Linux: Reducing the cost of upstream development to encourage collaboration

Martin Peres

Intel Open Source Technology Center Finland

November 20, 2017
Introduction

Upstream issues

Forked kernels' issues

Pros of upstream development

Making testing cheaper

Summary
Introduction

Linux is everywhere

- Most of the servers/networking equipments;
- 80% of smartphones (Android) and 65% of tablets;
- Entertainment systems (at home, cars, planes, ...);
- Majority of IoT devices.

World domination?

- No, because all products use outdated kernels!
- Most products actually use forked kernels...
Is that a problem?

Yes, it lowers collaboration and leads to:

- Less features: All features do not get upstreamed/backported;
- Poorer Quality/Security: Less eyes per tree, fixes duplicated.
Summary

1. Introduction
2. Upstream issues
3. Forked kernels’ issues
4. Pros of upstream development
5. Making testing cheaper
Introduction

Upstream issues

Forked kernels’ issues

Pros of upstream development

Making testing cheaper

Why upstream is no good for vendors?

Upstream from a vendor’s perspective

Objectives of making a product

Get it as good as possible, and as quickly as possible

Challenges with upstream

- Linux development not product-oriented:
  - Releases not in sync with products;
  - Conflicting objectives: upstream wants generic solutions
- Code sharing between drivers mandated: AMD’s DAL/DC;
- Stable user ABIs, no user-visible regressions;
⇒ Increased dev. cost and Time-To-Market (TTM)

Forked kernel?

- Full control over the code;
- None of the above challenges!
Summary

1 Introduction
2 Upstream issues
3 Forked kernels’ issues
4 Pros of upstream development
5 Making testing cheaper
Issues with forked kernel

What should be done when the next product comes?
- Re-use the previous product’s kernel? ⇒ technical debt;
- Rebase changes: can amount to a full re-implementation.

Challenges with forked kernels
- You don’t get to shape the future of Linux:
  - Out-of-tree code is not supported;
  - Risk that internal changes break your features and userspace;
- Maintenance?
  - Automotive products need 10+ years of maintenance;
  - Linux Long-Term Support (LTS) maintained for 2 years;
  - LTS releases only get fixes, no new features;
  - Rebasing generates no revenue.
Summary

1. Introduction
2. Upstream issues
3. Forked kernels’ issues
4. Pros of upstream development
5. Making testing cheaper
Pros of upstream development

Nice features of upstream development
- Non-regression of the user ABI makes updates easy;
- Never need to rebase: Others improve Linux and your code;

Problem: Testing isn’t free!
- Unless constantly tested, a feature gets accidentally broken;
- Without continuous testing, updating isn’t free!
Summary

1. Introduction
2. Upstream issues
3. Forked kernels’ issues
4. Pros of upstream development
5. Making testing cheaper
How to make testing cheaper?

Reducing manual testing to 0

- Pre-merge testing is the best way to prevent regressions;
- Linux accepts about 8 changes per hour, in average;
- ⇒ all testing needs to be automated!

Problems with automated testing

- The full product needs to be tested;
- Requires system-level testing;
- ⇒ Need for better HW-assisted test suites!
How to make testing cheaper?

Example of full product testing: Project trebble

- Android 8 de-couples the UI from the vendor-provided system;
- The vendor interface is fully unit tested;
- ⇒ could be used for continuous integration!

What can we do on our side?

- Lead by example: provide regression free graphics!
How to provide regression-free graphics?

Many dependencies

- Improve the coverage of Open Source test suites to test:
  - all graphic-related features of the kernel;
  - all drivers.
- Validation HW:
  - Chamelium everywhere for testing DP/HDMI/VGA and sound
- CI platform:
  - running the relevant test suites on all drivers;
  - decentralized so as everyone can add platforms;
  - developed and maintained by everyone;
  - Controller instance hosted on fd.o?