Linux Kernel Selftests

The kernel contains a set of "self tests" under the tools/testing/selftests/ directory. These are intended to be small tests to exercise individual code paths in the kernel. Tests are intended to be run after building, installing and booting a kernel.

Kselftest from mainline can be run on older stable kernels. Running tests from mainline offers the best coverage. Several test rings run mainline kselftest suite on stable releases. The reason is that when a new test gets added to test existing code to regression test a bug, we should be able to run that test on an older kernel. Hence, it is important to keep code that can still test an older kernel and make sure it skips the test gracefully on newer releases.

You can find additional information on Kselftest framework, how to write new tests using the framework on Kselftest wiki:

https://kselftest.wiki.kernel.org/

On some systems, hot-plug tests could hang forever waiting for cpu and memory to be ready to be offlined. A special hot-plug target is created to run the full range of hot-plug tests. In default mode, hot-plug tests run in safe mode with a limited scope. In limited mode, cpu-hotplug test is run on a single cpu as opposed to all hotplug capable cpus, and memory hotplug test is run on 2% of hotplug capable memory instead of 10%.

kselftest runs as a userspace process. Tests that can be written/run in userspace may wish to use the Test Harness. Tests that need to be run in kernel space may wish to use a Test Module.

Running the selftests (hotplug tests are run in limited mode)

To build the tests:

```
$ make -C tools/testing/selftests
```

To run the tests:

```
$ make -C tools/testing/selftests run_tests
```

To build and run the tests with a single command, use:

```
$ make kselftest
```

Note that some tests will require root privileges.

Kselftest supports saving output files in a separate directory and then running tests. To locate output files in a separate directory two syntaxes are supported. In both cases the working directory must be the root of the kernel src. This is applicable to "Running a subset of selftests" section below.

To build, save output files in a separate directory with O=

```
$ make O=/tmp/kselftest kselftest
```

To build, save output files in a separate directory with KBUILD_OUTPUT

```
$ export KBUILD_OUTPUT=/tmp/kselftest; make kselftest
```

The O= assignment takes precedence over the KBUILD_OUTPUT environment variable.

The above commands by default run the tests and print full pass/fail report. Kselftest supports "summary" option to make it easier to understand the test results. Please find the detailed individual test results for each test in /tmp/testname file(s) when summary option is specified. This is applicable to "Running a subset of selftests" section below.

To run kselftest with summary option enabled

```
$ make summary=1 kselftest
```

Running a subset of selftests

You can use the "TARGETS" variable on the make command line to specify single test to run, or a list of tests to run.

To run only tests targeted for a single subsystem:

```
$ make -C tools/testing/selftests TARGETS=ptrace run_tests
```

You can specify multiple tests to build and run:

```
$  make TARGETS="size timers" kselftest
```

To build, save output files in a separate directory with O=

```
$ make O=/tmp/kselftest TARGETS="size timers" kselftest
```

To build, save output files in a separate directory with KBUILD_OUTPUT

```
$ export KBUILD_OUTPUT=/tmp/kselftest; make TARGETS="size timers" kselftest
```

Additionally you can use the "SKIP_TARGETS" variable on the make command line to specify one or more targets to exclude from the TARGETS list.

To run all tests but a single subsystem:

```
$ make -C tools/testing/selftests SKIP_TARGETS=ptrace run_tests
```

You can specify multiple tests to skip:

```
$  make SKIP_TARGETS="size timers" kselftest
```

You can also specify a restricted list of tests to run together with a dedicated skiplist:

```
$  make TARGETS="bpf breakpoints size timers" SKIP_TARGETS=bpf kselftest
```

See the top-level tools/testing/selftests/Makefile for the list of all possible targets.

Running the full range hotplug selftests

To build the hotplug tests:

```
$ make -C tools/testing/selftests hotplug
```

To run the hotplug tests:

```
$ make -C tools/testing/selftests run_hotplug
```

Note that some tests will require root privileges.

Install selftests

You can use the "install" target of "make" (which calls the *kselftest_install.sh* tool) to install selftests in the default location (*tools/testing/selftests/kselftest_install*), or in a user specified location via the *INSTALL_PATH* "make" variable.

To install selftests in default location:

```
$ make -C tools/testing/selftests install
```

To install selftests in a user specified location:

```
$ make -C tools/testing/selftests install INSTALL_PATH=/some/other/path
```

Running installed selftests

Found in the install directory, as well as in the Kselftest tarball, is a script named *run_kselftest.sh* to run the tests.

You can simply do the following to run the installed Kselftests. Please note some tests will require root privileges:

```
$ cd kselftest_install
$ ./run_kselftest.sh
```

To see the list of available tests, the *-l* option can be used:

```
$ ./run_kselftest.sh -l
```

The *-c* option can be used to run all the tests from a test collection, or the *-t* option for specific single tests. Either can be used multiple times:

```
$ ./run_kselftest.sh -c bpf -c seccomp -t timers:posix_timers -t timer:nanosleep
```

For other features see the script usage output, seen with the *-h* option.

Packaging selftests

In some cases packaging is desired, such as when tests need to run on a different system. To package selftests, run:

```
$ make -C tools/testing/selftests gen_tar
```

This generates a tarball in the *INSTALL_PATH/kselftest-packages* directory. By default, *.gz* format is used. The tar compression format can be overridden by specifying a *FORMAT* make variable. Any value recognized by tar's auto-compress option is supported, such as:

```
$ make -C tools/testing/selftests gen_tar FORMAT=.xz
```

make gen_tar invokes *make install* so you can use it to package a subset of tests by using variables specified in Running a subset of selftests section:

```
$ make -C tools/testing/selftests gen_tar TARGETS="bpf" FORMAT=.xz
```

Contributing new tests

In general, the rules for selftests are

- Do as much as you can if you're not root;
- Don't take too long;
- Don't break the build on any architecture, and
- Don't cause the top-level "make run_tests" to fail if your feature is unconfigured.

Contributing new tests (details)

- Use TEST_GEN_XXX if such binaries or files are generated during compiling.

  TEST_PROGS, TEST_GEN_PROGS mean it is the executable tested by default.

  TEST_CUSTOM_PROGS should be used by tests that require custom build rules and prevent common build rule use.

  TEST_PROGS are for test shell scripts. Please ensure shell script has its exec bit set. Otherwise, lib.mk run_tests will generate a warning.

  TEST_CUSTOM_PROGS and TEST_PROGS will be run by common run_tests.

  TEST_PROGS_EXTENDED, TEST_GEN_PROGS_EXTENDED mean it is the executable which is not tested by default. TEST_FILES, TEST_GEN_FILES mean it is the file which is used by test.
- First use the headers inside the kernel source and/or git repo, and then the system headers. Headers for the kernel release as opposed to headers installed by the distro on the system should be the primary focus to be able to find regressions.
- If a test needs specific kernel config options enabled, add a config file in the test directory to enable them.

  e.g: tools/testing/selftests/android/config

Test Module

Kselftest tests the kernel from userspace. Sometimes things need testing from within the kernel, one method of doing this is to create a test module. We can tie the module into the kselftest framework by using a shell script test runner. `kselftest/module.sh` is designed to facilitate this process. There is also a header file provided to assist writing kernel modules that are for use with kselftest:

- `tools/testing/selftests/kselftest_module.h`
- `tools/testing/selftests/kselftest/module.sh`

How to use

Here we show the typical steps to create a test module and tie it into kselftest. We use kselftests for lib/ as an example.

1. Create the test module
2. Create the test script that will run (load/unload) the module e.g. `tools/testing/selftests/lib/printf.sh`
3. Add line to config file e.g. `tools/testing/selftests/lib/config`
4. Add test script to makefile e.g. `tools/testing/selftests/lib/Makefile`
5. Verify it works:

```
# Assumes you have booted a fresh build of this kernel tree
cd /path/to/linux/tree
make kselftest-merge
make modules
sudo make modules_install
make TARGETS=lib kselftest
```

Example Module

A bare bones test module might look like this:

```
// SPDX-License-Identifier: GPL-2.0+

#define pr_fmt(fmt) KBUILD_MODNAME ": " fmt

#include "../tools/testing/selftests/kselftest/module.h"

KSTM_MODULE_GLOBALS();

/*
```

```
 * Kernel module for testing the foobinator
 */

static int __init test_function()
{
        ...
}

static void __init selftest(void)
{
        KSTM_CHECK_ZERO(do_test_case("", 0));
}

KSTM_MODULE_LOADERS(test_foo);
MODULE_AUTHOR("John Developer <jd@fooman.org>");
MODULE_LICENSE("GPL");
```

Example test script

```
#!/bin/bash
# SPDX-License-Identifier: GPL-2.0+
$(dirname $0)/../kselftest/module.sh "foo" test_foo
```

Test Harness

The kselftest_harness.h file contains useful helpers to build tests. The test harness is for userspace testing, for kernel space testing see Test Module above.

The tests from tools/testing/selftests/seccomp/seccomp_bpf.c can be used as example.

Example

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)kselftest.rst, line 335)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: tools/testing/selftests/kselftest_harness.h
    :doc: example
```

Helpers

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)kselftest.rst, line 342)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: tools/testing/selftests/kselftest_harness.h
    :functions: TH_LOG TEST TEST_SIGNAL FIXTURE FIXTURE_DATA FIXTURE_SETUP
                FIXTURE_TEARDOWN TEST_F TEST_HARNESS_MAIN FIXTURE_VARIANT
                FIXTURE_VARIANT_ADD
```

Operators

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)kselftest.rst, line 350)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: tools/testing/selftests/kselftest_harness.h
    :doc: operators
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\dev-tools\(linux-master)(Documentation)(dev-tools)kselftest.rst, line 353)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: tools/testing/selftests/kselftest_harness.h
    :functions: ASSERT_EQ ASSERT_NE ASSERT_LT ASSERT_LE ASSERT_GT ASSERT_GE
                ASSERT_NULL ASSERT_TRUE ASSERT_NULL ASSERT_TRUE ASSERT_FALSE
                ASSERT_STREQ ASSERT_STRNE EXPECT_EQ EXPECT_NE EXPECT_LT
                EXPECT_LE EXPECT_GT EXPECT_GE EXPECT_NULL EXPECT_TRUE
                EXPECT_FALSE EXPECT_STREQ EXPECT_STRNE
```