CPU Scheduling Algorithms and Innovations

Himanshu Raj — himanshuraj97653@gmail.com ,
Dindi Prashanth Reddy — prashanthre99@gmail.com,
Gaganajit Singh Dhillon — dhillongaganajitsingh@gmail.com,

Lovely Professional University, Jalandhar-Punjab, India
www.lpu.in

Index

Abstract

This report will be a detailed study of the different CPU scheduling algorithms. The performance, weaknesses, and possible improvements for the different algorithms can be found. On traditional algorithms, such as First-Come, First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin, it is possible to make further comparisons on how those algorithms are different from one another in different computing environments. Further analysis has been conducted based on metrics including throughput, waiting time, response time, and CPU utilization for each algorithm. Other new topics covered in CPU scheduling include low-power algorithms that focus on reduction in energy consumption in current systems and real-time scheduling applied in the use of time-critical tasks in embedded and safety-critical systems. The report further does an analysis of the application of machine learning techniques in optimizing CPU scheduling with predictive models dynamically adapted to scheduling decisions in search of higher performance in variable workloads. Challenges of the Approach This section discusses some of the challenges associated with the approach: the trade-off between scheduling fairness and performance, as well as between real-time constraints and power efficiency. Finally, some improvements are proposed to overcome such challenges; hence, the discussion includes several hybrid scheduling approaches and adaptive algorithms.

1. Introduction

It plays a very significant role about how the access to CPU is allocated to the process in the operating system and which in turn is very important to protect the performance and efficiency of the system. Therefore, this is an inevitable need that may arise since computers and devices deal with bigger numbers of tasks, and so the OS has to determine a sequence as well as period for which each process has to access the CPU. This naturally leads to defining key performance metrics: Throughput, response time, CPU utilization, and fairness. Several classic scheduling algorithms have been developed to fulfill this need, including First-Come, First-Served, Shortest Job First, Round Robin, and Priority Scheduling.

In addition to those traditional techniques of scheduling, significant advancement has taken place in scheduling innovations that fulfill the need for modern computing environments. For instance, real-time systems used in medical devices or industrial controls are subject to specific algorithms for scheduling, like RMS and EDF, that ensure fundamental timing constraints. Energy efficiency becomes yet another very relevant consideration in the context of CPU scheduling, especially in mobile and embedded systems where battery life is considered significant. Power is scaled down without hurting performance too badly by techniques like Dynamic Voltage and Frequency Scaling (DVFS). It is widely applied to a domain that machine learning operates in, which is CPU scheduling. Systems continuously learn dynamic adaptation to changing workloads and adapt scheduling based on patterns and predictions. In a sense, innovation has been driven by the complexity and variability of modern computing environments-a "perfect storm" that ensures CPU scheduling remains well-tuned to forward needs.[1],[2],[3]

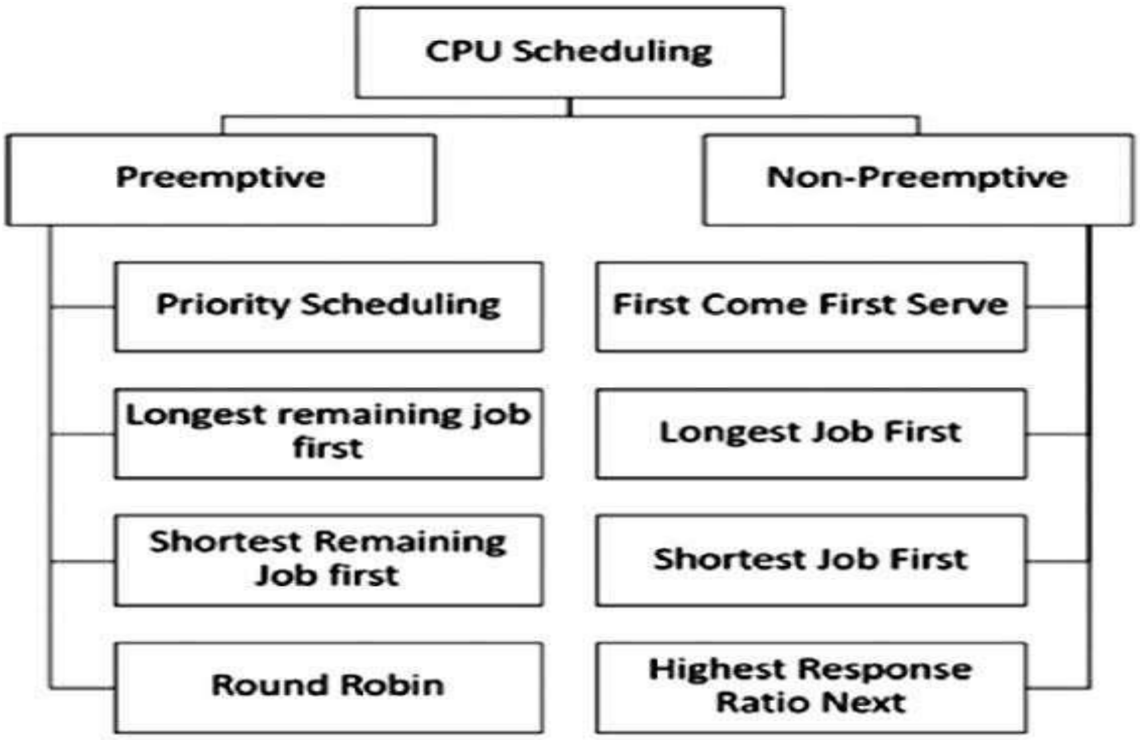

Fig 1 .The figure depicts a classification of CPU scheduling algorithms into **Preemptive** (e.g., Round Robin, SRTF, Preemptive Priority) and **Non-Preemptive** (e.g., FCFS, SJF, Non-Preemptive Priority) categories.[5]

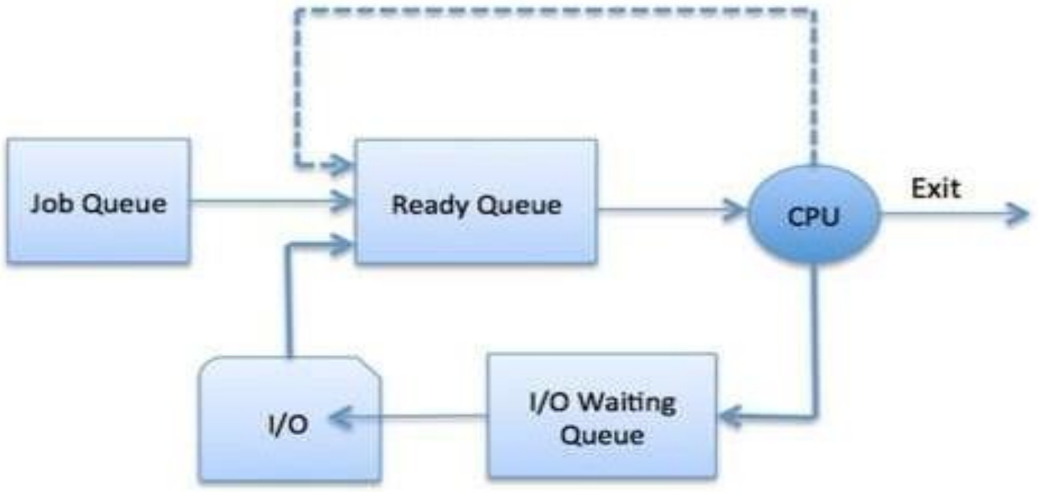

Fig 2 .The figure illustrates a job scheduling flow where jobs first enter the **Ready Queue**. From there, the **Dispatcher** either directs jobs to the **CPU** for execution or moves them to the **Waiting Queue** if resources are unavailable. This process shows the decision paths for jobs in a CPU scheduling system.[5]

2. Comparison of CPU Scheduling Algorithms

2.1 First-Come, First-Served (FCFS)

The one of the simplest CPU scheduling algorithms is FCFS. This algorithm executes the processes in the order of their arrival. Although it is easy to implement, due to the convoy effect where shorter tasks have to wait for longer ones, the average waiting time of those requiring less cycles suffer.

For example, in the case of a three-process system, if time for CPU for P1=10ms, P2=5ms, and P3=1ms, and they would arrive in the order P1, P2, and P3, then P3 would wait for the completion of both P1 and P2, which would increase its waiting time. [2],[3],[5]

2.2 Shortest Job First (SJF)

For instance, when a short job comes when a long job is being processed, SJF can make proper use of the processor to lessen the waiting time by executing the shorter job first. This is effective in systems with predictable job execution times but may introduce errors in the system when it comes to unknown or dynamically changing job length. [2],[3],[5]
Advantages:
• Reduces average waiting time compared to FCFS
• Suitable for minimizing waiting time in most of the cases
Disadvantages:
• Estimating Job Lengths: It cannot predict lengths of future processes which is not possible.
• Starvation: It may become so that long processes never gets executed.
Example
• Let process with burst times : P1 = 10ms, P2 = 5ms, P3 = 1ms.
o P3 runs first then P2 then P1 .
o Waiting Time: P1 = 6ms, P2 = 1ms, P3 = 0ms.

2.3 Round Robin (RR)

Round Robin. This is a fair scheduling algorithm in which every process gets a predefined time quantum. The CPU switches among processes cyclically. Though RR ensures no process is starved, it could have very high context switching overheads if the time quantum is very low.
Advantages:

Fairness: Every process gets equal chance and thus no starvation.

Preemptive: Processes can be preempted and another can take over.

Disadvantages:

High Context Switching: If time quantum is low then majority of the clock would go in switching and very little would go in execution of the process

Approaching FCFS: If time quantum is high then it will almost behave like FCFS.

Example:

Suppose three processes P1, P2, and P3 with time quantum of

2.4 Priority Scheduling

In Priority Scheduling, processes get different priorities. High-priority processes are executed first. This algorithm can also lead to starvation of lower-priority processes because of which aging techniques become important.

2.5 Performance Comparison

For various algorithms, performance depends on various parameters such as the system workload, the length of jobs, and the mix of CPU-bound and I/O-bound processes. On average, waiting time is concerned, SJF is better than other algorithms though requires that the length of jobs be predicted precisely. RR ensures fairness but may incur greater overhead due to the context switch. Priority scheduling is flexible but dangerous and prone to starvation in the majority of systems **that contain high-priority jobs**

3. Improvement of Round Robin Scheduling

More appropriate for Round Robin would be dynamically changing the time quantum based on process behavior. In adaptive Round Robin, fair effort should be balanced with efficiency-that is, reduce the overhead of context switching while at the same time enhancing the performance of the system as a whole. One improvement would be to keep track of the average burst time of processes and set the quantum accordingly. If most the processes have short burst times, a smaller quantum reduces context-switches without compromising fairness. **[2],[4]**

4. Real-Time CPU Scheduling

Real-time systems require scheduling algorithms that ensure processes meet strict timing requirements. In these systems, missing a deadline can have serious consequences, especially in applications like air traffic control or medical devices.

4.1 Rate-Monotonic Scheduling (RMS)

Rate-Monotonic Scheduling is a fixed-priority algorithm where tasks with shorter periods are assigned higher priority. RMS is used in systems with periodic tasks that have predictable execution times. However, RMS is not optimal for all real-time systems, especially if tasks have irregular periods.[3],[4]

5. Energy-Efficient Scheduling

Energy-efficient CPU scheduling has emerged as a critical component in the design of current mobile and embedded systems due to its capability to balance performance with power consumption. This problem is particularly significant for various battery-powered gadgets, such as smartphones and tablets and IoT devices, wherein their life extensions are of much concern. Several techniques for optimizing power use have been developed, such as DVFS and algorithms for sleep and wake times. So, let's dive a little deeper into these concepts:

Dynamic Voltage and Frequency Scaling(DVFS)

DVFS is the policy of dynamically changing the voltage and frequency at which the CPU runs, based on workload. The technique works:

High workload: In the high workload case, when the gadget is performing an intensive task like online gaming or video processing, the CPU clock is running with a higher frequency and voltage to offer the required performance

- Low workload: When the workload is low and less stressful, for instance, when the device is idle or running background applications, its frequency and voltage are scaled down; this saves a lot of power without impacting the performance much.

Sleep/Wake Algorithms (Idle Time Management)

The other aspect of energy-efficient scheduling includes the handling of idle time. As long as there is no execution in the CPU or it sits idle, waiting for any execution, it is not essential to have it in full power state. Modern CPUs have several states-they could be active, idle or deep sleep. Under such low power states, it can quickly wake up at any instant and resume its operations to conserve energy:

- Sleep States: If the CPU needs to wait for periods of inactivity, it can go to successively deeper sleep states, consuming less power but requiring more and more time and cycles to wake up again. The scheduler controls the basis of how to change state under predicted idle times.

- Idle Power Management: Modern cellular devices spend a large amount of their time idle, and good management of this can save a lot of energy. The OS uses heuristics to predict when the device is going to be idle and so selectively put the CPU in low power states during these times.

Hybrid of DVFS with Sleep/Wake

Now-a-days, the latest CPUs from many modern manufacturers contain both the DVFS and sleep/wake algorithms in order to optimize the performance. It dynamically adjusts the frequency and the voltage while observing the idle times. In case it predicts a long idle period, it pushes the CPU into a low power state. Thus, theoretically, it may reduce its energy consumption not only when idle but also during active periods.

Use in Mobile Devices

Due to such variability in workloads within mobile devices, depending upon the nature of activities of users (example: scrolling within a web page versus playing video games), DVFS and sleep/wake methodologies are very important in mobile devices:

- Background Tasks: Most applications run in the background in mobile, but usually do not require the full power of the CPU. By reducing the frequency and placing the CPU in lower power states, the mobile can save its battery.

6. Machine Learning in CPU Scheduling

Application of machine learning (ML) is applied to CPU scheduling such that the CPU identifies its dynamic workload and adjusts for optimized resource allocation and efficiency in performing work. Some of the key techniques include the following:

1. Predictive Models for Estimation of Job Length

It predicts the job length by using data related to the past; these could be regression or neural networks. Therefore, a good model will allow schedulers to rank their tasks better so that waiting time can be reduced and in turn improve throughput of the system. These can be dynamic adaptation models that adapt with time as the workload keeps changing. [2],[3]

2. Pattern recognition in the pattern of workloads

The clustering algorithms, such as k-means or time-series models (e.g., RNNs), identify patterns in workloads. Examples of these patterns may be spikes in CPU demand at specific times of day. Therefore, proactive scheduling adjustments take place as well as peak loads are dealt with effectively. [3],[4]

7. Conclusion

This report discusses in-depth CPU scheduling algorithms, from traditional ones such as FCFS, SJF, and RR to advanced techniques about energy-efficient and machine learning-based scheduling. It is shown that with the approach in performance comparisons, no algorithm is the best suited for all cases; however, each has relative strengths and weaknesses under different workloads and system requirements. Future work may focus on hybrid scheduling techniques to combine the best of both worlds between the two traditional and modern approaches.

8. Reference

1. **Stallings, W. (2015).** Operating Systems: Internals and Design Principles (9th ed.). Pearson.

- A straightforward textbook that covers basic operating system concepts, including CPU scheduling algorithms.

2. **Silberschatz, A., Galvin, P. B., & Gagne, G. (2018).** Operating System Concepts (10th ed.). Wiley.

- A widely used book that explains key scheduling algorithms with examples and comparisons.

3. **Gallo, C., & Barletta, M. (2020).** "An Introduction to CPU Scheduling." Journal of Computer Science & Technology

- A beginner-friendly article that introduces CPU scheduling and compares common algorithms.

4. **Sinha, P., & Choudhury, A. (2018).** "Understanding CPU Scheduling Algorithms." International Journal of Computer Applications.

- A simple overview of various CPU scheduling algorithms and their characteristics.

5. https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems