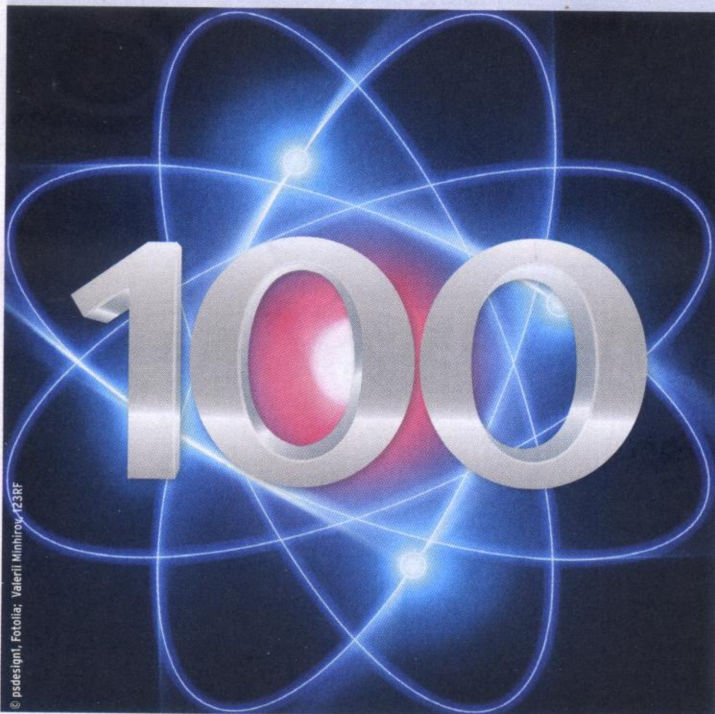
Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 100

Kern-Technik

100. Folge – im Fernsehen erreichen gemeinhin nur Heimatserien wie „Der Bergdoktor" dieses Jubiläum. Dass einer (seit 2005) zweimonatlichen Serie in einer Zeitschrift dies gelingt, ist selten. Die „Kern-Technik" geht auf Zeitreise in eigener Sache. Eva-Katharina Kunst, Jürgen Quade



© psdesignl, Fotolia; Valerii Minhirov, 123RF

Europa ächzt unter einer Hitzewelle mit 40 Grad und mehr, fast genauso heiß ist der Kernel 2.5.40, in Mexiko rollt der letzte VW Käfer vom Band, der deutsche Bundespräsident heißt Johannes Rau und die Band Modern Talking löst sich auf – es ist Hochsommer des Jahrs 2003 und im Linux-Magazin 8 erscheint die erste „Kern-Technik" (**Abbildung 1**).

Jetzt, 15 Jahre später, ist der Kernel 5.0 zum Greifen nahe und die 2.5er Serie verstaubt in den Archiven. Im Rückblick gelten vielen Version 2.4 ohnehin als erstes professionell einsetzbares Linux und die 2.5 als Vorstufe des Dauerbrenners 2.6.

Die Autoren

Eva-Katharina Kunst ist seit den Anfängen von Linux Fan von Open Source. Jürgen Quade ist Professor an der Hochschule Niederrhein. Ihr gemeinsames Buch „Linux-Treiber entwickeln" ist Ende 2015 in vierter Auflage erschienen.

Seid umschlungen, Milliarden

Wie die Informationstechnik als Ganzes hat sich auch Linux in den 27 Jahren seiner Existenz gewandelt – besonders intensiv in den letzten 15 Jahren. Als Betriebssystemkern für eine dedizierte Singlecore-Hardware entstanden, ist Linux heute das plattformunabhängige Betriebssystem für Multicore-Prozessoren auf Milliarden Geräten. Zudem überrascht das freie System mit Realzeit-Eigenschaften, die man ihm angesichts seiner klassischen Architektur nicht zutrauen würde.

Neben der Echtzeit hat Linux schon weitere dicke Bretter gebohrt, zum Beispiel beim Schutz kritischer Abschnitte, bei der Hardware und beim Thema IT-Sicherheit (**Tabelle 1**). Doch wie die Informatik selbst ist auch Linux weit davon entfernt, ausentwickelt zu sein. Für Linus Torvalds und die Kernelcommunity lautet also der Dauerauftrag: weitermachen!

Wettrennen mit ungewissem Ausgang

Der Bereich Schutz kritischer Abschnitte ist wohl das herausragende Beispiel für den noch jugendlichen Zustand der Informatik. Ständig finden Fachleute neue, bis dato unbekannte kritische Abschnitte. Bereits die Kern-Technik 2 hatte gezeigt, wie Gerätetreiber Rechenprozesse professionell mit dem Makro »wait_event_interruptible()« schlafen legen, ohne in eine Deadlock-Falle zu geraten.

Was heute ein alter Hut ist, war damals neu. Schließlich hatten die Programmierer bis dahin das Schlafenlegen simpel durch Modifikation einer Variablen realisiert: »current- > state = TASK_INTERRUPTIBLE«. Dass dabei ein Aufwecken vor dem eigentlichen Schlafenlegen möglich ist, hatten viele einfach noch nicht auf dem Schirm (**Abbildung 2**).

15 Jahre später ploppen weiterhin ständig neue Race Conditions auf. Auf unrühmliche Weise populär geworden sind die aktuellen Beispiel Meltdown und Spectre, welche mühelos die Kern-Techniken 97 und 98 füllten. Für diese Sicherheitslücken gibt es bis heute keine rundum zufriedenstellende Lösung.

Das verdeutlichten sogar schon Intels Lizenzbedingungen zum angebotenen CPU-Microcode-Update: Benchmarking verboten! Nach der Kritik von Bruce

100 Kern-Techniken als E-Book

Wer alle gedruckten Linux-Magazine seit der Ausgabe 08/03 zur Hand hat oder zumindest die Jahres-DVDs ab dem Jahrgang 2003, ist nicht nur treue(r) Leserin oder Leser, sondern nun auch im Besitz der 100 Kern-Technik-Serienteile. An alle anderen hat die Linux-Magazin-Redaktion gedacht, als sie anlässlich der 100. Ausgabe ein 500-seitiges E-Book-Bundle mit allen Kern-Technik-Folgen geschnürt und für knapp 20 Euro als Download unter [http://www.linux-magazin.de/downloads/100-ausgaben-kern-technik/] bereitgelegt hat. Wohl bekomm's!

Perens **[1]** hat Intel diesbezüglich aber einen Rückzieher gemacht.

CPU als Hütchenspielerin

Eine schon länger bekannte Race Condition mit direktem Hardware-Bezug betrifft das Reordering von CPU und Compiler. Diese erlauben es sich nämlich, die programmierte Befehlsreihenfolge umzusortieren. Hat der Programmierer A-B-C programmiert, kann sich die CPU entscheiden, die Befehle in der Reihenfolge B-A-C abzuarbeiten. Natürlich macht sie das nicht vollkommen willkürlich. Die CPU reordert nur, wenn sie glaubt, dadurch am Ergebnis nichts zu verändern, dabei aber performanter beziehungsweise effizienter zu sein.

Leider gibt es diverse Situationen, in denen die CPU die Lage falsch einschätzt, und es ist am Programmierer, durch das Setzen so genannter Memory Barriers (Kern-Technik 5) die geplante Abarbeitungsreihenfolge zu erzwingen. Unglücklicherweise gibt es ausreichend viele Entwickler, die noch nie etwas von der Problematik und von Memory Barriers gehört haben. Daher haben die Kernelentwickler beim Aufkommen von Raspberry Pi & Co. und in aktuellen Versionen Zugriffsfunktionen auf Hardware definiert, die intern Memory Barriers einsetzen (zum Beispiel »gpio_set_value()«, »gpio_get_value()«, Kern-Technik 70).

Big Abschalte

Ursache für Race Conditions ist zumeist Parallelität. Und die ist allgegenwärtig, seit die Ära der glühenden CPUs und aufwändigen Lüfter vorbei ist und in erster Linie Energie-effiziente Rechner up to date sind. Denn während die Verdopplung der Rechenleistung über eine höhere CPU-Taktung den Stromverbrauch vervierfacht, benötigt ein Dualcore-Prozessor nur die doppelte Energiemenge. Diese leicht verständliche Rechnung begründet den Siegeszug von Multicore, dem Linux seit etwa Mitte der 90er Jahre auch funktionell Rechnung trägt.

Auf Singlecore-Systemen gibt es eine vereinfachte Form von Parallelität im Kernel nur, wenn die CPU einen Interrupt auslöst. Den einfachen Trick, in kritischen Situationen die (quasi) Parallelität durch Sperren von Interrupts auszuschalten, wandten schon die frühen Unix-Systeme an. Die ersten Multicore-Linux-Versionen dehnten dieses bewährte Konzept einfach auf alle vorhandenen Cores aus – der Big Kernel Lock war geboren.

Abbildung 1: **Die erste Folge der Kern-Technik stellte und beantwortete grundlegende Fragen: „Wie schreiben künftige Kernel-Gurus ihr erstes Modul? Welche Anpassungen sind erforderlich, um einen 2.4-Treiber auf 2.6 zu portieren? Wie realisiert der neue Kernel Hardwarezugriffe? Und welche Werkzeuge und Techniken sind zur Treiberprogrammierung nötig?"**

Ab Quadcore Hardcore

Bis zum Quadcore-System skaliert das pauschale Sperren der Interrupts leidlich gut – nur die merklich steigenden Latenzzeiten stören. Wer mehr als vier CPU-Kerne besitzt – heutzutage ist das schon bei einem Mittelklasse-Smartphone der Fall –, profitiert von deren Rechenkraft infolge des Big Kernel Lock praktisch kaum. Linus Torvalds hat zusammen mit dem deutschen Kernelentwickler Thomas Gleixner daher den Quellcode gründlich überarbeitet. Der Big Kernel Lock wich vielen dedizierten Lock-Mechanismen. Seither lautet das Mantra: Multicore ist Standard, Singlecore old school.

Multicore beeinflusste aber nicht nur das Kernel-Locking, sondern bedingte auch einen neuen Multicore-Scheduler. Das Thema beschäftigt heute weiterhin die

Informatik. Selbst wenn im Linux-Kernel immer wieder einmal ein neuer Singlecore-Scheduler auftaucht (Kern-Technik 67), im Prinzip ist das Singlecore-Scheduling Forschungs- und Implementierungstechnisch abgefrühstückt. Verbesserungen gibt es nur noch im Detail oder für spezifische Situationen.

Der Multicore-Scheduler dagegen ist hochkomplex, beileibe nicht ausgeforscht und wird die Kern-Technik in Zukunft sicher noch mehrmals beschäftigen. Wie in Kern-Technik 33 beschrieben, evaluiert der Kernel beim Hochfahren die Hardware-Architektur und setzt dann auf komplexe mathematische Funktionen, um die Kosten für eine Taskmigration und den damit erzielbaren Gewinn zu berechnen. In der Liste der Rechenprozesse ist der Multicore-Scheduler über den Job mit dem Namen »migration« zu finden.

Tabelle 1: Linux-Errungenschaften
Ausgewählte Funktionalitäten
Hardware-Ankopplung
Gpio-Subsystem
Pinctrl
Device Tree
Threaded Interrupts
32 und 64 Bit nativ
Secure Boot
Virtualisierung
Namespaces
Kvm
Scheduling
Multicore-Scheduling
»TASK_KILLABLE« (Prozesszustände)
Effizienz-Scheduling
»wait_event_interruptible()« - Wechsel Taskzustand
»kernel_thread()« – Kernelthreads generieren
Modulkonzept
Loadable Modules
Signierte Kernelmodule
Dateisysteme
Ext 2 bis 4, Reiser, ZFS, ...
Virtuelle Filesysteme (Proc-FS, Sys-FS, Temp-FS, ...)
Schutz kritischer Abschnitte
Spinlocks – Verbesserung des Codes
»global_lock« wird ausgemerzt
Umgang mit Zeit
Tickless System
Zeitbasis »struct timespec«
Hrtimer
Orts- und Raum-Unabhängigkeit, »CLOCK_MONOTONIC«
Debugging (Kgdb)

Kurz vor Drehschluss

Auch abseits der Prozessoren haben sich in den vergangenen Jahren kleine Revolutionen ereignet. Beim Permanentspeicher in PCs und Servern drehten Jahrzehnte lang magnetisierbare Scheiben mit 5400 oder 7200 Umdrehungen pro Minute ihre Runden. Doch deren Summen verstummt nach und nach, weil mechanisch unauffällige und zudem schneller arbeitende SSDs ihre Rolle übernehmen.

Für den Linux-Kernel bedeutet das den Rauswurf einer ganzen Menge jetzt überflüssigen Codes, der bisher für den optimierten Zugriff bei mechanischen Verzögerungen gesorgt hat (Kern-Technik 85). Wohl wissend, dass Betriebssysteme den Zugriff auf Festplatten per IO-Scheduler optimiert haben (Kern-Technik 19 und 20), bauen unglücklicherweise die Hersteller ihre SSDs so, dass sich diese wie Festplatten verhalten. Das ist ärgerlich, wird auf diese Art der mögliche Performance-Vorteil zunächst wieder verspielt.

ARM dran

In der berühmt gewordenen Debatte zwischen Andrew Tanenbaum und Linus Torvalds hat sich der Minix-Erfinder 1992 nicht nur mit dem Erfolg von Linux gewaltig verschätzt, er lag auch bei der Lebensdauer der x86-Architektur (fünf Jahre) völlig daneben **[2]**. Seine Prognose: Sun Sparc wird die Architektur der Zukunft. ... Immerhin: Die Sparc-Plattform existiert noch und ist nicht vollständig vom Markt gefegt.

X86-Prozessoren, hergestellt von Intel oder AMD, führen weiterhin den bescheidenen Markt der Personal Computer und Server an. Smartphones, Wearables und Tablets dagegen, in der Summe ein Vielfaches, sind so gut wie alle mit einem oder gar mehreren ARM-Prozessoren bestückt. ARM-Prozessoren sind nämlich erheblich Energie-effizienter und ermöglichen den Bau portabler Geräte ohne surrende Lüfter.

Außerdem ist ARM zumindest etwas offener als Intel. So verkauft die Firma keine Prozessoren, sondern nur das Know-how, Intellectual Property, zum Bau derselben. Das gibt Apple & Co. die Möglichkeit, eigene Chips rund um den definierten Befehlssatz designen und herstellen zu lassen und sich von der Konkurrenz durch ihre Chips abzusetzen.

Dass die Lizenzkosten pro CPU im ein- bis zweistelligen Cent-Bereich liegen, tut ein Übriges zur massiven Verbreitung. In der Welt der eingebetteten Systeme ist ARM mit Abstand der am meisten eingesetzte (32-Bit-)Prozessor. In Abgrenzung zu ein paar Hundert Millionen x86-Chips kommen mehrere Milliarden ARM-Prozessoren pro Jahr beim Verbraucher an. Mit neuen Prozessoren versucht ARM auch den Desktop und Serverbereich zu erobern. Mehrere Geräte, die einen x86-Prozessor emulieren und auf denen damit Windows läuft, sind seit Kurzem zum Verdruss von Intel im Markt.

Wachsender Device Tree

Der Vielfalt an ARM-lizenzierten CPUs wohnt aber auch der Nachteil inne, dass ARM-Linux mit einem ganzen Zoo an Plattformen klarkommen soll, und jede braucht ihren eigenen Kernel. Weil das auf Dauer immer lästiger wurde, haben die Kernelentwickler nachgelegt und die Hardware-Konfigurationsdaten, etwa die Hardware-Adressen, und die eigentliche Verarbeitung separiert.

Sobald der Kernel eine passende Hardwarebeschreibung findet, kann er die zugehörige Plattform bedienen. Der Plattform-Spezifikation gaben die Entwickler den Namen Device Tree (Kern-Technik 68). Anfangs bediente sich der Baum bei einer kompakten Datei, heute sind die Informationen auf mehrere Verzeichnisse verteilt (Kern-Technik 93).

Neben ARM etabliert sich überraschenderweise eine weitere Plattform: Risc V. Dafür gibt es erwartungsgemäß bereits eine Linux-Portierung. Wie kommt es, dass eine neue Hardware gegen den Platzhirschen ARM anstinken kann? Risc V besinnt sich auf die Linux-Tugenden: Open Source und Lizenzkosten-Freiheit!

Pünktlich wie ein Linux

Eine wichtige Großbaustelle im Linux-Kernel war das Einbringen von Realzeit-Eigenschaften. Baumeister ist hier der

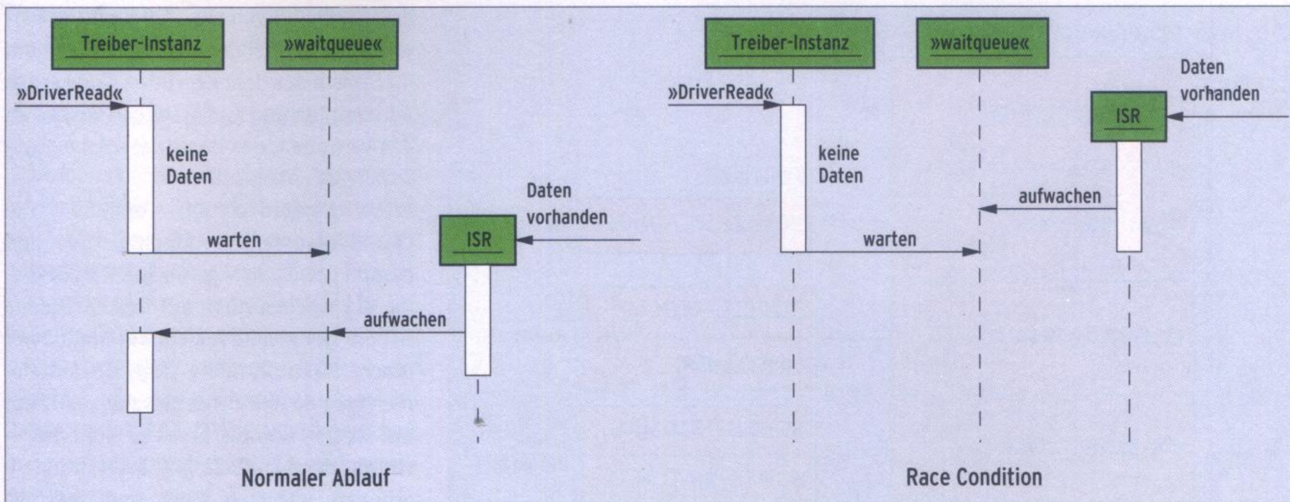


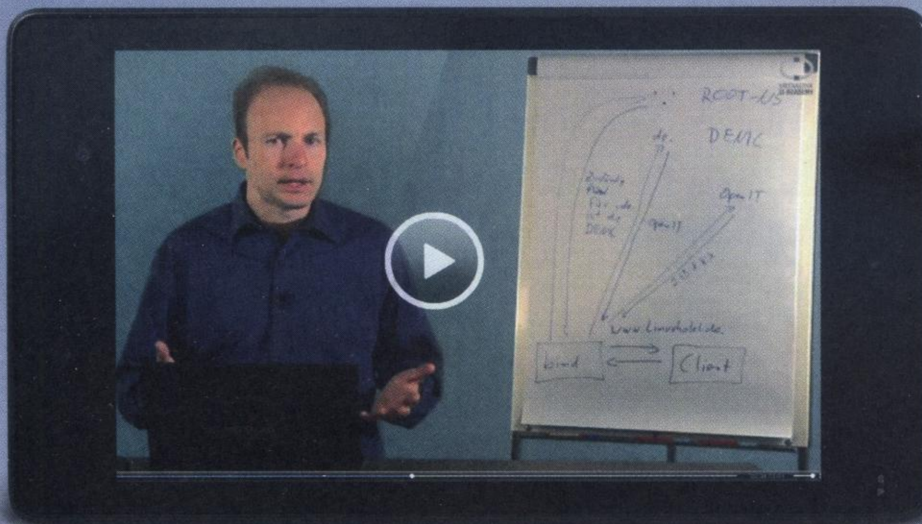
Abbildung 2: Dass zu frühes Aufwecken eines schlafen gelegten Treibers zum Deadlock führen kann, war 2003 weithin unbekannt.

deutsche Entwickler Thomas Gleixner. Das mit einem periodischen Timer-Tick betriebene Linux wandelte sich auf diese Weise zum Tickless-System, das nur noch dann aktiv wird, wenn wirklich Aufgaben anliegen.

Im Kontext mit den verkürzten kritischen Abschnitten führt das nicht nur zu einem berechenbareren, verbesserten Zeitverhalten, sondern auch zu einer deutlich effizienteren Verarbeitung – für aktuelle Mobilgeräte ein Muss. Die lange praktizierten, im Millisekunden-Bereich messenden Jiffies sind Zeitstempeln mit Nanosekunden-Genauigkeit gewichen.

Darüber hinaus hat Linux dank Realzeit-Mutexen gelernt, mit so genannten Prioritäts-Inversionen umzugehen und bei

Linux-Zertifizierung LPIC-1 / LPIC-2
Mit Ingo Wichmann
- Lernen Sie mit LPI-zertifizierten Trainern und Dozenten!
- 100% abgestimmt auf die originalen Lehrpläne des LPI!
- Bereiten Sie sich optimal auf die LPIC-1- und LPIC-2-Prüfungen vor!



IT-Onlinetrainings
Mit Experten lernen.

LPIC-Prüfungsvorbereitung
mit Ingo Wichmann, Linuxhotel

LPIC-1 Kurs LPI 101 299 €
LPIC-2 Kurs LPI 201 299 €
LPIC-1 Kurs LPI 102 299 €
LPIC-2 Kurs LPI 202 299 €
LPIC-1 Paket (101+102) 499 €
LPIC-2 Paket (201+202) 499 €

You Tube /ComputecAcademy
www.computec-academy.de

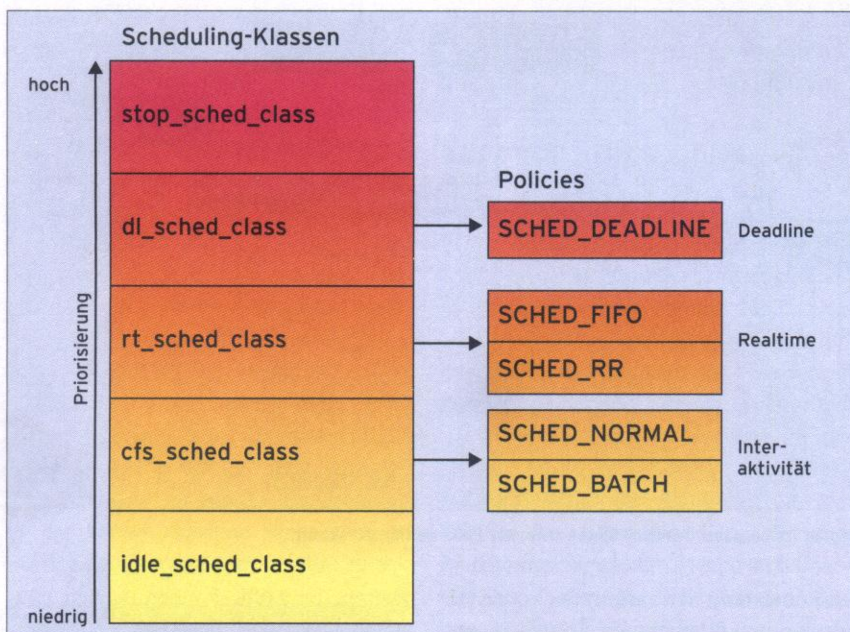


Abbildung 3: Scheduling-Klassen erlauben es, unterschiedliche Scheduling-Verfahren parallel einzusetzen.

Bedarf eine Prioritätsvererbung durchzuführen. Zur Befriedigung von Realzeit-Anforderungen ist neben den klassischen Prioritäten das Deadline-Scheduling hinzugekommen, und über das Konzept der Scheduling-Klassen dürfen Systeme sogar unterschiedliche Scheduling-Verfahren gleichzeitig verwenden (Kern-Technik 79, **Abbildung 3**).

Die Multicore-Architektur hat die Möglichkeit gebracht, einzelne Prozessorkerne für Realzeit-Aufgaben zu reservieren und diese Kerne schließlich auch vor Fremdaufgaben zu isolieren (Kern-Technik 33). Kombiniert der Systemarchitekt dies mit Threaded Interrupts, kann er in vielen Fällen ein deterministisches Verhalten garantieren (Kern-Technik 81).

Securitas maxima

Wenn es um sichere Mainstream-Betriebssysteme geht, ist Linux erste Wahl. Den Anwendern der proprietären Konkurrenz (Windows Server, die verbliebenen Unixe, Windows 10, Mac OS, Windows Embedded, OS 9, …) bleibt es verwehrt, den Quellcode zu studieren und ihr System aus dem geprüften Code selbst zu kompilieren. Entsprechend gering ist das Vertrauen, das Benutzer solchen Systemen schenken sollten.

Linux dagegen erfüllt – erstens – die Grundvoraussetzung Open Source und es gibt – zweitens – genügend Entwickler, die ein waches Auge auf den Quellcode haben. Der zweite Aspekt der Manpower macht Linux attraktiv gegenüber anderen Open-Source-Betriebssystemen. Denn auf Dauer kommt andere, sicherheitstechnisch eigentlich gut beleumundete Software wie Free, Open und Net BSD immer häufiger unter die Räder, weil den vergleichsweise kleinen Communities immer mehr Bugs durchrutschen **[3]**.

Zufall zur Hilfe

Von diesen Aspekten abgesehen, unterstützt Linux alle gängigen Sicherheitsmechanismen, Address Space Layout Randomization (ASLR) und Data Execution Prevention (DEP) beispielsweise. Die Kern-Technik 38 konnte aufzeigen, dass es nicht nur darauf ankommt, die Technik als solche zu unterstützen, sondern auch in ausreichender Qualität. ASLR verteilt die Adresslagen der einzelnen Segmente eines ausführbaren Programms zufällig im Hauptspeicher, sodass der arme Angreifer nur noch raten kann. DEP schließ-

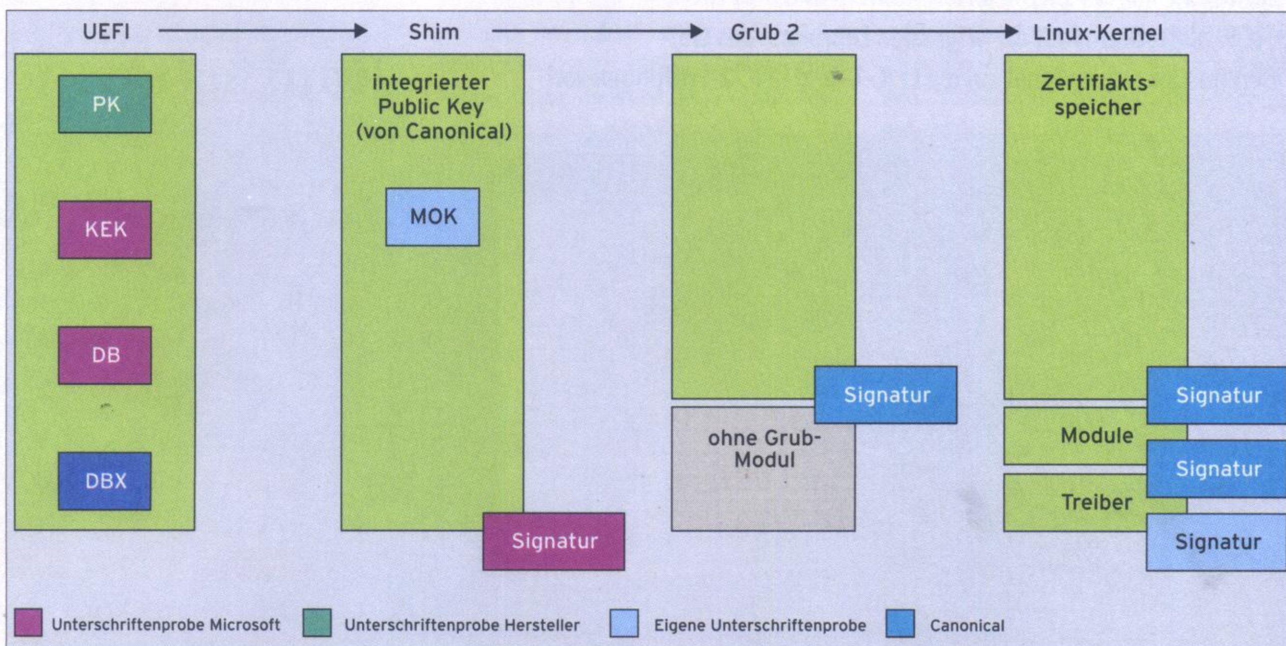


Abbildung 4: Bei aktiviertem Secure Boot fährt Linux zwar sicher, aber nur von Microsofts Gnaden hoch.

lich hindert ihn daran, Code auszuführen, der auf dem Stack liegt.

Geschützte Startrampe

Daneben haben die Linux-Entwickler diverse weitere Sicherheitsmechanismen implementiert. So lässt sich Linux per Secure Boot hochfahren (Kern-Technik 94), wenngleich nur mit Microsofts Erlaubnis (**Abbildung 4**). Danach lädt das System natürlich nur Komponenten, die eine gültige, digitale Unterschrift tragen (Kern-Technik 82). Diese allerdings kann mit etwas Handarbeit der Entwickler selber leisten.

Gewiss ist, dass Sicherheit der IT im Allgemeinen und Linux im Besonderen als Dauerbrenner erhalten bleiben wird. Bezüglich Firewalling beispielsweise bahnt sich im Kernel gerade eine große Veränderung an – aber das ist auch ein Thema für eine künftige Kern-Technik.

Ruhe da draußen!

Es ist als gutes Zeichen zu werten, dass die oft umfangreichen Änderungen am Linux-Kernel für die meisten Anwender unsichtbar bleiben und dieser demnach über Jahre genau das tut, was er soll: funktionieren. Als Auftragsmaschine arbeitet er geflissentlich die über die Applikationen erteilten Systemcalls sorgsam und zuverlässig ab.

Das ist natürlich begrüßenswert, bringt aber den pädagogischen Nachteil mit sich, dass immer weniger Leute Linux' Motorhaube öffnen und das Meisterwerk innen zu sehen bekommen. Software, nach den Regeln der Kunst geschrieben: objektorientiert ohne objektorientierte Sprache, stets die neueste Technologie repräsentierend. Immer wieder gut für Überraschungen, die wir Serien-Autoren in den Kern-Techniken ab der Nummer 101 thematisieren werden. *(jk)*

Fäkalien fließen ab

Dass Linux erwachsener geworden ist, lässt sich auch an der Sprache der Entwickler festmachen [4]. Die Kern-Technik 50 hatte sich zur Halbzeit aufs verbale Örtchen begeben und die Kraftausdrücke im Linux-Kernel-Quellcode gezählt. Zwischenzeitlich ist der Quellcode erheblich umfangreicher geworden, Kraftausdrücke sind allerdings kaum dazugekommen (Abbildung 5).

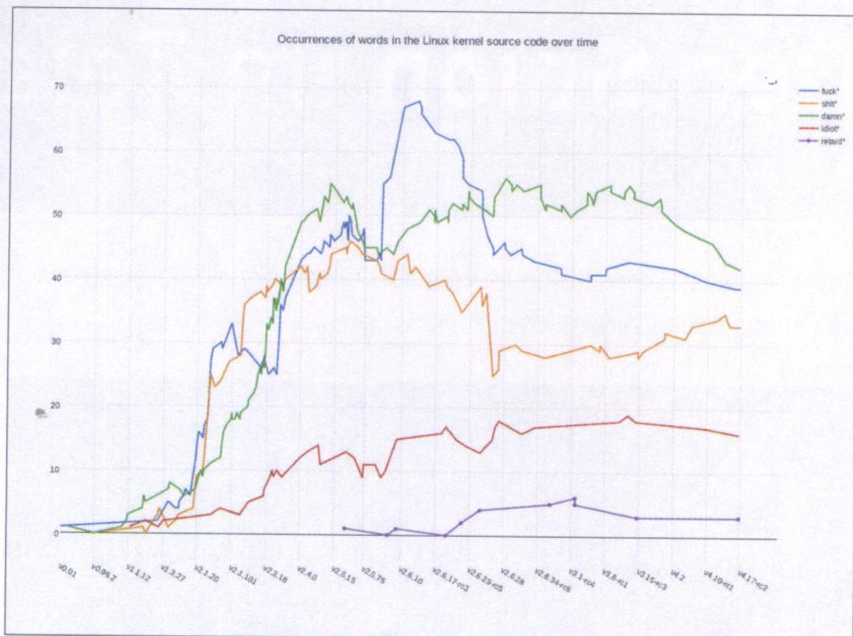


Abbildung 5: Trotz wachsendem Umfang beherbergt der Linux-Quellcode eher weniger Kraftausdrücke.

Infos

[1] Bruce Perens, „Intel Publishes Microcode Security Patches": [https://perens.com/2018/08/22/new-intel-microcode-license-restriction-is-not-acceptable/]

[2] Tanenbaum vs. Torvalds: [https://en.wikipedia.org/wiki/Tanenbaum%E2%80%93Torvalds_debate]

[3] Ilja van Sprundel, „Are all BSDs created equally? – A survey of BSD kernel vulnerabili9es": [https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEFCON-25-Ilja-van-Sprundel-BSD-Kern-Vulns.pdf]

[4] Vidarholen, „Wordcount": [https://www.vidarholen.net/contents/wordcount/]

LINUX MAGAZIN ONLINE

NEWSLETTER FÜR IT-PROFIS

Newsletter

News

Stadt Dortmund prüft Einsatz freier Software und offener Standards

- Tagesaktuelle IT-News
- Security-Infos des DFN-CERT
- Online-Stellenmarkt

Jetzt kostenfrei abonnieren! www.linux-magazin.de/newsletter