testing

Abdul Rahim

23 July 2024

Developer testing, integration testing, regression, and stress testing have different individual goals. However, from 1,000 feet up, the end goal is the same: to ensure the software continues to work as it did before adding a new body of code, and that new features work as designed. It is time to look at testing.

Automated build bots and continous integration (CI) test rings run tests on various kernel repositories. The 0-day build bot can pull in patches and run build tests on several configurations and architectures. It is helpful in finding compile errors and warnings that might show up on other architectures that developers might not have access to.

Continuous Integration (CI) rings are test farms that host several platforms and run boot tests and Kernel Selftests on stable, linux-next, and mainline trees, these tests run in real hardware as well as qrmu environments. covering a wide range of archectures and configurations.

Applying patches

- Each Linux patch is a self-contained change to the code that stands on its own, unless explicitly made part of a patch series. New patches are applied as follows:

```
1  git apply --index file.patch
```

- If your use `patch` command; The problem with this is that: git reset –hard command will not remove the newly created files and a subsequent `git pull` will fail; hence it is not recommended.

basic testing

- check `dmesg`
- Is networking (wifi or wired) functional?
- Does ssh work?
- Run rsync of a large file over ssh
- Run git clone and git pull
- Start a web browser
- Read email
- Download files: ftp, wget, etc.
- Play audio/video files
- Connect new USB devices - mouse, USB stick, etc.

dmesg

Checking for regressions in `dmesg` is a good way to identify problems, if any, introduced by the new code. As a general rule, **there should be no new crit, alert, and emerg level messages in dmesg**. There should be no **new err level messages**. Pay close attention to any new warn level messages as well. Please note that **new warn messages are not as bad**. At times, new code adds new warning messages which are just warnings.

```
dmesg -t -l emerg
dmesg -t -l crit
dmesg -t -l alert
dmesg -t -l err
dmesg -t -l warn
dmesg -t -k
dmesg -t
```

Are there any stack traces resulting from WARN_ON in the dmesg? These are serious problems that require further investigation.

stress test

Performance problems are hard to debug. The first step is to detect them. Running several compiles in parallel is a good overall stress test that could be used as a performance regression test and overall kernel regression test, as it exercises various kernel modules like memory, filesystems, dma, and drivers.

```
1  time make all
```

debug options for testing your code

As you are making changes to drivers and other areas in the kernel, there are a few things to watch out for. There are several configuration options to test for locking imbalances and deadlocks. It is important to remember to enable the Lock Debugging and CONFIG_KASAN for memory leak detection. We do not intend to cover debugging in depth in this chapter, but we want you to start thinking about debugging and configuration options that facilitate debugging.

Enabling the following configuration option is recommended for testing your changes to the kernel:

- CONFIG_KASAN
- CONFIG_KMSAN
- CONFIG_UBSAN
- CONFIG_LOCKDEP
- CONFIG_PROVE_LOCKING
- CONFIG_LOCKUP_DETECTOR

Running `git grep -r DEBUG | grep Kconfig` can find all supported debug configuration options.

Checkout *LF mentorship series*

You can find resources on static and dynamic program analysis, fuzz testing, and Kernel testing using KUnit and Kselftest.