

Embedded AOSP

Lecture 3. Android HAL


09-13 July

Polytechnic University of Bucharest

Partnered with



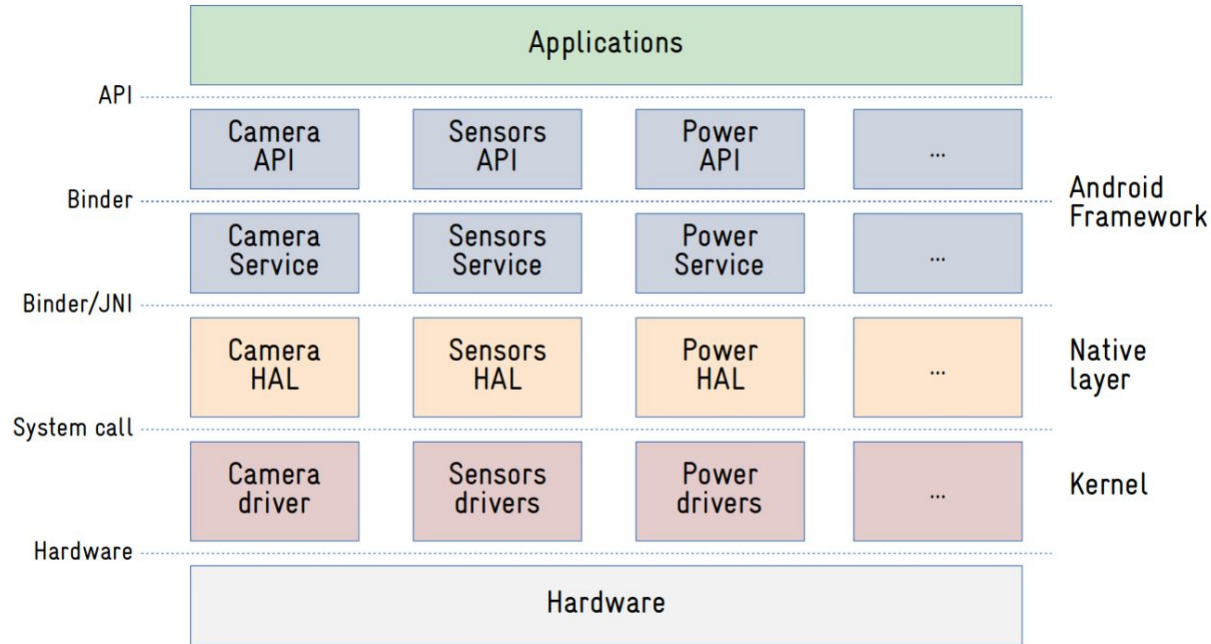
Outline

- Why HAL?
- IPC. AIDL.
- Binder
- Versioning & Vendoring
- SELinux 

Hardware Abstraction Layer

- All OSes do HAL via drivers!
- Android has a higher-level HAL component on top of kernel
 - System Services + Frameworks
 - E.g.: camera, sound, sensors, display etc.
- Android HAL designed for compatibility & extensibility!
 - Linux drivers' API/ABI are NOT stable inside the kernel!
- Abstract interfaces?
 - Must be able to describe APIs with strict form!
 - Solution: Interface Description Languages (IDL)!
 - History: first was IHDL for hardware interfaces, now unified using AIDL!
- Inter-process Communication (Binder)

Android HAL Architecture



Android Interface Definition Language (AIDL)

- Custom, Java-like syntax for defining interfaces, data types etc.
- Cannot contain implementations!
- Tools to generate .java / .cpp / .rs classes (**Stubs**)
- Services must implement stub interface
- Users (apps) will import stub interface and call its methods!
- IPC will glue them ;)

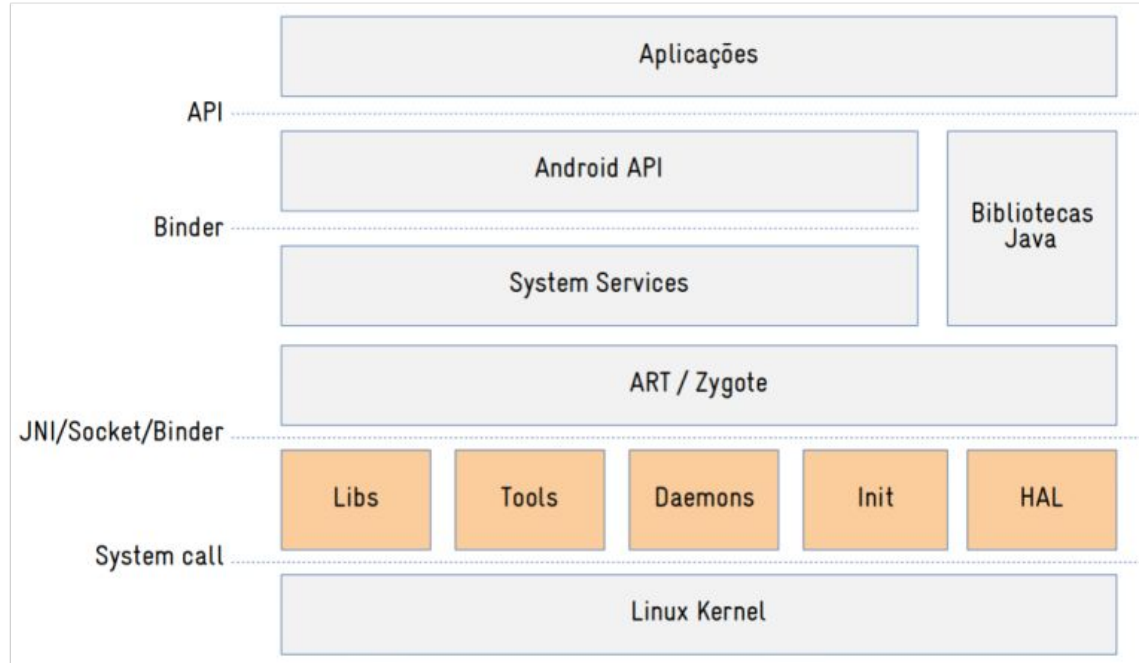
```
// IRemoteService.aidl
package com.example.android;

// Declare any non-default types here with import
statements.

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service. */
    int getPid();

    /** Demonstrates some basic types that you can
     * use as parameters and return values in AIDL.
     */
    void basicTypes(int anInt, long aLong, boolean
aBoolean, float aFloat,
                    double aDouble, String aString);
}
```

The Android native layer



Binder

- History: in Android ≤ 7 , HALs were implemented in shared libraries (.so), consumed by Java framework services via JNI :|
- Main IPC / RPC mechanism in Android, enforces strict versioning & security rules
- All communication between Android components (applications, services, daemons and HALs) => via Binder
- Uses custom kernel devices:
 - The Binder interface is exposed to the user using /dev files, accessed via `ioctl()` calls:

```
# ls /dev/*binder
```

```
/dev/binder
```

```
/dev/hwbinder
```

```
/dev/vndbinder
```

Vendoring & Versioning

- 2017: Project Treble – modular interfaces
 - starting with Android 8, stabilized and made the interface between the Android framework and the hardware manufacturer's implementation (vendor/oem) independent!
- Google ensures that the GSI (Generic System Image) is backwards compatible with the last 3 releases of the operating system
- Several changes were made to the operating system architecture to make this possible (Binderized HALs, HIDL, VTS, VNDK, VINTF, GSI, etc.)
- HALs and other customizations are distributed in separate partitions (/vendor, /product, /odm), allowing the /system partition to be updated independently
 - The **VINTF** manifest (Vendor Interface Object) allows to specify & retrieve compatibility information between the Android framework and the vendor's implementation

SELinux

- Security-Enhanced Linux (SELinux) is a mandatory access control (MAC) system integrated into Android to enforce security at the kernel level. It ensures that all processes and files are labeled with a security context, and policies dictate how these contexts can interact. This significantly mitigates the impact of security vulnerabilities.
- Everything in Android, from processes and files to sockets, has a specific label (e.g., `u:object_r:system_file:s0`)
- Policies: These are the rules that specify which contexts can access others and what actions are allowed (e.g., read, write, execute). Android's global policy is complemented by device-specific policies.