Brayden Sawyer

Professor Chiu

CSCI 475

3 May 2024

Tech Report - iOS

Introduction

In the era of mobile technology, iOS stands as an orchestration of innovation and user experience. Launched in 2007 alongside the revolutionary iPhone, iOS has consistently pushed the boundaries of user experience, security, and functionality. iOS has undergone a transformative journey, evolving from a simple operating system for iPhones to a sophisticated platform that powers a wide array of Apple devices (Costello).

Its significance in the mobile operating system landscape is undeniable, offering a seamless, secure, and intuitive user interface that has set the standard for mobile computing. This essay aims to dissect the technical underworkings of iOS, focusing on its process and thread management, CPU scheduling policies, and memory management. These facets are crucial to iOS's operational excellence and its pivotal role in shaping the mobile industry.

Process and Thread Management in iOS

In iOS, a process is an instance of a running application, and each process can contain one or more threads. A thread is a path of execution within a process, and it's the smallest unit of processing that can be performed in an OS ("Processes and Threads"). iOS manages both processes and threads to ensure the efficient execution of applications. The main thread, also known as the UI thread, is where the user interface is processed, making it crucial for maintaining a responsive app (Apple Inc.).

Compared to standard process and thread management, iOS implements a more constrained environment. For example, iOS apps typically run a single process, and while they can spawn multiple threads, these threads share the same memory space. This is in contrast to some other operating systems that allow multiple instances of the same app to run as separate processes (Patil et al.).

CPU Scheduling Policies of iOS

iOS uses a run loop scheduling mechanism to manage the execution of code based on events and input sources, which is a distinctive approach compared to traditional thread scheduling. iOS's process and thread management include unique features such as the Grand Central Dispatch (GCD), which abstracts thread management away from the developer. GCD allows developers to define tasks and add them to dispatch queues.

There are three types of queues: Serial — to perform work sent to a queue in the order they are received, first in, first out. These are also known as private dispatch queues. Concurrent — to perform work sent to a queue at the same time. The order in which each task is started is the same order in which they were added to the queue. The main difference between Serial and Concurrent queues is that in Serial queues, the next task does not start until the first task ends, In Concurrent queues, the next task may not require as much time to complete as the first task and may end before the first task completes. Main dispatch queue — this is the application's main thread, or where your application lives.

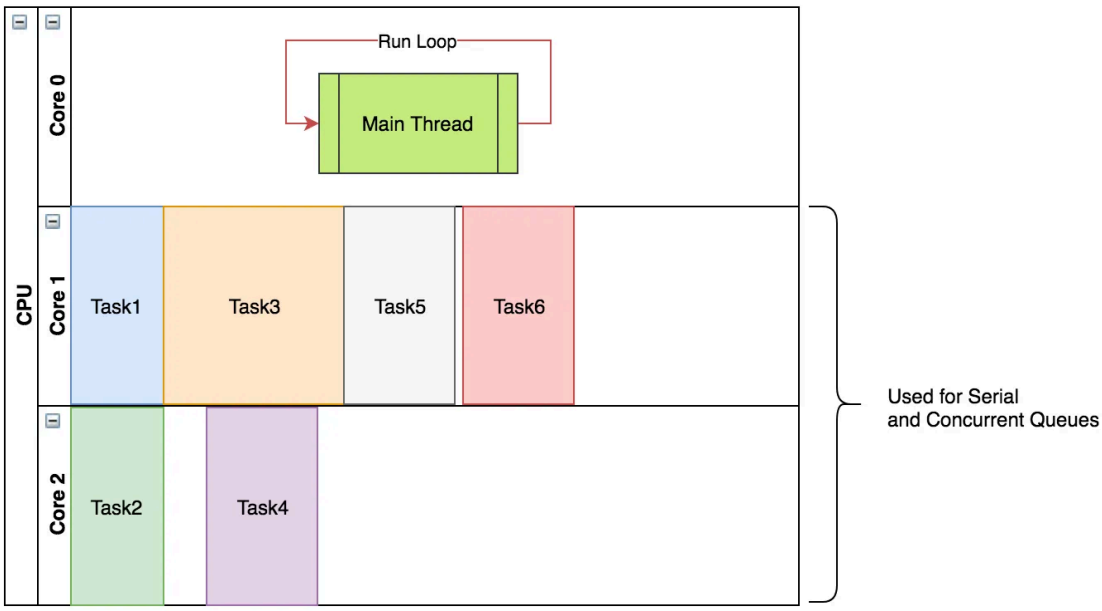

Figure 1(Roebling).

iOS's scheduler is particularly adept at handling multiple types of workloads, from interactive tasks that require immediate attention to background tasks that can be deferred. The system uses Quality of Service (QoS) classes to prioritize tasks. These classes range from user-interactive, which is given the highest priority, to background tasks, which have the lowest priority. This prioritization ensures that the system remains responsive to the user's needs while also managing the execution of less critical tasks in an energy-efficient manner. (Apple Inc.)

Moreover, iOS implements a cooperative multitasking environment, where applications are expected to voluntarily yield control of the CPU. This contrasts with the preemptive multitasking found in many other operating systems, where the system has more control over task execution.

Memory Management in iOS

Memory management is a pivotal aspect of iOS app performance. It involves the allocation and deallocation of memory resources to applications, ensuring smooth and efficient operation. In iOS, memory management is primarily handled through Automatic Reference Counting (ARC), which automates the tracking and handling of object memory usage.

ARC automatically frees up the memory for class instances when those instances are no longer needed.Every time you create a new instance of a class, ARC allocates a chunk of memory to store information for you. Inside of the memory, information about the type of the instance and the values of any stored properties associated with that instance are held.

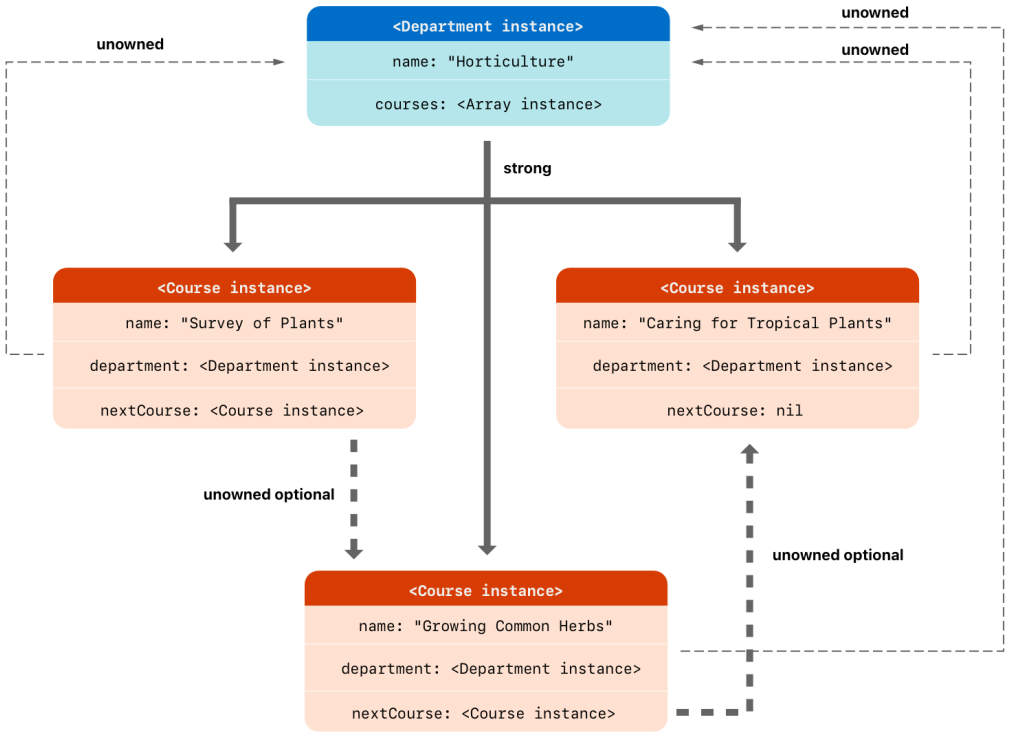

Figure 2 (Kodashima)

iOS implements virtual memory to create an abstraction layer between the physical memory and the application's memory requests. This system allows applications to use more memory than is physically available on the device by temporarily storing data on disk when not in use. Notably, iOS differs from other operating systems as it does not page out dirty memory to disk, which means it relies on compressing and decompressing memory blocks to manage the limited physical memory (Kelley).

The memory management decisions in iOS have a significant impact on both performance and security. Effective memory management policies contribute to the responsiveness and stability of iOS applications. The enforcement of memory protection and the implementation of virtual memory contribute to the robust security architecture of iOS, safeguarding against potential threats and ensuring that user data remains secure.

I/O devices in iOS

Input/Output (I/O) devices are integral to the iOS experience, allowing users to interact with their Apple devices in various intuitive ways. These devices range from the iPhone and iPad to the iPod touch, each running iOS software that facilitates user interaction through touchscreens, buttons, and sensors. iOS devices are equipped with a variety of I/O components, such as touchscreens, accelerometers, gyroscopes, and cameras. The touchscreen is perhaps the most prominent I/O device, providing a direct and tactile interface for user input. It supports multi-touch gestures, which are essential for the functionality of iOS apps and the operating system itself.

In iOS, the main thread is responsible for I/O operations, especially those tied to the user interface. It is the initial thread of execution in an iOS app, tasked with a loop that runs every frame to render the current screen, process UI events like touches, and execute tasks queued in dispatchQueue.main (Bhise). This thread operates with high priority, ensuring that any tasks assigned to it are executed promptly. This is precisely why UI-related code is expected to run on the main thread, as it maintains a responsive and fluid user interface.

However, it's crucial to balance the main thread's workload. Overburdening it with extensive tasks, such as synchronous networking or file operations, can impede its ability to swiftly handle screen rendering and UI updates ("Processes and Threads"). To avoid such performance bottlenecks, developers are advised to offload heavy I/O operations to the background thread (Varda).

The main thread's role is not just about execution but also about maintaining the quality of interaction that users have come to expect from iOS devices. By managing UI elements on the main thread, iOS ensures a seamless experience, contrasting with other operating systems where I/O management might lead to a less consistent user experience (Thapa).

Comparing iOS with other operating systems like Android, we find differences in how I/O devices are managed. While both platforms provide similar I/O capabilities, iOS is known for its streamlined and consistent user experience across all devices. In contrast, Android offers more customization options but can vary significantly between devices due to its open-source nature. iOS's approach to I/O is tightly integrated with its hardware, ensuring a seamless interaction that is often cited as being more user-friendly (Muchmore and Zamora).

Conclusion

In conclusion, this essay has investigated the intricate workings of iOS, highlighting its evolution as a leading mobile operating system and the technical intricacies that underscore its success. From the technical process and thread management to the sophisticated CPU scheduling policies, iOS demonstrates a commitment to performance and security. The memory management strategies, particularly the implementation of virtual memory, reflect a balance between efficiency and user experience. The exploration of I/O devices showcases iOS's user-centric design philosophy. iOS exhibits remarkable strengths, such as its seamless integration and intuitive interface.

The balance of efficiency and power makes iOS the champion of mobile computing. As technology advances, the future of iOS is poised to continue its trajectory of refinement and adaptation, shaping the landscape of mobile computing for years to come.

Bibliography

An, An. "How does the iOS app handle threads?" *Medium*, 4 October 2023, https://medium.com/@frentebw/how-does-the-ios-app-handle-threads-b677b4cf8a33.

Apple Inc. "Energy Efficiency Guide for iOS Apps: Prioritize Work with Quality of Service Classes." *Apple Developer*, 13 September 2016, https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/PrioritizeWorkWithQoS.html#//apple_ref/doc/uid/TP40015243-CH39-SW1.

Apple Inc. "Thread Management." *Apple Developer*, 15 July 2014, https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/Multithreading/CreatingThreads/CreatingThreads.html#//apple_ref/doc/uid/10000057i-CH15-SW2.

Bhise, Nikita. "Multithreading in iOS." 27 January 2023, https://www.linkedin.com/pulse/multithreading-ios-nikita-bhise/.

Costello, Sam. "The History of iOS, from Version 1.0 to 17.0." *Lifewire*, 5 June 2023, https://www.lifewire.com/ios-versions-4147730.

Kelley, Brian T. "Size Matters: An Exploration of Virtual Memory on iOS." *Always Processing*, 20 February 2022, https://alwaysprocessing.blog/2022/02/20/size-matters.

Kodashima, Kenta. "(iOS) ARC: Memory Management in iOS | by Kenta Kodashima | Medium." *Kenta Kodashima*, 29 September 2018, https://kentakodashima.medium.com/ios-arc-memory-management-in-ios-30aae3da92cf.

Muchmore, Michael, and Gabriel Zamora. “Android vs. iOS: Which Phone OS Really Is the Best?” *PCMag*, 15 November 2023, https://www.pcmag.com/comparisons/android-vs-ios-which-mobile-os-is-best.

Patil, Sankalp, et al. “Process Management In Android And IOS.” *International Journal of Engineering Research & Technology*, November 2023, https://www.ijert.org/research/process-management-in-android-and-ios-IJERTV12IS110230.pdf.

“Processes and Threads.” *Apple Developer*, https://developer.apple.com/documentation/foundation/processes_and_threads.

Roebling, Bob. “Beginning Swift Programming Part 11 — Grand Central Dispatch and Closures.” *Medium*, 15 April 2018, https://medium.com/swift2go/beginning-swift-programming-part-11-grand-central-dispatch-and-closures-293132b6a69d.

Salari, Mihail. “Boosting iOS App Performance: Memory Management, Core Data, and Multithreading Explained.” *Medium*, 27 July 2023, https://medium.com/@mihail_salari/boosting-ios-app-performance-memory-management-core-data-and-multithreading-explained-d6d328300930.

Thapa, Prayash. “Concurrency & Multithreading in iOS.” *Viget*, 25 February 2020, https://www.viget.com/articles/concurrency-multithreading-in-ios/.

“Thread.” *Apple Developer*, https://developer.apple.com/documentation/foundation/thread.

Varda, Filip. “iOS Threads and Memory Management.” 15 February 2022, https://q.agency/blog/ios-threads-and-memory-management/.

Varda, Filip. “iOS Threads and Memory management | by Filip Varda | Medium.” *Filip Varda*, 10 May 2021, https://filip-varda.medium.com/ios-threads-and-memory-management-b9c82d55b69a.