A Taste of Android Oreo (v8.0) Device Manufacturer

Keun Soo Yim, Iliyan Malchev, and Dave Burke
Android Platform Engineering, Google, Inc.
{yim,malchev,daveburke}@google.com

ABSTRACT

In 2017, over two billion Android devices developed by more than a thousand device manufacturers (DMs) around the world are actively in use. Historically, silicon vendors (SVs), DMs, and telecom carriers extended the Android Open Source Project (AOSP) platform source code and used the customized code in final production devices. Forking, on the other hand, makes it hard to accept upstream patches (e.g., security fixes). In order to reduce such software update costs, starting from Android v8.0, the new Vendor Test Suite (VTS) splits hardware-independent framework and hardware-dependent vendor implementation by using versioned, stable APIs (namely, vendor interface). Android v8.0 thus opens the possibility of a fast upgrade of the Android framework as long as the underlying vendor implementation passes VTS. This tutorial teaches how to develop, test, and certify a compatible Android vendor interface implementation running below the framework. We use an Android Virtual Device (AVD) emulating an Android smartphone device to implement a user-space device driver which uses formalized interfaces and RPCs, develop VTS tests for that component, execute the extended tests, and certify the extended vendor implementation.

1 GOAL

Android v8.0 (Oreo) has a modular operating system (OS) software architecture (namely, *Treble*[1]). It splits Android platform into hardware-independent framework and hardware-dependent vendor implementation layers by using a set of versioned, stable APIs which form the *Vendor Interface*. This new OS architecture helps entities in the Android device ecosystem respect separate ownerships of various software components in the Android platform and thus can deserialize Android device manufacturing chains. For example, it allows silicon vendors and device manufacturers to develop their portion of software in isolation and then integrate them as long as both layers are compatible with the targeted vendor interface APIs, which are tested and certified by our new *Vendor Test Suite* (VTS)[2]. This tutorial will give hands-on experience on extending a vendor interface implementation, developing VTS tests for that, and running various kinds of VTS tests against an Android virtual device (AVD) instance for training, rapid prototyping, continuous testing at early software development phase, vendor interface compatibility evaluation, and device fidelity assessment.

SOSP'17, October 2017, Shanghai, China

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

2 ORGANIZATION

(i) Android Vendor Interface and Android v8.0 on a Virtual Phone. *(a) Concept:* to understand Android Treble architecture specifically focusing on its HIDL (HAL[1] Interface Definition Language) and HwBinder (i.e., analogous to existing Android IDL (AIDL) and Binder), common kernel interface, and vendor NDK (i.e., analogous to existing NDK (Native Development Kit)). *(b) Codelab:* based on a virtual smartphone emulator and locally built Android v8.0 platform images for that emulator[2], to create an AVD instance and interact with that using a graphical user interface and an `adb` shell terminal. *(c) Discussion:* to analyze the key benefits of Treble architecture and compare Treble with techniques used in other existing OSes (e.g., component-based design for C++).

(ii) Device Implementation and Tuning. *(a) Concept:* to learn HIDL HAL coding guidelines by analyzing existing HIDL HAL designs and implementations. *(b) Codelab:* to modify an HIDL HAL and rebuild the AVD image in order to restart the AVD instance using that new image and interact with the modified HIDL HAL where a HAL modification task includes associated SELinux policy changes and performance profiling (using `systrace`). *(c) Discussion:* to evaluate the performance and power overheads of HIDL*ized* HALs (*vs.* previous shared library HALs) on devices with multiple processor cores, visualize the priority inversion and inheritance issues when an HAL client uses Linux FIFO or RR scheduling priority, and analyze the death notification mechanism designed to handle when a HIDL server crashes and its fault tolerance implications.

(iii) Test Development and Execution. *(a) Concept:* to understand Android VTS, CTS (Compatibility Test Suite), and the associated licensing processes; and to learn various other automated testing, performance profiling, and fuzzing techniques designed for Android OS. *(b) Codelab:* to implement a VTS structural test, a profiling test, and a fuzz test for an HAL, build a VTS package, and run a VTS test plan against the AVD instance in order to get a test report. *(c) Discussion:* to compare test-driven development practices between systems development and application development, to understand various approaches to coordinate developers who develop and extend VTS tests, to discuss how VTS can statistically guarantee the forward compatibility of a set of tested vendor interface APIs, and to brainstorm what methodologies are available for us to develop and assess VTS tests which provide such statistical guarantee.

REFERENCES

[1] AOSP. 2017. Treble. https://source.android.com/devices/architecture/treble. (21 August 2017).
[2] AOSP. 2017. VTS. https://source.android.com/devices/tech/vts/. (21 August 2017).

[1] Hardware Abstraction Layer where a HAL module is conceptually similar to a user-space device driver which typically wraps the respective kernel space driver module.
[2] Audience prerequisites: https://sites.google.com/site/keunsooyim/open-lecture/sosp2017tutorial