Contribution ID: **249** Type: **not specified**

KUnit Testing Insufficiencies

Friday 12 December 2025 12:00 (25 minutes)

High-integrity applications require rigorous testing to ensure both reliability and compliance with industry standards. Current testing frameworks for the Linux kernel, such as KUnit, face challenges in scalability and integration, particularly in environments with strict certification requirements.

KUnit tests, which are currently the most widely accepted and used solution for testing Linux kernel source code, have a number of drawbacks in the way they are built and executed. The KUnit framework lacks feature parity with other modern unit test frameworks, making it difficult to achieve comprehensive and robust low-level test coverage. Furthermore, the way KUnit tests are built and executed creates a lack of scalability, which is necessary to create and maintain the many thousands of tests that will be required to verify functionality in a robust, complete, and automatable way.

KUnit tests are integrated into the Linux binary. This requires building the kernel and running it to execute the tests. Additionally, the high volume of tests needed for adequate coverage would increase the size of the kernel beyond what is manageable. This makes it necessary to divide the tests so that multiple kernels with different sets of tests are built. This, in turn, necessitates additional analysis to prove that the changes made in each of these individual kernel builds do not negatively impact the fidelity of the tests for the targeted features. Considering the lengthy build and execution times, along with the need to build and analyze multiple kernels, it is evident that scaling up to the creation and maintenance of thousands of tests poses significant challenges.

In addition to the scalability issues, KUnit lacks essential features needed for testing highly reliable systems. It does not provide a consistent, maintainable, and automatable way to isolate small sections of code. This is crucial for low-level testing and coverage. For example, consider a function A that has dependencies on three other functions, B, C, and D. When testing function A, the results of functions B, C, and D may influence the execution path in function A. However, it is not desirable to actually test the implementation of functions B, C, and D. In most common unit test environments, it is straightforward to create either a fake or mock implementation of those functions in the test file, or to link to an object file that contains a fake or mock of those functions. In KUnit, achieving "mocking"(or at least a similar effect) requires creating a patch that modifies the kernel. The simplest approach is to use conditional compilation macros that are controlled through kernel configuration to generate mock versions of the tested function's dependencies. Every time a mock or fake is used, it must be manually created and maintained through patches. When this effort is multiplied by thousands of tests, the maintenance burden becomes evident.

Topics for discussion:

- Addition of unit test capabilities (extending KUnit or otherwise) that allow the compilation and testing of small, isolated sections of the Linux kernel source out of tree.
- Ability to test Linux kernel source in tests that run in user space (requires the aforementioned compilation out of tree with external tools).
- Integration of mocking of test dependencies (preferably automatically).

Primary author: WHITEHEAD, Matthew (The Boeing Company)

Presenter: WHITEHEAD, Matthew (The Boeing Company)

Session Classification: Safe Systems with Linux MC

Track Classification: Safe Systems with Linux MC