Integrating User-Defined Priority Tasks in a Shortest Job First Round Robin (SJFRR) Scheduling Algorithm

Hannah S. Caranto
University of the Cordilleras
Governor Pack Road, Baguio City
Benguet, Philippines
anna_caranto@yahoo.com

Wyli Charles L. Olivete
University of the Cordilleras
Governor Pack Road, Baguio City
Benguet, Philippines
azeptoblade@gmail.com

Jeune Vincent D. Fernandez
University of the Cordilleras
Governor Pack Road, Baguio City
Benguet, Philippines
mcjeunefernandez@gmail.com

Cecilia Agatha R. Cabiara
University of the Cordilleras
Governor Pack Road, Baguio City
Benguet, Philippines
cabiara.ca@gmail.com

Rey Benjamin M. Baquirin
University of the Cordilleras
Governor Pack Road, Baguio City
Benguet, Philippines
reybenbaq@gmail.com

Eugene Frank G. Bayani
University of the Cordilleras
Governor Pack Road, Baguio City
Benguet, Philippines
ebayani@gbox.adnu.edu.ph

Roma Joy D. Fronda
University of the Cordilleras
Governor Pack Road, Baguio City
Benguet, Philippines
rdfronda@yahoo.com

ABSTRACT

The Shortest Job First (SJF) algorithm reduces the average waiting time of processes in a CPU environment; however, it may lead to the starvation of long processes. The Round Robin (RR) algorithm solves starvation by equally distributing processing time. This paper explores user-friendly improvements by allowing users to specify which processes should be finished first. The SJF, RR, and the user-priority SJFRR were compared using four test cases in the following criteria: context switching, average waiting time, and average turnaround time. The user-priority SJFRR was found to have reduced average waiting time and average turnaround time in all test cases except for the first. Further research is also recommended in order to find out whether the algorithm's performance is attributed to the randomized assignment of priority and to find a proper basis for user designation of priority tasks.

CCS Concepts

• **Software and its engineering → Software organization and properties → Contextual software domains→ Operating systems → Process management → Scheduling**

Keywords

Scheduling, CPU, SJF, RR

1. INTRODUCTION

The process of allocating CPU resources to processes in order to maximize CPU usage is called CPU Scheduling [1]. Many CPU scheduling algorithms have been created in order to satisfactorily perform multitasking and to fully utilize CPU resources [2].

ICCDE '20, January 4–6, 2020, Sanya, China
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7673-0/20/01…$15.00
https://doi.org/10.1145/3379247.3379291

The Round Robin algorithm uses the concept of time-sharing in order to equally distribute the same amount of processing time to all processes. The amount of processing time distributed is referred to as the quantum time, for which there is no set standard [3]. Setting the quantum time too long or too short leads to problems, such as lower CPU efficiency and poor response time [4].

Another scheduling algorithm is the Shortest Job First, in which the task with the shortest completion time is executed first; this can lead to longer processes being starved in the event that new shorter processes keep being added to the queue [5].

In order to minimize the flaws of both Round Robin and Shortest Job First algorithms, there has been much research on combining both to create a more efficient hybrid CPU scheduling algorithm [6].

This paper focuses on exploring the possibility of implementing a method to allow for user-defined tasks to be prioritized in a hybrid CPU scheduling algorithm based on both Round Robin (RR) and Shortest Job First (SJF). This proposed algorithm is referred to as user-priority Shortest Job First Round Robin or user-priority SJFRR. The user-priority SJFRR is compared to the RR and SJF scheduling algorithms in order to determine whether there is an improvement in context switching, average waiting time, and average turnaround time.

2. REVIEW OF RELATED LIT.

There are two types of scheduling algorithms: non-preemptive and preemptive. Non-preemptive algorithms, such as the First Come First Serve (FCFS) and Shortest Job First (SJF), engage with a chosen task until it is completed [7]. Contrarily, preemptive algorithms may pause in the execution of one process in order to execute another process with higher priority [8]. There are many proposed scheduling algorithms today that use a combination of non-preemptive and preemptive scheduling algorithms in order to minimize the drawbacks of the individual algorithms used [9].

Priority in scheduling refers to how quickly a task or process must be implemented. Different scheduling algorithms treat processes with different priorities. For example, in RR algorithms, tasks are given the same priority while in SJF, shorter tasks are given priority over longer ones [2]. The implementation of priority in scheduling algorithms is important since there are crucial processes that have to be executed earlier than others [10].

The Round Robin algorithm is the most frequently used scheduling algorithm due to its timesharing trait. Its popularity as a scheduling algorithm has led to much research on improving this algorithm's performance, thus explaining why there are many RR variants [11]. One of the most common ways of tweaking the RR is by changing how the time quantum is computed, such as computing the time quantum to be equal to the median of the processes in the ready queue [8, 9, 12].

There are many ways of evaluating scheduling algorithms, with different algorithms having better performance than others in certain criteria. For this paper, the algorithms used will be compared based on the following criteria: context switching, average waiting time, and average turnaround time.

Context switching or context switch refers to the process of storing the state of a CPU in order to restore it and resume execution at a later point in time [9]. Context switching can lead to wasted time and memory, so it must be minimized in an optimal scheduling algorithm [7]. CPU scheduling algorithms do not affect how long it takes to execute a process or to do input/output. It only affects the amount of time a process spends in the ready queue before it is executed, which is referred to as the waiting time [13]. Turnaround time refers to how long a process takes to complete, starting from its arrival time to its completion time [14].

3. METHODOLOGY

In this paper, the proposed algorithm combined RR and SJF. The proposed algorithm also integrates a method for users to identify tasks as priority tasks. The proposed algorithm referred to as the user-priority SJFRR algorithm can be described as follows:

1. Before the processes enter the queue, the user may define them as priority tasks.
2. Processes not identified as priority are placed in the non-priority queue and arranged in ascending order of burst time. If two or more processes have the same amount of burst time, these processes are ordered according to arrival time.
3. Processes identified as priority tasks by the user are placed in the priority queue and arranged in ascending order of burst time. If two or more processes have the same amount of burst time, these processes are ordered according to arrival time.
4. The time quantum for the RR is set as the median of the processes in each queue.
5. 1 cycle of RR is executed in the priority queue, then in the non-priority queue.
6. When the queue completes 1 cycle, the program is checked for newly arrived processes. If there is none, the two queues execute for 1 more cycle of RR. If there is, these processes undergo steps 1 to 4, before another cycle of RR is executed.
7. Repeat the process until there are no more tasks in the queue.

3.1 Pseudocode

The pseudo code used in this paper is a modified version of the pseudocode for the Shortest Remaining Burst Round Robin (SRBRR) [15], which is essentially a combination of RR and SJF. The modifications come in the form of the addition of user-identified priority, an additional queue for priority processes, and a different way of getting the time quantum for the RR.

Let TQ1 be the time quantum for the priority queue, TQ2 be the time quantum for the non-priority queue, n be the number of processes accepted as input, P be the non-priority tasks, and PT be the priority tasks.

```
1. User defines which processes are PT as it enters the
   ready queue.
2. All non-priority processes present in the ready queue are
   sorted in ascending order of burst time. Processes with
   the same burst time are sorted according to arrival time.
3. Processes marked as PT are placed in the priority queue
   and sorted in ascending order of burst time. Processes
   with the same burst time are sorted according to arrival
   time.
4. While(ready queue!= NULL)
      a. TQ1 = median of processes in priority queue
      b. TQ2 = median of processes in non-priority
         queue
5. Assign TQ to processes
      a. PTi -> TQ1
      b. Pi -> TQ2
6. If (i<n) then go to step 5.
7. If a new process has arrived,
      a. Update the counter n and go to step 1.
      b. End of while
8. Average waiting time, average turnaround time and
   number of context switches are calculated.
9. End
```

3.2 Test Cases

All processes in the test cases were assumed to be: 1) executed in a single processor, 2) CPU bound, and 3) have known burst time and arrival time. The benchmark algorithms used are the SJF and RR algorithms. In all test cases, arrival time was set to 0.

The proposed algorithm and benchmark algorithms were compared using average turnaround time (ATAT), average waiting time (AWT), and the number of context switches (CS). ATAT is computed as the mean time from submission to completion of processes. AWT is computed as the average length of time processes are waiting in the queue before execution. CS is counted as the number of times the CPU switches from a process to another. The lower the values for ATAT, AWT, and CS, the better the performance for the algorithm.

3.2.1 Test Case 1

Assume five processes, presented in Table 1, arriving with random burst time. The tasks are randomly assigned as priority and non-priority tasks.

Table 1. Test Case 1

Process	Burst Time	Priority
P_1	5	Yes
P_2	29	No
P_3	100	Yes
P_4	52	Yes
P_5	76	No

3.2.2 Test Case 2

Assume five processes arriving with longer priority processes arriving earlier while shorter non-priority tasks arrive later.

Table 2. Test Case 2

Process	Burst Time	Priority
P_1	69	Yes
P_2	78	Yes
P_3	48	No
P_4	2	No
P_5	12	No

3.2.3 Test Case 3

Assume five processes arriving with shorter priority processes arriving later while longer non-priority tasks arrive earlier.

Table 3. Test Case 3

Process	Burst Time	Priority
P_1	80	No
P_2	49	No
P_3	30	Yes
P_4	15	Yes
P_5	22	Yes

3.2.4 Test Case 4

Assume five processes arriving with longer processes arriving earlier. Tasks are randomly assigned as priority/non-priority tasks.

Table 4. Test Case 4

Process	Burst Time	Priority
P_1	78	Yes
P_2	86	No
P_3	44	Yes
P_4	10	Yes
P_5	28	No

4. FINDINGS

4.2 Results

4.2.1 Results on Test Case 1

Table 5. Results Test Case 1

Algorithm	ATAT	AWT	CS
user-priority SJFRR	134.4	82.0	9
RR	124.2	71.8	5
SJF	109.8	57.4	5

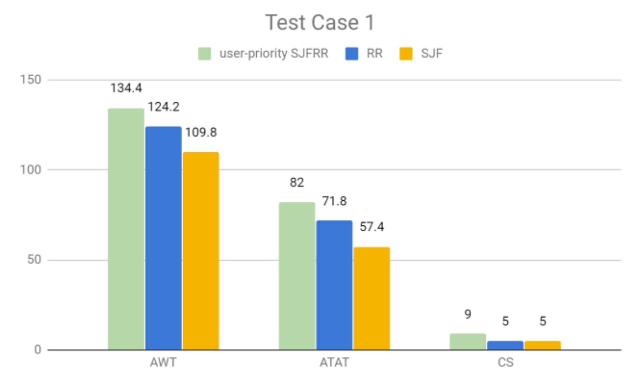

Figure 1. Comparative Results for Test Case 1

The user-priority SJFRR had the highest ATAT, AWT, and CS. SJF had the lowest ATAT and AWT, with RR and SJF sharing an equal CS.

4.2.2 Results on Test Case 2

Table 6. Results Test Case 2

Algorithm	ATAT	AWT	CS
user-priority SJFRR	147.4	105.6	9
RR	167.2	125.4	7
SJF	83.6	41.8	5

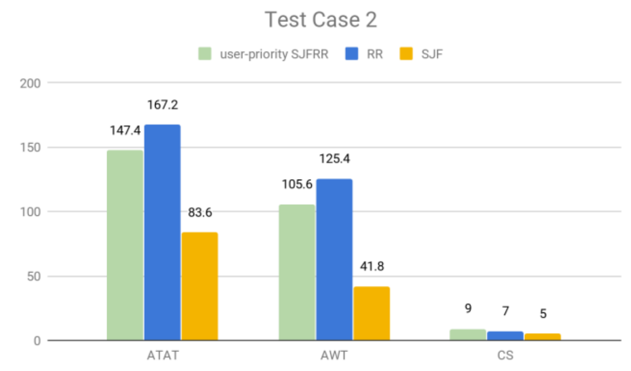

Figure 2. Comparative Results for Test Case 2

The user-priority SJFRR was between RR and SJF in terms of ATAT and AWT, but had the highest CS.

4.2.3 Results on Test Case 3

Table 7. Results Test Case 3

Algorithm	ATAT	AWT	CS
user-priority SJFRR	89.2	50.0	8
RR	138.8	99.6	8
SJF	86.2	47.0	5

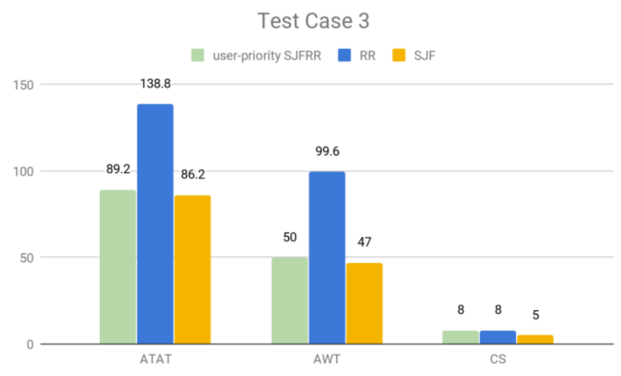

Figure 3. Comparative Results for Test Case 3

The user-priority SJFRR was between RR and SJF in terms of ATAT and AWT, but had the same CS with RR.

4.2.4 Results on Test Case 4

Table 8. Results Test Case 4

Algorithm	ATAT	AWT	CS
user-priority SJFRR	125.6	76.4	9
RR	178.8	129.6	7
SJF	107.2	58.0	5

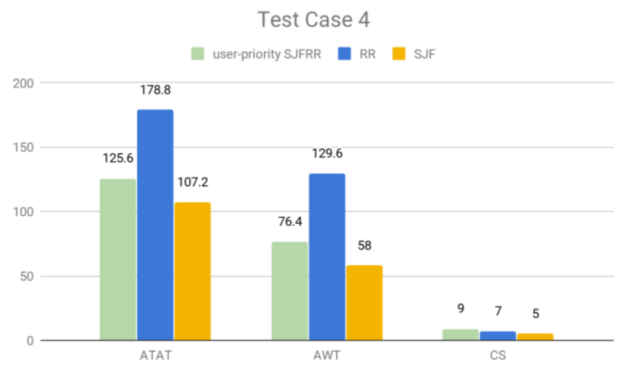

Figure 4. Comparative Results for Test Case 4

The user-priority SJFRR was between RR and SJF in terms of ATAT and AWT, but had the highest CS.

4.2.5 Average Results Across All Test Cases

Table 9. Average Results Across All Test Cases

Algorithm	ATAT	AWT	CS
user-priority SJFRR	124.15	78.5	8
RR	152.25	106.6	6.75
SJF	96.7	51.05	5

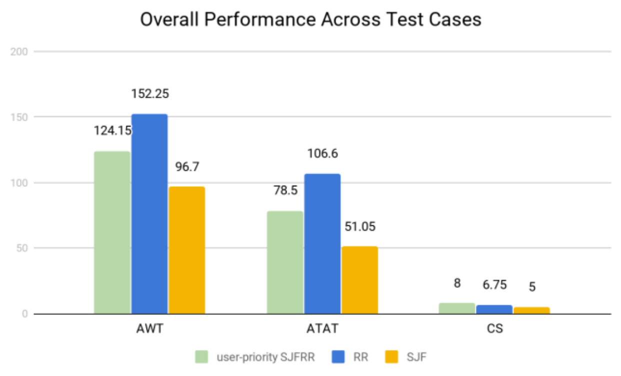

Figure 5. Comparative Results Across All Test Cases

For three out of four test cases the user-priority SJFRR had an ATAT and AWT in between the benchmark algorithms. For three out of four test cases, the user-priority SJFRR had the most CS. For one out of four test cases, the user-priority SJFRR had the highest ATAT, AWT, and CS. To summarize the results, the average scores of each algorithm for each criterion were computed, The table below shows the average ATAT, AWT and CS for each algorithm for all the test cases.

5. DISCUSSION

The user-priority SJFRR can be a better alternative to the RR algorithm. It performed well when priority was assigned consistently, as evidenced by how the user-priority SJFRR algorithm performed last in test case 1. This could imply that the algorithm works best when the user does not assign random priority to processes. However, further research would be needed to conclusively prove this.

While SJF may have consistently performed better than both RR and user-priority SJFRR, it is not as applicable realistically compared to the two as a constant input of short processes can lead to the starvation of long processes in the ready queue. This implies that while SJF is better in cases where there are no new processes arriving in the middle of execution, such as seen in the test cases used in this paper, it would not be suitable for use in real-life applications.

Based on the average overall performance, the user-priority SJFRR had superior performance in terms of AWT and ATAT compared to the RR algorithm. The RR had better performance in terms of CS, but the increased CS of the user-priority SJFRR can be explained by the fact that two queues, the non-priority and priority, were used.

6. CONCLUSIONS

Based on the test cases, the user-priority SJFRR can reduce average waiting time and average turnaround time. However, the algorithm was not consistent in terms of context switching. There was also no conclusive evidence as to why SJF performed better in all test cases. Additional different test cases are recommended in order to find the cause. Further research is also recommended in order to find out whether the algorithm's performance is attributed to the randomized assignment of priority and to find a proper basis for user designation of priority to tasks.

7. ACKNOWLEDGMENTS

The researchers would like to thank the staff of the College of Information Technology and Computer Science of the University of the Cordilleras for their support.

8. REFERENCES

[1] Goel, N., Garg, R. B. 2012. A comparative study of CPU scheduling algorithms. *International Journal of Graphics & Image Processing.* 2, 4 (Nov. 2012), 245-251. Retrieved from https://www.researchgate.net/publication/249645533_A_Comparative_Study_of_CPU_Scheduling_Algorithms.

[2] Rajput, I. S., Gupta, D. 2012. A priority based round robin CPU scheduling algorithm for real time systems. *International Journal of Innovations in Engineering and Technology.* 1, 3 (Oct. 2012), 1-11. Retrieved from https://www.idc-online.com/technical_references/pdfs/information_technology/A%20Priority%20based.pdf.

[3] Siregar, M. U. 2012. A new approach to CPU scheduling algorithm: Genetic round robin. *International Journal of Computer Applications.* 47, 19 (Jun. 2012), 18-25. DOI= https://doi.org/10.5120/7296-0459.

[4] Singh, P., Singh, V., Pandey, A. 2014. Analysis and comparison of CPU scheduling algorithms. *International Journal of Emerging Technology and Advanced Engineering.* 4, 1 (Jan. 2014), 91-95. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.450.7715&rep=rep1&type=pdf.

[5] Shahzad, B., Afzal, M. T. 2006. Optimized solution to shortest job first by eliminating the starvation. *Proceedings of the 6th Jordanian International Electrical & Electronic Engineering Conference.* (Mar. 2006). DOI= http://doi.acm.org/10.1145/161468.16147.

[6] Mishra, M. K., Rashid, F. 2014. An improved round robin CPU scheduling algorithm with varying time quantum. *International Journal of Computer Science, Engineering and Applications.* 4, 4 (Aug. 2014), 1-8. DOI= 10.5121/ijcsea.2014.4401.

[7] Mondal, R. K., Nandi, E., Sarddar, D. 2015. Load balancing scheduling with shortest load first. *International Journal of Grid Distribution Computing.* 8, 4 (Aug. 2015), 171-178. DOI= http://dx.doi.org/10.14257/ijgdc.2015.8.4.17.

[8] Kishor, L., Goyal, D. 2013. Time quantum based improved scheduling algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering.* 3, 4 (Apr. 2013), 955-962. Retrieved from https://www.researchgate.net/publication/321482375_Enhanced_round_robin_CPU_scheduling_with_burst_time_based_time_quantum.

[9] Manuel, J. I., Baquirin, R. B., Guevera, K. S., Tandingan, D., Jr. 2019. Fittest job first dynamic round robin (FJFDRR) scheduling algorithm using dual queue and arrival time factor: A comparison. *IOP Conf. Ser.: Mater. Sci. Eng. 482 012046.* 482, 1 (2019), 1-9. DOI= 10.1088/1757-899X/482/1/012046.

[10] Ghanbari, S., Othman, M. 2012. A priority based job scheduling algorithm in cloud computing. *International Conference on Advances Science and Contemporary Engineering.* 50 (Jan. 2012), 778-785. DOI= 10.1016/j.proeng.2012.10.086.

[11] Yeboah, T., Odabi, I., Hiran, K.K. 2015. An integration of round robin with shortest job first algorithm for cloud computing environment. *International Conference On Management, Communication and Technology.* 3, 1 (Apr. 2015), 245-251. Retrieved from http://www.ijictm.org/admin/html/mail/attach/2015-08-07-09-42-43.pdf?fbclid=IwAR2wMfoWXjfRMcRyoKZKCBcPCdS5vHqcsMHp9Oa2qS7VvYXBUTmi5zbaRvw.

[12] Matarneh, R. J. 2009. Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes. *American Journal of Applied Sciences.* 6, 10 (Oct. 2009), 1831-1837. DOI= 10.3844/ajassp.2009.1831.1837.

[13] Yadav, R. K., Mishra, A. K., Prakash, N., Sharma, H. 2010. An improved round robin scheduling algorithm for CPU scheduling. *International Journal on Computer Science and Engineering.* 2, 4 (Jul. 2010), 1064-1066. Retrieved from https://www.researchgate.net/profile/Navin_Prakash2/publication/49619229_An_Improved_Round_Robin_Schedduling_Algorithm_for_CPU_Scheduling/links/558d70a008ae1e1f9bab15e4/An-Improved-Round-Robin-Schedduling-Algorithm-for-CPU-Scheduling.pdf.

[14] Helmy, T., Dekdouk, A. 2007. Burst round robin as a proportional-share scheduling algorithm. *International Journal of Graphics & Image Processing.* 2, 4 (Jan. 2007), 245-251. Retrieved from https://eprints.kfupm.edu.sa/1462/1/d3_s18_p3_1569045631.pdf.

[15] Mohanty, R., Behera, H. S. Patwari, K. M., Dash, M. 2010. Design and performance evaluation of a new proposed shortest remaining burst round robin (SRBRR) scheduling algorithm. *Proceedings of International Symposium on Computer Engineering & Technology.* (2010). Retrieved from https://www.academia.edu/1548400/Design_and_Performance_Evaluation_of_a_new_proposed_Shortest_Remaining_Burst_Round_Robin_SRBRR_scheduling_algorithm.