International Journal of New Computer Architectures and their Applications (IJNCAA) 8(3): 142-147
The Society of Digital Information and Wireless Communications, 2018 ISSN 2220-9085 (Online); ISSN 2412-3587 (Print)

Proportional Weighted Round Robin: A Proportional Share CPU Scheduler in Time Sharing Systems

Samih M. Mostafa
Math. Dept., Faculty of Science, South Valley University, Qena, Egypt.
samih_montser@sci.svu.edu.eg

ABSTRACT

In time sharing systems, many processes reside in the ready queue and compete for execution by the processor. Therefore, scheduling these processes on CPU as it can run only one process at a time is needed. In this paper, a modified version of Round-Robin (RR) called proportional weighted round robin (PWRR) is proposed. The proposed scheduler is a proportional share scheduler designed explicitly for time sharing systems. The proposed scheduler improves some scheduling criteria by minimizing turnaround times, waiting times, and context switches for the running processes. A threshold is considered to determine whether the system cannot take away the CPU from the process until it finishes or the process is interrupted due to the expiration of its time slice assigned by the RR policy. According to evaluation results, the proposed scheduler minimizes some scheduling criteria (turnaround times, waiting times, and context switches in this context)..

KEYWORDS

Process scheduling, Time sharing system, Round Robin, CPU scheduler, Round Robin.

1 INTRODUCTION

1.1 Overview

Allowing multiple processes to run concurrently in a system and share resources (e.g., CPU time) is a recent architectural technique that improves resource utilization. Such kind of systems is called time sharing systems, in which, multiple processes in memory, all compete for execution. The main objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. CPU utilization is an important concern that arose due to time sharing. So, the issue is to schedule the processes such that CPU does not sit idle. Since CPU can run a single process at a time, other processes must wait. When multiple processes are running on a single CPU, CPU time needs to be shared among all these processes. However, it seems difficult to manage this time among all the processes manually. The software that selects the processes to be scheduled according to a specific algorithm is called scheduler [1]. The scheduler that selects a process from the ready queue and dispatches to the CPU is called short term scheduler. Short term scheduler is very frequent; the process stopped temporarily from execution and is sent to the queue and competes again for execution [13].

1.2 Motivation

The behavior of a system may depend on various general criteria such as waiting time, the total time spent in the ready queue by a process, turnaround time, the time since the process was submitted to the time of completion, and so on [14]. The behavior of the system must be optimized. Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another. Choosing which scheduling algorithm is used for a specific system depends on which characteristics are used for comparison between CPU-scheduling algorithms. Many criteria have been suggested for comparing CPU-scheduling algorithms. The CPU scheduler is designed to allocate the resources, CPU time in this context, to all processes. The scheduling problem can be stated shortly as: which process should be moved when, to where and for how long it will run [2, 3, 4].

In general form, scheduling algorithms have been found to be NP-complete (i.e., it is believed that there is no optimal polynomial-time algorithm for them [4, 5]). Good scheduling performance may lead to a give-and-take situation, such that improving performance in one trend hurts performance in another [2]. The proposed work in this paper improves the system performance by minimizing user-based

scheduling criteria (i.e., waiting time, turnaround time, and context switches). There are many different CPU scheduling algorithms. First Come First served (FCFS) algorithm is the simplest one. The policy of FCFS is managed with a First In First Out (FIFO) queue. Shortest Job First (SJF) is a different approach to CPU scheduling. The CPU is assigned to the process that has the smallest next CPU burst time. Another algorithm called Round-Robin (RR) is designed especially for time sharing systems. It is similar to FCFS scheduling, but preemption is added by defining a small unit of time, called time slice, to enable the system to switch between processes. Efficiency and effectiveness of RR are arising from its low scheduling overhead of O(1), which means scheduling the next process takes a constant time [6, 7, 8].

1.3 Problem Statement

The most important issue in RR policy is choosing the time slice. The size of time slice affects the performance of the system [15]. If the size of the time slice is too long, this will cause convoy effect (i.e., processes wait for the one big process to get off the CPU). On the other hand, if the size of the time slice is too short, more context switches occurs. Context switch time is pure overhead, because the system does no useful work while switching. Hence, the performance of the system will be degraded [9]. Allowing short processes to get more CPU time gives a share in decreasing context switches overhead. It can be concluded that the time slice must be selected in a balanced range (i.e., not be as too short or too large).

1.4 Research Contribution of this Paper

In this paper, a proportional share CPU scheduler is proposed to optimize the performance of the system by reducing some user based scheduling criteria. In this paper, the time slice assigned to a process will be proportional to its burst time, and under a predefined condition short process will gain more CPU time. In this work, a modification is implemented to RR. The modification is based on changing the time slice of each process depending on its burst time. The improvement of the proposed scheduler over RR scheduling is experimentally demonstrated, and experimental results show that the proposed scheduler can minimize the scheduling criteria.

The rest of this paper is structured as follows: section 2 presents the related research. Section 3 discusses the proposed scheduler PWRR. The experimental results are given in section 4. Section 5 presents the conclusion.

2 RELATED RESEARCHES

Various scheduling algorithms are discussed in this section.

2.1 Burst Round Robin Algorithm:

Burst Round Robin (BRR) [10] is a weighting algorithm based on burst times of processes. The higher weight, the more time slice. Short processes will leave the CPU earlier because it will be given more CPU time.

2.2 Changeable Time Quantum

CTQ algorithm [2] finds a value of time slice that gives the minimum waiting time. Depending on the burst times of running processes, the value of time slice changes in each round.

2.3 Enhanced Round Robin Algorithm

Tajwar et al., [11] proposed a dynamic round robin algorithm. In every round, the proposed algorithm assigns a new time slice equals to the mean burst time of all running processes.

2.4 Dynamic Average Burst Round Robin

DABRR [12] is a dynamic scheduling algorithm based on RR. The processes are sorted in an ascending order based on burst times.

3 THE PROPOSED ALGORITHM

3.1 PWRR Definitions

PWRR is a proportional CPU scheduler that assigns a time slice to each process proportional to its burst time. The terminology list used in this work is defined in Table i.

Table i. List of terminology.

PID	Process identification
n	Number of processes
BT[i]	Original burst time of process i
BT[i][r]	Burst time of process i in round r
W[i][r]	Weight of process i in round r
RR_TQ	Time quantum assigned by RR policy

NTQ[i][r]	New time quantum of process i in round r
TSH	Predefined threshold
WT[i]	Waiting time of process i
TAT[i]	Turnaround time of process i
AVGWT	Average waiting time
AVGTAT	Average turnaround time
CS[i]	Context switches

The design issues with the functioning of PWRR are discussed as following:

i. The queue is a FIFO queue.
ii. The weight of the process equals the process's burst time divided by summation of all burst times in the queue:

$$W[i][r] = \frac{BT[i][r]}{\sum_{i=1}^{n} BT[i]}, BT[i][1] = BT[i] \quad (1)$$

iii. The process's time slice:

$$NTQ[i][r] = (1 - W[i][r]) \times RR\_TQ \quad (2)$$

iv. The burst time:

$$BT[i][r] = BT[i][r-1] - NTQ[i][r-1] \quad (3)$$

v. The burst time of the selected process to be executed is compared with time slice assigned by the RR policy under a predefined threshold (the value of the threshold is an implementation choice).
vi. If the condition in (v) is true, this short process will complete execution until termination.
vii. If the condition in (v) is false, the running process will be interrupted by the expiration of time slice obtained from (iii) and is placed back at the tail of the ready queue.

Figure 1shows the flow chart of the proposed scheduler.

3.2 Illustrative Example

The following example simplifies the proposed consideration. Four processes arrive at the same time, each with burst time given in milliseconds (Table ii).

Table ii. Set of processes with different CPU burst times.

PID	BT[i]
P1	21
P2	20
P3	6
P4	13

a. Under RR

The following Gantt chart shows the result under RR (RR_TQ = 10ms). The scheduling criteria calculated are shown in Table iii.

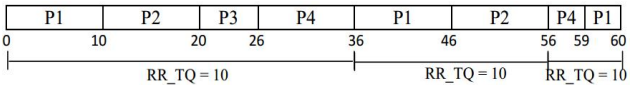


Table iii. The scheduling criteria under RR.

Process	BT[i]	WT[i]	TAT[i]	CS[i]
P1	21	39	60	2
P2	20	36	56	1
P3	6	20	26	0
P4	13	46	59	1
		AVGWT = 35.25 ms	**AVGTAT = 50.25 ms**	**No. CS = 4**

b. Under PWRR

In this work, the threshold is taken to be $TSH = 0.3 \times RR\_TQ$. Table iv shows the weight of the processes and the time slice for each process.

Table iv. Processes, their weights and time slices under PWRR in first round.

PID	BT[i]	BT[i][1]	W[i][1]	NTQ[i][1]
P1	21	21	0.35	6.5
P2	20	20	0.333333333	6.666666667
P3	6	6	0.1	9
P4	13	13	0.216666667	13
sum	**60**			

Process P4 achieves the condition in v, then it will continue execution until termination. The following Gantt chart shows the result under proposed scheduler:

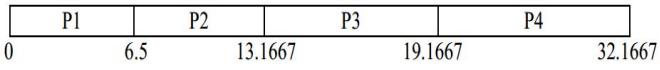


Table v shows the burst times in the second round and the time slices assigned for each process.

Table v. Processes, their weights and time slices under PWRR in second round.

PID	BT[i]	BT[i][2]	W[i][2]	NTQ[i][2]
P1	21	14.5	0.241667	7.58333
P2	20	13.3333	0.222222	13.3333
P3	6	Terminated		
P4	13	Terminated		
sum	**60**			

Process P2 achieves the condition in v, then it will continue execution until termination.

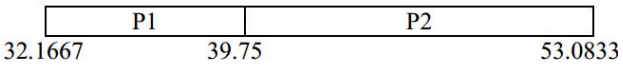
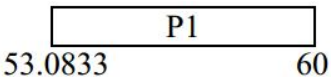


Table vi shows the burst times and the time slices in the third round.

Table vi. Processes, their weights and time slices under PWRR in third round.

PID	BT[i]	BT[i][3]	W[i][3]	NTQ[i][3]
P1	21	6.91667	0.115278	6.91667
P2	20	Terminated		
P3	6	Terminated		
P4	13	Terminated		
sum	**60**			



The scheduling criteria calculated are shown in Table vii.

Table vii. The scheduling criteria under PWRR.

Process	BT[i]	WT[i]	TAT[i]	CS[i]
P1	21	39	60	2
P2	20	33.0833	53.0833	1
P3	6	13.1667	19.1667	0
P4	13	19.1667	32.1667	0
		AVGWT = 26.1042 ms	**AVGTAT = 41.1042 ms**	**No. CS = 3**

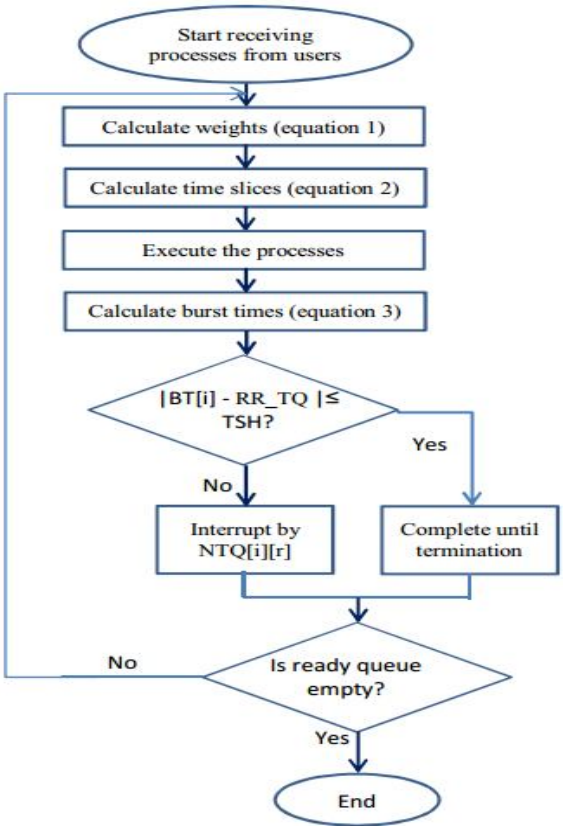


Figure 1.Proposed scheduler flow chart.

4 EXPERIMENTAL RESULTS

To demonstrate the effectiveness of PWRR, some experimental data quantitatively comparing PWRR performance against RR considered on different combinations of number of processes in two scenarios. In the first scenario, the processes arrive at the same time. In the second scenario, the processes arrive at different times. The experiments were repeated many times for each scenario. In each scenario, there are two cases. In the first case, the processes are running in the order they come. The scheduler is called US_PWRR. In the second case, the processes are sorted in an ascending order. The process sets were generated by process generator routine. Each process has its id, arrival time, and burst time. The process arrival was modeled as a Poisson random process. The scheduler is called S_PWRR. The results show a significant improvement in terms of minimizing scheduling criteria (average waiting time, average turnaround time, and number of context switches). Figures 2, 3 and 4 show the average waiting times, turnaround times and number of context switches comparisons respectively in first scenario. Figures 5, 6 and 7 show the average waiting times, turnaround times and number of context switches comparisons in second scenario respectively.

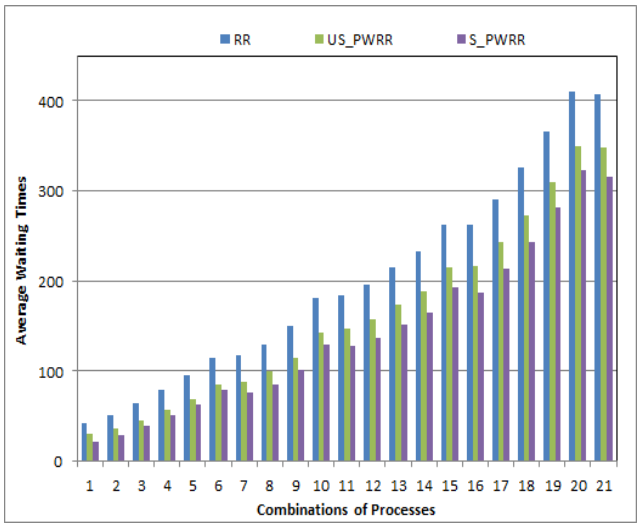


Figure 2: Average waiting times comparison in first scenario.

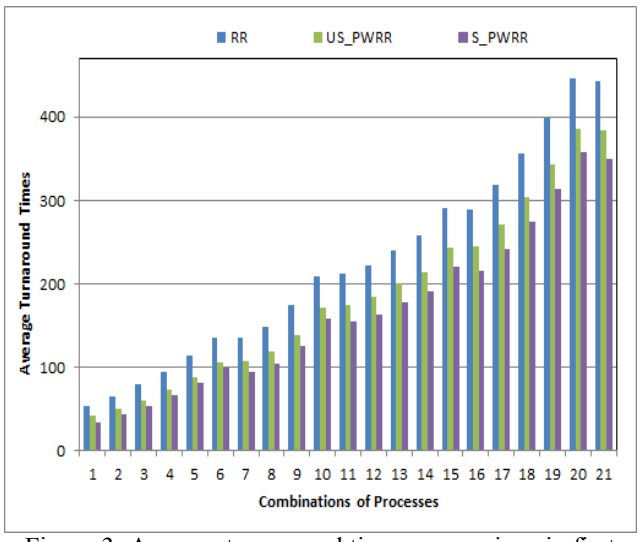


Figure 3: Average turnaround times comparison in first scenario.

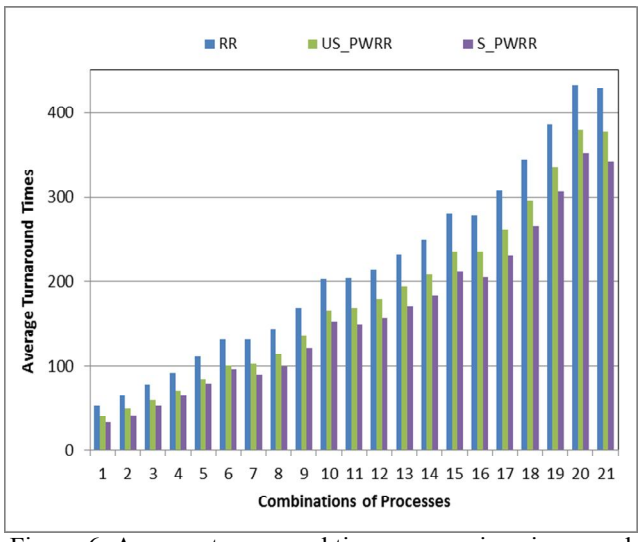


Figure 6: Average turnaround times comparison in second scenario.

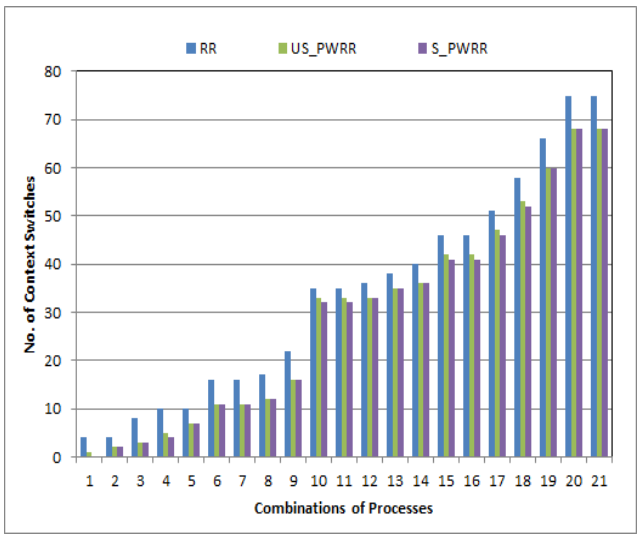


Figure 4: Number of context switches comparison in first scenario.

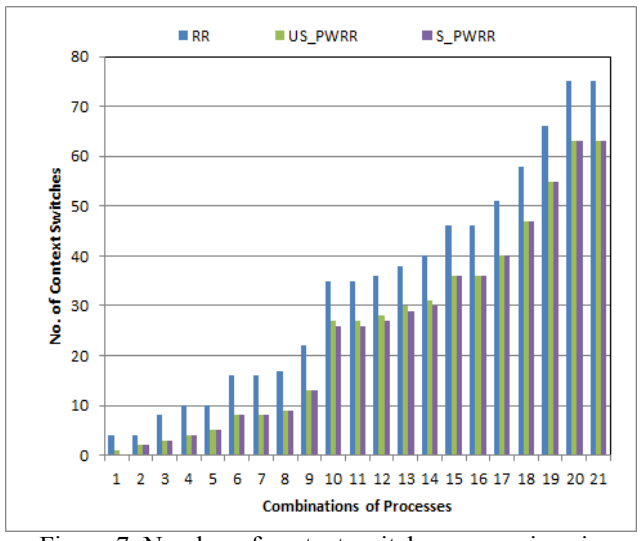


Figure 7: Number of context switches comparison in second scenario.

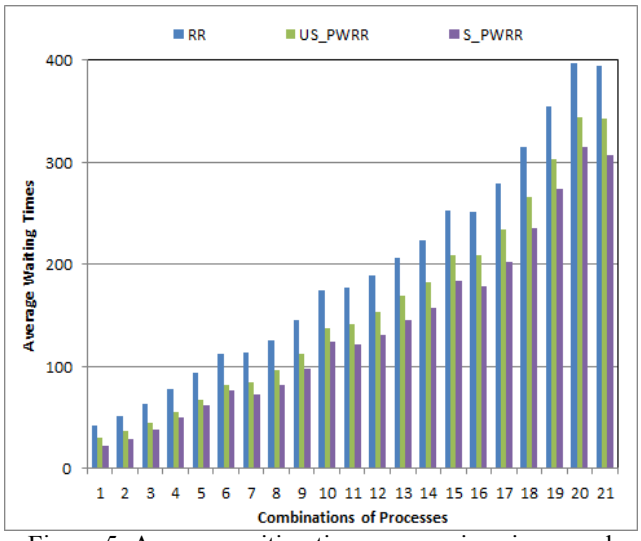


Figure 5: Average waiting times comparison in second scenario.

CONCLUSION

In this paper, a novel CPU scheduler has been proposed to benefit from low scheduling overhead of RR algorithm. The proposed scheduler based on RR scheduler because RR is designed especially for time sharing systems. The proposed algorithm assigns new time slice to each process proportional to its burst time. Short process will be given a chance to get more CPU time. The results show a significant improvement in terms of minimizing scheduling criteria. The simulation study showed that the performance of the proposed algorithm is higher than RR in general time sharing systems.

REFERENCES

1. A. Silberschatz, P. B. Galvin, and G. Gange, "Operating Systems Concepts", John Wiley and Sons.9th Ed, International Student Version, (2013).
2. S. M. Mostafa, S. Z. Rida, and S. H. Hamad. "Finding Time Quantum of Round Robin CPU Scheduling Algorithm in General Computing Systems using Integer Programming", International journal of Research and Reviews in Applied Sciences. (October 2010).
3. S. M. Mostafa and S. Kusakabe, "Achieving Better Fairness for Multithreaded Programs in Linux using Group Threads Scheduler", 2013 International Workshop on ICT at Beppu, Kamenoi Hotel, Beppu, Oita, Japan, 12th to 14th. (December 2013).
4. J. D. Ullman, "Polynomial complete scheduling problems", In Proc. of the fourth ACM symposium on Operating system principles, pp. 96 – 101, 1973.
5. K. Ramamritham and J. A. Stankovic, "Scheduling algorithms and operating systems support for realtime systems", Proceedings of the IEEE, vol. 82, no. 1, (1994).
6. L. Abeni, G. Lipari, and G. Buttazzo, "Constant bandwidth vs. proportional share resource allocation", In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Florence, Italy, June 1999.
7. B. Caprita, W.C. Chan, and J. Nieh, "Group Round-Robin: Improving the Fairness and Complexity of Packet Scheduling", Technical Report CUCS-018-03, Columbia University, June 2003.
8. B. Caprita, W.C. Chan, J. Nieth, C. Stein, and H. Zheng, "Group ratio round-robin: O(1) proportional share scheduling for uni-processor and multiprocessor systems", In USENIX Annual Technical Conference, 2005.
9. N. Chauhan, "Principles of Operating Systems", Oxford University Press. (2014).
10. T. Helmy and A. Dekdouk, "Burst round robin as a proportional-share scheduling algorithm", In Proceedings of The fourth IEEE-GCC Conference on Towards Techno Industrial Innovations, pp. 424-428, 11-14, at the Gulf International Convention Center, Bahrain. (2007).
11. M. M. Tajwar, Md. N. Pathan, L. Hussaini, and A. Abubakar, "CPU Scheduling with a Round Robin Algorithm Based on an Effective Time Slice", Journal of Information Processing Systems (JIPS) vol. 13, no. 4, pp. 941 – 950. (August 2017).
12. A. R. Dash and S. K. Samantra, "An optimized round Robin CPU scheduling algorithm with dynamic time quantum", International Journal of Computer Science, Engineering and Information Technology (IJCSEIT) 5 (1):7–26. doi:10.5121/ijcseit.2015.5102. (2016).
13. S. M. Mostafa, S. Z. Rida & S. H. Hamad, "Improving Scheduling Criteria of Preemptive Processes Scheduled Under Round Robin Algorithm Using Changeable Time Quantum", Journal of Computer Science & Systems Biology (JCSB). JCSB, vol. 4.4-04-071. (2011).
14. S. M. Mostafa and S. Kusakabe, "Towards Maximizing Throughput for Multithreaded Processes in Linux", International Journal of New Computer Architectures and their Applications (IJNCAA) 4(2): 70-78, (2014).
15. S. Elmougy, S. Sarhan, and M. Joundy, "A novel hybrid of shortest job first and round Robin with dynamic variable quantum time task scheduling technique", J Cloud Computing 6(1):12. (2017).