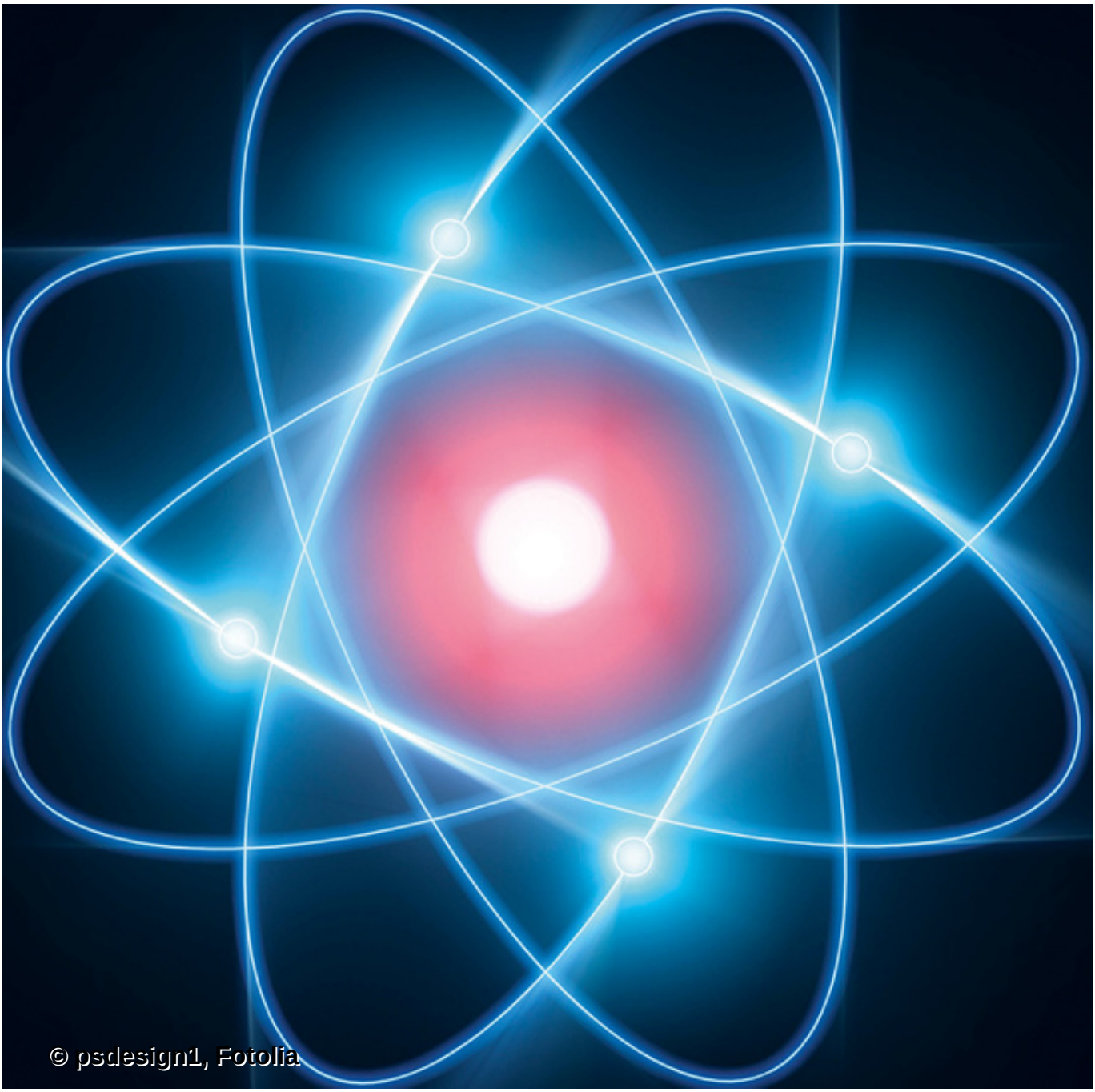
Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 91

Kern-Technik

Geschichtete Dateisysteme bilden eine wichtige Basistechnik für Container und verhelfen dem Flashspeicher von eingebetteten Systemen zu längerem Leben. Ihre jetzige Reife und Einsatzfähigkeit haben sie erst nach Jahrzehnten der Entwicklung erreicht – warum das so ist, erklärt die Kern-Technik. *Jürgen Quade*, *Eva-Katharina Kunst*



Die Anfang 2010 im Schmelztiegel Oracle Corporation aufgelöste Unix- und Java-Firma Sun Microsystems zeichnete in den besseren Tagen ihrer 28 Jahre andauernden

Geschichte für diverse Innovationen verantwortlich. Auch bei den so genannten Union Mounts war sie als Ideengeber tätig: 1988, also vor beinahe 30 Jahren, überraschte ihr Translucent-Filesystem mit der Möglichkeit, mehrere Verzeichnisse über einen gemeinsamen Verzeichnisnamen anzusprechen. Beim Zugriff über das gemeinsame Verzeichnis blieben die Dateien sämtlicher assoziierter Verzeichnisse sichtbar.

Kernelhacker Werner Almsberger hatte es bereits 1993 in Linux 0.99 mit dem Inheriting Filesystem (IFS) versucht, diese Funktionalität nachzubilden. Das Betriebssystem Plan 9, ein Unix-Nachfahre, implementierte rund ein Jahr später ebenfalls Union Mounts, ebenso wie BSD mit der Version 4.4.

Werner Almsberger entfernte allerdings seinen Code wieder aus dem Linux-Kernel. Seine Begründung: Er sei zu komplex und habe zu viele Abhängigkeiten zu den übrigen Subsystemen. Unter den anderen Versuchen, Union Mounts arbeits- und salonfähig zu machen, kam das 2006 eingeführte AU-FS (Another Union File System) am weitesten.

Allerdings blieb zwei Jahrzehnte lang allen Linux-nativen Union-Mount-Implementierungen die höchste Weihe verwehrt: Linus Torvalds verweigerte dem zwar funktionsreichen, aber eben auch "unleserlichen, unkommentierten" Code [1] die Aufnahme in seinen Standardkernel. Stattdessen adelte Torvalds 2014 das zwischenzeitlich von Novell entwickelte Overlay-Filesystem mit der offiziellen Aufnahme. Seitdem drängt das Overlay-Filesystem die Bedeutung aller früheren Lösungen zurück.

Der Wettstreit um die Vorherrschaft findet nicht in einer Nische statt: Union Mounts stellen eine Basistechnologie für die populären und im Vergleich zur klassischen Virtualisierung Ressourcen-schonenden Container wie Docker oder LXC [2] dar. Die klassische Virtualisierung muss alle Systemteile so oft im Speicher vorhalten, wie virtuelle Maschinen auf dem Hypervisor laufen. Oft genug sind in allen VMs Kernel, Bibliotheken, Verzeichnisstruktur und Systemprogramme komplett identisch, nur die Konfigurationen und die einzelne Applikationen unterscheiden sich.

Diese Ressourcenverschwendung vermeiden Container, indem sie identische Komponenten nur ein Mal im Dateisystem und im Hauptspeicher ablegen. Union Mounts und die Namespaces [2] sind die Techniken, mit denen Containerhosts das gelingt. Dazu legen sie in jedem Container über die Verzeichnisstruktur des Hostsystems eigene leere Verzeichnisse, beispielsweise »/etc«. Jedes so überlagerte Verzeichnis muss nur die Abweichungen – im Wesentlichen Konfigurationen – speichern (Abbildung 1).

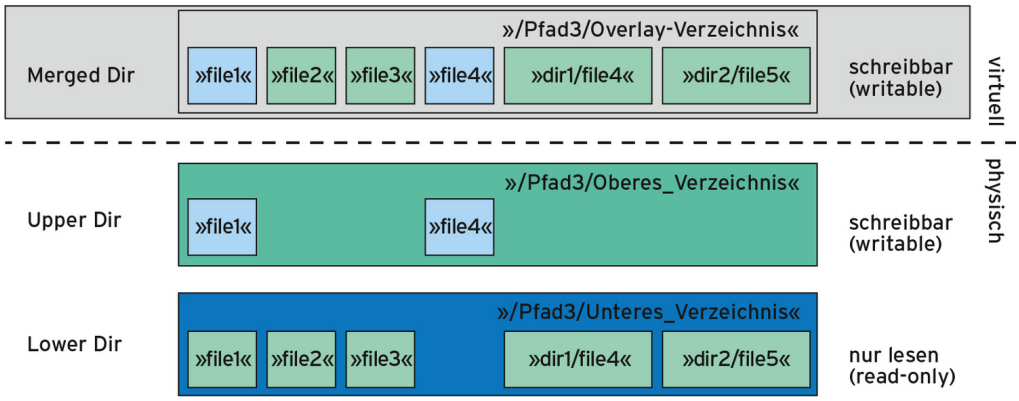


Abbildung 1: Das Overlay-Verzeichnis stellt eine virtuelle Sicht auf die übereinandergelegten Ordner dar.

Verschleiß mindern

Eingebettete Systeme profitieren ebenfalls von der alten Sun-Idee. Geeignet umgesetzt verhindert der Mechanismus allzu häufiges Schreiben auf empfindlichen Flashspeicher, den Embedded-Systeme gerne einsetzen. Nebenbei sorgt er dafür, dass beim abrupten Abschalten des Geräts – auch das ist Embedded-typisch – das Dateisystem auf dem Flashmemory konsistent bleibt.

Um beides zu erreichen, setzt der Embedded-Entwickler das Root-Filesystem auf read-only und legt darüber einen schreibbaren Verzeichnisbaum, der auf einer Art RAM-Disk, einem Tmp-FS, liegt. Schreibzugriffe finden also nur auf den Haupt-, nicht mehr auf den Hintergrundspeicher (Flash, SD-Karte, SSD) statt. Natürlich sind bei dieser Architektur alle Änderungen futsch, falls das System plötzlich stromlos geschaltet wird. Für das dauerhafte Speichern wichtiger Daten muss sich der verantwortungsvolle Entwickler einen alternativen Weg überlegen.

Wissenswertes über Copy-ups …

Beiden Anwendungsfällen – Containern und eingebetteten Systemen – liegt die gleiche Charakteristik zugrunde (Abbildung 1). Das Spiel bestimmen drei Verzeichnisse: ein unteres (lower), ein oberes (upper) und ein gemeinsames (merged). Das untere Verzeichnis ist klassischerweise nur lesbar, das obere lässt sich beschreiben. Im unteren und oberen Ordner sind die Dateien und Unterverzeichnisse physisch abgelegt. Das gemeinsame Verzeichnis schließlich stellt die Vereinigungsmenge der darunterliegenden Verzeichnisse dar.

Solange die Datei- oder Ordnernamen in den gestapelten Verzeichnissen unterschiedlich sind, sind beide erwartungsgemäß sichtbar. Gibt es aber zwei Dateien gleichen Namens in jeder Ebene, bleibt immer nur die obere sichtbar. Sofort ausführbar und damit unproblematisch sind Änderungen an Datei-Inhalten, die im oberen Verzeichnis liegen.

Anders stellt sich die Sache dar, wenn die Änderungen über das gemeinsame Verzeichnis anstehen, die Dateien im unteren, schreibgeschützten Verzeichnis betreffen. Dann findet ein so genanntes Copy-up statt (Abbildung 2). Das Copy-up kopiert die betreffende Datei vom unteren Verzeichnis in das obere und führt die Modifikation dann dort durch.

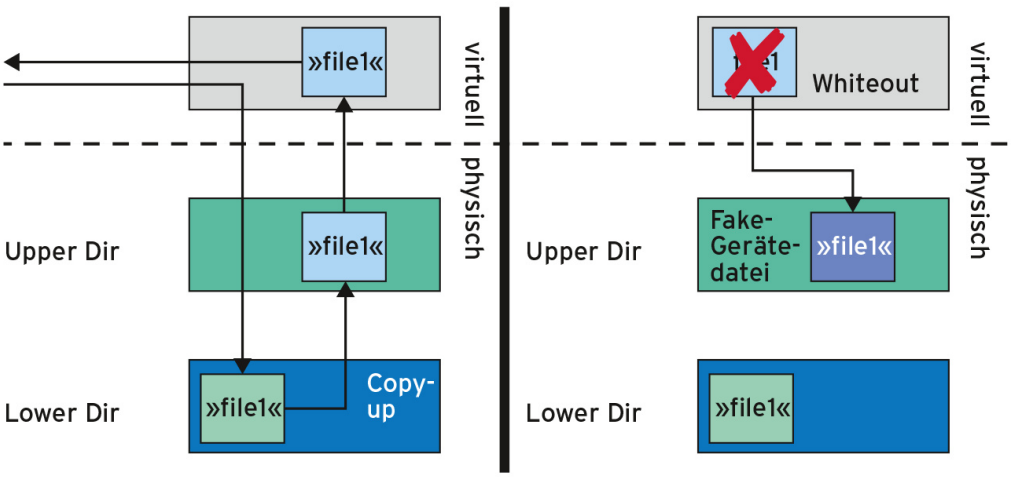


Abbildung 2: Der Mechanismus zum Verändern und Löschen von Dateien im Overlay-Filesystem.

… Whiteouts und undurchsichtige Verzeichnisse

Tricky ist vor allem das Löschen eines Objekts, das physisch im unteren Ordner beheimatet ist. Ist das Objekt eine normale Datei, legt Linux im oberen Ordner eine Fake-Gerätedatei mit dem Namen der zu löschenden Datei an. Die Kernel-Gurus nennen diesen Vorgang Whiteout (Abbildung 2).

Die zeichenorientierte Gerätedatei (Character Device) bekommt die illegale Gerätenummer »0/0«. Die Algorithmen für den Overlay-Zugriff erkennen daran eine gelöschte Datei und blenden sie aus: Im unteren Verzeichnis existiert die Datei weiterhin wie gehabt, im oberen entsteht ein Character-Device gleichen Namens und aus dem gemeinsamen Verzeichnis verschwindet sie.

Handelt es sich bei dem im gemeinsamen Verzeichnis zu löschenden Objekt um ein Unterverzeichnis, sprechen die Entwickler von Opaque Directories, undurchsichtigen Verzeichnissen. Gemäß Kerneldokumentation [3] realisiert Linux die Funktionalität mit Hilfe der erweiterten Datei-Attribute (Extended File Attributes, Xattr). Demnach setzen Entwickler das Attribut »trusted.overlay.opaque« auf »y«.

Falls beim Zugriff auf das gemeinsame Verzeichnis ein derart undurchsichtiges Verzeichnis im Upper Dir existiert, ignoriert Linux Verzeichnisse im unteren Ordner mit dem gleichen Namen. Der Praxistest zeigt aber, dass die Dokumentation nicht mehr aktuell ist. Tatsächlich benutzt Linux heute eine zu normalen Dateien identische Technik: Aus dem Verzeichnis-Eintrag erzeugt das System eine Fake-Gerätedatei mit Gerätenummer »0/0«. Möglicherweise haben die Kernelentwickler diese Änderung vorgenommen, weil nicht alle Dateisystem-Typen die zur Realisierung benötigten erweiterten Datei-Attribute unterstützen.

Mounten in der Praxis

Die beschriebene Wirkungs- und Funktionsweise des Overlay-Filesystems lässt sich gut anhand eines einfachen Beispiels verdeutlichen. Es benötigt allerdings vier Verzeichnisse: Neben »lower.readonly« und »upper.writeable«, die das Beispiel stapeln soll, die Verzeichnisse »merged« und »work«.

Das »work«-Verzeichnis braucht der Kernel aus technischen Gründen, um darüber das Copy-up zu realisieren. Das Verzeichnis »merged« stellt die Inhalte des unteren und oberen Verzeichnisses gemeinsam dar. In den zu stapelnden Verzeichnissen wird das Beispiel eine Reihe von Dateien anlegen. Insbesondere soll es die Datei »drei« sowohl im unteren als auch im oberen Ordner geben.

Zum Overlay-Mounten bekommt der Kernel mit Hilfe der Option »-o« drei Parameter übergeben: »lower« nimmt den Pfad und den Namen des unteren Verzeichnisses entgegen, »upper« spezifiziert Pfad und Namen des oberen Verzeichnisses, und »workdir« schließlich informiert den Kernel über das Arbeitsverzeichnis, das dieser benötigt.

Linux-Systeme erwarten beim Einhängen von Dateisystemen typischerweise den Namen der Gerätedatei (die Partition einer Festplatte) und den Einhängepunkt (Mountpoint). Da das Overlay-Dateisystem ein virtuelles Filesystem ist, das nicht über eine Gerätedatei repräsentiert wird, trägt man beim Mount-Befehl »none« ein, üblich wäre auch »overlay«. Am Ende steht das Zielverzeichnis, im konkreten Fall ist das »merged«.

Der Selbstversuch

Um das Beispiel in die Tat umzusetzen, tippt der Benutzer das Skript »overlay-test.sh« aus Listing 1 entweder in seinen Texteditor oder holt es von [4]. Mit »chmod +x overlay-test.sh« macht er das Skript ausführbar und ruft es per »./overlay-test.sh« auf. Die Ausgabe des Kommandos »ls« in Zeile 15 ist in Listing 2 zu besichtigen. Es zeigt die Inhalte aller Verzeichnisse.

Wie zu erwarten, befinden sich im unteren Ordner die Dateien »eins«, »zwei« und »drei« mit dem Unterverzeichnis »subdir1«, im oberen die Dateien »drei« und »vier«. Das Verzeichnis »merged« zeigt erwartungsgemäß alle Dateien von »eins« bis »vier« und »subdir1« an. Die Datei »drei« liegt dabei in der Version des oberen Verzeichnisses vor und schattet die Datei »drei« aus dem unteren Verzeichnis ab. Veränderungen, die Dateien jetzt in »merged« widerfahren, führen zu unmittelbaren Änderungen im Verzeichnis »upper.writeable«.

Listing 1: *overlay-test.sh*

```
01 #!/bin/bash
02
03 mkdir lower.readonly upper.writeable merged work
04 mkdir lower.readonly/subdir1
05 echo "unteres" >lower.readonly/eins
06 echo "unteres" >lower.readonly/zwei
07 echo "unteres" >lower.readonly/drei
08 echo "oberes" >upper.writeable/drei
09 echo "oberes" >upper.writeable/vier
10
11 sudo mount -t overlay -o \
12   lowerdir=lower.readonly,upperdir=upper.writeable,workdir=wor
13   overlay merged
14
15 ls -l lower.readonly upper.writeable merged work
```

Listing 2: Ausgabe von *./overlay-test.sh*

```
01 lower.readonly:
02 total 16
03 -rw-r--r-- 1 root root    8 Jan 31 20:20 drei
04 -rw-r--r-- 1 root root    8 Jan 31 20:20 eins
05 drwxr-xr-x 2 root root 4096 Jan 31 20:20 subdir1
06 -rw-r--r-- 1 root root    8 Jan 31 20:20 zwei
07
08 merged:
09 total 20
10 -rw-r--r-- 1 root root    7 Jan 31 20:20 drei
11 -rw-r--r-- 1 root root    8 Jan 31 20:20 eins
12 drwxr-xr-x 2 root root 4096 Jan 31 20:20 subdir1
13 -rw-r--r-- 1 root root    7 Jan 31 20:20 vier
14 -rw-r--r-- 1 root root    8 Jan 31 20:20 zwei
15
16 upper.writeable:
17 total 8
18 -rw-r--r-- 1 root root 7 Jan 31 20:20 drei
```

```
19 -rw-r--r-- 1 root root 7 Jan 31 20:20 vier
20
21 work:
22 total 4
23 d--------- 2 root root 4096 Jan 31 20:20 work
```

Technisch anspruchsvoller ist, wie erläutert, das Löschen einer Datei, die sich physisch im unteren Verzeichnis befindet. Abbildung 3 zeigt die Ausgabe des Kommandos »ls«, nachdem die Datei »merged/drei« und das Verzeichnis »merged/subdir1« gelöscht wurden. Im oberen Verzeichnis tauchen die zeichenorientierten Fake-Gerätedateien gleichen Namens mit der Gerätenummer »0/0« auf. Dadurch sind sie im gemeinsamen Verzeichnis ausgeblendet. Zum Aufräumen des Ganzen führt der Benutzer die Kommandos aus Listing 3 nacheinander aus.

Listing 3: Kommandos zum Aufräumen

```
01 sudo umount merged
02 rm -rf lower.readonly
03 rm -rf upper.writeable
04 rm -rf work
05 rm -rf merged
```

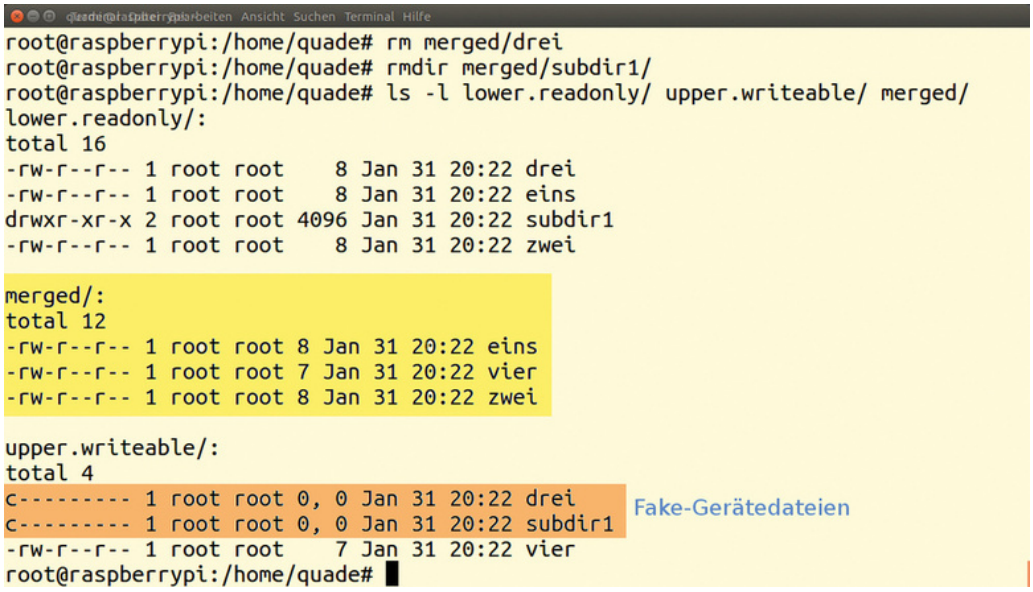


Abbildung 3: Whiteouts und Opaque Dirs sind über Fake-Gerätedateien realisiert.

Gut Ding will Weile haben

Dass Union Mounts vom ersten Ansatz für den Linux-Kernel 1993 bis zur offiziellen Aufnahme in den Kernel 2014 zwei Jahrzehnte brauchten, lag nicht zuletzt an der Komplexität der Aufgabe. Hinzu kam die Linux-typische Vielfalt an Dateisystemen, von denen manche mehr als zwei Schichten stapeln können. Der Kasten "Komplexe Problematik" erläutert einige der technischen Hürden.

Das hier schon erwähnte Überlagern in professionellen Embedded-Systemen zum Schutz des Flashspeichers ist übrigens ebenfalls kein Selbstläufer. Eine künftige Kern-Technik wird sich mit dem Thema auseinandersetzen. (jk)

Komplexe Problematik

Verzeichnisse überlagern und die Vereinigungsmenge aus den Inhalten bilden scheint beim ersten Blick simpel zu sein. Bei genauerer Betrachtung jedoch ist die Realisierung technisch sehr anspruchsvoll und nur mit Kompromissen zu realisieren [5].

So zeigen Overlay-Filesysteme manchmal ein unerwartetes Zeitverhalten, etwa beim Entfernen von vermeintlich leeren Unterverzeichnissen. Denn waren Unterordner vorher mit vielen Dateien aus dem darunterliegenden Verzeichnis gefüllt, die irgendwann im gemeinsamen Verzeichnis gelöscht wurden, verbleiben im darübergelegten Verzeichnis noch die korrespondierenden Fake-Gerätedateien. Die muss der Kernel zeitgleich mit dem Unterverzeichnis erst einmal entfernen. Das kostet Zeit.

Das Umbenennen tritt eine Lawine los

Ein anderer überraschender Effekt tritt auf, wenn jemand ein gut gefülltes Unterverzeichnis umbenennt – eigentlich ein harmloses Unterfangen. Mit Overlay führt es jedoch zu einer opulenten Kopieraktion. Denn beim Rename muss das System das gesamte Unterverzeichnis als Kopie neu erzeugen.

Als nächste Fußangel für die Kernelentwickler liegen die Hardlinks aus. Jeder neue Hardlink, der zusammen mit seinen Artgenossen auf ein und denselben Inode zeigt, liegt mit einer recht hohen Wahrscheinlichkeit in einem anderen Verzeichnis als das Original. Ändert nun jemand über einen der Dateinamen den Inhalt der Datei, muss der Overlay-Algorithmus die Datei per Copy-up in das obere Verzeichnis kopieren. Und zudem: Das muss für sämtliche Hardlinks passieren, wobei der Mechanismus die durch eine unter Umständen aufwändige Suche finden muss.

Auch wenn das untere Verzeichnis schreibgeschützt sein sollte, ist es technisch kein Problem, dort neue Dateien anzulegen, existierende umzubenennen oder Unterordner zu verändern. Linux erlaubt Veränderungen am darunterliegenden Verzeichnis eigentlich nicht, scheint damit aber glücklicherweise in der Praxis dennoch gut umgehen zu können.

Stück für Stück

Kernelentwickler diskutieren immer wieder den problematischen Systemcall »readdir()«, der einen Verzeichnisinhalt ähnlich wie eine große Datei stückchenweise ausliest. Für dieses "stückchenweise" ist das Retten des aktuellen Zustands notwendig, der bei Veränderungen an einem der darunterliegenden Verzeichnisse nicht mehr stimmt. Lösbar wird das Problem durch speicherintensives Caching.

Die Unterschiedlichkeit der Dateisysteme hat den Entwicklern ebenfalls zu schaffen gemacht. NFS beispielsweise erwartet eindeutige, persistente Inodes, die auch einen Reboot überleben. Für ein virtuelles Dateisystem ist das die hohe Schule. Das Overlay-Filesystem im Linux-Kernel setzt dafür eine komplexe Tabelle ein.

Nicht zuletzt bereiten auch einzelne Systemcalls Kopfzerbrechen. Per »mmap()« beispielsweise lässt sich der Inhalt einer Datei als Speicher direkt in den

Adressraum einer Applikation einbinden. Was ist aber, wenn sich zwischenzeitlich der Datei-Inhalt ändert und ein Copy-up notwendig macht? Inkonsistenzen entstehen hier praktisch zwangsläufig.

Infos

1. Valerie Aurora, "Unioning file systems, Implementations, part 2": [https://lwn.net/Articles/327738/]
2. Quade, Kunst, "Container-Virtualisierung", Kern-Technik 55: Linux-Magazin 02/11, S. 104, [http://www.linux-magazin.de/Ausgaben/2011/02/Kern-Technik]
3. Neil Brown, "Overlay Filesystem" (Kernel-Dokumentation): [http://lxr.free-electrons.com/source/Documentation/filesystems/overlayfs.txt]
4. Listings zum Artikel: [http://www.linux-magazin.de/static/listings/magazin/2017/04/kern-technik/]
5. Valerie Aurora, "Unioning file systems: Architecture, features, and design choices": [https://lwn.net/Articles/324291/]

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.