

**Operating Systems**

**Assignment#3**

**Submitted to: Ms. Warda Aslam**

**BSCS-V-C**

**Date: 27-12-2024**

**Shazil Jamil (221430)**

**---------------------------------------------------------------------------------------------------------------------------------------**

**This is a Report about a project which is based on three Numerical Analysis Techniques (Newton Divided Difference, Newton Forward Difference and Newton Backward Difference) which are used to estimate non-uniform values, values at start and value at end of dataset respectively.**

**Numerical** **Analysis** **and** **Computation**

**Comparative Analysis of iOS and Android Through Operating System Concepts**

**RESEARCH PAPERS USED:**

1. **Process Management in Android and iOS.**

**What It Covers**:Explains how Android and iOS handle processes, schedule tasks, and manage resources.

**Key Points**:

**Android**: Uses the Linux kernel with the **Completely Fair Scheduler (CFS)** for sharing CPU time fairly. Includes features like the **Zygote process** (to start apps faster) and a **Background Task Manager** to prioritize important tasks.

**iOS**: Focuses on stability and efficiency with its **XNU kernel**, using both fixed and flexible scheduling to ensure tasks run smoothly.

**Why It’s Useful**: Helps compare how Android and iOS handle multitasking, CPU scheduling, and process management.

1. **A Comparative Study of Operating Systems: Case of Windows, UNIX, Linux, Mac, Android, and iOS.**

**What It Covers**:Compares Android and iOS with other operating systems, focusing on features like file systems, security, and multitasking.

**Key Points**:

**Android**: Flexible and open-source, using the **ext4 file system**. Offers detailed permissions for apps but is more prone to security risks

**iOS**: Closed and controlled, with a stronger focus on security and using the **APFS file system** for fast, reliable storage.

**Why It’s Useful**: Great for understanding file storage, security, and general differences between Android and iOS.

1. **A Comprehensive Study of Kernel Issues in Different Operating Systems.**

**What It Covers**:Looks at the core (kernel) of Android and iOS, focusing on how the systems manage memory, schedule tasks, and keep data secure.

**Key Points**:

**Android**: Uses multicore processors to divide tasks efficiently. Handles memory with paging and ensures apps stay isolated for security.

**iOS**: Prioritizes security and stability with strict resource controls and sandboxing (keeping apps separate).

**Why It’s Useful**: Explains technical details about how Android and iOS handle tasks and memory at the core system level.

1. **Mobile Operating System: Analysis and Comparison of Android and iOS**

**What It Covers**:Compares Android and iOS, focusing on security features and app ecosystems.

**Key Points**:

**Android**: Offers flexible permissions and file encryption. Protects users with **Google Play Protect**, but its open nature can increase security risks.

**iOS**: Has stronger built-in security, like end-to-end encryption and the **Secure Enclave** (a chip for protecting sensitive data). Apps are carefully checked before being allowed on the App Store.

**Why It’s Useful**: Best for understanding how Android and iOS protect data and manage security.

**Comparison of Key OS Concepts of Android & IOS.**

**Process Management**

* **How processes are created, scheduled, and managed.**
* **Multitasking capabilities and inter-process communication mechanisms.**

**ANDROID**

1. **Process Creation:**

Android relies on the **Linux kernel** for process creation, using the **Zygote** process. The Zygote process serves as a template for app processes, minimizing memory overhead and speeding up process creation by **forking** preloaded system libraries and resources. This method ensures efficient use of system resources during app launches.

1. **Process Scheduling:**

Uses the **Completely Fair Scheduler (CFS)** from the Linux kernel to allocate CPU time among processes. CFS ensures fair distribution of CPU resources, balancing system responsiveness and performance.

1. **Process Management:**

Android categorizes processes into five states based on their priority and resource usage:

1. **Foreground:** Apps are actively interacting with the user.
2. **Visible**: Apps are visible to the user but not in focus.
3. **Service**: Background services are performing tasks without UI interaction.
4. **Background**: Apps are not visible but kept in memory for quick access.
5. **Empty**: Processes are cached for potential re-use.

Background processes are often terminated by the system to conserve memory and power.

1. **Multitasking Capabilities:**

Android supports robust multitasking with the help of the **Background Task Manager**. Manages background tasks by prioritizing foreground processes while limiting or terminating low-priority background processes to conserve resources. Developers can create services to run in the background, such as media playback or downloading data.

1. **IPC Mechanisms:**

Android uses the **Binder** mechanism for secure and efficient IPC. Binder allows components like Activities, Services, and Content Providers to communicate across processes while maintaining security. Other IPC mechanisms include **intents**, **messengers**, and **AIDL (Android Interface Definition Language)** for structured communication.

**iOS**

1. **Process Creation:**

iOS employs a static and dynamic state management system. Processes are tightly controlled to minimize resource usage and ensure security. Uses a **sandboxing mechanism** to isolate processes, ensuring security and preventing unauthorized memory access.

1. **Process Scheduling:**

iOS uses a hybrid scheduling approach, leveraging the **Mach kernel** to manage task prioritization and resource allocation efficiently. Tasks are scheduled based on priority, with higher emphasis on user-facing and time-sensitive processes.

1. **Process Management:**

iOS focuses on **state preservation and restoration**, allowing apps to save their state when not in use and resume seamlessly when reopened. Processes running in the background are restricted to specific tasks (e.g., audio playback or location updates) to maintain system performance and battery life.

1. **Multitasking Capabilities:**

iOS adopts a more restrictive multitasking approach. Background processes are limited to specific use cases, such as playing music, navigation, or completing uploads. It Implements features like **App Nap** to pause background apps and conserve energy.

1. **IPC Mechanisms:**

iOS employs **Mach IPC** as part of its Mach kernel for message passing between processes. Provides robust security through process isolation and sandboxing. Focuses on limiting IPC to maintain system stability and security.

**COMPARISON TABLE**

|  |  |  |
| --- | --- | --- |
| Features | Android | iOS |
| Process Creation | Uses Zygote process to make new apps quickly. | Manages processes tightly with sandboxing. |
| |  | | --- | | **Process Scheduling** |  |  | | --- | |  | | Fair scheduling with Linux CFS. | Prioritizes user tasks with Mach kernel. |
| Process Management | Five levels of importance, aggressive resource saving. | Saves app states, limits background tasks. |
| Multitasking | Flexible, with Background Task Manager. | Stricter, with App Nap for energy saving. |
| IPC Mechanisms | Binder, intents, AIDL. | |  | | --- | |  |  |  | | --- | | Mach IPC with strong security. | |

**Memory Management**

* **Techniques for memory allocation and deallocation.**
* **Use of virtual memory, caching, and memory protection.**

**ANDROID**

**1. Techniques for Memory Allocation and Deallocation**

Android uses the **Zygote process** for efficient **memory allocation**. Zygote serves as a pre-loaded template containing system libraries and resources. When a new app is launched, it creates a new process by "forking" the Zygote, saving memory and speeding up app launches.

**Memory deallocation** is handled by the **garbage collection** mechanism in the Android Runtime (ART) and Dalvik Virtual Machine (DVM). When an object is no longer in use, garbage collection automatically frees the memory for reuse.

**Kernel-level memory allocation** is managed by the **Linux kernel**, which divides memory into segments and allocates resources dynamically based on process priority.

**2. Use of Virtual Memory**

**Virtual memory** in Android is managed by **paging** and **memory mapping**. Unmodified memory-mapped files (such as code) can be paged out of RAM when the system needs to reclaim memory.

The **Linux kernel's virtual memory** subsystem uses **page tables** to map physical memory to virtual addresses, enabling efficient memory utilization and preventing fragmentation.

**3. Caching and Memory Protection**

Android employs **empty process caching** to keep inactive processes in memory for faster relaunch. These processes are retained until the system requires memory for higher-priority tasks, at which point they are terminated.

Memory protection is enforced through **sandboxing**, which isolates each app in its environment, preventing unauthorized access to other app data or system resources.

The kernel-level caching mechanism optimizes frequently accessed data to improve performance.

**iOS**

**1. Techniques for Memory Allocation and Deallocation**

iOS uses a combination of static and dynamic memory management techniques. **Apps allocate memory** for tasks, and the **system deallocates** it when the task is completed or when the app moves to the background.

**State preservation and restoration** are used to manage memory effectively. Background apps save their state and release unnecessary resources, reducing memory usage while maintaining a seamless user experience when reopened.

The **iOS kernel** ensures secure **memory allocation** through **sandboxing**, which isolates apps and prevents memory interference.

**2. Use of Virtual Memory**

iOS also uses virtual memory to handle resource-intensive tasks. Memory pages are swapped between RAM and disk storage as needed, ensuring efficient use of limited physical memory.

Virtual memory is tightly integrated with iOS’s **sandboxing** to maintain security and stability while isolating app memory spaces.

**3. Caching and Memory Protection**

iOS uses similar memory protection mechanisms through **sandboxing** to ensure app isolation and security.

The operating system minimizes memory usage for background apps by freezing their state and using caching strategies to optimize performance and reduce power consumption.

**COMPARISON TABLE**

|  |  |  |
| --- | --- | --- |
| Features | Android | iOS |
| Memory Allocation | Zygote process for fast app launches. | Dynamic and static allocation with state saving. |
| |  | | --- | | **Memory Deallocation** |  |  | | --- | |  | | Garbage collection via ART/DVM. | System-driven deallocation with sandboxing. |
| Virtual Memory | Paging and memory mapping (Linux kernel) | Integrated virtual memory for resource-intensive tasks. |
| Caching | Empty process caching for faster relaunch | Background app state freezing and caching. |
| Memory Protection | Sandboxing to isolate app environments | |  | | --- | |  |  |  | | --- | | Sandboxing with strict app isolation. | |

**File System**

* + - **How files are stored, accessed, and organized.**
    - **Differences in file system structures (e.g., HFS+ vs. APFS vs. ext4).**

**NOTE:** I am comparing Android vs iOS hence the file system structures compared are **ext4 vs APFS**.

**ANDROID (ext4)**

**1. File Storage**

Android uses the **ext4 file system**, which is a journaling file system designed for Linux-based systems.

ext4 stores files in blocks and uses **inodes** to maintain metadata about the files (e.g., size, permissions, timestamps).

Journaling ensures data integrity by keeping a record of changes to files before committing them to the disk, reducing the risk of data corruption during unexpected shutdowns.

**2. File Access**

File access is managed through the Linux kernel’s Virtual File System (VFS), which acts as an interface between user applications and physical storage.

**ext4** supports **delayed allocation**, which improves performance by deferring the allocation of storage blocks until data is written.

**3. File Organization**

Files are stored in hierarchical directories, with support for large file sizes and directories containing millions of entries.

**ext4** also provides support for **extents**, which are contiguous blocks that reduce fragmentation and speed up file access.

**iOS (APFS)**

**1. File Storage**

iOS employs the **Apple File System (APFS)**, which is optimized for flash and SSD storage commonly used in iPhones.

APFS uses a **copy-on-write (COW)** mechanism, which ensures that data is never overwritten. Instead, modifications are written to a new block, preserving the original data until the changes are finalized.

APFS supports encryption natively, allowing each file to have its own unique encryption key.

**2. File Access**

File access in iOS is tightly integrated with the **sandboxing** environment, ensuring apps can only access their specific directories unless explicitly granted permission.

The **COW** mechanism enhances data integrity and access speed by minimizing write operations to storage.

**3. File Organization**

Files in APFS are organized into containers, with each container having its volume. This allows dynamic resizing of volumes within the same container.

APFS supports **snapshots**, which are read-only copies of the file system at a given point in time, useful for backups and system recovery.

**COMPARISON TABLE**

|  |  |  |
| --- | --- | --- |
| Features | Android (ext4) | iOS (APFS) |
| Design Purpose | General-purpose journaling file system for Linux, adapted for mobile devices. | Specifically designed for flash and SSD storage in Apple devices. |
| |  | | --- | | **Data Integrity** |  |  | | --- | |  | | Journaling to log changes before committing them to disk. | Copy-on-write (COW) mechanism to ensure original data integrity during writes. |
| Performance | Delayed allocation and extents for reduced fragmentation and faster access. | Optimized for SSD with fast file access and reduced latency. |
| Encryption | |  | | --- | |  |  |  | | --- | | Ext4 can support encryption via external tools (not native). | | Native encryption with unique keys for each file or directory. |
| Snapshots | Not supported. | |  | | --- | |  |  |  | | --- | | Supports snapshots for backups and system recovery. | |
| File Organization | Hierarchical directories with support for large files and directories. | Container-based organization with dynamic volume resizing. |

**Security**

* **Mechanisms for ensuring system and user data security.**
* **Use of permissions, encryption, and authentication.**

**ANDROID**

**1. Mechanisms for Ensuring System and User Data Security**

**Sandboxing:** Each app runs in its isolated environment, ensuring that apps cannot access each other’s data unless explicitly allowed. This is achieved through Linux-based process isolation.

**Google Play Protect:** A built-in malware detection and prevention system scans apps installed from the Play Store for malicious behavior.

**Openness and Risks:** Android’s open ecosystem allows users to install apps from third-party sources. While this increases flexibility, it also raises the risk of malware, as apps outside the Play Store are not always thoroughly vetted.

**Frequent Updates:** Security patches are released regularly but are dependent on device manufacturers and carriers, causing delays in updates for many users.

**2. Use of Permissions**

**Granular Permissions:** Android employs a runtime permission model where users are prompted to grant permissions when an app attempts to access sensitive data or resources for the first time. Permissions are categorized as normal (granted by default) and dangerous (require explicit user approval).

**Custom ROM Risks:** Android’s openness allows the use of custom ROMs, which can bypass default security policies, posing a potential risk if not managed carefully.

**3. Use of Encryption:**

**File-Based Encryption (FBE):** Android uses FBE to encrypt individual files, allowing finer control over file access and decryption.

**Adoptable Storage:** External storage can be encrypted when formatted for use with a specific device, enhancing data security on removable media.

**Key Management:** Encryption keys are stored in a Trusted Execution Environment (TEE) for secure handling.

**4. Use of Authentication:**

**Biometric Authentication:** Android supports fingerprint and facial recognition, depending on the device’s hardware capabilities. These methods are integrated into the operating system through standardized APIs.

**Multi-Factor Authentication:** Google accounts, which are central to Android usage, support two-factor authentication (2FA) for added security.

**Custom Lock Screens:** Users can implement PINs, patterns, or passwords for device-level authentication, but the openness of Android allows custom lock screens, which could introduce vulnerabilities.

**iOS**

**1. Mechanisms for Ensuring System and User Data Security**

**Closed Ecosystem:** iOS apps are exclusively distributed through the App Store, where apps undergo strict vetting for compliance with Apple’s security guidelines.

**App Sandboxing:** Similar to Android, iOS ensures that apps are sandboxed. However, Apple's tighter control makes the sandboxing mechanism more robust, reducing the risk of unauthorized access.

**Rapid Updates:** Security updates are deployed directly by Apple, ensuring that all supported devices receive patches quickly.

**2. Use of Permissions**

**Tighter Permission Controls:** iOS permissions are strictly managed, with apps required to seek explicit approval for accessing sensitive features like location, camera, or microphone. Permissions are tightly coupled with sandboxing, ensuring apps cannot bypass restrictions through vulnerabilities.

**Limited Background Access:** Apps in iOS have more restricted access to background data and features compared to Android, reducing the likelihood of misuse.

**3. Use of Encryption:**

**End-to-End Encryption:** iOS offers default encryption for all data stored on the device and iCloud. Messages sent via iMessage and FaceTime are end-to-end encrypted.

**Hardware Security:** Encryption keys are tied to the Secure Enclave, a dedicated co-processor that ensures secure cryptographic operations and protects sensitive data.

**Unique File Encryption:** Each file is encrypted with a unique key, further enhancing data security.

**4. Use of Authentication:**

**Face ID and Touch ID:** Apple's biometric authentication systems are integrated with the Secure Enclave, ensuring that authentication data is never exposed outside the device.

**Passcodes:** iOS enforces strong passcodes and disables access after repeated failed attempts, adding a layer of brute-force protection.

**Apple ID Security:** Apple ID accounts support multi-factor authentication, ensuring secure access to the ecosystem.

**COMPARISON TABLE**

|  |  |  |
| --- | --- | --- |
| Aspect | Android | iOS |
| System Security | Open ecosystem, frequent updates (delayed). | Closed ecosystem, fast updates. |
| |  | | --- | | **Permissions** |  |  | | --- | |  | | Granular permissions, risks with custom ROMs. | Tighter controls, limited background access. |
| Encryption | File-based encryption, adoptable storage. | End-to-end encryption, Secure Enclave. |
| Authentication | |  | | --- | |  |  |  | | --- | | Biometric support, flexible lock screens. | | Face ID, Touch ID, stricter passcode policies. |

**Scheduling:**

* **CPU scheduling algorithms used.**
* **Real-time processing and handling of multiple users/processes.**

**ANDROID**

**1. CPU Scheduling Algorithms Used**

Android relies on the **Completely Fair Scheduler (CFS)**, part of its Linux kernel base, to manage CPU scheduling.

**CFS Characteristics**: Allocates CPU time based on a fair distribution model, ensuring no single process dominates the CPU. Maintains a balance between responsiveness and performance by assigning tasks dynamically according to priority and resource requirements.

**Task Groups**: **Foreground processes** are given higher priority, ensuring a smooth user experience for active apps. **Background tasks** are moved to a lower-priority group to prevent them from impacting system responsiveness.

**2. Real-Time Processing:**

Android incorporates **real-time scheduling policies** for critical tasks, such as handling UI threads and multimedia processing.

It uses thread prioritization, where foreground tasks are assigned higher "niceness" levels to ensure they receive adequate CPU time, preventing delays in real-time operations.

Real-time tasks leverage multicore processors through CPU affinity, allowing specific threads to run on designated cores for improved performance.

**3. Handling Multiple Processes:**

Android categorizes processes into different priority groups (foreground, visible, service, background, and empty). The **Background Task Manager** ensures that resources are allocated to critical processes while low-priority tasks are terminated or suspended as needed to conserve power and memory.

**iOS**

**1. CPU Scheduling Algorithms Used**

iOS uses a hybrid scheduling mechanism built on its **XNU kernel**, which combines features of the Mach microkernel and BSD kernel.

**Dynamic and Static Scheduling**:

Dynamically adjusts task priorities based on user interactions and app requirements. Includes static scheduling elements for predefined tasks, ensuring predictable system behaviour.

Focuses on efficiency and power conservation, particularly for mobile devices with limited battery resources.

**2. Real-Time Processing:**

iOS emphasizes real-time responsiveness for user-facing tasks through tight control of task scheduling.

The system dynamically adjusts resource allocation for active apps and background tasks, ensuring smooth performance even under heavy multitasking loads.

Uses energy-efficient techniques like **App Nap** to reduce resource usage for inactive tasks without impacting critical processes.

**3. Handling Multiple Processes:**

iOS maintains a strict policy for background execution, limiting the number of active processes to ensure system stability and conserve resources.

Background tasks are only allowed to perform specific operations, such as media playback or location updates, ensuring that system performance remains unaffected by non-essential processes.

**COMPARISON TABLE**

|  |  |  |
| --- | --- | --- |
| Feature | Android | iOS |
| CPU Scheduling Algorithm | |  | | --- | |  |  |  | | --- | | Completely Fair Scheduler (CFS). | | Hybrid scheduling (dynamic and static). |
| |  | | --- | | **Real-Time Processing** |  |  | | --- | |  | | Thread prioritization, multicore support. | Dynamic resource allocation for active tasks. |
| Multitasking | Flexible multitasking with background task groups. | Strict multitasking with limited background execution. |
| Power Efficiency | |  | | --- | |  |  |  | | --- | | Background Task Manager optimizes resources. | | App Nap reduces resource usage for inactive apps. |

**Creative Analogy and Explanation**

Think of Android and iOS as two kinds of **cars.**

**Android** is like a customizable sports car. You can pick the color, add custom parts, and tweak the engine. It gives you the freedom to make it your own and take it anywhere. But with all that freedom, you’re responsible for keeping it in good condition, and some custom parts (apps) might not always work perfectly together. Android’s openness, powered by features like **ext4 file system** and **Completely Fair Scheduler (CFS)**, makes it flexible but requires careful handling.

**iOS** is like a luxury sedan. It comes fully built, polished, and optimized. You can’t make many changes to how it looks or runs, but it’s reliable, smooth, and secure. You don’t have to worry about repairs or unexpected problems because everything is tightly controlled and designed to work perfectly. The **APFS file system** and **hybrid scheduling system** ensure smooth performance and stability.

**Flexibility**

**Android**: Android is ideal for people who like to explore and customize their devices. You can change settings, install custom apps, and even tweak the system, giving you a lot of control.

**iOS**: iOS, on the other hand, limits flexibility. It’s designed to work perfectly as it is, without allowing much customization. This makes it reliable but less adaptable to personal preferences.

**Stability**

**Android**: Android offers good stability, but its openness means that some apps or customizations might cause glitches if not handled properly.

**iOS**: iOS focuses heavily on stability. Its tightly controlled ecosystem ensures apps and updates run smoothly with fewer crashes or issues.

**Customization**

**Android**: Android is perfect for creative people who want to customize every aspect of their device. From widgets to custom ROMs, you can shape it to fit your needs.

**iOS**: iOS doesn’t offer much room for customization. You’re mostly limited to the options Apple provides, which ensures consistency but reduces personalization.

**Security**

**Android**: Android is more open, which can make it vulnerable if you’re not cautious. However, tools like **Google Play Protect** and permissions give you ways to secure your device.

**iOS**: iOS is like a gated community with strict security. Features like **sandboxing**, **end-to-end encryption**, and controlled app downloads make it extremely secure by default.

In the end, Android is great for people who like to customize and play around with their device, while iOS is perfect for those who want something simple and easy to use. Both are great choices, it just depends on what you prefer!

**Insights and personal observations on**

**the OS differences.**

1. **Flexibility vs. Control**:

Android feels more open and customizable. It gives users a lot of freedom to tweak settings, install apps from various sources, and even modify the system through custom ROMs. However, this openness comes with risks, like malware or unreliable updates from manufacturers.

iOS is the opposite; everything is tightly controlled. Apps can only be installed through the App Store, and Apple's strict guidelines keep the ecosystem secure. While this makes iOS safer, it can feel limiting, especially if you like personalizing your device.

1. **Performance and Resource Management:**

Android does a good job balancing performance across different devices using the **Completely Fair Scheduler (CFS)**. It makes sure no app hogs too much CPU time. But since Android runs on so many types of hardware, performance can vary.

iOS, on the other hand, is designed for Apple's specific hardware. This means it is super optimized and runs smoothly. The **hybrid scheduling system** in iOS ensures real-time responsiveness for apps while conserving battery life.

1. **Security:**

Android's security depends a lot on the user. If you stick to the Google Play Store and manage app permissions wisely, you're generally safe. Features like **Google Play Protect** and file encryption help, but Android's openness makes it easier for malware to sneak in.

iOS takes security very seriously. Its **sandboxing** and **end-to-end encryption** for iMessage and FaceTime make it one of the most secure platforms. Apple's closed ecosystem adds another layer of protection, but it can also feel restrictive.

1. **Multitasking:**

Android is great for multitasking. Features like **split-screen mode** and the **Background Task Manager** let you switch between apps seamlessly and keep more tasks running in the background.

iOS takes a stricter approach to multitasking. It limits what background apps can do to save battery life and keep the system stable. While this means fewer crashes, it can feel like you are forced to give up control.

1. **File Management:**

Android's **ext4 file system** makes it feel more like a mini-computer. You can move files between folders, use external storage, and even encrypt SD cards for added security.

iOS uses the **APFS file system**, which is optimized for speed and data integrity. It is great for casual users, but the lack of a traditional file system and limited external storage options can be frustrating if you are used to Android.

1. **Updates and Ecosystem:**

Android updates are inconsistent. Some devices get regular updates, while others lag behind because manufacturers control the rollout. This can leave some users stuck with outdated software.

iOS updates are quick and consistent across all supported devices. This makes iOS feel more reliable, especially when it comes to security patches.

**Final Thoughts**

**To put it shortly,**

Android is all about flexibility. Its **Completely Fair Scheduler (CFS)** handles multitasking well, and the **ext4 file system** makes file management easy. It’s great for customization, but its security depends on users being careful and getting timely updates.

**WHILE**

iOS focuses on stability and security. Its **hybrid scheduling** keeps performance smooth, and the **APFS file system** is fast and reliable. Features like **sandboxing** make it very secure, and updates are consistent. Android is best for those who want freedom, while iOS is ideal for those who value reliability and security.