hochauflösende Timer in Linux — hrtimer

IT-Klinger — Andreas Klinger
ak@it-klinger.de

11. März 2012

Vor der Integration des hrtimer-Frameworks in den Standard-Linux-Kernel gab es nur die Option, basierend auf dem starren „Timer-Wheel“ zeitlich gesteuerte Vorgänge auszulösen und Zeitmessungen vorzunehmen. Die zeitliche Granularität im Millisekundenbereich und die eingeschränkten Programmiermöglichkeiten sind für aktuelle Anwendungsprobleme gerade im Echtzeitbereich bei weitem nicht ausreichend.
Durch die neuen hrtimer hat sich dies grundlegend geändert. Jetzt hat der Entwickler neben dem alten System auch die Option mit sogenannten „hochauflösenden Timern“ (High-Resolution-Timer) zu arbeiten.
Diese sind im Gegensatz zum alten System eventgesteuert und damit nicht an feste Zeitraster gebunden. Darüber hinaus bieten sie noch eine ganze Reihe weiterer Verbesserungen für Linux.
Obwohl sich das hrtimer-Framework schon seit Kernel 2.6.21 im Mainstream befindet, wird es nur von wenigen Anwendungen und Treibern aktiv genutzt. Eine Ursache mag sein, dass es noch kaum für Entwickler geeignete Dokumentationen gibt.
Wie das hrtimer-Framework aufgebaut ist und wie man es sich direkt im Treiber oder auch in der Userspace-Anwendung zunutze machen kann, beschreibt dieser Artikel.

1 Aufbau des hrtimer-Frameworks

Für jede CPU existieren zwei nach dem Zeitpunkt sortierte Red-Black-Bäume (rb-Trees). Ein Baum bezieht sich auf die Kalenderzeit (`CLOCK_REALTIME`) und ein anderer auf die monoton gerechnete Zeit seit System-Boot (`CLOCK_MONOTONIC`). Jeder Knoten im Baum repräsentiert ein Ereignis in der Zukunft. Dabei werden im Wesentlichen die Informationen Ablaufzeitpunkt, Callback-Funktion, Aufruf-Kontext und die Zeitbasis in Form einer Struktur (`struct hrtimer`) gekapselt.

Für was werden nun die einsortierten Informationen benötigt?
Der Ablaufzeitpunkt ist der Zeitpunkt zu dem ein Timer-Interrupt das System

unterbrechen soll. Die Interrupt-Behandlung im hrtimer-Framework entscheidet dann, wie eine Callback-Funktion aufgerufen werden soll. Diese Callback-Funktion kann frei erstellt werden und ist die Reaktion auf das Eintreten eines zeitlichen Ereignisses. Für viele kernelinterne Funktionen existieren bereits solche Callback-Funktionen, beispielsweise für Scheduling-Ereignisse, Wartezeiten, und so weiter. Genauso können an dieser Stelle auch eigene Funktionen entworfen und eingehängt werden.

Mit dem Aufruf-Kontext wird angegeben, wie die Callback-Funktion auszuführen ist. Das kann zum einen direkt durch den Interrupt-Kontext des Timer-Interrupts erfolgen. Damit erreicht man die geringstmögliche Latenzzeit. Zum anderen kann dies indirekt mittels eines SoftIRQ's, welcher durch den Timer-Interrupt ablaufbereit gemacht wird, erfolgen.

Neben einmaligen Ereignissen gibt es auch wiederholte und zyklische. Wiederholte Ereignisse werden in der Callback-Funktion gleich wieder in den rb-Tree eingehängt; mit dem nächsten Ablaufzeitpunkt. Durch diesen Mechanismus kann eine konstante Anzahl an Abtastzeitpunkten pro Zeiteinheit realisiert werden. Bei der Digitalisierung von Audio-Signalen ist dies essentiell notwendig.

2 Zeitliche Abfolge

Ein hrtimer soll zu einem bestimmten Zeitpunkt einen Interrupt liefern und dieser wiederum eine benutzerdefinierte Callback-Funktion aufrufen. Wie ist die Abfolge dabei zu gestalten?
Zuerst einmal wird ein neues Ereignis (`struct hrtimer`) erstellt und mit den Informationen Ablaufzeitpunkt, Callback-Funktion und Zeitbasis gefüllt. Dieses Ereignis wird dem hrtimer-Framework übergeben, welches es dann in den rb-Tree entsprechend dem Ablaufzeitpunkt einsortiert. Das hrtimer-Framework prüft, welches das nächstliegende Ereignis ist und programmiert den HW-Timer entsprechend auf diesen Zeitpunkt.

Wenn der Zeitpunkt eintritt und der Timer interrupted, wird das Ereignis aus dem Baum entfernt. Es wird das zeitlich nächstliegende Ereignis aus dem rb-Tree geholt und der HW-Timer auf diesen Zeitpunkt programmiert.

Abbildung 1 zeigt die Vorgänge bei einer direkt im Interrupt-Kontext aufgerufenen Callback-Funktion in den Funktionsbereichen der timer-Hardware, dem Interrupt-

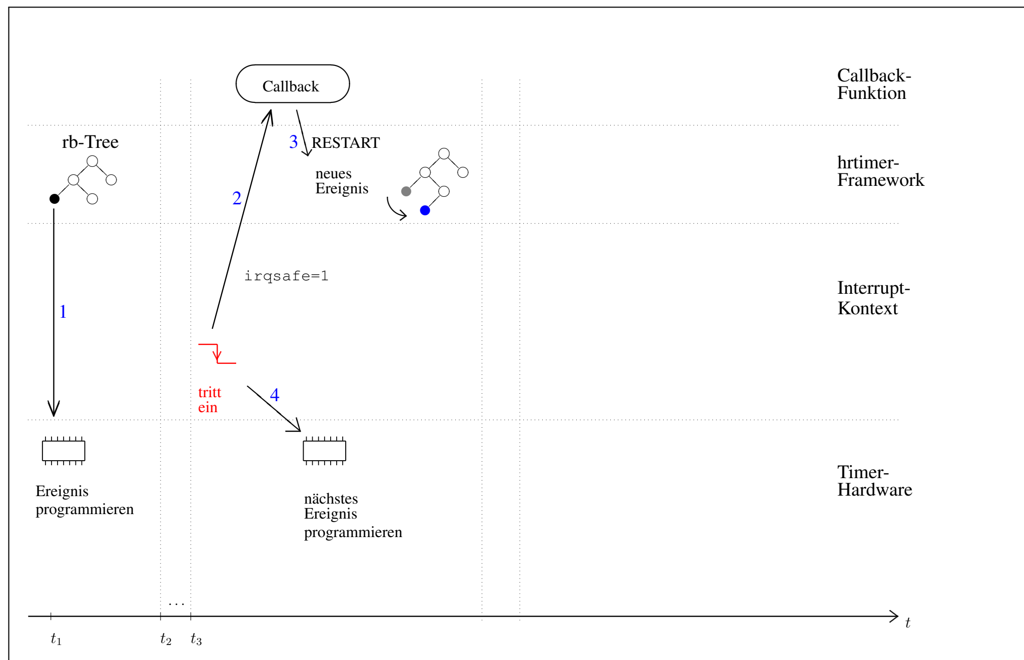

Abbildung 1: Callback-Funktion wird direkt vom Timer-Interrupt aufgerufen

Kontext im Kernel (ISR), dem hrtimer-Framework sowie der eigentlichen Callback-Funktion im Kernel.
Aus dem rb-Tree wird das nächstliegende Ereignis geholt und die Timer-Hardware auf den betreffenden Zeitpunkt programmiert (Schritt 1). Wenn der Interrupt auftritt, wird unmittelbar die Callback-Funktion aufgerufen (Schritt 2). Falls die Callback-Funktion entscheidet, dass der Timer erneut aufgerufen werden soll (wiederholtes Ereignis) wird ein neues Ereignis in den rb-Tree eingehängt (Schritt 3). Nach der Abarbeitung des Interrupts wird das nächste Ereignis aus dem rb-Tree geholt und die timer-Hardware erneut programmiert (Schritt 4).
Da in diesem Szenario alle Vergänge im Interrupt-Kontext stattfinden geht alles auch sehr zeitnah vor sich, sofern die Callback-Funktion effizient programmiert wurde und nicht zu viele derartige Ereignisse das System belasten.

In der Abbildung 2 wurde das soeben gezeigte Verhalten um die Mechanismen einer indirekt aufgerufenen Callback-Funktion erweitert. Im Schritt 2 wird nicht mehr direkt die Callback-Funktion aufgerufen, sondern das anstehende Ereignis in die Warteliste (Pending-List) eingetragen und ein SoftIRQ namens „HRTIMER“ geraised. So bald der Interrupt-Kontext verlassen wird und keine weiteren SoftIRQ’s bevorzugt rechnen dürfen wird der HRTIMER-SoftIRQ loslaufen, die Ereignisse aus der Warteliste abarbeiten und die korrespondierenden Callback-Funktionen aufrufen (Schritt 4).

Indirekt aufgerufene Callback-Funktion verhalten sich kooperativer gegenüber dem Rest des Linux-Systems und sollten bevorzugt verwendet werden. Lediglich in echtzeitrelevanten Treibern oder zeitkritischen Systemteilen sollte auf die direkt aufgerufenen Callbacks zurückgegriffen werden.

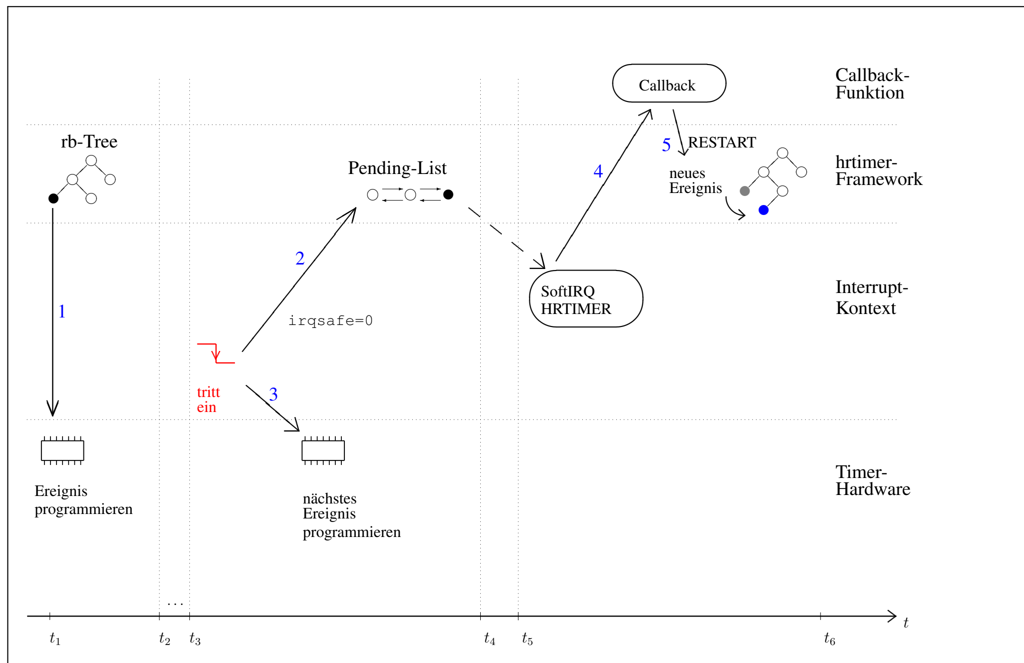

Abbildung 2: HRTIMER-SoftIRQ ruft indirekt die Callback-Funktion auf

Die Interrupt-Behandlung ist damit abgeschlossen. Sobald der SoftIRQ Rechenzeit zugewiesen bekommt, ruft er die Ereignisse aus der Warteliste nacheinander auf, bis die Liste leer ist. Jedes ausgeführte Ereignis wird aus der Liste entfernt.

3 Verwendung in Kernel-Modulen

In einem Kernel-Modul können die hrtimer sehr einfach und intuitiv verwendet werden. Dazu sind die entsprechenden Funktionen aus der hrtimer-API zu verwenden. Diese sind in der Include-Datei `hrtimer.h` deklariert.

Im oberen Teil der Abbildung 3 ist ein Auszug aus dem Userspace-Prozess zu sehen, welcher die Treiberfunktionen `open()` und `ioctl()` aufruft. Anmerkung: Die Abbildungen sind qualitativ zu verstehen; Parameter und Rückgabewerte der Funktionen wurden nicht dargestellt. Auf der linken Seite ist der

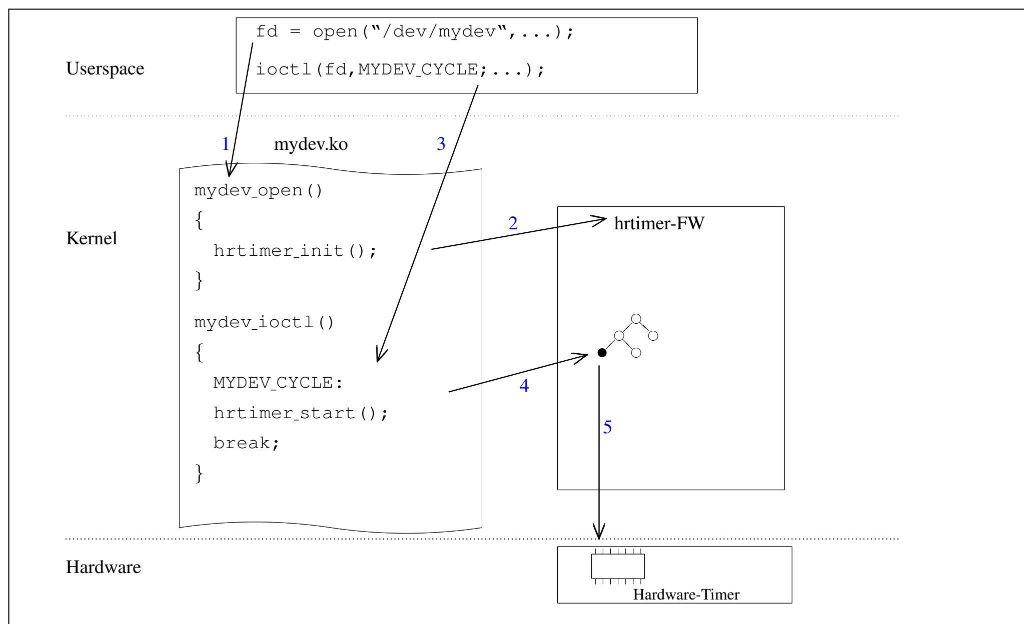

Abbildung 3: Programmierung eines hrtimer-Ereignises durch einen Kernel-Treiber

Kernel-Treiber mit dem Namen `mydev.ko` zu sehen. Er verwendet die Funktionen des hrtimer-Frameworks auf der rechten Seite. Im unteren Bereich ist die Timer-Hardware des Systems dargestellt.
In der Treiberfunktion `mydev_open()` wird der im Beispiel verwendete hrtimer mit der Funktion `hrtimer_init()` initialisiert. Damit ist die Datenstruktur vorbereitet, jedoch noch kein Ereignis aktiviert. In der Funktion `mydev_ioctl()` wird im Zweig `MYDEV_CYCLE` ein Ereignis erstellt und in den rb-Tree mit der Funktion `hrtimer_start()` eingefügt. Jetzt wird das Ereignis aktiv vom hrtimer-Framework verwaltet.

In der Abbildung 4 sind die Vorgänge beim Eintreten des Ereignisses zu sehen. Im Zweig `MYDEV_PERIOD` wird der Userspace-Task schlafen gelegt. Dieser wartet an der Funktion `wait_event()` so lange, bis er durch die Funktion `wake_up()` aufgeweckt wird.

Der Timer interrupted und macht den SoftIRQ ablaufbereit; er wird „geraised“. Sobald der HRTIMER-SoftIRQ Rechenzeit bekommt, ruft er die Callback-Funktion des Treibers auf. Diese wiederum weckt mit dem Aufruf von `wake_up()` den schlafenden Prozess im Zweig `MYDEV_PERIOD` auf. Der Prozess bekommt Rechenzeit, kehrt in der Userspace zurück und kann nun die zum Zeitpunkt erforderliche Funktionalität ausführen.

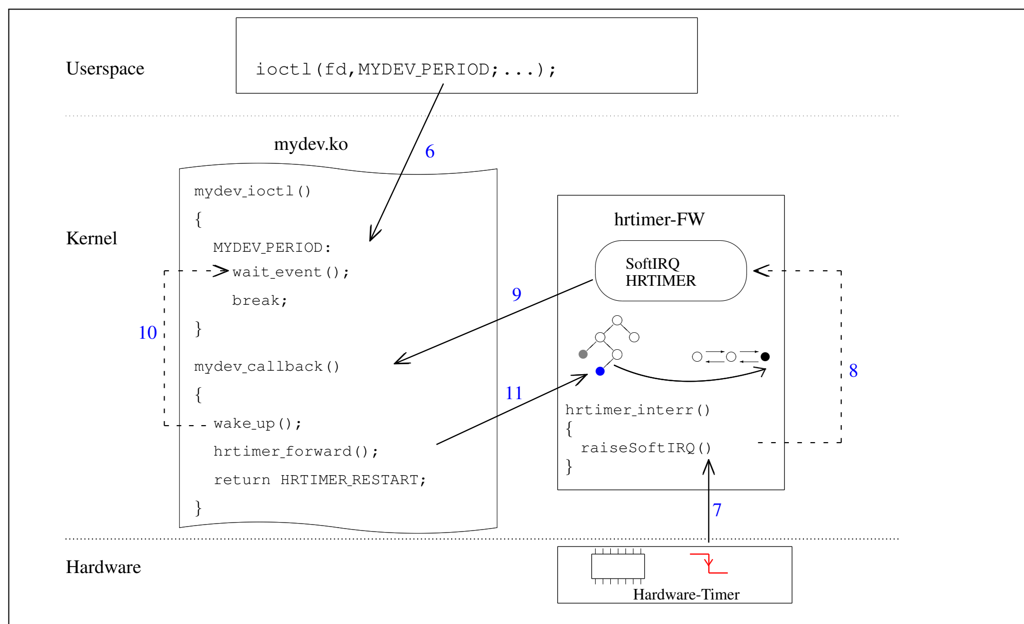

Abbildung 4: Timer-Interrupt weckt wartenden Userspace-Prozess auf

Dieses Beispiel zeigt die für ein Anwendungssystem typische Aufruf-Kette bei der Verwendung des hrtimer-Frameworks unter Zuhilfenahme eines Kernel-Treibers.

4 Verwendung in Systemfunktionen

Im Linux-System-Kern sind nicht alle Funktionen der Systemschnittstelle auf das neue hrtimer-Framework umgestellt. Dadurch ist die Auswahl an hrtimer-basierender Funktionalität im Userspace begrenzt. Die nachfolgend vorgestellten Posix-konformen Funktionen basieren auf dem hrtimer-Framework.

Die Funktion `nanosleep()` legt einen Task für eine bestimmte Zeit schlafen. Der einstellbare Zeitbereich bewegt sich dabei von ca. 5 μs bis zu 2^{32} Sekunden.

Mit der Intervall-Timer-Funktion `setitimer()` kann ein Prozess Signale erhalten, wenn jeweils ein bestimmtes Zeitintervall abgelaufen ist. Möglich ist jeweils ein Timer pro Zeitdomäne (Kalenderzeit, CPU-Zeit), was eine deutliche Einschränkung in der Anwendung darstellt.

Die Universalfunktionen für die Verwendung von Timern im Userspace sind

`timer_create` und `timer_settime`. Mit ihnen lassen sich praktisch beliebig viele Timer einrichten. Die Benachrichtigung erfolgt wahlweise über ein Signal oder auch über einen dann vom System automatisch gestarteten Timer-Thread. Damit können Intervall-Timer oder auch One-Shot-Timer realisiert werden, gestützt durch das leistungsfähige hrtimer-Framework des Kernels.

Detaillierte Beschreibungen zu den genannten Systemfunktionen finden sich in den Manpages.