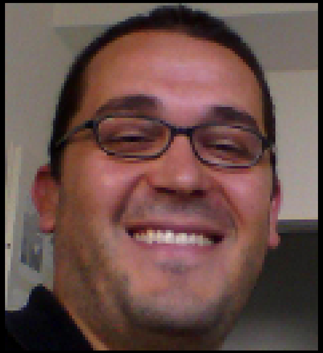
The maintainer's POV

Borislav Petkov

AMD
together we advance_

Disclaimer

- IMNSVHO: all opinions are mine and not of my employer’s
- Regular LKML readers would find most of this very familiar



The development process: a rickety fence

Stuff flying over the fence

- On one side
  - An ever-growing number of code submitters: most are employed so business interest to get code upstream
  - Users and testers reporting back: never enough
  - Occasional experimenters
  - Random bored yahoos who are looking for someone to play with

Code submitters today

- Flood maintainers mboxes
- Hardly, if ever, help out with review, but “Ping! When will you look at my patches?”
- “What is taking you so long – it is a simple patch?!”
- “I just flooded you with my patchset two days ago. Lemme send it again in case you’ve forgotten.”
- “I should build the patch? Nah, too busy besides it is obvious.”
- Design? Meh, throw my “code” at the maintainer - she/he will think for me and tell me what to do
- If it gets applied: most drop off the code and “disappear.” Maintainers get to mop up and forward-port it indefinitely

The “my code must be upstream now!” chimera

- There’s this senseless rush to get stuff upstream which makes people do silly things
  - Send series untested
  - Review feedback gets ignored
  - Git send-email should have a timeout…
  - Cause more frustration than necessary
- No, you don’t want to rush your stuff upstream
- Srsly, you don’t
- If your manager rushes you, he probably has no clue how the process works
- What do you think happens when your code gets upstream?
- Do you think people and distros will jump on it immediately?
- Or can it wait simply for the next kernel which is right around the corner and you can save yourself all that malarkey

Linux kernel maintainers today

- On the other side of the fence: the maintainers
- They still cannot scale…
- Workload grows and grows
- Most active subsystems do maintainer “groups” but scaling still hard
  - Bug fixes
  - Code integration
  - New features review
  - Oh, there’s this day job thing too…
  - The new fun: embargoed hw issues – researchers love speculating CPUs...

The code submitters and maintainers “symbiosis” today

- There’s this fence
- stuff is flying over it constantly and bidirectionally
- maintainers either catch it in-flight or
- scrape it off if they were too late
- And it never ends and never stops…

together we advance_

Add hw vendors to the symbiosis

- It’s only software, right?
- Hardware is cast in stone, mostly
- Sure, we can build you a house of cards
- What happens if we move stuff on the ground floor?
- Or if we need to remodel the house because there’s others in the neighborhood

The unfortunate symbiosis

- The current situation is not sustainable
- Huge backlog of patchsets to review
- Huge, ever-growing TODO lists full of items which seldom get addressed – more likely become obsolete
- “expire” when the hw expires/becomes obsolete.
- Barely time for refactoring and cleanups – a life-saving task nowadays
- Maintainers are not born grumpy – they become grumpy

The Fix: make you all reviewers and maintainers

What do I mean by that

- If code submitters try walking in maintainers’ shoes, the whole process would be a lot smoother
- Perhaps become maintainers themselves
- Not even absolutely necessary – enough if code submitters try to share some of the maintainers' work
- Might help them understand why maintainers almost always say “no”
- And why they’re so grumpy most of the time

The development process: what it should be

- The long-term prosperity of the kernel is the main goal
- Source code maintainability
- Clean design capable of accommodating functional extensions: no bolting of stuff hastily because manager or code drop deadline says so
- Refactoring/redesign: hell yeah! It is important. Especially when sending new enablement patchset
- Apply sane thinking everywhere
- Think about the big picture
- And all – contributors and maintainers alike - should pursue the same goals

How can you make the maintainers’ life easier

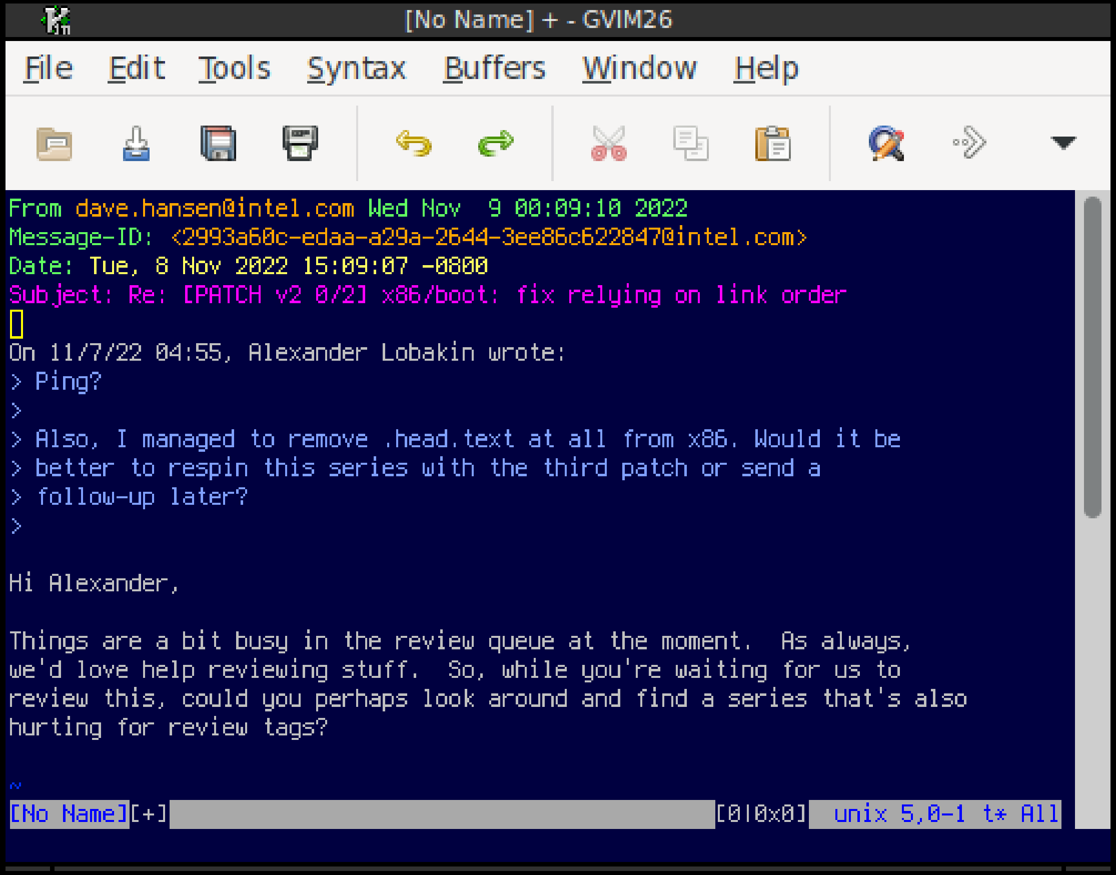
- So you’ve sent your patches, the maintainers are busy and you’re wondering what to do?
  - While waiting for review => review. Srsly, I mean it
  - Test linux-next
  - Look at a bug report or three
  - Show compassion

Review

- Code review is, without a doubt, the most unthankful and, at the same time, the most important work to do on the kernel
- Getting good review is priceless and should be appreciated
- Find an interesting patchset on LKML (and there’s no shortage of patchsets there) and **BECOME A MAINTAINER**
- Put yourself in a maintainer’s shoes
  - Does this code make sense?
  - If this were your kernel, would you accept it?
  - Would you support it for gazillion years?
  - ...

Review

Dave Hansen put it eloquently:



```
From dave.hansen@intel.com Wed Nov  9 00:09:10 2022
Message-ID: <2993a60c-edaa-a29a-2644-3ee86c622847@intel.com>
Date: Tue, 8 Nov 2022 15:09:07 -0800
Subject: Re: [PATCH v2 0/2] x86/boot: fix relying on link order

On 11/7/22 04:55, Alexander Lobakin wrote:
> Ping?
>
> Also, I managed to remove .head.text at all from x86. Would it be
> better to respin this series with the third patch or send a
> follow-up later?
>

Hi Alexander,

Things are a bit busy in the review queue at the moment.  As always,
we'd love help reviewing stuff.  So, while you're waiting for us to
review this, could you perhaps look around and find a series that's also
hurting for review tags?

~
[No Name][+]                                [0|0x0]  unix 5,0-1 t* All
```

AMD
together we advance_

The stickler for details

- Sometimes maintainers are complaining about every minor thing, nitpicking
- It might sound too picky to the unenlightened, perhaps too much
- But there's a deep and sensible reason for it all
- It might make more sense once I explain the more important things maintainers complain about
- Keep listening...

Commit title

- Prefix: subsys/component: Concise commit title summarizing the patch in imperative tone
- git log <filename touched in the patch> gives good examples, most of the time
- Q: Why is it important? A: The uniformity argument
- Helps considerably when dealing with a **lot** of commits and doing patch tetris
- The longer a maintainer deals with style, the less time for actual review

The commit message

- Should be written by humans for humans
- Definitely not write-only
- Write it as clear and as understandable as possible
- You don’t have to be a native speaker of English - sometimes even they do a lousy job
- Clarity is the main goal here
- The maintainers will aim at fixing up style issues
- Nevertheless, please still do try genuinely

The commit message

- The most common observation: too laconic commit messages
- Completely leaving out the context preparation
- Must always keep git archeology in mind – leave enough breadcrumbs
- Why most maintainers frown upon whitespace and checkpatch whole-file cleanups
- The other fun one: a non-native speaker using way too complex sentences because <insert reason here>, and failing => KISS!

The commit message: contents

- Try to explain the issue as if you’re writing a note to your future self, who’s going to read it months, years from now. Are you explaining it in enough detail so that your future self will understand it again?
- Explain the “why” of a patch and not the “what”
- Explain the non-obvious aspects of the change
- Another fun one: “We” in commit messages. “We” who? Use imperative and impartial formulations for maximal clarity
- People and your future self included will be thankful for the effort

together we advance_

The commit message: structure

- Prepare the context briefly: “There’s this fancy kernel contraption which does A and B…”
- Define abbreviations and concepts before using them – not everyone is in your head :-)
- Explain the problem at hand
- Explain why it happens: it is important for your future self to know why that is. Might help with the analysis of some related, future bug
- Fix the issue by doing X
- (Potentially do Y, while at it) – related, minor things which don’t warrant a separate patch
- If in doubt: Documentation/process/submitting-patches.rst – see? :-)

Tags

- Fixes: <sha1 abbrev> (“<commit title>”)
  - Patches fixing an issue, must have in for downstream propagation
  - Automating tracking and selection of those
- SOB chain: traceable path of a patch's origin to its “final resting place”: a lot of submitters don't really know that
- Link: a pointer to the mail thread containing the “meat” of the discussion
  - A lot easier nowadays with lore.kernel.org
  - Link: https://lore.kernel.org/r/<Message-ID>

Official vendor documentation

- Abysmal examples
- That doesn’t mean we will perpetuate it in the kernel – kernel can and does better most of the time
- Try to use common sense, try to distill the issue to the important points
- Drop the marketing speak people do pick up at the company coolaid corner
- HW vendors: oh, c’mon, it’s just software. Yeah, right.

Merge window

- The regular, recurring, two week insanity exercise for maintainers
- Write pull requests
- Check for merge conflicts, test merge resolutions, handle urgent bug fixes
- Track what goes where
- Other unexpected patch tetris
- Now why would you send your new patchset then?
- You know it won't be applied then and it won't be reviewed either
- So why not simply hack on something else interesting or review some outstanding pile?
- Even staring out of the window is a better activity than sending new features then
- A win-win situation

Testing

- Testing is overrated, right?
- Every patch in your submission needs to build and boot correctly → bisectability
- Make sure you test your submission
- I really mean it
- Rushing something untested will not get it applied => it'll make everything worse
- Oh, and one more thing: make sure you test
- If free cycles, do test linux-next, tip tree, etc. and report issues → priceless!

One more thing: make sure you test!

- The other most important thing to do besides code review
- If free cycles, do test linux-next, tip tree, etc. and report issues → priceless!

A maintainer is never satisfied

- People are asking why we care so much about code design and give all that review feedback
- And why maintainers are so damn picky about every little thing
- It is good enough this way, why don't you simply apply it?
- The thing about uniformity, design patterns and using well-known idioms -> easier review
- Did I say how beneficial it is for everyone involved to read Documentation/process/submitting-patches.rst from time to time? :-)
  - especially new submitters
  - old and rusty maintainers too :-P

Comments

- Why enforce exact comments style? The uniformity thing
- Do not comment the obvious: you're wasting a perfectly good line → distraction
- Rather, comment the important, non-trivial place in the code → breadcrumbs
- Add kernel-doc to functions: it helps to keep the function's implementation from getting “misused” and it changing; and ofc explains what it does
- Proper function and variable naming can make a comment superfluous
- document locking requirements

Spelling

- Spellcheck: if you can’t spellcheck your changes, why bother at all?
- Srsly, we should not be even talking about this – it is integrated in the relevant editors, tools, etc. Hell, even AI can spellcheck for ya

Further reading

- Documentation/process/ and submitting-patches.rst especially
- Documentation/process/maintainer-tip.rst, specific for the tip tree
- And as always, apply common sense for clarity and keep it simple

Summary

I’d let Greg do that for me:

Seriously. It's easier for the maintainer to not accept your code at all. To accept it, it takes time to review it, apply it, send it on up the development chain, handle any problems that might happen with the patch, accept responsibility for the patch, possibly fix any problems that happen later on when you disappear, and maintain it for the next 20 years.

That's a lot of work that you are asking someone else to do on your behalf…

So your goal is, when sending a patch, to give me no excuse to not accept it. To make it such that if I ignore it, or reject it, I am the one that is the problem here, not you.

Greg Kroah-Hartman

Thanks!

Questions?

together we advance_

Copyright and disclaimer

AMD
together we advance_