iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.75865

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

A Comparative Study of CPU Scheduling Algorithms for Efficient Process Management

Khushboo
B. Tech CSE, Pre-Final Year Student, School of Computer Science and Engineering, Lovely Professional University, Phagwara, India

Abstract: ***CPU scheduling is a fundamental component of operating system design, ensuring that computational resources are allocated optimally among multiple processes. This paper examines the concepts and methods used in CPU scheduling, focusing on major algorithms like First-Come, First-Served (FCFS), Shortest Job Next (SJN), Round Robin (RR), and Priority Scheduling. Through a detailed comparison of performance metrics including waiting time, turnaround time, and CPU utilization, we examine how these algorithms perform under varying system loads and requirements. The study reveals the trade-offs between simplicity and efficiency, highlighting the advantages of pre-emptive methods in dynamic environments and the challenges faced by non-pre-emptive approaches. Finally, the paper discusses the potential of hybrid and adaptive scheduling techniques, emphasizing their role in meeting the demands of modern, multi-core systems. These results offer meaningful insights into improving CPU efficiency in modern operating systems.***
Keywords: CPU Scheduling, Operating Systems, Round Robin (RR), Priority Scheduling, Pre-emptive Scheduling.

I. INTRODUCTION

In a single-processor environment, only one process can execute at a time, while others must wait until the CPU becomes available again. The main goal of multiprogramming is to ensure that the CPU remains active as much as possible, thereby increasing its overall utilization. The concept is straightforward when a process is forced to pause, usually due to waiting for an I/O operation to finish, the CPU would otherwise remain idle and unproductive.

To make better use of this waiting time, multiprogramming allows multiple processes to reside in memory simultaneously. When one process is waiting, the operating system quickly switches the CPU to another process that is ready to execute. This cycle continues repeatedly, ensuring that the CPU is efficiently used. Such scheduling of processes is one of the core responsibilities of an operating system. Since nearly all system resources are allocated and managed through scheduling, CPU scheduling plays a vital role in the performance and design of operating systems.

In modern computing, the central processing unit (CPU) is one of the most critical resources in a system, and its effective utilization directly impacts overall performance. CPU scheduling, a fundamental function of operating systems, ensures that multiple processes can run efficiently by determining the execution order of tasks in a way that optimizes resource usage and system responsiveness. Scheduling decisions influence key performance metrics such as waiting time, turnaround time, throughput, and CPU utilization, making it a vital area of study in computer science.

Over time, several CPU scheduling algorithms have been designed and refined to meet the diverse requirements of computing systems.

For example, algorithms like First-Come, First-Served (FCFS) prioritize simplicity, while Shortest Job Next (SJN) minimizes waiting time for processes with shorter execution durations. Preemptive techniques, such as Shortest Remaining Time First (SRTF) and Round Robin (RR), aim to improve responsiveness and fairness in time-sharing systems, while Priority Scheduling allows processes with higher importance to be executed sooner. Despite their individual advantages, these algorithms come with inherent trade-offs, such as potential starvation, context-switch overhead, or inefficiencies in handling diverse workloads.

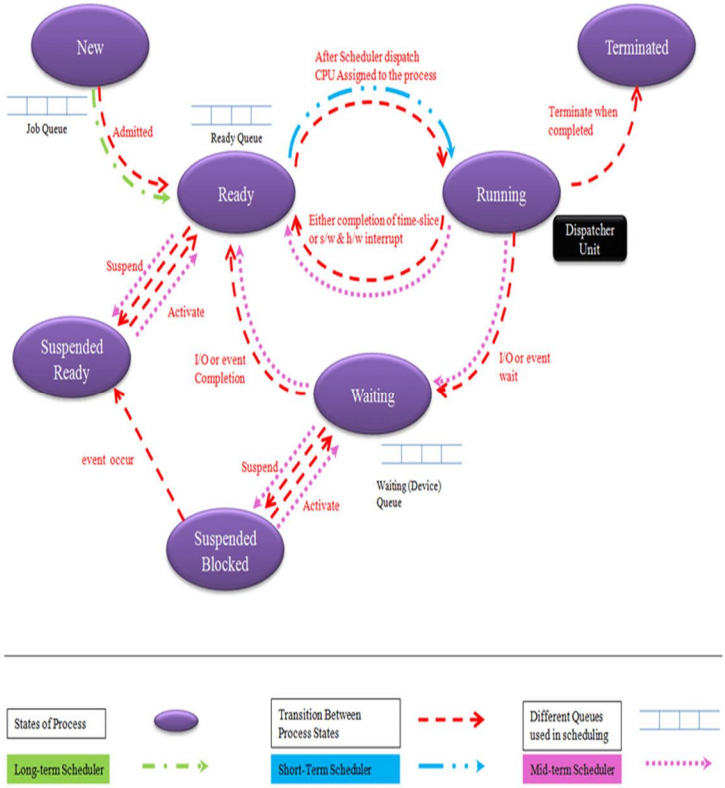
As computing systems evolve, so do the challenges associated with process scheduling. With the advent of multi-core processors, real-time applications, and dynamic workloads, there is a growing need for hybrid and adaptive scheduling strategies that can cater to complex environments. This paper aims to explore the principles of CPU scheduling, examine the strengths and weaknesses of traditional algorithms, and discuss the future potential of intelligent scheduling methods that integrate modern computational techniques.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

II. PROCESS STATE

During its life cycle, a process moves through various stages to complete its execution. While the names of these stages can differ depending on the system, at least five core states are generally recognized. The primary stages that a process passes through are described below:

1) New State: The new state refers to the stage when a program is initiated for execution but has not yet entered the system's main memory. At this point, the process is in secondary storage and being prepared for loading
2) Ready State: When a process has been completely prepared and loaded into the main memory, it moves into the ready state, where it waits for its turn to be executed by the CPU. In this phase, process waits for the CPU to assign it resources for execution. In environments that support multiprogramming, multiple processes may remain in the ready state simultaneously.
3) Running State: A process transitions to the running state once the is CPU allocated to it. This is the active phase where the process instructions are executed.
4) Terminated State: After a process successfully finishes its execution, it transitions into the terminated state, marking the end of its lifecycle. At this stage, the operating system deletes the process control block (PCB) associated it, with effectively ending the process
5) Blocked or Waiting State: If a process needs to perform an input/output operation or is waiting for a resource that is currently unavailable, it moves into the blocked or waiting state. Once the required operation is complete or the resource is free, the process transitions back to the ready state.
6) Suspend Ready State: When system memory is full, and a higher-priority process needs to execute, a lower-priority process in the ready state may be moved to the suspend ready state. This frees up memory for the higher priority task. The suspended process will remain in this state until system memory becomes available, at which point it transitions back to the ready state
7) Suspend Wait State: Similarly, if a higher-priority task requires execution and system memory is full, a lower-priority process in the waiting state may be moved to the suspend wait state. Once the required resource becomes available, the process moves into the suspend ready state, indicating that it is ready to resume execution when scheduled. From there, it moves to the ready state when memory is freed



III. CPU SCHEDULING

CPU scheduling is a crucial function of the operating system, responsible for determining which task or process will utilize the CPU at any given moment. Since multiple programs often run concurrently, the operating system must allocate CPU time effectively to ensure that every program receives a fair opportunity to execute. The main objective of CPU scheduling is to optimize system performance and ensure efficiency, resulting in a faster and more responsive system.

This process is fundamental to the operation of an operating system, as it manages the selection of tasks for the CPU to execute. While the CPU can only manage one task at a time, numerous processes frequently require attention, making scheduling essential for balancing workloads and maintaining system functionality.

CPU scheduling is the mechanism that enables one process to utilize the CPU while another process is waiting for unavailable resources like I/O operations, ensuring optimal utilization of the CPU. In simple terms, CPU scheduling decides the order and priority in which processes are executed, distributing CPU time according to factors like CPU utilization, throughput, turnaround time, waiting time, and response time. Its main objective is to improve the system's overall efficiency, speed, and fairness in process execution.

A. *Objectives of CPU Scheduling*

The goals of CPU scheduling are to improve the system's efficiency, speed, and fairness:

1) Efficiency: CPU scheduling ensures that each process gets appropriate CPU time according to its priority, preventing any single process from dominating CPU usage.
2) Speed: By minimizing dispatch latency, CPU scheduling enables quick context switching, allowing seamless transitions between processes for smoother execution.
3) Fairness: CPU scheduling provides equitable access to system resources, ensuring that no process is unfairly deprived of CPU time.

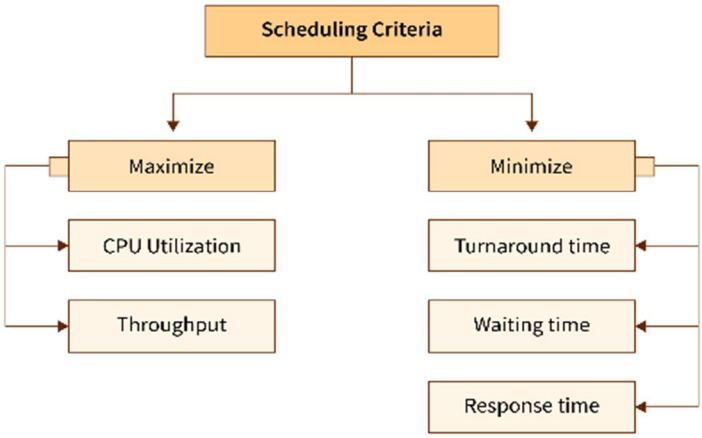
B. *Types of CPU Scheduling*

1) Preemptive Scheduling: This method is used when a process shifts from the running state to the ready state or from the waiting state to the ready state.
2) Non-Preemptive Scheduling: In this approach, the CPU is released only when a process finishes its execution or transitions from the running state to the waiting state.

C. *Scheduling Criteria*

3) CPU Utilization: The goal of any CPU scheduling method is to keep the CPU actively engaged as much as possible. While theoretically, utilization can range from 0% to 100%, in practical systems, it typically varies between 40% and 90% based on the system's workload.
4) Throughput: Throughput is a metric that evaluates the number of processes a CPU completes within a given timeframe. The rate varies according to the complexity and execution time of the processes managed by the system.
5) Turnaround Time: Turnaround time refers to the total period from the moment a process enters the system until it completes execution. It includes all the time spent in queues, processing, and performing I/O operations.
   Formula: Turnaround Time = Completion Time − Arrival Time
6) Waiting Time: Waiting time indicates the total time a process remains in the ready queue before it gets CPU access for execution.
   Formula: Waiting Time=Turnaround Time − Burst Time
7) Response Time: Response time is particularly important in interactive systems. It measures the delay between when a process request is submitted and when it receives the CPU for the first time. This metric is crucial for systems where quick feedback is needed. Formula: Response Time=First CPU Allocation Time−Arrival Time
8) Completion Time: Completion time is the timestamp when a process completes all its tasks, including CPU execution and I/O operations. This marks the end of the process's lifecycle in the system.
9) Priority: Priority-based scheduling gives precedence to processes based on their importance. Higher-priority tasks are executed first, but algorithms should also ensure fairness by preventing lower-priority tasks from being perpetually delayed.
10) Predictability: A predictable system executes processes consistently, ensuring that under similar workloads, tasks are completed in comparable amounts of time. This reliability fosters trust and ensures balanced system performance.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com



IV. CLASSIFICATION OF CPU SCHEDULING ALGORITHM

1) First Come, First Serve (FCFS): The FCFS algorithm operates on a simple rule: the process that arrives first in the queue is executed first. This scheduling method is implemented using a First-In-First-Out (FIFO) queue and ensures straightforward task management.
2) Shortest Job First (SJF): SJF prioritizes processes with the shortest burst time, meaning the task requiring the least amount of CPU time is executed first. It can function as either a pre-emptive or non-pre-emptive method and effectively minimizes the average waiting time for processes.
3) Longest Job First (LJF): As the opposite of SJF, the LJF algorithm prioritizes tasks with the longest CPU burst time. This scheduling method is non-pre-emptive, focusing on processes that require the maximum execution time.
4) Priority Scheduling: This algorithm assigns CPU time based on the priority of a process. Higher-priority tasks are executed first, and ties between processes with the same priority are resolved using the FCFS method. This scheduling can be pre-emptive or non-pre-emptive.
5) Round Robin (RR): Round Robin assigns each process a fixed time slice or quantum and cycles through all processes in the queue. It is a pre-emptive scheduling approach designed for time-sharing systems to ensure fair and efficient task execution.
6) Shortest Remaining Time First (SRTF): SRTF is a pre-emptive variation of SJF. It continuously checks the remaining burst times of processes and allocates the CPU to the task closest to completion.
7) Longest Remaining Time First (LLRTF): LRTF is a pre-emptive scheduling method where tasks with the longest remaining execution time are given priority. This approach ensures systematic handling of processes with high computational demands.
8) Highest Response Ratio Next (HRRN): HRRN is a non-pre-emptive scheduling algorithm that selects the process with the highest response ratio, calculated based on waiting time and burst time. Once chosen, the process executes without interruption until it is complete.
9) Multilevel Queue Scheduling: In this approach, processes are categorized into different classes, such as interactive and batch processes, based on their characteristics. Each class follows a distinct scheduling strategy to cater to its specific needs.
10) Multilevel Feedback Queue Scheduling (MLFQ): MLFQ improves upon multilevel queue scheduling by allowing processes to move between queues based on their behaviour and execution history. This dynamic nature makes it more efficient and adaptable for varying workloads.

V. DETAILED ANALYSIS OF ALGORITHM

A. *First Come First Serve*

Characteristics of FCFS (First-Come, First-Served):

The First-Come, First-Served (FCFS) algorithm can be implemented as either a pre-emptive or non-pre-emptive CPU scheduling method. The key principle behind FCFS is that tasks are executed in the order they arrive in the queue. It is simple to implement and understand, but the performance tends to be suboptimal, leading to longer waiting times.

Advantages of FCFS:

- Easy to implement and straightforward.
- Follows a simple first-come, first-serve methodology.

Disadvantages of FCFS:

- Prone to the convoy effect, where a prolonged process delays the execution of shorter tasks.
- Average waiting time tends to be high compared to other algorithms.

B. Shortest Job First

Characteristics of SJF (Shortest Job First):

Shortest Job First (SJF) is an algorithm that selects the process with the shortest burst time for execution. It results in minimal average waiting time, making it efficient in managing resources, especially in batch systems. However, SJF can lead to starvation when shorter tasks keep arriving, which can be mitigated through aging techniques.

Advantages of SJF*:*

- Reduces average waiting time, which makes it more efficient than FCFS.
- It is often used for long-term scheduling in batch systems.

Disadvantages of SJF:

- Prone to starvation for long tasks if shorter tasks keep arriving.
- Predicting the exact burst time of upcoming processes can be challenging.

C. Longest Job First

Characteristics of LJF (Longest Job First):

In the Longest Job First (LJF) algorithm, the CPU always executes the process with the longest burst time first. If two processes share the same burst time, the one that arrives first is selected for execution, like FCFS. LJF can be implemented in both preemptive and non-preemptive modes.

Advantages of LJF:

- No other task can be scheduled until the longest task is completed.
- Helps achieve a synchronous completion of all tasks.

Disadvantages of LJF:

- Leads to higher average waiting time and turnaround time.
- Can cause a convoy effect.

D. Priority Scheduling

Characteristics of Priority Scheduling:

Priority Scheduling assigns each process a priority value. Processes with higher priority are scheduled before those with lower priority. If a process with a higher priority arrives while another with lower priority is executing, the former preempts the latter. A lower numeric value corresponds to higher priority.

Advantages of Priority Scheduling:

- Results in lower average waiting time than FCFS.
- Less complex to implement than other algorithms.

Disadvantages of Priority Scheduling:

- Can lead to starvation, where low-priority processes are indefinitely delayed.
- A process that is ready to run but waiting for the CPU can be considered blocked.

E. Round Robin Scheduling

Characteristics of Round Robin (RR):

Round Robin scheduling is considered one of the simplest pre-emptive CPU scheduling algorithms. Each process is assigned a fixed time slice or quantum, after which it is placed at the end of the queue if it has not completed. This ensures fairness, as all processes receive an equal share of the CPU.

Advantages of Round Robin:

- Provides a fair share of CPU time to all processes.
- Starvation-free, as all processes get executed in cycles.

Disadvantages of Round Robin:

- When demand for resources is high, the system can become overloaded, and performance can degrade. This can lead to a backlog of pending work.
- Spends more time on context switching.

F. Shortest Remaining Time First

Characteristics of SRTF (Shortest Remaining Time First):

Shortest Remaining Time First (SRTF) is a pre-emptive version of SJF, where the process with the least remaining execution time is selected. If a new process arrives with a shorter remaining time than the current process, the CPU switches to the new one. Although it can result in faster processing, frequent context switching can add overhead.

Advantages of SRTF:

- Handles short processes quickly.
- Low overhead, as decisions are made when a process finishes, or a new one arrives.

Disadvantages of SRTF:

- Can cause starvation for longer processes.
- Frequent context switches can reduce efficiency

G. Longest Remaining Time First

Characteristics of LRTF (Longest Remaining Time First):

Longest Remaining Time First (LRTF) allocates the CPU to the process with the longest remaining burst time. Like LJF, LRTF can be preemptive or non-preemptive, and if processes have the same burst time, they are scheduled using FCFS.

Advantages of LRTF:

- Maximizes throughput for long processes.
- Reduces context switching compared to other algorithms.

Disadvantages of LRTF:

- Results in high average waiting and turnaround times.
- Prone to the convoy effect.

H. Highest Response Ratio Next

Characteristics of HRRN (Highest Response Ratio Next):

HRRN is an enhancement to SJF that calculates the response ratio for each process, balancing the waiting time and burst time. The process with the highest ratio is selected for execution, which reduces the risk of starvation seen in SJF.

Advantages of HRRN:

- Typically performs better than SJF, as it reduces waiting time for longer tasks.
- Encourages a more balanced approach to scheduling.

Disadvantages of HRRN:

- Hard to implement, as it is difficult to predict burst times in advance.
- May lead to overload on the CPU if too many processes are queued up.

I. Multilevel Queue Scheduling

Characteristics of Multilevel Queue Scheduling:

Multilevel Queue Scheduling divides processes into distinct queues, each with its own priority and scheduling rules. This approach is efficient in managing different types of processes, such as interactive or batch processes, with minimal scheduling overhead.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

Advantages of Multilevel Queue Scheduling

- You can use multilevel queue scheduling to apply different scheduling methods to distinct processes.
- It will have low overhead in terms of scheduling.

Disadvantages of Multilevel Queue Scheduling:

- Can suffer from starvation, especially in lower-priority queues.
- Inflexible, as processes are permanently assigned to one queue.

J. Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling allows processes to move between different queues based on their behavior. For instance, a CPU-heavy process may be moved to a lower-priority queue, while interactive tasks are given higher priority.

Advantages of Multilevel Feedback Queue Scheduling:

- Offers flexibility, allowing processes to shift between queues.
- Efficient in handling diverse workloads.

Disadvantages of Multilevel Feedback Queue Scheduling:

- Introduces CPU overhead.
- Complex to implement and maintain. While easy to implement, it is less efficient in managing resources effectively.

VI. CHALLENGES IN CPU SCHEDULING

1) Starvation: Starvation occurs when a low-priority process is repeatedly delayed due to the continuous execution of higher-priority processes. This can result in prolonged waiting times or the possibility that the low-priority process is never executed.
2) Response Time vs. Throughput: In scheduling, there is often a trade-off between minimizing the response time for user interactions, which is crucial for interactive systems, and maximizing throughput, which refers to the number of processes completed in each time, a key metric for batch systems.
3) Context Switching Overhead: Switching between processes on the CPU involves context switching, which can lead to additional overhead. This overhead becomes significant when context switches occur frequently, impacting system performance.
4) Predicting Burst Time: Estimating the duration a process will spend on the CPU, known as burst time, is challenging. Inaccurate predictions can reduce the efficiency of scheduling algorithms.
5) I/O-Bound vs. CPU-Bound Processes: Processes can be categorized based on their resource usage: I/O-bound processes spend most of their time waiting for I/O operations, while CPU-bound processes utilize the CPU extensively. Identifying these types can help optimize scheduling strategies.
6) Priority Inversion: Priority inversion happens when a high-priority process is unable to proceed because it depends on a resource held by a lower-priority process. This can lead to unexpected delays in system performance.
7) Dynamic Arrival of Processes: Processes often enter the system at unpredictable intervals. Efficient queue management is necessary to handle these dynamic arrivals and ensure smooth system operation.

VII. MODERN TRENDS IN CPU SCHEDULING

1) Adaptive Algorithms

Dynamic priority updates: Adjusting process priorities based on factors like recent execution time, I/O wait time, or resource usage to better cater to changing workload demands.

Feedback-based scheduling: Utilizing feedback from system performance metrics to refine scheduling decisions over time.

2) Machine Learning in Scheduling

Predictive scheduling: Using machine learning models to predict future process behavior (like
CPU burst time) and proactively allocate CPU time.

Reinforcement learning: Employing reinforcement learning algorithms to learn optimal scheduling policies based on system state and reward signals.

3) Multi-core and heterogeneous architecture optimization

Thread-aware scheduling: Considering thread affinity and dependencies when assigning tasks to cores for better performance.
Heterogeneous core scheduling: Optimizing task allocation across different types of cores (e.g., high-performance, power-efficient) based on workload characteristics.

4) Energy-aware scheduling

Power-aware priority assignment: Prioritizing processes with lower power consumption when possible. Dynamic voltage and frequency scaling (DVFS): Adjusting CPU clock speed to match workload demands and reduce power consumption.

VIII. CONCLUSION

Efficient CPU scheduling is essential for enhancing the performance and effectiveness of operating systems. By effectively managing the order and allocation of CPU time among various processes, it ensures maximum utilization of resources, minimizes waiting times, and enhances overall system throughput. Each scheduling criterion, such as turnaround time, waiting time, and response time, addresses specific aspects of system performance, highlighting the need for tailored algorithms based on the requirements of different applications.

The research demonstrates that while traditional algorithms like FCFS, SJF, and Round Robin have laid the foundation for CPU scheduling, modern trends such as hybrid and adaptive approaches are paving the way for more dynamic and efficient systems. However, challenges such as handling dynamic workloads and ensuring fairness across processes remain areas for further exploration.

Many contemporary computer systems support multiple processors and allow each processor to schedule itself independently. Typically, each processor maintains its own private queue of processes (or threads), all of which are available to run. Additional issues related to multiprocessor scheduling include processor affinity, load balancing, and multicore processing. A real-time computer system requires that results arrive within a deadline period; results arriving after the deadline has passed are useless. Hard real-time systems must guarantee that real-time tasks are serviced within their deadline periods. Soft real-time systems are less restrictive, assigning real-time tasks higher scheduling priority than other tasks. Real-time scheduling algorithms include rate-monotonic and earliest deadline-first scheduling. Rate-monotonic scheduling assigns tasks that require the CPU more often a higher priority than tasks that require the CPU less often. Earliest-deadline-first scheduling assigns priority according to upcoming deadlines—the earlier the deadline, the higher the priority. Proportional share scheduling divides up processor time into shares and assigning each process several shares, thus guaranteeing each process a proportional share of CPU time.

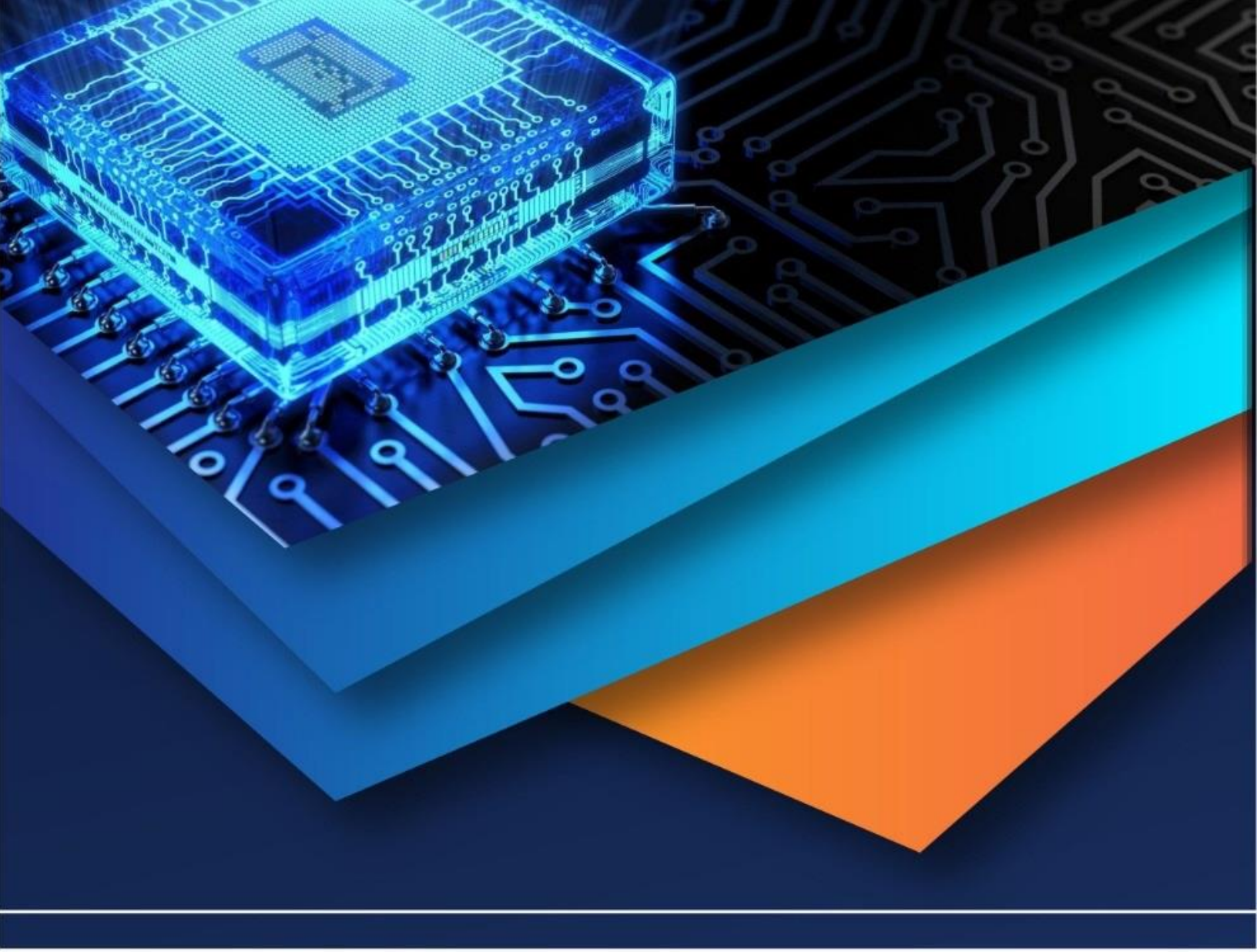
The POSIX Pthread API provides various features for scheduling real-time threads as well. Operating systems supporting threads at the kernel level must schedule threads—not processes—for execution.

This is the case with Solaris and Windows. Both systems schedule threads using pre-emptive, priority-based scheduling algorithms, including support for real-time threads. The Linux process scheduler uses a priority-based algorithm with real-time support as well. The scheduling algorithms for these three operating systems typically favour interactive over CPU-bound processes.

In conclusion, selecting the right CPU scheduling algorithm is crucial for achieving a balance between efficiency, predictability, and system responsiveness, making it an essential focus for ongoing research and development in operating systems.

REFERNCES

[1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts (10th ed.). Wiley.
[2] Tanenbaum, A. S., & Bos, H. (2015). Modern operating systems (4th ed.). Pearson Education.
[3] Warford, J. S. (2009). Operating systems: A systematic view (6th ed.). Jones & Bartlett Learning.
[4] Anderson, T., & Dahlin, M. (2014). Operating systems: Principles and practice (2nd ed.). Recursive Books.
[5] Scaler. (2024). CPU scheduling in operating systems. Retrieved from https://www.scaler.com/topics/
[6] ResearchGate. (2023). Comparative analysis of CPU scheduling algorithms. Retrieved from https://www.researchgate.net/
[7] ProQuest. (2023). Research on process scheduling algorithms. Retrieved from https://www.proquest.com/
[8] AISSMS Polytechnic. (2023). Operating systems study material. Retrieved from https://aissmspoly.org.in/
[9] INFLIBNET. (2024). E-books on operating systems. Retrieved from https://ebooks.inflibnet.ac.in/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY