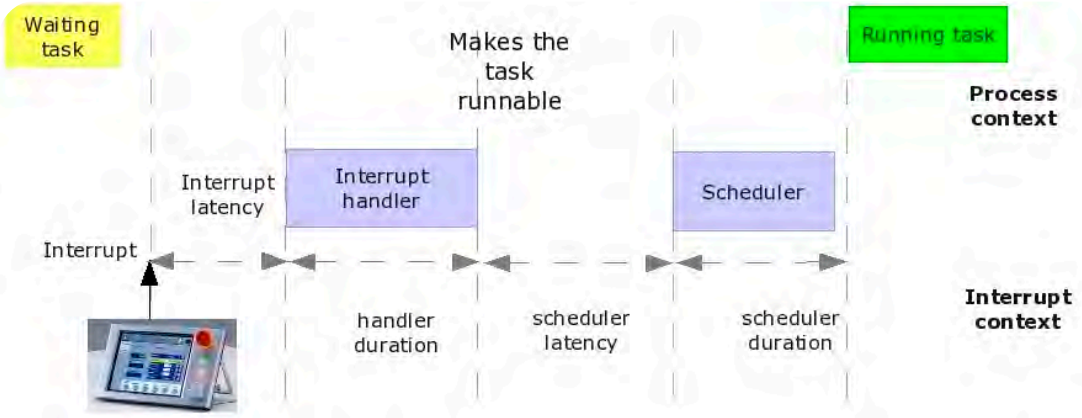

Abb.1: Definition Latenzzeiten

Lösungsansätze

Thin Kernel oder auch Micro Kernel Ansatz.

Hier wird ein zweiter, kleiner Kernel zusätzlich zum Linux Kernel genutzt. Die Aufgabe des 2. Kernels ist die Bereitstellung der Infrastruktur für die Echtzeit Tasks und das Management der Interrupts. Der Thin Kernel fängt eventuell auftretende Interrupts ab und sorgt damit dafür, dass der Linux Kernel den Thin Kernel nicht unterbrechen kann. Damit kann man dann sogar Hard Real-Time Anforderungen erfüllen.

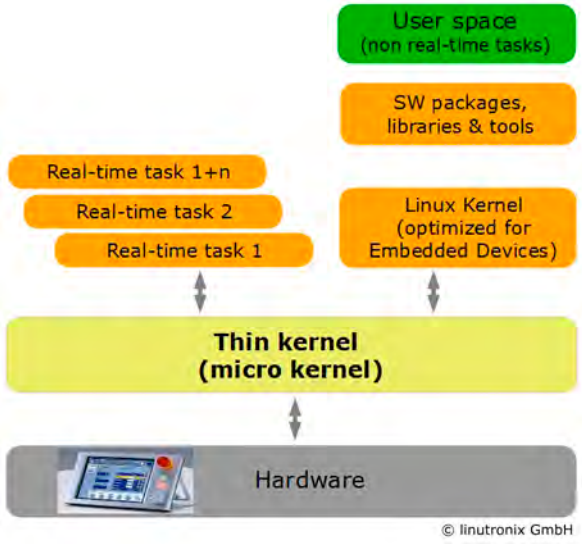

Abb. 3: Thin Kernel approach

Dies klingt zunächst einmal sehr gut, hat aber auch gravierende Nachteile. Da beide Kernel – und damit die Echtzeit Tasks und die Nicht-Echtzeit Tasks – unabhängig voneinander sind, ist die Entwicklung von Applikationen und vor allem deren Debugging, nun, zumindest sehr viel schwieriger als in einem Einzel-Kernel System. Dazu kommt noch, dass die Linux Anwendungen durch die gewählte Konstruktion nicht notwendigerweise den vollen Leistungsumfang von Linux nutzen können.

Die wichtigsten Vertreter dieses Ansatzes sind RTLinux (jetzt eine proprietäre Lösung von Wind River) und RTAI (***r***eal-***t***ime ***a***pplication ***i***nterface).

Nano-Kernel

- das ist der zweite Ansatz. Das Prinzip unterscheidet sich nicht viel vom Thin-Kernel Ansatz. Dieser Ansatz ist allerdings einen Schritt radikaler bei dem, was der zusätzliche Kernel leisten muss. Im Ergebnis handelt sich dabei weniger um einen Kernel als um eine Hardware Abstraktionsschicht (HAL= ***h***ardware ***a***bstraction ***l***ayer). Dieser HAL stellt die singulär vorhandenen HW Ressourcen,

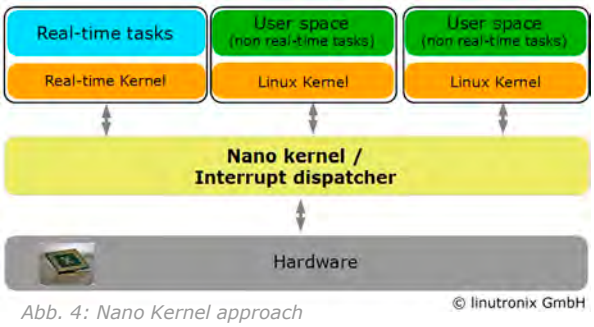

Abb. 4: Nano Kernel approach

vor allem auch die Interrupts, verschiedenen Betriebssystemen zur Verfügung. In seiner Grundkonzeption ähnelt ein Nano-Kernel mehr einem Hypervisor Typ I als einem kompletten Kernel. Auch ist seine Wirkungsweise sehr ähnlich den Arbeitsaufgaben eines Hypervisors.

Ein Vertreter dieses Ansatzes ist ADEOS (= ***a***daptive ***d***omain ***e***nvironment for ***o***perating ***s***ystems). Die Produktvariante in Zusammenhang mit Linux ist Xenomai.

Ressource Kernel

ist der dritte Ansatz, um das Thema Echtzeit mit Linux zu verheiraten. Hier wird der Kernel durch ein Modul erweitert, welches die Zuteilung von definierten Ressourcen verwaltet. Jede dieser Ressourcen wird über verschiedene Parameter wie benötigte Rechenzeit, Dauer der Wiederholungsperiode und der Deadline verwaltet. Über definierte APIs können (Echtzeit-)Tasks diese Ressourcen anfordern. Das Ressourcen-Modul übernimmt die Verwaltung dieser Anforderungen (z. Bsp. anhand von Earliest-Deadline-First Anforderungen) und übergibt sie an den Kernel, der dann die dynamische Zuteilung von Rechenzeiten (gemäß der Priorisierung) vornimmt.

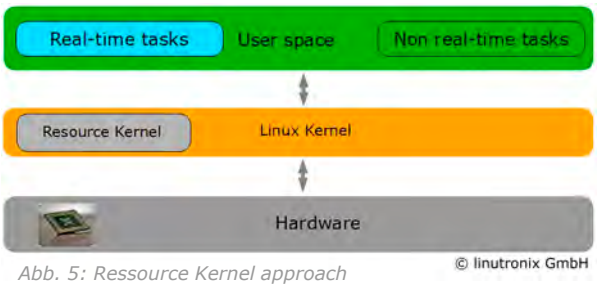

Abb. 5: Ressource Kernel approach

Dieser Ansatz wurde hauptsächlich an der Carnegie Mellon University entwickelt.

Seine Entsprechung im Linux Umfeld fand dieser Ansatz in der TimeSys Linux/RT Implementierung.

Echtzeit-Betriebssystem

Last but not least der 4. Ansatz – der wohl Geradlinigste, aber nicht unbedingt der Einfachste. Das Betriebssystem wird für den Einsatz als Echtzeitsystem ausgelegt und entwickelt. Ist das Betriebssystem aber schon existent wie im Falle von Linux und primär nicht als Echtzeitsystem ausgelegt worden, dann wird es spannend. Linux soll hierbei zu einem RTOS weiter entwickelt werden durch Hinzufügen von Funktionen wie Preemption Fähigkeit des Kernels, Prioritätsvererbung, Echtzeit Scheduler, High Resolution Timer und Threaded Interrupts, um nur einige der wichtigsten Änderungen zu nennen.

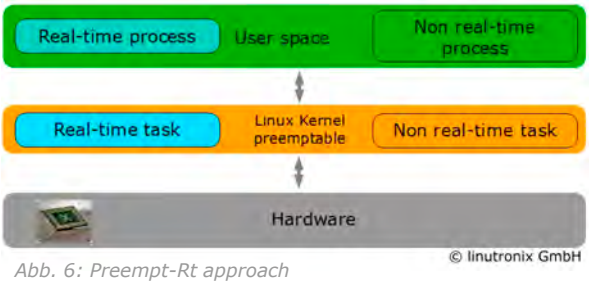

Abb. 6: Preempt-Rt approach

Der Ansatz nennt sich RT-Preempt Patch und ist, Stand September 2011, inzwischen zum größten Teil in das sog. Mainline Linux (www.kernel.org, Vanilla Kernel) integriert worden. Übrigens, bisher wurde noch nie der Versuch unternommen ein GPOS (general purpose operating system) nachträglich zu einem RTOS zu verwandeln

„Industrielle" Umsetzungen

Alle diese genannten Ansätze sind bereits als Produkte umgesetzt worden und sind bzw. waren in unterschiedlicher Häufigkeit im industriellen Einsatz. Nicht alle davon werden heute noch weiter entwickelt.

RTAI

Ein Vertreter des Thin Kernel oder auch Micro Kernel Ansatz ist RTAI, eine Open Source basierte Entwicklung des Mailander Polytechnikums unter Prof. Paolo Montegazza. RTAI betrachtet Linux als „idle" Task, also als diejenige Task mit der niedrigsten (Echtzeit-)Priorität.

Bei diesem Ansatz kontrolliert RTAI die Ressource „Interrupt" und verteilt diese entsprechend der zuvor gewählten Priorität und Zuständigkeit an die Echtzeit Tasks oder an Linux. Linux selbst ist um eine Komponente, den RTHAL (real-time hardware abstraction layer) erweitert worden. Diese Komponente entzieht Linux die Kontrolle über die Interrupts und leitet sie zum RTAI Thin Kernel um. Die Echtzeit Tasks kommunizieren untereinander mittels Mailboxen und Semaphoren. Die Kommunikation zwischen den Linux Anwendungen, also den Nicht Echtzeit Tasks, und den Echtzeit Tasks geschieht über shared memory Bereiche und FIFOs. RTAI ist auch für Multi-Core Prozessoren verfügbar.

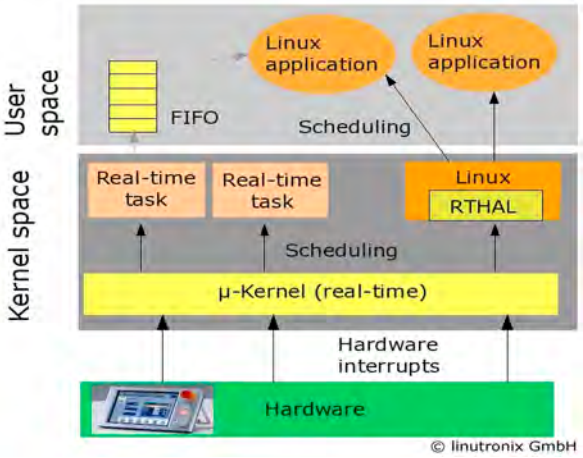

Abb. 7: Structure RTAI

Die Entwicklungsgeschwindigkeit von RTAI hat abgenommen. Das Hauptinteresse liegt auf der x86 Architektur, mit einem klaren Fokus auf einer Optimierung der Latency Zeiten. RTAI hat bei verschiedenen Aufgabenstellungen und bei früheren Untersuchungen immer wieder bewiesen, dass es für Echtzeit Aufgaben geeignet ist. Zu beachten ist bei der Beurteilung der Ergebnisse, dass in einigen Fällen RTAI den SMB (system management bus) abgeschaltet hat, um keine Ausreißer bei den Meßergebnissen zu erhalten. Ein derartiges Abschalten des SMB ist aber nicht unbedingt zu empfehlen, da es hierbei unter Umständen zu einem Totalverlust der Hardware (Hitzetod etc.) kommen kann. Allerdings zeigt RTAI mit diesem Ansatz bei Hardware, die von diesem SMB Verhalten betroffen ist, ein deutlich besseres (= kürzere Latenzen) Verhalten als die anderen Echtzeit-Ansätze.

RTAI bedingt durch seine Struktur eine aufwendige Entwicklung. Neben dem Linux API muss der Anwender auch noch die RTAI API beherrschen und seine Applikation(en) aufteilen in einen Echtzeit- und einen Nicht-Echtzeit Teil. Und dann muss er noch dafür sorgen, dass diese 2 Komponenten miteinander kommunizieren können, falls die Echtzeit Task nicht komplett abgeschlossen von der anderen Welt lebt. Und das dürfte wohl nie der Fall sein.

ADEOS / Xenomai

Ein Vertreter der Nano Kernel Technologie ist ADEOS bzw. das darauf aufbauende Produkt Xenomai. Xenomai hat eine wechselvolle Geschichte hinter sich. In 2001 wurde das Projekt aus der Taufe gehoben, in 2003 wurde mit RTAI/fusion der Versuch unternommen, mit RTAI zu mergen. In 2005 hat sich Xenomai wieder verselbständig und von RTAI gelöst. Xenomai basiert auf einem

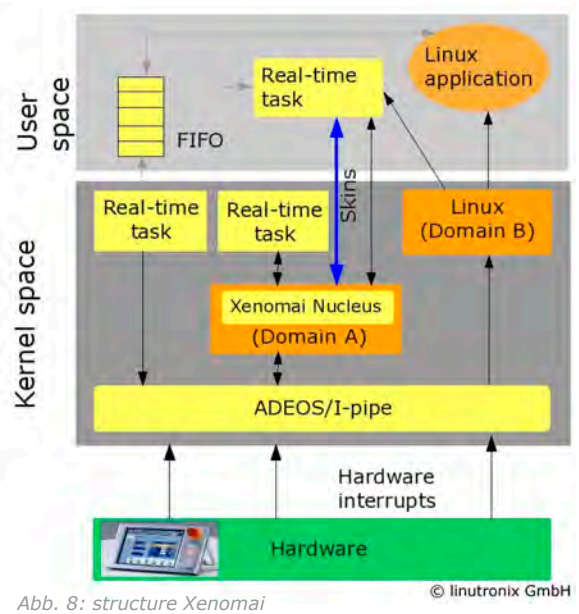

Abb. 8: structure Xenomai

abstraktem RTOS Core, der es unter anderem ermöglicht, jedes gewünschte Echtzeit API zu realisieren. Diese APIs, Skins genannt, können zum Beispiel einem API eines bereits existenten RTOS entsprechen. Dies ermöglicht Anwendern dieses RTOS einen einfachen Umstieg auf die Xenomai Lösung, da sich das API für den Anwendungsentwickler nicht ändert.

ADEOS selbst (also der Nano Kernel) ist als Linux Patch verfügbar und implementiert im Wesentlichen eine Interrupt Pipeline. Diese verteilt die ankommenden Interrupts nach einer Prioritätsliste. Die hierfür benötigte SW-Struktur wird auch „I-pipe" genannt. Prinzipiell sieht die Arbeitsweise von ADEOS/I-pipe so aus, dass zuerst Xenomai (der RTOS core) über einen Interrupt benachrichtigt wird. Wenn Xenomai den IRQ nicht bearbeitet (da nicht zugeteilt), wird der IRQ von ADEOS der nächsten Domain (in der Regel also dem Linux Kernel) übergeben.

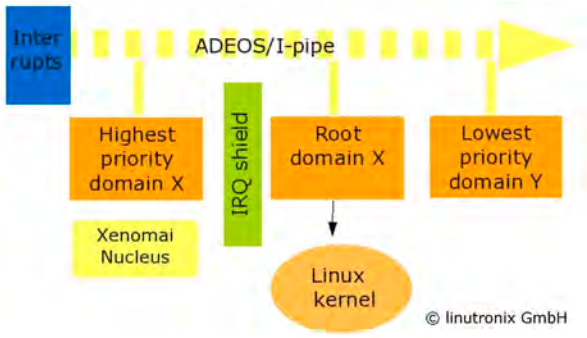

Abb. 9: ADEOS / I-pipe domain structure

Die Vorteile von Xenomai sind zum einen in der Vielzahl von vorhandenen APIs zu suchen. Im Moment sind folgende APIs unterstützt: Posix, RTDM, VxWorks, pSOS+, uITRON, VRTX und natürlich das eigene API. Und in der Anzahl der unterstützten CPU Architekturen. Hierzu zählen i386, x86_64(32bit), PPC, PPC64, IA64, Blackfin und auch viele ARM Prozessoren.

Die Entwicklung von Xenomai wird sich zukünftig stärker auf die Nutzung von Linux mit RT-Preempt Erweiterung konzentrieren. Die dadurch frei werdenden Ressourcen sollen für die Erstellung bzw. Weiterentwicklung der Skins Verwendung finden.

Linux mit Preempt-RT

Das ist wohl der zukunftsträchtigste Ansatz für Linux und Echtzeit. Der Ansatz wird von der Linux Entwicklergemeinschaft (Community) getragen und hat zahlreiche Verfechter und Freunde gefunden. Stand Linux 3.0 sind bereits gut 85% des benötigten Codes im Kernel integriert, die restlichen Teile, die heute noch als Patch eingespielt werden müssen, werden auch in den Kernel übernommen werden. Der Ansatz ist so einfach wie schwer – das gesamte Betriebssystem wird preemptiv, bekannte Probleme wie „priority inversion" werden innerhalb des Betriebssystems gelöst. Der Einsatz von High Resolution Timern ermöglicht eine Auflösung bis zu einer 1 Mikrosekunde, also fein genug für alle mit heutiger Hardware (general purpose CPUs) lösbaren Aufgaben. Der „big kernel lock" Einfluss musste reduziert bzw. eliminiert werden, was durch die Umstellung von spin locks in Mutexe ermöglicht wurde. Der Interrupt Code wurde für alle Architekturen vereinheitlicht, Interrupts wurden unterbrechbar („threaded interrupts")

Die wesentlichen Arbeiten an diesem Ansatz wurden von Ingo Molnar, Steven Rostedt (Red Hat) und von Thomas Gleixner (linutronix) durchgeführt. Heute unterstützt eine große Anzahl an Entwicklern die kontinuierliche Weiterentwicklung.

Das Ergebnis ist ein skalierbares Betriebssystem, einsetzbar auf allen von Linux unterstützten Hardware Plattformen (i386, x86_64, ppc, powerpc, sh, sh64, IA, ARM …) mit Echt-zeit Fähigkeit. Auch ist die Leistungsfähigkeit dieser Lösung beeindruckend. Selbst auf „kleinen" ARM

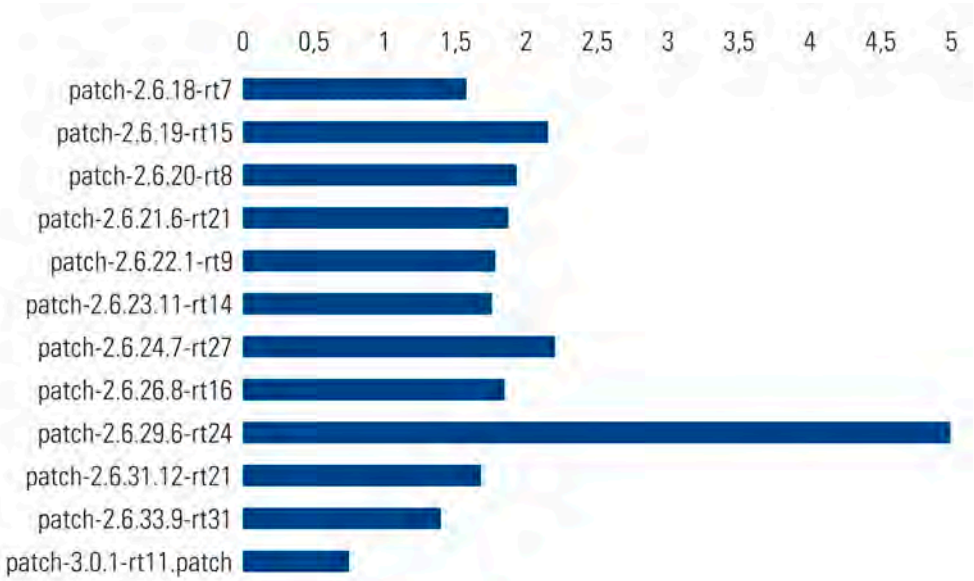

Abb. 10: Preempt-RT patch size 10/2006 - 08/2011 [in MByte]

Systemen, z. Bsp. mit einem ARM11, lassen sich Interrupt Latenzzeiten von max. 65 Mikrosekunden erreichen, mit Umschaltzeiten zwischen IRQ -> User IRQ Task -> ursprüngliche User Anwendung von maximal 135 Mikrosekunden Seit dem Beginn der Arbeiten am Linux Kernel und heute wurde ferner eine große Anzahl an Werkzeugen geschaffen, die den Einsatz von Linux in Echtzeit Anwendungen unterstützen. Zu nennen sind hier vor allem „cyclic test" und „ftrace"

Zusammenfassung

Der große Vorteil des Preempt-RT Ansatzes liegt in seiner Schlichtheit für den Benutzer. Das Programmiermodell bleibt einfach, es ist nur eine API, nämlich die von Linux, zu verstehen und zu benutzen. Der zweite große Vorteil für den Nutzer ist die Skalierbarkeit dieses Ansatzes. Eine einmal entwickelte Applikation lässt sich auf allen von Linux unterstützen Plattformen nutzen. Ohne dass man sich darum kümmern muss, ob es hierfür eine Portierung des Thin- oder Nano Kernel gibt. Ohne dass man sich Gedanken machen muss, ob die neueste Version von Linux auf dieser Hardware mit diesem aktuell hierfür erhältlichem zusätzlichen Thin Kernel einsetzbar ist. Der Nutzer wird frei von Zwängen, was die Hardware oder die Software anbelangt. Er kann nun auch bei Echtzeit Anwendungen die Freiheit von Linux nutzen.

Aus den genannten Gründen, kombiniert mit der technischen Leistungsfähigkeit, wird Linux mit dem RT-Preempt Patch kurz- bis mittelfristig alle anderen Ansätze verdrängen.

Weitere Informationen zum Thema Linux und Echtzeit mit RT-Preempt finden Sie u.a. auf der Homepage der OSADL (Open Source Automation Development Lab), www.osadl.org, oder auf unserer Homepage.

Haben wir Ihr Interesse geweckt? Wollen Sie mehr wissen?
Rufen Sie uns einfach an! Oder senden Sie uns eine Email.

linutronix GmbH
Auf dem Berg 3
88690 Uhldingen – Mühlhofen

Tel.: +49 8342 898 703
Fax: +49 8342 898 704
E-Mail: **info@linutronix.de**
www.linutronix.de