KUnit: New Features and New Growth

Brendan Higgins <brendanhiggins@google.com>
David Gow <davidgow@google.com>
Roadmap

- Growth Since Last Year
- Blockers and Problems
- Features Added
- Non-Technical Growth Efforts
Growth and Usage Stats
Growth Since Last Year

● v5.8 released on Aug 2 2020
  ○ 10 test files
  ○ 9 test contributors
  ○ 103 test cases

● v5.14 released on Aug 29 2021
  ○ 28 test files (x2.8)
  ○ 48 test contributors (x5.3)
  ○ 281 test cases (x2.7)
Test cases per release

X-axis: Dates
Y-axis: Test cases

Dates:
- Nov 24, 2019
- Jan 26, 2020
- Mar 29, 2020
- May 31, 2020
- Aug 02, 2020
- Oct 11, 2020
- Dec 13, 2020
- Feb 14, 2021
- Feb 28, 2021
- Jun 27, 2021
- Aug 29, 2021
- Sep 12, 2021
Some interesting new tests

- DAMON: Data Access MONitor
- SLUB cache error detection
- `time64_to_tm()` and `rtc_time64_to_tm()`
  - Used to validate a more performant implementation
- KFENCE (alongside KCSAN and KASAN)
- thunderbolt
  - Continuing to add several new tests
- ALSA SoC topology
- FAT filesystem (timestamps and filename checksums)
- `lib/rational`
- Many more
Blockers and Problems
Blockers and Problems

- **KUnit is pretty good at testing isolated "library" code**
  - Code which interacts heavily with kernel or hardware state is more difficult
  - This will always be the case: unit testing works best with "functional" code (limited global state, dependencies, etc)
- **Architecture/hardware-dependent tests difficult to work with**
  - `kunit_tool` only supported UML
  - Dependencies only known at runtime
- **Difficult to configure and run only some tests**
  - Need a way of running only tests for a particular subsystem
- **Knowing when to use KUnit**
  - When to use KUnit versus (e.g.) `kseltest`
  - Achieving feature parity / compatibility with other test frameworks
New Features
QEMU Support in kunit_tool

tools/testing/kunit/kunit.py run --arch=arm --cross_compile=arm-linux-gnueabihf-

[10:36:01] Configuring KUnit Kernel ...
Generating .config ...
Populating config with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
[10:36:03] Building KUnit Kernel ...
Building with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
[10:36:10] Starting KUnit Kernel ...
Running tests with:
$ qemu-system-arm -nodefaults -m 1024 -kernel .kunit/arch/arm/boot/zImage -append 'mem=1G console=tty kunit_shutdown=halt console=ttyAMA0 kunit_shutdown=reboot' -no-reboot -nographic -Serial stdio -machine virt

[10:36:11] ============================================================
[10:36:11] ======== [PASSED] snd_soc_tplg_test ========
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_comp
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_ops
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_fw
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_magic
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_obj
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_size
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_payload_size
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_comp
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_card
[10:36:11] Testing complete. 11 tests run. 0 failed. 0 crashed. 0 skipped.
[10:36:11] Elapsed time: 9.989s total, 1.865s configuring, 7.401s building, 0.000s running
QEMU Support in kunit_tool

tools/testing/kunit/kunit.py run --arch=arm --cross_compile=arm-linux-gnueabihf-

[10:36:01] Configuring Kunit Kernel ...
Generating .config ...
Populating config with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- 0=.kunit
Building kunit Kernel ...
Populating config with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- 0=.kunit
Building with:
$ make ARCH=arm --jobs=8 CROSS_COMPILE=arm-linux-gnueabihf- 0=.kunit
Starting kunit Kernel ...
Running tests with:
$ qemu-system-arm -nodefaults -m 1024 -kernel .kunit/arch/arm/boot/zImage -append 'mem=1G console=tty kunit_shutdown=halt console=ttyAMA0 kunit_shutdown=reboot'
-no-reboot -nographic -Serial stdio -machine virt

[10:36:11] ============================================================
[10:36:11] ========= [PASSED] snd_soc_tplg_test ========
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_comp
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_ops
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_fw
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg
[10:36:11] [PASSED] snd_soc_tplg_test_load_emptyTplg_bad_magic
[10:36:11] [PASSED] snd_soc_tplg_test_load_emptyTplg_bad_obj
[10:36:11] [PASSED] snd_soc_tplg_test_load_emptyTplg_bad_size
[10:36:11] [PASSED] snd_soc_tplg_test_load_emptyTplg_bad_payload_size
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_comp
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_card

[10:36:11] Testing complete. 11 tests run. 0 failed. 0 crashed. 0 skipped.
[10:36:11] Elapsed time: 9.989s total, 1.865s configuring, 7.401s building, 0.000s running
QEMU Support in kunit_tool

```bash
> tools/testing/kunit/kunit.py run --arch=arm --cross_compile=arm-linux-gnueabihf-
> [10:36:01] Configuring KUnit Kernel ...
>  Generating .config ...
>  Populating config with:
>  $ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
> [10:36:03] Building KUnit Kernel ...
>  Populating config with:
>  $ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
> Building with:
>  $ make ARCH=arm --jobs=8 CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
> [10:36:10] Starting KUnit Kernel ...
> Running tests with:
>  qemu-system-arm -nodefaults -m 1024 -kernel .kunit/arch/arm/boot/zImage -append
> 'mem=1G console=tty kunit_shutdown=halt console=ttyAMA0 kunit_shutdown=reboot'
> -no-reboot -no graft -serial stdio -machine virt
> [10:36:11] ============================================================
> [10:36:11] ======== [PASSED] snd_soc_tplg_test ========
> [10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_comp
> [10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_ops
> [10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_fw
> [10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg
> [10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_magic
> [10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_ops
> [10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_size
> [10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_payload_size
> [10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg
> [10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_comp
> [10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_card
> [10:36:11] Testing complete. 11 tests run. 0 failed. 0 crashed. 0 skipped.
> [10:36:11] Elapsed time: 9.989s total, 1.865s configuring, 7.401s building, 0.000s
> running
```

Building with:

```
$ make ARCH=arm --jobs=8 CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
```
QEMU Support in kunit_tool

> tools/testing/kunit/kunit.py run --arch=arm --cross_compile=arm-linux-gnueabihf-
[10:36:01] Configuring KUnit Kernel ...
Generating .config ... Populating config with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
[10:36:03] Building KUnit Kernel ...
Building with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
[10:36:10] Starting KUnit Kernel ...
Running tests with:
$ qemu-system-arm -nodefaults -m 1024 -kernel .kunit/arch/arm/boot/zImage -append 'mem=1G console=tty kunit_shutdown=halt console=ttyAMA0 kunit_shutdown=reboot'
-no-reboot -nographic -serial stdio -machine virt

[10:36:10] Starting KUnit Kernel ...

Running tests with:
$ qemu-system-arm -nodefaults -m 1024 -kernel .kunit/arch/arm/boot/zImage -append 'mem=1G console=tty kunit_shutdown=halt console=ttyAMA0 kunit_shutdown=reboot'
-no-reboot -nographic -serial stdio -machine virt

[10:36:11] Testing complete. 11 tests run. 0 failed. 0 crashed. 0 skipped.
Elapsed time: 9.989s total, 1.865s configuring, 7.401s building, 0.000s running

[10:36:11] [PASSED] snd_soc_tplg_test ========
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_comp
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_ops
[10:36:11] [PASSED] snd_soc_tplg_test_load_with_null_fw
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_magic
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_obj
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_size
[10:36:11] [PASSED] snd_soc_tplg_test_load_empty_tplg_bad_payload_size
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_enum
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_comp
[10:36:11] [PASSED] snd_soc_tplg_test_load_pcm_tplg_reload_card

Testing complete. 11 tests run. 0 failed. 0 crashed. 0 skipped.
Elapsed time: 9.999s total, 1.865s configuring, 7.401s building, 0.000s running
QEMU Support in kunit_tool

```bash
> tools/testing/kunit/kunit.py run --arch=arm --cross_compile=arm-linux-gnueabihf-
[10:36:01] Configuring KUnit Kernel ...
Generating .config ...
Populating config with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
[10:36:03] Building KUnit Kernel ...
Populating config with:
$ make ARCH=arm olddefconfig CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
Building with:
$ make ARCH=arm --jobs=8 CROSS_COMPILE=arm-linux-gnueabihf- O=.kunit
[10:36:10] Starting KUnit Kernel ...
Running tests with:
$ qemu-system-arm -nodefaults -m 1024 -kernel .kunit/arch/arm/boot/zImage -append
'mem=1G console=tty console=ttyAMA0 kunit_shutdown=reboot'
```

```
[10:36:11] Testing complete. 11 tests run. 0 failed. 0 crashed. 0 skipped.
Elapsed time: 9.989s total, 1.865s configuring, 7.401s building, 0.000s running
```
QEMU Support in kunit_tool

- We support the following out of the box:
  - i386
  - x86_64
  - arm
  - arm64
  - alpha
  - powerpc
  - riscv
  - s390
  - sparc
QEMU Support in kunit_tool

- We support the following out of the box:
  - i386
  - x86_64
  - arm
  - arm64
  - alpha
  - powerpc
  - riscv
  - s390
  -sparc
  - Don’t see your architecture? No problem.
SKIP Test Support

- Tests often have dependencies or hardware requirements.
  - Usually handled with Kconfig.
  - Sometimes these aren't known until runtime.
- Solution: SKIP the test at runtime if the dependency isn't around
  - e.g. KCSAN needs an SMP system; KCSAN tests are SKIPped if run on a single-core machine.
- (K)TAP specification provides a "skip" result
  - Also allows a "reason" for the test to be skipped.
  - kselftest has supported this for a while.
  - (Along with similar 'xfail' status)
SKIP Test Support

- Use the kunit_skip() or kunit_mark_skipped() macros to skip a test.
  - Take a "reason" parameter.
  - `kunit_skip(test, "foo required but not present");`

- Results will be of the form:
  - `ok 43 - kasan_bitops_tags # SKIP Test requires CONFIG_KASAN_GENERIC=n`
- And show up in kunit_tool as being skipped (in a yellow colour)
Now possible to define specific test configurations:

- Use the --kunitconfig argument to kunit_tool to select a particular .kunitconfig file
  - ./tools/testing/kunit/kunit.py run --kunitconfig=fs/ext4/.kunitconfig
- Pass in a subdirectory to use the .kunitconfig file in that directory
  - ./tools/testing/kunit/kunit.py run --kunitconfig=fs/ext4
- The default config now runs all tests with satisfied dependencies
Test Filtering

- KUnit typically runs all tests which are compiled in
- It's now possible to filter the test suites being run by name
  - Support for filtering individual tests (as described below) is under review at the moment.
- New kunit.filter_glob kernel command-line parameter
  - e.g. kunit.filter_glob=list-kunit-test.*del*
  - Runs only the list delete tests
- Also usable from kunit_tool
  - Just add it as a parameter:
  - ./tools/testing/kunit/kunit.py run 'example*'
Test statistics

Both kselftest and a number of ad-hoc test modules printed summaries of tests passed/failed/skipped/etc.

- kunit_tool supports this, but it's not available for modules
- Test statistics support makes it easier to read a summary of the results without relying on kunit_tool
  - Based on kselftest's similar output lines
  - This was a functionality regression limiting some tests being ported to KUnit
- Prints two lines to handle nested tests properly:
  - # example: pass:1 fail:0 skip:2 total:3
  - # Totals: pass:1 fail:0 skip:2 total:3
Documentation Improvements

● Kernel Testing Guide
  ○ Describes the differences between different kernel testing and validation tools
  ○ Not exhaustive, but a good introduction.
  ○ Feel free to contribute!

● Made improvements to KUnit documentation
  ○ Improvements to kunit_tool documentation
  ○ Tech Writer working on improving KUnit's documentation
Non Technical Growth Efforts
KTAP Standardisation

- Both KUnit and kselftest use variants on TAP (Test Anything Protocol) as a result format.
  - Needed to be extended and modified for kernel use.
    - Nested tests
  - kselftest and KUnit's extensions are incompatible.
  - Tooling like kunit_tool only supported the features KUnit used.
- Aiming to come up with a standard specifically for kernel use.
  - Any parser should be able to handle both kselftest and KUnit results
  - Reworking the kunit_tool parser to support more features
    - Hopefully will support parsing kselftest results as well.
  - Thanks Rae Moar for much of this work, and Tim Bird and Kees Cook for reviews.
Interns/LKMP

- Every year we have had either interns or LKMP mentees
  - 12 interns/LKMP mentees/engres*!
- Hoping to expand testing and KUnit among new developers
- Presented at Linux Foundation Mentorship series
KUnit Hackathons

- FLUSP and LKCAMP organized a KUnit hackathon
  - 4 test conversions
  - 9 student developers
  - 5 student mentors
  - (thanks Vitor Massaru Iha!)

- We are trying to help them find “more interesting” parts to test
  - DRM subsystem (Daniel Vetter) is interested.
  - Hoping to find some others?
  - More helpful and more fun.
Faking

- Testing code with hardware or complex dependencies is difficult.
- Solution: replace dependencies and other state with 'fakes'
  - Instead of talking to real hardware, implement a "fake" device which responds in a certain way.
  - Can have several "fake" devices which trigger different codepaths
  - Equally, global functions can be replaced with "fake" versions.
  - e.g. A kalloc() function which always errors, or a fake read() from a file.
- Some notes on how to achieve this: [https://kunit.dev/mocking.html](https://kunit.dev/mocking.html)
  - Not planning to implement a full "mocking" framework at the moment.
- Hoping to try this out with drivers and filesystems.
Subsystem Testing

- Now we have a decent amount of tests spread across the entire kernel
- Some “layers” of the kernel don’t seem to attract tests
- Idea: Whiteglove test a sizable subsystem in the kernel
  - Once people see how an entire subsystem is tested, serve as model for other parts
  - Lots of people look for “similar code and then base new code on it”
Discussion
Discussion

● What shouldn’t we be doing?
● What should we be doing that we aren’t?
● How can we improve what we are already doing?
  ○ Test Porting
  ○ Removing blockers
  ○ Interns/LKMP
  ○ Faking
  ○ KTAP Standardization
  ○ KUnit Hackathon
  ○ Subsystem Testing
Backup Slides
Growth: Porting Tests

- There are a number of test modules which would work better as KUnit test suites.
  - Typically written without a framework, sometimes with kselftest
  - Usually manually loaded as a module, and results printed out
  - Results are often checked manually
  - Don't use (K)TAP

- Some of these are candidates for being ported to KUnit
  - Can be built-in, as well as run as modules
  - Automatically run both by kunit_tool and other CI (e.g. LKFT)
  - Return a pass/fail result.

- Tests ported include:
  - test_list_sort, test_sort, KASAN, mptcp
QEMU Support in kunit_tool

- Make some config called: `myarch.py`:

```python
from ..qemu_config import QemuArchParams

QEMU_ARCH = QemuArchParams(linux_arch='arm',
    kconfig='''
CONFIG_ARCH_VIRT=y
CONFIG_SERIAL_AMBA_PL010=y
CONFIG_SERIAL_AMBA_PL010_CONSOLE=y
CONFIG_SERIAL_AMBA_PL011=y
CONFIG_SERIAL_AMBA_PL011_CONSOLE=y''',
    qemu_arch='arm',
    kernel_path='arch/arm/boot/zImage',
    kernel_command_line='console=ttyAMA0',
    extra_qemu_params=['-machine virt']
)```
QEMU Support in kunit_tool

- Run config like this:

```bash
tools/testing/kunit/kunit.py run
   --cross_compile=../my-arch-cross-
   --qemu_config=./myarch.py
```
Hermetic Testing

- Ideally tests should restore any global state they change (i.e., have no side-effects)
- This is not always the case, particularly when debugging
- Hermetic testing is a new (in-progress) feature to run tests independently
  - Split execution of tests across several kernel invocations
  - Collate the results

Future work:
- Randomise test ordering
Growth Plans:
Present and Future
Blockers and Problems