Read-only rootfs

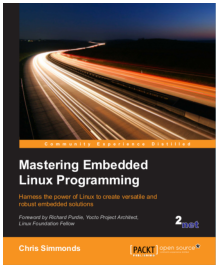
Theory and practice

Chris Simmonds

Embedded Linux Conference Europe 2016

License

About Chris Simmonds



- Consultant and trainer
- Author of *Mastering Embedded Linux Programming*
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at `http://2net.co.uk/`

`https://uk.linkedin.com/in/chrisdsimmonds/`

`https://google.com/+chrissimmonds`

Overview

- Why you need a read-only rootfs
- Where it all goes wrong
- Putting it right

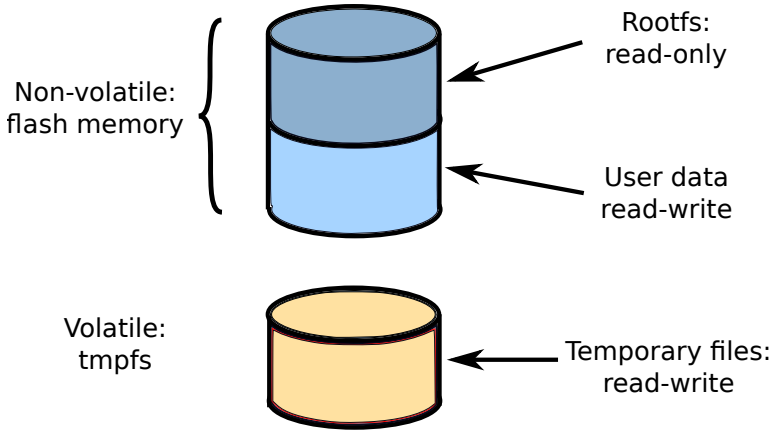
Why you need a read-only rootfs

- Reduce wear on flash memory
- Eliminate system file corruption
- Avoid accidents
- Enable rootfs image to be updated (manually or OTA)
- Make reset to factory defaults much easier

Where it all goes wrong

- You can’t just mount the rootfs ro
- Some files have to be created or updated at runtime
- Examples:
  - Passwords
  - Random seed
  - SSH keys
  - Network configuration
  - wpa_supplicant parameters

Categories of storage



Becoming stateless

- The state (stuff that changes) is either
  - **Non-volatile**: state that needs to be preserved
  - **Volatile**: state that is only needed for the session
- To create a **stateless** rootfs:
  - Move non-volatile state into an area of permanent storage reserved for that purpose
  - Move Volatile state to temporary storage (tmpfs RAM disk)

The ideal of a stateless rootfs

- No state stored in rootfs
- Each component can revert to sensible default configuration if there is no local configuration

Factory reset

- Just wipe the non-volatile state

System update

- Simply write a new rootfs image
- Two common mechanisms:
  - A/B rootfs partitions : A is live while B is being updated, then swap
  - Or, normal/recovery rootfs: boot into the recovery rootfs to update the normal rootfs
- This is the subject of several other presentations this week
  - One of which is mine: *Software Update for IoT: The Current State of Play, Wednesday 14:00*

Identifying state

- Which files are being written to?
- And by whom?
- Tools
  - diskstats
  - block_dump

diskstats

- `/proc/diskstats` contains information about disk reads and writes
- Useful stuff is field 1: reads completed and field 5: writes completed
  - Format is documented in `Documentation/iostats.txt`
  - `vmstat -d` prints the same information but better
- Example

```
# cat /proc/diskstats
 179       0 mmcblk0 2640 543 48969 207880 127 171 596 9990 0 26380 217830
 179       1 mmcblk0p1 99 0 1169 6260 0 0 0 0 0 1790 6260
 179       2 mmcblk0p2 2118 425 39212 160130 118 170 576 9860 0 23250 169950
 179       3 mmcblk0p3 97 0 2088 15510 0 0 0 0 0 1860 15510
 179       4 mmcblk0p4 2 0 10 150 0 0 0 0 0 150 150
 179       5 mmcblk0p5 271 118 5426 23540 9 1 20 130 0 2830 23670
```

Number of reads = 2640; Number of writes = 127

block_dump

- But, what is the cause of those writes? Which files?
- Turn on block system kernel logging using `block_dump`
  - See `Documentation/laptops/laptop-mode.txt` in kernel source
- Example (rootfs is on `/dev/vda`):

```
# echo 1 > /proc/sys/vm/block_dump
# echo hello > world.txt
# dmesg
sh(234): dirtied inode 701 (.ash_history) on vda
sh(234): dirtied inode 701 (.ash_history) on vda
sh(234): dirtied inode 701 (.ash_history) on vda
sh(234): dirtied inode 694 (world.txt) on vda
sh(234): dirtied inode 694 (world.txt) on vda
jbd2/vda-8(78): WRITE block 802 on vda (2 sectors)
jbd2/vda-8(78): WRITE block 804 on vda (2 sectors)
```

Logging block I/O from bootup

- Set `block_dump` in early boot script (may need to bump the size of the kernel log buffer to capture everything)
- Then filter out the junk:

```
# dmesg | grep dirtied | grep "on vda" | sort
chown(150): dirtied inode 548 (passwd) on vda
dd(298): dirtied inode 716 (random-seed) on vda
dd(298): dirtied inode 716 (random-seed) on vda
dropbearkey(325): dirtied inode 717 (dropbear_rsa_host_key) on vda
dropbearkey(325): dirtied inode 717 (dropbear_rsa_host_key) on vda
gzip(197): dirtied inode 714 (udev-cache.tar.gz) on vda
gzip(197): dirtied inode 714 (udev-cache.tar.gz) on vda
gzip(197): dirtied inode 714 (udev-cache.tar.gz) on vda
login(347): dirtied inode 543 (motd) on vda
[...]
```

Only the file name is displayed, not full path name, but it's enough

Common problems

- A lot of stuff happens on first boot
  - e.g. Dropbear writes SSH keys to `/etc/dropbear`, udev snapshot saved to `/etc/udev-cache.tar.gz`
  - Some of these things can be done at build time
  - Others need be changed to put stuff in a non-volatile state partition
- System config, including network parameters
- Saving random-seed
- Log files

Putting it right

- In reality, packages store state in various places
- Pragmatic solutions include:
  - Adding symlinks from rootfs to non volatile storage: e.g. /etc to /data/etc
  - Using unionfs or similar to overlay rootfs with non volatile storage

Putting it right: first pass

- Create a new partition for non-volatile state
  - For example `/data` (an idea borrowed from Android)
- Remount rootfs readonly
- Use tmpfs for volatile state (/run and /var/volatile)

`/etc/fstab`

```
/dev/root  /              auto       defaults,ro           1  1
/dev/vdb   /data          ext4       defaults              0  0
tmpfs      /run           tmpfs      mode=0755,nodev,nosuid,strictatime 0  0
tmpfs      /var/volatile  tmpfs      defaults              0  0
[...]
```

log files

- Many daemons write log files to `/var/log`
  - Including syslogd
- Usually, such log files are unnecessary for embedded
- Solution: make log files part of the volatile state by mounting a tmpfs on `/var/log`

Poky core-image-minimal does this already:

```
# ls -ld /var/log
lrwxrwxrwx    1 root     root            12 Oct  6 14:11 /var/log -> volatile/log
# mount
tmpfs on /var/volatile type tmpfs (rw,relatime)
```

random-seed

- Saving random-seed at shutdown is necessary when using `/dev/urandom`
- Solution: make it part of non-volatile state by symlinking `/var/lib/urandom` to `/data/var/lib/urandom`
- Or, modify the shell script that creates and restores random-seed

ssh keys

- The Dropbear ssh daemon stores keys in `/etc/dropbear` or `/var/lib/dropbear`
- Keys should be unique per device
- Solution: either move to non-volatile state by symlinking `/var/lib/dropbear` to `/data`
- Or, generate and pre load keys when device is provisioned

Doing less on first boot

- Build packages with sensible defaults so they work without config files
- Or, pre-load config files into the `/data` partition
- Generate per-device configuration at build-time rather than at first-boot
- Populate configuration files that are common for many packages (e.g. `passwd, group` with the composite requirements of all

What about Android/Brillo?

Android is a good example

- rootfs moved into `/system`, which is read only and stateless
- Non-volatile state in `/data`
- Has factory reset
- Has OTA update

What about Yocto Project?

Current versions of Yocto Project have a partial solution

- Add to your config:

```
IMAGE_FEATURES = "read-only-rootfs"
```

- Sets `ROOTFS_READ_ONLY=yes` in `/etc/default/rcS`
- Mounts rootfs ro and /var/lib as tmpfs
- But, nowhere to store non volatile state

Conclusion

- Read-only rootfs makes for better embedded systems
- Make rootfs stateless by moving state into non-volatile and volatile filesystems
  - diskstats and block_dump will help identify state

- Questions?

Slides on Slide Share
http://www.slideshare.net/chrissimmonds/readonly-rootfs-theory-and-practice
‘

Containers

- Containerised operating systems (e.g CoreOS) are a popular way to implement Internet/cloud services
- They have a similar problem:
  - Removing state from containers makes deployment easier
- Consequently, they are leading the way on statelessness
- Containers may be a useful way to deploy embedded
- swupd from Clear Linux
- resoin.io

Embedded != cloud service

- Use cases differ
- Cloud
- can assume high speed network to supply config (via LDAP, NFS, DHCP, etc)
- Emphasis on run-time integration of different app images into a base os
- Embedded
- Static app image (in a monolithic rootfs)
- Have to function stand-alone