03 6. Februar 2007

Offizieller Medienpartner embeddedworld2007 Exhibition&Conference Nürnberg

4,90€ 9.- sfr.

www.elektroniknet.de

Elektronik

Fachzeitschrift für industrielle Anwender und Entwickler

Standard-Linux wird echtzeitfähig

Blick unter die Haube

Seit dem Linux-Kernel 2.6.18 vom August 2006 sind große Teile des so genannten Realtime-Preempt-Patches Bestandteil des Mainline-Linux-Kernels geworden. Darüber hinaus enthalten die Release-Kandidaten des Linux-Kernels 2.6.20 erstmals Module zur Virtualisierung auf Kernel-Ebene. Beide Neuerungen eröffnen interessante, bislang unbekannte Einsatzmöglichkeiten im Bereich der Automatisierungs- und Maschinenindustrie. Der Beitrag stellt die Funktionsprinzipien der neuen Kernel-Funktionen vor, deren Installation und Konfiguration.

Von Carsten Emde und Thomas Gleixner

Im Gegensatz zu den klassischen Echtzeit-Betriebssystemen wie OS-9, QNX oder VxWorks wurde bei der Entwicklung von Linux zunächst nicht auf das Echtzeit-Verhalten geachtet – im Vordergrund standen Datendurchsatz und Stabilität der vorwiegend als Server eingesetzten Linux-Systeme. Dies entsprach der Tradition klassischer Unix-Rechner. Erst durch den Bedarf, Linux auch in anderen Gebieten einzusetzen, entstand die Notwendigkeit, dem Linux-Kernel Echtzeit-Fähigkeit zu geben. Treibende Kräfte dafür waren unter anderem die Einsatzgebiete:

- Aufnahme und Wiedergabe von Audiosignalen,
- Embedded-Systeme in der Maschinenindustrie und in militärischen Projekten,
- zeitkritische Anwendungen bei Business-Software, z.B. zuverlässige Zeitstempel bei Datenbank-Transaktionen,
- Voice-over-IP und Streaming-Video für den Einsatz bei Telekommunikations-Providern.

Spezielle Nachfrage nach einem echtzeitfähigen Embedded-Linux kommt auch von Anwendern der genannten klassischen Echtzeit-Betriebssysteme, die vermehrt nach einer Open-Source-Alternative suchen. Noch im Jahre 2001 wurde in einer Markterhebung die unzureichende Echtzeit-Fähigkeit als wichtigster Hinderungsgrund genannt, Linux in Embedded-Systemen einzusetzen [1]. Allerdings waren bereits zu diesem Zeitpunkt zwei unterschiedliche Strategien erkennbar, Linux mit Echtzeit-Fähigkeit auszustatten: Erstens die Verwendung eines getrennten Echtzeit-Kernels, der den Linux-Kernel kontrolliert, dies ist der so genannte Dual-Kernel-Ansatz (z.B. RTAI), und zweitens die Bereitstellung von Patches, mit denen der Linux-Kernel selbst Echtzeit-Fähigkeit erhält. Diese Patches wurden in Gestalt von Realtime-Preempt-Patches unter der Regie von Ingo Molnar und Thomas Gleixner entwickelt. Nachdem sich Linus Torvalds lange gegen Echtzeit-Erweiterungen im Mainline-Kernel gewehrt hatte, kam im Sommer 2006 die lang erhoffte Kehrtwende [2], und wesentliche Teile der so genannten Realtime-Preempt-Patches sind in den Kernel 2.6.18 eingeflossen. Bis spätestens Ende 2007 sollen die übrigen Komponenten der Realtime-Preempt-Patches folgen. Damit ist Linux nun ganz offi-

ziell auf dem Weg, ein Echtzeit-Betriebssystem zu werden, und es hat einen großen Teil dieses Wegs inzwischen schon zurückgelegt.

Die wichtigsten Komponenten der Realtime-Preempt-Patches

Interrupt-Threading

Bisher wurden Interrupt-Service-Routinen vollständig im Kontext der Interrupt-Verarbeitung ausgeführt. Wenn der Zugriff auf Hardware und Datenstrukturen gegen Funktionen, die nicht im gleichen Kontext ausgeführt werden, serialisiert werden musste, wurden so genannte Spinlocks verwendet. Da Interrupt-Service-Routinen vergleichsweise lange Laufzeiten aufweisen können und – solange sie im Interrupt-Kontext ausgeführt werden – nicht zu unterbrechen sind, führte dies zu langen Latenzen, die mit einer Echtzeit-Fähigkeit nicht vereinbar waren. Die genannten Spinlocks ließen sich auch nicht einfach in unterbrechbare Spinlocks umwandeln, da dies ebenfalls von deren Ausführung im Interrupt-Kontext verhindert wurde. Das mit den Realtime-Preempt-Patches eingeführte Interrupt-Threading verändert nun das Interrupt-Handling dahingehend, dass nur die Quittierung des Interrupts im Interrupt-Kontext ausgeführt wird. Zur eigentlichen Bearbeitung des Interrupts wird ein diesem Interrupt zugeordneter Kernel-Thread gestartet. Diese Maßnahme reduziert die Bearbeitungszeit im Interrupt-Kontext auf ein Minimum, und die eigentliche, möglicherweise langwierige Interrupt-Service-Routine wird im Thread-Kontext ausgeführt. Dadurch bleibt die Interrupt-Service-Routine unterbrechbar, und auch die weiterhin erforderlichen Spinlocks können in unterbrechbare Spinlocks umgewandelt werden, weil diese nun nicht mehr im Interrupt-Kontext arbeiten. Die Interrupt-Service-Routine einschließlich ihrer Spinlocks kann nun durch den Scheduler gesteuert werden. Die Verwendung von Interrupt-Threads bietet auch im Nicht-Echzeit-Bereich Vorteile, da die bisherige Trennung von Funktionen in Interrupt- und Thread-Kontext aufgehoben werden kann und sich komplexe Locking-Probleme – speziell bei der Fehlerbehandlung – vermeiden lassen.

Um möglichst alle Prozessorarchitekturen auf Interrupt-Threading umzustellen und dessen Weiterentwicklung nicht durch hardwarespezifischen Code zu erschweren, musste der generische IRQ-Layer überarbeitet werden, denn er enthält die von der Prozessorarchitektur unabhängige Behandlung von Interrupts. Zu diesem Zweck wurde das bisher verwendete „All-in-One"-Interrupt-Konzept auf so genannte Flow-Handler umgestellt. Diese ermöglichen eine individuelle Behandlung und Optimierung der unterschiedlichen Interrupt-Typen wie pegel- oder flankengesteuerte Interrupts. Seit dieser Überarbeitung enthalten die Interrupt-Threads praktisch keinen hardware-abhängigen Code mehr. Die Überarbeitung des generischen IRQ-Layers führte außerdem zu einer Konsolidierung der Interrupt-Routinen für i386, x86_64, ARM und PowerPC und zur sauberen Integration von so genannten Message-Signaled-Interrupts (MSI).

High-Resolution Timers

Die Auflösung der Timer, die für Applikationen und den Kernel zur Verfügung stehen, war bisher an den periodischen Systempuls gebunden. Dieser hatte bis einschließlich Linux-Kernel 2.4 grundsätzlich eine Intervalldauer von 10 ms; inzwischen sind Intervalle von 10 ms, 4 ms und 1 ms konfigurierbar (entsprechend 100 Hz, 250 Hz und 1000 Hz). Alle Versuche, den bestehenden Timercode mit einer höheren Auflösung zu versehen, scheiterten daran, dass weite Teile des Betriebssystems auf einen Timertakt in der genannten Größenordnung angewiesen waren. Thomas Gleixner und Ingo Molnar beschritten daher einen neuen Weg und lösten die Timer, die für hochauflösende Funktionen relevant sind, aus dem bisherigen Code heraus und führten ein eigenes „hrtimer"-Subsystem ein [3]. Das Basissystem, das noch vom periodischen Puls getrieben wird, wurde bereits in die Kernelversion 2.6.16 übernommen. Und die Erweiterungen, die das Basissystem hochauflösend machen, sind seit langem als separater Patch verfügbar und Bestandteil der Realtime-Preempt-Patches. Inzwi-

OSADL – Automatisierer und Maschinenbauer treiben Linux voran

Die Linux-Erweiterungen Realtime-Preempt und kvm sowie viele andere Aspekte von Open-Source-Software sind besonders relevant für die Automatisierungsindustrie. Allerdings gab es bisher noch keine Organisation, die sich genau für dieses Einsatzgebiet einsetzt und die Interessen dieser Anwendergruppe vertritt. Aus diesem Grunde wurde das Open Source Automation Development Lab (OSADL) [9] gegründet und 2006 als Genossenschaft eingetragen. In der Präambel des Genossenschaftsvertrages heißt es: „Die Automatisierungsindustrie und ihre Zulieferunternehmen profitieren in besonderem Maße von quelloffenen Betriebssystemen wie z.B. Linux, da hierdurch lange Produktionszyklen, rasche Fehlerbeseitigung sowie die Unabhängigkeit von einzelnen Software-Herstellern gewährleistet werden. Allerdings benötigt diese Branche spezifische Erweiterungen des Betriebssystems wie zum Beispiel Echtzeit-Fähigkeit, es muss die Kompatibilität mit diesen Erweiterungen zertifiziert werden können, und es müssen standardisierte Software-Schnittstellen verfügbar sein. Die Entwicklung dieser Voraussetzungen ist das Ziel des Open Source Automation Development Lab OSADL." Entsprechend werden die Mitgliedsbeiträge dazu verwendet, um Kernel-Maintainer zu finanzieren, Testlabore zu betreiben, Arbeitsgruppen zu organisieren, Standards zu definieren, Konformitäts-Zertifikate zu erteilen und – nicht zuletzt – um Entwicklungsaufträge zu vergeben, damit für die Automatisierungsindustrie erforderliche Software-Komponenten wie spezielle Treiber bereitgestellt werden können. Bereits in den ersten Monaten seiner Geschäftstätigkeit hat das OSADL wesentliche, für die Echtzeit-Fähigkeit des Linux-Kernels relevante Software-Projekte angestoßen. Beispielsweise wurden Aufträge für die Anpassung der PowerPC- und ARM-Architekturkomponenten an die Realtime-Preempt-Patches vergeben, so dass diese jetzt verfügbar sind. Auf der Embedded World 2007 in Nürnberg (13. bis 15. Februar) sind das OSADL sowie viele OSADL-Mitglieder im Rahmen des „Embedded Linux Park" in Halle 11, Stand 306, vertreten. Unter anderem werden praxisbewährte Maschinen-Integrationen und Demo-Applikationen aller in diesem Artikel genannter Funktionen zu sehen sein.

schen ist auch das Feature „dynamic tick“ darin enthalten, das den periodischen Puls während der Idle-Periode abschaltet. Dies ist vor allem für batteriebetriebene Systeme relevant, wenn die CPU nur dann aus dem Idle-Sleep-State aufgeweckt wird, sobald tatsächlich Arbeit zu verrichten ist.

Priority Inheritance/ Realtime-Mutexe

Mit Prioritätsumkehr bezeichnet man das Phänomen, dass ein niedrig priorisierter Prozess eine nur einem einzigen Prozess gleichzeitig zur Verfügung stehende System-Ressource belegt und damit möglicherweise einen anderen höher priorisierten Prozess behindert, der diese Ressource ebenfalls benötigt. Die Priorität des höher priorisierten Prozesses wird dadurch gleichsam umgekrempelt und faktisch auf die Priorität des niedrig priorisierten Prozesses gesetzt. Dies kann die Priority-Inheritance verhindern; dabei „erbt“ die Ressource die Priorität des sie belegenden Prozesses und verhindert somit eine Prioritätsumkehr. Diese Lösung war allerdings bei den Kernel-Entwicklern lange Zeit umstritten und wurde von Linus Torvalds kategorisch abgelehnt („Friends don’t let friends use priority inheritance“). Dennoch wurde mehrmals versucht, Priority-Inheritance in den Linux-Kernel zu integrieren, allerdings scheiterte dies an der Komplexität der Patches und mangelnder Akzeptanz bei den Entwicklern der GNU C-Library. Ingo Molnar, Thomas Gleixner und die C-Library-Entwickler Ulrich Drepper und Jakub Jelinek fanden schließlich eine Lösung, Priority-Inheritance sauber in die bestehende Infrastruktur der Userspace-Mutexe (pthread_mutex) zu integrieren [4]. Mutex bedeutet *Mutual Exclusion*, zu deutsch: gegenseitiger Ausschluss. Mutexe verhindern, dass quasi-gleichzeitig ablaufende Threads gleichzeitig auf Daten zugreifen, und sind deshalb auch geeignet, den Zugriff auf Ressourcen zu regeln, die nur von einem Prozess oder Thread genutzt werden können. Diese Aufgabe übernehmen die nunmehr implementierten Realtime-Mutexe (rt_mutex) – ein neues Serialisier-Element im Linux-Kernel. Realtime-Mutexe sind ein Basis-Element für die vollständige Unterbrechbarkeit des Linux-Kernels.

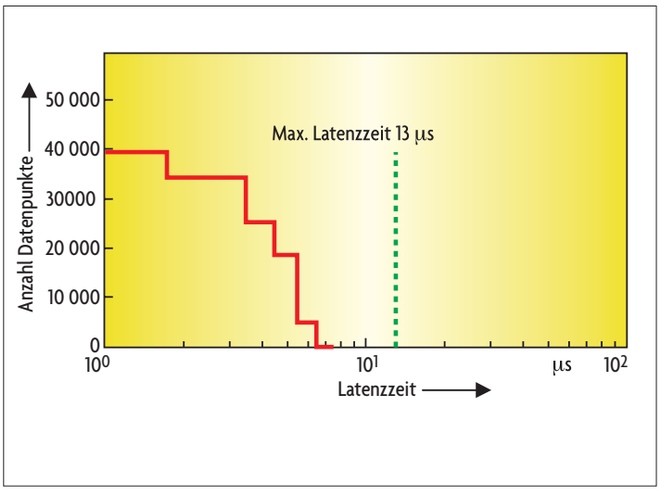

Bild 1. Mit den integrierten Diagnose-Funktionen der Realtime-Preempt-Patches lässt sich z.B. die maximale Latenzzeit messen – in diesem Fall 13 µs.

Vollständige Kernel-Unterbrechbarkeit

Der Linux-Kernel wurde in der Entwicklungsphase der Version 2.5 mit dem Preemption-Patch im Prinzip weitgehend unterbrechbar gemacht, so dass ein niedrig priorisierter Prozess während der Abarbeitung eines Systemaufrufs nicht mehr einen höher priorisierten Prozess blockieren kann. Allerdings blieben noch immer relativ lange Abschnitte übrig, in denen die Unterbrechbarkeit explizit abgeschaltet war. Die meisten dieser Abschnitte wurden durch die bereits zuvor erwähnten nicht unterbrechbaren Spinlocks serialisiert. Nach Einführung unterbrechbarer Spinlocks, die auf Realtime-Mutexen basieren, konnten auch diese Kernel-Abschnitte angepasst werden und sind nun unterbrechbar geworden. Anzahl und Dauer nicht-unterbrechbarer Kernel-Abschnitte wurden dadurch signifikant verringert. Darüber hinaus verfügen diese Mutexe über die erwähnte Priority-Inheritance, mit der sich eine Prioritätsumkehr verhindern lässt.

Diagnose-Tools

Die Realtime-Preempt-Patches enthalten eine Reihe von Diagnose-Tools, um das Verhalten des Systems zur Laufzeit zu analysieren. Dazu gehören u.a. die statistische Analyse von Timern, Tools zur Laufzeitanalyse verschachtelter Serialisierer (Aufdeckung von Deadlock-Situationen) und Tools zur Analyse von überlangen Latenzen. Diese Tools sind in erster Linie für Systementwickler gedacht und teilweise bereits in den Mainline-Kernel eingeflossen. Die konsequente Anwendung dieser Tools führte zur Beseitigung von etlichen, bisher nur schwer diagnostizierbaren Problemen im Mainline-Kernel. Im Laufe der Entwicklung wurden mehrere Bereiche des Kernels konsolidiert und mehr als 1000 Bugfixes in den Mainline-Kernel eingebracht. Das grundsätzliche Ziel war dabei immer, festgestellte Probleme ursächlich zu beheben und nicht nur schnelle „Workarounds“ bereitzustellen. Gerade für diese ursächliche Behebung ist die Verfügbarkeit guter Diagnose-Tools wichtige Voraussetzung. Nicht zuletzt sei auch darauf hingewiesen, dass viele der Bugfixes, die im Rahmen der Realtime-Preempt-Patches bereitgestellt wurden, auch der Qualität und Leistungsfähigkeit eines nicht für Realtime-Anwendungen konfigurierten Kernels zugute kommen.

Echtzeit-Funktionen konfigurieren

Wie alle anderen Eigenschaften des Linux-Kernels werden auch die Echtzeit-Eigenschaften in der Datei *.config* definiert. Typischerweise finden sich in einem entsprechend konfigurierten Echtzeit-Kernel die folgenden Einstellungen:

```
CONFIG_RT_MUTEXES=y
# CONFIG_PREEMPT_NONE is not
set
# CONFIG_PREEMPT_VOLUNTARY is
not set
# CONFIG_PREEMPT_DESKTOP is
not set
CONFIG_PREEMPT_RT=y
CONFIG_PREEMPT=y
CONFIG_PREEMPT_SOFTIRQS=y
CONFIG_PREEMPT_HARDIRQS=y
CONFIG_PREEMPT_BKL=y
CONFIG_PREEMPT_RCU=y
CONFIG_DEBUG_PREEMPT=y
# CONFIG_DEBUG_RT_MUTEXES is
not set
CONFIG_RT_MUTEX_TESTER=y
# CONFIG_CRITICAL_PREEMPT_TI-
MING is not set
CONFIG_WAKEUP_TIMING=y
CONFIG_WAKEUP_LATENCY_HIST=y
```

In Produktionssystemen sollte allerdings die Definition der Variablen CONFIG_DEBUG_PREEMPT auskommentiert werden, da sich dadurch die Echtzeit-Performance verschlechtern kann.

Eingebaute Diagnosefunktionen

Eine Reihe von Diagnostik-Tools erleichtert die Entwicklung der Realtime-Preempt-Patches. Diese lassen sich aber auch gezielt zur Analyse und zur Überwachung der Echtzeit-Eigenschaften eines individuellen Systems verwenden. Durch das Kommando

```
echo 0 >/proc/sys/kernel/
preempt_max_latency
```

wird die Trace-Funktion aktiviert. Danach liefert diese Variable die bis dahin gemessene höchste Scheduling-Latency (max. Latenzeit), zum Beispiel:

```
cat /proc/sys/kernel/
preempt_max_latency
13
```

Das entsprechende Histogramm mit einer Granularität von einer Mikrosekunde steht in der Datei */proc/sys/kernel/latency_hist* zur Verfügung. Auf diesen Daten beruht das exemplarische Histogramm in **Bild 1**.

Mit POSIX-Funktionen Prioritäten setzen

Für die Nutzung der Echtzeit-Fähigkeit musste kein neues API entwickelt werden, vielmehr wird auf vorhandene POSIX-Aufrufe zurückgegriffen. So wird z.B. die Priorität eines Prozesses mit der ab POSIX.1b und aktuell in POSIX.1-2001 definierten Funktion

```
#include <sched.h>
sched_setscheduler(pid_t pid,
int policy, const struct
sched_param *p)
```

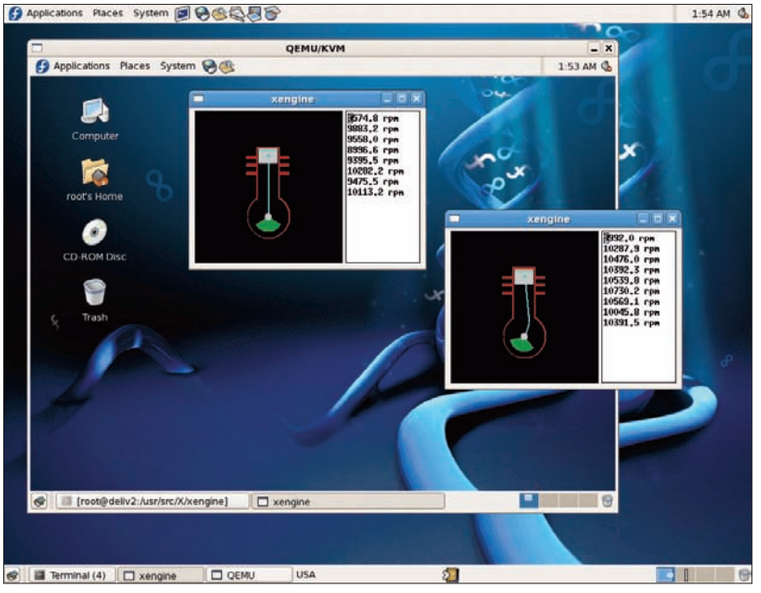

Bild 2. Infolge der Hardware-Unterstützung neuer Prozessoren von Intel (Vanderpool bzw. VMX-Technologie) und AMD (Pacifica-Technologie) für virtuelle Rechner können die Gast-Betriebssysteme ohne Modifikation installiert werden. Der Screenshoot zeigt Fedora Core 6 mit einem weiteren Fedora Core 6 in kvm-Virtualisierung.

eingestellt, wobei das Verhalten (Policy) des Schedulers folgende Werte annehmen kann:

- SCHED_OTHER – Standard-Policy des Linux-Kernels, für Echtzeit-Fähigkeit ungeeignet.
- SCHED_BATCH – Ähnlich wie SCHED_OTHER, aber etwas geringere Ausführwahrscheinlichkeit. Diese Policy ist speziell für Hintergrund-Prozesse mit hoher Prozessorlast, aber geringer Priorität vorgesehen (Linux-spezifisch), ebenfalls für Echtzeit-Fähigkeit ungeeignet (kein POSIX-Standard).
- SCHED_IDLE – Identisch zu SCHED_BATCH (ebenfalls kein POSIX-Standard)
- SCHED_FIFO – First-In-First-Out. Ein Prozess mit dieser Policy erhält grundsätzlich Vorrang vor allen anderen SCHED_OTHER-Prozessen und vor allen anderen SCHED_FIFO- und SCHED_RR-Prozessen mit geringerer Priorität, sobald dieser Prozess Rechenzeit benötigt. Wenn alle Prozesse, die über die höchste SCHED_FIFO-Priorität verfügen, kontinuierlich Rechenzeit beanspruchen, erhält kein anderer User-Prozess Rechenzeit, so dass das System auf User-Ebene komplett geblockt wird. Diese Policy ist für Echtzeit-Fähigkeit geeignet.
- SCHED_RR – Round-Robin. Ähnlich wie SCHED_FIFO, aber dem Prozess steht immer nur ein bestimmtes maximales Zeitfenster zur Verfügung, dessen Dauer folgende Funktion abfragt:

```
#include <sched.h>
int
sched_rr_get_interval(pid_t
pid, struct timespec *tp);
```

Wie bei SCHED_FIFO kann es bei kontinuierlichem Bedarf an Rechenzeit zu einer Blockade des Systems kommen. Auch diese Policy ist für Echtzeit-Fähigkeit geeignet. Die gewünschte Priorität wird im Element sched_priority der Struktur sched_param definiert. Der mögliche Wertebereich der Prioritäten lässt sich mit folgenden Funktionen für eine bestimmte Policy erfragen:

```
#include <sched.h>
int sched_get_priority_
max(int policy);
int sched_get_priority_min
(int policy);
```

Virtualisierung auf Kernel-Ebene

Die Virtualisierung einer kompletten Computer-Hardware hat sich in den letzten Jahren zu einem gebräuchlichen Verfahren entwickelt. Wohl am weitesten verbreitet ist die Software VMware, die seit 1998 von der VMware Inc. [5] entwickelt und vertrieben wird. Die Möglichkeit, auf einem einzigen Rechner mehrere unterschiedliche Betriebssysteme („Gast-Systeme") gleichzeitig laufen zu lassen, ist in zweifacher Hinsicht attraktiv: Erstens werden Software-Support und -Test erheblich erleichtert, sofern alle in Frage kommenden Betriebssystem-Varianten unmittelbar und in kompakter Form auf einem einzigen System zur Verfügung stehen, ohne dass es beispielsweise eines Neustarts bedarf. Zweitens erlaubt eine derartig simulierte Hardware-Umgebung einen konstanten und von der Verfügbarkeit bestimmter Hardware-Komponenten völlig unabhängigen Betrieb. Auf diese Weise lässt sich z.B. proprietäre Software, die für aktuelle Betriebssysteme nicht mehr verfügbar ist, praktisch uneingeschränkt weiternutzen, da die benötigte Betriebssystemversion, die auf realer Hardware schon längst nicht mehr lauffähig ist, auf der virtuellen Hardware immer noch funktioniert.

Beide Aspekte machen die Virtualisierung für die Maschinenbauindustrie interessant. Denn hier besteht häufig der Wunsch, sowohl die echtzeitfähige Maschinensteuerung als auch die Bedienoberfläche auf einer einzigen Hardware laufen zu lassen. Dabei handelt es sich um Funktionen, die – historisch bedingt – auf unterschiedlichen Betriebssystemen wie Linux und Windows implementiert wurden. Darüber hinaus bietet die Virtualisierung zum ersten Mal die Möglichkeit, eine lange Produktlebensdauer unabhängig von der Update-Politik des Herstellers zu gewährleisten. Selbstverständlich sollte zunächst immer geprüft werden, ob die vorhandene Visualisierungs-Software mit vertretbarem Aufwand portiert werden kann, damit diese nativ unter Linux lauffähig ist. In diesem Fall lassen sich Maschinensteuerung und Bedienoberfläche noch einfacher auf einem einzigen System betreiben.

Bild 3. Friedlich unter einem Dach – Virtualisierung macht's möglich. Fedora Core 6 mit Windows XP in kvm-Virtualisierung.

Die Umschaltung zwischen den unterschiedlichen Betriebssystemen erfolgt durch eine neuartige Funktionsebene, den sogenannten Virtual-Machine-Monitor (VMM). Da eine solche Funktion in der Hardware nicht vorgesehen war, mussten die ersten Virtualisierungs-Systeme vollständig innerhalb der Software realisiert werden. Die Herausforderung war dabei, dass eine zusätzliche Privilegierungsebene eingeführt werden musste, damit der VMM die Kontrolle über die Gast-Systeme behielt und die Gast-Systeme nicht auf die Ressourcen des VMM zugreifen konnten. Auf der anderen Seite musste das Gast-System bestimmte Privilegien behalten, um dessen Funktion nicht zu beeinträchtigen. Dies war nur möglich durch Änderungen an gewissen Instruktionen des Gast-Systems. Dadurch war die Entwicklung eines geeigneten VMM relativ schwierig, und es war von vornherein absehbar, dass eine derartige Technik nicht in das Mainline-Linux übernommen werden würde. Dies änderte sich rasch, als neue Prozessoren mit Hardware-Unterstützung für die Virtualisierung erschienen. Unter dem Code-Namen „Vanderpool-Technologie" (VT) wurde eine solche Erweiterung bei Intel-Prozessoren geführt, wobei die Abkürzung VT später als Virtualisierungs-Technologie übersetzt und kürzlich durch die Abkürzung „vmx" abgelöst wurde [6]. AMD hat inzwischen ein vergleichbares Konzept unter dem Namen „Pacifica" vorgestellt. Die Hardware-Unterstützung für die Virtualisierung hat dazu geführt, dass ein vergleichsweise kleines Linux-Kernel-Modul hergestellt werden konnte, das die Virtualisierung auf Kernel-Ebene bewerkstelligt. Ein solches Modul hat Avi Kivity entwickelt und *kvm* genannt [7]; es wurde inzwischen – nur wenige Monate nach dem ersten Release – mit Kernel-Version 2.6.20 in das Mainline-Linux aufgenommen. Die Laufzeitumgebung des kvm wird zurzeit noch mit einer modifizierten Version des Open-Source-Emulators *qemu* realisiert [8]. Später ist dies nicht mehr erforderlich. **Bild 2** und **Bild 3** zeigen jeweils eine Bildschirmansicht eines Fedora-Core-6-Systems mit einem weiteren Fedora-Core-6-System oder mit einem Windows-XP-System als Gast. Die Gast-Systeme mussten dafür nicht im geringsten modifiziert werden.

Ein wesentlicher Vorteil der Virtualisierung auf Kernel-Ebene ergibt sich aus der Tatsache, dass das gesamte Gast-System auf der Host-Plattform praktisch wie ein einzelner Prozess erscheint, der sich mit den üblichen Tools wie *nice, top, kill* usw. verwalten und inspizieren lässt. Ein spezieller *Super-*

visor oder andere Management-Software sind nicht erforderlich. Installation und Konfiguration sind ab Kernel 2.6.20 unkompliziert. Die folgenden Zeilen konfigurieren kvm-Support für Intel- und AMD-Prozessoren:

```
CONFIG_KVM=m
CONFIG_KVM_INTEL=m
CONFIG_KVM_AMD=m
```

Und mit einfachen Kommando wird das entsprechende Modul geladen:

```
modprobe kvm-intel
```
oder
```
modprobe kvm-amd
```

Sollte ein System nicht über einen geeigneten Prozessor verfügen, wird die Log-Meldung *kernel: kvm: no hardware support* ausgegeben. Eine für kvm geeignete Version des Emulators qemu sowie die Quellprogramme des kvm-Moduls für andere Versionen des Linux-Kernels können von der URL http://sourceforge.net/projects/kvm/ heruntergeladen werden.

Echtzeitfähiges Host-System

Da es sich beim kvm-Modul im Vergleich zu den anderen Verfahren um ein relativ kompaktes Modul handelt, wird davon ausgegangen, dass sich – nach entsprechender Entwicklungsarbeit – die Echtzeit-Fähigkeit des Host-Systems auch bei laufendem Gast-System erhalten lässt. Das Gast-System wird aber wohl kaum jemals Echtzeit-Fähigkeit erlangen. Aber genau diese Kombination eines echtzeitfähigen Host-Systems mit einem nicht echtzeitfähigen Gast-System wird von der Maschinenbauindustrie mit großem Interesse erwartet. Denn dadurch lassen sich die Maschinensteuerung einerseits und die grafische Bedienoberfläche andererseits in idealer Weise auf einem gemeinsamen System kombinieren. Entwicklungsarbeit, Kosten und Reparaturanfälligkeit werden sich so aller Voraussicht nach senken und zugleich die Zuverlässigkeit des Gesamtsystems erhöhen lassen. *jk*

Carsten Emde
hat sich bereits seit den Anfängen der Mikrocontroller mit Embedded-Systemen beschäftigt und für Forschungszwecke genutzte echtzeitfähige Systeme, zunächst unter OS-9, entwickelt und programmiert. Inzwischen heißt sein bevorzugtes Betriebssystem Linux und bestimmt den größten Teil seiner beruflichen Tätigkeit als Software-Entwickler, System-Integrator und Trainer. Seit Gründung des Open Source Automation Development Lab (OSADL) eG ist er dessen Geschäftsführer.
C.Emde@osadl.org

Thomas Gleixner
blickt auf eine 20-jährige Erfahrung mit industriell eingesetzten Embedded-Systemen zurück. In den letzten beiden Jahren hat er sich als Linux-Kernel-Entwickler zusammen mit Ingo Molnar an dessen Realtime-Preemption-Projekt hervorgetan; unter anderem hat er maßgeblich zur Entwicklung des High-Resolution-Timer-Subsystems beigetragen. 2001 gründete er seine Firma Linutronix; innerhalb des Open Source Automation Development Lab (OSADL) eG nimmt er die Funktion des Kernel-Maintainers und Chef-Entwicklers wahr.
Th.Gleixner@osadl.org

Weiterführende Informationen

[1] *Lanfear, C.; Balacco, S.:* Linux's future in the embedded systems market. Venture Development Corporation, April 2002, http://linuxdevices.com/articles/AT6328992055.html

[2] Kernel Summit 2006: Realtime, http://lwn.net/Articles/191782/

[3] High resolution timer design notes, http://rt.wiki.kernel.org/index.php/High_resolution_timer_design_notes

[4] Priority inheritance in the kernel, http://lwn.net/Articles/178253/

[5] VMware Inc., www.vmware.com/

[6] Intel Virtualization Technology, www.intel.com/technology/itj/2006/v10i3/

[7] KVM: Kernel-based Virtual Machine for Linux, http://kvm.sourceforge.net/

[8] QEMU Open Source Processor Emulator, http://fabrice.bellard.free.fr/qemu/

[9] Open Source Automation Development Lab (OSADL) eG, www.osadl.org

Kontaktinformation:
Open Source Automation Development Lab (OSADL) eG
Homagstr. 3-5
72296 Schopfloch
Tel +49(7443)13-3073
Fax +49(7443)13-8-3073
http://www.osadl.org
info@osadl.org