Keita Katsumi

Professor Gilbert Young

CS 4310

9th January 2024

Comparative Analysis of Process Scheduling Algorithms in Leading Operating Systems

The operating systems (OS) have a critical role as the crucial interface that ensures functionality between computer hardware and the user. They manage hardware resources, provide and adjust an environment where software runs efficiently, and handle user interactions and experience. Process scheduling algorithms are integral components of OS that consider system performance and user experience, especially in multitasking environments. They determine the order and duration for which processes access the CPU and ensure efficient utilization of the CPU and responsiveness of the system. Effective process scheduling is essential in affecting system reliability, efficiency, and the ability to handle simultaneous tasks. The evolution of process scheduling algorithms is a response to changes in hardware and diverse operating systems. As technology has grown with various user demands, the scheduling algorithms have been adapted by different operating systems.

This research aims to clarify major OS scheduling algorithms used by Windows, macOS, Linux, and Android, highlighting their unique approaches and adaptations. It investigates distinct process scheduling algorithms such as preemptive multitasking, cooperative multitasking, and the Completely Fair Scheduler and contextualizes their application in different OS environments. Each scheduling algorithm has unique characteristics that represent a different approach to managing CPU resources and process execution to optimize CPU performance. This paper describes how these algorithms are adapted to each operating system and clarifies why they are particularly suited to their respective environments. Understanding these algorithms is not only of academic interest but also gives future research topics in diverse areas, such as software development, system optimization, and emerging technologies in artificial intelligence and cloud computing.

Keita Katsumi

Professor Gilbert Young

CS 4310

9th January 2024

Windows technical director Alex Ionescu states the Windows OS adopts preemptive multitasking, which enables the operating system to take control of the CPU from one process and reassign it to another with dynamic task prioritization (Ionescu and Pavel 214). The prioritization, conducted by the OS's scheduler, preempts and organizes running processes along with time or priority base predefined criteria. In this system, a process's priority is dynamic and can be changed throughout its life cycle. Preemptive multitasking ensures that foreground applications receive adequate CPU time to enhance the user experience.

A critical requirement for a desktop operating system like Windows is responsiveness. Users expect immediate responses during their interactions with the OS and its applications. Preemptive multitasking facilitates realizing the users' demands by allowing Windows to alternate between processes swiftly. It ensures that applications in active use remain responsive, even when numerous background tasks are in operation. For instance, a conferencing application may be prioritized more during a video call. Preemptive multitasking makes the call stable and responsive, regardless of other ongoing background processes such as updates or scans.

A distinctive feature within Windows' multitasking framework is the concept of Quantum. Quantum refers to the predetermined duration a process can run before being preempted by the scheduler (Ionescu and Pavel 214). This short-time unit is determined by the scheduler's decisions to allocate CPU time and prevent a single process from occupying the system for an extended period. It resembles the round-robin time slice scheduling algorithm discussed in class where CPU time is distributed among processes in a balanced manner.

Through the implementation of preemptive multitasking, which is complemented by the quantum feature, Windows efficiently manages multiple processes. This system maintains high

Keita Katsumi

Professor Gilbert Young

CS 4310

9th January 2024

responsiveness and ensures a balanced experience for users. Integrating preemptive multitasking into Windows contributes to a responsive, stable, and secure environment for user interaction.

In contrast to Windows OS, macOS adapts a different approach to process scheduling, combining preemptive and cooperative multitasking aspects. Amit Singh, an OS researcher at Google, explains the difference between these two methods as follows: "In preemptive multitasking, the operating system can preempt one entity to run another, as needed. In cooperative multitasking, a running entity must give up control of the processor – cooperatively – to allow others to run" (Singh 771). Integrating preemptive and cooperative multitasking in macOS reflects Apple's commitment to flexibility and efficiency. It suits their specific user base and the diverse range of applications prevalent in the macOS environment.

The transition from the classic Mac OS, which predominantly utilized cooperative multitasking, to macOS marked Apple's strategic shift towards a more modern and efficient preemptive multitasking model (Singh 23). This shift aimed to balance robustness and user-centric flexibility, aligning with modern demands in video editing and music production requiring robust multitasking capabilities.

A key aspect of macOS is its use of the Mach kernel, known as its microkernel architecture. This architecture distinctively segregates core kernel functions from higher-level services, contributing to a modular and efficient design that is particularly beneficial for process scheduling. According to Singh, Apple historically integrated FreeBSD, known as Darwin, a collection of open-source technologies, to form a fundamental part of Mac OS X (Singh 47). FreeBSD is still the main component of recent macOS with advanced networking, performance, and compatibility features. The synthesis of the Mach kernel with FreeBSD enhances overall system performance and plays a

Keita Katsumi

Professor Gilbert Young

CS 4310

9th January 2024

vital role in energy efficiency. It integrates into Apple's diverse hardware range, from high-performance desktops to portable laptops.

The scheduling algorithm in macOS is designed for its unique operational environment, including broad aspects of hardware and user needs. The combination of preemptive and cooperative multitasking, along with the advanced architecture of the Mach kernel and FreeBSD features, enables macOS to provide users with a seamless, efficient, and responsive experience.

The Linux operating system employs a unique scheduling algorithm, distinctly different from both Windows and macOS, known as the Completely Fair Scheduler (CFS). Robert Love, a chief architect of Linux Desktop at Novell, describes CFS calculates how long a process should run as a function of the total number of runnable processes instead of adapting certain values from time slice calculation. CFS allows adequate value to weigh the proportion of the processors (Love 49). It utilizes a concept called 'virtual runtime,' which represents the expected runtime of a process on the CPU. This approach aims to equalize the virtual runtime across all processes and ensures equal distribution of CPU resources (Love 51). The design of CFS is highly scalable, making it efficient for a diverse range of systems, from small embedded devices to large multi-core servers.

According to Love, CFS picks the process with the smallest virtual runtime to optimize efficiencies, as we discussed the shortest job first in the class. CFS uses a red-black tree, a self-balancing binary search tree, to manage the list of runnable processes and efficiently find the smallest virtual runtime process (Love 53). This data structure enables the scheduler to swiftly identify the following process with the least virtual runtime, optimizing CPU allocation. Additionally, CFS considers the priority and nature of tasks, such as interactive versus batch processing tasks, adjusting the virtual runtime accordingly.

Keita Katsumi

Professor Gilbert Young

CS 4310

9th January 2024

In summary, CFS in Linux is a sophisticated scheduler that prioritizes fairness and efficiency in CPU time allocation. Its use of the red-black tree for task management and flexible scheduling options underscores Linux's adaptability, making it suitable for various system requirements, from personal desktops to high-end servers.

The Linux kernel has influenced the evolution of a new OS system. Android OS, built atop the Linux kernel, inherits the foundations of CFS. However, it distinctively evolves this scheduler to meet the unique demands of mobile devices. These custom modifications specifically address the constraints inherent in mobile environments, such as limited processing power, memory, and battery life. The scheduler's adaptations ensure that Android remains responsive and efficient, even with constrained hardware resources.

Android Kernel implements unique features to meet mobile environment limitations. Blake Meike, an engineer contributing to Android development at organizations, introduces Wakelocks and Alarms. The Wakelock is one of the special features required by Android to utilize the most important resource, the battery. It is a simple binary flag, and once the responsible application is done with useful work, it releases any wakelocks. If no other applications hold wakeloks, the kernel immediately puts the device to sleep (Schiefer and Meike 55). Android depends on the ability to wake itself up at a pre-scheduled time. It has an alarm driver that allows applicants to request that the kernel schedule a wakeup call at a specific time. The combination of an alarm driver and a wakelock decides the schedule when the process starts; otherwise, the application will be asleep (Schiefer and Meike 61).

The modifications to the CFS and Android's Kernel features enable Android's scheduling algorithm to finely tune for challenging modern mobile devices condition. It balances immediate

Keita Katsumi

Professor Gilbert Young

CS 4310

9th January 2024

responsiveness with the constraints of battery life and hardware limitations. Android's scheduler stands out as highly suitable for the mobile environment and addresses the challenges of limited resources, delivering an efficient and effective performance crucial for mobile computing.

Each operating system has utilized its process scheduling algorithm to suit its environment and user needs. Windows and macOS emphasize responsive user experiences with dynamic priority adjustments, Linux focuses on fairness and flexibility, and Android adapts Linux's approach for mobile-specific requirements, emphasizing power efficiency and responsiveness. In the era of big data, where data sets have grown exponentially, the efficiency of scheduling algorithms becomes increasingly crucial, particularly significant in fields like artificial intelligence and cloud computing, where data processing demands are continuously escalating. The research project leads me to understand the importance of scheduling algorithms with hands-on experience from another programming project. Coordinating theoretical and practical elements encourages me to conduct more detailed research about designing future scheduling algorithms.

Keita Katsumi

Professor Gilbert Young

CS 4310

9th January 2024

Works Cited

Ionescu, Alex, and Pavel yosifovich. *Windows Internals Seventh Edition Part 1*. Microsoft Press, 2017.

Love, Robert. *Linux Kernel Development*. Person Education, 2010.

Schiefer, Larry, and Blake Meike, *Inside the Andoroid OS: Builidng, Customizing, Managing and Operating Android System Services*. Person Education, 2022.

Singh, Amit. *Mac OS X Internals: A System Approach*. Person Education, 2007.