Keeping up with LTS Linux Kernel Functional Testing on Devices

Tom Gall
Director, Linaro Mobile Group
Who is Linaro?

- Linaro is leading software collaboration in the ARM ecosystem
- Instead of duplicating effort, competitors share development costs to accelerate innovation and time to market
- Linaro is member funded and delivers output to members, and into open source projects
Open Source Project Contributions - Partial List
Linux Kernels on Devices

Quick review about upstream...

- Android Common
  - Tracks LTS (4.4, 4.9, 4.14)
  - Tracks Mainline
And then you see this...

- 4.4.13 is positively ancient
  - Released: June 8th 2016
- 4.4.78 better but
  - Released: July 21st 2017
- Security fixes are being cherry picked, however
  - LTS security fixes aren’t necessarily labeled as security fixes
  - LTS tests with all patches in an LTS release, not some cherry pick
  - Cherry picking can entirely miss complicated interactions where other patches were required
Project Sharp Introduction
Making Community and Android Kernels Better

- Catch kernel regressions across architectures and kernel versions before they make it into LTS releases or Android Common
  - 4.4, 4.9, 4.14, current stable, mainline
  - X86_64, ARMv7, ARMv8
  - GCC and soon clang
  - 48 hour window (build -> results -> triage -> bisect)
- Help make more older LTS kernels more viable
- Examine communities for fixes
- Display testing data and test histories
- Empower developers
- Triage problems
- Add to kernel testing effort
## LKFT compared with KernelCI

*LKFT and KernelCI will cautiously converge when/where it makes sense*

<table>
<thead>
<tr>
<th>LKFT</th>
<th>KernelCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Testing as a first-order design requirement</td>
<td>Boot Testing as a first-order design requirement</td>
</tr>
<tr>
<td>Full userspace</td>
<td>Minimal Userspace</td>
</tr>
<tr>
<td>Functional Test Coverage</td>
<td>Boot Test quickly</td>
</tr>
<tr>
<td>Limited hardware due to userspace</td>
<td>Larger class of hardware supported</td>
</tr>
<tr>
<td>requirements</td>
<td>Community Consensus Driven</td>
</tr>
<tr>
<td>Linaro Member Needs Driven</td>
<td>Linux Community Goal Driven</td>
</tr>
<tr>
<td>Linaro Member Goal Driven - Sharp, extend LTS, LSK testing, et al.</td>
<td>Can functional test w/ minimal userspace</td>
</tr>
<tr>
<td>Does boot-test limited hardware</td>
<td>Limited by pace of community consensus</td>
</tr>
<tr>
<td>Limited only by Linaro &amp; member development pace</td>
<td>Open Devices only</td>
</tr>
<tr>
<td></td>
<td>Cannot publish results under access control</td>
</tr>
</tbody>
</table>
LKFT

LKFT - Linux Kernel Functional Test framework.

The mission of LKFT is to perform functional regression testing on select Linux kernel branches in real time (as they’re updated) and report any regressions as quickly as possible. This is performed by executing a variety of functional-tests on a selection of user-space environments such as Open Embedded and Android.

The goals of LKFT are to shorten derivative Linux kernel release intervals, increase the confidence of upstream Linux kernel engineers in the quality of their releases, and increase the confidence of downstream adopters of those Linux kernel trees. Ultimately the goal is that LKFT will encourage downstream hardware vendors to more frequently update the Linux kernel that runs on their devices in order for consumers to benefit from bug and security updates.
LKFT System Overview

1. Upstream/Internal tree changes
2. Fetch git kernel tree repo
3. Build system images
4. Publish image builds to snapshot server
5. Submits jobs to the Labs (LAVA - Linaro Automation Validation Architecture)
6. LAVA request build download
7. Schedule jobs on target hardware
8. Perform tests on target hardware
9. Store results to LAVA database
10. Results made available on LAVA frontend
11. Qa-reports pulls Results data from LAVA database
12. Present results in qa-reports dashboard
13. Send Email reports
LKFT Infrastructure

The infrastructure for LKFT is composed of several autonomous components:

- Commit triggered image building by using a Jenkins instance to build OE & AOSP images and submit jobs to LAVA: [https://ci.linaro.org/](https://ci.linaro.org/)

- Device automation to support scheduling, image flashing, automated testing, and results gathering (and storage) via a dedicated LAVA instance: [https://lkft.validation.linaro.org](https://lkft.validation.linaro.org)

- Email reporting and results dashboard via a dedicated Squad instance: [https://qa-reports.linaro.org/lkft](https://qa-reports.linaro.org/lkft)  [https://qa-reports.linaro.org/android-lkft](https://qa-reports.linaro.org/android-lkft)
When an RC occurs

4.4, 4.9, 4.14, 4.15, mainline, next

- 1 build for each architecture/board combo
- 20 LAVA test jobs per kernel version
- 5572 individual tests per kernel version
## What hardware is in use?

### Scheduler Status

<table>
<thead>
<tr>
<th>Overall status</th>
<th>Devices</th>
<th>TestJobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online devices</td>
<td>All devices</td>
<td>All Jobs</td>
</tr>
<tr>
<td>Passing health checks</td>
<td>Active Devices</td>
<td>Queued Jobs</td>
</tr>
<tr>
<td>Running test jobs</td>
<td>Devices Health</td>
<td>Active Jobs</td>
</tr>
</tbody>
</table>

### Device Type Overview

<table>
<thead>
<tr>
<th>Name</th>
<th>Idle</th>
<th>Offline</th>
<th>Busy</th>
<th>Restricted</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>b2260</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dragonboard-410c</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi6220-hikey</td>
<td></td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>junco-r2</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qemu</td>
<td>19</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x15</td>
<td>5</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x86</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experience with Devices

- 96Boards an obvious ARM platform
  - Small form factor
  - Suited to large scale deployments

- Reliable connectivity costs money
  - High quality, shielded USB cables
  - Reliable, software controllable, USB hubs

- Firmware updates cost engineering time
  - Changes in interaction breaks automation

- Scaling up challenges
  - Four cables per board
    - Serial, USB OTG, Ethernet and power
    - Power bricks take space
    - Solutions being sought
kselftest - Linux Kernel Testing Framework

https://kselftest.wiki.kernel.org

- Use the latest stable version of the test against all LTS kernel releases
  - This was somewhat controversial
  - Can be challenging due to failures caused by mismatched versions
  - Upstream isn’t always interested in running this combination or addressing issues discovered by it
- Up to various kernel maintainers to either use or ignore
- Testcase consistency (design, setup, running)
- Reporting infrastructure could be improved. (TAP13)
- Pushed many patches to improve testing infrastructure and address obvious bugs
- A good start to kernel testing, we’d like to see more focus on it’s improvement
LTP - Linux Test Project

https://linux-test-project.github.io

- We don’t run the entire set due to suitability
  - 19 suites currently in use (syscalls, timers, …)
- Test suite is updated every 4 months as per upstream releases (latest 20180118)
- We have a CI loop with LTP master running on mainline to improve future releases
Experiences with ‘complicated’ test suites

- Automation of test runs?
  - Running ‘tradefed family’ tests (VTS, CTS) requires host side.
  - Some LTP tests make hidden assumptions about the hardware they run on.
  - Running pre-built version of kselftests brings a lot of compatibility issues.

- Reporting?
  - There is no unified standard for reporting results/logs.
  - VTS logs are reported differently than CTS even though they use the same shell (tradefed).
  - Kselftests logs are saved in /tmp.
  - Kselftests apparently support TAP13, but not all tests implement this approach (*).

- Skipped tests
  - There are a lot of tests failing on arm/arm64.
  - Tests make assumptions which are not always met (for example sources of entropy).
Experiences with Triaging Android

- Android Common has mainline, 4.4, 4.9, 4.14
  - A set of (decreasing in size) out of tree kernel patches are included in the mix
- On Android we don’t run the exact same of tests as Open Embedded
  - LTP has a number of tests designed specifically for Linux
  - Dependencies not satisfied, etc
- VTS does run a subset of kselftest, LTP
- CTS is uniquely an Android testsuite
  - User space tests can push the kernel in interesting ways Ex: just using the network or BT
- Open Embedded (currently) leads the charge to look for kernel regressions, class of failures detected tend to be Android specific
Keeping up with LTS

Expectations

- 4.4, 4.9, 4.14 generally have 1, maybe 2 cycles per week
  - Couple dozen patches to couple hundred
- Patches included in RC have 48 hours
- Build -> Run -> Report Results
  - Triage Errors -> Bisect -> Fix
- Schedule is lose (on purpose!)
- RC branches are rebased frequently, making building and reproducibility tricky
Example: Pushing results upstream

Summary

kernel: 4.4.121
git branch: linux-4.4.y
git commit: 8b5ab55d254f36e689b1b53aee7223d2d102483e
git describe: v4.4.121
Test details: https://qa-reports.linaro.org/infra/linux-stable-rc-4.4-oe/build/v4.4.121

No regressions (compared to build v4.4.120-37-gce7ba34ae77c)

Boards, architectures and test suites:

juno-v2 - arm64
* boot - pass: 20,
  * kselftest - pass: 34, skip: 29,
  * libhugetlbfs - pass: 90, skip: 1,
  * ltp_cap_bounds-tests - pass: 2,
  * ltp Containers-tests - pass: 28, skip: 53,
  * ltp-fcntl-locktests-tests - pass: 2,
  * ltp-filecaps-tests - pass: 2,
  * ltp-fs-tests - pass: 61, skip: 2,
  * ltp-fs_bind-tests - pass: 2,
  * ltp-fs_perms_simple-tests - pass: 19,
  * ltp-fsfx-tests - pass: 2,
  * ltp-hugetlb-tests - pass: 22,
  * ltp-io-tests - pass: 3,
  * ltp-ipc-tests - pass: 9,
  * ltp-math-tests - pass: 11,
  * ltp-npti-tests - pass: 2,
  * ltp-pty-tests - pass: 4,

- stable@vger.kernel.org
- Goal: Quick summary
  ○ Or Bisected failure
Example: a test report, also available via email

```markdown
lkft » linux-stable-rc-4.14-oe

<table>
<thead>
<tr>
<th>Project Summary</th>
<th>Builds</th>
<th>Metrics</th>
</tr>
</thead>
</table>

**Last build** - v4.14.20-168-g20a80dd2bb00 Feb. 21, 2018, 1:26 p.m. 8 hours ago

<table>
<thead>
<tr>
<th>git branch</th>
<th>linux-4.14.y</th>
</tr>
</thead>
<tbody>
<tr>
<td>git commit</td>
<td>20a80dd2bb095f1b2ae2e72143f12ff8b605382</td>
</tr>
<tr>
<td>git describe</td>
<td>v4.14.20-168-g20a80dd2bb0</td>
</tr>
<tr>
<td>make_kernelversion</td>
<td>4.14.21-rc1</td>
</tr>
</tbody>
</table>

**Latest builds**

<table>
<thead>
<tr>
<th>Build</th>
<th>Test Runs</th>
<th>Completed</th>
<th>Tests</th>
<th>Pass</th>
<th>Skip</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>v4.14.20-168-g20a80dd2bb0</td>
<td>80</td>
<td>80</td>
<td>6259</td>
<td>5571</td>
<td>668</td>
<td>5 hours ago Feb. 21, 2018, 4:20 p.m.</td>
</tr>
<tr>
<td>v4.14.20-119-g1b1ab1d5c50b</td>
<td>40</td>
<td>40</td>
<td>3120</td>
<td>2789</td>
<td>331</td>
<td>1 day, 6 hours ago Feb. 20, 2018, 3:18 p.m.</td>
</tr>
<tr>
<td>v4.14.20</td>
<td>80</td>
<td>79</td>
<td>6246</td>
<td>5560</td>
<td>668</td>
<td>4 days, 1 hour ago Feb. 17, 2018, 8:17 p.m.</td>
</tr>
<tr>
<td>v4.14.20</td>
<td>80</td>
<td>1 incomplete</td>
<td>26.145</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Example: a test report

lkft » linux-stable-rc-4.14-oe

<table>
<thead>
<tr>
<th>Project Summary</th>
<th>Builds</th>
<th>Metrics</th>
</tr>
</thead>
</table>

**Last build** - v4.14.20-168-g20a80dd2bb0 Feb. 21, 2018, 1:26 p.m. 8 hours ago

<table>
<thead>
<tr>
<th>git branch</th>
<th>linux-4.14.y</th>
</tr>
</thead>
<tbody>
<tr>
<td>git commit</td>
<td>20a80dd2bb095f1b2ae2e72143f12ff8b605382</td>
</tr>
<tr>
<td>git describe</td>
<td>v4.14.20-168-g20a80dd2bb0</td>
</tr>
<tr>
<td>make_kernelversion</td>
<td>4.14.21-rc1</td>
</tr>
</tbody>
</table>

**Latest builds**

<table>
<thead>
<tr>
<th>v4.14.20-168-g20a80dd2bb0</th>
<th>80 tests</th>
<th>80 completed</th>
<th>6268 tests</th>
<th>5571 pass</th>
<th>668 skip</th>
<th>26.145</th>
<th>5 hours ago</th>
<th>Feb. 21, 2018, 4:20 p.m.</th>
</tr>
</thead>
<tbody>
<tr>
<td>v4.14.20-119-g1b1ab1d5c50b</td>
<td>40 tests</td>
<td>40 completed</td>
<td>3120 tests</td>
<td>2789 pass</td>
<td>331 skip</td>
<td>25.894</td>
<td>1 day, 6 hours ago</td>
<td>Feb. 20, 2018, 3:16 p.m.</td>
</tr>
<tr>
<td>v4.14.18-23-g8d861f5b27b0</td>
<td>81 tests</td>
<td>80 completed</td>
<td>6258 tests</td>
<td>5562 pass</td>
<td>668 skip</td>
<td>25.255</td>
<td>1 week, 5 days ago</td>
<td>Feb. 9, 2018, 9:16 p.m.</td>
</tr>
</tbody>
</table>
Example: Where is it failing?

Test results

<table>
<thead>
<tr>
<th>Test suite</th>
<th>Platform</th>
<th>Total tests</th>
<th>Pass</th>
<th>Skipped</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltp-syscalls-tests</td>
<td>x15 - arm</td>
<td>1150</td>
<td>1051</td>
<td>97</td>
<td>2</td>
</tr>
<tr>
<td>ltp-syscalls-tests</td>
<td>juno-r2 - arm64</td>
<td>1150</td>
<td>999</td>
<td>149</td>
<td>2</td>
</tr>
<tr>
<td>ltp-syscalls-tests</td>
<td>hi6220-hikey - arm64</td>
<td>1150</td>
<td>996</td>
<td>152</td>
<td>2</td>
</tr>
<tr>
<td>ltp-syscalls-tests</td>
<td>x86_64</td>
<td>1150</td>
<td>1030</td>
<td>118</td>
<td>2</td>
</tr>
</tbody>
</table>

156/156 test suites with no failures hidden Show
Example: How about a log?

Lkft » linux-stable-rc-4.14-oe » Build v4.14.18-23-g8d861f5b27b0 » Test run 115078 » Test results for ltp-syscalls-tests

Test environment: x15 - arm

Suite: ltp-syscalls-tests 20180118

Test results

- fanotify06 — FAIL
- runltp_syscalls — FAIL
Example: What does the trend look like?

<table>
<thead>
<tr>
<th>Build</th>
<th>Date</th>
<th>juno-r2 - arm64</th>
<th>hi6220-hikey - arm64</th>
<th>x15 - arm</th>
<th>x86_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>v4.14.18-23-g8d861f5b27b0</td>
<td>Feb. 9, 2018, 2:52 p.m.</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
<tr>
<td>v4.14.18-19-g44b8fc264b98</td>
<td>Feb. 8, 2018, 3:58 p.m.</td>
<td>pass</td>
<td>pass</td>
<td>n/a</td>
<td>skip</td>
</tr>
<tr>
<td>v4.14.17-65-g81d0cc85caab</td>
<td>Feb. 7, 2018, 9:51 p.m.</td>
<td>pass</td>
<td>pass</td>
<td>skip</td>
<td>skip</td>
</tr>
<tr>
<td>v4.14.17-65-g161cd48d7ece</td>
<td>Feb. 5, 2018, 6:51 p.m.</td>
<td>pass</td>
<td>pass</td>
<td>skip</td>
<td>skip</td>
</tr>
<tr>
<td>v4.14.17</td>
<td>Feb. 4, 2018, 12:32 p.m.</td>
<td>pass</td>
<td>pass</td>
<td>skip</td>
<td>skip</td>
</tr>
<tr>
<td>v4.14.16-157-gc9c30584bc1c</td>
<td>Feb. 2, 2018, 3:13 p.m.</td>
<td>pass</td>
<td>pass</td>
<td>skip</td>
<td>skip</td>
</tr>
</tbody>
</table>
Example: So we create a bug ...
Example: Turns out it was a test case issue...

This commit 28507e514c(safe_mount: Do not try mount() syscall for FUSE fs)
involves FUSE fs check in safe_mount(), so we should give the "fs_type" when
calling that in case the system kill our program.

```
    cmdline="fanotify06"
    contacts=""
    analysis=exit
    <<<test_output>>>
    tst_test.c:980: INFO: Timeout per run is 0h 10m 00s
    tst_test.c:1025: BROK: Test killed by SIGSEGV
```

Signed-off-by: Li Wang <liwang@redhat.com>
---

testcases/kernel/syscalls/fanotify/fanotify06.c | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/testcases/kernel/syscalls/fanotify/fanotify06.c b/testcases/kernel/syscalls/fanotify/fanotify06.c
index e63e457..8cbelaad 100644
--- a/testcases/kernel/syscalls/fanotify/fanotify06.c
+++ b/testcases/kernel/syscalls/fanotify/fanotify06.c
@@ -221,7 +221,7 @@ void test01(void)
 static void setup(void)
 {
    SAFE_Mkdir(MOUNT_NAME, 0755);
-   SAFE_MOUNT(MOUNT_NAME, MOUNT_NAME, NULL, MS_BIND, NULL);
+   SAFE_MOUNT(MOUNT_NAME, MOUNT_NAME, "none", MS_BIND, NULL);
    mount_created = 1;
    SAFE_CHDIR(MOUNT_NAME);
```
Hi Michal -

We (Linaro) run the libhugetlbfs test suite continuously against mainline and recently (Feb 1), the 'counters' test started failing on with the following error:

```bash
root@localhost:~# mount_point="/mnt/hugetlb/
root@localhost:~# echo 200 > /proc/sys/vm/nr_hugepages
root@localhost:~# mkdir -p "/mnt/hugetlbfs"
root@localhost:~# mount -t hugetlbfs hugetlbfs "$mount_point"
root@localhost:~# export LD_LIBRARY_PATH=/root/libhugetlbfs/libhugetlbfs-2.20/obj64
root@localhost:~# /root/libhugetlbfs/libhugetlbfs-2.20/tests/obj64/counters
Starting testcase "/root/libhugetlbfs/libhugetlbfs-2.20/tests/obj64/counters", pid 3319
Base pool size: 0
Clean...
FAIL   Line 326: Bad HugePages_Total: expected 0, actual 1
```

Line 326 refers to the test source @
https://github.com/libhugetlbfs/libhugetlbfs/blob/master/tests/counters.c#L326

I bisected the failure to this commit. The problem is seen on multiple architectures (tested x86-64 and arm64).

Thanks,
Dan
Getting involved

- **Linux-stable**
  - LTS RCs, testing results, candidate patches
  - Mailing List: stable@vger.kernel.org

- **Kselftest**
  - [https://kselftest.wiki.kernel.org](https://kselftest.wiki.kernel.org)
  - linux-kselftest@vger.kernel.org

- **LTP**
  - [https://linux-test-project.github.io](https://linux-test-project.github.io)
Making the Universe Better

In Summary

- Finding kernel regressions is important
  - More boards
  - More eyes
- Exercise more kernel functionality
  - More tests!
  - More testing!
KernelCI Capabilities Compared to LKFT

Why LKFT and not a functional test framework extension of KernelCI?

At the time LKFT was created KernelCI did not have any aspirations for functional test (or they weren’t public).

From the beginning LKFT has been focused on functional testing specific kernel trees (that match Linaro’s membership motivations).

Even now, as support for kselftest is being added to KernelCI, there is minimal filesystem support, so it does not yet match, 1-for-1, the functional test capabilities of LKFT.
LKFT Mission & Reach

As part of Linaro’s mission to improve the Arm architecture ecosystem, the LKFT team reports discovered regressions to Linaro kernel developers, Linaro members, and upstream Linux kernel engineers.

It is important to the Arm ecosystem that Linaro also fix as many failures as are found. The LKFT team invests time into identifying, reporting, and fixing upstream kernel regressions, identifying kernel regressions in select member-hardware SoC (system-on-a-chip) trees, fixing test-suites by contributing to upstream testing projects, fixing kernel configurations, improving full OS stack integration (firmware, kernel, userspace), and improving Arm device automation integration.
lkft.linaro.org and qa-reports.linaro.org

https://lkft.linaro.org is a website for kernel engineers, business partners, and managers to get up-to-date information on functional test results against the latest commits to a variety of Linux kernel source trees.

https://qa-reports.linaro.org/lkft is a website that provides full details of the latest and historical functional test results, as well as a variety of comparison and reporting tools. Its purpose is to aide kernel triage engineers in discovering the cause of functional test failures.
- Qemu
- Generic x86
  - X86-64 64 bit
- TI Beagleboard X15
  - AM5728 32bit A15
- ARM Juno
  - 64 bit Axx/Axx/Mali
- Lemaker HiKey
  - HiSilicon Octa 64 bit A53/Mali