KUnit - Linux Kernel Unit Testing

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\kunit\[linux-master][Documentation][dev-tools][kunit]index.rst, line 7)

Unknown directive type "toctree".

```
.. toctree::
        :maxdepth: 2
        :caption: Contents:

        start
        architecture
        run_wrapper
        run_manual
        usage
        kunit-tool
        api/index
        style
        faq
        tips
        running_tips
```

This section details the kernel unit testing framework.

Introduction

KUnit (Kernel unit testing framework) provides a common framework for unit tests within the Linux kernel. Using KUnit, you can define groups of test cases called test suites. The tests either run on kernel boot if built-in, or load as a module. KUnit automatically flags and reports failed test cases in the kernel log. The test results appear in TAP (Test Anything Protocol) format. It is inspired by JUnit, Pythonâ€™s unittest.mock, and GoogleTest/GoogleMock (C++ unit testing framework).

KUnit tests are part of the kernel, written in the C (programming) language, and test parts of the Kernel implementation (example: a C language function). Excluding build time, from invocation to completion, KUnit can run around 100 tests in less than 10 seconds. KUnit can test any kernel component, for example: file system, system calls, memory management, device drivers and so on.

KUnit follows the white-box testing approach. The test has access to internal system functionality. KUnit runs in kernel space and is not restricted to things exposed to user-space.

In addition, KUnit has kunit_tool, a script (`tools/testing/kunit/kunit.py`) that configures the Linux kernel, runs KUnit tests under QEMU or UML (User Mode Linux), parses the test results and displays them in a user friendly manner.

Features

- Provides a framework for writing unit tests.
- Runs tests on any kernel architecture.
- Runs a test in milliseconds.

Prerequisites

- Any Linux kernel compatible hardware.
- For Kernel under test, Linux kernel version 5.5 or greater.

Unit Testing

A unit test tests a single unit of code in isolation. A unit test is the finest granularity of testing and allows all possible code paths to be tested in the code under test. This is possible if the code under test is small and does not have any external dependencies outside of the test's control like hardware.

Write Unit Tests

To write good unit tests, there is a simple but powerful pattern: Arrange-Act-Assert. This is a great way to structure test cases and defines an order of operations.

- Arrange inputs and targets: At the start of the test, arrange the data that allows a function to work. Example: initialize a statement or object.
- Act on the target behavior: Call your function/code under test.
- Assert expected outcome: Verify that the result (or resulting state) is as expected.

Unit Testing Advantages

- Increases testing speed and development in the long run.
- Detects bugs at initial stage and therefore decreases bug fix cost compared to acceptance testing.
- Improves code quality.
- Encourages writing testable code.

How do I use it?

- Documentation/dev-tools/kunit/start.rst - for KUnit new users.
- Documentation/dev-tools/kunit/architecture.rst - KUnit architecture.
- Documentation/dev-tools/kunit/run_wrapper.rst - run kunit_tool.
- Documentation/dev-tools/kunit/run_manual.rst - run tests without kunit_tool.
- Documentation/dev-tools/kunit/usage.rst - write tests.
- Documentation/dev-tools/kunit/tips.rst - best practices with examples.
- Documentation/dev-tools/kunit/api/index.rst - KUnit APIs used for testing.
- Documentation/dev-tools/kunit/kunit-tool.rst - kunit_tool helper script.
- Documentation/dev-tools/kunit/faq.rst - KUnit common questions and answers.