Android OS

Documentation

Done by:

Akhil Sai Vodnala

Contents

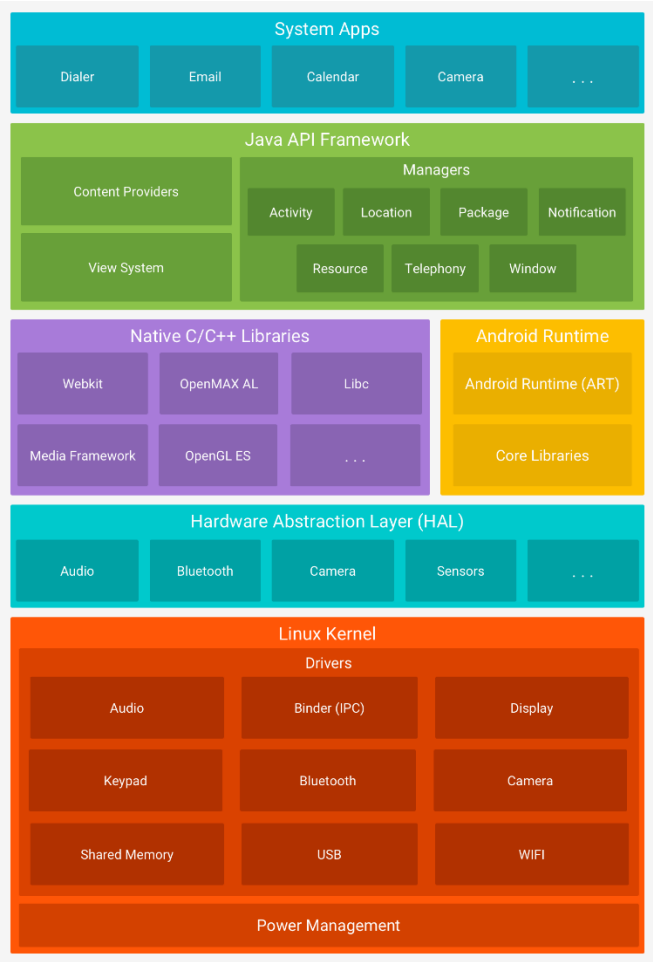
Introduction

Android Architecture defines how the Android Operating System is structured internally, from **applications at the top** down to the **Linux kernel at the bottom**. It is designed to provide: - Hardware abstraction - Process isolation and security - Efficient resource management - Support for both Java/Kotlin and Native (C/C++) applications.

Understanding Android Architecture is essential for **AOSP development**, **NDK-based applications**, **system programming**, and **embedded Android systems**.

High-Level Android Architecture

Android architecture is commonly represented as a **layered stack**:



Each layer has a clearly defined responsibility and communicates with adjacent layers using well-defined interfaces.

1. Applications Layer

Description

This is the **topmost layer**, consisting of all user-facing applications: - System apps (Phone, Settings, Camera) - Third-party apps - Custom AOSP applications

Key Characteristics

- Written primarily in **Java/Kotlin**
- Can also include **native C/C++ code via NDK**
- Each app runs in its **own Linux process and UID**

Security

- Application sandboxing enforced by Linux kernel
- Permission-based access to system resources

2. Application Framework Layer

Role

The Application Framework provides **high-level system services** that applications use to interact with the Android system.

Important Framework Components

Service	Responsibility
Activity Manager	Manages app lifecycle & tasks
Window Manager	Manages windows and UI
Package Manager	App installation & permissions
Resource Manager	Access to strings, layouts, drawables
Notification Manager	Notifications handling
Location Manager	GPS & location services
Camera Manager	Camera access (Java API)

Characteristics

- Exposed as **Java APIs**
- Internally communicate with system services using **Binder IPC**
- Abstracts hardware and system complexity from apps

3. Android Runtime (ART)

Purpose:

ART is responsible for **executing Android applications** written in Java/Kotlin.

Key Features:

- Ahead-Of-Time (AOT) compilation
- Just-In-Time (JIT) compilation
- Garbage Collection (GC)
- Thread and memory management

Execution Flow:

.java/.kt → .dex → oat (native code)

ART runs inside each application process, ensuring isolation and performance.

4. Native C/C++ Libraries

Description

Android provides a rich set of **native libraries** written in C/C++ that power the core system.

Common Native Libraries

Library	Purpose
Bionic libc	Android's C standard library
Media Framework	Audio/Video playback & recording
SurfaceFlinger	Display composition
OpenGL ES / Vulkan	Graphics rendering
SQLite	Database engine
SSL / WebKit	Secure communication

Usage

- Used internally by Android framework
- Accessible to apps via **NDK**

5. Hardware Abstraction Layer (HAL)

Purpose

HAL acts as a **bridge between hardware drivers and Android framework**.

Why HAL Exists

- Allows Android to run on different hardware
- Avoids framework changes for hardware differences

Examples of HAL Modules

- Camera HAL
- Audio HAL
- Sensors HAL
- GPS HAL
- Bluetooth HAL

Interface

- Historically HIDL
- Modern Android uses **AIDL-based HALs**

6. Linux Kernel Layer

Role

The Linux kernel is the **foundation of Android OS**.

Responsibilities

- Process management
- Memory management
- Device drivers
- Networking
- Power management
- Security (SELinux)

Android-Specific Kernel Features

- Binder IPC driver
- Ashmem (deprecated, replaced by memfd)
- Wakelocks
- SELinux enforcement

7. Binder IPC Mechanism

What is Binder?

Binder is Android's **core IPC mechanism** enabling communication between processes.

Used By

- Application Framework
- System services
- HAL services

Binder IPC Communication Flow

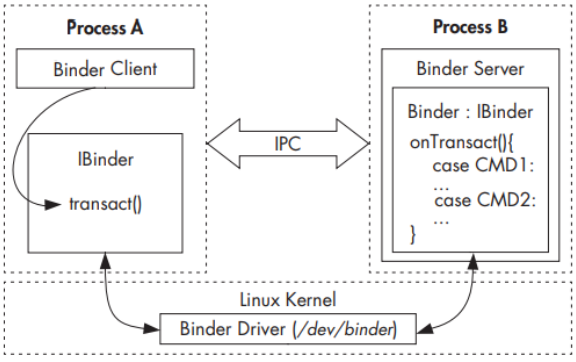


Figure 1-3: Binder IPC

Explanation: - Binder enables secure, high-performance IPC - Used by ActivityManager, CameraService, MediaService

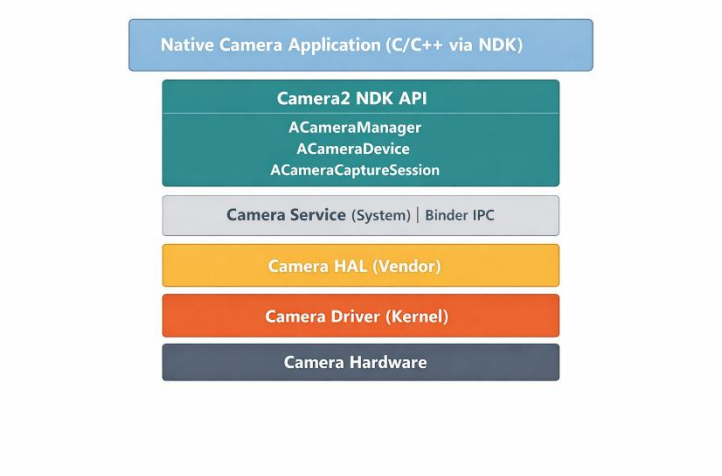
Binder ensures: - Secure IPC - High performance - Reference counting

8. Android Boot Process

1. Bootloader
2. Linux Kernel
3. Init process
4. Zygote
5. System Server
6. Launcher & Apps

Each step gradually brings the Android system to a usable state.

Android Camera Stack



Explanation: - Your app bypasses Java Camera APIs - Direct interaction with **Camera Service** via NDK - HAL translates framework requests into hardware commands

Conclusion

Android Architecture is a layered, modular system designed for security, performance, and hardware independence. Understanding the interaction between **Applications, Framework, Runtime, HAL, and Kernel** is essential for AOSP development, NDK programming, and system-level Android engineering.