CE Workgroup

Test Standards – Can Fuego, Lava and others agree?

September 2017

Tim Bird

Architecture Group Chair

LF Core Embedded Linux Project

Outline

- Open source tests and test frameworks for Linux:
  - kselftest, LTP, KernelCI, LAVA, Fuego, Avacado, kerneltest, zero-day and more...
- Standards:
  - To Share infrastructure and Interoperate.
  - Areas:
    - Test dependencies
    - Results formats
    - Board control hardware.
      - Interfaces to commonly-used utility programs

Open source testing

- Lots of test frameworks
- Still too much left as an exercise to the tester:
  - What tests to run?
  - How to perform the test?
    - How to build the test?
    - What parameters to use?
  - Test dependencies
  - Test results
    - Results collection
    - Visualization
  - Interpretation and analysis
    - What do results mean? What is important to look at? What result should I expect on my board?
  - How to automate board control

Tests and Frameworks

- Kselftest
- LTP
- KernelCI
- LAVA
- Fuego
- Avacado
- kerneltest
- zero-day

Kselftest

- Unit test system inside kernel source tree
- Recent work:
  - Lots more regression tests (preferred place for syscall compatibility/regression tests (over LTP)
  - Converting to TAP (Test Anything Protocol) for test output
  - Support for “make O=<somedir>” (KBUILD_OUTPUT)

LTP – Linux Test Project

- A huge collection of tests for Linux
  - Lots of different areas covered: syscalls, realtime, posix, etc.
- Some unification of results output
- Fairly complex to build, deploy
- Very difficult to interpret results
  - Lots of failure on most boards, due to configuration, environment, etc.
  - Tester has to know what to ignore, and why

Fuego

- Framework for collaborating on tests and test infrastructure for Linux
- V1.1 features (April 2017)
  - Upgrade to latest Jenkins
  - Test script refactoring
  - Fuego container directory layout change
  - About 40 new tests
- V1.2 plans (coming soon)
  - Unified output format
    - Convert all test results to JSON – KernelCI compatible
  - Support LAVA as a transport & board manager
  - Test dependency system

Kernelci.org

- Massive build/boot testing for top-of-tree kernel
  - Builds hundreds of trees continuously, then reports any errors
  - In many different labs
- http://kernelci.org
- Presentations:
  - ELC and ELCE 2016 – by Kevin Hilman
  - Linaro Connect:
    - Kernelci and lava update - See https://lwn.net/Articles/716600/
- The most successful public, distributed build and test system for Linux, in the world!

LAVA

- Linaro Automation and Validation Architecture
- Good board control and job scheduling
- V2
  - Job files now use Jinja2 templates
    - Was previously hand-written JSON
  - Jobs are run asynchronously, without polling,
  - ZeroMQ is used for communications.
  - ReactOBus is used to run jobs from messages.
  - Requires more explicit board configuration

Kerneltests

- Builds all architectures and boots on many (if there’s a qemu for the platform), on a daily basis
  - 14 architectures, 113 platforms
- Summary report for stable release candidates
- Results at kerneltest.org

0-day

- Large set of tests that are run daily on top-of-tree
- Large test bed
- Reports build test failures for individual patches contributed to kernel mailing lists
  - Bisects to isolate defective code
  - e-mails authors before maintainer gets to the patch
  - 60% of failures reported in 2 hours, 90% in 24h

Avacado

- Virtual machine tester
- Lots of interesting features
  - test server
  - matrix testing
  - multiple results format outputs
  - Simple interface to Jenkins

Investigation vs Proposals

- Investigation
  - Things I’m still researching in the industry:
    - List of tests to run
    - Test dependencies
    - Board control
- Proposals
  - Things I’d like to propose standardizing on
    - Test Output Format
    - Test Results Format
      - TGUID
      - kernelCI (test_suite/test_set/test_case/measure)

List of tests to perform

- Why needed?
  - Different boards and different use cases require different sets of tests
  - Different phases of testing require different tests (or different test parameters)
    - e.g. quick vs comprehensive
- Fuego has: testplan
  - json file indicating tests to run, specs, timeouts
  - Some plans:
    - For AGL (automotive grade Linux)
    - For LTSI (long-term stable kernel initiative)
    - For generic kernel testing

Test dependencies

- Why needed?
  - To avoid wasting time with tests that won’t work for a given platform
  - To document pre-requisites for a test
- What kind of dependencies:
  - memory
  - kernel configuration
  - storage
  - sub-systems and libraries
  - hardware

Existing support

- 0day:
  - need_kernel_headers: true
  - need_kconfig
  - need_memory
  - need_cpu – number of CPUs
- Fuego:
  - NEED_MEMORY
  - NEED_FREE_STORAGE
  - NEED_KCONFIG
- Others?

Dependencies – Notes

- Both 0day and Fuego use declarative syntax
  - Suitable for static analysis
  - Important for scalability
    - Does not require test execution, or even test installation
- Envision an online “test store” with tests that can be matched against board characteristics
  - Tests for specific hardware (e.g. CAN bus)
- Fuego also has some imperative checks:
  - assert_define - a test variable is defined
  - is_on_target - target has a file, library or program
  - is_on_sdk - the sdk has a required library or header

Proposals

- Preferred test output format:
  - TAP13
- Test results format:
  - TGUID
  - KernelCI:
    - Test_suite, test_set, test_case, measurement
- Fuego:
  - Run.json, criteria.json

TGUID - Test globally unique id

- Define a string that uniquely identifies a particular testcase or benchmark measure
  - Ex: LTP.syscall.abort01.1
  - Ex: bonnie.Sequential_Output.Block.speed
  - Ex: Interbench.Video.Write
- Useful for data and information interchange
- Similar to web's URL
- Can refer unambiguously to a test case
  - Some issues with this (LTP test types (syscall) are really like test_sets)
  - Aliasing and nesting
    - Is unlimited nesting allowed?

TGUID benefits

- Good for discussion
- Good for data mining across frameworks
  - Can identify problematical tests
- Can have meta-data about a test case independent of the framework
  - Descriptions
  - Analysis
- The first step to sharing information is a consistent reference mechanism for shared objects

Output format

- The output from the actual test
- Should be human readable, but machine parsable
- Is really ad-hoc
  - Testers just use whatever they feel like
  - Luckily, many are line-oriented, and have fixed strings corresponding to results (ie. PASS, FAIL, Error, etc.)
- Kseltest adopting TAP (Test Anything Protocol)
  - Specifically TAP13 - https://testanything.org/tap-version-13-specification.html

TAP – Test Anything Protocol

- See https://testanything.org/
- Very simple
  - Plan (1..n) line indicates number of tests
  - Test line has result (‘ok’ or ‘not ok’), test number, description
- Example:

```
1..4
ok 1 - Input file opened
not ok 2 - First line of the input valid
ok 3 - Read the rest of the file
not ok 4 - Summarized correctly # TODO Not written yet
```

Results formats (existing)

- Xunit (junit)
  - XML
  - lists results counts, and error information
    - Oddly missing PASS results for individual testcases
- Kernelci
  - Test_suite, test_set, test_case, measurement
  - Is really the kernelci json API
    - See https://api.kernelci.org/schema-test-suite.html

Results parsing

- Abstraction for converting non-standard test output to standard results format:
- Fuego:
  - log_compare() – simple line-oriented parsing
  - parser.py() – arbitrarily complex parsing
    - input = test program output (test log)
    - output = dictionary of {tguid: result}
      - result: for measure is numeric, for testcase is PASS, FAIL, or SKIP
  - System constructs run.json with results for test run
  - Uses criteria.json file to determine status of test
    - Can specify ignored failures
- LAVA/KernelCI: ???

Board and test environment control

- Power control
- File transfer
- Remote execution
- Hardware control
  - Bus control
  - Buttons, keys

Interface to external functions

- Tools that provide abstractions:
  - wic/mic – image preparation
  - pduclient – power control
  - ttc – Sony's board management abstraction tool
- Core interfaces:
  - Power control
  - Kernel install
  - Distro install
  - File get/put
  - Execute command
  - Button control
  - Bus control

LAVA core board control operations

- power_off_command
- power_on_command
- connection_command
- hard_reset_command
- ... other _commands

ttc

- Define a core set of commands for operating with a target
  - get_kernel, get_config, kbuild, kinstall, fsbuild, fsinstall, reset, reboot, copy_to, run, copy_from, console, login, rm
- Thin wrapper for abstracting board-specific operations:
  - Fuego has a model of direct interaction with the target
  - LAVA appears to have a “setup and go” model

power control

- LAVA
  - pduclient
  - snmp_pdu_control
  - pdu_control_off
  - ipmi_tool
- TTC
  - power_control
  - echo (to usb-serial ports with specialized interpreters)
  - web-relay

file transfer

- Android: adb put/get
- LAVA:
  - scp, ser2net
- Fuego: ov_transport_get, ov_transport_put
  - Using serio, scp, and cp
- ttc: copy_to_cmd, copy_from_cmd
  - Using scp, cp

command execution

- Android: adb run
- LAVA: connection_command
  - usually using ser2net and telnet
- Fuego: ov_transport_cmd
  - usually using ssh
- TTC: run_cmd
  - usually using ssh_exec or telnet_exec

Un-standardized board control

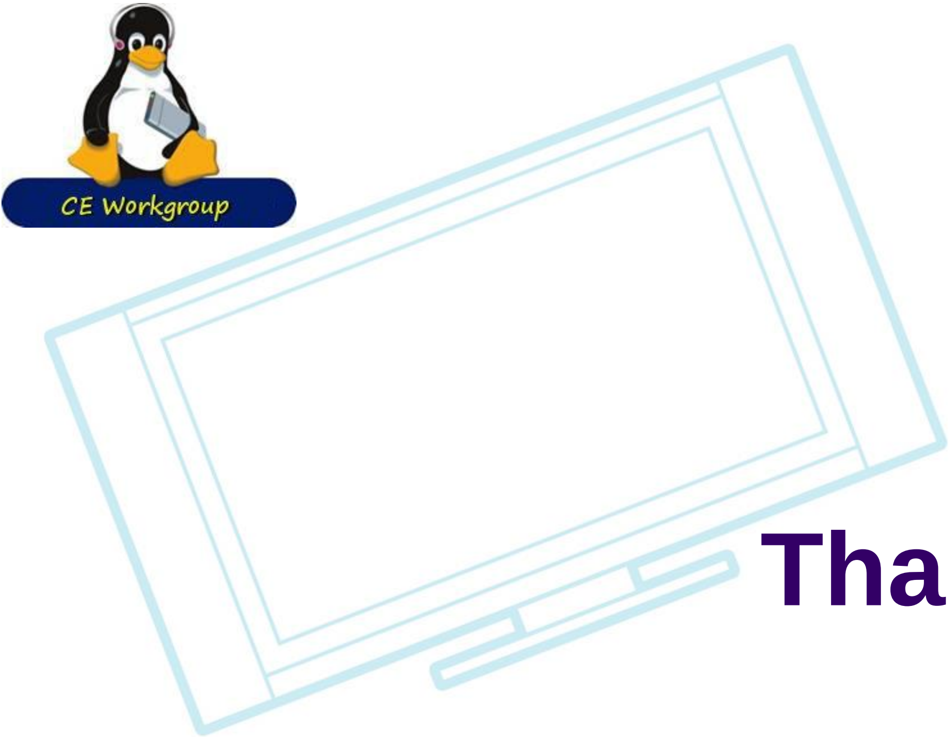
- Both Fuego and LAVA appear to be missing button and bus control
- This is required for lots of hardware tests
  - plug & unplug devices
  - USB switching
  - complex boot modes on production devices
    - e.g. phone 3-button resets
  - re-route devices
    - So a machine can load data or prepare file systems separate from DUT

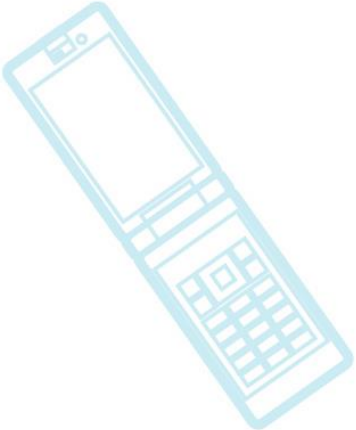
Other areas

- Test descriptions?
  - Human interpretation of results
- criteria files?
  - What tests should you expect to fail?
  - What tests are flaky and sometimes fail incorrectly?
- board variables

Next steps?

- How to actually standardize something?
- Just start using the same things and hope the industry notices? (defacto standards)
- Produce a spec?
- Contribute support for a standard to other frameworks?
  - They are Open Source projects, after all
- Plan an event or summit to coordinate.





Thanks!