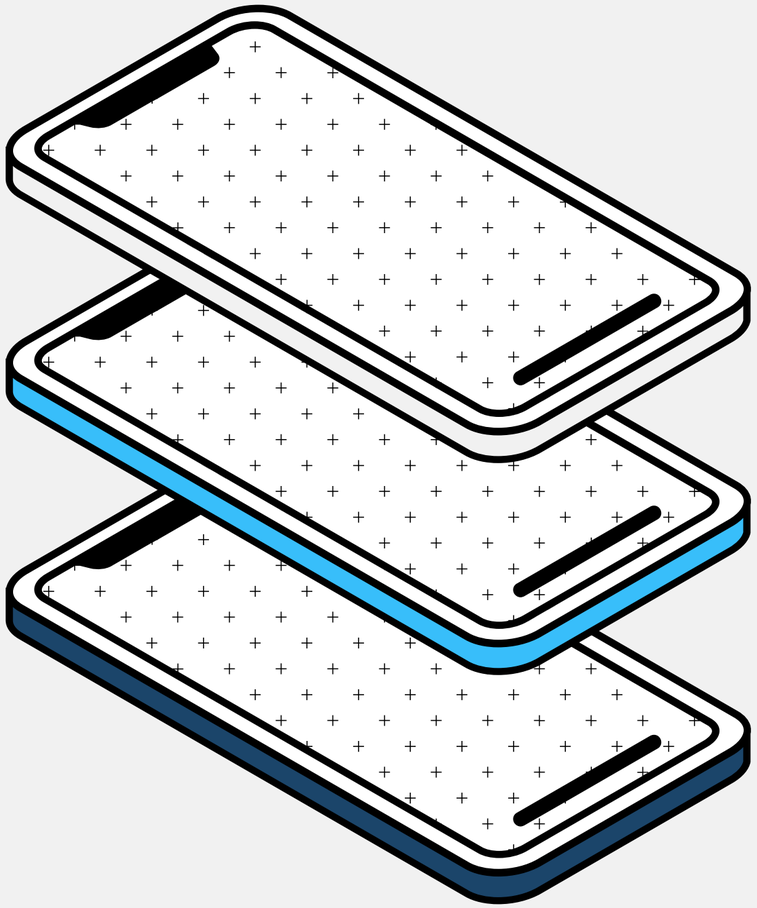
บทที่ 3

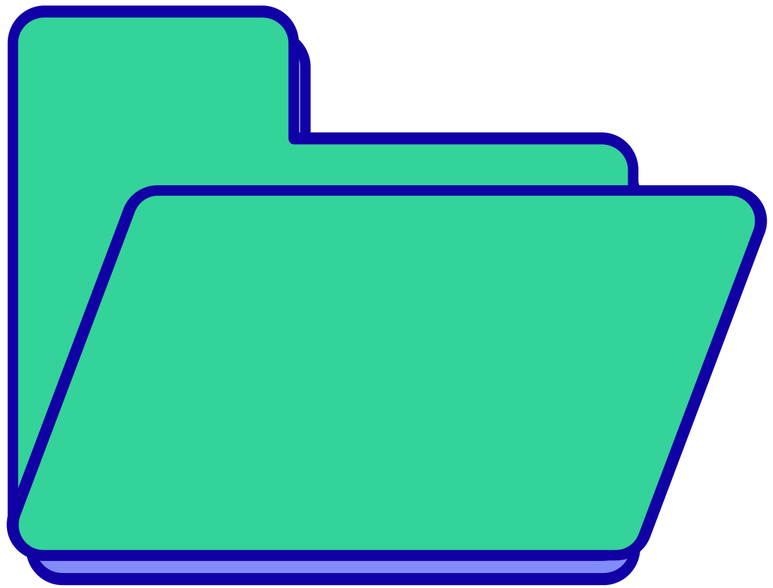
สถาปัตยกรรมระบบปฏิบัติการ



ดร.พงษ์ดนัย จิตตวิสุทธิกุล

Topic

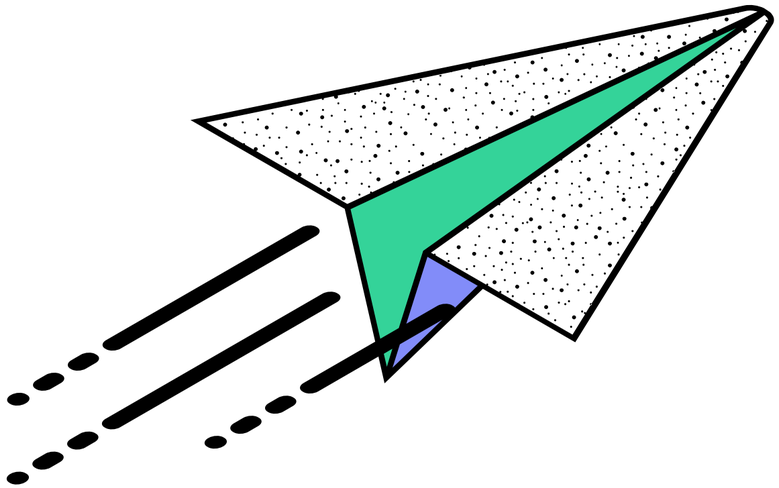
6 หัวข้อหลัก:



- สถาปัตยกรรมระบบปฏิบัติการ Android
- สถาปัตยกรรมระบบปฏิบัติการ iOS
- โมเดลความปลอดภัย
- วงจรชีวิตของแอปพลิเคชัน
- เปรียบเทียบเชิงลึกและอภิปราย
- ติดตามเส้นทางการทำงานของคำสั่ง

Objective behavior

3 วัตถุประสงค์เชิงพฤติกรรม:



ระบุหน้าที่ของแต่ละชั้นในสถาปัตยกรรม Android และ iOS และอธิบายความแตกต่างเชิงปรัชญาของทั้งสองระบบปฏิบัติการได้

เปรียบเทียบโมเดลความปลอดภัยและกลไกการทำงานของ Application Sandboxing ใน Android และ iOS ได้

อธิบายและเปรียบเทียบสถานะต่าง ๆ ในวงจรชีวิตของแอปพลิเคชัน (Application Lifecycle) ของ Android และ iOS ได้

จาก Hardware สู่ Software

ความเดิมตอนที่แล้ว

- SoC: เปรียบเสมือน "มหานคร" ที่รวม CPU, GPU, NPU ไว้ด้วยกัน
- RISC: ปรัชญา "ประสิทธิภาพต่ออวัตต์"
- แต่ฮาร์ดแวร์ที่ดี... ต้องการ "ผู้จัดการ" ที่เก่งกาจ

เนื้อหาในบทนี้


Android Architecture (5 Layers)


iOS Architecture (4 Layers)


Security Model (Sandboxing)


App Lifecycle


Trace Path (การเดินทางของคำสั่ง)

Android Architecture: 5 Layers Stack

System Apps / User Apps

Java API Framework

Native C/C++ Libraries & ART

Hardware Abstraction Layer (HAL)

Linux Kernel

ปรัชญา: ความเปิดกว้าง (Openness)

Android ถูกออกแบบมาให้ยืดหยุ่น ทำงานได้บนอุปกรณ์ที่หลากหลาย โครงสร้างแบบชั้นช่วยแยกส่วนฮาร์ดแวร์ออกจากซอฟต์แวร์แอปพลิเคชัน

Key Concept: นักพัฒนาเขียนโค้ดครั้งเดียว รันได้ทุกเครื่อง เพราะมีชั้น HAL และ ART คอยจัดการความแตกต่างของฮาร์ดแวร์

เจาะลึก Android: รากฐาน (Lower Layers)

1. Linux Kernel
   - เป็นหัวใจหลัก จัดการ Process, Memory, Power Management
   - บรรจุ Device Drivers (เช่น กล้อง, WiFi, จอภาพ)
   - ดูแลเรื่อง Security พื้นฐาน (User Permissions)

2. Hardware Abstraction Layer (HAL)
   - ทำหน้าที่เป็น "ล่ามแปลภาษา" (Interface มาตรฐาน)
   - กำหนดมาตรฐานให้ผู้ผลิตชิป (Vendors) เขียน Driver มาเชื่อมต่อ โดยไม่ต้องแก้ OS หลัก

3. Native Libraries & ART

Native C/C++ Libraries: ชุดเครื่องมือประสิทธิภาพสูง
   - OpenGL/Vulkan: กราฟิก 3D
   - *WebKit:* แสดงผลหน้าเว็บ
   - *SQLite:* ฐานข้อมูล

Android Runtime (ART):
   - ใช้เทคนิค AOT (Ahead-Of-Time) คอมไพล์โค้ดตั้งแต่ติดตั้ง ทำให้เปิดแอปเร็ว
   - จัดการหน่วยความจำด้วย Garbage Collection (GC)

เจาะลึก Android: ชั้นบริการ (Higher Layers)

4. Java API Framework

ชุดเครื่องมือ (Toolkit) ที่นักพัฒนาเรียกใช้บ่อยที่สุด:

- Activity Manager: จัดการหน้าจอและวงจรชีวิตแอป
- Content Providers: แชร์ข้อมูลข้ามแอป (เช่น ดึงรายชื่อติดต่อ)
- Notification Manager: จัดการการแจ้งเตือน
- View System: สร้าง UI (ปุ่ม, ข้อความ, ตาราง)

5. System Apps & User Apps

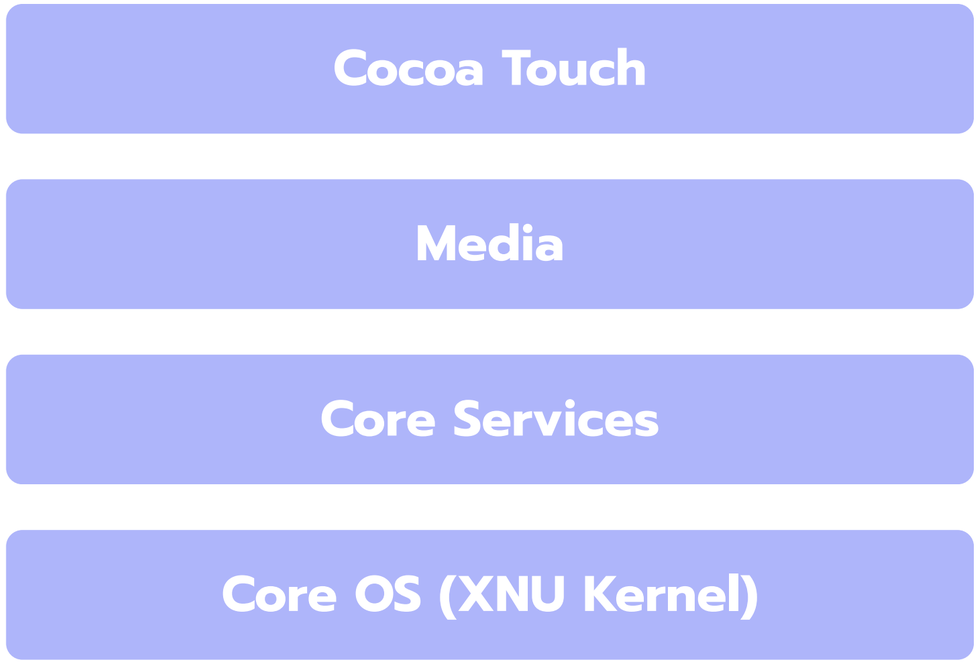
ชั้นบนสุดที่ผู้ใช้มองเห็นและสัมผัส:

- รวมทั้งแอปที่มากับเครื่อง (System) เช่น โทรศัพท์, ปฏิทิน
- และแอปที่โหลดเพิ่ม (Third-party)

"ในมุมมองของสถาปัตยกรรม

แอปของระบบและแอปของเรา มีสถานะเท่าเทียมกัน"

iOS Architecture: Vertical Integration



ปรัชญา: การบูรณาการแนวดิ่ง

Apple ควบคุมทั้ง Hardware และ Software ทำให้โครงสร้างมีความกระชับ (Compact) และปรับจูนมาเพื่อฮาร์ดแวร์เฉพาะรุ่น

- รากฐานมาจาก UNIX (Darwin) ที่แข็งแกร่ง
- เน้นประสบการณ์ผู้ใช้ที่ลื่นไหล (Fluidity) และความปลอดภัย (Privacy)

เจาะลึก iOS: 4 Layers Stack

1. Core OS Layer
   - ฐานคือ Darwin (XNU Kernel): เป็น Hybrid Kernel (Mach + BSD)
   - จัดการ Low-level tasks: Networking, Security, Accessory Access

2. Core Services Layer
   - Foundation Framework: จัดการข้อมูล String, Array
   - Core Location: ระบุตำแหน่ง GPS
   - Core Data: จัดเก็บข้อมูลถาวร

3. Media Layer

ขุมพลังมัลติมีเดีย:
   - Metal: API ระดับล่างเพื่อรีดพลัง GPU (กราฟิก/เกม)
   - Core Animation: เบื้องหลังความลื่นไหลของ UI
   - AVFoundation: จัดการเสียงและวิดีโอ

4. Cocoa Touch Layer
   - UIKit / SwiftUI: เครื่องมือสร้างหน้าจอ UI ทั้งหมด
   - เป็นประตูหลักที่นักพัฒนาใช้งาน

Security Model: Application Sandboxing

แนวคิด "กระบะทราย": จำกัดพื้นที่เล่นของแอป เพื่อความปลอดภัยของระบบ

Android Mechanism

- Unique User ID (UID): เคอร์เนล Linux มองแอปแต่ละตัวเป็น "ผู้ใช้คนหนึ่ง" ที่มี ID ต่างกัน
- Process Isolation: แอป A ไม่มีสิทธิ์แตะต้องไฟล์ของแอป B โดยตรง
- Permissions: ต้องประกาศขออนุญาตใน AndroidManifest.xml เพื่อขอใช้ทรัพยากร (เช่น กล้อง)

OS Mechanism

- App Container: ทุกแอปถูกขังอยู่ในโฟลเดอร์ของตัวเอง อ่าน/เขียนได้แค่ในนั้น
- No Direct Access: ห้ามเข้าถึงไฟล์ระบบหรือแอปอื่นเด็ดขาด
- Entitlements: ความสามารถพิเศษ (เช่น iCloud, Push Notif.) ต้องถูกระบุและ Digitally Signed มากับแอป แก้ไขทีหลังไม่ได้

Application Lifecycle (วงจรชีวิตแอป)

Android: Activity Lifecycle

จัดการในระดับ **หน้าจอ (Activity)** มีสถานะซับซ้อน:

- Resumed: อยู่หน้าสุด ใช้งานได้
- Paused: ถูกบังบางส่วน (เช่น มี Dialog เด้ง) -> *Save ข้อมูลที่นี่*
- Stopped: ถูกบังมิด (สลับแอป)
- Destroyed: ถูกทำลาย (คืนหน่วยความจำ)

*ข้อควรระวัง: การหมุนหน้าจอ = Destroy แล้ว Create ใหม่

iOS: App Lifecycle

จัดการในระดับ **แอปพลิเคชัน** หรือ **Scene**:

- Active: ใช้งานอยู่ปกติ
- **Inactive:** อยู่หน้าจอแต่ไม่ได้รับ Event (เช่น มีสายเข้าแทรก)
- **Background:** ทำงานเบื้องหลังชั่วคราว
- **Suspended:** "แช่แข็ง" อยู่ใน RAM แต่ไม่รันโค้ด CPU

*iOS เน้นการ Freeze แอปเพื่อประหยัดแบตเตอรี่สูงสุด

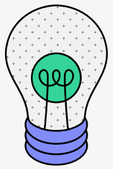
Trace Path: กรณีศึกษา "การเปิดกล้อง" (Android)

เมื่อนิ้วแตะไอคอนกล้อง... คำสั่งเดินทางอย่างไร?

1. App Layer: Launcher ส่ง Intent ขอเปิดแอป
2. Framework: Activity Manager เริ่ม Process ของแอป
3. App Code: เรียกใช้ CameraManager API
4. HAL: ส่งคำสั่งผ่าน Interface มาตรฐาน
5. Kernel: เรียกใช้ Camera Driver ของผู้ผลิต
6. Hardware: เลนส์กล้องทำงาน รับแสง

↑ Return: ส่งข้อมูลภาพย้อนกลับขึ้นไปแสดงผลบนจอ

Questions & Discussion



"... เราได้เห็น Layered Architecture ที่แยกส่วนชัดเจน... ได้เห็นกลไก Security ที่ปกป้องข้อมูลผู้ใช้... และ Lifecycle ที่บริหารจัดการทรัพยากร..."