**Operating System**



Name Ali Abdullah

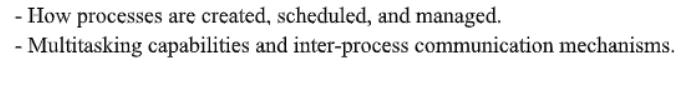
Registration ID 221376

Section BSCS-5C

Submitted to Mam Warda Aslam  
  
Department of Computer Sciences

Assignment # 03

**Comparative Analysis of Study of Android and MacOS:**  
  
**Process Management:**



Android:

In Android, processes are created using the **Zygote** process, which forks a copy of itself to launch new apps. This copy-on-write approach shares memory between parent and child processes until modifications are needed, reducing startup time and memory usage. Scheduling is managed by the Linux **Completely Fair Scheduler (CFS)**, which prioritizes foreground apps for better user experience while deprioritizing background processes. To conserve resources, Android terminates low-priority processes using a Least Recently Used (LRU) strategy when memory is low.

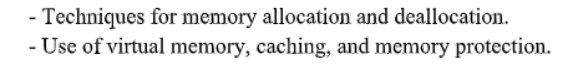
Android supports multitasking by prioritizing active apps while pausing or terminating background apps to conserve memory and battery. Communication between processes is handled by **Binder IPC**, a secure and efficient mechanism that allows apps to interact with system services or other apps, such as sending data to the location service. Binder also enforces strict permissions to enhance security.

MacOS:

MacOS uses the **fork-and-exec model**, where a new process duplicates an existing one and then loads a new program into memory. The **XNU kerne**l combines Mach’s task management with BSD’s(Barkeley Software Distribution) file system and networking support, enabling robust multitasking. MacOS employs a **priority-driven preemptive multitasking system**, dynamically adjusting process priorities to ensure smooth performance for resource-intensive tasks like animations or video editing.

In MacOS, multitasking focuses on high performance with processes running continuously, supported by features like **App Nap**, which reduces CPU usage for inactive apps to conserve energy. Inter-process communication is managed through **Mach messaging**, where processes exchange structured messages via uniqu e ports. This method enables seamless data sharing, such as in video editing software that communicates with rendering engines, while maintaining system stability and efficiency.

**Memory Management:**



Android:

In Android, memory allocation is managed through the **Android Runtime (ART)**, which employs garbage collection to handle memory deallocation. ART uses **generational garbage collection**, where objects are divided into young and old generations based on their lifespan. Short-lived objects are collected frequently, while long-lived objects remain in memory until necessary. This ensures efficient memory usage without interrupting app performance. Native code in Android, however, requires manual memory management, which developers handle using techniques like reference counting or explicitly freeing memory.

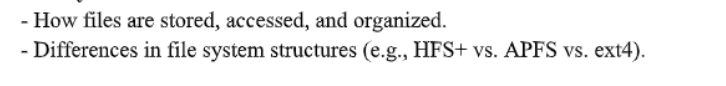
Android relies on the **Linux kernel’s virtual memory system**, which creates an abstraction layer allowing apps to use more memory than physically available. When physical memory is low, Android uses swap space on storage, though sparingly to reduce performance impacts. Android employs caching to improve performance, such as keeping frequently accessed data in memory for quicker retrieval. **Memory protection** in Android isolates each app within its own memory space, preventing apps from accessing or modifying another app’s data, a critical feature for security.

MacOS:

Memory allocation is managed by its **Virtual Memory Manager (VMM)**. This system dynamically allocates memory to applications and deallocates it when no longer needed. The VMM also leverages demand paging, where memory pages are loaded only when accessed, reducing overhead. macOS applications can use **memory-mapped files**, which map file contents directly into the application’s address space for efficient I/O operations. Memory deallocation is automated for most applications through macOS’s built-in memory management frameworks, such as Objective-C’s **Automatic Reference Counting (ARC)**.

MacOS also uses a sophisticated virtual memory system that employs **demand paging** to load only the necessary memory pages into RAM. When memory is insufficient, the system moves less-used data to the disk, creating the illusion of virtually unlimited memory. macOS further optimizes memory through its **Unified Memory Architecture (UMA)** on Apple Silicon, where the CPU and GPU share a single memory pool for faster access and reduced duplication. Strong memory protection mechanisms ensure that applications cannot access memory allocated to others, preserving system integrity and security.

**File Systems:**



Android:

Files are stored and managed using the **ext4 file system**, which is optimized for flash storage. Ext4 employs features like journaling, which logs file changes before they are applied, reducing corruption risks during unexpected shutdowns. File organization follows a hierarchical structure, with system-critical files stored in directories like /system, and user-accessible files in /sdcard. Access to files is controlled by strict permissions, with apps confined to their specific directories in /data unless additional permissions (like storage access) are granted. Android also supports secure storage mechanisms, such as **File-Based Encryption (FBE)**, where individual files are encrypted and decrypted based on user authentication.

The **ext4** file system used in Android is a journaling file system, which ensures data integrity by logging changes before committing them to the storage. It is lightweight and optimized for devices with limited storage resources. Ext4 supports features like large file support, extents (to reduce fragmentation), and delayed allocation for improved performance.

MacOS:

Files are managed using the **Apple File System (APFS)**, which was designed specifically for modern storage like SSDs. APFS uses advanced features like **snapshots**, which capture the state of the file system at a specific time, enabling quick recovery in case of failure. File access in macOS is fast and reliable, thanks to metadata-rich indexing and directory structures that facilitate quick searches. The system also integrates tightly with macOS features like **Time Machine**, which uses snapshots to create incremental backups. Organization in macOS supports a similar hierarchical structure, with /System, /Library, /Users, and /Applications directories for system, shared resources, user data, and installed applications.

On macOS, **APFS** is a modern file system that replaces the older **HFS+**. APFS is designed for high-speed SSDs, offering features like **space sharing**, where multiple volumes share the same physical storage dynamically. This allows macOS to optimize disk usage across different partitions. Unlike ext4, APFS supports **atomic-safe saves**, ensuring that files are never left in a partially written state. It also includes native encryption for file-level and full-disk security, which is more advanced than ext4’s encryption capabilities. HFS+, the predecessor to APFS, was primarily used for HDDs and lacked features like snapshots or space sharing, making it less efficient for modern SSD storage.

**Security:**





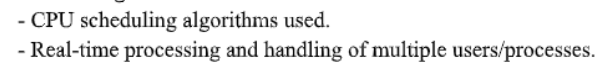
Android:

Android implements a multi-layered security approach designed to protect both the operating system and user data. A key mechanism is **Sandboxing**, which isolates each app in its own environment, preventing apps from accessing other apps' data or system-critical files unless explicitly permitted. Android also uses **SELinux (Security-Enhanced Linux)** at the kernel level to enforce mandatory access controls, limiting what processes and users can do on the system. Regular security updates and Google Play Protect further enhance security by scanning apps for vulnerabilities and ensuring they comply with security standards.  
  
In Android, **permissions** are central to user security. Apps must request explicit permissions to access sensitive data or hardware features, such as location or camera access. Permissions are user-granted at runtime, allowing for more granular control. Android uses **File-Based Encryption (FBE)** to secure individual files with unique keys, ensuring that unauthorized access is blocked. Additionally, the **Android Keystore System** provides a secure environment for storing cryptographic keys. Authentication mechanisms include PINs, passwords, patterns, and biometrics (like fingerprint or face recognition), ensuring user-friendly and secure access to devices and apps.

MacOS:

MacOS employs robust security mechanisms like **System Integrity Protection (SIP)**, which restricts even administrative users from modifying critical system files. SIP protects against malware and accidental changes to the core operating system. Another key feature is **Gatekeeper**, which ensures that only apps signed by identified developers or downloaded from the Mac App Store can be installed. macOS also includes **XProtect**, a built-in antivirus system that scans for known malware and automatically blocks it from executing.  
  
  
MacOS integrates **permissions** into its UNIX-based architecture, enforcing strict access controls for files and system resources. Apps are sandboxed, limiting their ability to interact with other processes or data. For encryption, macOS uses **FileVault**, a full-disk encryption feature that secures all user data on the system. APFS further enhances this by supporting native file-level encryption. Authentication in macOS is robust, with support for **Touch ID**, **Face ID** (on compatible devices), and secure passwords. Keychain, macOS’s password management system, securely stores credentials and encryption keys, ensuring that sensitive data is both accessible and protected.

**Scheduling:**



Android:

In Android, CPU scheduling is managed by the **Linux kernel’s Completely Fair Scheduler (CFS)**. CFS operates on the principle of equal CPU time distribution, ensuring fairness across all running tasks. It maintains a red-black tree data structure to track tasks and their respective priorities, allowing efficient selection of the next task to run. CFS prioritizes tasks based on their "niceness" value, which determines how much CPU time they get. Foreground applications typically receive higher priority to ensure smooth user experiences, while background apps are deprioritized to conserve resources like battery and memory. Additionally, Android uses **real-time scheduling policies** for critical tasks, such as audio or video playback, to prevent latency and maintain responsiveness.  
  
Android supports **real-time processing** for tasks that require strict timing guarantees, such as media playback or voice calls. For these tasks, Android uses **real-time scheduling policies**, including **SCHED\_FIFO** (First In, First Out) and **SCHED\_RR** (Round Robin), provided by the Linux kernel. These policies ensure that time-sensitive tasks are executed without interruption. Android also categorizes processes based on their importance to the user (e.g., foreground, visible, or background). Foreground processes are prioritized for CPU access, while background tasks are throttled or terminated if system resources are constrained, ensuring optimal performance for active apps.

MacOS

In macOS, CPU scheduling is handled by the **XNU kernel**, which uses a **priority-based preemptive scheduling algorithm**. Processes and threads are assigned dynamic priorities based on their current behavior and system needs. For instance, interactive tasks, such as UI rendering, are given higher priority than background processes like file indexing. macOS also supports **real-time threads**, which are assigned fixed high priorities for time-critical operations like audio processing or rendering animations. The kernel ensures that these threads preempt lower-priority ones, maintaining system responsiveness.  
  
In macOS, real-time processing is achieved through **Mach scheduling**. Mach supports multiple scheduling classes, including time-sharing and real-time, enabling the kernel to balance regular and critical tasks efficiently. For instance, macOS assigns real-time threads to tasks like audio rendering or video playback to ensure uninterrupted performance. When handling multiple processes, macOS uses a combination of fixed and dynamic priorities, adjusting priorities in real-time to ensure system responsiveness while maintaining fairness. macOS also supports multiple users, allocating CPU resources fairly across users logged into the system, ensuring a seamless multitasking experience.

**Articles/References:**

For MacOS:

Idris, M., Alhassan Idris Isma’il, Ibrahim, M., Abubakar, A. I., & Diginsa, M. U. (2022). A Comparative Study of Operating Systems: Case of Windows, Mac and Linux. In *Semi-Arid Journal of Academic Research and Development* (pp. 1–6). Binyaminu Usman Polytechnic, Hadejia. <https://www.researchgate.net/publication/369245267>

For Android:

Memory Management on Mobile Devices. (2024). *Proceedings of the 2024 ACM SIGPLAN International Symposium on Memory Management (ISMM ’24)*, 15. <https://doi.org/10.1145/3652024.3665510>

**THE END…………..^-^**