Comparison and Analysis of Scheduling Algorithms: Exploring Performance, Time, Fairness, and Applicable Scenarios

Wei Pan *

School of Software, South China University of Technology, Guangzhou, Guangdong Province, 510000, China

* Corresponding Author Email: 202230482056@mail.scut.edu.cn

Abstract. Scheduling algorithm is an important part of CPU work and plays a key role in the program system. Successful scheduling algorithms can effectively improve system efficiency and stability. This paper aims to analyze the characteristics of several scheduling algorithms comparatively through some general criteria to help determine the applicable algorithms in different situations. This paper first introduces the important role of scheduling algorithms. It then discusses and analyzes in detail the complexity of scheduling algorithms, allocation methods, average waiting time, and starvation. Finally, it draws the advantages and disadvantages of different scheduling algorithms through comparison. The types of scheduling algorithms applicable to batch systems and real-time/interactive systems respectively are concluded. The study results provide a theoretical basis for system designers to achieve the optimal utilization of CPU resources in the system. By analyzing and comparing the advantages and disadvantages of different scheduling algorithms, this paper lays a solid foundation for the future research with more scheduling methods.

Keywords: CPU scheduling algorithm, waiting time, batch system, real-time system.

1. Introduction

Scheduling is one of the core functions of an operating system, which directly affects the performance of the central processing unit (CPU) and the utilization of system resources. The design and implementation of scheduling algorithms determines the efficiency of the flow of processes through the system. How to manage the processes in the ready queue, to improve the system throughput and minimize the average waiting time while ensuring the fairness of the processing are all factors that need to be considered comprehensively in the scheduling strategy. During the scheduling process, task execution may be interrupted due to both the need for I/O operations and time slice expiration. Interruption mechanisms such as these ensure that other processes in the queue have access to processing opportunities, thereby improving system fairness and resource utilization efficiency. In a multi-process environment, processes are diverse, for example, they are categorized as time-sharing processes and cooperative processes. These processes exist in different forms in the computer system, each with unique characteristics and needs. A time-sharing process means that multiple processes share CPU time by occupying CPU resources for a period of time and then switching to other processes. Cooperative processes refer to multiple processes that work together by sharing resources or data, and therefore require reasonable synchronization and mutual exclusion to make the system work properly. The CPU must efficiently manage the computational tasks and data flows of all these different types of processes through appropriate scheduling algorithms. Efficient scheduling ensures optimal utilization of system resources and is effective in preventing bottlenecks and reducing idle time.

The rapid growth in computational demands due to technological advances and increased application complexity has further exacerbated the need for powerful and efficient scheduling algorithms. The application of Very Large Scale Integrated Circuit (VLSI) technology has enabled the creation of high-performance processors, further emphasizing the importance of complex scheduling mechanisms. In such advanced and complex computing environments, achieving high CPU utilization and maintaining system stability are key objectives in the design of current scheduling algorithms.

In addition, modern computing environments often involve real-time applications and interactive systems where delays and inefficiencies can lead to significant performance problems and response issues. Effective scheduling algorithms are essential to meet real-time demands, ensuring that high-priority tasks are handled in a timely manner while balancing the demands of other processes. This balance is essential to maintain high system performance and good user satisfaction.

Currently, common CPU scheduling algorithms include First-Come-First-Served (FCFS), Shortest-Job-First (SJF), Multi-Level Feedback Queuing (MLFQ), and Polled Scheduling (RR). Each of these algorithms has its characteristics but is also deficient in different aspects. For example, FCFS algorithm is simple and easy to implement but may lead to long waiting time, SJF algorithm optimizes the average waiting time but it is difficult to predict the task length accurately, Multi-Level Feedback Queuing (MLFQ) scheduling is prone to starvation problem, and RR algorithm needs to set a suitable time slice to balance the efficiency and response time. In addition to scheduling algorithms, operating systems rely on different types of schedulers to manage processes. Schedulers in the operating system can be categorized into long-term schedulers, medium-term schedulers, and short-term schedulers, each responsible for different phases of process management and scheduling tasks, thus forming a complex process management system. The long-term scheduler decides which processes can enter the ready queue, the medium-term scheduler is responsible for the exchange of processes between main memory and secondary memory, and the short-term scheduler decides which specific process will be executed on the CPU. The scheduling algorithm used by each scheduler may also vary.

This paper aims to analyze and compare these basic CPU scheduling algorithms in depth to select the most suitable scheduling strategy based on specific workloads and system requirements. By comparing the performance of different algorithms and applicable scenarios, this paper will provide a comprehensive view to help readers understand the advantages and limitations of each algorithm. The next part of this paper will introduce scheduling standards, various scheduling strategies in turn, and at the end, we will rely on related work to compare and analyze these strategies to provide theoretical support and guidance for practical system design.

2. Introduction to the fundamentals of scheduling algorithms

There is a wide variety of scheduling algorithms, and some common and modern evaluation criteria are usually needed to quantitatively compare various scheduling algorithms. Evaluating and recognizing the characteristics of different scheduling algorithms can help you choose the right algorithm.

2.1. Common Criteria for Measuring Scheduling Algorithms

When evaluating the performance of scheduling algorithms, several key criteria are commonly considered:

First, CPU utilization is an important metric, which refers to the proportion of the time the CPU spends executing instructions in a given time period. Systems typically want to maintain a high CPU utilization to ensure system resources are fully utilized, thereby improving overall performance.

Second, throughput is also a key criterion, which refers to the number of tasks accomplished per unit of time. A higher throughput means that the system is able to handle more tasks in a shorter period of time, which reflects the ability and efficiency of the system to handle the tasks.

Then, turnaround time is the time experienced from when a process is submitted to its completion. Turnaround time includes wait time and execution time, and possibly I/O time. In non-I/O systems, turnaround time is usually equal to completion time minus arrival time. A shorter turnaround time means that processes can complete more quickly, resulting in increased user satisfaction and system efficiency.

Wait time is the sum of the time a process spends in the ready queue waiting to be scheduled by the CPU. The wait time is usually equal to the completion time minus the arrival time minus the turnaround time. The system usually wants to keep the average wait time as low as possible to

minimize the amount of time a process waits in the ready queue and thus increase system responsiveness.

Response time is the time that elapses from when a process submits to when it first receives a response from the system. In batch systems, the response time requirement is usually low, whereas in interactive systems, response time is a very important metric because the user expects the system to respond quickly to his/her request.

Finally, fairness refers to the balanced allocation of CPU resources. This includes modes such as balanced allocation to each process and allocation to each user. Fairness ensures that all processes or users receive a reasonable amount of CPU time, thus preventing certain processes or users from being denied resources for long periods and avoiding starvation problems.

In summary, these criteria together form an important basis for evaluating the performance of scheduling algorithms, helping system designers to select and optimize scheduling strategies suitable for specific application scenarios.

2.2. Common Scheduling Algorithm Strategies

Scheduling strategy refers to the rules and methods that the operating system uses to manage and allocate CPU resources in a multitasking environment. Different scheduling strategies have an important impact on system performance, resource utilization and user experience. Common scheduling strategies are mainly categorized into three types, the third of which is formed by combining the first two types.

First, non-preemptive scheduling is a scheduling strategy in which once a CPU resource is allocated to a process, the process will not be interrupted unless the process has finished or explicitly declared to relinquish the occupied CPU resource. This means that nonpreemptive scheduling does not involve context switches during process execution, and the process can run continuously until it finishes. The advantage of this strategy is that it is simple to implement and has low overhead, but the disadvantage is that it may cause some processes to occupy CPU resources for a long period of time, thus affecting the response time and fairness of the system.

Second, preemptive scheduling is a scheduling strategy that allows the system to interrupt the current process and allocate CPU to other processes under certain conditions. In preemptive scheduling, the operating system can decide when to interrupt the current process and switch to another process based on priority, time slice, and other factors. The advantage of this strategy is that it can improve the system's response speed and resource utilization, especially in multitasking and interactive systems. However, preemptive scheduling is more complex to implement, and frequent context switches increase system overhead.

Finally, hybrid scheduling is a scheduling method that adapts specific policies to system requirements. Hybrid scheduling combines the advantages of non-preemptive scheduling and preemptive scheduling, and flexibly applies different scheduling strategies in different situations. For example, non-preemptive scheduling can be used to reduce context switching overhead when the system load is light, while preemptive scheduling is used when the system load is heavy or fast response is required. This strategy can provide better performance and user experience in different application scenarios.

To summarize, non-preemptive, preemptive, and hybrid scheduling are common scheduling strategies, each with unique advantages and applicable scenarios. Choosing the right scheduling strategy is crucial for optimizing system performance and improving user satisfaction.

2.3. Common Scheduling Algorithms

Scheduling algorithms are further based on the scheduling policy to determine how to determine how to allocate resources for processes. Specific scheduling algorithms have a greater impact on the allocation of resources and scheduling details for processes than scheduling policies. For example, common scheduling algorithms include first-come-first-served, shortest-job-first, polling, multi-level feedback queuing, and priority scheduling.

FCFS is an algorithm that allocates CPU resources in order processes arrive. It is generally realized by maintaining a ready queue, which is a FIFO queue. Processes in the queue can only leave the queue and use the resources when the process that is occupying the CPU finishes execution or voluntarily gives up CPU resources. This scheduling algorithm is the simplest to implement, but there are a number of problems associated with it. For example, suppose the CPU is currently occupied by a long process or an I/O-heavy process. In that case, the short process in the ready queue will be forced to wait for the process to finish occupying it, resulting in a very long average system wait time. Therefore, this scheduling algorithm is suitable for situations where the task length variability is not large and the CPU is busy.

SJF is to prioritize the scheduling of processes with the shortest expected execution time. A variety of data structures such as minimal heap, sorted linked list, balanced binary search tree and hash table can implement it. The shortest job prioritization scheduling algorithm can be preemptive or non-preemptive. This algorithm is theoretically the best choice for reducing the average waiting time and turnaround time of processes. The biggest problem with the shortest job-first algorithm is that in practice it is almost impossible to accurately predict the time required for a process to execute to completion. In addition, for some specific cases such as continuous addition of short processes in the queue can lead to starvation of long processes by not scheduling them for a long time.

RR is a scheduling algorithm that assigns a fixed time slice to each process, and all processes in the ready queue are scheduled in a sequential loop. The polling scheduling algorithm is relatively simple to implement and can be realized using data structures such as double-ended queues and circular lists. This scheduling algorithm allows all processes to get a response within a short period, which is helpful in reducing the average response time of system processes, so it is suitable for interactive systems. The relative disadvantage is that the choice of time slice can greatly affect the actual use of scheduling. Too large a time slice can close the polling scheduling algorithm to a first-come-first-served scheduling algorithm, which is difficult to function in the face of large job length variability. Too small a time slice will result in frequent context switches, leading to wasted CPU resources and slow advancement of the process queue. Choosing the right time slice size is very demanding for programmers.

MLFQ is a complex and flexible scheduling algorithm that uses multiple queues to manage and accomplish scheduling. These queues are generally of different priorities, and the time slices within each queue may differ. Various other scheduling algorithms can be used for scheduling in the consent queue, such as first-come-first-served, polling, etc. Newly arriving processes are first entered into the highest priority queue, and after obtaining a certain amount of scheduling according to the queue scheduling algorithm if the process has not completed, it is moved into a lower level queue. The time slice length of the lower queue is usually designed to be longer. By dynamically adjusting the process priority, the response time and throughput can be effectively balanced to adapt to more process types to a greater extent. However, the multilevel feedback queue scheduling algorithm also has a similar problem to the shortest job priority scheduling, i.e., if a high priority process is continuously added to the system, it will lead to starvation of the remaining processes in its priority queue.

3. Comparative study of scheduling algorithms

By using common measures of scheduling algorithms, the performance of various scheduling algorithms in various aspects can be judged more quantitatively. In this paper, we conduct a comparative analysis of four scheduling algorithms: first-come-first-served, shortest job first, polling, and multilevel feedback queuing.

3.1. Complexity Comparison

The FCFS scheduling algorithm is the simplest and its implementation only requires queues to complete the entry and exit of tasks [1]. The obvious example is the restaurant pickup queue. In contrast, the SJF algorithm is more complex and needs to be implemented through data structures

such as a minimal heap or a sorted linked list. This sorting requirement significantly increases the complexity of the algorithm.The complexity of the RR algorithm, on the other hand, depends on the size of the time slice. If the time slice is too large, the complexity is close to FCFS, whereas if the time slice is too small, the complexity is increased by frequent context switching [2]. Choosing the right time slice size keeps the complexity of the polling algorithm low. Though higher, the complexity of the MLFQ algorithm is also affected by the time slice size as its implementation is similar to RR.

3.2. Comparison of allocation methods

The FCFS algorithm allocates processes in the order of their arrival, the process that arrives first has priority to use the CPU, and later processes must wait until the current process is no longer occupied before they can start to occupy the CPU resources.The SJF algorithm allocates based on the priority that has the lowest demand for CPU resources, and compared to FCFS, SJF increases the selectivity of the invocation, and is no longer called purely according to the order of arrival.The RR algorithm divides the CPU time into fixed sized time slices TQ and allocates them cyclically according to a certain logic [3], by which all processes have a chance to get a response in a shorter period of time.The MLFQ algorithm allocates based on inter-queue rules and intra-queue rules, where inter-queue rules refer to the priority differentiation of the different queues, and intra-queue rules refer to the potentially different sizes of time slices within each level of the queue, scheduling rules, demotion rules, etc [4].

3.3. Comparison of average waiting time

The FCFS algorithm has a high average waiting time, especially when there is a large difference in job lengths. The main reason for this is the phenomenon of queue head blocking, where short processes in the queue are forced to wait for the completion of the currently executing long process, significantly increasing the average waiting time. In contrast, the SJF algorithm usually provides the shortest average waiting time, and is designed to effectively reduce the average waiting time by prioritizing the completion of shorter processes to ensure that head blocking caused by longer processes does not occur.H. Arora, D. Arora, B. Goel and P. Jain et al. show that SJF is very effective in reducing the average waiting time compared to FCFS. average waiting time. In their computational case the average waiting time of FCFS is 14.33ms while SJF takes only 4ms.The average waiting time of RR algorithm is time slice size dependent but is usually higher. The main purpose of the polling algorithm is not to reduce the average waiting time, which is higher when the time slice is small because a large amount of time is spent on context switching; when the time slice is large, the performance is close to FCFS [5]. The study of the role of time slice in round robin algorithms by Sakshi, Chetan Sharma et al. proves this point. In addition, the average scheduling time of polled scheduling algorithms is usually long. the average waiting time of MLFQ algorithms is usually low and adaptive. mlfq combines the advantages of sjf and rr to deal with short process tasks in a timely manner through high-priority queues while taking care of the response time and throughput issues.

3.4. Comparison of starvation problem

The FCFS algorithm does not lead to the starvation problem because the algorithm does not schedule with reference to priorities, and all processes must be able to obtain CPU resources after a finite amount of time [6]. In contrast, the SJF algorithm may lead to the starvation problem, in which if short processes keep joining the queue, the long processes will be starved and will never get the chance to execute, or even when they get the CPU resources, they have lost the point of executing [7].The RR algorithm, on the other hand, does not lead to starvation problem, because each process can get the chance to execute cyclically [8].The MLFQ algorithm may lead to the starvation problem by a mechanism similar to SJF, as a variant of SJF, where low priority processes will not be scheduled for a long period of time or execution no longer makes sense by the time the required resources are available if processes keep entering the high priority queue [9].The study by McCann, C., Vaswani

et al. points out the application of dynamic processor allocation policies in a multi-channel program environment, revealing that the starvation problem that multilevel feedback queues can cause.

3.5. Summary of Advantages and Disadvantages

The advantages of the FCFS algorithm are its simple implementation and relative fairness, especially in the case of small variability in process time lengths, making it more suitable for stable batch systems. However, the main disadvantage of this algorithm is its long average waiting time, especially in the case of large variability in process time lengths which is prone to queue-head blocking, and hence it is not suitable for use in real-time systems.The advantages of the FCFS algorithm for batch systems and its limitations in real-time systems are pointed out in the study by A Pant et al. [10]. Comparatively, the SJF algorithm can minimize the average waiting time. However, in practice, it is difficult to accurately estimate the time required for process execution, which limits its application in real-time systems and may also lead to starvation problems.These drawbacks have been pointed out in the literature by Pemasinghe et al [11].The RR algorithm is better suited for real-time and interactive systems by equitably distributing the time slices to ensure that each process is responded to in a shorter finite period of time. However, the disadvantages of this algorithm are the relatively high average waiting time and average turnaround time, and the high overhead of context switching, whose performance actually depends on the size of the time slice.The MLFQ algorithm is more flexible and adaptive, and can effectively balance the response time and throughput. However, the disadvantage is that it also faces the problem of starvation, which is difficult to implement and requires the programmer to adjust the parameters and method details according to the actual situation, which is more demanding for the programmer.

4. Conclusion

This paper focuses on various scheduling algorithms (FCFS, SJF, RR, MLFQ) and their advantages, disadvantages and applicability scenarios in operating system resource allocation.

The paper concludes that by comparing and analyzing the scheduling algorithms using common measures of scheduling algorithms, it can be seen that different scheduling algorithms have their advantages and disadvantages in terms of complexity, allocation method, average waiting time and starvation problem. The selection and application of these scheduling algorithms depends on the requirements of the system and the environment in which they are used. FCFS may be a suitable choice for stable batch systems because of its simple and relatively fair implementation. However, multilevel feedback queues or RRs may be more appropriate for real-time and interactive systems because they ensure that each process is responded to quickly and can effectively balance response time and throughput.

Directions for future work may include further research and optimization of existing scheduling algorithms, as well as exploring new scheduling algorithms to adapt to changing system requirements. For example, combining different kinds of scheduling algorithms could be considered to adjust the priority and allocation of processes more accurately. In addition, research can be done on how to implement scheduling algorithms more efficiently in multi-core processors and multi-tasking operating systems to improve the overall performance and throughput of the system.

Finally, it is worth noting that the selection and application of scheduling algorithms is not isolated; it also requires synergy and cooperation with other system components and strategies. For example, in memory management, there is a close connection between the page replacement algorithm and the scheduling algorithm. Therefore, in future research, all aspects of the system can be considered more comprehensively to achieve more efficient and stable system performance.

References

[1] H. Arora, D. Arora, B. Goel and P. Jain, An Improved CPU Scheduling Algorithm, International Journal of Applied Information Systems (IJAIS),2013, 6, 6, 7.

[2] McGuire C, Lee J. Comparisons of improved round robin algorithms[C]//Proceedings of the World Congress on Engineering and Computer Science. 2014, 1: 158-161.

[3] Bandarupalli S B, Nutulapati N P, Varma P S. A Novel CPU Scheduling Algorithm–Preemptive & Non-Preemptive. International Journal of Modern Engineering Research (IJMER), 2012, 2(6): 4484-4490.

[4] Raheja S, Dadhich R, Rajpal S. Designing of vague logic based multilevel feedback queue scheduler. Egyptian Informatics Journal, 2016, 17(1): 125-137.

[5] Sharma C, Sharma S, Kautish S, et al. A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time. Alexandria Engineering Journal, 2022, 61(12): 10527-10538.

[6] Omar H K, Jihad K H, Hussein S F. Comparative analysis of the essential CPU scheduling algorithms. Bulletin of Electrical Engineering and Informatics, 2021, 10(5): 2742-2750.

[7] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 1973, 20(1): 46-61.

[8] Tanenbaum A. Modern operating systems[M]. Pearson Education, Inc., 2009.

[9] McCann C, Vaswani R, Zahorjan J. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. ACM Transactions on Computer Systems (TOCS), 1993, 11(2): 146-178.

[10] Pant A. A Comparison between FCFS and Mixed Scheduling. IJCST, 2011, 2(2): 76-79.

[11] Pemasinghe S, Rajapaksha S. Comparison of CPU scheduling algorithms: FCFS, SJF, SRTF, Round Robin, priority based, and multilevel queuing[C]//2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC). IEEE, 2022: 318-323.