Unit tests for Linux kernel hackers

How to avoid hours of painful debugging

Mihir Mehta

Systems Core Group
Samsung Research Institute - Noida

mihir.mehta@samsung.com

February 3, 2014

Overview

Overview of the material.

- What are unit tests?
- Why unit tests?
- Accessing unit test results
- Summary

What are unit tests?

What are unit tests?

- Ideally, a unit test involves calling a *single* function and *verifying* its return value against an *expected* value.
- Running a test can have two outcomes.
  1. Pass: Congrats! Your function works correctly on this test input!
  2. Fail: This again raises multiple outcomes:
     - 2.1 Function body is wrong. (bad, but manageable if you know what you're doing.)
     - 2.2 Expected value is wrong (indicates you don't know what you're doing, i. e. very bad.)

Why unit tests?

- Central component of test-driven design.
- Provide convenient way to verify our code.
- Save manual labour (i.e. less time at desk!)
- Crucial: help prevent regressions.

Accessing unit test results

- In a userspace program, (i.e. what we're used to) it's simple to make a separate test executable and link the functions we are testing into this executable.
- This way, we maintain a logical separation of the test code from the codebase itself.
- We also avoid bloating our executables.
- Making a separate test executable is not possible within kernel code - why?

Accessing unit test results

- Making a separate test executable is not possible within kernel code - why?
  - Can't be a userspace program - userspace programs can't link to kernel code (privilege levels and other issues)
  - Can't be a kernel - how are you going to run two kernels at once?
- So, how do we implement unit tests for kernel code now?

Accessing unit test results

- So, how do we implement unit tests for kernel code now?
  - Kernel modules!
- With modules, we can easily access kernel functions and test them.
- We can run all the tests at once, in the init function of the module, after setting a load priority for the module.
- If we use loadable modules (instead of static modules), we can avoid kernel bloat too - simply unload the module when done.