LF *live*

MENTORSHIP SERIES

LF Live MENTORSHIP SERIES

KUnit Testing Strategies

Brendan Higgins, Google

LF live MENTORSHIP SERIES

What's this talk about?

- Testing more "unit-y"
  - Why make tests more "unit-y"?
  - How to test (kernel) code in isolation?
  - What if my code is not testable?
- Cool testing features

Why “unit” test?

What's all this "unit", "integration", "functional", etc?

- Don't get too hung up on it.
- Most kernel tests test user reachable APIs
  - e.g. syscalls, device/proc/... files (/proc/self)
  - That's good! Keep doing that too!
- Internal kernel APIs are often totally untested
  - linked list functions, printf(), etc
  - These are important too!
- **Think of "unit" testing as "in-kernel" testing.**

Why should I test internal APIs?

- Do other people use your APIs?
  - Yes: People should know if they change.
  - No: Other devs should know how they work.
- Do your APIs change a lot?
  - Yes: Testing helps people understand how they change.
  - No: Any change should probably be well tested.

Why should I test internal APIs?

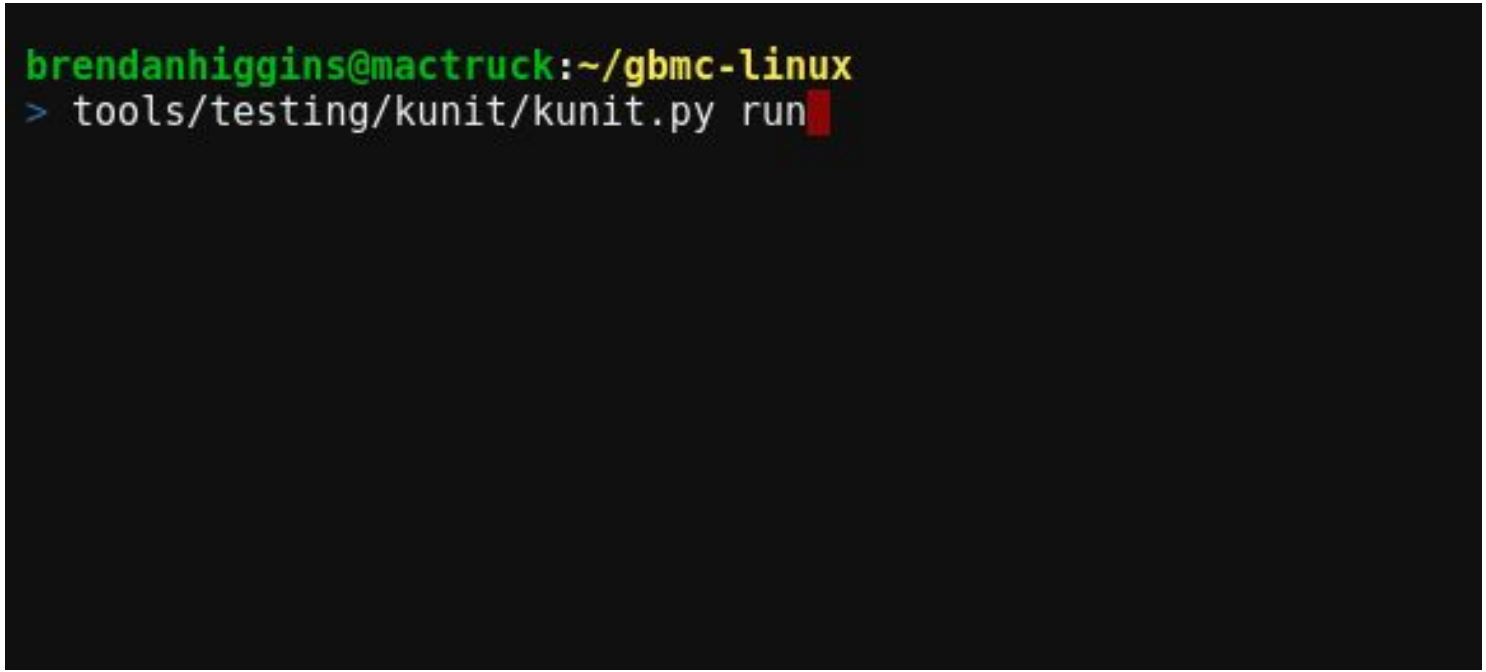
- A common thought is that tests provide stability.
- Tests are also documentation.
  - Documentation gets stale. Tests don’t.
- Tests show how code behaves.
- Code states increase combinatorially with number of functions

How to use KUnit?

Brief overview of KUnit features

- KUnit has two parts:
  - Internal KUnit testing library
  - kunit_tool - python scripts to build, run, get results
  - Can be used separately.

Running KUnit



Running KUnit

```
14:02:08] Building KUnit Kernel ...
14:02:45] Starting KUnit Kernel ...
14:02:49] ============================================================
14:02:49] ======== [PASSED] kunit-try-catch-test ========
14:02:49] [PASSED] kunit_test_try_catch_successful_try_no_catch
14:02:49] [PASSED] kunit_test_try_catch_unsuccessful_try_does_catch
14:02:49] ============================================================
14:02:49] ======== [PASSED] kunit-resource-test ========
14:02:49] [PASSED] kunit_resource_test_init_resources
14:02:49] [PASSED] kunit_resource_test_alloc_resource
14:02:49] [PASSED] kunit_resource_test_destroy_resource
14:02:49] [PASSED] kunit_resource_test_cleanup_resources
14:02:49] [PASSED] kunit_resource_test_proper_free_ordering
14:02:49] [PASSED] kunit_resource_test_static
14:02:49] [PASSED] kunit_resource_test_named
14:02:49] ============================================================
14:02:49] ======== [PASSED] kunit-log-test ========
14:02:49] [PASSED] kunit_log_test
14:02:49] ============================================================
14:02:49] ======== [PASSED] string-stream-test ========
14:02:49] [PASSED] string_stream_test_empty_on_creation
14:02:49] [PASSED] string_stream_test_not_empty_after_add
14:02:49] [PASSED] string_stream_test_get_string
14:02:49] ============================================================
14:02:49] ======== [PASSED] example ========
14:02:49] [PASSED] example_simple_test
14:02:49] ============================================================
14:02:49] Testing complete. 14 tests run. 0 failed. 0 crashed.
14:02:49] Elapsed time: 45.041s total, 3.756s configuring, 37.136s building, 0.000s running
```

KTAP

```
TAP version 14
1..5
    # Subtest: kunit-try-catch-test
    1..2
    ok 1 - kunit_test_try_catch_successful_try_no_catch
    ok 2 - kunit_test_try_catch_unsuccessful_try_does_catch
ok 1 - kunit-try-catch-test
    # Subtest: kunit-resource-test
    1..7
    ok 1 - kunit_resource_test_init_resources
    ok 2 - kunit_resource_test_alloc_resource
    ok 3 - kunit_resource_test_destroy_resource
    ok 4 - kunit_resource_test_cleanup_resources
    ok 5 - kunit_resource_test_proper_free_ordering
    ok 6 - kunit_resource_test_static
    ok 7 - kunit_resource_test_named
ok 2 - kunit-resource-test
    # Subtest: kunit-log-test
    1..1
put this in log.
this too.
add to suite log.
along with this.
    ok 1 - kunit_log_test
ok 3 - kunit-log-test
    # Subtest: string-stream-test
    1..3
    ok 1 - string_stream_test_empty_on_creation
    ok 2 - string_stream_test_not_empty_after_add
    ok 3 - string_stream_test_get_string
ok 4 - string-stream-test
    # Subtest: example
    1..1
    # example_simple_test: initializing
    ok 1 - example_simple_test
ok 5 - example
```

Configuring KUnit

- KUnit tests are selected/configured with Kconfig

.kunitconfig

Configuring KUnit

- KUnit tests are selected/configured with Kconfig
- To make this a bit easier, we support the use of .kunitconfigs
  - .kunitconfig is just a minconfig
  - Used by kunit_tool to configure, build, and run tests

Let’s write a test!

Review

Writing your first test

- Look for a similar test to copy.
- Look in menuconfig for a similar looking test case.
- If all else fails, copy kunit_example_test

Get the test suite working

- Hardest part is usually getting the initial test suite setup
- Start with a really simple “sanity check” - just get the function running
- Adding more test cases is easy

How to write test cases

- Start off with a simple test case
- Test Edge Cases
  - “Test first, test middle, test last”
  - Try testing error conditions
- Test all code paths
  - Check that every condition in test is covered
  - Code coverage tools can be helpful

How to write test cases

- Make tests readable
  - Try not to test too much in a single test case
- Overly repetitive test cases are bad too
- Using helper functions is okay
- Generally follow best coding practices when testing

Questions?

Backup Slides

Other testing strategies

- Managing state
- Making code more testable (driver testing, things with annoying dependencies, etc)

Managing State

- Establish a known starting state
- Perform a manipulation you wish to test
- Verify the new expected state
- See linked list example (lib/list-test.c)

Making Code More Testable

- Dependency Injection
  - Pass in a pointer to difficult dependency
  - In test, replace dependency with fake
- Compiletime Indirection
  - Use macros to change dependencies when testing
- Ftrace
- See doc from Daniel

LF live MENTORSHIP SERIES

Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The LF Mentoring Program is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- Outreachy remote internships program supports diversity in open source and free software
- Linux Foundation Training offers a wide range of free courses, webinars, tutorials and publications to help you explore the open source technology landscape.
- Linux Foundation Events also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at events.linuxfoundation.org.