Mobile Programming

Android System Structure

Lecturer: Ali Najimi
Faculty of Computer Engineering
Sharif University of Technology
Fall 2027

Android Architecture | Lifecycle | Permissions | APK



Table of Contents

1. The Android Architecture Layers
2. The Linux Kernel & Hardware Abstraction
3. Android Runtime (ART) & Dalvik
4. The Application Framework & APIs
5. The Application Layer & Sandbox
6. Android Manifest: The Control Center
7. The APK File Format
8. Deep Dive: App Lifecycle & States

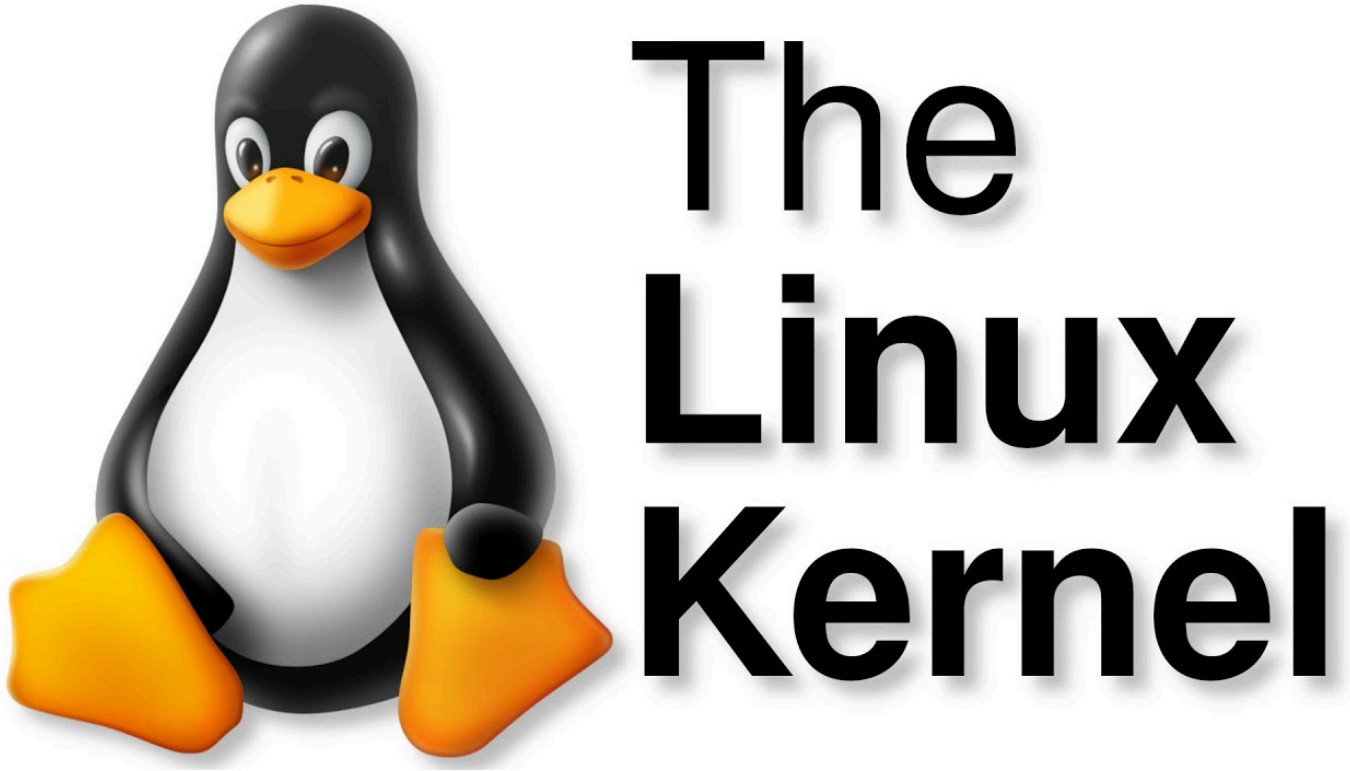
The Android Architecture Layers

Android is a **stack of software** components. Understanding this stack is crucial.



Key Takeaway: Your app sits at the top, but it relies heavily on the layers below.

The Linux Kernel



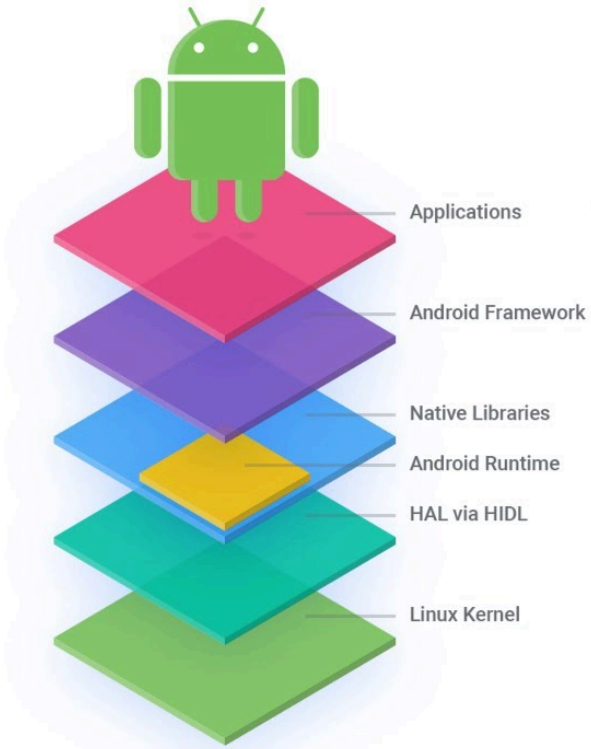
The Linux Kernel

The foundation. It is the **only** part of the OS that interacts directly with the hardware.

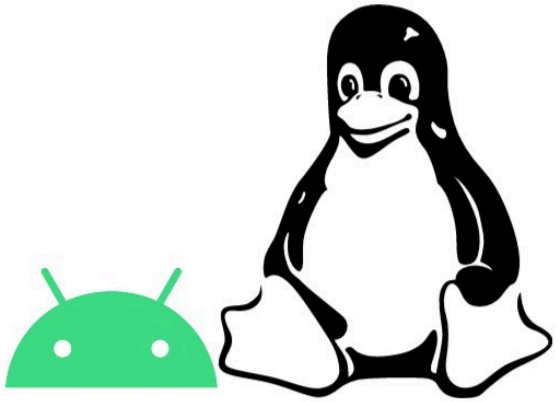
- **Process Management:** Memory allocation, CPU scheduling, thread management.
- **Security:** User-based permissions (UID/GID) isolate apps from each other.
- **Drivers:**
  - Display, Camera, Bluetooth, Wi-Fi.
  - **Binder:** The driver that enables Inter-Process Communication (IPC).

> **Note:** Android does **not** use the standard Linux kernel exactly; it has specific modifications for mobile power management.

Native Libraries & HAL



Understanding the internals of Android Stack Architecture and how it relates to Linux



Native Libraries & HAL

The **HAL (Hardware Abstraction Layer)** provides standard interfaces that expose device hardware capabilities to the higher-level Java API.

- **Standard C/C++ Libraries:**
  - **OpenGL ES:** 3D graphics.
  - **Media Framework:** Playback/Recording (AAC, MP4, MP3).
  - **SSL:** Web security.
  - **SQLite:** The core database engine for Android.
  - **Libc (Bionic):** Standard C library optimized for mobile.

Android Runtime (ART)

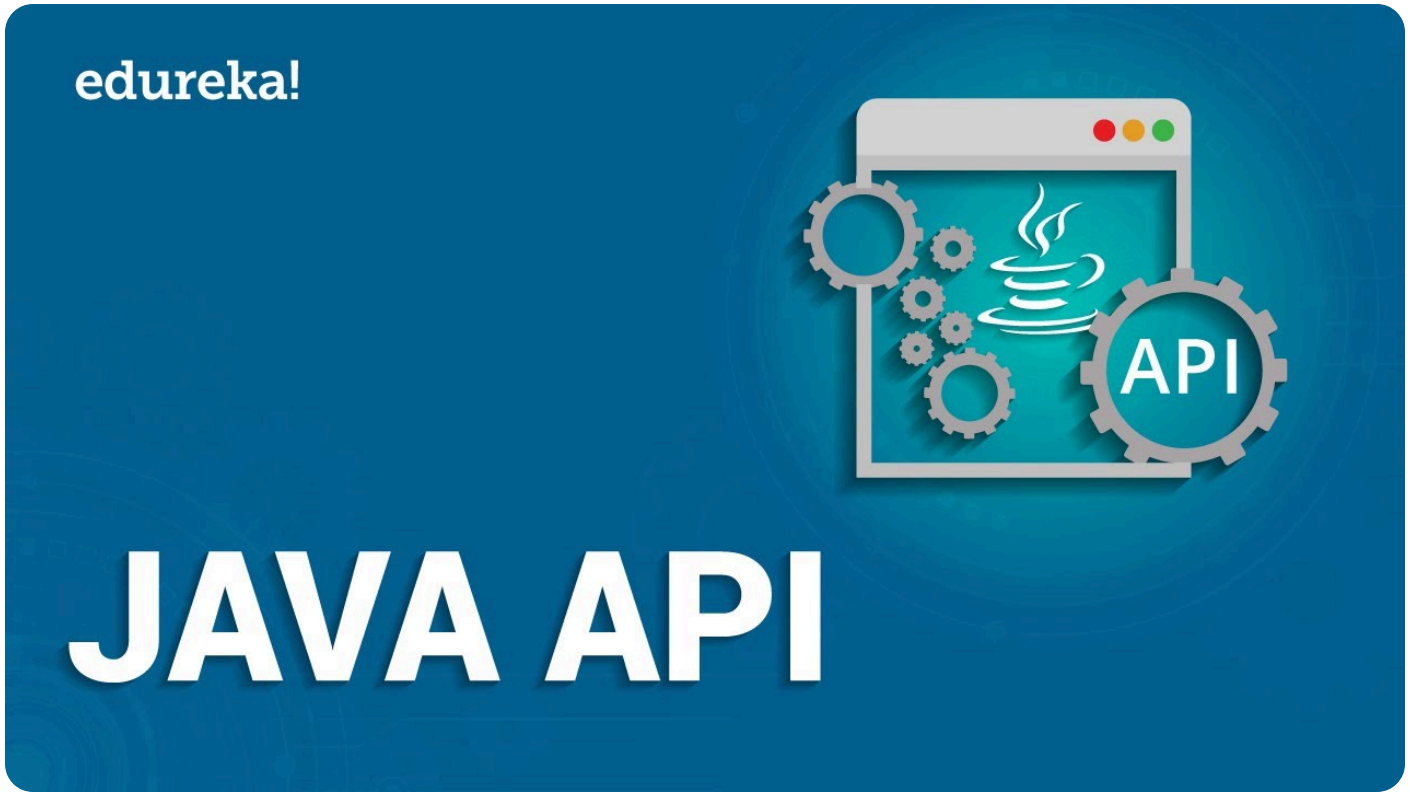


Android Runtime (ART)

Every Android app runs in its own process, with its own instance of the ART virtual machine.

- **ART (Android Runtime):** Replaced Dalvik (API 21+).
  - Uses **AOT (Ahead-Of-Time)** compilation.
  - Converts bytecode into native machine code at install time.
  - Results in faster app execution and lower battery usage compared to Dalvik's JIT (Just-In-Time).

Java API Framework

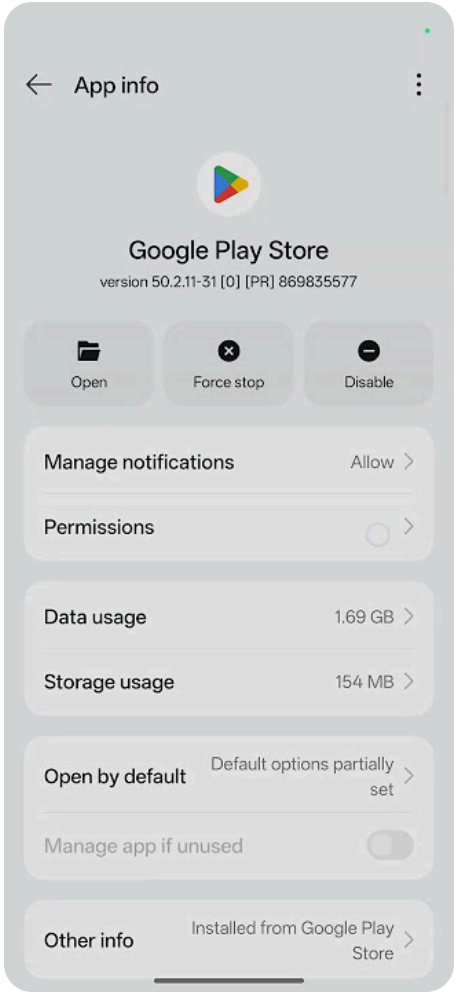


Java API Framework

The set of tools and APIs you use to build Android apps.

- **View System:** UI elements (Buttons, Lists, Layouts).
- **Resource Manager:** Accessing strings, graphics, and layout files.
- **Notification Manager:** Alerts in the status bar.
- **Activity Manager:** Managing app lifecycle and navigation stack.

Application Layer



Application Layer

Where the user interacts.

- **System Apps:** Home (Launcher), Dialer, Contacts, Camera.
- **User Apps:** Apps that you (or we) develop.
- **Sandbox:** Each app runs in its own Linux process, with a unique user ID (UID).
  - App A cannot directly access App B's data or memory.

The Android Manifest

`AndroidManifest.xml` is the **control file** for your app. Without it, the OS doesn't know your app exists.

Key Roles:

1. **Identity:** Package name (e.g., `edu.sharif.mp`).
2. **Components:** Declares Activities, Services, Receivers, Providers.
3. **Permissions:** Requests access to sensitive data (Camera, Location).
4. **Intent Filters:** Defines which "events" your app can handle (e.g., `android.intent.action.MAIN`).

APK Structure

An **APK (Android PacKage)** is just a **ZIP file** containing all app resources.

- `AndroidManifest.xml` : The binary XML manifest.
- `classes.dex` : The compiled Dalvik bytecode (your code).
- `resources.arsc` : Precompiled resources (strings, styles).
- `res/` : Folder for images, layouts, XML.
- `lib/` : Native libraries (`.so` files) for specific CPU architectures (ARM, x86).
- `META-INF/` : Signatures (MANIFEST.MF, CERT.SF, CERT.RSA).

App Lifecycle & States

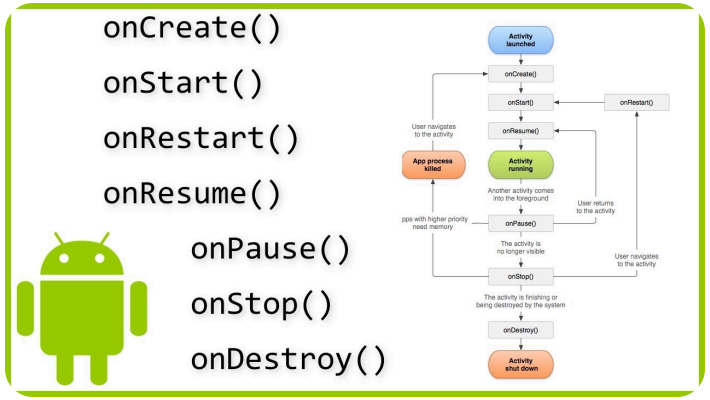
The **Activity Lifecycle** is the most critical concept to master. If you get this wrong, you leak memory and crash.

Active States

- **Created:** Layout inflated.
- **Started:** Visible to user.
- **Resumed:** Interactable (Foreground).

Killer States

- **Paused:** Partially obscured (e.g., dialog).
- **Stopped:** Hidden (e.g., user pressed Home).
- **Destroyed:** Process is killed.



Deep Dive: The "Killer" Scenario

Why did my app disappear?

1. **User presses Home:** App is `Stopped`. Process is kept in memory as a cache.
2. **System needs memory:** Android (LMK - Low Memory Killer) kills the process to reclaim RAM.
3. **User returns to App:** Android must **re-create** the process.
4. **Your Job:** Save data in `onPause()` or `onSaveInstanceState()` so the UI looks the same when restored.

> **Golden Rule:** Do **not** trust that your app's static variables will survive a background kill.

Summary: System Thinking

- **Architecture:** Layered stack (Kernel → HAL → Native → Framework → App).
- **Runtime:** ART compiles DEX code to machine code.
- **Security:** Sandbox + UID + Permissions.
- **Lifecycle:** You do not control when your app dies. You only control how it *reacts* to lifecycle events.

Mini Lab

System Behavior Inspector

Objective: Create a simple app that logs lifecycle events.

1. **Log Events:** Override `onStart`, `onResume`, `onPause`, `onStop`, `onDestroy`.
2. **Visualize:** Print logs to `Logcat`.
3. **Simulate Kill:**
   - Run app.
   - Press Home.
   - Use Terminal (`adb shell`) to force stop the process.
   - Relaunch and observe `onCreate` running again.

```
adb shell am kill edu.sharif.mp
```

Real-Life Application

Background Audio Player Logic

- **Requirement:** Music must continue when screen is off.
- **System Conflict:** `Stopped` activity usually kills background threads.
- **Solution:** Use a **Foreground Service**.
  - Shows a persistent notification.
  - Tells OS: "I am important, do not kill me."
  - Survives the `onStop` lifecycle event.

```
// High-level pseudo-code for Flutter implementation
startForegroundService() {
  // Keeps process alive even if UI is destroyed
}
```

Thank You! 🙌

Questions?

"A UI engineer walks into a bar.
The bartender says, 'Sorry, we don't serve your thread here.'
The engineer replies, 'It's okay, I'll just wait on the Main Thread.'"