DAA
Assignment 3

Session-Fall-24
Submitted by:
Name: Muhammad Usman
ID:221400
Section: BSCS-5-C

Submitted to: Sir Junaid Nazar

Date: 22-Dec-2024

Department of Computer Science
Faculty of Computing and AI
Air University, Islamabad

Analysis Report of Android and macOS

1. Introduction :

Operating systems (OS) are the foundation of modern computing, serving as the interface between users, applications, and hardware.

→This report examines the architecture and core functionalities of a mobile OS (Android) and a desktop OS (macOS) in five critical areas: process management, memory management, file systems, security, and scheduling.

→ My analysis incorporates insights from recent research papers and includes a creative analogy to illustrate the differences.

2. Process Management :

Android:

- **Process Creation and Management:** Android uses a Linux-based kernel to manage processes. Each application runs in its own process, isolated by the Linux kernel for security and stability. Processes are created using the **fork()** system call, and the Zygote process enables quick app initialization by preloading shared resources.

  Multitasking and Scheduling: Android employs the Completely Fair Scheduler (CFS) to manage multitasking, ensuring fair CPU time allocation. The system's Activity Manager oversees task prioritization, ensuring that foreground applications receive sufficient resources while background tasks are paused or terminated during memory

- Android employs the Linux kernel for process management, enabling preemptive multitasking.
- Processes are created using fork and exec system calls.
- Inter-process communication (IPC) is facilitated via the Binder framework, which supports efficient and secure communication.
- The Zygote process significantly optimizes app launch time by preloading common libraries and forking new processes from a template.

macOS:

- **Process Creation and Management:** macOS uses Mach-O as its binary format and relies on the Mach kernel alongside BSD subsystems to handle processes. Processes are created via fork() and exec(), and their execution is managed with fine granularity using priority queues and task groups.

- **Multitasking and Scheduling:** macOS employs a hybrid scheduling algorithm combining priority-based and round-robin techniques. Grand Central Dispatch (GCD) is a cornerstone of task scheduling, enabling developers to optimize application

performance by efficiently managing concurrent operations across multi-core processors.

- macOS utilizes the XNU kernel, which combines Mach microkernel and BSD subsystems.
- Processes in macOS are managed using Mach tasks and threads, allowing fine-grained control over execution.
- IPC is implemented using XPC services and Mach messages, ensuring secure communication.
- macOS excels in supporting heavy multitasking with robust resource allocation.

Comparison:

- Android prioritizes battery efficiency and lightweight multitasking, whereas macOS emphasizes performance and scalability.
- The Binder framework in Android is more lightweight compared to the more feature-rich XPC in macOS.

3 Memory Management

Android:

Allocation and Deallocation: Memory management in Android is tightly integrated with the Dalvik Virtual Machine (DVM) and the Android Runtime (ART). The Zygote process preloads core libraries and frameworks into shared memory, reducing overhead during app startup. Garbage collection in ART ensures efficient memory reclamation, minimizing latency for UI threads.

Virtual Memory and Caching: Android's reliance on the Linux kernel enables robust virtual memory management, including paging and memory overcommit handling.

- **Memory Protection:** The Linux kernel enforces process isolation, preventing direct memory access between applications. Android further enhances security with app sandboxing, ensuring that malicious apps cannot access data from other apps or the operating system.

- Android employs Dalvik/ART (Android Runtime), which includes garbage collection to manage memory automatically.
- Memory allocation and deallocation are closely tied to app lifecycle events.
- Uses a combination of physical memory and Linux's virtual memory for efficient memory utilization.
- Swap space and zRAM (compressed memory) optimize memory for low-resource devices.
- **Memory Protection:** The Linux kernel enforces process isolation, preventing direct memory access between applications. Android further enhances security with app sandboxing, ensuring that malicious apps cannot access data from other apps or the operating system.

macOS:

- **Virtual Memory and Caching:** macOS employs a sophisticated virtual memory system that supports paging, compression, and caching. Memory compression in macOS optimizes the use of physical RAM, enhancing performance during heavy multitasking.

- **Memory Protection:** Security is a priority in macOS, with mechanisms like Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP) to protect against buffer overflow and code injection attacks. Memory pages are marked as read-only or executable to limit unauthorized access.

- macOS uses advanced virtual memory management, mapping logical addresses to physical memory.
- Incorporates a dynamic paging system and prioritizes apps actively in use.
- Memory compression reduces pressure on physical memory, allowing more efficient multitasking.

Comparison:

- Android's memory management is tailored for low-power devices, while macOS targets high-performance systems with abundant resources.
- Garbage collection in Android can introduce latency, whereas macOS's proactive memory compression ensures smoother performance.

---

4. File System

Android:

- Modern Android devices use the ext4 file system, known for robustness and journaling.
- File access is managed through standard Linux APIs.
- Scoped storage and sandboxing isolate app data for security.

macOS:

- macOS employs the Apple File System (APFS), designed for flash and solid-state drives.
- Features include encryption, cloning, and snapshots for efficient data handling.
- Metadata-rich file organization improves system reliability and performance.

Comparison:

- APFS is optimized for performance and security, while ext4's focus is on compatibility and resource efficiency.
- macOS supports advanced features like cloning and snapshots, which are absent in Android's ext4.

Difference Between APFS and ext4:

- APFS is optimized for SSDs, offering features like snapshots, copy-on-write, and native encryption, whereas ext4 is a general-purpose file system designed for traditional and modern storage media with basic journaling and file allocation.

- APFS performs better in handling large files and intensive I/O operations, making it ideal for macOS, while ext4 provides stability and compatibility for Android devices across diverse hardware.
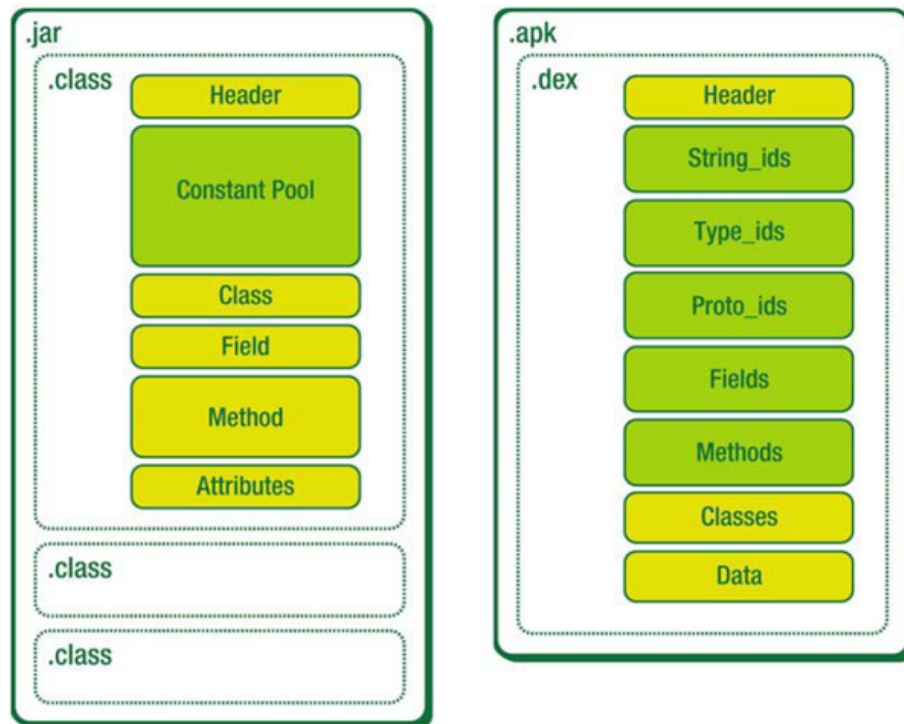

Figure 3-2. *Class file vs DEX file*

---

5. Security

Android:

- Relies on Linux kernel's security features, including SELinux for mandatory access control.
- Sandboxing isolates apps, limiting potential damage from malicious software.
- The Keystore system handles encryption and secure key storage.

macOS:

- Incorporates hardware security features like Secure Enclave for key management.
- Employs Gatekeeper, FileVault, and system integrity protection (SIP) to secure user data and system files.
- Extensive app notarization ensures only trusted apps run on macOS.

Comparison:

- macOS offers more comprehensive out-of-the-box security features, whereas Android's security model is reliant on app developers implementing best practices.
- macOS's Secure Enclave provides hardware-backed key management, offering an edge over Android's software-based Keystore.

6. Scheduling

Android:

- Android uses the Completely Fair Scheduler (CFS) from the Linux kernel, optimizing CPU usage based on fairness.
- Real-time tasks (e.g., audio) are handled using priority inheritance mechanisms.
- Battery and resource constraints influence scheduling decisions.

macOS:

- macOS employs a hybrid scheduler combining real-time and priority-based scheduling.
- Focuses on balancing user tasks and background processes for consistent performance.
- Thread groups and quality-of-service (QoS) classes enable better resource allocation.

Comparison:

- macOS's scheduling is tuned for high-performance hardware, while Android's is optimized for energy efficiency.
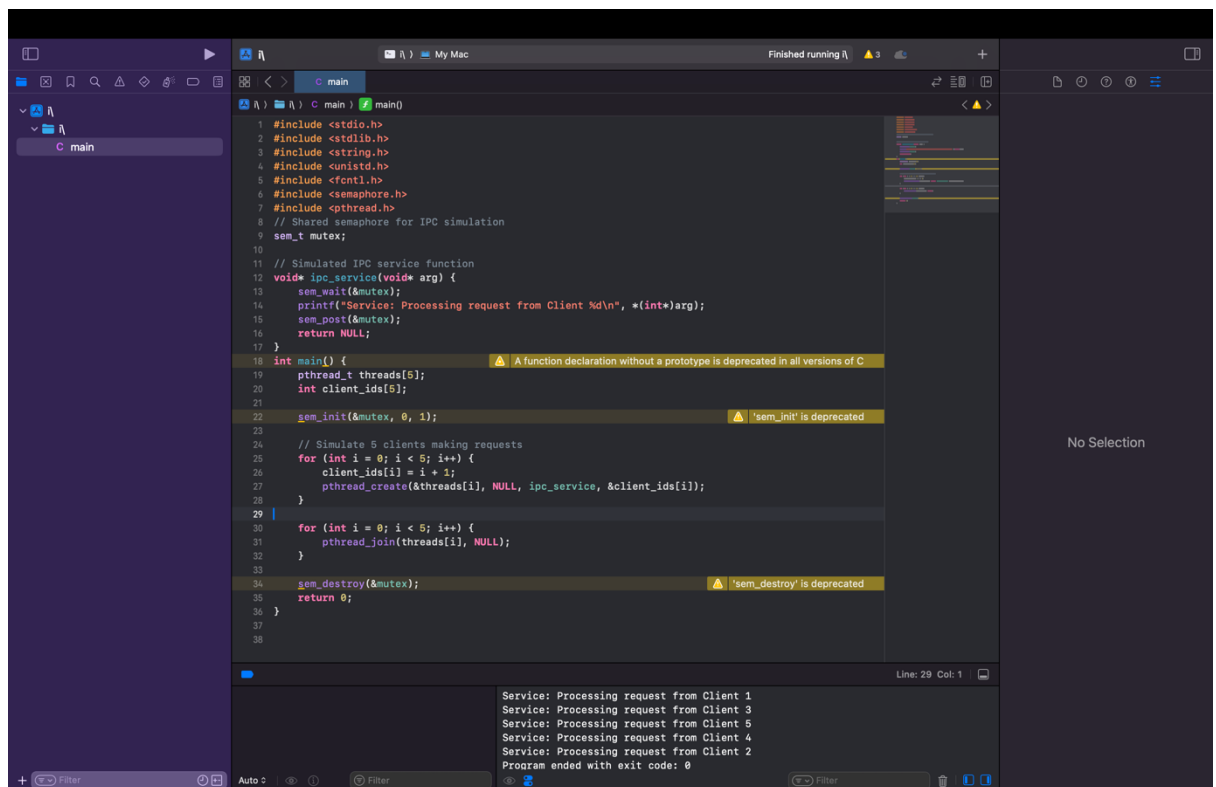
- Android's focus on real-time tasks ensures smooth mobile operations, whereas macOS balances desktop workloads effectively.

---

7. Creative Analogy:

Imagine Android as a compact, fuel-efficient car and macOS as a luxury SUV:

- **Android** prioritizes efficiency, ensuring tasks are completed with minimal resource usage. Its Binder framework is like a well-maintained, lightweight engine that ensures smooth rides for short distances.

  Android: Service with IPC

- **macOS**, on the other hand, is designed for power and comfort. Its XPC services resemble a powerful engine with advanced navigation systems, capable of handling long journeys and heavy loads without compromising comfort.
- Memory management in Android is akin to packing only essentials for a trip, while macOS's memory compression is like having expandable luggage for unexpected needs.

  macOS: File Encryption with Keychain Integration

8. Insights and Observations

- Android's architecture excels in resource-constrained environments, but macOS offers superior performance and security for high-end hardware.
- Both systems reflect their intended use cases: Android for lightweight, battery-efficient tasks and macOS for resource-intensive, high-performance workflows.
- The divergence in design philosophies underscores the importance of tailoring OS features to specific user needs.

9. Conclusion

This comparative analysis highlights how Android and macOS are designed to serve distinct ecosystems. Android's focus on efficiency and scalability contrasts with macOS's emphasis

on performance and security. Understanding these differences offers valuable insights into OS design principles and their implications for users and developers alike.

Additional Link
 For access to original articles and code examples, visit the