The Embedded Linux Problem

Mark.gross@intel.com
Android-Linux kernel Architect

February 2013

outline

- Little about me
- Intro
- History
- Environment
- Key questions
- Techniques
- Moving modules out of tree
- Summary / Questions

intel

About me

I’ve been doing Android since 2008 and Linux kernel since 2001

What I do at Intel these days:

- Help to solve the embedded problem.
- Help to solve the scaling problem
- Help get ahead of anticipated transitions
- Help with SW on new architectures and devices
- Do code reviews
- Teach and help others
- Help with unplanned surprises

INTRO

What's the embedded problem?

Def: When the time for hardening a device-specific-Linux-kernel is longer than the time for new kernel releases, the effort needed to move to new kernels becomes prohibitive.

- Upstream development becomes impractical.
- Inertia to stay on older kernel versions becomes high.

intel

What makes hardening take so long?

- New hardware
- New drivers
- New use cases
- Everything builds from bottom up.
  - HW, FW, kernel, HAL, middleware, OS-stack
  - Testing depends on the full stack to be mature enough that the tests can be run.
  - Bugs are found late.

intel

Why bother at all with new kernel versions?

Customers expect “modern” kernels for new design wins.

- Def: modern kernel is a LTS version <18 months old when high-volume manufacturing starts. (Today its 3.4; next Q3 it may be 3.9)

Driver and integration teams can't handle supporting multiple kernel trees or branches concurrently.

- When forced to move for one product, then everything moves unless it’s in “maintenance mode”.

intel

History

Kernel versions:

Baseline	Targets	Comments
2.6.32	MRST	Forked off moblin kernel (éclair + froyo)
2.6.35	MRST, MFLD	Forked off Meego kernel (froyo, GB)
3.0	MFLD, CTP, soc1	ICS + JB
3.4	MFLD, CTP, soc1, BYT, TBD	JB + K

intel

Enviornment

How device hardening happens

- Done on a single branch supporting multiple products and SOCs
- Developer changes are merged in a CI manner
  - Testing is done per change and on daily and weekly builds.
  - Experiences about 50-100 changes a week to the kernel
- Goes on for 1 to 2 years per kernel version
  - Experiences significant feature creep
  - Has 30 to 80 non-upstream drivers
    - About 50% are 3rd party “reference” drivers hacked to work
    - Has some changes made to core and arch code

intel

Management don’t care about new kernels.

Projects for new devices are planned assuming a one-off kernel version.

Projects have assumptions that SW effort ends at RTM. i.e. months after kickoff.

- the amount of effort for new SW validation and hardening goes to zero.

Moving to new kernel versions is only done under pressure.

intel

Honey badgers don’t care about new kernel versions

- Driver and product integration teams don't care about new kernel versions.
- They care about the recovery time from any regressions introduced by new kernel migration.
  - Seen as >2 months
- They delay new kernel migration as much as possible.
- They are not capable of doing concurrent development on multiple kernel versions.
  - They are pushed by management to focus on bugs in the current kernel only.

intel

What about upstream goodness?

Quality side-effects of pushing drivers upstream-first are not seen as compelling enough justification for driver or integration teams over existing TTM pressures.

Up streaming drivers is seen as a way to reduce TTM and TCO for the next kernel version used.

- But isn't seen to add value to current deliverables.
- Isn't given budget to driver teams.

intel

Moving to a new kernel takes too long

It takes 2-3 months to do a first pass at a new kernel by kernel developers (not involving the driver teams).

- This is enough time for 1200 changes to happen to the old version.
- Demonstrating that new kernel version has all driver bug fixes included is impossible.

The auditing problem for new kernel versions is a BIG deal.

Problem exacerbated by push outs to planned switch over date.

intel

Problems with new kernel migration

- 1000s of patches to old baseline + back ports intertwined into a single threaded development branch that is impossible to rebase
- There is are patches changing both new driver code and existing upstream code within a single patch.
- Driver teams don't care about new kernels.
- After moving to a new kernel version, it takes 2 more months to partially harden after first order porting is completed.
  - Just for alpha/beta quality
  - Effort involves all the developers in the organization.

intel

What are the problems with kernel migration?

Single threaded integration branch with > 3K changes

- Most times we end up forklifting drivers manually.

Bring up of SW Stack on new kernels is done in isolation by Kernel FT while all the other teams continue adding stuff to the old kernel.

FT's keep adding 100+ changes a week the old tree causing test blockers.

We can't say if we have all the bugs fixed in the old kernel accounted for in the new one through static analysis.

To put it another way.

Our single treaded trees are not manageable.

Our HW can't boot on upstream kernels enough to enable upstream contributions.

Moving to new kernel versions == retesting 1000's of BZ's

Most of the workforce is focused on product integration on existing kernel and doesn't care about newer ones.

intel

So where does this leave us?

- Customer expectation is for once-a-year "hardened" migration to modern LTS version with rolled out in Q4 each year.
- Difficulties working with multiple kernel versions concurrently.
- Upstream-first methodology seen by management as not realistic for products needing hardening.
- Ability to upstream drivers we want to upstream is limited.

intel

How Linux is developed:

- Decentralized development done on many tracking branches
  - Devs rebase to master prior to sending a pull request.
- Hierarchical integration:
  - Linus has trusted maintainers do most of the merging and he pulls from them.
- Time between RC releases is 1 week or more.
- Volunteer-based testing of master branch
- Hardening gets done on linux-stable

intel

Key Questions

Are we looking at the problem wrong?

- How can we use kernel versions more effectively WRT new features?
- How can we isolate multiple concurrent kernel version development from the driver and integration developers who can't handle it?
  - How can we emulate tracking branches within a single-branch integration life cycle?
- How can we isolate changes to the core kernel form new kernel additions?
- How can we isolate massive backport changes?
- How can we reduce the time it takes to move to a new kernel version and harden it.

intel

Techniques

So what to do?

- Have an upstream prioritization.
- Disallow changes touching both upstream and non-upstream code.
- Use merge commits for all back ports from “back port” tracking branch
- Only kernel team should be allowed to make changes to non-driver upstream code
  - Upstream code changes done as merge commits from core-kernel tracking branch
- Make the changes to header files painful.
- Demand that user mode code always builds independently of kernel source tree.
- Isolate non-upstream code from kernel tree in different git projects as much as possible.

Prioritize upstream goals

Upstream is seen as a way to reduce TTM on new kernel migration.

Want upstream to:

- boot target to working ramdisk console
- Have working P, C and suspend to RAM states
- Have working persistent storage (eMMC)
- Have work working USB gadget (android adb)
  - Done already by community.

intel

Changes that touch upstream and non-upstream code are evil.

- Changes that touch upstream code cause
  - Merge conflict risks for linux-stable updates
  - New kernel conflicts doing rebase –onto
  - Not cool unless you upstream it
- Not as easy to prevent as hoped
  - We added a checkpatch function to reject all such changes, but ran into issues with the feature teams touching upstream header files.

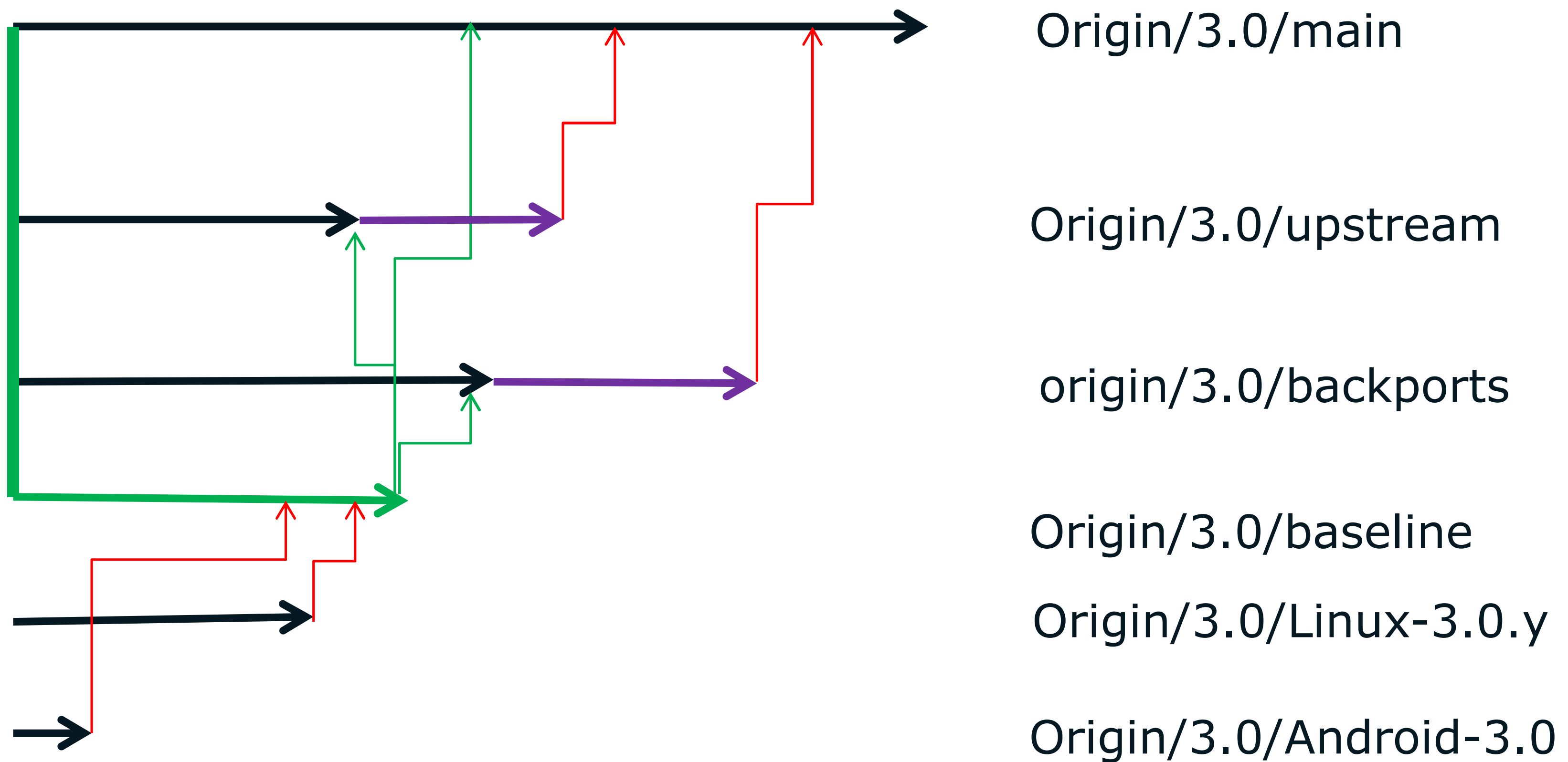
intel

Use merge commits for back ports and non-driver upstream file changes

- Put all back ports on a branch and merge into old kernel.
- On new kernel migration simply drop back port branches from the rebase.
- Core kernel changes are easily reviewed for up streaming.
- Core kernel changes are simpler to rebase.

intel

branch family for main integration branch



Make it hard to change upstream code!

- Integration and feature teams have a tendency to hack wherever they feel it's needed.
- Touching upstream code needs to be made expensive and require extra process, handling and documentation.
- Changes to upstream code that don't get pushed upstream are a significant risk to kernel migration.

intel

Make all header changes expensive!

- Changes to interfaces need to be given extra care and documentation.
- Incompatibilities between kernel versions tend to happen primarily with *.h changes.
  - They can break compiling of OS stack.
  - They can break runtime execution with new kernel versions.
- These incompatibilities are test-blockers for new kernel versions.
- Push some of the costs associated with header changes to the honey badgers making the changes.
  - Document ABI changes.
  - Document interface changes.

intel

Demand user mode builds without kernel sources

- Tight coupling between user mode and kernel is evil (and sometimes illegal)
- Such coupling results in test-blocking build issues when dropping a new kernel version into the OS stack build tree.
  - Delays development for new kernel versions significantly.
- Forces honeybadgers to not be so lazy and to define proper interface specifications
- Removes temptation to do bad practices

intel

Moving drivers out of tree

Isolate non-upstream drivers into separate git projects

- If the drivers were each in their own out-of-tree git project, a new kernel-tracking branch could be maintained by the kernel team for each project.
  - The auditing problem is solved for each driver because by definition you always have the latest version of the driver when you rebase that tracking branch to the tip of the driver for the old kernel.
  - The problem of having driver teams incapable of working on multiple kernel versions is solved as the kernel team will be maintaining the tracking branches that go on top of drivers to make them work with new kernels.

intel

Separating git projects == tracking branches.

- With the non-upstream drivers each kept in their own out-of-kernel git, we allow them to do single-branch development.
- We enable the kernel developers to keep tracking branches per external driver git. This enables upstream kernel development and testing.
- We solve the auditing problem.
- We reduce the TTM for new kernel-enabling of a device without overburdening the driver and integration teams with concurrent development on multiple kernel versions.

intel

Summary

Upstream-first does not apply to device hardening

Prioritize your upstream efforts based on TTM for next migration

Disallow changes that touch both upstream and non-upstream code.

Use merge commits for all back ports

- Only let the kernel team touch this branch.

Use merge commits from tracking branch for changes to upstream code.

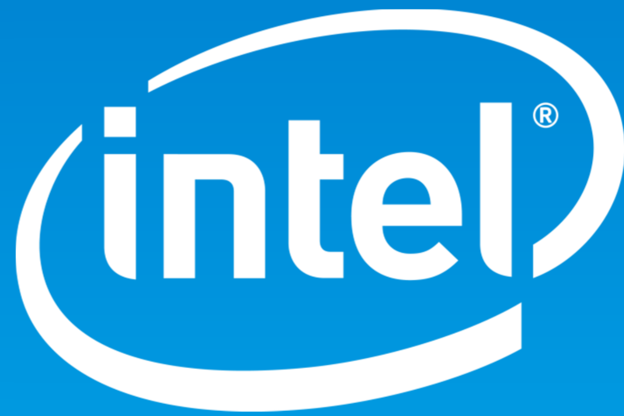
- Only let the kernel team touch this branch.

Demand user mode code to always build independently of kernel source tree.

Make changes to ANY header file in the kernel painful.

Isolate non-upstream drivers from kernel tree in different git projects as much as possible.

Questions?


intel