What small teams should know when building embedded Linux systems

Gregory Fong <gregory.fong@virgingalactic.com>
Sr. Software Engineer, Virgin Galactic
Embedded Linux Conference 2017

First, a question:

What are you **really** trying to build?

Where do you start?

- Is hardware defined yet? If so, is it good enough?
  - What functionality do you absolutely need?
  - What would be nice to have in the future?
- Are vendor reference boards available?
- What level of software support do you need?

Strong foundations

Vendor-provided SDK (and/or BSP)

- Whether based on Yocto, OpenWRT, Buildroot… it doesn't really matter
- Large amounts of commonly used software
- Usually missing pieces specifically for your niche
- Easy to add in functionality—look for guidelines!

Don’t ignore these


Support

Things to watch for

- Does the vendor provide thorough hardware and software release notes?
- Can you get a direct support channel, or will you have to go through a reseller?
- How long is software support typically provided for a chipset?

What if I have custom hardware?

Reference designs are your friends

Keep track of the differences, and note impact on project

- Use an issue tracker so things don’t get lost
- Estimate time to avoid surprises
- Scheduling reduces scope creep

Work with the visible derivations, note differences

- Similar board schematic?
- Same processor, memory, storage?
- Any common networking peripherals?
- Sensors kept, added, or removed?

Figure out what you'll need to update

- Figure out differences in pinmux between your board and reference
- Update device tree (most platforms)
- Add in drivers for added peripherals

Use a separate kernel git repo if you diverge significantly from reference

Finally, integrate your application

- Note compile- and run-time dependencies
- Appliance? Need to start stuff at boot, look how other services install themselves
- Deployment strategy—development vs production

Now that we have a the basic idea, some best practices…

Use version control!

~~Use version control!~~

Better:

Use version control like upstream

Why is upstreaming important? (aka how do I convince my boss?)

- Reduce maintenance costs over time
- Improved code quality
- Low-cost positive PR

Upstream.
You can’t afford not to.

Build system tips

- Use build system option for local mirrors
- Take advantage of shared caches to reduce build times, and share among developers
- Set up a Continuous Integration system (Jenkins, buildbot, etc.), deploy to TFTP server for network boot

Do code reviews.
Review **before** merge.

Summary

- Be involved in hardware design
- Use reference boards and vendor SDK
- Use version control
- Work with upstream as much as possible

That’s it!

Questions?