The next generation QEMU functional testing framework

Daniel P. Berrangé <berrange@redhat.com>
Thomas Huth <thuth@redhat.com>

a comic-style picture of a friendly, smiling emu that is standing like a captain in a bridge of a space ship similar to the old star trek series.

a friendly emu that holds a test tube with some chemicals that just exploded and covered the emu in coal dust

Legal

a cartoon emu as a solicitor in court

- Disclaimer: Opinions are our own and not necessarily the views of our employer
- The somewhat absurd emu pictures have been generated with AI (Gemini; prompts noted inline). All other content in this presentation is the work of humans.

comic-style picture of a friendly and smiling emu that flies through the air using rockets

A little bit of history

- QEMU had many test suites
  (unit tests, “qtests”, “iotests”, TCG-tests, …)
- Missing end-to-end functional testing of VMs
  with real payloads (kernel, etc.)
- Needed to write tests more easily. e.g. python + helpers for assets & caching
- Maintainers from the Avocado project offered help in 2018
- The Avocado-based functional test suite was introduced (initially called “acceptance” tests, later renamed to “avocado” tests)

a comic-style picture of a friendly emu that is standing in a huge amount of avocados

The issues with Avocado for QEMU

- Avocado is a fairly complex system
- The average QEMU developer is a C hacker, probably not a Python expert
- If something did not work, QEMU developers rarely understood how to fix it
- Little proactive help from the Avocado developers after the initial code drop
  - ➢ But no other QEMU maintainers felt responsible for the subsystem
- Attempts to upgrade Avocado to a newer release failed for various reasons
  - ➢ Stuck with outdated & unmaintained Avocado v88.1 (from 2021)
- Last straw in 2024: Avocado v88.1 not compatible with Python 3.12
  - ➢ A solution had to be found ASAP

(ok, this picture was rather created with gimp from the previous one, since the AI was not able to come up with a nice picture of an emu putting its head into the ground)

Re-considering the testing situation

- Avocado was performing two jobs, providing
  - ➢ (a) test runner / harness
  - ➢ (b) test creation infrastructure APIs
- 2018: A zoo of historically grown test harness for various test subsystems
- 2024: Meson provides the parallel test runner for most QEMU frameworks
  - ➢ Avocado tests were an oddball here (v88 was single-threaded)
  - ➢ Avocado test harness results difficult to interpret & complicated debugging failures
- 2022: “Introduce new acpi/smbios python tests using biosbits”
  - ➢ Tried a new stand-alone Python-based functional test separate from Avocado
- Could Avocado tests be migrated to standalone python tests run by Meson ?

comic-style image of a friendly emu that scratches its head and looks like it is thinking hard

Example of an Avocado test (slightly modified)

```
from avocado_qemu import QemuSystemTest, wait_for_console_pattern
from avocado.utils import archive

class CanonA1100Machine(QemuSystemTest):

    timeout = 90

    @skipUnless(os.getenv(QEMU_TEST_FLAKY_TESTS), 'Test might be unstable')
    def test_arm_canona1100(self):
        """
        :avocado: tags=arch:arm
        :avocado: tags=machine:canon-a1100
        """
        tar_url = ('https://qemu-advcal.gitlab.io/…/day18.tar.xz')
        tar_hash = '068b5fc4242b29381acee94713509f8a876e9db6'
        file_path = self.fetch_asset(tar_url, asset_hash=tar_hash)
        archive.extract(file_path, self.workdir)
        self.vm.add_args('-bios', self.workdir + '/day18/barebox.canon-a1100.bin')
        self.vm.set_console()
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')
```

a cartoon emu viewing the matrix

Example of an Avocado test (slightly modified)

```
from avocado_qemu import QemuSystemTest, wait_for_console_pattern
from avocado.utils import archive

class CanonA1100Machine(QemuSystemTest):

    timeout = 90

    @skipUnless(os.getenv(QEMU_TEST_FLAKY_TESTS), 'Test might be unstable')
    def test_arm_canona1100(self):
        """
        :avocado: tags=arch:arm
        :avocado: tags=machine:canon-a1100
        """
        tar_url = ('https://qemu-advcal.gitlab.io/…/day18.tar.xz')
        tar_hash = '068b5fc4242b29381acee94713509f8a876e9db6'
        file_path = self.fetch_asset(tar_url, asset_hash=tar_hash)
        archive.extract(file_path, self.workdir)
        self.vm.add_args('-bios', self.workdir + '/day18/barebox.canon-a1100.bin')
        self.vm.set_console()
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')
```

Basic test class

a cartoon emu viewing the matrix

Example of an Avocado test (slightly modified)

```
from avocado_qemu import QemuSystemTest, wait_for_console_pattern
from avocado.utils import archive

class CanonA1100Machine(QemuSystemTest):

    timeout = 90

    @skipUnless(os.getenv(QEMU_TEST_FLAKY_TESTS), 'Test might be unstable')
    def test_arm_canona1100(self):
        """
        :avocado: tags=arch:arm
        :avocado: tags=machine:canon-a1100
        """
        tar_url = ('https://qemu-advcal.gitlab.io/…/day18.tar.xz')
        tar_hash = '068b5fc4242b29381acee94713509f8a876e9db6'
        file_path = self.fetch_asset(tar_url, asset_hash=tar_hash)
        archive.extract(file_path, self.workdir)
        self.vm.add_args('-bios', self.workdir + '/day18/barebox.canon-a1100.bin')
        self.vm.set_console()
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')
```

Timeout setting for the test runner

Subtest method

a cartoon emu viewing the matrix

Example of an Avocado test (slightly modified)

```
from avocado_qemu import QemuSystemTest, wait_for_console_pattern
from avocado.utils import archive

class CanonA1100Machine(QemuSystemTest):

    timeout = 90

    @skipUnless(os.getenv(QEMU_TEST_FLAKY_TESTS), 'Test might be unstable')
    def test_arm_canona1100(self):
        """
        :avocado: tags=arch:arm
        :avocado: tags=machine:canon-a1100
        """
        tar_url = ('https://qemu-advcal.gitlab.io/…/day18.tar.xz')
        tar_hash = '068b5fc4242b29381acee94713509f8a876e9db6'
        file_path = self.fetch_asset(tar_url, asset_hash=tar_hash)
        archive.extract(file_path, self.workdir)
        self.vm.add_args('-bios', self.workdir + '/day18/barebox.canon-a1100.bin')
        self.vm.set_console()
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')
```

Decorator

a cartoon emu viewing the matrix

Example of an Avocado test (slightly modified)

```
from avocado_qemu import QemuSystemTest, wait_for_console_pattern
from avocado.utils import archive

class CanonA1100Machine(QemuSystemTest):

    timeout = 90

    @skipUnless(os.getenv(QEMU_TEST_FLAKY_TESTS), 'Test might be unstable')
    def test_arm_canona1100(self):
        """
        :avocado: tags=arch:arm
        :avocado: tags=machine:canon-a1100
        """
        tar_url = ('https://qemu-advcal.gitlab.io/…/day18.tar.xz')
        tar_hash = '068b5fc4242b29381acee94713509f8a876e9db6'
        file_path = self.fetch_asset(tar_url, asset_hash=tar_hash)
        archive.extract(file_path, self.workdir)
        self.vm.add_args('-bios', self.workdir + '/day18/barebox.canon-a1100.bin')
        self.vm.set_console()
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')
```

Tags

a cartoon emu viewing the matrix

Example of an Avocado test (slightly modified)

```
from avocado_qemu import QemuSystemTest, wait_for_console_pattern
from avocado.utils import archive

class CanonA1100Machine(QemuSystemTest):

    timeout = 90

    @skipUnless(os.getenv(QEMU_TEST_FLAKY_TESTS), 'Test might be unstable')
    def test_arm_canona1100(self):
        """
        :avocado: tags=arch:arm
        :avocado: tags=machine:canon-a1100
        """
        tar_url = ('https://qemu-advcal.gitlab.io/…/day18.tar.xz')
        tar_hash = '068b5fc4242b29381acee94713509f8a876e9db6'
        file_path = self.fetch_asset(tar_url, asset_hash=tar_hash)
        archive.extract(file_path, self.workdir)
        self.vm.add_args('-bios', self.workdir + '/day18/barebox.canon-a1100.bin')
        self.vm.set_console()
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')
```

Asset handling

a cartoon emu viewing the matrix

Requirements for a new test framework

What?	Replacement
Test runner	Meson test runner (+ pycotap)
Test discovery	meson.build + unittest class (+ pycotap)
Basic test class	Recycle glue code (based on unittest class)
Decorators	Python unittest class (+ adding our own)
Asset download, caching & extracting	Replace with our own implementation
Logging	Python logging + custom setup
Tags (for selecting subsets of tests)	No replacement (possibly decorators?)

a cartoon emu writing a list of parts

Test files discovery and timeouts

- Avocado
  - ➢ Scanned all *.py files on invocation for available tests
  - ➢ A “timeout” variable within the test class for test timeout
- New functional tests
  - ➢ .py files and timeouts listed in tests/functional/*/meson.build files
- Slightly more work to add a test…
  - ➢ … but it’s the common pattern with meson to explicitly list files
- Meson allows to specify a priority
  - ➢ Meson can run all subsystem tests in parallel (qtests, iotests, unit, …)
  - ➢ High priority for tests with large timeout makes long tests start first

comic-style picture of an friendly, smiling emu that looks through a looking glass

How to handle subtests with meson

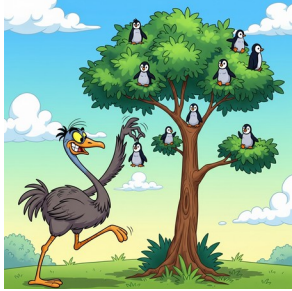
- Python ‘unittest’ class can be used for discovering & running subtest methods within a file. For running test standalone, simply add:

```
if __name__ == '__main__':
    QemuSystemTest.main()
```

- By default, the meson test runner treats one file as one big test
- For showing progress on subtest level: TAP – Test anything protocol
- Tried a bunch of implementations (‘tappy’, …), but none worked very well with the meson test runner (issues with stderr)
- Finally found ‘Pycotap’ that does the job and is very small!
  - ➢ Shipped as a wheel with QEMU now

a cartoon emu surrounded by lots of baby emus

Handle large test assets



- Most tests use assets (kernel, initrd, rootfs, firmware, etc.)
- Downloading assets is potentially quite slow
  - ➢ could cause test timeouts if done on demand
- Avocado had a local cache of assets that must be retained
- Asset class instances declared as class level variables
- Launching test with ‘QEMU_TEST_PRECACHE=1’ pre-caches asset files
- If any assets fail to download, CI jobs will skip the affected tests
  - ➢ …except HTTP 404 codes, which are likely non-transient errors
- Added locking/waiting logic for downloading assets in parallel

a cartoon emu picking penguins from a tree

File management

- Tests need to reference & create files in various locations
- Helper APIs provided on QemuBaseTest to standardize locations
  - self.socket_dir() => location for UNIX sockets
  - self.data_file(...) => source file relative to tests/functional dir
  - self.build_file(...) => build system created file relative to root of build dir
  - self.scratch_file(...) => any file created by test case, auto-deleted on exit
  - self.log_file(...) => any file recording log messages, uploaded as asset in CI jobs
  - self.plugin_file(...) => any TCG plugin relative to tests/tcg/plugins/
- APIs construct paths from components
  - qualified_path = self.scratch_file(“foo”, “bar”, “wizz”)
  - Not
    - qualified_path = self.scratch_file(“foo/bar/wizz”)

a cartoon emu in a library covered in falling books

Archive management

a cartoon emu wheeling a crate away from the viewer in a warehouse with crates as far as the eye can see

- Many assets are compressed or archives
- Lots of different python APIs for each format
- Helper APIs provided on QemuBaseTest to give easy access
- Archive extraction (tar, zip, cpio, deb)
  - self.archive_extract(self.ASSET_....., ...) => unpacks to ‘scratch_file’ location
- File decompression (gz, xz, zstd)
  - self.uncompress(self.ASSET_..., ...) => uncompressed to ‘scratch_file’ location
- Formats guessed from archive URL path file extension

Decorators

- Typical python testing practice is to use decorators to control execution
- Functional test system provides standard decorators
  - skipIfMissingCommands => check ‘binary’ in $PATH
  - skipIfOperatingSystem => exclude listed host OS
  - skipIfNotMachine => require a specific VM machine type
  - skipFlakyTest => don’t run non-deterministic tests (GitLab issue URL required)
  - skipUntrustedTest => don’t run potentially dangerous tests
  - skipBigDataTest => don’t run tests which create huge files (> ~500 MB)
  - skipSlowTest => don’t run tests which are excessively slow (many minutes)
  - skipIfMissingImports => check ‘module’ in $PYTHONPATH
  - skipLockedMemoryTest => require permission to lock RAM

comic-style picture of a friendly emu that decorates a wall with a picture of a female emu.

Example of a new functional test

```
from qemu_test import QemuSystemTest, Asset, skipFlakyTest
from qemu_test import wait_for_console_pattern

class CanonA1100Machine(QemuSystemTest):

    ASSET_BIOS = Asset('https://qemu-advcal.gitlab.io/…/day18.tar.xz',
        '28e71874ce985be66b7fd1345ed88cb2523b982f899c8d2900d6353054a1be49')

    @skipFlakyTest('https://gitlab.com/qemu-project/qemu/-/issues/xyz')
    def test_arm_canona1100(self):
        self.set_machine('canon-a1100')

        bios = self.archive_extract(self.ASSET_BIOS,
                                    member="day18/barebox.canon-a1100.bin")
        self.vm.set_console()
        self.vm.add_args('-bios', bios)
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')

if __name__ == '__main__':
    QemuSystemTest.main()
```

a cartoon emu inside the matrix dodging flying penguins

Example of a new functional test

```
from qemu_test import QemuSystemTest, Asset, skipFlakyTest
from qemu_test import wait_for_console_pattern

class CanonA1100Machine(QemuSystemTest):

    ASSET_BIOS = Asset('https://qemu-advcal.gitlab.io/…/day18.tar.xz',
        '28e71874ce985be66b7fd1345ed88cb2523b982f899c8d2900d6353054a1be49')

    @skipFlakyTest('https://gitlab.com/qemu-project/qemu/-/issues/xyz')
    def test_arm_canona1100(self):
        self.set_machine('canon-a1100')

        bios = self.archive_extract(self.ASSET_BIOS,
                                    member="day18/barebox.canon-a1100.bin")
        self.vm.set_console()
        self.vm.add_args('-bios', bios)
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')

if __name__ == '__main__':
    QemuSystemTest.main()
```

Replacement for the tags

a cartoon emu inside the matrix dodging flying penguins

Example of a new functional test

```
from qemu_test import QemuSystemTest, Asset, skipFlakyTest
from qemu_test import wait_for_console_pattern

class CanonA1100Machine(QemuSystemTest):

    ASSET_BIOS = Asset('https://qemu-advcal.gitlab.io/…/day18.tar.xz',
        '28e71874ce985be66b7fd1345ed88cb2523b982f899c8d2900d6353054a1be49')

    @skipFlakyTest('https://gitlab.com/qemu-project/qemu/-/issues/xyz')
    def test_arm_canona1100(self):
        self.set_machine('canon-a1100')

        bios = self.archive_extract(self.ASSET_BIOS,
                                    member="day18/barebox.canon-a1100.bin")
        self.vm.set_console()
        self.vm.add_args('-bios', bios)
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')

if __name__ == '__main__':
    QemuSystemTest.main()
```

New asset handling

a cartoon emu inside the matrix dodging flying penguins

Example of a new functional test

```
from qemu_test import QemuSystemTest, Asset, skipFlakyTest
from qemu_test import wait_for_console_pattern

class CanonA1100Machine(QemuSystemTest):

    ASSET_BIOS = Asset('https://qemu-advcal.gitlab.io/…/day18.tar.xz',
        '28e71874ce985be66b7fd1345ed88cb2523b982f899c8d2900d6353054a1be49')

    @skipFlakyTest('https://gitlab.com/qemu-project/qemu/-/issues/xyz')
    def test_arm_canona1100(self):
        self.set_machine('canon-a1100')

        bios = self.archive_extract(self.ASSET_BIOS,
                                    member="day18/barebox.canon-a1100.bin")
        self.vm.set_console()
        self.vm.add_args('-bios', bios)
        self.vm.launch()
        wait_for_console_pattern(self, 'running /env/bin/init')

if __name__ == '__main__':
    QemuSystemTest.main()
```

New decorator

For running standalone

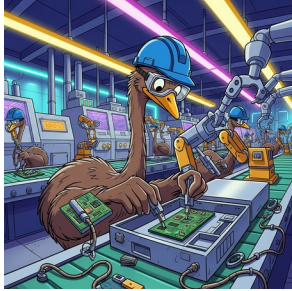
a cartoon emu inside the matrix dodging flying penguins

Troubleshooting

- Run tests standalone outside meson for easier debugging (strace)
  - $ export PYTHONPATH=../python:../tests/functional
  - $ export QEMU_TEST_QEMU_BINARY=$PWD/qemu-system-x86_64
  - $ pyvenv/bin/python3 ../tests/functional/test_file.py
- Getting information out of CI infra is always a challenge
- Logging is critical to understanding failures
  - $BUILD/tests/functional/x86_64/$TEST_FILE.$TEST_CLASS.$TEST_METHOD/
  - base.log ⇒ python ‘logging’ output from test class (includes CLI args of QEMU)
  - console.log ⇒ serial console output from QEMU guest
  - default.log ⇒ stdout/err from spawned QEMU
- All log files are uploaded as artifacts in GitLab CI jobs

a cartoon emu pulling apart computers in a server room tangled in cables

Integration into the test suites



- “make check-functional” (or “make check-functional-ppc” etc.)
- Don’t want to do this by default during a normal “make check”
  (a good internet connection is required for downloading the assets!)
- But some tests don’t need assets, i.e. should be run by default
  - ➢ Need a way to distinguish them
- Existing test suites already have speed classes:
  - ➢ *quick*, *slow* and *thorough*
- Only *quick* tests are run by default, i.e. add tests without assets here
- Add functional tests with assets to the *thorough* category
  - ➢ make -j$(nproc) check SPEED=thorough

a cartoon emu working on an assembly line in a computer factory

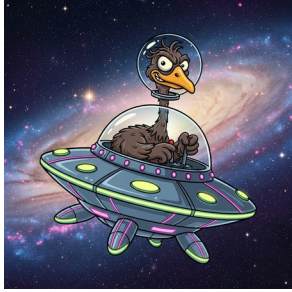
Demo

thuth@thuth-p1g4:~/tmp/qemu-build

```
$ make -j$(nproc) check-functional
[0/1] Running external command precache-functional (wrapped by meson to set env)
make[1]: Entering directory '/home/thuth/tmp/qemu-build'
[1/36] Generating qemu-version.h with a custom command (wrapped by meson to capture output)
/home/thuth/tmp/qemu-build/pyvenv/bin/meson test  --no-rebuild -t 1 --setup thorough  --num-processes 16 --print-errorlogs  --suite func  --suite func-quic
k  --suite func-thorough
  1/267 qemu:func-thorough+func-ppc64-thorough+thorough / func-ppc64-hv                    SKIP             0.13s   0 subtests passed
  2/267 qemu:func-thorough+func-aarch64-thorough+thorough / func-aarch64-smmu              SKIP             9.03s   0 subtests passed
  3/267 qemu:func-thorough+func-aarch64-thorough+thorough / func-aarch64-raspi4            OK              39.32s   2 subtests passed
  4/267 qemu:func-thorough+func-arm-thorough+thorough / func-arm-aspeed_gb200nvl_bmc       OK              41.92s   1 subtests passed
  5/267 qemu:func-thorough+func-mipsel-thorough+thorough / func-mipsel-replay              SKIP             0.32s   0 subtests passed
  6/267 qemu:func-thorough+func-arm-thorough+thorough / func-arm-aspeed_catalina           OK             101.70s   1 subtests passed
  7/267 qemu:func-thorough+func-aarch64-thorough+thorough / func-aarch64-device_passthrough OK             104.67s   1 subtests passed
  8/267 qemu:func-thorough+func-arm-thorough+thorough / func-arm-orangepi                  OK             110.46s   3 subtests passed
  9/267 qemu:func-thorough+func-arm-thorough+thorough / func-arm-aspeed_bletchley          OK             113.38s   1 subtests passed
 10/267 qemu:func-thorough+func-arm-thorough+thorough / func-arm-bpim2u                    OK             118.96s   3 subtests passed
 11/267 qemu:func-thorough+func-aarch64-thorough+thorough / func-aarch64-virt_gpu          OK             136.45s   2 subtests passed
 12/267 qemu:func-thorough+func-x86_64-thorough+thorough / func-x86_64-replay              OK              60.75s   1 subtests passed
[13-28/267] 🌓 qemu:func-thorough+func-arm-thorough+thorough / func-arm-aspeed_ast2500                  174/720.0s   1 subtests passed
```

a cartoon emu presenting a new computer product on stage for the "Emu Tools" company launch event

Future plans



- Evicting obsolete assets from the download cache
- Improve error handling, e.g. if QEMU crashes
- Enforce mypy, flake8, pylint, etc; format with ‘black’
- Add more test for uncovered areas
  - ➢ If you know how to test one of the missing machines, please help:

https://wiki.qemu.org/Testing/Machines

a cartoon emu piloting a flying saucer across the galaxy

Any questions ?

(Or find us in the hallway later)

a cartoon emu in a room of broken computers in the style of the "this is fine" meme