COMPARATIVE ANALYSIS OF FCFS, SJN & RR JOB SCHEDULING ALGORITHMS

Luhutyit Peter Damuut[1] and Pam Bulus Dung[2]

[1]Computer Science Department, Kaduna State University, Nigeria;
[2]Computer Department, FCE Pankshin, Nigeria

ABSTRACT

One of the primary roles of the operating system is job scheduling. Oftentimes, what makes the difference between the performance of one operating system over the other could be the underlying implementation of its job scheduling algorithm. This paper therefore examines, under identical conditions and parameters, the comparative performances of First Come First Serve (FCFS), Shortest Job Next (SJN) and Round Robin (RR) scheduling algorithms. Simulation results presented in this paper serve to stimulate further research into the subject area.

KEYWORDS

Scheduling; Task; Thread; Process; Algorithm; Operating Systems; Scheduling

1. INTRODUCTION

The Operating System (OS) is one of the most important parts of any computer system and other modern safety-critical systems including hospital equipment, nuclear weapons, aerospace gadgets and navigational equipment among others. The authors of [1] view an OS as the most essential piece of software that facilitates the execution of user programs.

Similarly, it is well established that there exists a direct relationship between the operating systems and the computer architecture. It is noteworthy that the earliest (1940s) digital computers had no operating systems. Programs were entered one bit at a time on rows of mechanical switches (i.e., plug boards). Programming languages and hence assembly languages were unknown. By the early 1950's however, some improvement was recorded with the introduction of punch cards.

General Motors (GM) Research Laboratories implemented the first operating systems in the 1950's for their IBM 701. This operating system generally ran one job at a time. These systems were referred to as single-stream batch processing systems because programs and data were submitted in groups or batches.

Although the systems of the 1960's were also batch processing systems, they were however, able to execute several jobs at a time. Consequently, operating systems designers developed the concept of multiprogramming, where several jobs are loaded into the main memory at once. In this scheme, a processor switches jobs as needed while keeping the peripheral devices active.

With the development of Large-Scale Integration (LSI) circuits, operating system advanced correspondingly to herald the advent of personal computers and workstations [2].

DOI: 10.5121/ijcsit.2019.11304

Microprocessor technology evolved to enable desktop computers to be as powerful as the mainframes of the 1970s.

Although the modern OS renders several functions to both the computer system and the user, this paper focuses on job scheduling.

In [3], scheduling is defined as a method through which a task, specified by some means, is assigned to resources required to carry out the specified task. The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards.

The scheduler is responsible for the scheduling activities and is often implemented with the aim of keeping computing resources busy. Scheduling is fundamental to computation itself, and constitutes an intrinsic part of the execution model of a computer system. Scheduling enables effective multitasking involving a single central processing unit (CPU). A scheduler may aim at achieving specific goals. Examples of such goals include: throughput maximization, minimization of response time or latency minimization as the case may be.

The rest of the paper is organised as follows: Section 2 presents a review of related literature; Section 3 presents highlights of the scheduling algorithms this paper focuses on. The simulation setup including test results are presented in section 4. The paper is concluded in Section 5 with the summary of findings as well as recommendation for further research on the subject.

2. REVIEW OF RELATED LITERATURE

According to [4], an operating system exploits the hardware resources of one or several processors to provide a set of services to system users. The OS also manages secondary memory and input/output devices on behalf of the user. Based on the foregoing, it is safe to say that the OS plays mediatory role between the user and the computer hardware whose main function is to handle the more technical operations of the computing system.

The functionality of the operating system changes as the needs of the user changes in complexity [5]. Moreover, the goals of the operating systems also vary depending on its cost.

The structure of OSs has also evolved over the years. A Java Application Programming Interface (API) for example, is changing the way operating systems are structured lately. Instead of having a set of system calls and unique commands, operating systems are gradually adapting and accommodating middleware [6].

According to [3], operating systems can be categorized as real-time, multi-user and embedded operating systems respectively. Modern operating systems generally have three major goals in common that are generally accomplished through the execution of running processes in low-privilege modes while providing service calls that invoke the kernel in high-privilege states. These goals include: hardware abstraction, resources allocation to processes as well as the provision for pleasant and effective user experiences.

The functionalities of the OS have evolved due to demands of mobility and disconnected operations. Regardless of the prevalence of diversity in operating system software, a typical OS has basically three major components [3]. The first component comprises of a technical layer responsible for driving the system hardware. Secondly, it has a file sub-system that logically organizes the program. The third component comprises of a command language which enables the user executes programs and manipulates files accordingly.

According to the [1], the operating system renders a number of functions including: program execution, input-output operations, file manipulation, process communications, error detection, memory management, file management and networking.

3. SCHEDULING ALGORITHMS

Scheduling is a method through which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). This is usually done to load-balance a system effectively in order to achieve a specific quality of service (QoS) target.

The need for a scheduling algorithm arose from the requirement for most modern systems to multitask and multiplex [7].

Consequently, the scheduler in any OS is primarily concerned with throughput maximization, reduction in latency as well as reduction in response time. Moreover, the scheduler is also expected to be fair to all processes in the system.

In practice, these goals often conflict (e.g. throughput vs latency), thus a suitable compromise is often sought depending on user preferences. In real-time environments, the scheduler must ensure that processes meet target deadlines. This is crucial to ensure system stability.

Although different scheduling algorithms are prevalent in most modern operating systems, each possessing distinct features, advantages and drawbacks, the following subsections highlight those that this paper focuses on:

3.1 First Come First Served (FCFS)

This scheduling algorithm uses the First in First Out (FIFO) policy to select which process to execute next from the waiting queue. The FCFS policy is the simplest scheduling strategy which does not invoke any task constraints. Ready-to-run tasks are organized in a list, whereby the task at the top of the list is executed first. Whenever a task is ready for execution, it is added to the end of the ready list. Thus, tasks execute in the order in which they become ready. FCFS is non- pre-emptive, implying that each task is allowed to run to completion before the next [8].

FCFS is noted for being simple and easy to implement. Moreover, scheduling decision costs are minimal in this case.

However, the FCFS scheduling algorithms are often criticised because short jobs get stuck behind long ones. In addition, performance is highly dependent on the order in which jobs arrive without provision for efficient CPU utilization. FCFS scheduling algorithms also suffer from unfair resource allocation resulting in long waiting time for processes on the ready queue.

3.2 Shortest Job Next (SJN)

The SJN algorithm is biased towards the processes with short service time requirements. In this algorithm, short processes are preferred over long ones irrespective of their arrival times at the ready queue. SJN is non-pre-emptive – once CPU is given to a process, it cannot be pre-empted until it completes its burst time [9].

Pre-emption is only allowed if a new process arrives with lesser CPU burst time than the remaining time of the currently executing process. This scheme is also referred to as Shortest-Remaining-Time-First (SRTF).

Pre-emptive SJN results in optimal performance of the system due to the fact that it reduces the minimum average waiting time for the processes in the system.

Starvation is however common in SJN algorithms. This often arises when a process is blocked or is repeatedly being overtaken by other processes with lesser CPU burst times.

3.3 Round Robin (RR)

The RR algorithms assign, to each thread, a small amount of CPU time to execute, and cycle among all other threads.

The RR policy is analogous to the pre-emptive version of the FCFS technique. Tasks are still arranged in the ready list in the order in which they arrive at the ready queue and are executed within a fixed time [10]. If a task is still running at the end of its time slice, it is forcibly removed from the processor and is replaced with the task at the head of the ready queue. The pre-empted task is sent to the tail of the ready queue.

Each process in the RR scheme is allocated 1/N of the CPU time in chunks of at most Q (i.e., time quantum) microseconds.

Intuitively, the performance of the algorithm depends on the size of the time quantum, Q. When the time quantum is too large, response time degrades. If on the other hand, Q is too small, throughput suffers and percentage overhead increases while an infinite Q implies the algorithm performs like its FCFS counterpart.

The RR algorithm exhibits fairness among the processes generally. In other words, all the processes are treated equally and are given equal processor time. The average waiting time is considerably lesser in Round Robin algorithms compared to FCFS scheduling algorithms.

The downside of the RR implementation is that its performance is heavily dependent on the size of the time quantum. Determining the optimal time quantum is often a non-trivial task. Moreover, the performance of the RR algorithm degrades when the jobs are of equal sizes.

4. SIMULATION RESULTS AND DISCUSSION

In order to compare the relative performances of the three algorithms under consideration, the following parameters (as illustrated in **table 1**) were used:

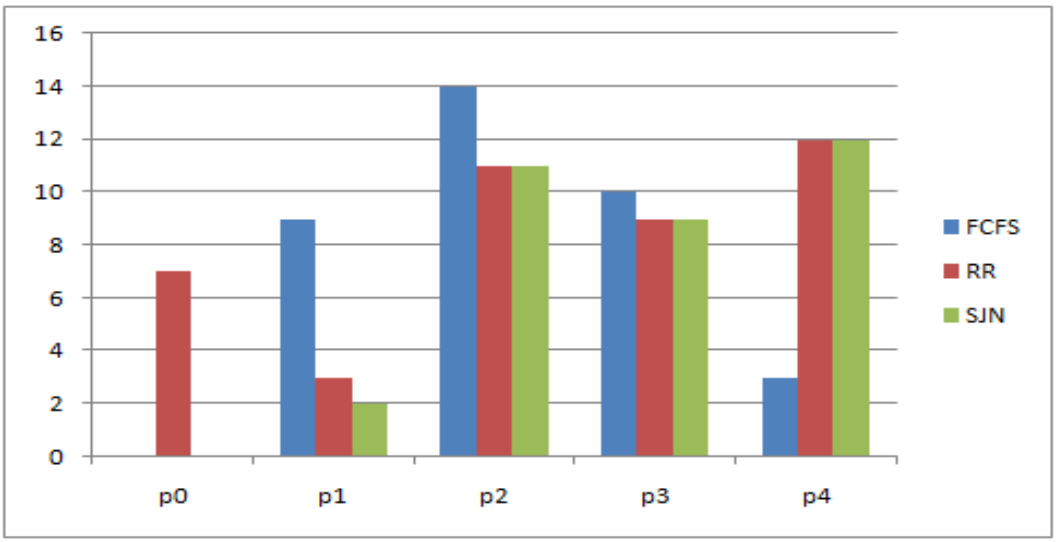
Table 1. Parameters used in executing the algorithms

Process ID	Arrival time	Burst/Execution time
P0	0	3
P1	1	1
P2	3	8
P3	1	4
P4	0	6

Five processes P0 ,…, P4 each, having different execution times. The arrival times for some of the processes are however identical with the exception of P2.

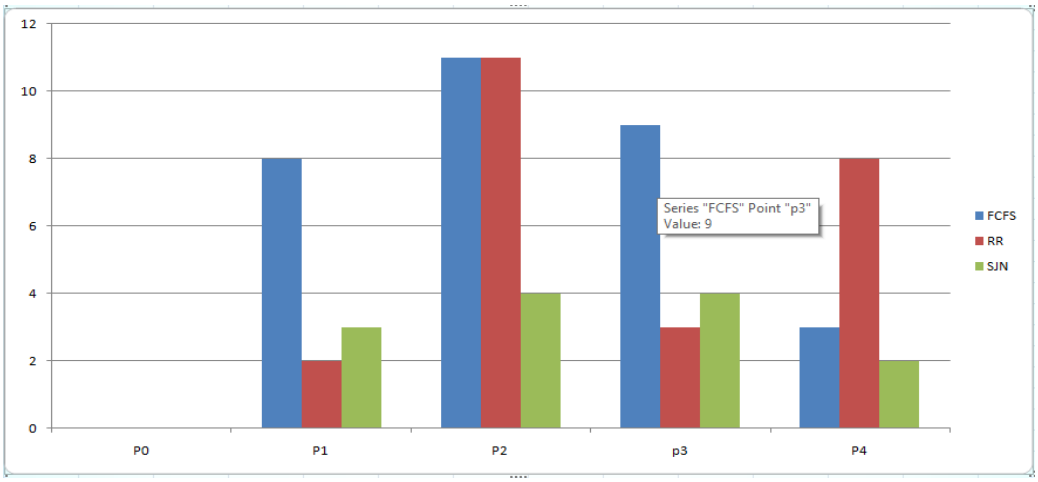
After executing the tree different job scheduling algorithms, a plot 'of the waiting times for each of the five processes is given by Fig. 1. From the figure, it is not surprising that P2 suffers the highest waiting time with the FCFS since P2 arrives last on the ready queue. On the other hand, P1 has the best waiting time for SJN due to its job size, although it is not the first to arrive at the ready queue.

Fig.1 Inter-process waiting time plot for FCFS, RR and SJN scheduling algorithms



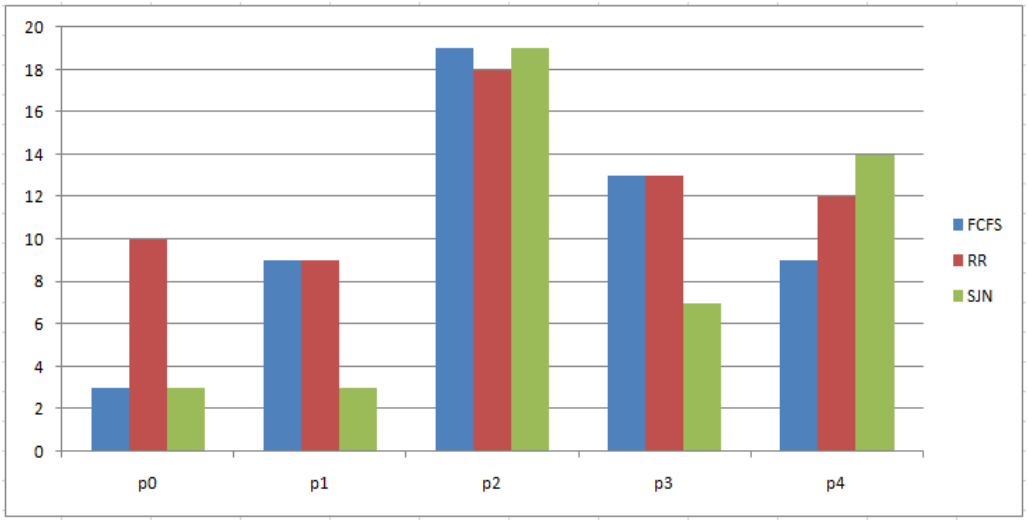
Considering response time as shown in Fig. 2, the highest values are recorded by P2 when executing the FCFS and RR scheduling algorithms. This is attributable to the fact that P2 is the process with the highest execution time besides being the last process to arrive at the ready queue. It is also evident in Fig.2 that P0 does not incur any response time with respect to any of the three scheduling algorithms due to the fact that between the first two processes to arrive at the ready queue, it has the least execution time.

Fig.2 Inter-process response time plot for FCFS, RR and SJN scheduling algorithms



The turnaround time (depicted by Fig. 3) records the highest values by P2 for all the scheduling algorithms. This is not unconnected with the fact that it is the process with the highest execution time as well as bring the last to arrive at the ready queue.

Fig.3 Inter-process turnaround time plot for FCFS, RR and SJN scheduling algorithms



The summary of the results obtained in terms of the three criteria (i.e., waiting, response and turnaround times respectively) clearly shows that the RR scheduling algorithm has the least average response time but the highest average turnaround time. In terms of the average waiting time however, the SJN scheduling algorithm outperforms the other two as shown in Table 2.

Table 2. Summary of results

	FCFS	**SJN**	**RR**
Average Waiting time	7.2	5.8	7.4
Average Response time	6.2	4.8	2.6
Average Turnaround time	10.6	8.2	11.4

5. CONCLUSIONS

The operating system is crucial for ensuring the smooth and efficient operation of any computing device. I this paper an attempt is made to empirically compare the performance of First-Come-First Served (FCFS), Shortest Job Next (SJN) and Round Robin (RR) scheduling algorithms.

After a series of simulations using several criteria, the results obtained were compared based on waiting, response and turnaround times respectively. The results clearly reveal that the RR scheduling algorithm returns the least average response time compared to FCFS and SJN respectively. The RR scheduling algorithm however returns the highest average turnaround time compared to the others. In terms of the average waiting time, the SJN scheduling algorithm outperforms the other two. Besides the fact that the FCFS scheduling algorithm is simple to implement, it has not shown any superior performance in terms of the three QoS criteria used in this paper.

The three scheduling algorithms used in this paper are by no means exhaustive therefore, a further work could explore other scheduling algorithms such as Priority, Multi-Level Queue, Multi-level Feedback Queue respectively among others. Moreover, the choice of using identical criteria for the three scheduling algorithms is intuitive and justifiable since a fair comparative performance informs the goal of this paper.

ACKNOWLEDGEMENTS

This is to appreciate the hand of the God for the needed inspiration as well as the provision of the wherewithal to undertake this investigative research work. Profound appreciation also goes to colleagues at the Mathematical Sciences Department, Kaduna State University Nigeria as well as those at Computer Department, Federal College of Education Pankshin who constructively criticised this work. Lastly, the cooperation and support provided by authors' families .in carrying out this research is by no means invaluable.

REFERENCES

[1] A. S. Tanenbaum, Modern Operating System, Pearson Education, Inc, 2009.

[2] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman and M. B.R., "Single-chip Microprocessor That Communicates Directly Using Light," Nature, p. 534, 2015.

[3] A. Silberschatz, B. G. Peter and G. Greg, Operating System Concepts Essentials, John Wiley & Sons, Inc, 2014.

[4] D. Dodge, A. Danko, S. Marineau-Mes, P. V. D. Veen, C. Burgess, T. Fletcher and B. Stecher, "Adaptive Partitioning for Operating System". USA Patent 8,387,052, 26 February 2013.

[5] S. Peter, J. Li, I. Zhang, D. R. Ports, D. Woos, A. Krishnamurthy, T. Anderson and T. Roscoe, "The Operating System is the Control Plane," ACM Transactions on Computer Systems, p. 11, 2016.

[6] M. A. Razzaque, M. Milojevic-Jevric, A. Palade and S. Clarke, "Middleware for Internet of Things: A Survey," IEEE Internet of Things Journal, vol. 3, no. 1, pp. 70-95, 2016.

[7] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, Wisconsin: Arpaci-Dusseau Books, 2014.

[8] N. B. Shah, K. Lee and K. Ramchandran, "When do Redundant Requests Reduce Latency?," IEEE Transactions on Communications , pp. 715-722, 2016.

[9] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea and J. Kołodziej, "Resource-aware Hybrid Scheduling Algorithm in Heterogeneous Distributed Computing," Future Generation Computer Systems, pp. 61-71, 2015.

[10] B. Fataniya and M. Patel, "Survey on Different Method to Improve Performance of the Round Robin Scheduling Algorithm," 2018.

AUTHOR

Luhutyit Peter Damuut received the B.Tech. degree in Computer Science from the Abubakar Tafawa Balewa University (ATBU), Bauchi, Nigeria, in 1999; M.Sc. degree in Computing from the Robert Gordon University (RGU), Aberdeen, UK, in 2004 and the Ph.D. degree in Computer Science from the University of Essex, UK in 2014, respectively. Currently, He is a Senior Lecturer at Kaduna State University (KASU) Nigeria. His teaching and research interests include computational intelligence, wireless sensor networks and mobile computing Dr. Damuut may be reached at luhutyit.damuutp@kasu.edu.ng.