Was ist eigentlich dieses initramfs?

Niklas Rother

Geo++ GmbH

Chemnitzer Linux Tage, 23.03.25

```
update-initramfs: Generating /boot/initrd.img-6.8.0-51-generic
```

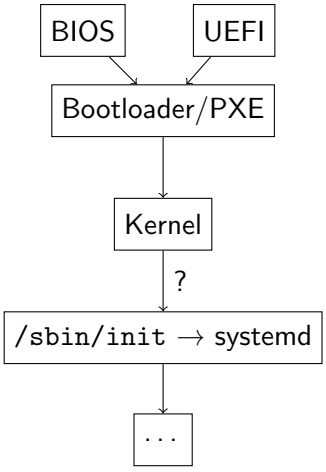
Gliederung

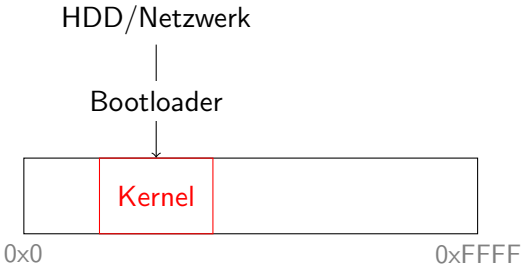
1 Linux Boot-Prozess

2 Das initramfs

3 Eigene Skripte einbinden

Linux Boot-Prozess





Warum initramfs?

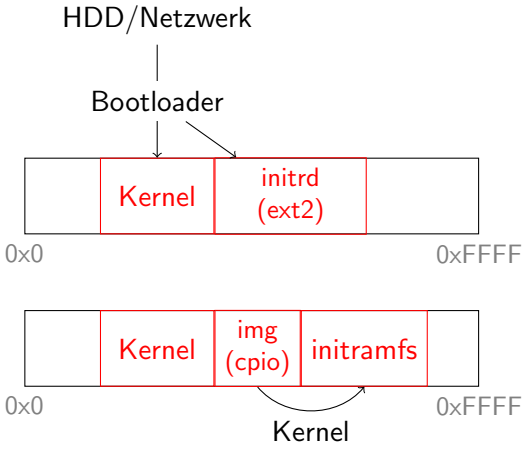
- Problem: Vorher kommt `/sbin/init`?
  - Root-Device (`/`) muss eingebunden sein
- Früher: Kernel Commandline: `root=/dev/sda`
- Aber: Die Welt wird komplizierter

Warum initramfs?

- Viele Möglichkeiten für das Root-Device (/):
  - Lokale Festplatte
  - Verschlüsselte Partition
  - NFS-Freigabe
  - Software-RAID
  - iSCSI-Target
  - Overlayfs
  - All of the above, combined
- Zu viel, um alles in der Kernel einzubauen

Historie

- Lösung: Kernel startet immer von einem minimalen Dateisystem
  - das bindet das echte Root-Device ein
  - lädt dabei ggf. Kernel-Module
  - unabhängig vom Kernel anpassbar
- Seit Kernel 2.4: initrd
  - Initial ram disk
  - Virtuelle Blockdevice im RAM, mit ext2
  - Feste Größe, umständlich
- Seit Kernel 2.6: initramfs
  - cpio-Archiv
  - Wird beim Start entpackt
  - Flexible Größe, kein Dateisystem nötig



Gliederung

1 Linux Boot-Prozess

2 Das initramfs

3 Eigene Skripte einbinden

Was steckt drin?

- `unmkinitramfs -v /boot/initrd.img ~/initramfs/`
- Ubuntu 24.04:
  - `/boot/initrd.img`: 65 MB (gepackt)
  - 447 Ordner, 1554 Dateien → gar nicht so wenig!

```
rother@Rother-Desktop:~/initramfs$ grep -R FIXME .
./main/scripts/nfs: # FIXME This needs error checking
./main/scripts/local: # FIXME This has no error checking
./main/scripts/local: # FIXME This has no error checking
```

Inhalt initramfs

- bin/ → usr/bin/
- conf/
- cryptroot/
- etc/
- **init**
- lib/ → usr/lib/
- lib64/ → usr/lib64/
- lib.usr-is-merged/ → usr/lib.usr-is-merged/
- run/
- sbin/ → usr/sbin/
- **scripts/**
  - functions
  - init-bottom/
  - init-premount/
  - init-top/
  - local
  - local-block/
  - local-bottom/
  - local-premount/
  - local-top/
  - nfs
  - panic/
- usr/
- var/

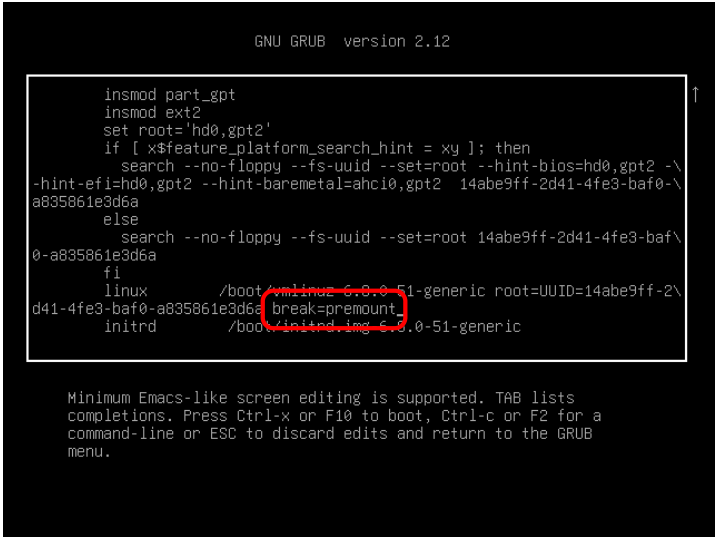
Ordner `scripts/`

- init-top/
  - 00_mount_efivarfs
  - all_generic_ide
  - blacklist
  - console_setup
  - framebuffer
  - udev
- init-premount/
  - cloud-initramfs-dyn-netconf
  - plymouth
- local-top/
  - cryptopensc
  - cryptroot
  - iscsi
- local-block/
  - crgptroot
  - mdadm
- local-premount/
  - btrfs
  - fixrtc
  - ntfs_3g
  - resume
- local-bottom/
  - cryptgnupg-sc
  - cryptopensc
  - cryptroot
  - iscsi
  - mdadm
  - ntfs_3g
- init-bottom/
  - cloud-initramfs-dyn-netconf
  - copymods
  - lvm2
  - plymouth
  - udev

Aufbau `init`-Skript

- `/init`: Shell-Skript
  - Auch in Zeiten von systemd ist der Anfang aller Dinge immer noch ein Shell-Skript...
- `man initramfs-tools`
- Verschiedene Phasen:
  - top
  - modules
  - premount
  - mount
  - mountroot
  - bottom
  - init
- Debugging: Kernel Commandline: z.B. `break=premount`
- Skripte interaktiv ausprobieren

break-Option im Bootloader setzen



Debug-Shell im initramfs

```
[    3.075515]  sda: sda1 sda2
[    3.077403] sd 2:0:0:0: [sda] Attached SCSI disk
[    3.682516] e1000 0000:00:03.0 eth0: (PCI:33MHz:32-bit) 08:00:27:fc:d3:6d
[    3.685983] e1000 0000:00:03.0 eth0: Intel(R) PRO/1000 Network Connection
[    3.691011] e1000 0000:00:03.0 enp0s3: renamed from eth0
Begin: Loading essential drivers ... [    5.039828] raid6: sse2x4   gen() 18710
MB/s
[    5.070813] raid6: sse2x2   gen() 19716 MB/s
[    5.103848] raid6: sse2x1   gen() 15588 MB/s
[    5.106176] raid6: using algorithm sse2x2 gen() 19716 MB/s
[    5.135778] raid6: .... xor() 13045 MB/s, rmw enabled
[    5.138564] raid6: using ssse3x2 recovery algorithm
[    5.142755] xor: measuring software checksum speed
[    5.146344]    prefetch64-sse  : 24528 MB/sec
[    5.149381]    generic_sse     : 18017 MB/sec
[    5.151888] xor: using function: prefetch64-sse (24528 MB/sec)
[    5.156186] async_tx: api initialized (async)
done.
Spawning shell within the initramfs


BusyBox v1.36.1 (Ubuntu 1:1.36.1-6ubuntu3.1) built-in shell (ash)
Enter 'help' for a list of built-in commands.

(initramfs) _
```

Gliederung

1 Linux Boot-Prozess

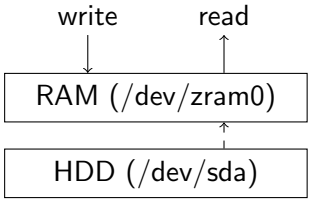
2 Das initramfs

3 Eigene Skripte einbinden

Eigene Skripte einbauen

- Distributions-Skripte: `/usr/share/initramfs-tools/scripts/`
- Eigene Anpassungen: `/etc/initramfs-tools/scripts/`
  - ein Unterorder je nach Phase (init-top, local-bottom, etc.)
- Danach: `update-initramfs -u`

- Beispiel: Unkaputtbares Linux mit dm-snapshot
  - Overlay über dem eigentlichen Root-Device
  - Alle Schreibzugriffe landen im RAM
  - Nach einem Neustart alles wieder frisch



Vorgehensweise Overlay-Skript

1. Module dem initramfs hinzufügen
   - `/etc/initramfs-tools/modules` bearbeiten
     - `zram` (Temp. Blockdevice im RAM)
     - `dm_snaphot` (Overlay-Funktion für den Device-Mapper)
2. Eigenes Skript unter `/etc/initramfs-tools/scripts/init-bottom` anlegen
   - `custom-overlay.sh`
3. Neues initramfs bauen
   - `update-initramfs -u`

custom-overlay.sh (1/2)

```
#!/bin/sh
case $1 in
prereqs)
    echo "" #pre-requisites
    exit 0
    ;;
esac

. /scripts/functions
```

- Anfang bei allen Skripten ähnlich
- Bei Argument `prereqs` Vorbedingungen ausgeben
- Datei mit Hilfsfunktionen einlesen

Verfügbare Funktionen und Variablen

- Hilfsfunktionen
  - `log_success_msg`
  - `log_failure_msg`
  - `log_begin_msg`
  - `log_end_msg`
  - `panic`

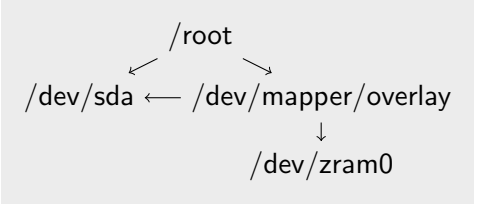
- Exportierte Variablen
  - `$ROOT` (z.B. `/dev/sda`)
  - `$rootmnt` (`/root`)
  - `$readonly` (`y` oder `n`)
  - ...

custom-overlay.sh (2/2)

```
if [ $readonly != "y" ]; then
    log_success "Overlay skipped"
    exit 0
fi

modprobe zram
echo 2G > /sys/block/zram0/disksize
umount $rootmnt
rootsize=$(blockdev --getsz $ROOT)
echo "0 $rootsize snapshot $ROOT /dev/zram0 N 16" | dmsetup create overlay
mount -o ro /dev/mapper/overlay $rootmnt
```

$readonly "y"
$rootmnt "/root"
$ROOT "/dev/sda"



Test des Overlays

```
mount | grep overlay
```

```
/dev/mapper/overlay on / type ext4 (rw,relatime)
```

```
cp /boot/initrd.img ./testfile && ll -h testfile
```

```
-rw-r--r-- 1 root root 66M Mar 10 09:18 testfile
```

```
dmsetup status overlay
```

```
0 52422656 snapshot 28032/4194304 0
```

```
zramctl
```

```
NAME ALGORITHM DISKSIZE DATA COMPR TOTAL STREAMS MOUNTPOINT
/dev/zram0 lzo-rle 2G 61.7M 50.5M 52M 2
```

Weitere Ressourcen

- Manual-Eintrag: `man initramfs-tools`
- Neueres System: dracut
  - Moderneres System zum Bauen des initramfs
  - Eigentliche Funktion bleibt aber gleich
  - Standard bei Fedora 12+, RHEL 6+, openSUSE 13.2+
- Für unzerstörbares Linux: overlayfs
  - Als Paket in den meisten Distributionen enthalten
  - Arbeitet auf Ebene des Dateisystems nicht des Blockdevices
  - Skript für initramfs schon enthalten

Zusammenfassung

- Linux-Boot-Prozess
  - Bootloader lädt Kernel und initramfs in den Speicher
  - Skripte im initramfs binden das Root-Device ein
  - Ermöglicht komplexe Setups ohne Unterstützung des Kernels
- Anpassung und Debugging
  - `break=x` zum Debugging
  - Eigene Skripte unter `/etc/initramfs-tools/scripts/`
- Fallbeispiel: Unzerstörbares Linux mit `dm-snapshot`
  - Einbinden der Kernelmodule
  - Skript in der *init-bottom*-Phase

Fragen? niklas.rother@geopp.de