Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 96

Kern-Technik

An Kernelthreads herrscht in modernen Linux-Systemen wahrlich kein Mangel, wohl aber an brauchbarer Dokumentation dazu. Dem misslichen Umstand begegnet diese „Kern-Technik"-Folge, indem sie die Tasks systematisch erfasst und verstehbar macht, die fürs Speichermanagement zuständig sind. Eva-Katharina Kunst, Jürgen Quade

© psdesign1, Fotolia

In den letzten Entwicklungsjahren hat Linus Torvalds Stück für Stück den Code zuvor im Kernel eingewebter Funktionalitäten herausgeschält und in einzelne Kernelthreads überführt. Sicher hatte der Meister dafür gute Gründe. Zugleich konfrontiert – ja verwirrt – er die Anwenderschaft mit Details, von denen sie bislang kaum etwas wusste.

Die letzte „Kern-Technik" hat berichtet, dass gewöhnliche Distributionen nach dem Booten weit über 200 Tasks mit teils kryptischen Namen vorweisen – mehr als die Hälfte sind Kernelthreads. Selbst der Experte gerät beim Versuch, alle Tasks eines Linux-Systems zu erläutern, ins Stottern, Dokumentationen zu dem wenig intuitiven Thema sucht er im Internet vergebens.

Ein Schlaglicht ins Dunkle hat bereits die erwähnte Kern-Technik [1] mit ihrer Liste der Kernelthreads-Top-Ten gerichtet. Die heutige Folge knöpft sich die Tasks vor, die den Speicher sowie das Blockgeräte-Subsystem verwalten (siehe Tabelle 1), und erklärt dabei die Basis-Mechanismen zum effizienten Zugriff auf Haupt- und auf Hintergrundspeicher, also auf SSDs, Festplatten, USB-Sticks und so weiter.

Wenn es dem Kernel um den Hauptspeicher geht, sind die Dienste der Kernelthreads »vmstat«, »kcompactd«, »khugepaged« und »ksmd« gefragt. So bereitet der Thread »vmstat« die Informationen über die Hauptspeicherbelegung von Tasks, die Nutzung virtuellen Speichers, das Auslagern von Hauptspeicherinhalten, IO- oder CPU-Aktivität auf. Der Kernel spiegelt Vmstats Ergebnis im Proc-Verzeichnis, der Admin profitiert davon, wenn der in der Konsole das gleichnamige Kommando »vmstat« ausführt.

COW-Boys lernen schreiben

Der KSM-Daemon (Kernel Shared Memory) überprüft den Speicher nach Seiten identischen Inhalts. Findet er welche, löscht Linux die Dubletten und korrigiert die Verweise innerhalb der Speicherverwaltung (Abbildung 3). Die damit mehrfach referenzierten Speicherseiten markiert das System als COW (Copy On Write). Solange alle Threads nur lesend auf die dedoublierten Seiten zugreifen,

Prozessparameter

Jeder kennt die Kommandos »ps«, »top« oder »htop«, welche die im System vorhandenen Tasks auflisten. Je nach Anzeige tauchen dabei die PID, der Name des Besitzers (»USER« in Abbildung 1), der Prozesszustand (»S« oder »STAT«), der Name der Task (»Command«) und die verbrauchte Rechenzeit (»TIME«) auf.

Meist interessieren den Anwender vor allem die durch den Job bewirkte prozentuale CPU-Auslastung (»CPU%«) und der benutzte Anteil des physischen Speichers (»MEM%«). Der belegte Speicher wird auch als »RSS« (Resident Set Size) bezeichnet. Der virtuell verwendete Speicher findet sich in der Spalte »VSZ« beziehungsweise »VIRT«.

Die von den Kommandos dargestellten Prozesszustände nehmen die Werte D (nicht unterbrechbares Schlafen), R (lauffähig), S (unterbrechbares Schlafen), T (angehalten per Job Control), t (angehalten per Debugging) oder Z (Zombie) an (Abbildung 2). Auch wenn die Abkürzung R für Running steht, ist damit lauffähig gemeint. Welchen Job die CPU gerade tatsächlich abarbeitet (Zustand: aktiv), ist an der Prozessliste nicht abzulesen. Zum Zombie wird ein Prozess, der sich beendet und seinen Speicher zurückgegeben hat, dessen Elterntask es aber versäumt hat, den Exitcode des Kindes abzuholen. Solange nämlich bleibt die leere Hülle – der Task-Kontrollblock – im System präsent.

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.

gibt es keine Probleme. Möchte ein Job aber die Seite verändern, bekommt er wieder eine eigene, private Kopie.

Die Suche nach doppelten Seiten hat Linux übrigens erst mit aufkommender Virtualisierung eingeführt, in der Kernel-Standardkonfiguration ist sie deaktiviert. Per »echo 1 > /sys/kernel/mm/ksm/run« setzt sie der Admin in Gang. Über die Datei »pages_shared« unterhalb von »/sys/kernel/mm/ksm/« lässt er sich die Anzahl gemeinsam genutzter Seiten ausgeben.

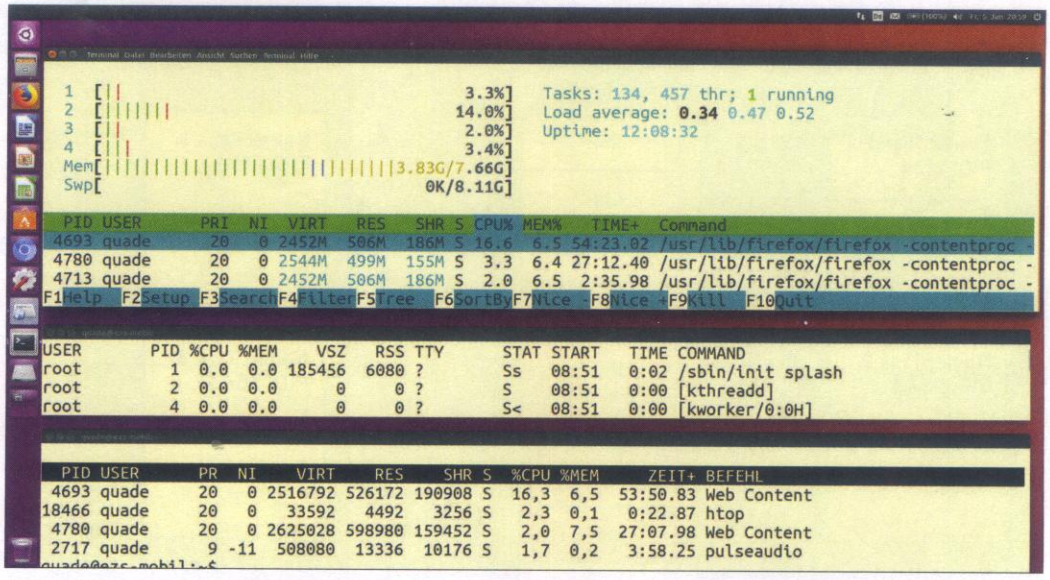

Abbildung 1: Die Kommandos »htop« (oben), »ps« (Mitte) und »top« (unten) listen diverse Prozessparameter auf.

Große Häppchen

Komplexer funktioniert »khugepaged«. Er entlastet den Programmierer dabei, sich Gedanken über die Größe des physischen Hauptspeichers zu machen. Denn ob eine Maschine mit viel oder mit wenig Hauptspeicher bestückt ist: Die Applikation funktioniert ohne Anpassung – höchstens die Performance variiert.

Damit das so ist, bekommt die Applikation physischen Speicher nicht direkt bereitgestellt, sondern virtuellen. Der Kernel kümmert sich darum, den virtuelle Speicher auf physische Speicherbereiche abzubilden (zu mappen). Dazu teilen Betriebssystem und CPU den Speicher in Blöcke (Pages) ein. Normal sind Blöcke der Größe 4 KByte.

Für Applikationen, die den virtuellen Adressraum ausnutzen, fällt allerdings eine große Menge an Verwaltungsinformationen an, was sich wiederum negativ auf die Zugriffszeit auswirkt. Vereinfacht gesagt – manchmal sind drei oder mehr Hauptspeicherzugriffe notwendig, um ein Datum aus dem Hauptspeicher zu holen.

Intel & Co. haben auf das Problem mit der Möglichkeit reagiert, mit größeren Pages von beispielsweise 2 MByte – also dem 512-Fachen – zu arbeiten. Zunächst hatte Linus Torvalds es den Entwicklern überlassen, Hugetables für ihre Jobs zu verwenden. Mit Kernel 2.6.38 schwenkte der Chefentwickler dann aber zu so genannten transparenten Hugetables um **[2]**, bei denen der Kernel selbstständig und nur bei Bedarf und Möglichkeit große Tabellen verwendet.

Dazu versucht der Kernelthread »khugepaged« regelmäßig größere, physisch benachbarte Speicherbereiche zu reservieren und bei Erfolg einen Job zu finden, dem er diesen zusammenhängenden Speicher im Austausch gegen mehrere kleine Seiten unterjubelt. Dafür müssen aber ausreichend große physische Speicherbereiche vorhanden sein.

Ist »khugepaged« bei der Reservierung kein Erfolg beschieden, wird der nächste Kernelthread aktiv: der »kcompactd«. „Verdichter", so die direkte Übersetzung ins Deutsche, trifft seine Funktion nicht perfekt, „Defragmentierer" passt besser. Seine Aufgabe besteht darin, physisch zusammenhängenden Speicher zu besorgen. Jonathan Corbet stellt in **[3]** das Verfahren der Defragmentierung

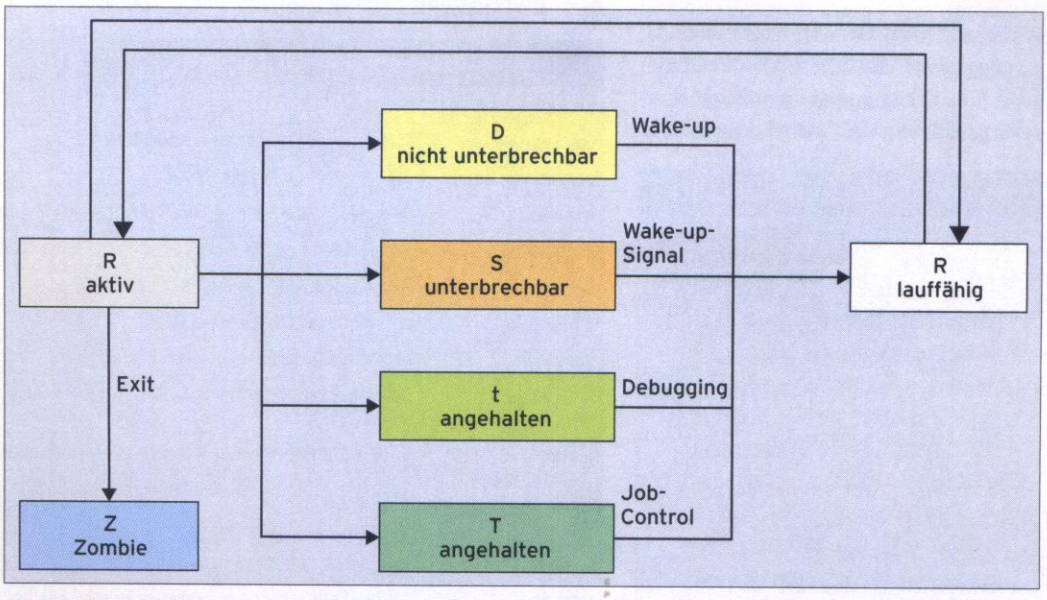

Abbildung 2: Vereinfachtes Zustandsdiagramm einer Task in Linux. Verwalter der lauffähigen Prozesse und Threads ist der Scheduler.

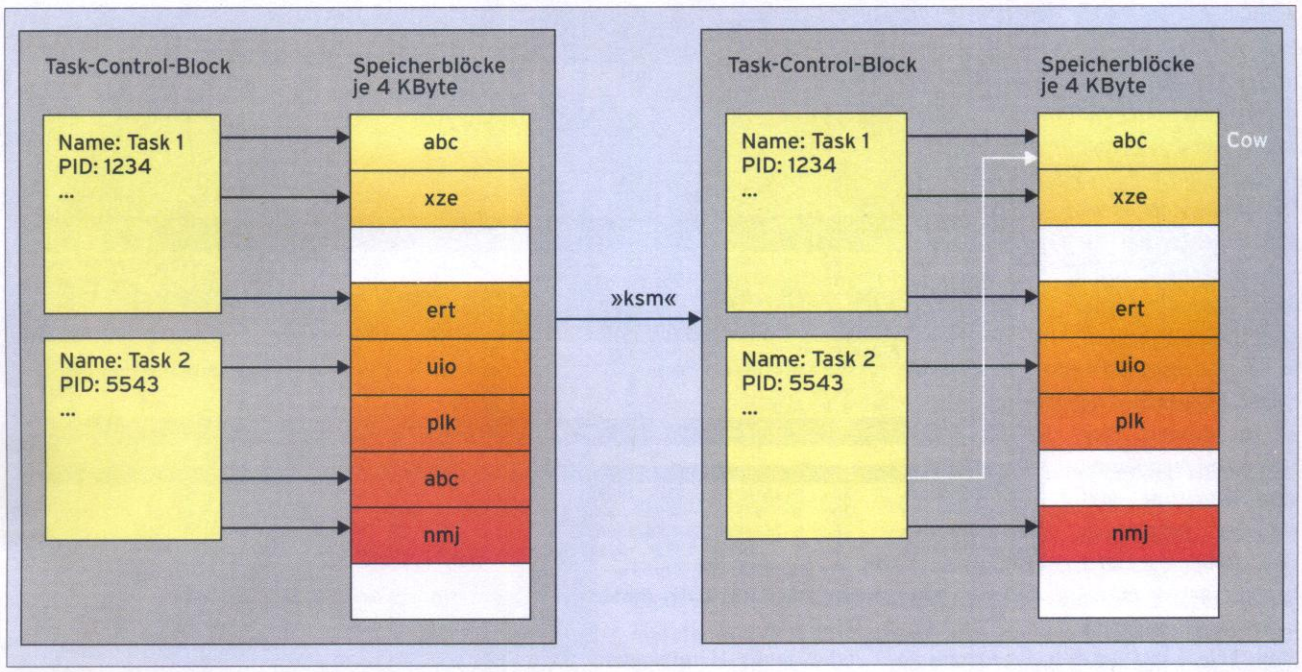

Abbildung 3: Der Kernelthread »ksm« räumt Speicherseiten identischen Inhalts weg.

von Hauptspeicher vor. Der Algorithmus führt, beginnend bei den niedrigeren Speicherseiten, eine Liste mit belegten Pages (**Abbildung 4**). Parallel dazu pflegt er eine Liste der freien Pages, bei den oberen Seiten beginnend. Sobald sich die beiden Durchläufe treffen, initiiert der Thread – sofern möglich – das Verschieben (Kopieren) der belegten Pages in die freien Bereiche. Gelingt das Verschieben, erzeugt der Algorithmus die erwünschten zusammenhängenden Speicherbereiche.

Das Blockgeräte-Subsystem

Während die Kernelthreads »ksmd«, »khugepaged« und »kcompactd« vorwiegend für so genannte anonyme Speicherseiten zuständig sind – also das, was für die Applikationen der normale Hauptspeicher ist –, initiiert der Kernelthread »writeback« periodisch den Transfer der Seiten im Pagecache auf die SSD oder die Festplatte (Hintergrundspeicher).

Als Pagecache bezeichnen Kernelhacker die Speicherseiten, die den Inhalt von Dateien zwischenspeichern und damit den Dateizugriff erheblich beschleunigen. Denn damit landet nicht jede kleine Änderung im Dateisystem sofort auf dem zuständigen Blockgerät, sondern erst, wenn es sich lohnt. Das kann der Benutzer beobachten, wenn er das Kommando »cat /proc/meminfo | grep "Dirty"« bemüht. In **Listing 1** zeigt es beim ersten Mal die Anzahl Bytes an, die noch nicht auf den Hintergrundspeicher geschrieben waren. Das Kommando »sync« aktiviert danach den »writeback«-Kernelthread, sodass beim zweiten Aufruf die Dirty-Bytes verschwunden sind.

Während »writeback« das Schreiben der Daten nur initiiert, ist der »kblockd« als Teil des IO-Schedulers für das Schreiben selbst zuständig. »writeback« optimiert also das Schreiben einzelner Blöcke, und »kblockd« optimiert die Reihenfolge der Schreib- und auch der Lese-Aufträge. Das lohnt sich, weil rückwärtslaufende

Listing 1: Erzwungenes Writeback

```
01 root# dd if=/dev/zero of=testfile.txt bs=1M count=10
02 10+0 Datensätze ein
03 10+0 Datensätze aus
04 10485760 bytes (10 MB, 10 MiB) copied, 0,050544 s,
   207 MB/s
05 root# cat /proc/meminfo | grep Dirty
06 Dirty:          10276 kB
07 root# sync
08 root# cat /proc/meminfo | grep Dirty
09 Dirty:              0 kB
```

Tabelle 1: Ausgewählte Kernelthreads

Thread-Name	Aufgabe
dmcrypt_write	Schreiben der verschlüsselten Daten
ecryptfs-kthread	Dateien ver- und entschlüsseln
kblockd	Optimierte Abarbeitung von IO-Aufträgen (Teil des IO-Schedulers)
kcompactd	Zusammenhängende Speicherbereiche freischaufeln (Hugetable)
kcryptd	Dateisysteme ver- und entschlüsseln
kcryptd_io	Lesen der verschlüsselten Daten
kdmflush	Daten des Device Mapper schreiben
khugepaged	Den Adressenumsetzungs-Aufwand optimieren durch Verwendung von Pages größer 4 KByte
kintegrityd	Unterstützung von SCSI-Integritätsinformationen
ksmd	Speicherseiten gleichen Inhalts suchen und Dubletten löschen
md	Multiple Devices: Software-Raid
writeback	Pagecache und Hintergrundspeicher (SSD, Festplatte) synchronisieren
vmstat	Bereitet statistische Informationen zur Speicherverwaltung auf

Zugriffe auf Datenblöcke einer Festplatte grob doppelt so lang dauern wie vorwärtsgerichtete. Daher sammelt der IO-Scheduler erst einige Aufträge, sortiert sie nach einem IO-Scheduling-Algorithmus um und führt sie erst dann aus [4].

Prüfsummen gegen Fehler

Beim Ablegen von Daten auf Hintergrundspeicher, also auf Festplatten oder SSDs, können nicht nur infolge von Hardwaredefekten Fehler entstehen. Auch die beim Speichern beteiligte Software im Kernel, auf dem Controller und in dem Medium selbst ist mittlerweile so komplex, dass hier Fehler auftreten, die zu Datenverlusten führen.

Anhand von Checksummen fallen diese Fehler typischerweise beim Einlesen auf, was aber nicht verhindert, dass die gespeicherten Nutzdaten verloren sind. Das technische Komitee T10, das für die SCSI-Platten-Protokolle zuständig ist, hat mit DIF (Data Integrity Field, [5]) eine Erweiterung vorgenommen, die zu jedem Block 8 Byte Metadaten definiert. Darüber kann das Betriebssystem bereits beim Schreiben der Daten deren Integrität prüfen und im Fehlerfall Gegenmaßnahmen treffen. Das ist Aufgabe des Integrity-Daemon [6].

Virtuelle Blockgeräte

Ähnlich wie der Hauptspeicher eine virtuelle Entsprechung hat, gibt es bei Blockgeräten virtuelle Vertreter. Sie ermöglichen etwa Dateisysteme zu verschlüsseln, Software-Raids oder das Anlegen von Dateisystem-Snapshots. Der Linux-Kernel realisiert virtuelle Blockgeräte über den Device Mapper (DM), der für die Abbildung (Mapping) und die Modifikation von Blöcken auf die physischen Blockgeräte zuständig ist.

Der Kernelthread »kdmflush« erledigt Aufgaben für den Device Mapper, die nicht im Interruptkontext ausführbar sind, insbesondere das Schreiben der Daten respektive das Weiterleiten an die eigentlichen Blockgeräte.

Geht es um die Modifikation von Daten innerhalb des Device-Mapper-Subsystems, treten die Kernelthreads »kcryptd«, »kcryptd_io« und »dmcrypt_write« auf den Plan. Sie ver- und entschlüsseln die Daten ganzer Dateisysteme (Dm-crypt), also etwa die Rootpartition.

Während »kcryptd« die eigentliche Ver- und Entschlüsselung vornimmt, zeichnen für die Ein- und Ausgabe »kcryptd_io« (der Spezialist für das Lesen) und »dmcrypt_write« (schreibt nur) verantwortlich.

Die Auftrennung der Aufgabe in mehrere Threads ist notwendig, da beim Lesen im Falle von Speichermangel der »kcryptd« blockiert. Dann wäre er auch nicht mehr in der Lage, den Speicher der gerade abgeschlossenen Aufträge freizugeben.

Der Kernelthread »ecryptfs-kthread« ver- und entschlüsselt ebenfalls, tut dies aber auf Datei-Ebene. Ecryptfs chiffriert also zum Beispiel die Dateien im Homeverzeichnis eines Users.

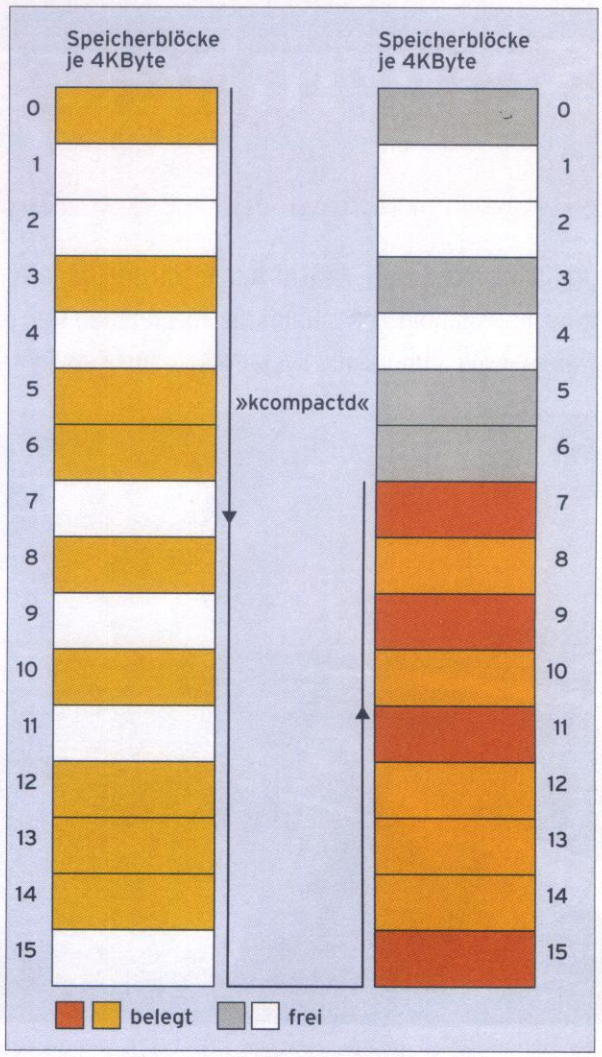

Abbildung 4: Der Kcompactd versucht verwendete Speicherseiten in freie Speicherseiten zu verschieben.

Multiple Devices

Wer ein Software-Raid einrichten will, kann alternativ den Device Mapper »md« (Multiple Devices) bemühen. Denn auch das MD-Subsystem stellt virtuelle Geräte (über Gerätedateien) zur Verfügung. Das Einrichten gelingt mit Hilfe des Werkzeugs »mdadm« [7].

Zum Blockgeräte-Subsystem gehören noch mehr Kernelthreads, beispielsweise »nvme« und Verwandte wie »jbd2/nvme0n1p3-«. Die sind im System nur dann existent, wenn im Rechner eine SSD mit NVM-Express-Interface steckt. Viele Threads starten nämlich nur, wenn Linux die zugehörige Hardware findet. Das verhindert nebenbei, sich eine vollständige Liste prinzipiell möglicher Kernelthreads anzufertigen. Gelänge es trotzdem – sie wäre sehr, sehr lang. *(jk)*

Infos

[1] Quade, Kunst, „Kern-Technik", Folge 95 zu den zehn wichtigsten Kernelthreads: Linux-Magazin 01/18, S. 76

[2] Jonathan Corbet, „Transparent huge pages in 2.6.38", Linux Weekly News: [https://lwn.net/Articles/423584/]

[3] Jonathan Corbet, „Memory Compaction", Linux Weekly News: [https://lwn.net/Articles/368869/]

[4] Quade, Kunst, „Kern-Technik", Folge 19 zu IO-Scheduling: Linux-Magazin 03/05, S. 88

[5] T10 Technical Committee, Document 03-111 zum Thema DIF: [http://www.t10.org/ftp/t10/document.03/03-111r0.pdf]

[6] Martin Peterson, „Linux Data Integrity Extensions": [https://www.landley.net/kdocs/ols/2008/ols2008v2-pages-151-156.pdf]

[7] Software-Raid: [https://wiki.ubuntuusers.de/Software-RAID/]