ECHTZEITFÄHIGKEIT VON LINUX-SYSTEMEN EMPIRISCH BESTIMMEN

Latenzen auf der Spur

Für die Bestimmung der Echtzeitfähigkeit aktueller Computerboards mit modernen Hochleistungsprozessoren ist die klassische Pfadanalyse ungeeignet, doch mit empirischen Testverfahren lässt sich die Größenordnung der maximalen Systemlatenz bestimmen. Der erforderliche Aufwand richtet sich danach, ob eine neue Hardware entwickelt wurde, ein definiertes System in einem speziellen Anwendungsfall zu testen ist oder Tests im Rahmen der kontinuierlichen Qualitätskontrolle erfolgen sollen.

CARSTEN EMDE
THOMAS GLEIXNER
ROBERT SCHWEBEL

Damals, in den guten alten Tagen des Mikroprozessors, gab es noch keinen nennenswerten Cache und keine Mehrprozessorsysteme, und jedes Peripheriegerät verfügte über einen eigenen Interrupt oder einen eigenen Interruptvektor. Damals war es möglich, die maximale Latenz eines Systems theoretisch zu bestimmen. Für diesen Zweck suchte der Entwickler den längstmöglichen Codepfad, addierte die Anzahl der dafür erforderlichen CPU-Zyklen und multiplizierte das Ergebnis mit der jeweiligen Zyklusdauer des verwendeten Systems. Wer sicher sein wollte, bei dieser Pfadanalyse keinen Weg zu vergessen, führte sicherheitshalber noch Latenzmessungen durch. Jahrelang war die Pfadanalyse ein verlässliches Mittel zur Bestimmung der maximalen Latenz eines Computersystems.

Seit der Einführung von teilweise sehr großen Cache-Speichern, die häufig in komplexer Weise hintereinander geschaltet beziehungsweise von mehreren Prozessorkernen benutzt werden, von konkurrierenden Bus-Mastern bei DMA- und Mehrprozessor-Systemen und von gemeinsam benutzten Interruptleitungen ohne Vektorisierung ist die theoretische Bestimmung der höchstmöglichen Latenz eines Computersystems durch Pfadanalyse jedoch extrem erschwert oder sogar unmöglich geworden. Selbst wenn in einzelnen Fällen die Bestimmung noch machbar erscheint, ist nicht auszuschließen, dass dabei der eine oder andere Verzögerungseffekt übersehen wurde. Für diese Fälle und für solche, in denen die Pfadanalyse von vornherein ausgeschlossen ist, muss es Methoden zur Latenzmessung geben, welche die empirische Bestimmung der maximalen Latenz eines Computersystems gestatten. Neben derartigen Latenzmessungen sind aber auch Messbedingungen zu definieren, die den Prüfling einer definierten Last aussetzen, um damit potenzielle Latenzquellen sichtbar zu machen. Denn ein Computersystem ohne Last wartet gewissermaßen ständig auf ein Ereignis und kann daher in der Regel auch sehr kurzfristig auf ein solches reagieren.

Zunächst einmal ist aber zu definieren, mit welcher Anforderung Latenzmessungen erfolgen: Handelt es sich um die Messung bei einem Hard- oder Softwarehersteller, gilt es also, sämtliche unter irgendwelchen Umständen jemals möglichen Latenzen aufzudecken? Oder führt ein Maschinenbauer die Messung an einer speziellen Anlage durch, geht es also in erster Linie um Latenzen, die unter den individuellen Konfigurations- und Lastbedingungen dieser Anlage auftreten können? Weiterhin ist zu unterscheiden, ob die Messung einmaliger und grundsätzlicher Natur ist (also zum Beispiel vor der Festlegung auf eine bestimmte Hard- und Softwarekonfiguration) oder ob es sich um wiederholte Messungen zur kontinuierlichen Qualitätskontrolle handelt. Im letzteren Fall sind natürlich in erster Linie die im Rahmen der Produktpflege weiterentwickelten beziehungsweise veränderten Komponenten zu überprüfen. Entsprechend ist dieser Artikel eingeteilt in komplette »Über-Alles-Messungen«, also die Fahndung überall im System nach irgendwie möglichen Latenzen, und spezielle Messungen individueller Systeme, also das Bestimmen der höchstmöglichen Latenz, die unter den spezifischen Einsatzbedingungen einer Konfiguration auftreten kann.

Obwohl die Konzepte und Methoden in diesem Artikel für die meisten Betriebssysteme zutreffen, beziehen sich die Beispiele in erster Linie auf »Mainline-Linux«, das mit den so genannten »Realtime-Preempt-Patches« echtzeitfähig gemacht wurde. Nähere Angaben dazu finden sich im Realtime-Wiki (rt.wiki.kernel.org) und auf der Webseite des Open Source Automation Development Lab (OSADL).

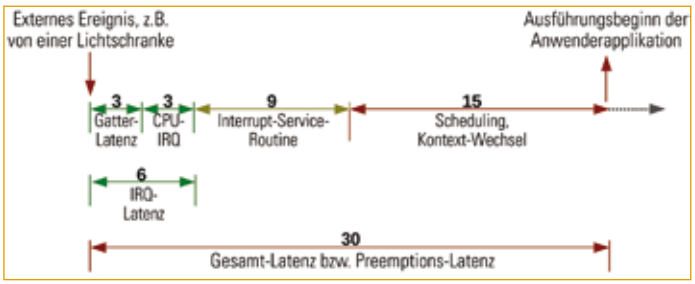


Bild 1: Die Bearbeitung eines externen Ereignisses in den verschiedenen logischen Segmenten des Betriebssystems, vom Eintreffen des Signals (hier einer Lichtschranke) bis zum Beginn eines Anwenderprogramms (hier einer Förderbandweiche) – Zahlenangaben: Verarbeitungszeiten in Mikrosekunden, wie sie z.B. bei einem 100-MHz-Prozessor denkbar sind

Vom Signal zum Anwenderprogramm

Bild 1 zeigt den Verlauf eines Eingangssignals einer exemplarischen Maschinensteuerung. Als Beispiel dient hier das Signal einer Lichtschranke zur Steuerung einer Förderbandweiche. Nach einem bestimmten Zeitintervall, das einen vordefinierten Wert nicht überschreiten darf, muss das Anwenderprogramm zur Ausführung kommen und die Förderbandweiche betätigen. Ist dies wegen einer zu langen Latenz nicht der Fall,

kommt es zu einer Fehlfunktion der Maschine.
Mögliche Verzögerungen können sich in den verschiedenen Segmenten ergeben, die das Signal durchlaufen muss:

- Gatter-Latenz:

Der Controller, an den das Ausgangssignal der Lichtschranke (z.B. TTL) angeschlossen ist, muss das Signal verarbeiten und am Ende dieser Verarbeitung die Interruptleitung des Prozessors aktivieren. Diese Zeit beträgt meist deutlich weniger als 1 µs, aber bei Fehlkonfiguration beziehungsweise Fehlfunktion des Controllers oder zu starker Dämpfung des Eingangssignals können in diesem Segment nennenswerte Latenzen entstehen.

- CPU-IRQ:

Nachdem die Interruptleitung des Prozessors aktiviert ist, muss dieser die aktuelle Verarbeitung unterbrechen, den Kontext sichern, die zu diesem Interrupt gehörende Interruptroutine bestimmen, den erforderlichen Kontext bereitstellen und mit der Ausführung der Interrupt-Serviceroutine beginnen. Auch dieses Segment wird in der Regel in sehr kurzer Zeit durchlaufen, kann aber in einzelnen Fällen ebenfalls durchaus Ursache für Systemlatenzen sein, wenn beispielsweise bei Systemen mit Daten- oder Befehls-Cache für den Wechsel des Kontextes keine freien Cache-Linien verfügbar sind beziehungsweise der Code nicht aus dem Cache ladbar ist und daher Speicherzugriffe notwendig werden.

- Interrupt-Serviceroutine:

Diese besteht meist aus zwei funktional sehr unterschiedlichen Bestandteilen: Die erste Phase der Interrupt-Serviceroutine ist von häufigen Hardwarezugriffen geprägt. Diese sind erforderlich, um die Interruptlogik des Controllers zurückzusetzen und den Datentransfer vorzubereiten. Wenn sich mehrere Geräte eine Interruptleitung teilen, muss die Routine sogar noch die verschiedenen in Frage kommenden Geräte abfragen. Die zweite Phase regelt dann den Datentransfer, der mit dem jeweiligen Interrupt verbunden ist. In vielen Fällen kann aber schon nach Abschluss der ersten Phase die Kontrolle wieder an den Kernel zurückgehen, sodass die zweite Phase dann im Wechsel mit anderen Prozessen ablaufen kann.

- Scheduling und Kontext-Wechsel:

Sobald feststeht, welcher Prozess auf Daten des Gerätes wartet, das den Interrupt ausgelöst hat, und der Datentransfer beginnen kann, weckt der Kernel den wartenden Prozess auf. Dieser so genannte Wakeup-Vorgang bedeutet, dass der betroffene Prozess in die Warteschlange der aktiven Prozesse aufgenommen und je nach dessen Priorität ausgeführt wird. Für die Ausführung muss der Kernel den Kontext des Programms bereitstellen, in den Usermodus wechseln und den Prozessor das wartende Programm abarbeiten lassen.

Komplette Messung: Wo sind die Latenzen?

Zur Messung möglicher, irgendwo im System auftretender Latenzen speist man ein Eingangssignal für einen dafür geeigneten Controller ein, installiert ein Anwenderprogramm, das auf das Eintreffen dieses Signals wartet, und misst das Zeitintervall zwischen der Einspeisung des Signals und dem Beginn der Ausführung des Anwenderprogramms. Geeignete Eingangscontroller sind beispielsweise universelle digitale Input-Controller (GPIO), ersatzweise lässt sich aber auch die Eingangs-Handshake-Leitung eines seriellen Controllers nutzen. Der Ausführungsbeginn des Anwenderprogramms sollte vorzugsweise nicht mit Hilfe eines interruptgesteuerten Ausgabegerätes markiert werden, da hierdurch eine zusätzliche Verzögerung entsteht. Am besten sind universelle Output-Controller geeignet, deren Ausgangssignal sich unmittelbar durch einen Schreibvorgang in ein Controller-Register setzen lässt.
Die beschriebene Messung erleichtert die so genannte »OSADL-Latency-Box« (Bild 2), ein kombiniertes, unabhängig arbeitendes Gerät, das Triggergenerator und Messgerät vereinigt. Es stehen zwei digitale Ausgänge und vier digitale Eingänge zur Verfügung (siehe Kasten). Bis zu zwei unabhängige Messvorgänge kann das Gerät mit jeweils einem Trigger- und zwei Messkanälen ausführen. Gleichzeitig kann der zweite Messkanal zum Beispiel die Dauer vom Eingangssignal bis zum Beginn der Interrupt-Serviceroutine und von dort bis zum Ausführungsbeginn des Anwenderprogramms bestimmen. Sämtliche Messwerte speichert die »Latency-Box« in Form eines Histogramms mit einer Auflösung von 1 µs und einem Maximalwert von 1000 µs. Dabei speichert die letzte Histogramm-Zelle nicht nur die Häufigkeit von Werten zwischen 999 µs und 1000 µs sondern auch sämtliche Werte oberhalb davon. Nur wenn diese letzte Zelle am Ende einer Messung den Wert »0« enthält, kann also von einem verwendbaren Messergebnis ausgegangen werden. Anderenfalls sind Latenzwerte oberhalb von 1000 µs aufgetreten, deren genaue Größe aber unbekannt ist. Da die Messung als Ergebnis die maximale gemessene Latenz (auch »worst-case latency« genannt) liefern soll, muss natürlich sichergestellt sein, dass alle Messwerte ausschließlich innerhalb der Histogrammgrenzen von 0 µs bis 1000 µs liegen.
Die von der Firma Eltec Elektronik im Auftrag des OSADL entwickelte und gefertigte »Latency-Box« steht OSADL-Mitgliedern kostenlos leihweise zur Verfügung. Je nach Verfügbarkeit können allerdings auch Nichtmitglieder die »Latency-Box« nutzen beziehungsweise erwerben. Nach Abschluss einer Messung sind die Daten in Form einer ASCII-Datei über einen auf dem Gerät laufenden FTP-Server abrufbar. Die Rohdaten in dieser ASCII-Datei enthalten jeweils die Anzahl aufgetretener Latenzen mit einer Auflösung von 1 µs. Die weitere Verarbeitung der Daten erfolgt dann meist auf einem Desktop-PC. Für diesen Zweck stehen geeignete Skripts zur Herstellung grafischer Darstellungen mit Hilfe des Programms »gnuplot« zur Verfügung. Die Ausführung eines solchen Skripts ergibt eine Grafik, wie Bild 3 sie zeigt. Um eine optimale Beurteilung der Daten zu ermöglichen, ist die x-Achse (Latenzwerte) linear und die y-Achse (Anzahl an Messwerten pro Latenzwert) logarithmisch dargestellt. Der senkrechte Strich in grüner Farbe kennzeichnet die maximale Latenz, das wichtigste Ergebnis der Messung.

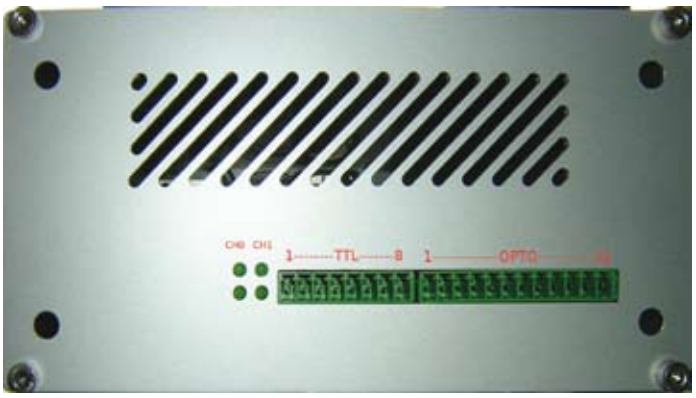


Bild 2: Frontplatte der »OSADL-Latency-Box«

Lastszenario erzeugen

Wie bereits erwähnt, ist eine Latenzmessung weitgehend sinnlos, wenn während der Messung keine geeigneten Stressbedingungen für das System bestehen. Dabei gilt es, die verschiedenen Subsysteme eines Betriebssystems zu berücksichtigen, die bekanntermaßen zu Latenzen führen, da diese Subsysteme besonders häufig auf gemeinsam genutzte Ressourcen zugreifen und ununterbrechbare Funktionen aufrufen. Als Folge davon müssen hier nämlich Ausführungspfade serialisiert werden, sodass nicht zu jedem beliebigen Zeitpunkt die sofortige Bearbeitung eines asynchron einfallenden externen Ereignisses möglich ist. Um eine geeignete Stressbedingung zu schaffen, müssen also eben diese Pfade während einer Latenzmessung besonders häufig, am besten kontinuierlich, ausgeführt werden. Bei den besonders in Frage kommenden Subsystemen handelt es sich um:

- Scheduler:

Es ist naheliegend, dass die hohe

CARSTEN EMDE

ist Geschäftsführer des Open Software Automation Development Lab

THOMAS GLEIXNER

ist Gründer und Geschäftsführer von Linutronix, einem OSADL-Gründungsmitglied

ROBERT SCHWEBEL

ist Gründer und Inhaber von Pengutronix, einem OSADL-Gründungsmitglied

Ausführungsfrequenz des Schedulers zu einer häufigen Konkurrenzsituation zwischen dem aktuell ausgeführten und dem mit Priorität gewünschten Scheduling-Vorgang führt. Daher sollte eine Stressbedingung des Schedulers diesen in hoher Frequenz aufrufen. Das von Ingo Molnar im Rahmen der Entwicklungsarbeit des »O(1)-Schedulers« des Linux-Kernels 2.6 erstellte Programm »hackbench« erfüllt diese Aufgabe besonders gut. Die parallele Ausführung des Shell-Kommandos in Codebeispiel 1 (siehe Kasten) hat bereits sehr häufig zur Aufdeckung versteckter Latenzen geführt. Das Kommandozeilen-Argument des Programms »hackbench« ist allerdings kritisch, da höhere Werte schnell zu einem hohen Speicherverbrauch führen. Hier hat sich die Formel »Hackbench-Argument = 2 + Speicherplatz in MByte / 32« bewährt. Bei einem 64-MByte-System wäre also ein Argument von 4, bei einem 1-GByte-System ein Argument von 34 angemessen.

- Memory-Manager und Speicherzugriffe:

Eine häufig verwendete Möglichkeit, den Memory-Manager und Speicherzugriffe unter Stressbedingungen zu versetzen, besteht in zyklischen Aufrufen des Tools »calibrator« (http://monetdb.cwi.nl/Calibrator/).

- Interrupt-Serviceroutinen:

Möglicherweise blockierende Eigenschaften der Interruptroutinen der installierten Geräte lassen sich am besten mit der jeweiligen I/O-Aktivität provozieren. Bei einer Netzwerkschnittstelle ist zum Beispiel das Flut-Ping-Kommando von einem anderen System im gleichen Netzwerk oder, noch besser, über eine Peer-zu-Peer-Verbindung gut geeignet. Die hohe Anzahl von hierdurch ausgelösten Interrupts lässt sich mit dem Interruptzähler in »/proc/interrupts« verifizieren.

- File-Manager:

Zum häufigen Aufruf von Funktionen eines File-Managers lassen sich zyklische Aufrufe zur rekursiven Ausgabe des Rootverzeichnisses ausführen, zum Beispiel die Codezeile in Codebeispiel 2.

- Standard-Benchmarkprogramme zur Erzeugung von Rechenlast wie zum Beispiel »UnixBench Rel. 4«:

Hierdurch werden Perioden mit sehr hoher und sehr häufig wiederkehrender Last in den einzelnen funktionellen Bereichen der CPU und des Kernels wie Speicherverwaltung, I/O, Integer-Arithmetik oder Fließkomma-Arithmetik erzeugt.

Latenzmessung kalibrieren

Vor einer Messung sollte sichergestellt sein, dass das Messsystem auftretende Latenzen auch entdeckt. Zu diesem Zweck eignet sich die Kalibrierung mit einem Treiber, der eine bekannte Latenz des Systems erzeugt. Ein solcher Treiber enthält zum Beispiel die in Codebeispiel 3 dargestellten Funktionsaufrufe und führt damit zu einer künstlichen Latenz für die Dauer der Warteschleife. Da sich aus den oben genannten Gründen die Dauer der Warteschleife nicht genau vorhersagen lässt, erfolgt eine Bestimmung der Schleifendauer innerhalb des Treibers, und diese Dauer wird nach Beendigung der Schleife im System-Logger ausgegeben. Einen solchen Treiber mit dem Namen »blocksys« und dem Userspace-Programm »mklatency« gibt es auf der Webseite www.osadl.org zum Download.

Technische Daten der OSADL-»Latency Box«

- CPU PowerPC 750FX @ 600 MHz, 512 KByte On-Chip-Level-2-Cache
- 64 MByte SDRAM auf SODIMM
- 10/100-MBit/s-Netzwerkinterface (10BaseT/100BaseTX)
- Zwei serielle Kanäle, RS232 und RS485 (optoentkoppelt)
- Embedded Linux BSP-Support
- 16 MByte (optional 32 MByte) Flash-EPROM
- Zwei optoentkoppelte TTL-Ausgangssignale
- Vier optoentkoppelte TTL-Eingangskanäle
- Vier LEDs zur Anzeige des Messzustandes

Spezielle Messung: Bestimmung des Ausgangswerts

Die Messung an einem individuellen System verläuft im Prinzip sehr ähnlich wie die oben beschriebene Komplettmessung. Allerdings sind bei der Messung an einem individuellen System nur die – in der Regel relativ begrenzten – Systemkomponenten zu konfigurieren und in das Lastszenario zu integrieren, die im jeweiligen System auch tatsächlich nötig sind. So muss die Messung eines Beispielsystems mit zwei Netzwerkcontrollern, bei dem aber nur eine Netzwerkschnittstelle an der Frontplatte liegt, den zweiten Controller weder initialisieren noch stressen.

Im Gegensatz zur initialen Messung muss die laufende Messung zur Qualitätskontrolle die Hardwarekomponenten zudem nicht mehr regelmäßig in die Messung mit einbeziehen, wenn sichergestellt ist, dass diese unverändert sind. Treiber und Kernel unterliegen hingegen nicht nur einer kontinuierlichen Entwicklung, sondern neue Versionen werden regelmäßig eingespielt und für aktuell auszuliefernde Produkte verwendet. Daher sind nach Software-Updates Tests durchzuführen, ob die zu Beginn festgestellte Echtzeitfähigkeit des Systems erhalten geblieben ist. Diese Tests müssen alle durch das Software-Update betroffenen Komponenten umfassen. Dies geschieht am besten mit dem Programm »cyclictest«, mit den eingebauten Kernellatenz-Histogrammen beziehungsweise mit Diagnosefunktionen im Anwenderprogramm.

Die Codebeispiele zum Beitrag

```
1.  while true; do hackbench 25; done
2.  while true; do ls -Ral /; done
3.  preempt_disable();
    local_irq_disable();
    while (nops--)
      asm(„nop“);
    local_irq_enable();
    preempt_enable();
4.  cyclictest -a -t -n -p99 -i100 -d25
5.  mount -t debugfs nodev /sys/kernel/debug
6.  nodev /sys/kernel/debug debugfs defaults 0 0
7.  CONFIG_WAKEUP_LATENCY_HIST=y
8.  CONFIG_MISSED_TIMER_OFFSETS_HIST=y
9.  CONFIG_INTERRUPT_OFF_HIST=y
10. CONFIG_PREEMPT_OFF_HIST=y
11. cd /sys/kernel/debug/tracing/latency_hist
    for i in `find . | grep /reset$`
    do
      echo 1 >$i
    done
```

»Cyclictest«

Thomas Gleixner entwickelte das Progamm »cyclictest« zunächst nur als internes Test-Tool zur Ermittlung von Regressionen im Test- und Release-Verfahren der Realtime-Preempt-Patches des Linux-Kernels. Inzwischen ist es aber zum Standard-Tool für die Bestimmung der internen Latenz eines Echtzeitsystems geworden. Das Hauptprogramm »cyclictest« startet eine definierbare Anzahl an Arbeits-Threads, die zyklisch jeweils auf einen Timer-Interrupt warten und die Zeit messen, die zwischen dem erwarteten und dem tatsächlichen Ausführungsstart vergeht. Diese Zeitdifferenz wird ausgegeben, und die maximale Latenz des Systems ergibt sich aus

der höchsten jemals gemessenen Zeitdifferenz. Dabei ist darauf zu achten, dass genügend Messungen erfolgen und zwischen den Messungen keine zu großen Intervalle bestehen, während derer bestehende Latenzquellen verloren gehen können. Als Faustformel sollte das Messintervall etwa doppelt so hoch sein wie die erwartete maximale Latenz des Systems und die Verzögerung zwischen den Arbeits-Threads etwa dem Messintervall geteilt durch die Anzahl an Prozessoren entsprechen. Bei einer erwarteten maximalen Latenz von 50 µs und vier Prozessoren lautet zum Beispiel der Aufruf des Programms »cyclictest« wie im Codebeispiel 4 im Kasten dargestellt.

Die Kommandozeilen-Argumente des Programms »cyclictest« bedeuten im Einzelnen:

`-a`

Affinity. Jeder Arbeits-Thread startet, wenn möglich, auf einer eigenen CPU, einem eigenen Prozessor-Core beziehungsweise einem eigenen Hyperthread. Gibt der Anwender ein Argument wie zum Beispiel

`-a0`

an, laufen die Threads ausschließlich auf diesem Prozessor.

`-t`

Threads. Es werden so viele Threads gestartet, wie CPUs, Prozessor-Cores beziehungsweise Hyperthreads vorhanden sind. Gibt der Anwender ein Argument wie zum Beispiel

`-t4`

an, wird genau diese Anzahl an Threads gestartet.

`-n`

Nanosleep-Clock. Dieses ist der Standard-Timer. Die anderen verfügbaren Timer sind nur zu Testzwecken implementiert und kommen meist nicht zur Anwendung.

`-p99`

Priorität. Diese Priorität bezieht sich auf den Thread 0. Weitere Threads erhalten jeweils diese Priorität, reduziert um die Nummer des Threads.

`-i100 -d25`

Thread-Intervall und Verzögerung zwischen den Threads (siehe oben).

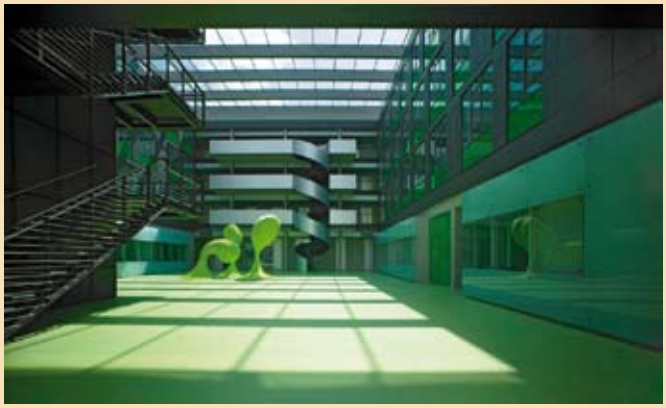
Für die erforderliche Anzahl an Messungen lässt sich keine allgemeine Angabe machen. Es ist aber von mindestens 10^9 durchzuführenden Messzyklen auszugehen. Im genannten Beispiel dauert eine Gesamtmessung dann etwa einen bis anderthalb Tage.

Kernel-Latenz-Histogramme

Nach Einspielung der RT-Preemption-Patches des Linux-Kernels stehen zusätzliche diagnostische Funktionen zur Verfügung. Unter anderem handelt es sich dabei um eingebaute kontinuierliche Latenzmessungen, die in einem Histogramm – sehr ähnlich wie bei der »OSADL-Latency-Box« – abgespeichert werden. Die Auflösung beträgt auch hier 1 µs, der Wertebereich der Histogramme liegt zwischen 0 µs und 10 240 µs. Um die Messwerte auslesen zu können, muss das Debug-Filesystem ansprechbar sein. Dies erfolgt entweder durch Eingabe der in Codebeispiel 5 dargestellten Kommandozeile oder durch Hinzufügen der Zeile in Codebeispiel 6 in die Datei »/etc/fstab«, damit das Debug-Filesystem nach einem Neustart sofort ansprechbar ist. Es stehen insgesamt sechs verschiedene Histogramme zur Verfügung, wobei die Daten für jeden Prozessor bzw. für jeden Prozessorkern oder bei Hyperthreading-Systemen für jeden Hyperthread getrennt abrufbar sind:

- Wakeup-Latency: eingeschaltet, wenn die entsprechende Konfigurationsoption (Beispiel 7 im Kasten) konfiguriert ist. Der Dateiname der Daten der ersten CPU lautet: »/sys/kernel/debug/tracing/latency_hist/wakeup/CPU0«. In dieses Histogramm werden nur Latenzen von Prozessen aufgenommen, in deren Wakeup-Phase kein anderer Prozess mit der gleichen oder einer höheren Priorität auf der jeweiligen CPU ausgeführt wurde. Eine zweite Gruppe von Wakeup-Latency-Histogrammen befindet sich im Unterdirectory »shared«; der Dateiname der Daten der ersten CPU dieser Histogramme lautet entsprechend: »/sys/kernel/debug/tracing/latency_hist/wakeup/shared/CPU0«. In dieses Histogramm werden Latenzen von Prozessen aufgenommen, in deren Wakeup-Phase andere Prozesse mit der gleichen Priorität pro CPU ausgeführt wurden.
- Missed-Timer-Offsets: eingeschaltet, wenn die entsprechende Konfigurationsoption (Beispiel 8 im Kasten) konfiguriert ist. Der Dateiname der Daten der ersten CPU lautet: »/sys/kernel/debug/tracing/latency_hist/missed_timer_offsets/CPU0«.
- IRQ-off-Latency: eingeschaltet, wenn die entsprechende Konfigurationsoption (Beispiel 9 im Kasten) konfiguriert ist. Der Dateiname der Daten der ersten CPU lautet: »/sys/kernel/debug/tracing/ latency_hist/irqsoff/CPU0«.
- Preemption-off-Latency: eingeschaltet, wenn die entsprechende Konfigurationsoption (Beispiel 10 im Kasten) konfiguriert ist. Der Dateiname der Daten der ersten CPU lautet: »/sys/kernel/debug/tracing/latency_hist/preemptoff/CPU0«.
- IRQ-und-Preemption-off-Latency: eingeschaltet, wenn die beiden Konfigurationen

Elfter »Real Time Linux Workshop« in Dresden



Vom 28. bis 30. September 2009 organisiert das Open Source Automation Development Lab (OSADL) den elften »Real Time Linux Workshop« in der Fakultät für Informatik der Technischen Universität Dresden. Möglich wird dies durch die dankenswerte Unterstützung des dortigen Lehrstuhls für Betriebssysteme (Prof. Hermann Härtig). Nach zehn Workshops unter anderem in den USA, Singapur, Spanien oder Frankreich kommt die Veranstaltung in diesem Jahr erstmals nach Deutschland. Dadurch bietet sich Entwicklern im deutschsprachigen Raum die einmalige Gelegenheit, die führenden Köpfe hinter Echtzeit-Linux vor Ort zu treffen und sich über den aktuellen Status von Echtzeit-Linux zu informieren.

- »Community Developers Track«

Die Organisatoren des Workshops weisen stolz darauf hin, dass eine große Anzahl führender Linux-Kernel-Echtzeit-Entwickler aus der ganzen Welt die Einladung nach Dresden angenommen hat. So wird es Vorträge zu vielen wichtigen Aspekten von Echtzeit-Linux im Allgemeinen und zu den PREEMPT_RT-Patches im Speziellen geben. Der »Community Developers Track« findet am 29. September ohne Parallelveranstaltungen statt.

- »Free Papers Session«

Aus den vielen eingesandten Beiträgen hat das Programmkomitee die besten ausgesucht und in acht Sessions »freier Vorträge« am 28. und 30. September eingeteilt.

- »Hands-on Sessions«

Parallel zu den freien Vorträgen stehen zudem »Hands-on Sessions« zu verschiedenen Echtzeitsystemen wie z.B. »XtratuM«, »L4« und »PREEMPT_RT« auf dem Programm. Experten werden erklären und vorführen, wie die jeweiligen Systeme funktionieren, und Unterstützung bei deren Inbetriebnahme geben.

- Podiumsdiskussion

Für den Nachmittag des 30. September ist eine Podiumsdiskussion geplant. Dabei geht es um das Potenzial und die Möglichkeiten, Universitäten und Kernel-Entwickler zusammenzubringen, um auf diese Weise eine weitere Verbesserung des Linux-Kernels und speziell seiner Echtzeiteigenschaften zu erreichen.

Die Kongresssprache ist Englisch – Vortragsprogramm und Online-Anmeldemöglichkeit unter www.osadl.org

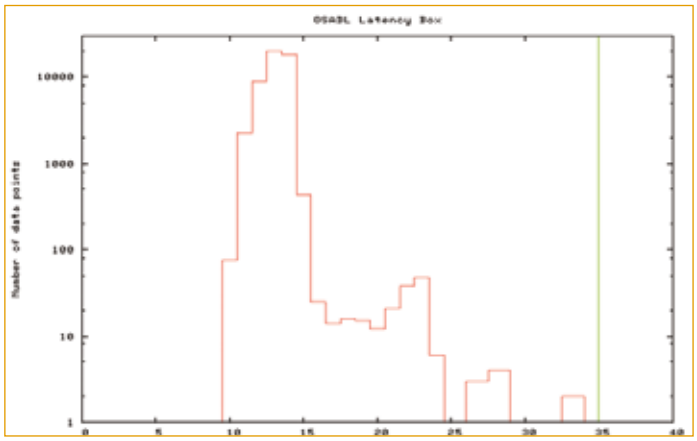


Bild 3: Beispielhafte Darstellung der Ergebnisse eines Latenz-Histogramms, gemessen mit der »OSADL-Latency-Box«

IRQ-off-Latency und Preemption-off-Latency konfiguriert sind. Der Dateiname der Daten der ersten CPU lautet: »/sys/kernel/debug/tracing/latency_hist/preemptirqsoff/CPU0«.

Die Zeitwerte der Wakeup-Histogramme entsprechen der Dauer zwischen der Aufnahme eines zu aktivierenden Prozesses in die Warteschleife und dessen tatsächlichem Ausführungsbeginn. Normalerweise sollten die Messwerte der Histogramme im Verzeichnis »/sys/kernel/debug/tracing/latency_hist/wakeup«, in denen nur Prozesse mit exklusiver höchster Priorität pro CPU erfasst werden, mit den Messergebnissen des Programms »cyclictest« übereinstimmen. Dies ist allerdings dann nicht der Fall, wenn aus irgendeinem Grunde Interrupts länger als normal gesperrt sind, so dass ein Prozess bereits mit Verspätung in die Warteschleife aufgenommen wird. Eine so entstandene Latenz würde von den Wakeup-Histogrammen nicht registriert. Zu diesem Zweck sind aber die Missed-Timer-Offsets-Histogramme vorhanden; denn diese messen das Ausmaß der Verspätung, mit welcher der Wakeup-Vorgang überhaupt erst ausgelöst wird. Die maximale Latenz eines Systems ergibt sich also aus der maximalen Verspätung plus der maximalen Wakeup-Latency. Die Messung dieser beiden Größen wurde so implementiert, dass eine möglichst geringe negative Auswirkung auf die Latenz zu erwarten ist. Dadurch ist es möglich, die Messung über einen sehr langen Zeitraum oder sogar auch während der gesamten Lebensdauer einer Maschine durchzuführen und die Latenzbestimmung – ähnlich wie die Bestimmung der MTBF (Mean Time Between Failure) empirisch und unter den Normalbedingungen einer Maschine zu validieren. Bild 4 zeigt eine derartige kontinuierliche Latenzmessung über 30 Stunden an einem 4-Weg-Prozessor (Intel Core 2 Quad).

Die anderen Histogramme enthalten jeweils die Dauer einzelner Kernel-Ausführungspfade, in denen Preemption, Interrupts oder beide abgeschaltet sind. Form und Werte dieser Histogramme können im Einzelfall wichtige Hinweise liefern, wo Latenzen entstanden sein mögen, wenn ein Prüfling kein zufriedenstellendes Echtzeitverhalten aufweist.

Als Nachteil der Histogramm-Messungen darf nicht unerwähnt bleiben, dass diese natürlich ihrerseits einen gewissen – wenn auch relativ geringen – Beitrag zur Latenz leisten. Daher ist mit einer Verschlechterung des Echtzeitverhaltens um, je nach Prozessortakt, einige Mikrosekunden zu rechnen. Auf der anderen Seite hat diese eingebaute Messung aber den unschätzbaren Vorteil, dass sie nicht nur eine bestimmte Auswahl an künstlichen Messprozessen schafft und misst, sondern jeden einzelnen Vorgang (Scheduling und Abschaltung von Interrupts beziehungsweise Preemption) erfasst, es also keinen Messfehler im Sinne einer zu optimistischen Schätzung gibt.

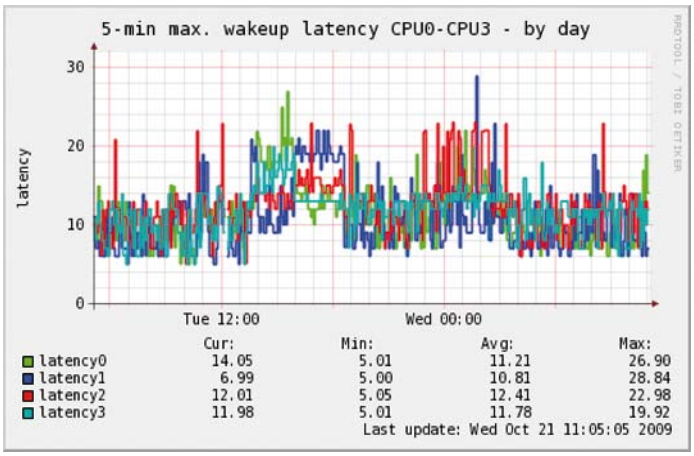


Bild 4: Kontinuierliche Latenzmessung (in µs) über 30 Stunden durch Erfassung sämtlicher Wakeup-Vorgänge von Prozessen, bei denen in dieser Phase kein anderer Prozess mit der gleichen oder einer höheren Priorität auf der jeweiligen CPU ausgeführt wurde.

Ähnlich wie bei der Messung mit der »OSADL-Latency-Box« stehen allerdings auch bei den internen Histogrammen keine Informationen über den Zeitpunkt der gemessenen Latenz und der dabei ausgeführten Kernelfunktionen zur Verfügung. Bei zu hohen gemessenen Latenzwerten muss auch hier im Anschluss eine spezielle Messung mit Funktions-Tracing zur weiteren Diagnose erfolgen.

Vor Beginn einer definitiven Messung sind alle Histogrammzähler auf »0« zu setzen. Für diesen Zweck ist die virtuelle Datei »reset« in den jeweiligen Verzeichnissen vorgesehen. Das Zurücksetzen erfolgt zum Beispiel mit dem in Codebeispiel 11 (siehe Kasten) dargestellten Shell-Skript. Im Prüffeld bietet es sich an, die internen Kernellatenz-Histogramme regelmäßig automatisch auszulesen und eine entsprechende Warnmeldung auszulösen, wenn erhöhte Werte festgestellt werden.

Schließlich ist auch zu erwähnen, dass das Konzept vieler Steuerungsprogramme eine zyklisch ausgeführte Hauptschleife vorsieht, deren Ausführungsdauer naturgemäß nicht länger sein darf als das vorgesehene Zyklusintervall. Entsprechend bietet sich an, bei jedem Durchlauf der Hauptschleife zu prüfen, ob der Durchlauf rechtzeitig gestartet ist, und eine Warnmeldung auszugeben, sollte dies nicht der Fall sein. Aber auch bei dieser Methode schließen sich dann weitere spezielle Messungen – zum Beispiel mit aktiviertem Kernel-Funktions-Tracing – an, um die jeweilige Ursache der zu langen Ausführungsdauer der Hauptschleife zu ermitteln. (cg)

OSADL
Telefon: 0 74 43/13 30 73
www.osadl.org

Linutronix
Telefon: 0 75 56/91 98 91
www.linutronix.de

Pengutronix
Telefon: 0 51 21/20 69 17 0
www.pengutronix.de

HEFT 08 · SEPTEMBER 2009 · € 7,50

www.elektroniknet.de

DESIGN & ELEKTRONIK

KNOW-HOW FÜR ENTWICKLER

EMBEDDED COMPUTING

SONDERPUBLIKATION

Real-Time Linux Workshop 2009
in Dresden vom 28. bis 30. September

open source meets industry

OSADL
Open Source Automation Development Lab eG

Bild: Rexroth

Softwareentwicklung
Linux-Latenzen auf der Spur

Industriecomputer
Neues von PICMG 1.3

Kommunikationstechnik
Ganzheitliche Antennenentwicklung

Displays
Electrowetting-Anzeigen

Modellbasierte Entwicklung
UML für Echtzeitsysteme

Ultra-Low-Power-Designs
Energieeffiziente Mikrocontroller