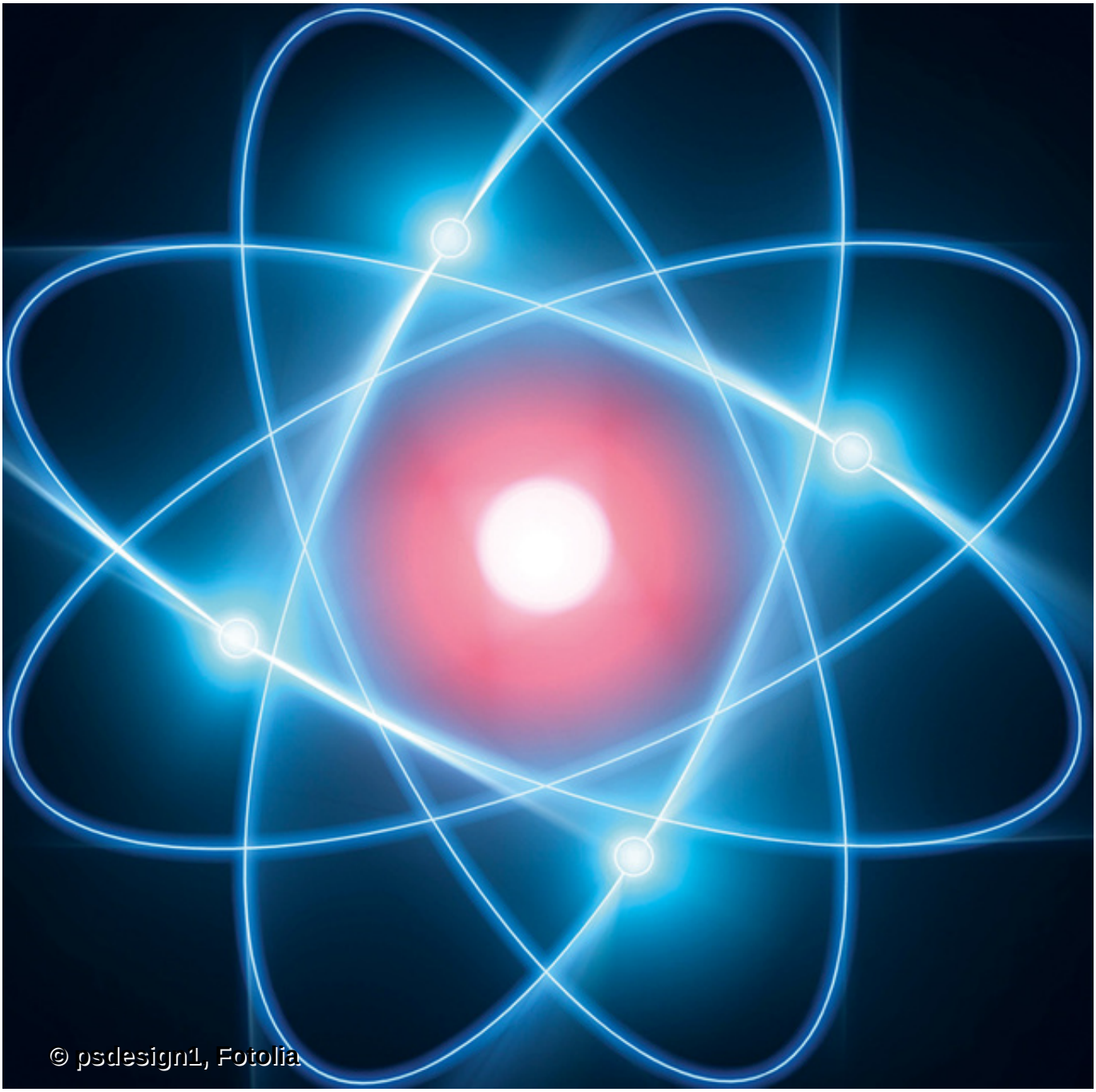
Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 92

Kern-Technik

Viele Schreibzugriffe und hartes Stromabschalten mögen Flashspeicher gar nicht gerne. Wer der SD-Karte seines Raspberry Pi ein langes Leben schenken und Anwendungen und Anwender trotzdem nicht reglementieren will, garniert sein Himbeertörtchen mit einem Overlay-Filesystem. Hier die Backanleitung.

Eva-Katharina Kunst, *Jürgen Quade*



Die klassische Festplatte ist ein Auslaufmodell, kein Zweifel. Noch punktet sie mit ihrem Preis und üppigem Speichervolumen, doch den ungleich besseren Zugriffszeiten der

Flashspeicher hat die Festplatte nichts entgegenzusetzen. Hinzu kommt die mechanische Anfälligkeit ihrer bewegten Teile. Smartphones, Tablets oder die Gruppe der Microcomputer à la Raspberry Pi setzen schon jetzt rein auf Flash, Letzterer in Form von leicht handhabbaren SD-Karten. Das Betriebssystem für den Minicomputer ist über den PC einfach auf die SD-Karte zu transferieren, und das im Vergleich zur Festplatte reduzierte Speichervolumen erweist sich für kaum einen Einsatzzweck als zu gering.

Wer aber länger und häufiger mit einem Raspberry Pi arbeitet, diesen vielleicht sogar im 24/7-Betrieb nutzt, bekommt eine eingeschränkte Zuverlässigkeit des Speichermediums zu spüren.

Flashspeicher kämpfen auch 2017 mit einer limitierten Zahl von Schreibzyklen. Linux reagiert auf diese Problematik mit Flash-optimierten Spezial-Dateisystemen, zum Beispiel F2fs. Damit lässt sich tatsächlich bei SSDs eine hohe Zuverlässigkeit erreichen. Mit den (billigen) SD-Karten, auf denen klassisch die alten Dateisysteme (Ex)-FAT und Ext 4 liegen, ist das nicht so. Da gibt es keinen besseren Schutz, als Schreibzugriffe komplett zu meiden oder wenigstens zu minimieren.

Für Linux ist das prinzipiell kein Problem. Der Kernel selbst muss zunächst nur lesen. Erst in Kombination mit dem Userland und mit den Applikationen treten Schreibzugriffe auf – und das nur, wenn es explizit passieren muss. Beispielsweise legen Unix-Systeme für jede Datei drei Zeitstempel ab: den Erzeugungszeitpunkt (Creation), den Zeitpunkt der letzten Änderung (Modification) und den des letzten Zugriffs (Access).

Jedes Lesen einer Datei löst daher einen Schreibzugriff auf den zugehörigen Zeitstempel aus – eigentlich, denn die Linux-Entwickler haben diese Abhängigkeit vor einigen Jahren zurückgefahren und aktualisieren standardmäßig den Zugriffszeitstempel nur beim allerersten lesenden Zugriff. Das Verhalten bekommt der Kernel beim Mounten eingeimpft (Tabelle 1).

Tabelle 1: Einfluss des Mount-Kommandos auf den Zugriffszeitstempel

Mount-Option	Verhalten
»relatime«	Das erste Lesen nach einem Write aktualisiert den Zeitstempel.
»noatime«	Der Zeitstempel wird nie aktualisiert.
»atime«	Bei jedem Lesen wird der Zeitstempel aktualisiert.

Robuste Systeme

Für Systeme, die weitgehend unbeaufsichtigt laufen, reicht das Reduzieren von Schreibzugriffen jedoch nicht aus. Wer sein Internet-of-Things- oder Industrie-4.0-Gerät schützen will – der Fachmann spricht von Ruggedizing –, hängt die Dateisysteme nur lesend ein [1]. Allerdings mucken einige nun schreibgehemmte Applikationen auf und verweigern den Betrieb.

Auch der Ansatzpunkt, die SD-Karte zu partitionieren und die kritische Boot- und Rootpartition nur lesend, eine unkritische Homepartition für die Applikationen zusätzlich schreibend einzuhängen, führt nicht zum Ziel. Hier funkt die Hardware des Flashspeichers quer: Das so genannte Wear Leveling macht nämlich nicht vor Partitionsgrenzen halt und

tauscht Sektoren zwischen den Partitionen, um eine gleichmäßige Verteilung der Schreibzugriffe zu erreichen.

Für ein zuverlässiges System geht also kein Weg an einem rein lesenden Zugriff vorbei. Folglich sind die Schreibzugriffe umzulenken, beispielsweise auf einen zweiten Flashspeicher (USB-Stick) oder eben – falls ausreichend vorhanden – auf den Hauptspeicher. Für Letzteres bietet der Linux-Kernel gleich zwei Techniken: Erstens das Initram-FS [2] und zweitens das in der letzten Kern-Technik vorgestellte Overlay-Filesystem [3].

Filesysteme im RAM

Der Initram-FS-Ansatz benötigt den Flashspeicher nur während des Bootprozesses, um den Kernel und ein Root-Filesystem in den Speicher zu schubsen (Abbildung 1). Das Root-Filesystem ist als Initram-FS ausgeprägt, das sämtliche Dateien und Verzeichnisse im Hauptspeicher abbildet. Ein klassisches Dateisystem wie Ext 4 gibt es nicht mehr, der Kernel greift ohne Umwege besonders effizient auf die Daten zu.

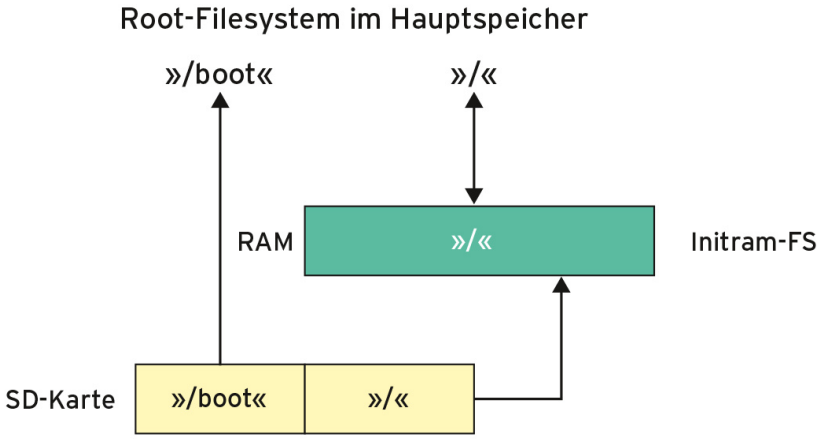


Abbildung 1: Schlanke Selbstbau-Linuxe können beim Hochfahren ihr Root-Filesystem komplett in den Hauptspeicher kopieren.

Der einzige Nachteil besteht im limitierten Platzangebot des Hauptspeichers. Systeme mit komplexen grafischen Oberflächen oder umfangreichen Datenbanken sind so nicht realisierbar. Wer jedoch sein System selber baut, etwa mit Hilfe von Buildroot [4], schafft zudem noch den Raum, um einen einfach handhabbaren Update-Mechanismus zu etablieren. Allerdings gehen bei Stromausfall neue Daten verloren und solche, die sich zwischenzeitlich geändert haben.

Für komplexere Systeme ist der Hybrid-Ansatz geeignet. Dann liegen die Daten wie bisher auf dem Flashspeicher, schreibende Zugriffe fängt aber ein Overlay-Filesystem ab, die geänderten Daten landen entweder auf einem Filesystem im Hauptspeicher (Abbildung 2) oder auf einem USB-Stick (Abbildung 3).

Wer im Internet nach Anleitungen zur Realisierung sucht, wird zwar schnell fündig (zum Beispiel [5]), ist bei der Umsetzung aber selten erfolgreich [6]. Die saubere Integration in den Bootprozess ist nicht trivial. So ist vor dem ersten schreibenden Zugriff für das Mission-critical-Rootverzeichnis auf die SD-Karte ein Overlay nötig. Eine funktionierende Lösung in Form eines Skripts stammt ursprünglich aus den Entwicklerhänden von Axel Heider und ist unter [7] zu finden, die Installation beschreibt der Kasten "Overlay auf dem Raspberry Pi einrichten" detailliert.

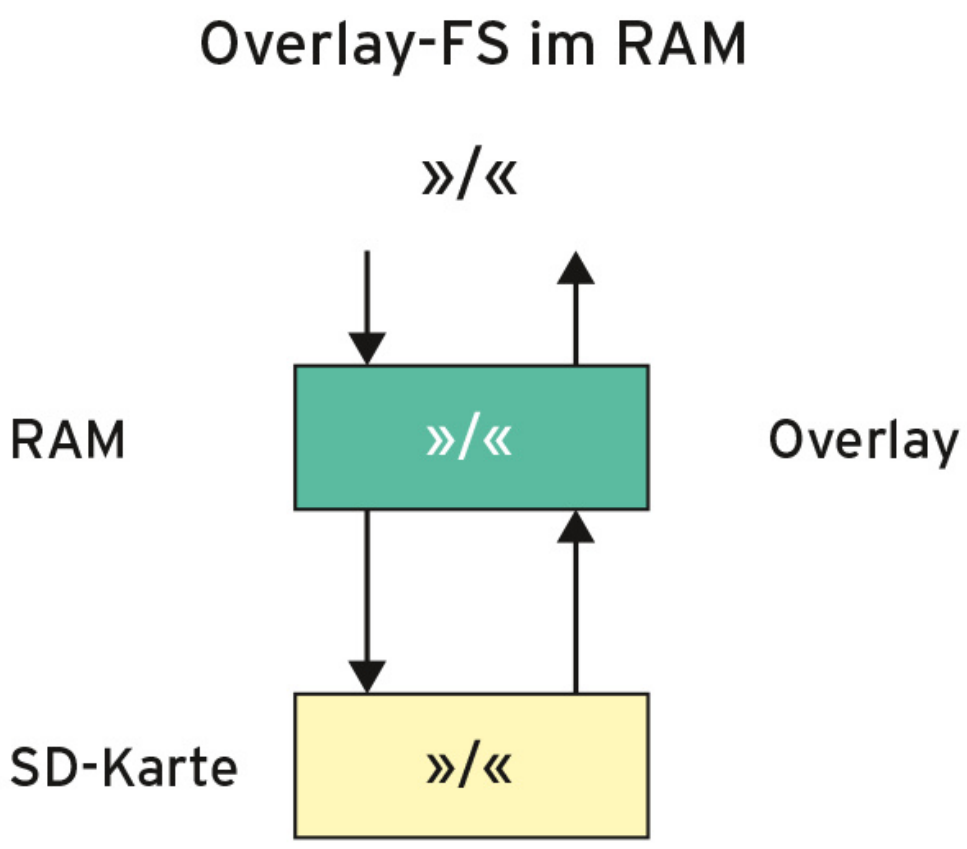


Abbildung 2: **Wer ein System mit Speicher-genügsamen Applikationen betreibt, fährt oft gut mit einem Overlay im RAM.**

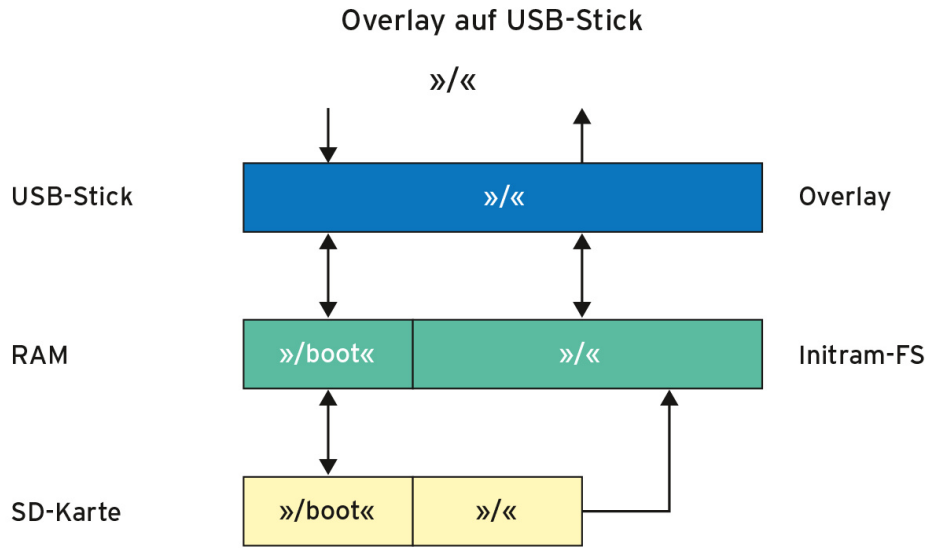


Abbildung 3: **Für komplexe Systeme, auch für Raspbian, eignet sich der Ansatz, das Overlay auf einem USB-Stick abzulegen.**

Overlay auf dem Raspberry Pi einrichten

Falls noch nicht geschehen, installiert der Raspi-Besitzer Raspbian (in der Version »2017-03-02-raspbian-jessie«) auf eine SD-Karte. Außerdem richtet er das System ein, indem er

- das Passwort mit dem Kommando »passwd« für den Default-User »pi« ändert,
- den SSH-Server per »raspi-config« aktiviert,

- die Systemsoftware aktualisiert: »apt-get update; apt-get dist-upgrade« und
- eventuell einen neuen User anlegt und mit Admin-Rechten ausstattet (»adduser loginname; adduser loginname sudo«).

Nun loggt sich der Benutzer in das vorbereitete System ein und wird durch Eingabe von »sudo su« zum Superuser. Jetzt geht es ans Konfektionieren des Initram-FS:

- Damit der Kernel das Modul »overlay« lädt, trägt der Superuser den Modulnamen in die Datei »/etc/initramfs-tools/modules« ein:

```
echo "overlay" >> /etc/initramfs-tools/modules
```

- Für die eigentliche Aktivierung des Overlay muss er das Skript »root-ro« an der richtigen Stelle in das Initram-FS aufnehmen und ausführbar machen:

```
cd /etc/initramfs-tools/scripts/init-bottom
wget https://gist.github.com/niun/34c945d70753fc9e2cc7/raw/3d6(
chmod +x root-ro
```

Danach geht es weiter mit dem Generieren des Initram-FS:

```
update-initramfs -c -k $(uname -r)
```

Das Initram-FS ist damit zwar gebaut, liegt aber noch im Bootverzeichnis »/boot/« unter einem unhandlichen Namen. Der Aufruf

```
mv initrd.img-$(uname -r) initrd7.img
```

benennt es daher um. Der letzte Schritt vor dem Reboot aktiviert das Initram-FS auf dem Raspberry Pi, indem

```
cd /boot
echo "initramfs initrd7.img" >>config.txt
```

den Bootloader anpasst. Achtung: Diese Konfigurationsoption verlangt kein Gleichheitszeichen zwischen Optionsname und Wert.

Fertig! Mit dem nächsten Reboot ist der Zugriff auf die SD-Karte nur noch lesend möglich.

Booten im Hauptspeicher

Um das Overlay vor dem ersten schreibenden Zugriff aufzusetzen, kommt das bereits erwähnte Initram-FS als Sprungbrett zu Ehren. Das für den Raspberry Pi noch zu generierende Initram-FS legt der Raspberry-Besitzer – wie den Kernel auch – auf der Bootpartition der SD-Karte ab, und der Bootvorgang wird ihn vom Bootloader in den Hauptspeicher transferieren.

Der Kernel startet aus dem im Hauptspeicher liegenden Initram-FS das Programm »init«. Dies lädt normalerweise die wichtigsten Treiber, schaltet auf das auf der Festplatte beziehungsweise auf der SD-Karte liegende Orginal-Root-Filesystem um und mutiert schließlich zum dort abgelegten (Standard-)Init.

Drei Root-Filesysteme

Für ein robustes (ruggedized) System klinkt sich das vorgestellte Setup in den ersten Initprozess ein, und zwar nachdem Init – wie es seine Art ist – das Root-Filesystem ohnehin erst einmal read-only eingehängt hat. Init startet danach das Skript »root-ro«, welches nach Auswertung der Bootparameter das Overlay-Filesystem etabliert. Da das Skript mit 290 Zeilen fürs Abdrucken zu groß ist, reflektiert Listing 1 nur die entscheidenden (Mount)-Kommandos zum Anlegen des Overlay.

Zum besseren Verständnis des komplizierten Vorgangs sollte man wissen, dass drei (Root-)Dateisysteme beteiltigt sind: Als erstes das zum Bootzeitpunkt aktive Root-Filesystem des initialen RAM-Filesystems, als zweites das Root-Filesystem auf der SD-Karte, auf das nur lesende Zugriffe erlaubt sein sollen, und als drittes das Overlay-Filesystem, das aus Sicht des laufenden Betriebssystems das eigentliche Root-Filesystem werden wird.

Das Skript »root-ro« erzeugt dazu nach Auswertung der Bootparameter im (noch aktiven) Init-Filesystem ein Verzeichnis als Mountpoint für das Filesystem auf der SD-Karte (»/mnt/root-ro/«) und ein Verzeichnis als Mountpoint für das neue Overlay-Filesystem (»/mnt/root-rw/«). Da das Initsystem die SD-Karte bereits mit lesendem Zugriff eingehängt hat, verschiebt das Kommando

```
mount -o move /root /mnt/root-ro
```

den Mountpoint der SD-Karte auf das neu kreierte Verzeichnis »/mnt/root-ro/« (Listing 1). Dieses Verschieben macht den Weg frei, um das Overlay auf das von Init verwendete Verzeichnis »/root/« für die SD-Karte umzuhängen. Ab diesem Zeitpunkt würden schreibende Zugriffe auf das Root-Filesystem bereits im Hauptspeicher landen (Abbildung 4).

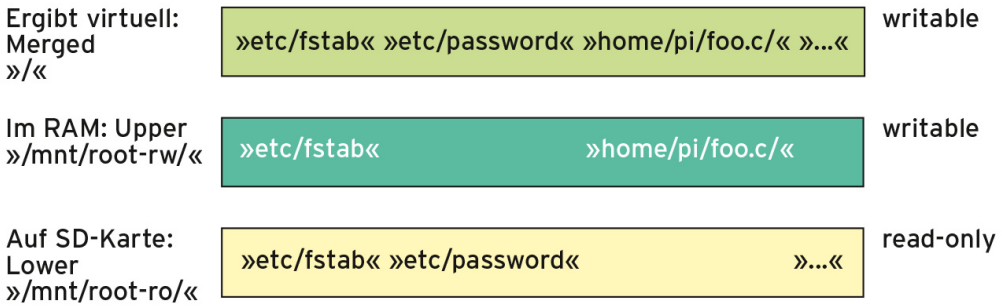


Abbildung 4: Die SD-Karte ist die untere, ein Temp-Filesystem die obere Schicht des Overlay-Filesystems.

Listing 1: *root-ro* (modifizierter Auszug)

```
01 [...]
02 ROOT_RO=/mnt/root-ro
03 ROOT_RW=/mnt/root-rw
04 ROOT_RW_UPPER=${ROOT_RW}/upper
05 ROOT_RW_WORK=${ROOT_RW}/work
06 rootmnt=/root
07 [...]
08 mount -t tmpfs tmpfs-root ${ROOT_RW}
09 [...]
10 # root is mounted on ${rootmnt}, move it to ${ROOT_RO}.
11 mount -o move ${rootmnt} ${ROOT_RO}
12 [...]
```

```
12 [...]
13 # mount virtual fs ${rootmnt} with rw-fs ${ROOT_RW} on top or
14 mount -t overlay -o \
15         lowerdir=${ROOT_RO},upperdir=${ROOT_RW_UPPER},workdir=
16         overlay ${rootmnt}
17 [...]
18 # move mount from ${ROOT_RO} to ${rootmnt}${ROOT_RO}
19 mount -o move ${ROOT_RO} ${rootmnt}${ROOT_RO}
20 # move mount from ${ROOT_RW} to ${rootmnt}${ROOT_RW}
21 mount -o move ${ROOT_RW} ${rootmnt}${ROOT_RW}
```

Trick mit der Fstab

Problematisch ist allerdings, dass Init nach getaner Arbeit in der ersten Stufe das Init-Filesystem wieder freigibt und als neues Root-Filesystem das Verzeichnis »/root/« verwendet. Bei der Freigabe fallen damit aber auch die von »root-ro« dort angelegten Verzeichnisse »/mnt/root-ro/« und »/mnt/root-rw/« weg. Daher muss das Skript sie vorher noch nach »/root/« verschieben, also in das neue Temp-Filesystem (siehe Kasten "Temporäres Filesystem").

Jetzt gilt es, den weiteren Bootverlauf anzupassen, denn Linux wird gemäß »/etc/fstab« – die steuert beim Bootprozess das Einhängen der Dateisysteme – die Rootpartition von der SD-Karte lesend und schreibend mounten. Der Trick, um das zu verhindern: Im Overlay schreibt das Skript die Fstab neu und modelt bei der Kopie den Eintrag für die Rootpartition (»/dev/mmcblk0p2«) auf read-only um (Abbildung 5).

```
quade@raspberrypi:~ $ cat /mnt/root-ro/etc/fstab
proc            /proc           proc    defaults          0       0
/dev/mmcblk0p1  /boot           vfat    defaults          0       2
/dev/mmcblk0p2  /               ext4    defaults,noatime  0       1
# a swapfile is not a swap partition, no line here
#   use  dphys-swapfile swap[on|off]  for that
quade@raspberrypi:~ $ cat /mnt/root-rw/upper/etc/fstab
#
#  This fstab is in RAM, the real one can be found at /mnt/root-ro/etc/fstab
#  The original entry for '/' and all swap files have been removed.  The new
#  entry for the read-only the real root fs follows. Write access can be
#  enabled using:
#    sudo mount -o remount,rw /mnt/root-ro
#  re-mounting it read-only is done using:
#    sudo mount -o remount,ro /mnt/root-ro
#

/dev/mmcblk0p2 /mnt/root-ro ext4 ro,relatime,data=ordered 0 0

#
#  remaining entries from the original /mnt/root-ro/etc/fstab follow.
#
proc            /proc           proc    defaults          0       0
/dev/mmcblk0p1  /boot           vfat    defaults          0       2
quade@raspberrypi:~ $
```

Abbildung 5: Während des Bootens wird die originale Fstab (oben) modifiziert (unten).

Temporäres Filesystem

Linux hat mit Kernel 2.4 die klassische RAM-Disk durch das temporäre Filesystem Tmp-FS ersetzt. Dies hält alle Daten im Hauptspeicher, auch im ausgelagerten. In Abgrenzung zur RAM-Disk benötigt ein Tmp-FS keinen Dateisystemtreiber, da es

direkt auf die internen Schnittstellen für den Dateizugriff aufsetzt. Damit arbeitet es effizienter.

Linux nutzt das Tmp-FS in diversen Varianten extensiv (Abbildung 6). Als Initram-FS beispielsweise beherbergt es ein rudimentäres Userland. Dort liegen insbesondere Programme zur Systemkonfiguration und zum Laden von Treibern.

Ubuntu setzt es als so genanntes Early Userland ein, um den Kernel ohne Treiber – die befinden sich nämlich dann im Initram-FS – auszuliefern. Nach dem Laden der Treiber wechselt Ubuntu auf die eigentliche Rootpartition über und löscht das Early Userland wieder aus dem Hauptspeicher.

Dadurch verbleiben keine überflüssigen Treiber oder Konfigurationen im RAM. Für den User ist der Vorgang unsichtbar. Gibt es aktualisierte oder zusätzliche Treiber, muss Ubuntu im Übrigen nur das Initram-FS neu installieren, nicht den Kernel selbst.

Jetzt funktioniert's

Mit diesen Änderungen ist die eigentliche Arbeit erledigt und das System kann normal hochfahren. Dazu gehört, dass das vom Initram-FS vorbereitete Rootverzeichnis »/root/« per Kommando »switch_root« zum neuen, letztgültigen Wurzelverzeichnis (»/«) wird und dass Linux dank modifizierter Fstab das Root-Filesystem der SD-Karte auch nur read-only einhängt (beziehungsweise es eingehängt bleibt).

Das vorgestellte Setup mountet die Bootpartition »/dev/mmcblk0p1« (auf »/boot/«) übrigens lesend und schreibend. Das ist zunächst unkritisch, da auf diese Partion im laufenden Betrieb keine Schreibzugriffe stattfinden. Andererseits müssen für den rein lesenden Zugriff in der orginalen Fstab nur die Mountoptionen »defaults« durch »ro,noatime« getauscht werden.

Nach dem Reboot zeigt der Mountbefehl in Abbildung 6 den Erfolg an: Die Rootpartiton ist nur lesend (»ro«) im Verzeichnis »/mnt/root-ro/« eingehängt, das Root-Filesystems selbst als Overlay mit Schreibrechten.

Abbildung 6: Erfolg! Das Root-Filesystem ist als Overlay gemountet.

Über den Pfad »/mnt/root-ro/« gelingt es übrigens, jederzeit auf die Orginaldateien lesend zuzugreifen (wie in Abbildung 5), die nachträglichen Änderungen sind unter »/mnt/root-rw/« zu finden. So listet »ls -R /mnt/root-rw/upper« rekursiv die dort abgelegten Dateien auf. Mittels »du -sh /mnt/root-rw« lässt sich feststellen, wie viel Hauptspeicher das Overlay-Filesystem belegt.

Dies sind nach dem Booten gut 100 MByte von den rund 450 zur Verfügung stehenden. Denn Linux reserviert beim Anlegen eines Temp-Filesystems rund die Hälfte des vorhandenen Hauptspeichers. Da der Raspberry Pi 3 etwa 1 GByte RAM hat, sind das 463 MByte. Wer mehr braucht, ergänzt in dem Skript »root-ro« die Zeile 166 um »-o size=550M«, wobei »550M« für 550 MByte steht und durch die benötigte Größe zu ersetzen ist. Im Ganzen sieht die Zeile dann so aus:

```
mount -t tmpfs -o size=550M \
      tmpfs-root ${ROOT_RW}
```

Aber Vorsicht: Mehr Speicher für das Overlay-Filesystem bedeutet zugleich weniger Speicher für die Anwendungen. Wer also mit größeren Datenbanken hantiert oder mit anderen Applikationen, die viele Daten schreiben, muss nach Alternativen suchen. So könnte er die Daten auf ein Netzwerklaufwerk oder einen USB-Stick transferieren. Letzterer eignet sich freilich auch als Lagerstätte für ein Overlay-Filesystem (Abbildung 3).

Häufige Schreibvorgänge ausdünnen

Da der Raspberry Pi jetzt ruggedized ist, sollte dessen Besitzer ihn noch für den Langzeitbetrieb sichern. So schreibt der Logdaemon »rsyslogd« im Moment noch regelmäßig Logdateien unter »/var/log/«. Diese lassen sich beispielsweise per »logrotate« in Schranken halten. Alternativ deinstalliert der Admin Rsyslogd per »dpkg --purge rsyslog« und ersetzt ihn durch die Busybox-Variante (Kommando »apt-get install busybox-syslogd«). Diese behält die Lognachrichten im RAM und stellt mit »logread« ein passendes Werkzeug zum Log-Lesen bereit.

Ein absolutes No-Go für das Ruggedized System ist Swapping. Beim Raspberry Pi ist der Auslagerungsmechanismus normalerweise auch abgeschaltet. Wer sichergehen will, entfernt ihn per »apt-get purge dphys-swapfile« komplett.

Wer mag, gibt alternativ dem Kernel Parameter mit auf den Weg, die der beim Booten auswertet. Die Parameter gehören in der Datei »/boot/cmdline«. Ein dort eingetragenes »noswap« setzt Swapping ebenfalls außer Gefecht. Ebenfalls hilfreich: »fastboot« überspringt den Filesystemcheck, der in einem reinen Lesebetrieb keinen Sinn ergibt. Die Option »ro« schließlich sorgt ebenfalls dafür, dass der Kernel nur read-only mountet.

Updates würden ins Leere laufen

Wichtig: Mit dem Overlay finden alle Änderungen im Filesystem im RAM statt. Mit dem nächsten Bootvorgang gehen diese verloren. Sollen Wartungs- beziehungsweise Konfigurationsarbeiten längerfristig Wirkung entfalten, muss der Besitzer sein System erst ohne Overlay-Mechanismus booten. Dazu kommentiert er in der Datei »/boot/config.txt« nur die letzte Zeile (»initramfs initrd7.img«) durch Einfügen eines Hashtag »#« am Zeilenanfang aus. Das sonst so essenzielle Herunterfahren von Linux darf man sich übrigens im gerade noch aktiven robusten Modus sparen und schadlos einfach den Strom abschalten.

Mit dem nächsten Reboot kehrt nun das normale, mit Schreibrechten auf die SD-Karte versehene System zurück. Aktualisierungen, beispielsweise mittels »apt-get dist-upgrade«, sind in diesem Zustand persistent. Sind alle Arbeiten erledigt, entfernt der Sysadmin in der letzten Zeile von »/boot/config.txt« den Hashtag am Zeilenanfang wieder. Diesmal muss er das System sauber herunterfahren, und die SD-Karte arbeitet mit dem nächsten Start wieder im Schongang.

Wegen kleiner Änderungen, die zum Beispiel ein Backup bewirkt, das System rebooten zu müssen, erscheint nicht sonderlich elegant. Und tatsächlich lässt sich das System auch live in den SD-Karten-Schreibmodus setzen. Dafür hängt Root das read-only eingehängte Filesystem neu mit Schreibrechten (read-write) ein:

```
sudo mount -o remount,rw /mnt/root-ro
```

Das Backup selbst kann Rsync erledigen. Einige Dateien und Ordner, zum Beispiel »/etc/fstab«, »/var/«, »/tmp/«, sollte ein Restore besser aussparen. Ein

```
sudo mount -o remount,ro /mnt/root-ro
```

nach dem Schreibvorgang macht alles wieder rückgängig.

Hart abschalten ohne Reue

So oder so: Bevor er sein haltbar gemachtes Himbeertörtchen in den Dauerbetrieb überführen kann, hat der Besitzer die Speicherauslastung über einen längeren Zeitraum zu beobachen und eventuelle Speicherfresser zu identifizieren und zu beseitigen. Bleibt der Memory-Footprint aber konstant, spricht nichts mehr gegen den Einsatz als eingebettetes Ruggedized System. Und damit etwas sonst Verbotenes ungestraft tun zu dürfen – zack: Stecker raus! – ist für den Raspberry-Pi-Fan das Sahnehäubchen. *(jk)*

Infos

1. Michael Kofler, "Raspbian-Lite für den readonly Betrieb konfigurieren": [https://kofler.info/raspbian-lite-fuer-den-read-only-betrieb/]
2. Quade, Kunst, "Kern-Technik", Folge 39 (über Initram-FS): Linux-Magazin 05/08, S. 98
3. Quade, Kunst, "Kern-Technik", Folge 91 (über Overlay-Filesysteme): Linux-Magazin 04/17, S. 78
4. Quade, "Embedded Linux lernen mit dem Raspberry Pi": Dpunkt-Verlag, 1. Auflage, 2014
5. "Raspbian with Read-only Root": [https://www.raspberrypi.org/forums/viewtopic.php?f=63&t=161416]
6. "Raspberry Pi Forum": [https://www.raspberKastenrypi.org/forums/viewtopic.php?f=63&t=145084]
7. Axel Heider, "Read-only Root-FS with overlayfs for Raspian": [https://gist.github.com/niun/34c945d70753fc9e2cc7]

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.