

## Prime Time

Unter dem Namen I-Hawk verkauft die Firma Concurrent Linux-Komplettsysteme, die nicht nur leistungs-, sondern auch echtzeitfähig sein sollen. Warum das dahinter stehende Shielded-CPU-Konzept zu überzeugen weiß, berichtet dieser Test. Jürgen Quade

Das **Echtzeitverhalten** eines Betriebssystems lässt sich an zwei Parametern festmachen: an der Interrupt- und der Task-Latenzzeit. Erstere ist die Zeitspanne zwischen dem Auftreten (Generieren) des Interrupts und dem Starten der Interrupt-Serviceroutine. Die Task-Latenzzeit umfasst die Zeitspanne vom Auftreten des Interrupts bis zum Start des zugehörigen Rechenprozesses. Latenzen werden gemeinhin als Worst-Case-Zeiten angegeben. Durchschnittswerte wären für so genannte harte Echtzeitsysteme nämlich vollkommen irrelevant (siehe **Kasten „Weiche und harte Echtzeit“**).

Dass Linux keine Garantie für Latenzzeiten übernimmt, hat sich herumgesprochen: ein Standard-Linux ist nicht echtzeitfähig. Das ist bedauerlich, denn es hat viele Vorteile, ein vom Schreibtisch bekanntes Betriebssystem auch für Echtzeitaufgaben einzusetzen. Das bekanntlich breit eingesetzte Linux unterstützt vielfältige Hardware. Und dass die Einarbeitung für den Entwickler wegfällt, geht ebenfalls klar auf die Haben-Seite.

Einen ersten Schritt in Richtung harte Echtzeit gehen RT-Linux und RTAI ([1]

### I-Hawk 862 Tower

**Komponenten:** 2 Xeon-HT-CPU's 2,8 GHz mit je 1 MByte L2-Cache, 2 PCIe-, 4 PCI-Slots, 1 GByte RAM, 2 SCSI-Platten à 73 GByte, PCIe-(x16)-Grafikkarte mit Nvidia Quadro (64 MByte)

**Schnittstellen:** 1x Gigabit-Ethernet, 2x seriell, 1x parallel, 8x USB, 2x Firewire

**Spezialhardware:** RCIM (Realtime Clock & Interrupt Module)

**Betriebssystem:** Redhawk Linux 2.2

**Preis:** rund 7100 Euro



bis [3]). Zwar machen sie Linux nicht im engeren Sinn echtzeitfähig, fügen jedoch dem Linux- einen Echtzeitkernel hinzu. Der Linux-Kernel ist die Idle-Task des Echtzeitkernels. Beide tauschen über Fifos oder gemeinsame Speicherbereiche Daten aus und harmonisieren ihr Verhalten auf einer CPU.

Das Einhalten von Interrupt- oder Task-Latenzen bleibt allerdings Sache der Prozesse auf dem Echtzeitkernel. Vorhandene Linux-Tasks werden dadurch nicht echtzeitfähig (**Abbildung 1a**). Ein Nachteil ist, dass sich der Entwickler von Echtzeitapplikationen in ein zweites Betriebssystem mit neuen Schnittstellen einarbeiten muss.

### Power satt

Die Firma Concurrent Computer Systems [4] möchte mit ihrem kommerziellen System I-Hawk 860 ein wirkliches Echtzeit-Linux anbieten. Das gibt es allerdings nur als komplettes System bestehend aus Hard- und Software zu kaufen. Gedacht ist es für komplexe Echt-

zeitanwendungen, sprich für rechenintensive Aufgaben wie die Steuerung eines Man-in-the-loop-Simulators (zum Beispiel eines Flugsimulators) oder zum Erfassen von Raketen- oder Jet-Triebwerksdaten.

### Doppelherz ist Pflicht

Mindestanforderungen sind ein zweiter Prozessor und eine Timerkarte (RCIM-Board). Ausstattung und Preis des Testgeräts im Labor des Linux-Magazins listet der **Kasten „I-Hawk 862 Tower“**, **Abbildung 2** zeigt es. Wer noch mehr Power braucht, dem liefert Concurrent eine Acht-Prozessor-Maschine, wahlweise mit Intel Xeons oder AMD-Opteron-Prozessoren. Ein Einstiegsmodell kostet knapp 5000 Euro, die Acht-Prozessor-Intel-Maschine mit 32 GByte RAM und 730 GByte Plattenplatz belastet das Konto mit 109.500 Euro.

An Software legt die im Markt der Echtzeitsysteme seit vielen Jahren bekannte Firma das modifizierte Linux Redhawk 2.2 sowie Tools zum Debugging und

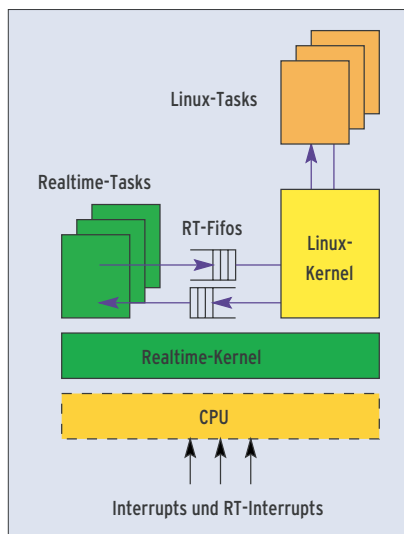


Abbildung 1a: RT-Linux ist konzeptionell ein extra Betriebssystemkern, auf dem Linux als Idle-Task läuft.

zum Betrieb der Echtzeitanwendungen bei. Redhawk Linux kommt auch als Quellcode ins Haus und steht mit Ausnahme des Frequency Based Schedulers unter der GPL. Die anderen, zum Betrieb nicht unbedingt notwendigen Applikationen sind nicht quelloffen.

Redhawk 2.2 beruht auf einem Red-Hat-Linux-Kernel 2.6.6, den Concurrent umfangreich um diverse Echtzeiteigenschaften und Debugging-Möglichkeiten erweitert hat. Als Stichworte seien hier Prioritätsvererbung, Spinlocks für Applikationen (!) oder die Posix Realtime Extensions genannt (siehe **Kasten „Redhawk's Echtzeiteigenschaften“**). Von zentraler Bedeutung sind das so ge-



Abbildung 2: Die Hardware des getesteten I-Hawk-Systems mit zwei Xeon-Prozessoren kommt von Dell, ergänzt um eine Timerkarte.

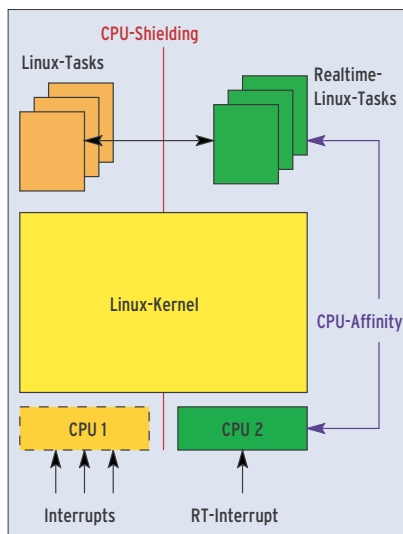


Abbildung 1b: Bei Redhawk Linux laufen stattdessen Echtzeit-Linux-Prozesse auf einer separaten CPU.

nannte CPU-Shielding und der Frequency Based Scheduler.

## Einen Riegel verschieben

Unter CPU-Shielding – das ist der Clou des Systems – versteht Concurrent das exklusive Reservieren eines Prozessors für eine oder auch mehrere Aufgaben (**Abbildung 1b**). Ordnet der Entwickler einer derart abgeriegelten CPU eine Interruptquelle zu, stellt er sicher, dass der dort eintreffende Interrupt sofort bearbeitet werden kann und auch unverzüglich bearbeitet wird.

Auch Rechenprozesse lassen sich dieser CPU zuordnen; damit wird die Last des abgeriegelten Prozessors berechenbar, das Echtzeitverhalten ist ga-

rantiert [6], [7]. Das Abriegeln funktioniert mit dem aktuellen Redhawk noch nicht klaglos: Einzelne so genannte interne Interrupts bewirken, dass trotz CPU-Shielding Latenzen auftreten.

Unberechenbar bleiben auch die Systemaufrufe der Rechenprozesse auf dem abgeriegelten Prozessor. Die Systemcalls nutzen im Kernel eventuell Datenstrukturen, auf die gerade andere, nicht echtzeitfähige Prozesse (und Prozessoren) zugreifen. Da Semaphore oder Spinlocks den sequenziellen (nicht den parallelen) Zugriff sichern, muss der Echtzeitprozess trotz seiner Priorität warten. Wie lange die Echtzeittask deswegen blockiert, ist unbekannt. Entwickler sollten daher in ihren Realtime-Applikations-Systemcalls möglichst meiden.

Concurrent hat nach eigener Angabe auch den Kernelcode nach kritischen Abschnitten durchforstet und – wenn möglich und bekannt – durch das Einspielen von Patches zeitlich optimiert. Gleichwohl warnt die englischsprachige Dokumentation davor, das Programm Hdparm zeitgleich zu Echtzeitprogrammen zu starten, zwischen den virtuellen Konsolen hin und her zu wechseln oder Kernelmodule zu entladen. Eigene Treiber seien dagegen kein Problem, doch möge der Entwickler selbst für die Kürze kritischer Abschnitte sorgen.

## Präzise Zuteilung

Die zweite wichtige Kernelerweiterung etabliert den Frequency Based Scheduler. Der FBS sorgt auf Wunsch dafür, dass eine Echtzeittask im Wortsinne rechtzeitig startet, also zu einem vom

### Redhawk's Echtzeiteigenschaften

**Prioritätsvererbung:** Diese Technologie verhindert bei einem Ressourcenkonflikt unnötiges Blockieren hochpriorer Rechenprozesse. Andernfalls könnte bei einer als Prioritätsinversion bezeichneten Situation eine mittelpriore Task eine hochpriore verzögern, selbst wenn beide nicht um eine gemeinsame Ressource konkurrieren.

**Spinlocks:** Die bisher den Kernelprogrammierern vorbehaltenen Spinlocks schützen kritische Abschnitte. Anders als Semaphore realisieren sie aktives Warten (Busy-Loop) und sind damit nur auf Mehrprozessormaschinen einsetzbar. Durch das Einsparen des Context-Switch sind sie bei kurzen kritischen Abschnitten effizienter.

**Posix Realtime Extensions:** Ein Standard, der Programmierinterfaces zu echtzeitfähigen Systemfunktionen eines Betriebssystems definiert. Hierzu gehören das Realtime Scheduling, das Memory Locking – es verhindert das Auslagern von Speicherseiten –, Kommunikationsmechanismen (Message Queues) und hochauflösende Timer.

Anwender zu definierendem Zeitpunkt. Dieses Feature macht insbesondere sehr genaue periodische Rechenprozesse möglich. Seine Zeitangaben bezieht der FBS über die hochauflösenden Realtime-Clocks des RCIM-Board.

## Werkzeugkiste

Fünf Tools sollen beim Entwickeln und Optimieren von Echtzeit-Applikationen helfen: Nightsim ist ein grafisches Interface für den FBS, das den Entwickler zyklischer Applikationen unterstützt (**Abbildung 4**). Mit Nightprobe lassen sich Programmdateien lesen und auch ändern. Nighttrace erfasst Kernel- und Applikationsereignisse Mikrosekunden-genau. Mit Nighttune ändert man CPU-Affinitäten, Shielding-Attribute und Scheduling-Parameter. Nightview schließlich ist ein Source-Level-Debugger.

Allerdings entspricht das Look&Feel dieser Applikationen nicht dem, was verwöhnte Linux-Benutzer von KDE- oder Gnome-Oberflächen her kennen. Hinzu kommt, dass das Wohlwollen eines License-Key-Servers erkämpft sein will. Wie fast immer bei solchen Schlüsselverwahranstalten war, nach Änderung des Hostnamens, auch bei der Testmaschine einige Handarbeit notwendig, bis die Toolchain anlieft.

## Die Stoppuhr klickt

Auf einer Zwei-Prozessor-Maschine garantiert Concurrent Computer eine Task-Latenzzeit von nur 30 Mikrosekunden. Eine Aussage zur Interrupt-Latenz gibt die Firma zwar nicht, da aber die Task-

Latenz der kritischere Wert ist, ist das zu verschmerzen. Ein Test zeigt, dass Concurrent mächtig am Zeitmanagement des Linux-Kernels gedreht hat: Ein Programm [5] maß überschlägig die Genauigkeit der Systemfunktion »nanosleep()« aus. Nach dem Aufruf der Funktion legen sich Rechenprozesse für einen einstellbaren Zeitraum schlafen.

Um die tatsächliche Schlafzeit zu bestimmen, nimmt das Programm unmittelbar vor und nach dem Aufruf von »nanosleep()« einen Zeitstempel und berechnet die Differenz. Den zeitlichen Fehler, der sich durch das Erfassen des Zeitstempels ergibt, ist hier zu vernachlässigen. Müsste sich dieser simple Benchmark an einem Standard-Linux versuchen, würde er gut und gerne Ungenauigkeiten von rund 2000 Mikrosekunden messen, selbst dann, wenn das System keine anderen Aufgaben bewältigen muss, also ohne Last fährt.

**Tabelle 1** zeigt, dass Redhawk-Linux das Testprogramm mit Abriegelung einer CPU (Shielded-Zeile) maximal 25 Mikrosekunden zu viel warten ließ – und das, auch wenn das System richtig unter Last stand. Ohne Abriegelung (Unshielded) ist das Zeitverhalten von Redhawk gegenüber einem Wald-und-Wiesen-Linux ebenfalls spürbar besser: Im Vergleich zu den 2000 Mikrosekunden verspätet sich das Echtzeit-Linux nur um maximal 20 Mikrosekunden. Es handelt sich stets um Worst-Case-Werte. So ist auch das Ergebnis von 7648 Mikrosekun-

den bei einem Test mit nicht abgeriegeltem Prozessor zu sehen.

Ein zweiter Benchmark bestätigt den positiven Eindruck. Ein über das RCIM-Board generierter periodischer Interrupt weckt einen Rechenprozess auf. Der wiederum misst die aktuelle Zeit, berechnet die Differenz zum letzten Weckzeitpunkt und legt sich wieder schlafen. Theoretisch müsste die Differenz zwischen dem zweimaligen Aufwecken genau der eingestellten Timerperiode entsprechen. Die Abweichung davon entspricht grob der Task-Latenzzeit.

## Abgeriegelt liegt vorn

Auch dieser Test lief unter mehreren Bedingungen ab: ohne Echtzeitpriorität, mit Echtzeitpriorität, aber ohne abgeriegelte CPU, und schließlich mit Echtzeitpriorität und abgeriegelter CPU, und das Ganze jeweils mit und ohne Last. **Tabelle 2** zeigt das Ergebnis: Die geringsten Latenzen (knapp 20 Mikrosekunden) treten auf, wenn das Testprogramm auf dem abgeriegelten Prozessor läuft. Im Unshielded- und im Normal-Betriebsfall sind die Latenzzeiten um Größenordnungen höher.

Dass trotz abgeriegelten Prozessors eine starke Abweichung bei den Fällen ohne Last und mit Last auftritt (rund 9 Mikrosekunden zu 19) liegt daran, dass das

**Tabelle 1: Abweichungen bei der Systemfunktion »nanosleep()«**

| Messung    | ohne Last | mit Last |                       |
|------------|-----------|----------|-----------------------|
| Unshielded | 20 µs     | 7648 µs  | mit Echtzeitpriorität |
| Shielded   | 19 µs     | 25 µs    | mit Echtzeitpriorität |

### Weiche und harte Echtzeit

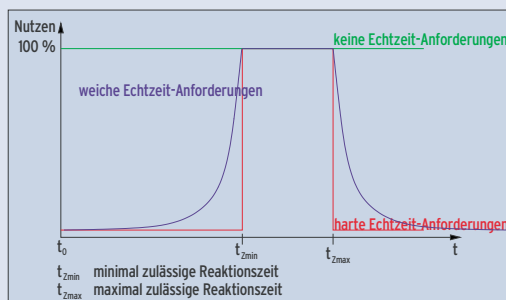
Erfüllt ein Echtzeitsystem die Anforderungen nicht, hat das je nach Einsatzbereich unterschiedlich dramatische Folgen: Im schlimmsten Fall sind Menschenleben bedroht (bei einer Flugzeugsteuerung beispielsweise), im weniger terminalen Fall ärgert sich ein Filmfreund vielleicht über einen Ruckler beim Abspielen einer DVD. Die beim Überschreiten der vorgegebene Zeitschranken (Deadlines) auftretenden Folgen lassen sich entweder in Form von entstehenden Kosten oder in Form des erzielbaren Nutzens ausdrücken.

**Abbildung 3** zeigt anhand der Nutzenfunktion (Benefit Function) den Unterschied zwischen weicher (Soft Realtime), harter und keiner

Echtzeit. Gibt es keinerlei zeitliche Anforderungen, ist der Nutzen unabhängig vom Bearbeitungszeitpunkt immer 100 Prozent. Von harten Echtzeitanforderungen sprechen die Entwickler, wenn bei einer Deadlineverletzung kein (null) Nutzen feststellbar ist. Bei weicher Echtzeit ist auch dann ein, wenn auch

**Abbildung 3: Benefit-Funktion:** Beim harten Echtzeitbetrieb muss der Computer zwischen  $t_{zmin}$  und  $t_{zmax}$  reagieren. Bei weicher Echtzeit ist auch eine verfrühte oder verspätete Reaktion tolerabel.

geringer Nutzen vorhanden, wenn mal eine Deadline verletzt wurde. Spätestens jetzt ist klar: Ein Echtzeitsystem muss nicht schnell sein, nur schnell genug.



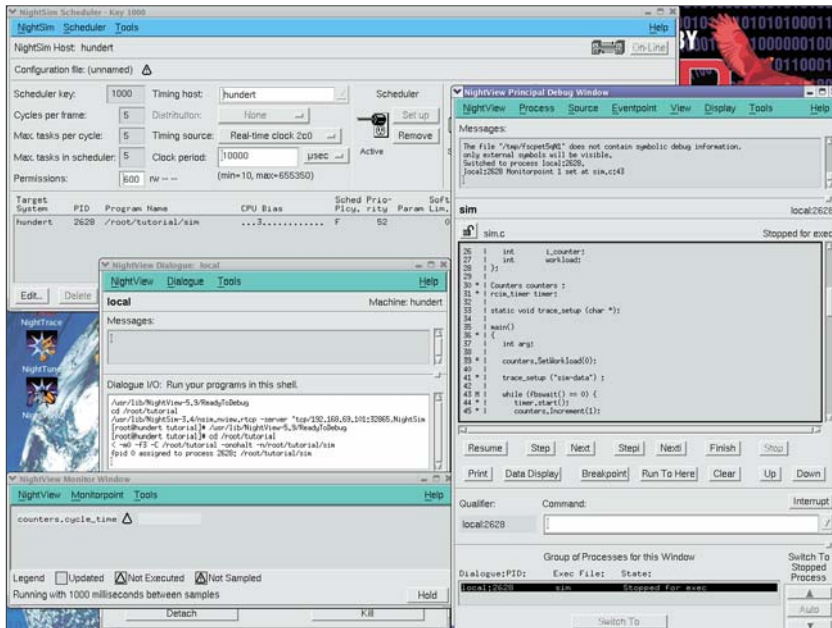


Abbildung 4: Fünf Werkzeuge helfen beim Entwickeln und Optimieren der Echtzeitapplikationen. Neben dem grafischen Interface NightSim zum SBS ist der grafische Debugger Nightview zu sehen.

Abriegeln eines Prozessors nicht lückenlos gelingt. Die nächste, für Juni 2005 geplante Redhawk-Version 2.3 soll – so Concurrent – das Auftreten interner Interrupts weiter reduzieren. Abhängig von der Hardware garantierte dies noch geringere Latenzzeiten.

## Fazit

Deterministisches Verhalten mit ganz normalen Linux-Prozessen realisieren ist nicht trivial. Die Idee, Echtzeit nicht durch einen integrierten Echtzeitkernel, sondern durch einen weiteren Prozessor zu garantieren, ist eingängig. Redhawk Linux wird durch die Shielded CPU zwar nicht umfassend echtzeitfähig, wohl aber laufen Echtzeitapplikationen auf den vorhandenen Linux-APIs harmonischer und – wichtiger noch – halten Schritt. Der Applikationsprogrammierer muss nur beim Aufruf der Systemcalls Vorsicht walten lassen.

Das Abriegeln von Prozessoren, das Setzen einer Interruptaffinität oder das

Starten einer Applikation mit Echtzeitpriorität erfolgt ohne Programmierarbeit mit einfach zu handhabenden Kommandos. Die mitgelieferte Dokumentation ist ausreichend, liegt allerdings nur in englischer Sprache vor. An der Hardware gibt es nichts auszusetzen. (jk) ■

## Infos

- [1] RT-Linux: [<http://www.rtlinuxfree.com>]
- [2] RTAI: [<http://www.rtai.org>]
- [3] Bernhard Kuhn, „Echtzeitbrüller – RT-Linux und RTAI“: Linux-Magazin 02/00, S. 52
- [4] Concurrent Computer: [<http://www.concurrent.de/start.html>]
- [5] Listing zum Nanosleep-Test: [<http://www.linuxmagazin.de/Service/Listings/2005/06/Redhawk>]
- [6] Steve Brosky, „Shielded CPUs: Real-Time Performance in Standard Linux“: Linux Journal, May 2004, online [<http://www.linuxjournal.com/article/6900>]
- [7] Brosky, Rotolo, „Shielded Processors: Garantieing Sub-millisekund Response in Standard Linux“: [<http://www.ccur.com/isdocs/wp-shielded-cpu.pdf>]

## Der Autor

Jürgen Quade, Professor an der Hochschule Niederrhein und Koautor der im Linux-Magazin laufenden „Kern-Technik“-Serie, hat sich bemüht aus dem Testbericht keine Vorlesung über Echtzeitsysteme werden zu lassen.



**Concurrent  
Computer GmbH**

Lena-Christ-Str. 46

Martinsried

82152 Planegg b. München

Tel.: + 49 (89) 85603-0

Fax: + 49 (89) 85603-150

[info@ccur.de](mailto:info@ccur.de)

[www.concurrent.de](http://www.concurrent.de)

Tabelle 2: Task-Latenzzeit gemessen mit dem RCIM-Timer

| Messung    | ohne Last     | mit Last       |
|------------|---------------|----------------|
| Normal     | 16885 $\mu$ s | 509002 $\mu$ s |
| Unshielded | 153,5 $\mu$ s | 29686 $\mu$ s  |
| Shielded   | 8,6 $\mu$ s   | 19,4 $\mu$ s   |