Harte Echtzeit mit Linux durch Preempt-RT zum Messen und Steuern

Zeitnah

Um Daten sicher innerhalb kurzer Zeiten einzulesen, zu verarbeiten und auszugeben, gibt es diverse Echtzeit-Betriebssysteme, aber in den meisten Fällen muss man nicht lange nach einem geeigneten suchen, sondern kann einfach ein Linux mit Preempt-RT-Patch nehmen. *Rolf Freitag*

Wenn Software Maschinen steuern oder die Zeit zwischen zwei Ereignissen aufzeichnen soll, muss sie stets zuverlässig innerhalb spezifizierter Grenzen reagieren, und dafür gibt es Echtzeit-Betriebssysteme. Diese sorgen für die so genannte harte Echtzeit. Der Begriff bezieht sich nicht primär auf Schnelligkeit der Reaktion des Systems, sondern auf die Vorhersagbarkeit der Reaktion.

Die Obergrenzen können also auch bei mehrere Sekunden liegen, in der Praxis aber meist innerhalb von wenigen Millisekunde oder weniger. So ist es typisch, dass Geräte an einem CAN-Bus eine empfangene Nachricht innerhalb von 200 Mikrosekunden zu beantworten haben oder dass ein Magnetventil in einer pneumatischen Vorrichtung binnen 5 Millisekunden zu schalten hat.

Aber auch die Benutzer von Desktoprechnern brauchen Echtzeit, wenn sie beispielsweise spielen und die sicht- oder hörbare Reaktion auf Tastendrücke innerhalb von 10 Millisekunden erwarten oder wenn sie ein Video trotz im Hintergrund laufender Applikationen anschauen, was von Bild zu Bild maximal 63 Millisekunden oder gleichmäßig mindestens 16 Bilder pro Sekunde bedeutet.

Wie Benutzer aus eigener Erfahrung wissen, ist das Antwortverhalten von Standardsystemen wie Linux oder MS Windows meistens ganz gut – aber beileibe nicht immer. Das als Lag bekannte Verhalten, also kurzzeitige Hänger, ein eingefrorenes Bild oder Tonaussetzer, weil irgendeine Routine eine Obergrenze überschritten hat, führt dazu, dass der Benutzer warten muss. Ursache ist häufig ein Treiber, der eine überraschend lange Zeit benötigt, zum Beispiel beim Zugriff auf einen USB-Speicherstick.

Weich und hart

Das Nicht-immmer-Einhalten der Obergrenzen wird weiche Echtzeit genannt und funktioniert nach dem Motto "It works most of the Time". Was für den Benutzer eines PC ab und zu nur eine ärgerliche Verzögerung beim Bedienen bedeutet, ist für eine gesteuerte Maschine funktional unzureichend, denn viele Aktionen – wie ein Teil auf einem Fließband durch Station *X* zu bewegen und dabei zu bearbeiten oder einen Schussfaden in ein Gewebe zu schießen – dauern nur eine Zehntelsekunde oder noch kürzer, sodass schon ein Überschreiben von wenigen Millisekunden oft einen Fehler verursacht, der zum Stillstand der Maschine führt und den das Personal manuell beheben muss.

Harte Echtzeit bedeutet in der Praxis nicht nur das Einhalten von Ober-, sondern oft auch von Untergrenzen, beispielsweise wenn ein Steuerbus für 100 plus/minus 20 Mikrosekunden einen High-Pegel führen soll oder wenn eine Maschinensteuerung noch 1 Millisekunde warten muss, bis eine Achse den richtigen Winkelbereich erreicht hat.

Gleichzeitig Unter- und Obergrenzen einhalten und Wartezeiten im Bereich von 50 Mikrosekunden bis 1 Millisekunde exakt gewährleisten, gelingt vielen Echtzeit-Betriebssystemen wie dem Microsys OS-9 (auf Intel x86, [1]) nur mit Busy-Waiting. Linux kann das Gleiche auch mit Timer-Interrupts realisieren, sodass es keinen CPU-Kern mit Warten blockieren muss. Ein anderer Vorteil von Linux ist, dass es alle CPU-Kerne nutzt und sich daher auch für größere Echtzeitsysteme eignet, während etliche andere Echtzeit-Betriebssysteme – wie OS-9 – nur den ersten Kern betreiben.

Linux bietet noch andere Vorteile, etwa dass es Open Source ist und keine Lizenzgebühren anfallen, dass es Tausende freie Tools und Anwendungen gibt und ein

Heer von Kernelprogrammierern und -testern ständig daran entwickelt.

Achtung: Stromfresser

Allerdings erzeugt Echtzeit mit Antwortzeiten im zweistelligen Mikrosekundenbereich auch in Linux zusätzlichen Aufwand in Form von Tuning, beispielsweise abgeschaltete CPU-Sleep-States und CPU-Frequency-Scaling, was auch mehr Stromverbrauch bedeutet. Und die ständigen Unterbrechungen durch den Kernel produzieren einen Overhead, unter dem die Performance einige Prozent leidet. Verantwortliche von Systemen, die mit hängenden Treibern rechnen müssen – siehe das USB-Stick-Beispiel oben – sollten für harte Echtzeit die problematischen Treiber nicht laden oder zumindest den Zugriff darauf limitieren.

Ein Echtzeit-Betriebssystem ist daher nicht für jeden Rechner sinnvoll, beispielsweise auf Notebooks, die eine lange Akkulaufzeit haben sollen. Daher gibt es als Kompromiss die Low-Latency-Kernel, die geringere Latenzen als Standardkernel aufeisen, aber kein zusätzliches Tuning erfordern.

Es gibt mehrere Möglichkeiten, harte Echtzeit zu realisieren. Ein stromsparender 16-Bit-Mikrocontroller wie der Texas Instruments MSP430 [2] eignet sich, wenn er die Aufgaben ohne Betriebssystem in einer schnell laufenen Endlosschleife (Main Loop) abarbeitet. Oder man nimmt einen Mikrocontroller mit einem sehr kleinen Echtzeit-Betriebssystem, auf dem die eigene Steuersoftware läuft. Dieser Artikel fokussiert eine dritte Möglichkeit: Der Entwickler wählt einen vergleichsweise leistungsstarken Rechner aus, einen Standard-PC etwa, installiert darauf ein Linux mit all seinen Möglichkeiten sowie die Echtzeitanwendung.

RTAI-Linux

Linux hat mehrere Pfeile im Köcher, um Echtzeitforderungen zu erfüllen [3], RTAI beispielsweise: Das Real Time Application Interface realisiert einen Echtzeitkernel, der den eigentlichen Linux-Kernel als Idletask laufen lässt ([4], Abbildung 1). Der Programmierer der Echtzeit-Tasks entwickelt zudem keine Linux-Programme, sondern Kernelspace-Code für den RTAI-Echtzeitkern.

Ihm stehen dabei also keine Userspace-Aufrufe zur Verfügung, sondern nur RTAI-Funktionsaufrufe wie »rt_task_init()«, »rt_task_delete()« oder »rt_busy_sleep()«. Diese Tasks sind naturgemäß daher nicht portabel und das Zielsystem muss sie mit »insmod« starten und mit »rmmod« beenden.

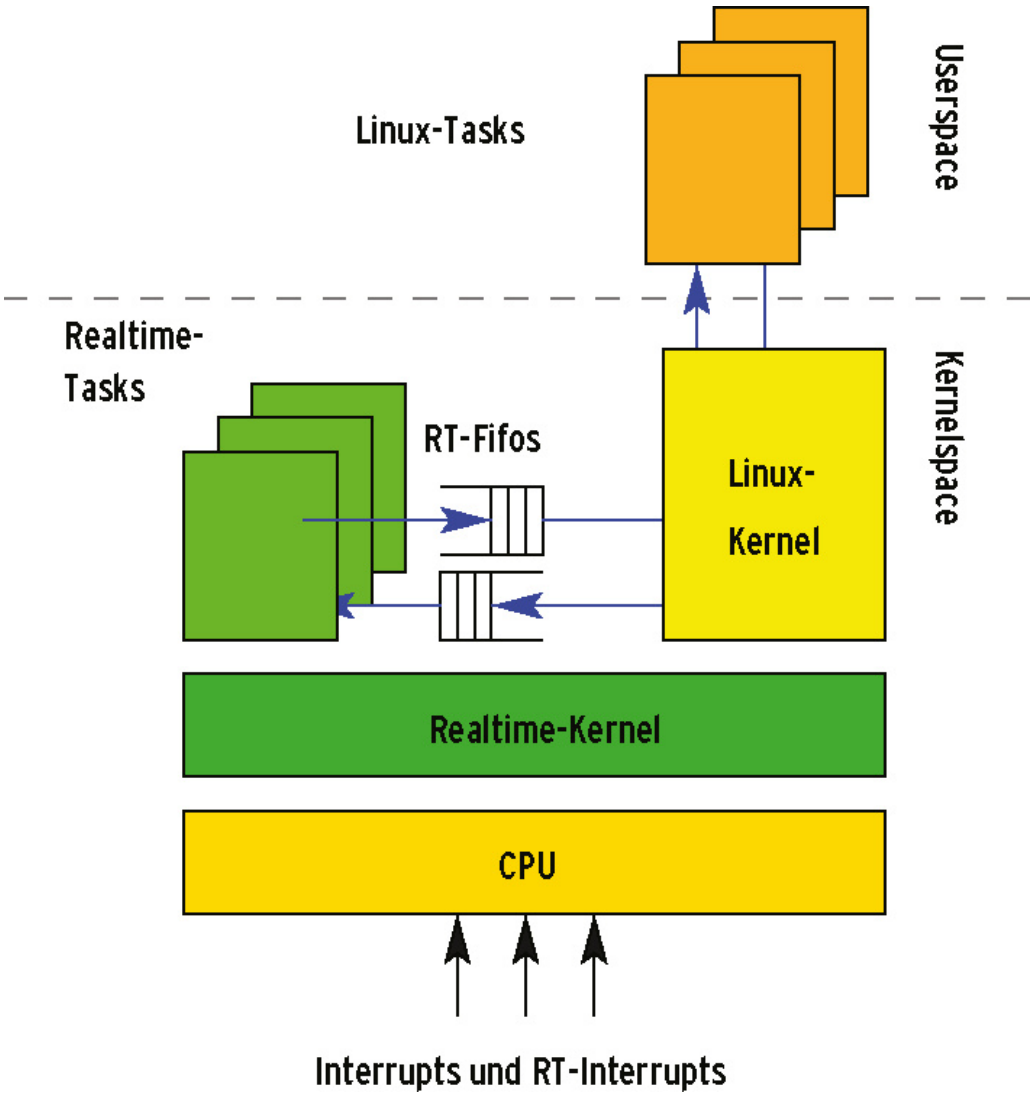

Abbildung 1: Dual-Kernel-Ansatz: Linux als Idle-Task eines Echtzeit-Kernels, der die Hardware steuert. Der Datenaustausch zwischen beiden Teilen erfolgt über Message-Queues.

Preempt-RT-Patch: Ein langer und ein Königsweg

Die meisten Linux-Echtzeit-Anwendungen laufen heutzutage auf einem Kernel mit dem Preempt-RT-Patch [5], das den Linux-Kernel selbst echtzeifähig macht. Solche Anwendungen laufen im Userspace und müssen sich nicht auf spezielle Echtzeit-Aufrufe beschränken. Sie lassen sich leicht sehr portabel gestalten, beispielsweise als Posix-Threads.

Dank des Preempt-RT-Patch erreicht der Entwickler also eine sehr gute Wiederverwendbarkeit seiner Software. Die Summe seiner Eigenschaften macht Linux mit Preempt-RT-Patch konkurrenzlos unter den Echtzeit-Betriebssystemen.

Um zu einem Echtzeit-Kernel zu kommen, gibt es zwei Wege. Die manuelle Variante, die bei den meisten Distibutionen funktioniert, besteht darin, die Vanilla-Kernelquellen und das Preempt-RT-Patch [6] zu holen, zu patchen, zu konfigurieren, zu kompilieren, zu installieren, zu booten und zu testen. Die Beschreibung dazu und einiges mehr liefert das RT-Preempt-Howto [7].

Das Vorgehen ist allerdings kompliziert, allein schon weil die Kernelkonfiguration je nach Make-Optionen für die Konfigurationsdatei (».config«) einige der für Echtzeit nötigen Optionen nicht anzeigt, sodass der Entwickler auch mit Aktivieren aller vorhandenen »PREEMPT«-Einträge meist keinen Echtzeitkernel erhält. Daher ist es sinnvoll, im ersten

Schritt die Konfiguration von einem problemlos laufenden Kernel mit »yes "" | make oldconfig« zu übernehmen.

Im zweiten Schritt holt der Entwickler die fehlenden »PREEMPT«-Optionen aus der Konfiguration (».config«), die er mit allen Optionen erstellt hat, also per »yes "" | make allyesconfig«. Das funktioniert unter Ubuntu 15.04 Server ganz gut, mit einer Worst-Case-Latenz von 35 Mikrosekunden auf einem Standard-PC.

Die zweite Variante hingegen, die vorgefertigte Komponenten benutzt, ist viel einfacher zu realisieren und weist daher der Königsweg: Seit Kernel 3.2 (Januar 2012) reicht es, auf Debian-Systemen

```
aptitude install linux-image-rt-amd64 linux-image-rt-amd64-dbg linux-he
```

zu tippen, und schon hat man einen Echtzeitkernel. Für andere Plattformen als AMD64 muss der Admin entsprechende andere Pakete angeben. Egal ob mit vorgefertigten Kernen oder zu Fuß: Das Preempt-RT-Patch funktioniert für viele Plattformen, auch den Raspberry Pi. Beispiele sind: [8] und [9] (zum selber Patchen) oder [10] (vorgefertigt).

Unabhängig von der Installationsvariante muss der Systemverwalter gegebenenfalls den Bootloader anpassen, um den Echtzeitkernel (RT-Kernel) automatisch zu booten. Das gelingt ihm zumeist mit dem Parameter »DEFAULT« in der »/etc/default/grub«-Datei. Ob ein RT-Kernel gebootet ist, verrät »cat /sys/kernel/realtime«, das im Positivfall mit »1« antwortet.

Die Hardware fit machen

Läuft der Kernel prinzipiell, kann es mit dem Echtzeit-Tuning der Hardware losgehen. Auch hierzu gibt das RT-Preempt-Howto [7] wertvolle Tipps. Primär geht es darum, Energiesparmaßnahmen wie CPU-Frequency-Scaling und -Sleepstates im Kernel und per Bios-Setup abzuschalten und im Gegenzug Turbo Boost und andere Übertaktungsmaßnahmen zu aktivieren. Als Ratgeber für den richtigen Weg eignet sich das Intel-Tool Powertop ([11], [12], Abbildung 2). Außerdem gehören virtuelle CPU-Cores (Hyperthreading) abgeschaltet.

Für Embedded-Boards wie den Raspberry Pi, wo es all dies nicht gibt, entfallen diese Tuningmaßnahmen natürlich. Für einige Prozent mehr Performance beziehungsweise geringere Worst-Case-Latenz sorgt aber das Übertakten. Bei den Applikationen sollte der Betreiber ein kleines Warm-up von knapp einer Sekunde einbauen und zur Qualitätskontrolle und zum Rückverfolgen von Aussetzern möglichst eine protokollierende Latenzen-Selbstüberwachung.

Dienlich ist es, das System schlank zu halten, nur wichtige Dienste laufen zu lassen und unbenutzte Hardware nicht per Treiber zu unterstützen. Wer kein USB benötigt, kann dies schon im Bios-Setup deaktivieren – ein Muss sind solche Abschaltmaßnahmen aber nur, wenn das System messbar zu hohen Worst-Case-Latenzen neigt.

Weiterhelfen können geschickt gesetzte IRQ-Prioritäten. Denn das RT-Patch zeichnet aus, dass Interrupthandler als eigene Threads laufen und somit untereinander und im Verhältnis zu den anderen Prozessen priorisierbar sind. Technisch möglich, aber selten sinnvoll ist es, die Threads an einzelne Kerne zu binden. Der Flaschenhals entwickelt sich, zumindest bei PCs, meist woanders: So versetzen SMIs (System Management Interrupts) den Rechner ab und zu in den System Management Mode (SMM) und halten das Betriebssystem dabei kurz an – was naturgemäß für eine Echtzeitanwendung fatale

Folgen hat. SMIs ist manchmal schwer beizukommen, weil sie das Bios beziehungsweise (U)EFI auslösen.

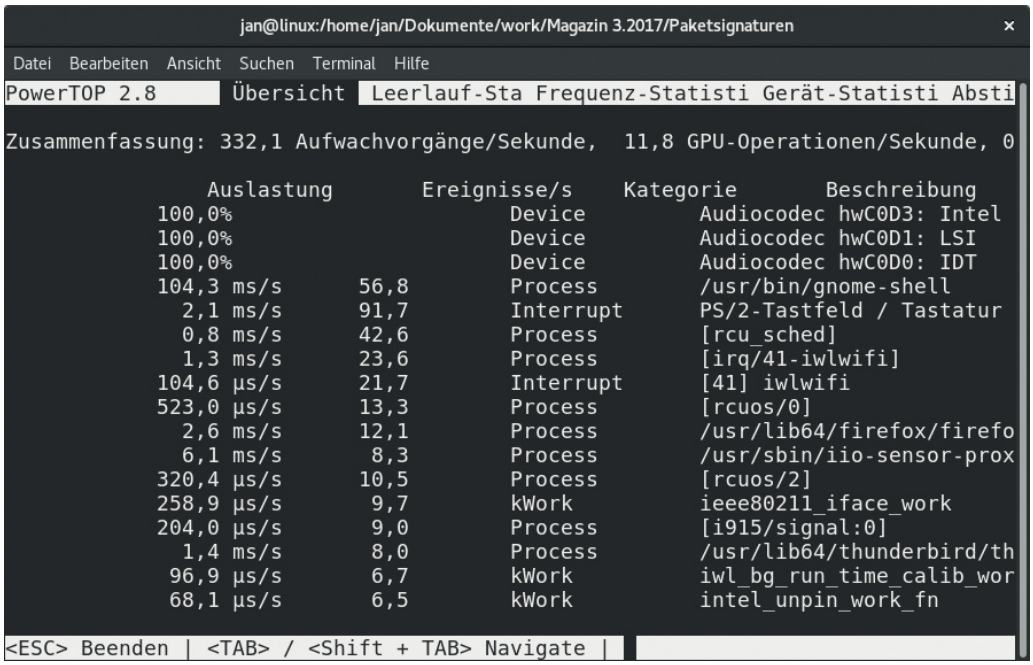

Abbildung 2: Powertop soll diesmal nicht beim Stromsparen helfen, sondern beim Performancetuning.

Stress machen beim Testen

Der Anwender eines RT-Kernels sollte zuerst testen, welche minimalen Latenzzeiten er mit seinem Zielrechner erreichen kann, damit er nicht versucht etwas zu programmierern, was die Hardware nicht hergibt. Zum Testen gibt es mehrere Benchmarkprogramme und als Hilfestellung auch Stressprogramme, welche die Latenzen hochtreiben.

Erste Versuche des Autors mit dem Spezialtool Worstcase Latency Test Scenario [13] schlugen allerdings fehl. Seinen Test-PC, einen alten Fujitsu Esprimo E9900 (Abbildung 3), testete das Tool in Grund und Boden (auch nach Reduzierung auf nur zwei aktive Kerne). Der Linux Watchdog Deamon (WDT) jedenfalls beschwerte sich per E-Mail:

```
Message from watchdog:
The system will be rebooted because of error 253!
```

Und der PC vollführte einen Reboot. Die so ermittelte Worst-Case-Latenz wäre praxisfern ausgefallen.

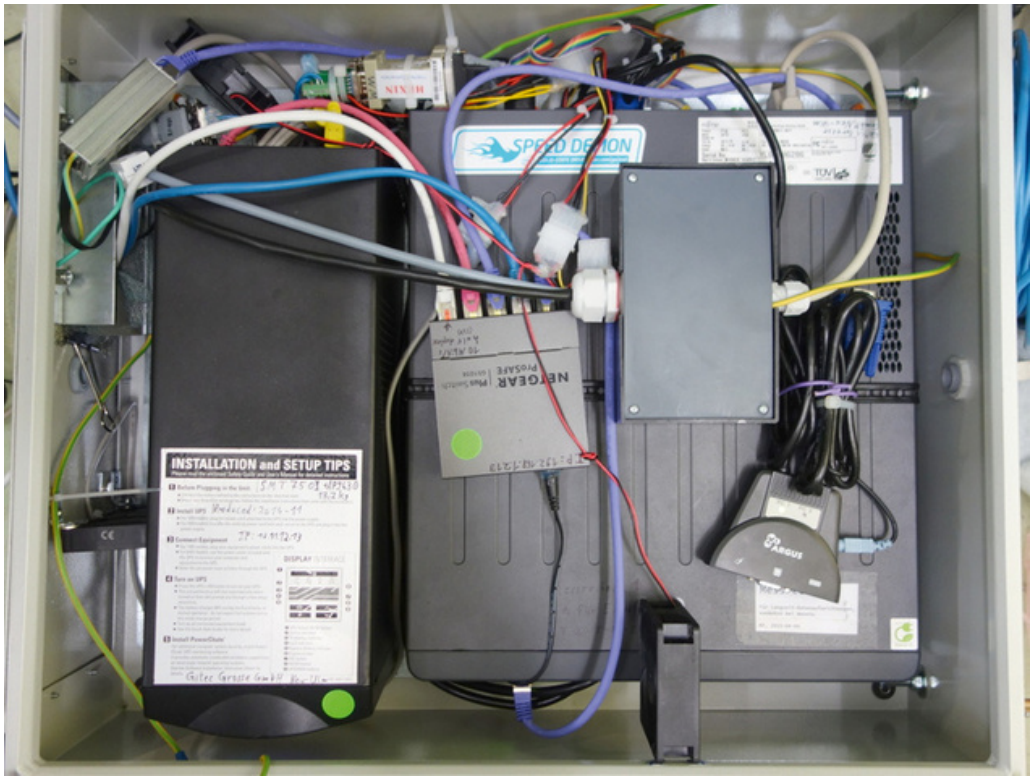

Abbildung 3: Diente dem Autor für ein halbes Jahr als Langzeit-Testsystem: Der Überwachungsrechner Esprimo E9900 eines Kunden aus der Textilindustrie, ausgestattet mit einem Parallelport und zwei serielle Schnittstellen.

Gute Ergebnisse dagegen lieferte das Benchmarktool Cyclictest [14], wenn man es mit einem Thread pro CPU-Kern über längere Zeit laufen lässt, zumindest ein paar Tage. Das gelingt am einfachsten mit einem Skript oder Alias wie:

```
alias cyclictesting='nice --20 ionice -c2 -n0 cyclictest -a -t -n -p99'
```

Cyclictest läuft hier im Userspace, nicht im Kernelspace wie bei anderen Echtzeit-Betriebssystemem. Allerdings muss Root es starten, damit es mit hoher Priorität arbeiten darf.

Das Tool verursacht nur zirka 1 Prozent CPU-Load, sodass es auch bei der eben verwendeten Priorität 99, der höchsten, andere Threads nicht behindert. Während des Tests sollte das System etwas unter Stress stehen, beispielsweise indem auf jedem Kern im Hintergrund ein CPU-Burn-Thread läuft und das Betriebssystem ein Update absolviert.

Die Stressaktivitäten sollen erreichen, dass sich die Worst-Case-Latenz einigermaßen zügig zeigt und nicht erst nach Wochen. Wichtig ist, an dem System möglichst alles zu testen, was im späteren Einsatz passieren kann, etwa das Lesen von einem USB-Stick, um später keine Überraschungen zu erleben.

Der Praxis-Vergleich

War auf dem Test-PC des Autors ein Kernel mit RT-Tuning aktiv, konnte Cyclictest eine Worst-Case-Latenz von zirka 35 Mikrosekunden messen, was periodische Aktionen mit dieser Zyklusdauer möglich macht, also rund 30 000 Aktionen pro Sekunde. Das änderte sich auch nicht, wenn nebenbei einiges andere lief, beispielsweise wenn der PC einen Kernel kompilierte, SSH- und FTP-Server sowie irgendwelche Spiele aktiv waren. Auch CPU-Burn-Programme veränderten die Worst-Case-Latenz nicht.

Ohne das Preempt-RT-Patch, also mit Kerneln vom Typ Generic oder Low Latency, zeigten sich Worst-Case-Latenzen von mehreren Dutzend Millisekunden bereits nach wenigen Minuten bis Tagen, abhängig von der Auslastung. Interessant ist auch, dass zu weit über 99,9 Prozent der Zeit die Latenz unter 10 Millisekunden lag, was bedeutet, dass der Rechner die meiste Zeit "sofort" reagiert.

Mit dem RT-Kernel, aber ohne Tuning zeigte sich ohne Stress zunächst eine Worst-Case-Latenz von nur 30 Mikrosekunden. Nach einigen Tagen (Hunderte Millionen Messungen später) kommt aber ein rund tausendmal höherer Wert, vergleichbar mit einem Kernel ohne Preempt-RT-Patch. Allein die Software zu optimieren genügt also nicht – für harte Echtzeit muss der Entwickler auch die Hardware tunen.

Anhand des gemessenen Worst-Case-Wertes kann der Entwickler seine Anwendung programmieren. Beim Test der eigenen Software darf dann wieder Cyclictest parallel laufen. Verwendet werden hierbei häufig Posix-Threads [15] und C, so auch beim vom Autor geschriebenen Programm Multithreaded_logger [16]. Es liest Daten über serielle Schnittstellen und Parallelports (Abbildung 4) ein, wobei der Anwender die Zykluszeit beim Einlesen der parallelen Daten als Kommandozeilenoption vorgibt.

Abbildung 4: Die Low-Profile-PCIe-Karte TP35 ist auch bekannt unter der Bezeichnung 9815EL-4, kostet zirka 50 Euro und besitzt dank ihrer vier Parallelports bis zu 52 Eingangspins.

Entstanden war das Programm, um Daten über serielle Schnittstellen mit hoher Zeitauflösung einzulesen und mit genauen Zeitstempeln zu versehen. Dann hat es der Autor ausgebaut, um auch parallele Daten einzulesen – über RS-232-Kontrolleingänge oder Parallelports. Das Ergebnis ist also ein Hybrid aus einem seriellen Datenlogger mit *x* seriellen Schnittstellen, einem Transientenrekorder mit *y* parallelen Schnittstellen und aus hochgenauen Zeitstempeln für serielle und parallele Daten.

Der Aufbau kurzgefasst: Es gibt einen Timer-Thread für periodische Aufgaben und zum Auswerten von Ist- und Sollzeit. Ein Main-Thread koordiniert den Ablauf. Außerdem bekommt jede Schnittstelle (seriell, parallel) einen extra Thread. Der Timer-Thread läuft mit der höchsten Priorität, die davon getriggerten (parallelen) mit der zweihöchsten, die

IRQ-getriggerten (seriellen) mit der dritthöchsten, der Thread zum Leeren des Augangspuffers der Logdatei läuft mit der niedrigsten. Die seriellen Threads erkennen und defragmentieren fragmentierte Datenpakete. Der Datenaustausch erfolgt ohne Locks über große Ringpuffer.

Die Worst-Case-Latenzen entsprechen denen von Cyclictest, weil die Portzugriffe unter 2 Mikrosekunden dauern und in separaten Threads passieren. Zum Starten sollte man ein Skript schreiben, das auch die Schnittstellen konfiguriert, beispielsweise Baudraten. Ein Beispiel zeigt der Test [17]. Wer einen Dauerbetrieb plant, bindet das Skript als Upstart-Service ein.

Fazit

Linux ist als Universal- und nicht als Echtzeit-Betriebssystem konzipiert, gleichwohl eröffnet es mehrere Möglichkeiten, eigenen Anwendungen zu hartem Realtime-Verhalten zu verhelfen. Die wohl beste Linux-Modifikation hört auf den Namen Preempt-RT-Patch, sie bringt Entwickler quasi in den programmiertechnischen Himmel.

Außerdem erreichen damit präparierte Systeme Worst-Case-Latenzen um die 35 Mikrosekunden sogar im Userspace, auch mit sehr portablen Programmen und Posix-Threads. Damit eignet es sich nicht nur für große Profi-Systeme, sondern auch für Anfänger – viele einfache Harte-Echtzeit-Aufgaben können sie mit einer simplen Endlosschleife erledigen, ganz ohne explizite Threads. (jk)

Infos

1. OS-9: [http://www.microware.com]
2. TI MSP430: [https://de.wikipedia.org/wiki/TI_MSP430]
3. Kunst, Quade, "Grundlagen: Echtzeitsysteme mit Linux": Linux-Magazin 06/08, S. 106, [http://www.linux-magazin.de/Ausgaben/2008/06/Gerade-echtzeitig/]
4. RTAI: [https://www.rtai.org]
5. Preempt-RT-Patch: [https://wiki.linuxfoundation.org/realtime/start]
6. Preempt-RT-Patch.Quellen: [https://www.kernel.org/pub/linux/kernel/projects/rt/]
7. RT-Preempt-Howto: [https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO]
8. Rainer Poisel, "Echtzeitfähige Linux-Systeme: Eine Beispielumsetzung anhand des Raspberry Pi's": [http://logicals.com/de/forum/raspberry-pi/15-echtzeitfaehige-linux-systeme-eine-beispielumsetzung-anhand-des-raspberry-pi-s]
9. "Raspberry Pi Going Realtime with RT Preempt": [http://www.frank-durr.de/?p=203]
10. Emlid, "Raspberry Pi real-time kernel `[SD image available]`": [https://emlid.com/raspberry-pi-real-time-kernel/]
11. Powertop: [https://01.org/powertop/]
12. Feilner, Kromer, Schürmann, Huber, Kleinert, "Tipps, Tricks und Tools, mit denen Notebooks länger laufen": Linux-Magazin 03/12, S. 22, [http://www.linux-

magazin.de/Ausgaben/2012/03/Rat-und-Tat]

13. Worstcase Latency Test Scenario: [https://rt.wiki.kernel.org/index.php/Worstcase_Latency_Test_Scenario]
14. Cyclictest: [https://rt.wiki.kernel.org/index.php/Cyclictest]
15. Posix-Threads: [https://en.wikipedia.org/wiki/POSIX_Threads]
16. Multithreaded_logger: [http://true-random.com/homepage/projects/logger/index.html]
17. Test 3: [http://true-random.com/homepage/projects/logger/test3.sh]

Der Autor

Dr. Rolf Freitag ist Informatiker und promovierter Physiker. Derzeit arbeitet er bei einem kleinen Unternehmen in Neu-Ulm.