KUnit for Userspace

David Gow <davidgow@google.com>

What Is KUnit?

What Is KUnit?

- A Unit Testing framework for the Linux Kernel.
- Upstream since 5.5
- Tests are written in C, run in kernel mode, and can call arbitrary kernel functions.
- Tools to run these tests, and parse the results:
  - ./tools/testing/kunit/kunit.py run
  - Uses User-Mode Linux by default, or QEMU for other architectures.
  - ./tools/testing/kunit/kunit.py run --arch x86_64

Recent and Advanced Features

- Test-managed devices: Create a new struct device / struct device_driver managed by KUnit
  - Devices sit on a new kunit_bus
  - Automatically cleaned up on test exit
- (Re-)run built-in tests after boot from debugfs
- Memory context support
  - kunit_vm_mmap
- A bunch of arch and documentation fixes:
  - New filename guidelines
  - Rust testing documentation and 32-bit UML support
- For the full list of changes, version by version, see https://kunit.dev/release_notes.html

Why userspace?

Why?

- Testing in kernel-space is annoying.
    - (Even with nice tools.)
    - Userspace code is easier to run, debug, and reason about.
- Building a whole kernel just to test one function is overkill.
    - Slow build and boot process.
    - Other parts of the kernel can trigger errors: something self-contained is nice.
- Userspace code is easier to share
    - If we can reproduce a bug in userspace, it's easier to share a minimal case with non-kernel developers.
- We have userspace code in the kernel tree we may want to test
    - tools/ directory
    - Build tools, user-facing code, etc.

Library code

- Data structures and algorithms
- Helper functions
- Parsers
- Anything 'self-contained' or 'pure'
- Code with explicit abstractions

For example:

- Rosebush:
    - https://lore.kernel.org/all/20240625211803.2750563-5-willy@infradead.org/
- Core VMA manipulation functions:
    - https://lore.kernel.org/lkml/cover.1722251717.git.lorenzo.stoakes@oracle.com/
    - Make them buildable outside the kernel.

Code shared between kernel and elsewhere

- There exists code, e.g. compression libraries, which are used both in userspace and in the kernel
- Having one test framework which works for both could be nice.

Tools which live in the kernel tree

- These aren't kernel code at all, but live in the kernel tree.
  - Kernel internal APIs not available.
- Want to be relatively self-contained (minimal external dependencies), and consistent with kernel code.

For example

- Perf
  - Has its own, vaguely KUnit-like unit testing framework
  - https://elixir.bootlin.com/linux/v6.10.1/source/tools/perf/tests/tests.h

Tests which need to be shared with non-kernel folks

- Typically tests of compiler-level features
  - Useful to have these easily reproducible to send to compiler bugtrackers.

For example

- The 'stackinit' KUnit test:
  - https://lore.kernel.org/all/20220224055145.1853657-1-keescook@chromium.org/
  - Originally proposed with a standalone version.

What about…?

Why not… just use a kselftest?

- kselftest tests run from userspace, so they should be a good fit.

But:

- kselftest misses some of the unit-test specific tooling
  - Structured test functions with executor.
  - Resource management.
  - Parameterised tests
- kselftest is really aimed at testing the running kernel — this is aimed at testing code before the kernel is built
  - Not useful for debugging the currently running kernel.
  - Still aiming for self-contained unit tests, not integration-level testing.
- Makes even less sense for tools like perf

Why not… just use an existing C/C++ unit test framework?

- There are several C Unit Test frameworks out there for userspace code, why another one?

But:

- Can't re-use tests in user and kernel mode
  - Useful for compiler and library tests.
  - (Not for tooling tests, like perf)
- Kernel developers may be more familiar with KUnit
  - (But non-kernel developers may be
- Can share implementation, documentation, etc.
  - Using external libraries is a pain in the kernel.

Why not… just use UML (or LKL)?

- KUnit already runs in userspace via User-Mode Linux (ARCH=um).
  - (And the Linux-Kernel-Library exists as a fork to treat a UML kernel as a library)

But:

- Only works on x86 & x86_64
- Not nearly as lightweight: has to build and boot an entire kernel.
- Doesn't make sense for non-kernel tools
- This is what we're already doing…

How?

kunit.h

Implement a minimal implementation of a subset of the API as a header replacement.

Pros:

- Super-simple. (#define kunit_log(x) printf(x), etc)
- Great for self-contained use-cases
- Easy to compile.

But:

- Missing a lot of features (or overcomplicated)
- Poor support for multi-file tests.
- Limited implementation code reuse.

KUnit 'backends'

Split out all of the kernel-specific bits of KUnit, and implement a userspace backend.

Pros:

- Generic: can be used everywhere.
- Feature-rich: everything which makes sense outside of the kernel can be implemented.

But:

- Where does the non-kernel code live?
  - What about the shared code?
- Lots of questions about how it would work.

Modular KUnit

Make some parts of KUnit modular enough to be used independently.

Pros:

- Can re-use code (e.g., KTAP emitter, stubbing/mocking frameworks) in non-KUnit tests.
- Generally nice to avoid coupling.

But:

- Most KUnit code depends heavily on the struct kunit* (or worse, current->kunit)
- Still have the problems of the above, re: non-kernel code location.

Open Questions

Is this useful?

- It's already being done in some places, so there's definitely some use.
- (We probably should at least standardise that.)

Where should it live?

- The existing include/kunit, lib/kunit directories?
- UAPI headers?
- tools/?
- Somewhere else?

- Depends on how big and complicated it gets.

Tooling to build / run / parse these?

- Lots of standalone binaries: need to know where they are.
  - Alongside code? In a separate (selftests?) directory?
- Makefile targets?
  - Just make them selftests?
  - make (thing)-test?
- Build everything into one test binary à la KUnit with UML
  - If we have the KUnit executor, this simplifies a lot.
  - Can reuse Kconfig or similar?
- Worst-case, we have `kunit.py parse` on results.

Documentation?

- Exactly what should be a:
    - selftest
    - (kernel-mode) KUnit test
    - (userspace) KUnit test
- Definitely some overlap.
- Documentation/dev-tools/testing-overview.rst

Plan of attack?

- Start with the minimal header and go from there?
- Start with the Perf test implementation.
- Extract a minimal version of the KUnit implementation?
- Start refactoring KUnit?
  - Mostly 'string-stream' and some other minimal use of kernel functions.

Something else?

Questions / Comments?

Or visit kunit.dev/ and subscribe to kunit-dev@googlegroups.com