Contribution ID: **235** Type: **not specified**

Adding Testable Code Specifications in the Linux Kernel

Thursday 11 December 2025 17:00 (45 minutes)

In the Linux kernel, design intent is an emergent property. There are a number of well understood reasons for this, most notably the evolving needs of user space and the distributed nature of its development. It is also reasonable to suggest that, although remarkably complex, kernel design intent can be framed in rather simple high level terms: to behave as an arbitration layer between hardware and application software. Therefore, in practice the Linux kernel is said to "work properly"when user space behaves as expected in specified (e.g. POSIX) and unspecified (e.g. scheduler behavior) ways.

Although a reasonable explanation for the path taken from humble beginnings to where we find ourselves today, the Linux kernel development process is straining under the weight of its success. The community has responded with tools like syzkaller, kselftest, KUnit, and vendor-specific test suites to help the kernel stay aligned with expectations. In addition, there is a robust documentation set and a vast archive of community knowledge documenting the design and use of the Linux kernel. Despite all of this, a persistent gap separates detailed design intent from all forms of testing. This matters because tests risk misalignment without explicit developer signoff.

Perhaps in very narrow situations one might find evidence of a maintainer affirming that a test accurately reflects the details of their intended design. But on a widespread basis, there is no evidence that the intent of the designer can be unambiguously traced to any form of repeatable testing. Even among kernel maintainers this is the case, with PREEMPT_RT developers famously needing to reverse engineer kernel scheduling behavior to determine its fairness and latency properties. In effect, all kernel testing is largely based on a system of piecemeal reverse engineering and educated guesses. This does not discount the significant value of community driven testing, but it is important to acknowledge that no repository housing test code can lay claim to the usage of a metaphorical (or literal) Test-Reviewed-by-Maintainer signoff tag.

Setting aside, for a moment, the sheer size of the Linux kernel, we can reason about a solution to this problem and then work our way towards a sensible approach. Design expression is found at many levels, some of which become impractical to reason about at the kernel source code level because they rely on complex assumptions of use. Therefore we restrict ourselves to the lowest reasonable level of design intent, described in RTCA DO-178C as, "software requirements from which source code can be directly implemented without further information". Best practices for developing these requirements, which we shall call "testable expectations", are out of scope for this presentation. Suffice to say, authorship of testable expectations is a specialized skill which forces one to reason very deeply about the intent of code.

With testable expectations in hand, we have the level of clarity required to apply a virtuous cycle to Linux kernel code that affirmatively ties code to design intent in a traceable way. With maintainer agreement (a "Signed-off-by" as it were), a test case can be derived from the testable expectation. When the test case passes, coverage tools like llvm-cov and gcov can be brought to bear to ensure that relevant code is not overlooked. With expectation, test, and coverage in agreement, we can affirm that code accurately reflects design intent, achieving the purpose of software verification.

We end with addressing the "elephant in the room", the sheer size of the Linux Kernel and potential for added burden to the maintainer community. As any maintainer knows, bug triage is a huge part of the job, and while bugs will never go away, this virtuous cycle will shift the burden away from verification ("is the code doing the thing right") to validation ("is the code doing the right thing"). Both verification and validation are necessary parts of a maintainer's job, but this approach promises to alleviate a significant portion of the

verification aspect. Thus the “juice is worth the squeeze”, but significant skepticism is warranted and input from the kernel developer community is needed to help refine the approach.

This talk will cover the following topics:
1) the current limitations of Linux Kernel low level design guidelines (as in Documentation/doc-guide/kernel-doc.rst)

2) Detailed examples of virtuous cycle behavior and progress to date in filling the gap from point 1) (a proposal is already being discussed in [1])

[1] https://lore.kernel.org/all/20250910170000.6475-1-gpaoloni@redhat.com/

Primary authors: WOLBER, Chuck; PAOLONI, Gabriele (Red Hat)

Co-author: STEWART, Kate (Linux Foundation)

Presenters: WOLBER, Chuck; PAOLONI, Gabriele (Red Hat); STEWART, Kate (Linux Foundation)

Session Classification: LPC Refereed Track

Track Classification: LPC Refereed Track