International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 8, pp 36-41, August-2026

https://doi.org/10.47001/IRJIET/2026.108004

Performance Analysis of Operating Systems and Scheduling Techniques

[1]Mahek Piyush Panchal, [2]Yashavi Manish Sanghani, [3]Krishna Kansara

[1,2,3]Information Technology, SVKM's Shri Bhagubhai Mafatlal Polytechnic & College of Engineering, Mumbai, India

E-mails: [1]mahekpanchal2510@gmail.com, [2]sanghani.yashavi@gmail.com, [3]krishna.kansara@sbmp.ac.in

***Abstract* - A modern computer system must handle many programs and requests for resources while providing acceptable performance and responsiveness. The operating system provides mechanisms to control processor time, memory, storage and other resources, providing an organized environment for executing applications [1], [2]. This review covers basic operating-system concepts and compares Windows, Linux and macOS with a focus on their system-management approaches [3]–[5]. It also considers CPU scheduling methods based on processor utilization, waiting time, responsiveness, fairness and workload suitability [1] [2] [7]. A comparison reveals different results of scheduling decisions according to the workload characteristics and the system objectives. Thus, considering the organization of the operating system along with scheduling strategies provides a broader view of how computing platforms handle the processing resources and the capability of supporting efficient program execution. Scheduling policies balance several performance objectives rather than maximizing one measure [1], [2].**

Keywords: Operating Systems, CPU Scheduling, Windows, Linux, macOS, Processor Utilization, Scheduling Algorithms, Waiting Time, System Performance, Resource Management, Workload Management, Process Scheduling.

I. INTRODUCTION

Computing systems are used in communication, information processing, business, education, and research. They operate on an operating system that manages interactions between application software and computer hardware [1], [2]. An operating system provides services for processor management, memory management, storage, and input/output operations, allowing programs to use system resources without directly controlling the underlying hardware [1], [2].

With the evolution of computing environments, several programs may be active at the same time and compete for processor capacity. Process scheduling is the solution to this problem, and involves choosing which tasks can run and how to share the processor's time between tasks [1], [2]. Such decisions may impact the waiting time, response time, throughput, processor utilization and fairness. Different scheduling policies pursue different objectives. Some want the ordering process to be simple while others want shorter wait times, more fairness or better responsiveness [1], [2], [7]. Hence, their effectiveness is a function of workload characteristics, rather than a single universal measure.

This paper is concerned with the relationship of operating-system design and processor scheduling. It discusses basic operating-system concepts, major operating-system categories, and the characteristics of Windows, Linux, and macOS. Classical scheduling methods are then examined and compared in relation to their operating principles, advantages, limitations, and suitable workload conditions [1]–[5], [7].Actual schedulers often combine multiple scheduling principles to suit different workloads [1]–[4].

II. LITERATURE REVIEW

The literature on operating systems offers a basis for comprehending scheduling, resource allocation, and process management. Tanenbaum and Bos situate these duties within the larger architecture of contemporary operating systems, but Silberschatz, Galvin, and Gagne address them as fundamental operating-system roles [1], [2]. The theoretical foundation for analyzing how processor resources are distributed across conflicting processes is provided by these publications.

Different rules are used by classical scheduling algorithms to decide processor access. Priority Scheduling employs assigned priority, Round Robin distributes CPU time thru periodic time intervals, SJF prioritizes shorter predicted CPU bursts, and FCFS adheres to queue order [1], [2], and [7]. These methods illustrate trade-offs between scheduling overhead, timeliness, waiting time, and fairness.

Platform documentation offers a useful viewpoint. Microsoft uses processor scheduling techniques and thread priorities to explain Windows scheduling [3]. While Apple documentation details the technologies enabling macOS [5], Linux documentation explains scheduling policies and their kernel implementation [4]. In order to compare scheduling concepts with real-world operating-system features, this evaluation integrates these theoretical and platform-level

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 8, pp 36-41, August-2026
https://doi.org/10.47001/IRJIET/2026.108004

viewpoints. Classical algorithms provide simplified models for understanding processor allocation [1], [2].

Author(s)	Year	Research Focus	Key Contribution	Limitation
Abraham Silberschatz et al.	2018	Operating System Concepts	Explained process management, memory management, scheduling, and resource allocation.	Focused on theoretical concepts rather than platform comparison.
William Stallings	2018	Operating Systems: Internals and Design Principles	Described scheduling algorithms and operating system architecture.	Limited comparison of modern desktop operating systems.
Andrew S. Tanenbaum & Herbert Bos	2015	Modern Operating Systems	Discussed operating system design and process scheduling techniques.	Primarily emphasized system architecture instead of practical comparison.
Microsoft Documentation	2024	Windows Architecture	Explained Windows process management and scheduling mechanisms.	Restricted to the Windows platform.
Linux Kernel Documentation	2024	Linux Scheduler	Described the Completely Fair Scheduler (CFS) and resource management.	Limited to Linux implementation details.
Apple Developer Documentation	2024	macOS Architecture	Explained process scheduling and system resource handling in macOS.	Focused only on Apple's ecosystem.

1) About Operating System

The operating system [1] represents the software layer that allows apps to run on the computer. It ensures the communication process between programs and hardware, thus eliminating the need for each application to control the computer's physical part [1], [2], [6]. The OS manages the CPU, memory, files, and input/output operations [1], [2]. In addition, the operating system now allows for distributed computing, networking and virtualization [2], [4]. In general, the OS creates a controlled environment in which various applications can operate without direct access to hardware [1], [2].

Function	Purpose
Process Management	Schedules and controls the execution of processes.
Memory Management	Allocates and monitors main memory efficiently.
File Management	Organizes, stores, and retrieves files.
Device Management	Coordinates communication with hardware devices.
Security Management	Protects system resources through authentication and access control.
User Interface	Provides graphical or command-line interaction with the system.

2) Types of Operating Systems

Operating systems [1] can be categorized based on how they handle workloads, allocate resources, and facilitate user communication. Various environments prioritize resource sharing, predictable execution, interactivity, or throughput [1], [2], [8].

a) Batch Operating System: During execution, jobs are gathered and handled in batches with little to no interaction. When doing repetitious tasks is more crucial than getting quick user input, this strategy works well [1], [8].
b) Time-Sharing Operating System: Short execution intervals are used to distribute processor time across running tasks. This preserves interactive response while enabling many users or programs to advance [1], [2].
c) Real-Time Operating Systems: Within predetermined time limitations, these systems prioritize predictable replies. When fulfilling execution dates is a crucial system requirement, they are suitable [1], [2].
d) Distributed Operating System: To enable shared processing and resource utilization, resources from linked machines are coordinated. When computational tasks are spread across multiple machines, this model can be helpful [1], [2]. Computer expertise [1][2]. Each operating-system category emphasizes a different execution requirement [1], [2].

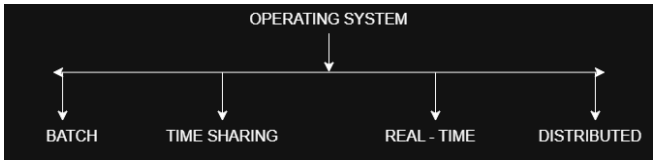

Figure 1: Types of Operating Systems [1][2]

III. COMPARISON OF WINDOWS, LINUX, AND MACOS

While Windows, Linux, and macOS all offer essential operating system functions, they differ in terms of program environments, hardware interactions, configurability, and system management strategies [3]–[5]. Therefore, the computer environment and application needs determine which platform is best.

Windows is made to work with a wide range of hardware and software. When necessary, its scheduling mechanism can preempt lower-priority tasks by using thread priorities to affect processor allocation [3]. Windows combines thread priority with time-based processor sharing [3].

Operating System	Primary Objective	Typical Applications
Batch	Executes grouped jobs with minimal user interaction	Payroll processing, billing systems
Time-Sharing	Supports multiple users through shared CPU time	Multi-user servers, university computer labs
Real-Time	Delivers responses within strict time limits	Medical equipment, robotics, air traffic control
Distributed	Shares resources across interconnected computers	Cloud computing, distributed databases, data centers

Linux offers an adaptable operating system where the kernel and related components can be set up to meet various needs. Multiple scheduling strategies are supported by the Linux kernel for real-time and general-purpose workloads [4]. Linux supports different scheduling policies for different workload requirements [4].

Because macOS is made for Apple hardware, the operating system and hardware can be developed together [5]. macOS uses priority and Quality of Service information to classify application work [5], [10]. Within the larger Apple ecosystem, its system services oversee processor and other hardware resources.

According to the comparison, macOS gains from controlled hardware-software integration, Windows prioritizes wide compatibility, while Linux provides significant configurability [3]–[5]. These features affect each platform's suitability for various workloads and user needs [1]–[5].

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 8, pp 36-41, August-2026

https://doi.org/10.47001/IRJIET/2026.108004

Feature	Windows	Linux	macOS
Developer	Microsoft	Open-source Community	Apple Inc.
License	Proprietary	Open Source	Proprietary
Kernel	Windows NT	Linux Kernel	XNU
Hardware Support	Wide	Wide	Apple Devices Only
Customization	Moderate	Very High	Limited
Security	High	Very High	High
Software Availability	Extensive	Moderate	Apple-focused
Typical Use	Home, Office, Gaming	Servers, Development, Cloud	Creative Work, Apple Ecosystem

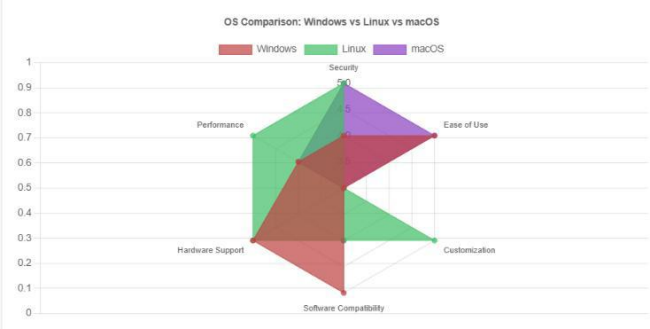

Figure 2: Radar chart: OS comparison [3][4][5]

IV. PROCESS SCHEDULING

When many tasks need processor access, the operating system uses process scheduling to choose a runnable process or thread. The scheduler chooses which task gets processor time and, based on the rules, when another task should run because CPU capacity is shared [1], [2].

Processor usage, throughput, reaction time, waiting time, and fairness can all be used to assess scheduling performance [1], [2], [7]. These strategies may be at odds with one another: prioritizing short activities may cause longer ones to be delayed, while increasing responsiveness may result in higher context-switching overhead. Therefore, scheduling is not an effort to maximize one statistic but rather a trade-off.

Workload characteristics, execution requirements, and system objectives determine the best policy. While practical operating systems employ more specific processes, classical algorithms offer helpful models for comprehending these trade-offs [1]–[4]. Scheduling decisions directly influence how competing tasks share CPU capacity [1], [2].

Objective	Description
CPU Utilization	Keeps the processor active for the maximum possible time.
Throughput	Increases the number of processes completed in a given period.
Turnaround Time	Reduces the total time required to complete a process.
Waiting Time	Minimizes the time a process spends in the ready queue.
Response Time	Improves the speed at which a process receives its first CPU allocation.
Fairness	Ensures equitable CPU allocation among competing processes.

Figure 3: Process State Transition Diagram [1],[2]

V. SCHEDULING ALGORITHMS

A. Windows

In Windows, the priority-based thread scheduling model is used. Therefore, scheduling priorities determine how threads gain access to the processor [3]. Moreover, threads that are at the same level may divide the processor time via time slices [3]. The scheduling algorithms below are presented as comparison models, which do not necessarily reflect how the operating system actually works.

1) First-Come, First-Served (FCFS)

FCFS schedules processes in the order of their arrival [2] in the ready queue. The process that has arrived first [1] is selected for execution before the other processes that have arrived later (1),(2). Once selected, the process is executed until it terminates or until it is interrupted, making the FCFS scheduling algorithm non-preemptive (1),(2).

Strengths: The FCFS algorithm was the earliest algorithm and has the advantage of being simple to understand and implement. In addition, the algorithm maintains the order of arriving processes and makes fewer scheduling decisions (1), (2).

Weaknesses: On the other hand, FCFS has weaknesses in terms of the average waiting time. Thus, in the case of a long running process, many short processes will have to wait in the queue, which will lead to increased waiting time, as well as poor interactive response (1), (2), (7).

Applicable: FCFS is best suited for batch processing applications where there is no need for immediate execution of processes (1), (2).

2) Shortest Job First (SJF)

SJF [1] is the algorithm which selects the process with the shortest expected CPU burst for execution. The algorithm has the ability to reduce the average waiting time [2] of the

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 8, pp 36-41, August-2026
https://doi.org/10.47001/IRJIET/2026.108004

processes when the burst-time [2] estimation is accurate [1], [2].

Strengths: It can provide low average waiting time and make it possible for short processes to complete quickly [1], [2].

Weaknesses: Actual CPU burst is not always known and it may be challenging to estimate it accurately and processes with longer burst time may experience starvation when processes with a shorter burst time are constantly arriving [1], [2].

Applicable to: the algorithm can be applied to the set of processes which have a predictable burst time [1], [2].

3) Priority Scheduling

The Priority scheduling [7] method assigns different priorities to all processes and then selects the one with the highest priority [2] for execution [1], [2], [7].

Strengths: can be used to ensure a certain degree of priority for specific processes, and can be utilized to meet deadliness requirements.

Weaknesses: processes with low priority may starve, and the effectiveness highly depends on the accuracy of assigning priorities [1], [2], [7].

Applicable to: environments, which involve processes with different levels of priority.

4) Round Robin (RR)

RR method uses the same time interval or quantum for all processes. As soon as a process completes its time slice [2], it is placed back to the end of the ready queue [1], [2], [7].

Strengths: processes are treated equally and good for interactive and responsive systems.

Weaknesses: context switching may take too much time, and performance greatly depends on the chosen interval [1], [2], [7].

Applicable to: interactive and other systems, which require fast response times.

B. Linux

Linux provides a variety of scheduling policies for ordinary and real-time tasks [4]. As a result, the scheduling characteristics could be differentiated depending on the chosen policies.

1) Completely Fair Scheduler (CFS)

CFS is designed to ensure fair distribution of processor time between active tasks, considering the virtual runtime as a key scheduling metric [1], [2], [4].

Pros: Fair allocation of processing time and good for general multitasking. Cons: More complex than some of the basic scheduling policies [1], [4].

2) SCHED_FIFO

SCHED_FIFO is the real-time policy that enables a runnable real-time task to resume execution until its completion, blocking, or being preempted by a higher-priority task [4].

Pros: Predictable priority and quick access to high-priority tasks. Cons: Starvation of low-priority tasks if high-priority ones are constantly running [4].

3) SCHED_RR

SCHED_RR is similar to SCHED_FIFO but offers time-sharing capabilities for real-time tasks, assuming that tasks with the same priority yield the processor to each other based on a defined time quantum [4].

Pros: Shares CPU between same-priority tasks and good for time-sharing. Cons: Extra context switches and sensitive to the time-quantum value [4].

4) EEVDF

EEVDF is the extension of the Linux fair-scheduling design that incorporates eligibility criteria and virtual deadlines in the scheduling decisions about which task should receive the processor [4].

Pros: Addresses time-sharing and fairness aspects and offers decent responsiveness for eligible tasks. Cons: More complicated than simple scheduling algorithms and depends on the implementation in the kernel [4].

C. MACOS

macOS operating system uses the XNU kernel, so it includes scheduling strategies for threads with priorities and different requirements [5]. In this regard, it should be noted that the macOS scheduler is by no means a textbook algorithm; instead, it should be viewed as an integral part of the operating system's overall design.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 8, pp 36-41, August-2026

https://doi.org/10.47001/IRJIET/2026.108004

1) Timeshare Scheduler

The timeshare scheduler is designed for regular processes, with CPU time being sliced and shared between threads based on scheduling priorities [5].

Pros: can handle interactive processes and concurrent job execution. Cons: has a more complicated algorithm than the FCFS scheduler [5].

2) Fixed-Priority Scheduler

The fixed-priority scheduler ensures that threads with higher priorities are executed before those with lower priorities [5].

Pros: predictability. Cons: low-priority threads can be indefinitely postponed [5].

3) Real-Time Scheduler

The real-time scheduler is designed for processes that have higher priority than regular applications [5].

Pros: can handle time-critical tasks. Cons: CPU allocation to real-time tasks affects other processes [5].

4) Priority-Based Preemptive Scheduler

This scheduler allows for CPU allocation to be preempted, which means that a running thread can be interrupted and moved to the ready queue if a higher-priority thread needs CPU time. In other words, preemptive scheduling can ensure that more critical tasks get CPU time when needed [5].

Pros: quick response to important tasks; allows for concurrency. Cons: context-switching penalties; requires priority management [5].

VI. COMPARATIVE ANALYSIS OF SCHEDULING ALGORITHMS

Aspect	FCFS	SJF	Priority	Round Robin
Scheduling Basis	Arrival order	Shortest burst time	Process priority	Fixed time quantum
Implementation Complexity	Low	Moderate	Moderate	Moderate
Response Time	Low	Moderate	High	Very High
Fairness	Moderate	Low	Low	High
CPU Utilization	Moderate	High	High	Moderate
Starvation Possibility	No	Yes	Yes	No
Suitable Environment	Batch Systems	Batch Processing	Real-Time Systems	Time-Sharing Systems

The reviewed algorithms are characterized by various scheduling criteria [1], [2]. Such criteria as the first come first served are simple to implement as they only require the processes to be executed in the order they arrive [1], [2].

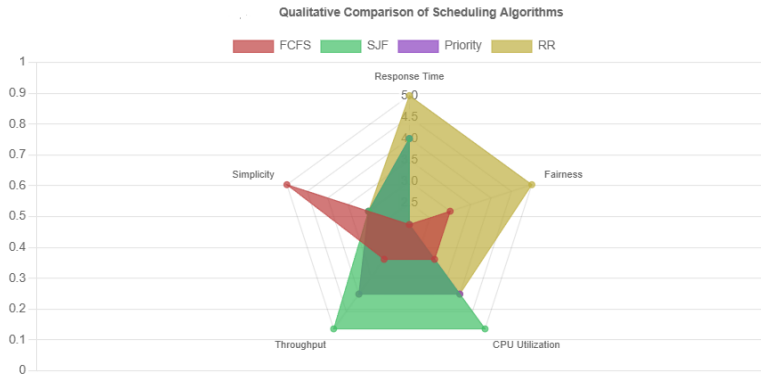

Figure 4: Radar chart: Qualitative Comparison [1][2][7]

The Shortest Job First is efficient as it minimizes the waiting time of the processes assuming that the burst time of the processes is predicted correctly [1], [2]. Priority Scheduling is based on the assignment of priorities to the processes and the relative power of the processes determines which one is executed first [1], [2]. The Round Robin method provides a time quantum which is allocated to each ready process in a circular manner to ensure that each process gets a fair share of the CPU [1], [2], [7]. Thus, each algorithm has its own advantages and disadvantages which should be considered when selecting the most appropriate scheduling method for a particular workload [1], [2].

Modern operating systems often implement various CPU scheduling policies for different workloads [3], [4]. For example, the Windows OS applies priority-based scheduling with time slicing while Linux offers multiple policies designed for both normal and real-time workloads [3], [4], [7]. Therefore, a comprehensive analysis of algorithms should include such performance metrics as waiting time, turnaround time, response time, throughput, CPU utilization, and fairness [1], [2]. It is essential to highlight that an algorithm which may be optimal for some criteria might demonstrate inferior performance for other workloads [1], [2]. Hence, the best scheduling method cannot be selected for all workloads.

VII. CONCLUSION

The reviewed scheduling algorithms used in Windows, Linux, and macOS operating systems are different in terms of resource management and task scheduling [3] [5]. The algorithms also differ concerning the waiting time, fairness, response, and CPU utilization [1] [2] [7]. However, the best choice depends on the workload and requirements rather than a universal approach applicable to all systems. It is vital to mention that future operating systems will still involve scheduling algorithms that can effectively adapt to hardware and workload [1] [2] [4]. Platform-specific scheduling reflects the different requirements of modern workloads [3]–[5].

ISSN (online): 2581-3048
Volume 10, Issue 8, pp 36-41, August-2026
https://doi.org/10.47001/IRJIET/2026.108004

REFERNCES

[1] A.Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts, *10th ed. Hoboken, NJ, USA: John Wiley & Sons*, 2018.

[2] A.S. Tanenbaum and H. Bos, Modern Operating Systems, *5th ed. Harlow, U.K.: Pearson*, 2022.

[3] Microsoft, "Windows Documentation." [Online]. Available: https://learn.microsoft.com/windows. [Accessed: Aug. 5, 2026].

[4] The Linux Kernel Organization, "Linux Kernel Documentation." [Online]. Available: https://docs.kernel.org/. [Accessed: Aug. 5, 2026].

[5] Apple Inc., "Apple Developer Documentation." [Online]. Available: https://developer.apple.com/documentation/. [Accessed: Aug. 5, 2026].

[6] IBM, "Operating System." [Online]. Available: https://www.ibm.com/think/topics/operating-system. [Accessed: Aug. 5, 2026].

[7] Geeks for Geeks, "CPU Scheduling in Operating Systems." [Online]. Available: https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/. [Accessed: Aug. 5, 2026].

[8] Tutorials Point, "Operating System Tutorial." [Online]. Available: https://www.tutorialspoint.com/operating_system/index.htm. [Accessed: Aug. 5, 2026].

Citation of this Article:

Mahek Piyush Panchal, Yashavi Manish Sanghani, & Krishna Kansara. (2026). Performance Analysis of Operating Systems and Scheduling Techniques. *International Research Journal of Innovations in Engineering and Technology - IRJIET*, 10(8), 36-41. Article DOI https://doi.org/10.47001/IRJIET/2026.108004

****

