Storing and Outputting Test Information: KUnit Attributes and KTAPv2

Rae Moar <rmoar@google.com>
Outline

- Background on KUnit and Supplemental Test Information
- KUnit Test Attributes
- How to Output Test Information in KTAP
- KTAP v2
- Questions
What Is KUnit?
What Is KUnit?

- A Unit Testing framework for the Linux Kernel.
- Upstream since 5.5
- Tests are written in C and run in kernel mode
- 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
Background on Test Information
Background on Test Information

● What do we mean by Test Information?
  ○ Not talking about the value of variable "x" used during test execution
  ○ Information primarily for user interaction either before or after test execution (like test name or speed of test)

● Current Norm of Test Information
  ○ Basics (test name, result) very well categorized
  ○ Everything else is pretty unorganized

● Is this a good system?

● Could we instead take that supplemental test information and make it more useful?
Background on Test Information

- There is a wide variety in test information
- Examples:
  - **speed**: The relative speed of a test (enum categories: normal, slow, very slow)
    - Inspiration: CONFIG_MEMCPY_SLOW_KUNIT_TEST
    - How is it useful?
      - Filter out tests based on speed
    - Needs: Manually Stored, Filtering, Outputted in KTAP
  - **is_init**: Whether the test uses init data or functions (bool)
    - How is it useful?
      - Filter out init tests if the init section could be discarded
    - Needs: Auto Generated and Stored, Filtering
Background on Test Information

● One More Example:
  ○ **output_files**: list of file path of auxiliary files that could be generated during test execution (list of strings)
    ■ How is it useful?
      ● Read during output to give extra context to test results
    ■ Needs: Stored during Test Execution, Outputted in KTAP

● What do we need to implement a framework for this test information?
  ○ Store information manually and auto-generated (a variety of data types)
  ○ Filter based on test information
  ○ Output this in a readable and parsable way (in KTAP)
KUnit Attributes
KUnit Attributes

- **Objective:** To save and access supplemental test information
- **Current Attributes:** speed, module name
- **What can you do with KUnit Attributes?**
  - Create attribute of any data type
  - Mark tests (manually or auto-generated)
  - Filter Tests
  - Output to KTAP
- **How to Use:**
  - Add attributes using the struct kunit_attributes
    - or KUNIT_CASE_SLOW()
  - To filter use module param kunit.filter or kunit.filter_skip
    - Tooling Flags: --filter or --filter_skip

```c
static const struct kunit_attributes example_attr = {
    .speed = KUNIT_SLOW,
    .is_flaky = true,
};

static struct kunit_case example_test_cases[] = {
    KUNIT_CASE_ATTR(example_test, example_attr),
};
```
KUnit Attributes Example

Running KUnit with Default Settings:

```
Testing complete. Ran 312 tests: passed: 304, skipped: 8
Elapsed time: 10.352s total, 0.001s configuring, 3.105s building, 7.212s running
```

Running KUnit with Default Settings Plus Filter out 6 Slow Tests:
(Command: ./.../kunit.py run --filter "speed>slow")

```
Testing complete. Ran 306 tests: passed: 298, skipped: 8
Elapsed time: 3.964s total, 0.001s configuring, 3.143s building, 0.814s running
```

**Major Difference in Time!**

- Elapsed Time: **63%** faster
- Running Time: **89%** faster
Potential KUnit Attributes

- **is_init**
  - Whether the test uses init data or functions (bool)
  - To prevent re-running init tests after boot

- **file**
  - File path of suite (i.e. lib/kunit_example.c)
  - Useful when looking up test that is failing

- **custom_tags**
  - A list of strings ("tags") to mark the test that can be personalized to the test suite

- **Parameterized test attributes**
  - Implement a generate_attrs function to assign attributes given a parameter

```c
static struct kunit_attributes example_gen_attrs(const void *test_param)
{
    struct kunit_attributes attrs = {{}};
    const int *val = test_param;

    if (*val == 2)
        attrs.speed = KUNIT_SPEED_SLOW;
    return attrs;
}
```
KTAP Output
How should KUnit Attributes and other test information be formatted in KTAP?

- Identifiable
- Parsable
- May need to add to KTAP v2 to create a specification

KTAP Metadata Proposal ([link](#))

- Metadata lines format
  - "# type: value"
  - Diagnostic line to aid existing parsers
  - Header is "# test: name"

- Location
  - **Suites**: Located between KTAP version line and test plan line (1..x)
  - **Test Cases**: Located prior to test result line

```
KTAP version 2
1..1
   KTAP version 2
   # test: test_suite
   # module: example
   1..2
   ok 1 test_1
   # test: test_2
   # speed: slow
   # test_2: initializing // diagnostic line
   ok 2 test_2
   ok 1 test_suite
```
KTAP Metadata

- Metadata types currently discussed:
  - Test names
  - File path and Module name
  - Speed
  - Config info
  - Supplemental files paths
  - What others should we consider? Does not need to be KUnit Attribute

- Custom metadata types:
  - Would not need to be in spec before using
  - Could use prefixes:
    - ktap_[type]: indicates KTAP specified
    - custom_[type]: indicated not KTAP specified
KTAP v2

- Proposals have been submitted for KTAP v2
  - KTAP Metadata
  - "skip" test result

- All changes on a separate branch within Frank's Tree
  - [https://elinux.org/Test_Results_Format_Notes#KTAP_version_2](https://elinux.org/Test_Results_Format_Notes#KTAP_version_2)

- Open Questions:
  - How should we continue to develop KTAP v2?
  - What is the ideal timeline for KTAP v2?
Open Questions
Open Questions

- What KUnit attributes should we be saving?
- Would generating KUnit attributes be helpful for parameterized tests? Are there specific attributes to add for parameterized tests?
- What are thoughts on the proposed format of KTAP Metadata? What test information should be saved as a specified type of KTAP Metadata?
- How can we support the development of KTAP v2?