Linux Kernel Functional Testing

Rémi Duraffort, Linaro Ltd
remi.duraffort@linaro.org
Who am I?

- Rémi Duraffort
- Principal Tech Lead at Linaro
- OSS developer since 2007
  - VLC media player
  - v8 js engine
  - PRoot/CARE
  - LAVA, KissCache, lavacli, meta-lava, DummySYS, lavafed, ...
  - tuxrun, tuxsuite cli, ...
- LAVA Architect for 8 years
LKFT
Linux Kernel Functional Testing
What is LKFT?

“Improve the Linux kernel quality on the Arm architecture by performing regression testing and reporting on selected Linux kernel branches and the Android Common Kernel (ACK) in real time.”

- Lead by Linaro
- Automated system to build and test a set of linux kernel trees
  - LTS trees
  - mainline
  - next
- 48 hour LTS regression reporting SLA
LKFT 2023 numbers

Linux Kernel
- 465 RC
- 2628 revisions
- 1.6M kernels
- 200M tests

Android Common Kernel
- 580M tests (VTS, CTS, ...)

Only 3 engineers
LKFT Architecture
How to build and test so many kernels?
LKFT architecture

GitLab → submit → Build and test

generate
LKFT Architecture

GitLab → submit → Cloud Build

Cloud Build → tuxmake → store → storage

Cloud Build → results → SQUAD

SQUAD → generate
LKFT Architecture

GitLab → submit → Cloud Build

Cloud Build → tuxmake → submit → Cloud Test

Cloud Test → storage → submit → Cloud Build

Cloud Build → store → SQUAD

SQUAD → generate → Cloud Build

Storage → results → SQUAD

Results → Cloud Test
LKFT Architecture

Cloud Build

GitLab

Cloud Test

storage

KissCache

Worker01

Worker02

Server

generate

submit

store

submit

submit

submit

results

results

results

SQUAD
Building
TuxMake

● OSS cli application
  ○ portable and repeatable Linux kernel builds
    - Containerized builds
  ○ https://tuxmake.org

● Multiple toolchains
  ○ gcc-8/9/10/11/12
  ○ clang-10/11/12/13/14/15/android/nightly

● Multiple target-archs
  ○ arm64/armv5/armv7
  ○ i386, x86_64
  ○ mips, powerpc, riscv
  ○ arc, hexagon, openrisc, parisc, s390, sh, sparc, um

Tuxsuite SaaS runs TuxMake at scale (5k builds in parallel) in the cloud
TuxMake explained

tuxmake --runtime podman --target-arch x86_64 --toolchain gcc-12 --kconfig defconfig

1. Pull the right container image
   a. docker.io/tuxmake/x86_64_gcc-12:latest...

2. Create a unique build directory
   a. ~/.cache/tuxmake/builds/XXX/build

3. Start the container with bindings
   a. Sources from CWD
   b. Build directory

4. Invoke make
   a. make --silent --keep-going --jobs=16 O=~/.cache/tuxmake/builds/XXX/build
      ARCH=x86_64 SRCARCH=x86 CROSS_COMPILE=x86_64-linux-gnu- defconfig

5. ...

   a. kernel, headers.tar.xz, modules.tar.xz
   b. metadata.json
TuxMake containers

- One container for each combination
  a. Toolchain version X target-architecture
     i. arm_gcc-11
     ii. arm_gcc-12
     iii. ...

- [https://hub.docker.com/u/tuxmake](https://hub.docker.com/u/tuxmake)
  a. 216 repositories
  b. Rebuild monthly
     i. Except for clang nightly
        1. Used by Clang CI pipeline
Testing
Virtual devices with TuxRun
TuxRun

- **OSS cli application**
  - **portable** and **repeatable** kernel tests
  - [https://tuxrun.org](https://tuxrun.org)

- **Multiple devices**
  - fvp-aemva (ARMv9.3)
  - fvp-morello
  - qemu-armv5/v7/v7be/64/64be
  - qemu-i386/x86_64
  - qemu-mips32/32el/64/64el, qemu-ppc32/64/64le, qemu-riscv32/64
  - qemu-s390/sh4/sparc

- **Multiple tests**
  - ltp-*, kunit, kselftest, rcutorture, perf, v4l2, libgpiod, libhugtlbfs

```
tuxrun --runtime podman --device qemu-arm64 --kernel Image --rootfs rootfs.ext4.zst
```

- Tuxsuite SaaS runs TuxRun at scale (5k tests in parallel) in the cloud
TuxRun explained

```bash
tuxrun --runtime podman --device qemu-arm64 --kernel Image --rootfs rootfs.ext4.zst
```

1. Download artefacts
   a. kernel, dtb, rootfs, modules, ...
      i. Provide default rootfs for every architecture
   b. Inject modules into rootfs
2. Start the container with artefacts embedded
3. Run qemu-system-aarch64
4. Parse the output for crashes
5. Run the tests
6. Store results.json
TuxRun rootfs

- Rootfs for multiple architecture are painful to build
- Default rootfs for each architect
  - a. Buildroot based: 19
  - b. Debian based: 19
  - c. Can still use custom ones
- Rebuilt regularly
  - a. Buildroot new releases
    - i. New ltp-testsuite package
  - b. Debian updates
  - c. Tested before deployment
    - i. Recently found multiple issues in qemu 7.2
TuxMake and TuxRun

- Combine TuxMake and TuxRun
- Bisect a run regression
  - Call git bisect
    - Checkout code
    - Cross-compile with tuxmake
    - Cross-run with tuxrun

```
git bisect start next-20230125 e5c645984a3884c92e124717c8c85635ba7a1857

git bisect run tuxmake --runtime podman --target-arch arm64 \ 
   --toolchain gcc-12 --kconfig defconfig \ 
   --results-hook "tuxrun --runtime podman --device qemu-arm64 --tuxmake ./ -- 'cd /opt/ltp && 
   .runltp -s hugeshmct101'"
```
Testing
Real devices with LAVA
LAVA

- **Linaro Automated Validation Architecture**
- Test execution system: **testing software on real hardware**
  - **Deploy, Boot** and **Test**
- **Usages**
  - Boot testing: kernelci
  - System level testing: LKFT
  - Bootloader/firmware testing
- **Supports 356 device-types**
Without LAVA

% power on board
% telnet localhost 2000
<enter>
=> dhcp
=> setenv serverip 10.3.1.1
=> [...]
=> bootm 0x01000000 - 0x03f00000
[...]
raspberry pi3 login: root
# run-test.sh
[...]
% power off board
LAVA explained

- Job Configuration
- worker
- kernel
dtb
rootfs
- Power control
- Serial relay
tftp\&nfs server
LAVA explained

Users

server

worker 1

worker 2

worker N

Power control

Serial relay

tftp&nfs server
Network performances

KissCache
KissCache

● LAVA downloads a lot of artifacts
  ○ Multiple times
  ○ In parallel (almost exact same time)

● SQUID should fix this?
  ○ Short answer NO!
  ○ Artefacts are served over https
    ■ Requires to fake SSL certificates
    ● Create a wildcard certificate (for every domains)
    ● Install on the clients
  ○ Multiple concurrent downloads of the same artefacts
    ■ SQUID will download multiple times the same artefacts
    ■ Cache only when a first download is completed
KissCache

● A simple and stupid caching server
  ○ Cache HTTPS resources
  ○ Download once while streaming to multiple clients
  ○ https://gitlab.com/Linaro/kisscache

● Not transparent (prefix based)
    ■ no need for fake SSL certificates
  ○ Need support in the clients

● Automatic retries on multiple errors
  ○ 408, 413, 420, 425, 429, 430, 500, 502, 503, 504, 507, 509, 529 and 598
  ○ Partial download
    ■ Will use range request to download remaining content
KissCache

- Over 2.5 years
  - 25 TB downloaded (from internet)
  - 1.3 PB served (local network) by KissCache
  - Network usage divided by 52x
  - Improved stability
Storing job results
SQUAD

- Software Quality Dashboard aka SQUAD
- A data lake
  - Gather results (builds, tests, measurements, …)
  - 3.3 billions results
- Create reports
  - Failures, regressions, …
- Links:
  - https://qa-reports.linaro.org/
  - Linaro instance
  - https://qa-reports.linaro.org/lkft/
  - LKFT project page
The end
Thank you