© NASA, https://images-assets.nasa.gov/image/KSC-20200119-PH-SPX01_0001/KSC-20200119-PH-SPX01_0001~orig.jpg

Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 123

Dimension der Zeit

Linux in Mission Critical Systems einsetzen? Kein Problem: Ein Raspberry Pi mit PREEMPT_RT-Patch zeigt eindrucksvoll die Realzeitfähigkeiten des Linux-Kernels auf.

Eva-Katharina Kunst, Jürgen Quade

Spätestens seitdem die NASA ihre Astronautinnen, Astronauten und Fracht der Crew-Dragon-Kapsel und der Falcon-9-Rakete der Firma SpaceX anvertraut, ist klar: Linux ist *ready for mission critical systems*. Auf Reddit hat sich das SpaceX-Software-Team unter der Rubrik AMA (ask me anything) im Juni 2020 den Fragen Neugieriger gestellt und bereitwillig geantwortet [1]. Die Entwickler berichteten von damals bereits über 30 000 Linux-Knoten im Orbit, die in Starlink-Satelliten verbaut sind, und von über 180 Jahren Linux-Laufzeit im Weltraum. Am Kernel selbst gibt es unabhängig von der Entwicklung eigener Gerätetreiber kaum Veränderungen. Das SpaceX-Team hat lediglich den PREEMPT_RT-Patch auf die Kernel-Quellen losgelassen, um ein deterministischeres Realzeitverhalten zu erreichen. Der Rest liege in der Applikation, heißt es seitens SpaceX.

Schon seit Jahren krempelt unter anderem der deutsche Entwickler Thomas Gleixner erfolgreich den Linux-Kernel in Richtung Realzeit um und erstellt ein Set von Patches, das dem Betriebssystemkern mehr Realzeitverhalten einhaucht: PREEMPT_RT. Zwischenzeitlich hat Chefarchitekt Linus Torvalds große Teile der Patches seinem Quellcodearchiv als fixen Kernel-Bestandteil einverleibt, den Standard-Kernel um neue Realzeitmechanismen erweitert und das Patch-Set damit immer weiter miniaturisiert. Es ist nur eine Frage der Zeit, bis der Kernel die letzten Patches aufgesaugt haben wird.

Die Autoren

Eva-Katharina Kunst ist seit den Anfängen von Linux Fan von Open Source. Jürgen Quade, Professor an der Hochschule Niederrhein, führt auch für Unternehmen Schulungen zu den Themen Treiberprogrammierung und Embedded Linux durch.

Realzeitsysteme zeichnen sich dadurch aus, dass sie neben den funktionalen auch zeitlichen Anforderungen genügen. Sie reagieren auf Ereignisse innerhalb eines klar definierten Zeitfensters 2, weil die Laufzeit von Kernel und Applikation leidlich vorherzusehen und deterministisch ist, sich entsprechend mit einer Oberschranke angeben lässt. Die Größe des Zeitfensters bezeichnen Profis als Reaktionszeit, während alles, was das eigentliche Bearbeiten des Ereignisses verzögert, die Latenzzeit darstellt.

Da die Reaktionszeit von der jeweiligen Aufgabenstellung (Laufzeit der Algorithmen) abhängt, ist die Latenzzeit eine der ausschlaggebenden Größen, um ein Realzeitsystem zu betrachten und zu bewerten. Je kleiner die Latenzzeit auch bei größter Auslastung und unter allen Umständen ausfällt, desto besser ist das System – oder, um es neutral zu sagen: desto mehr Einsatzmöglichkeiten gibt es.

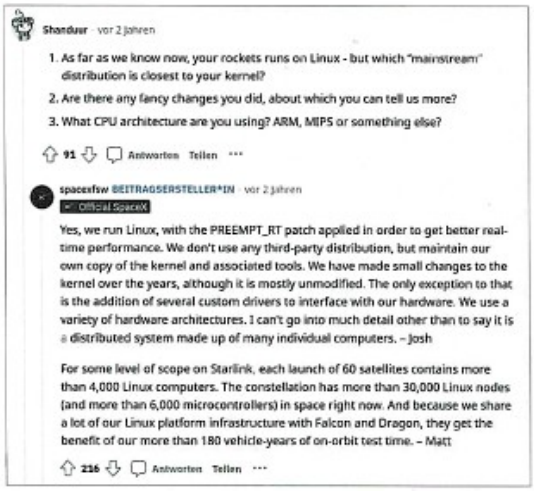
Die Ursachen für Latenzzeiten sind vielfältiger Natur. Ist der Betriebssystemkern mit etwas sehr Wichtigem beschäftigt, lässt er sich währenddessen nicht unterbrechen. In den Anfangstagen von Linux, als das Zeitverhalten lediglich unter dem Gesichtspunkt Durchsatz relevant war und Latenzzeiten für Standardbetriebssysteme keine Rolle spielten, ging man auf Nummer sicher, indem man generell Unterbrechungen unterband. Schließlich gehören Unterbrechungen zu den Hauptübeln für das Fehlverhalten von Softwaresystemen.

Latenzzeitkiller

Die Leistung der Entwicklerinnen und Entwickler beim Erstellen und Pflegen der RT-Patches besteht entsprechend darin, die nicht unterbrechbaren Bereiche zu identifizieren und unterbrechbar (preemptible) umzubauen. Daher stammt auch der Name des Patch-Sets: PREEMPT_RT. Unterbrechbarkeit ist aber nur eine Seite der Medaille und des Patch-Sets. Daneben sind auch Mechanismen wichtig, um beispielsweise gezielt auf die Abarbeitungsreihenfolge von Tasks (das Scheduling) Einfluss zu nehmen.

Das Scheduling auf Basis von Prioritäten ist für ein Realzeitsystem zwingend notwendig und ein Deadline-Scheduling, bei dem die Auswahl der anschließend zu bearbeitenden Task auf Basis der Reaktionszeit stattfindet, wünschenswert. Damit sich Interrupts priorisieren lassen, lagern Realzeitsysteme zeitintensive Teile der Interrupt-Service-Routinen in Kernel-Threads aus. Hier sprechen Expertinnen und Experten von Threaded Interrupts.

Die Prioritätsgeschichte in Kombination mit dem Schutz kritischer Abschnitte wiederum führt zur Prioritätsinversion, der man die Prioritätsvererbung entgegensetzt. Die Tabelle PREEMPT_RT zeigt weitere Techniken, die Linux im Rahmen der PREEMPT_RT-Entwicklung eingebaut hat.

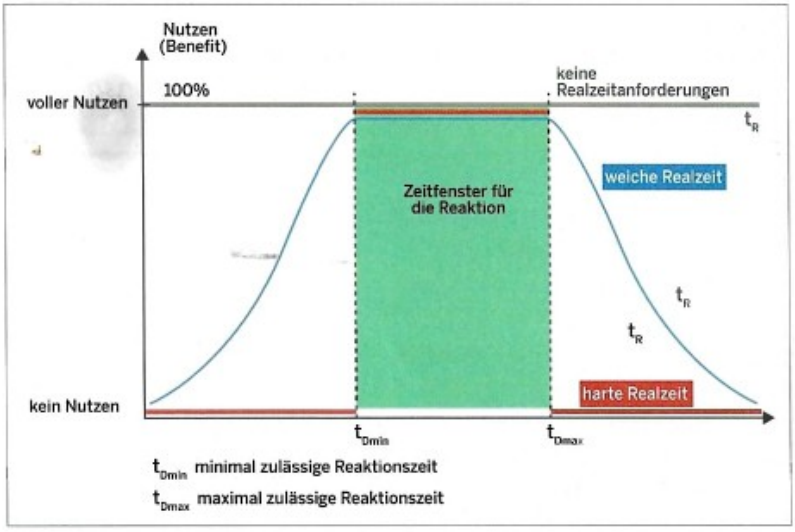


1 SpaceX vertraut ganz auf Linux mit PREEMPT_RT-Patch.

Selbst machen

Genug der Vorrede, hinein in die Praxis. Die Wirkungsweise eines PREEMPT_RT-Linux lässt sich sehr eindrucksvoll mit etwas Handarbeit verdeutlichen. Um unsere Arbeitsmaschine nicht modifizieren zu müssen, wählen wir als Versuchsobjekt einen Raspberry Pi 4, generieren einen (gepatchten) Kernel und installieren sowie booten ihn zusammen mit einigen Testprogrammen. Den zum jeweiligen Kernel passenden PREEMPT_RT-Patch finden Sie ebenso wie bereits gepatchte Kernel-Quellcodes in den Repositories von Kernel.org.

Listing 1 zeigt die Kommandos zum Cross-Kompilieren eines Linux-Kernels für den Raspberry Pi auf einem PC. Hintergrundinfos dazu finden Sie auf den Seiten der Raspberry Pi Foundation. Zunächst installieren Sie die Tools zum Cross-Kompilieren, darauf folgt das Herunterladen des aktuellen, stabilen Linux-Kernels so-



2 Bei harter Realzeit muss die Reaktion innerhalb des Zeitfensters erfolgen.

wie der Patch-Datei, die schließlich den Kernel auf volle Realzeit umbaut.

Aus Geschwindigkeitsgründen empfiehlt es sich, den Raspberry-Pi-Kernel auf dem PC zu erzeugen. Dazu setzen Sie die Umgebungsvariablen `KERNEL`, `ARCH` und `CROSS_COMPILE`. Für das zu erreichende Realzeitverhalten ist die Konfiguration entscheidend, bei der `make menuconfig` den PREEMPT_RT-Patch erst aktiviert. Die Einstellungen können Sie in der Kernel-Konfiguration unter dem Oberpunkt *General Setup* nachlesen. Sofern das nicht ohnehin schon als Vorgabe gesetzt ist, wählen Sie im ersten Schritt den Modus für *expert users* aus.

PREEMPT_RT

Kurzbeschreibung	Realzeit-Technologien
Preemption	Kernel-Code lässt sich jederzeit unterbrechen.
High Resolution Timer	Zeitfunktionen mit hoher Auflösung.
Threaded Interrupts	Auf Applikationsebene priorisierbare Interrupts.
Prioritätsvererbung	Auf Betriebsmittel wartende Tasks vererben ihre (hohe) Priorität an Betriebsmittel nutzende Tasks.
Tickless Betrieb	Der Kernel wird nicht periodisch aktiv, sondern bedarfsgesteuert.
Earliest Deadline First Scheduler	Auswahl zu bearbeitender Tasks auf Basis der Reaktionszeit.
Realzeit-Locks	Elemente zum Schutz kritischer Abschnitte.
Memory Locking	Verhindern des zu Latenzzeiten führenden Auslagerns von Speicherseiten.

Listing 1: Cross-Generierung eines PREEMPT_RT-Kernels

```
### Installation der notwendigen Werkzeuge
$ sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev
### Installation Cross-Compiler
$ sudo apt install crossbuild-essential-armhf
$ mkdir ~/preempt-rt-kernel
$ cd preempt-rt-kernel
### Quellcode herunterladen
$ git clone --depth=1 https://github.com/raspberrypi/linux
$ cd linux
### Patch herunterladen -- ACHTUNG: Auf die richtige Version achten!
$ wget https://cdn.kernel.org/pub/linux/kernel/projects/rt/5.15/
patch-5.15.44-rt46.patch.gz
### Kernel patchen
$ patch patch-5.15.44-rt46.patch.gz
$ patch -p1 <patch-5.15.44-rt46.patch
### Grundkonfiguration für Raspberry Pi 4
$ KERNEL=kernel7l
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2711_defconfig
### Aktivierung des PREEMPT_RT-Patches
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
### General Setup | Configure standard kernel features (expert users) ->
### General Setup | Preemption Model (Fully Preemptible Kernel (Real-Time)) ->
### exit -> exit -> yes
### Cross-Generierung von Kernel, Modulen und Devicetree
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j 4 zImage modules dtbs
```

Preemption-Modelle

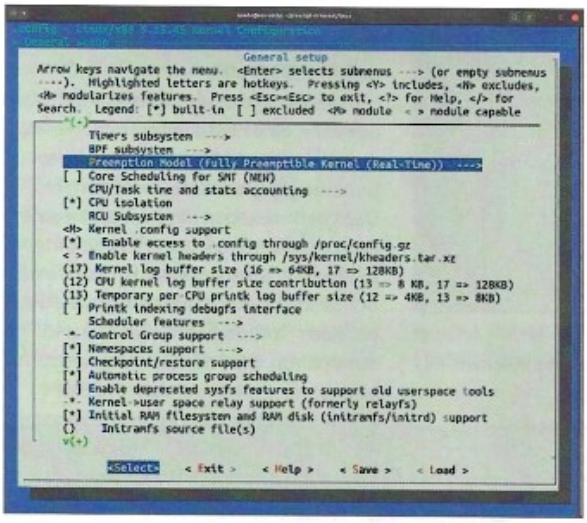
Grundsätzlich stehen vier Preemption-Modelle zur Wahl. Bei `CONFIG_PREEMPT_NONE` handelt es sich um das traditionelle Verhalten eines Unix-Kernels. Der Kernel-Code (Interrupt-Service-Routinen, Systemaufrufe) wird nicht unterbrochen. Diese Konfiguration garantiert durch wenige Kontextwechsel den bestmöglichen Durchsatz sowie eine gute Ausnutzung von CPU und Caches. Diese Einstellung passt am besten für Server.

Bei Einsatz von `CONFIG_PREEMPT_VOLUNTARY` lässt sich Kernel-Code an definierten Stellen unterbrechen (`might_sleep()`), an denen sich der Scheduler zusätzlich aktivieren lässt. Das beschleunigt die Reaktion auf Ereignisse. Diese Anpassung ist für einen normalen Desktop-Rechner vorgesehen.

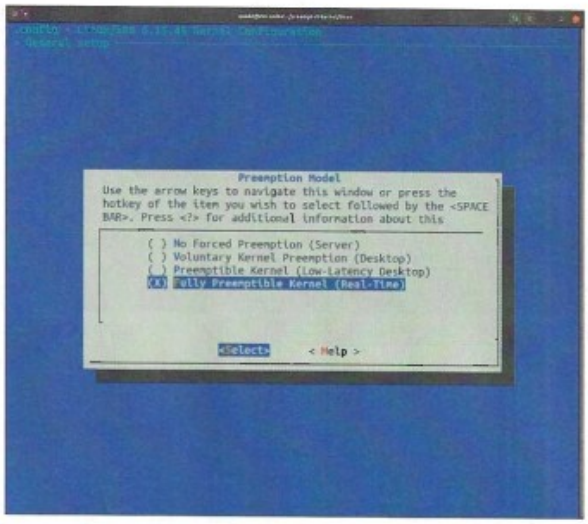
Verwenden Sie `CONFIG_PREEMPT`, lassen sich die meisten Stellen im Kernel-Code unterbrechen. Beim Auftreten eines Ereignisses (Interrupts) kann der Scheduler direkt eingreifen und muss nicht auf das Ende einer gerade aktiven Kernel-Codesequenz warten. Wer auf seinem Desktop schnelle Reaktionen wünscht, nutzt diese Einstellung.

Mit `CONFIG_PREEMPT_RT` lässt sich jeglicher Kernel-Code unterbrechen und typischerweise nicht schlafende Spinlocks können plötzlich schlafen. Interrupt-Service-Routinen sind als Threaded Interrupts realisiert und damit priorisiert. Die Konfiguration kommt bei Realzeitsystemen mit harten Zeitanforderungen zum Einsatz.

In unserem Fall setzen Sie das Preemption-Modell 3 auf *Real-Time* 4. Beim Verlassen des Konfigurationsprogramms werden die Änderungen abgespeichert.



3 Unterbrechungsmodell vor der Kernel-Generierung.



4 Für ein Realzeitsystem wählen Sie hier die vierte Option.

Kurze Weile

Mit der Generierung des Kernels, der Module und des Devicetrees ist eine Host-Maschine durchaus einmal ein gutes Stündchen beschäftigt. Die Installation des fertig generierten Kernels veranschaulicht Listing 2. Dazu stecken Sie die SD-Karte mit dem darauf bereits installierten Pi OS in den Rechner. Per Lsblk identifizieren Sie die Gerätedateien, über die Sie auf die beiden Partitionen der SD-Karte zugreifen. Anders als in der Anleitung angegeben, waren das bei uns im Test beispielsweise `/dev/mmcblk0p1` und `/dev/mmcblk0p2`.

Möglicherweise hat Ihr Linux-System die Partitionen auf der SD-Karte automatisch gemountet, was Sie an der Angabe eines Verzeichnisses wie `/media/quade/boot/` auf der rechten Seite der Ausgabe des Kommandos `lsblk` erkennen. In diesem Fall können Sie entweder diese Verzeichnisse für die Installation verwenden oder hängen die Partitionen aus und wie in Listing 2 wieder ein.

Legen Sie dazu zwei Verzeichnisse an, auf die Sie die beiden Partitionen mounten. Installieren Sie schließlich die Kernel-Module, die Devicetree-Overlays und den Haupt-Devicetree. Bevor Sie den Realzeit-Kernel Marke Eigenbau installieren, sollten Sie ein Backup des aktuellen Kernels erstellen. Nun hängen Sie die beiden Partitionen aus. Die SD-Karte stecken Sie zurück in den Raspberry Pi und versorgen ihn mit Strom, woraufhin der Mini-Rechner hochfährt.

Sein oder nicht sein

Nach dem Booten und Einloggen stellen Sie anhand zweier Kriterien fest, ob der PREEMPT_RT-Patch wirkt. Ein auf der Konsole eingegebenes `uname -a` sollte PREEMPT_RT ausweisen. Außerdem muss die Datei `/sys/kernelrealtime` existieren und eine `1` enthalten 5.

Um den eigentlichen Effekt des Patches zu testen, empfiehlt sich die Test-Suite `rt-tests`. Das Paket gelangt per `git clone` auf den RasPi (Listing 3) und lässt sich nach dem Wechsel ins neue Verzeichnis `rt-tests/` per Make generieren. Aus den zur Verfügung stehenden Testprogramme wählen Sie `cyclictest` und hackbench aus. Ersteres führt Messungen zur Latenzzeit durch, Letzteres bringt den Raspberry Pi gehörig ins Schwitzen und provoziert Worst-Case-Situationen.

Um die Latenzzeit schließlich zu bestimmen, legt sich Cyclictest per `clock_nanosleep()` für eine definierte Zeit schlafen. Sobald Sie die Task wieder aufwecken, liest Cyclictest die aktuelle Zeit und berechnet anschließend durch Differenzbildung zur erwarteten Weckzeit die Abweichung und auf diese Weise schließlich die Latenz. Das Ganze wird in einer Schleife wiederholt ausgeführt und statistisch ausgewertet.

```
Linux raspberrypi 5.15.45-rt46-v7l+ #3 SMP PREEMPT_RT Sat Jun 11 18:16:11
CEST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 24 10:07:19 2022 from 192.168.16.63
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.15.45-rt46-v7l+ #3 SMP PREEMPT_RT Sat Jun 11 18:16:11
CEST 2022 armv7l GNU/Linux
pi@raspberrypi:~ $ cat /sys/kernel/realtime
1
pi@raspberrypi:~ $
```

5 Erfolg: Linux ist bereit für die Realzeitverarbeitung.

```
root@raspberrypi:/home/pi/rt-tests# ./hackbench -f 10 -T 5 -l -1  &
[1] 9354
root@raspberrypi:/home/pi/rt-tests# Running in threaded mode with 10 groups using 20 fi
le descriptors each (== 200 tasks)
Each sender will pass -1 messages of 100 bytes

root@raspberrypi:/home/pi/rt-tests# ./cyclictest --smp
WARN: stat /dev/cpu_dma_latency failed: No such file or directory
policy: other/other: loadavg: 6.42 6.96 5.00 3/371 9568

T: 0 ( 9556) P: 0 I:1000 C: 683090 Min:     14 Act:   72 Avg:  121 Max:   11574
T: 1 ( 9557) P: 0 I:1500 C: 461997 Min:     15 Act:   73 Avg:  121 Max:   10410
T: 2 ( 9558) P: 0 I:2000 C: 348537 Min:     16 Act:   74 Avg:  126 Max:   10696
T: 3 ( 9559) P: 0 I:2500 C: 279767 Min:     16 Act:   73 Avg:  126 Max:    8526
```

6 **Auch ohne Realzeitprogrammierung lassen sich passable Latenzzeiten erreichen.**

Cyclictest gibt die minimale und durchschnittliche, vor allem aber die bei den Durchläufen aufgetretene maximale Latenz in Mikrosekunden aus. Tritt keine hohe Latenz auf, bedeutet das jedoch nicht, dass es sie nicht geben kann. Je länger die Messung dauert und je unterschiedlicher dabei die Lastsituationen für den RasPi ausfallen, desto wahrscheinlicher ist es, den Worst Case erwischt zu haben. Für eine erste Bewertung des PREEMPT_RT-Patches reicht uns das Ergebnis jedoch aus.

Es sei daran erinnert, dass eine Software nur dann eine niedrige Latenz aufweist, wenn man sie unter Realzeitgesichtspunkten programmiert und betreibt. Dazu gehört neben dem Vergeben einer Realzeitpriorität das zuverlässige Unterbinden von Speicherfehlern. Cyclictest bringt solche Mechanismen mit, die Sie per Kommandozeilenoption ein- und ausschalten. Die Prioritätswahl wiegt besonders schwer. Mit dem PREEMPT_RT-Patch sind fast alle Interrupt-Service-Routinen als Interrupt-Threads ausgeführt und damit priorisiert. Wenn Interrupts nicht das Testprogramm selbst ausbremsen sollen, müssen Sie die Priorität entsprechend hoch setzen.

Listing 2: PREEMPT_RT-Kernel auf SD-Karte installieren

```
cd ~/preempt-rt-kernel
lsblk
### Gerätedatei evaluieren. Sind die Partitionen der SD-Karte
eingehängt,
### werden sie mit den zwei nachfolgenden Kommandos wieder ausgehängt:
$ sudo umount /dev/mmcblk0p1
$ sudo umount /dev/mmcblk0p2
### Partitionen einhängen
$ mkdir mnt
$ mkdir mnt/fat32
$ mkdir mnt/ext4
$ sudo mount /dev/mmcblk0p1 mnt/fat32
$ sudo mount /dev/mmcblk0p2 mnt/ext4
### Kernel-Module installieren
cd linux
$ KERNEL=kernel7l
$ sudo env PATH=$PATH make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
INSTALL_MOD_PATH=mnt/ext4 modules_install
### Kernel und Devicetree installieren
$ cd ~/preempt-rt-kernel
$ sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
$ sudo cp linux/arch/arm/boot/zImage mnt/fat32/$KERNEL.img
$ sudo cp linux/arch/arm/boot/dts/*.dtb mnt/fat32/
$ sudo cp linux/arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
$ sudo cp linux/arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
$ sudo umount mnt/fat32
$ sudo umount mnt/ext4
$ sync
```

Erkenntnisgewinn

Abbildung **6** demonstriert den Aufruf von Cyclictest als normale Applikation (ohne Priorität, Memory Prefault oder sonstige Realzeittechniken). Die Latenzzeiten liegen recht passabel im zweistelligen Millisekundenbereich. Unter Last fallen sie niedriger aus als ohne. Was zunächst seltsam anmutet, hat mit häufiger auftretenden Unterbrechungen zu tun und dem damit verbundenen, zeitnäheren Abarbeiten von Zeitaufträgen und des Schedulings.

Starten Sie Cyclictest mit einigen Realzeittricks, liegen die Latenzen im zweistelligen Bereich, diesmal aber im Mikrostatt wie vorher im Millisekundenbereich **7** – beeindruckend für den RaspPi und sein ausgewachsenes Betriebssystem. Das Open Source Automation Development Lab (OSADL) nimmt Langzeitmessungen der Latenz verschiedener Realzeit-Linuxe vor, bestätigt unser Ergebnis

Dateien zum Artikel herunterladen unter
www.lm-online.de/dl/47349

Weitere Infos und interessante Links
www.lm-online.de/qr/47349

und liefert Plots dazu. Der Raspberry Pi 4 befindet sich in deren Testlabor in Rack 4, Slot 1. Das Team erstellt die Plots direkt mithilfe von Cyclictest und Gnuplot. Die Häufigkeitsverteilung der Latenzzeiten unserer Messungen auf den einzelnen CPU-Kernen zeigt Abbildung 8.

Neugierige können noch ein wenig mit Kommandozeilenoptionen und Prioritäten spielen. Reduzieren Sie etwa die Priorität von 80 auf 10, bekommen wir auf Core 0 der RasPi-CPU eine dreistellige Latenzzeit – ein signifikantes Zeichen für Threaded Interrupts. Höher priorisierte Interrupt-Threads unterbrechen die auf CPU-Kern 0 laufenden Tasks.

Wenige Tests liefern bereits spannende Erkenntnisse. Zunächst ermöglicht Linux mit dem PREEMPT_RT-Patch tatsächlich erfreulich niedrige Latenzzeiten. Der Patch allein reicht aber nicht aus – man muss den Werkzeugkoffer der Realzeitprogrammierung öffnen und auf System und Applikation anwenden. Was die Toolbox hergibt, verrät der Reddit-Thread der SpaceX-Entwickler. (*csi*)

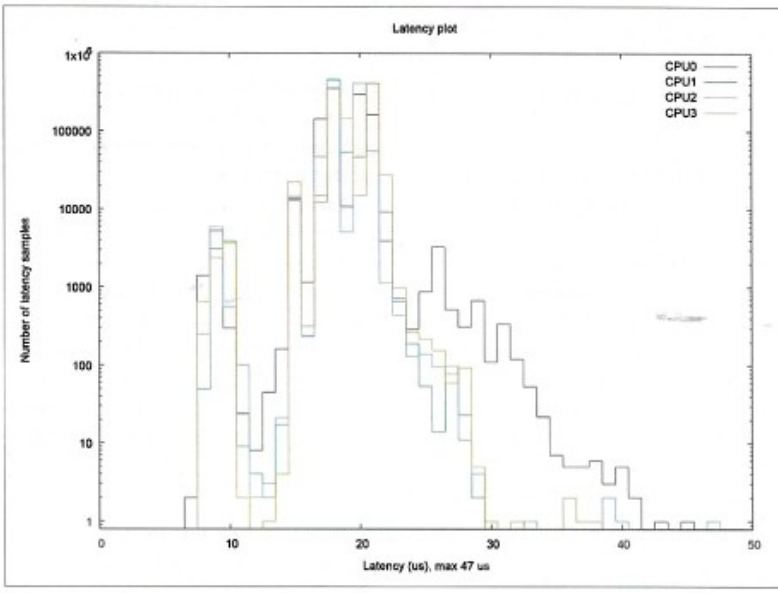
```
                    format: --policy=fifo(default) or --policy=rr
root@raspberrypi:/home/pi/rt-tests# ./cyclictest --mlockall --sm --priority=90 --interv
al=200 --distance=0
WARN: stat /dev/cpu_dma_latency failed: No such file or directory
policy: fifo: loadavg: 0.82 1.10 0.91 1/171 8714

T: 0 ( 8640) P:90 I:200 C:3785062 Min:      6 Act:   18 Avg:   18 Max:      46
T: 1 ( 8641) P:90 I:200 C:3784818 Min:      7 Act:   18 Avg:   16 Max:      57
T: 2 ( 8642) P:90 I:200 C:3784690 Min:      8 Act:   17 Avg:   16 Max:      49
T: 3 ( 8643) P:90 I:200 C:3784558 Min:      7 Act:   19 Avg:   17 Max:      61
```

7 Der PREEMPT_RT-Patch verkürzt die Latenzzeiten.

Listing 3: Testprogramme installieren

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/
rt-tests.git
$ cd rt-tests
$ make
```



8 Die Häufigkeitsverteilung der Latenzzeitmessung.