What's missing in embedded build systems

Arnout Vandecappelle
Mind

https://mind.be
Who is Arnout

• Embedded software architect
• Focus on Linux OS integration
• Mind consultant since 2008
• Worked for 40+ customers in multimedia, security, home automation, satellite, telecom, chips, …
• Buildroot maintainer (team of 4)
Overview

• Build systems
• Field Updates
• Persistence / factory reset
• Manufacturing and provisioning
• Verified boot
Build systems

- **OpenEmbedded / Yocto**
- **Buildroot**

→ very configurable
→ have everything for an embedded system
Build systems that cover all bases are specialised

- **OpenWRT**
  - explicit choices: ubus, proc, …
  - very focused on routers
  - missing a lot, e.g. Rust
- **foundries.io** [Linux microplatform](https://foundries.io/)
  - based on yocto, OStree, writable /
  - tied very tightly to foundries.io cloud service
Traditional distros cover most bases

- Traditional distros take care of the desktop and server use cases
- Boot installer, update via package manager, everything writeable
- Not even ideal for desktop use case
  →https://0pointer.net/blog/fitting-everything-together.html
Field updates

- Every system needs field updates
- Almost always A/B update kernel+rootfs
- Tools exist:
  - swupdate
  - rauc
  - mender
  - ...
- Build systems provide these tools
Field updates - what’s missing in build systems

- Distinction between
  - manufacturing image
  - update image
- Default image includes partitioning for updates
- Integration of tool to produce update image
Field updates - what’s missing in bootloaders

• U-Boot doesn’t directly support A/B updates
  – Need to create script for it
  – Better if the logic is directly in the boot code itself
    (U-Boot env can be corrupted)
  – (Full) U-Boot and TF-A also need to be updateable!

• Better on UEFI
  – RAUC can directly set EFI bootorder and bootnext
  – systemd-boot can choose based on version number
Persistence

- A/B update → persistent data not in rootfs
- Additional partition/volume needed
- Must be mounted somewhere
- Must be populated
Persistent partition must be mounted

- **overlayfs on /:** simple, but dangerous:
  - makes / read-write → security impact
  - requires initramfs
- **direct mount on /var**
  - symlinks or bind mounts for /etc
- **writable / + readonly /usr**
  - requires initramfs
- **must be populated (not for overlay)**
  - systemd-tmpfiles
    - but doesn’t remove files
Persistence should be standardized

• Not completely solved → improvements
• All solutions are non-trivial
  – e.g. require initramfs
• Many integration considerations
• Ties in somewhat with update system
Factory reset

- Wipe persistent partition
- Re-format and populate it next boot → Same mechanism as first-time populating
- Recover from corrupted persistent partition → fsck
Manufacturing

• Build system assumes an image that is flashed e.g. to SD card
• This is not what is done in reality
  – Installer on SD card or USB key
  – Bootloader loaded over USB, tftp the rest
  – ubiformat NAND flash
  – Resize partitions to match actual eMMC size
    • cfr. Yocto’s wic format
• Sometimes integrated with HW test image
Provisioning

- Per-device data
- Some generated on first boot
  - ssh keys
  - machine ID
- Some come from “manufacturing database”
  - Serial number
  - MAC address
  - Password
  - Device certificates
  → These need to persist factory reset
  → Separate partition needed (again)
Next steps on manufacturing and provisioning

• Currently everyone reinvents the wheel
• Define common tooling for manufacturing
  – similar to update tools
  – some amount of configurability, but not too much
• Define common partition layout for provisioning
• Integrate in build systems
Verified boot

- It’s complicated :-)  
- Somewhat standardized up to kernel  
  - UEFI secure boot  
  - ARM Trusted Firmware (but SoC-specific provisioning)
Verified boot - rootfs

- No real standard / typical verification of rootfs
- dm-verity
  - Requires initramfs to mount it
  - Requires extra partition + support at build time
  - Doesn’t (easily) work on UBI (needs block device)
- Sometimes encryption used as poor verification
  - Still doesn’t work on UBI (dm-crypt needs block device)
  - fscrypt does work on writable FS (incl. UBIFS)
    but not usable for rootfs (unless it uses shared secret)
- Remote attestation usually more important
Next steps on verified boot

- Many things still need to be improved
- Define common tooling
  - produce signed images
  - changes to bootloader + kernel to maintain trust chain
  - this is a place to discuss improvements
- Integrate in build systems
  - including impact on partitioning
Conclusions

• Developers still have to reinvent the wheel and make ad hoc choices during integration
• Build systems should make those choices
  – perhaps offer a few alternatives
  – part of openembedded-core, not just some layer
• Also additional tooling needed upstream from build systems
CONTACT DETAILS

Mind (division of Essensium)
Arenberg Science Park
Gaston Geenslaan 9
3001 Leuven
Belgium

Arnout Vandecappelle
arnout@mind.be
@arnout on GitLab
https://mind.be