Information Engineering Express
International Institute of Applied Informatics
2019, Vol. 5, No.1, 11-18

An Adjustable Round Robin Scheduling Algorithm in Interactive Systems

Samih M. Mostafa*, Hirofumi Amano†

Abstract

CPU scheduling is considered as the basic job within the operating system. Scheduling criteria including waiting time, context switches and others have been suggested for comparing CPU scheduling algorithms. In this paper, a modified version of Round Robin algorithm is introduced as an attempt to combine the advantageous of low scheduling overhead of Round Robin and favor short process to minimize the average waiting time and number of context switches of running processes in interactive (time-shared) systems. A threshold is considered to determine whether the running process will be interrupted because of the expiration of its time slice specified by the Round Robin policy or will continue execution until termination. Derived results show that the suggested modification minimizes the average waiting time and number of context switches compared to Round Robin algorithm.

Keywords: CPU scheduling, interactive system, time-sharing, Round Robin.

1 Introduction

CPU scheduling problem decides which of the processes in the ready queue to be allocated the CPU [1,2]. Scheduling algorithms are the mechanism by which a resource is assigned to a client. In this paper, the concept of a resource is restricted to CPU time and clients to processes. The decision of scheduling refers to the concept of selecting the next process for execution.

There are many different CPU-scheduling algorithms which have different properties, and the choice of a particular algorithm may favor one class of processes over another, therefore, selecting the appropriate algorithm is an issue. To choose which algorithm to be used in a particular situation, the properties of the various algorithms must be considered. Defining the criteria to be used in the selection is the first and important aspect to be considered [2–7]. Waiting time (i.e., the total time the process spent in the ready queue) is an important criterion to be minimized.

It is necessary to saving the context of a running process when it is interrupted or waits for an I/O device. This means that the current context of a process always resides in its process control block (PCB). For simplicity, whenever the running process stops, its context is saved in its PCB and the context of other scheduled process from its PCB is loaded. This is known as context switching.

*Mathematics Department, Faculty of Science, SVU University, Qena, Egypt
†Research Institute for Information Technology, Kyushu University, Japan

However, saving the context of the current process and loading the context of other process take some time. It is obvious that saving and loading the registers and other information will consume some time, which is known as context switch time. However, during this context switch time, the current process is not running and no new process has been scheduled for execution, the processor is idle during the context switch time. Therefore, the context switch is a pure overhead because the CPU is not doing any execution during this time. More context switches lead to more overheads resulting in less throughput and reduce CPU utilization. In this paper, a modified version of RR is proposed in favor of minimizing average waiting time and number of context switches.

Round Robin scheduling algorithm is more appropriate and designed especially for a time-shared (interactive) system. It is similar to First Come First Served (FCFS) scheduling algorithm, but preemption is added to enable the system to switch between processes. RR algorithm allocates the CPU to the first process in the ready queue for s time units, where s is the time slice (time slice and time quantum are used interchangeably in this text) [1,8]. After s time units, if the process has not relinquished the CPU it is preempted, and the process is put at the tail of the ready queue which is treated as First In First Out (FIFO) queue of processes. The length of the time slice is generally from 10 to 100 milliseconds. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after one time slice, and dispatches the process.

One of two things will then happen. The process releases the CPU voluntarily if its burst is less than one time quantum. The scheduler will then proceed to the next process in the ready queue. If the CPU burst of the currently running process is longer than one time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will occur, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue. The average waiting time under the RR policy is often long.

2 Related research

Scheduling algorithms have long been studied in OSs. The important property of these algorithms is to meet scheduling criteria. In this section related researches have been discussed.

2.1 Burst Round Robin Algorithm:

Helmy and Dekdouk [9] proposed Burst Round Robin (BRR), a proportional-share scheduling algorithm. BRR is a weighting adjustment technique for round robin for processes. The higher process weights mean relatively higher time quantum.

Pros: Processes that are close to their completion will get more chances to complete and leave the ready queue. This will reduce the number of processes in the ready queue by knocking out short processes relatively faster in a hope to increase the throughput and reduce the average waiting time.

Cons: The amount of context switches is 50.8% larger than RR, which is considered significantly large.

2.2 Changeable Time Quantum

Samih et al. [1] proposed a new CPU local scheduling based on RR, Changeable Time Quantum (CTQ). The authors proposed a method using integer programming to solve equations that decide a value of time slice that is neither too large nor too small.

Pros: CTQ combines the benefit of low overhead round-robin scheduling with low average response time and low average waiting time.

Cons: It doesn't have a predefined threshold to determine the length period in which the time slice exists. If the processes come in an ascending order, CTQ behaves somewhat similar to Shortest Job First algorithm (SJF), in other cases it behaves somewhat similar to FCFS.

2.3 Enhanced round robin algorithm

Tajwar et al. [10] presented a new effective round robin CPU scheduling algorithm. The effectiveness is represented in the fact that their algorithm depends on assigning time slice for running processes in each round dynamically. Their scheme is based on dynamic allocation of the time slice. The time slice was set to the mean burst time for all running processes. In the subsequent rounds, a new time slice was set to the average of the remaining burst times of the processes.

Pros: The new time slice depends on the burst times of the running processes (i.e., it is not fixed) achieving improved performance in terms of average turnaround time, average waiting time, and context switches.

Cons: The new time slice is equal to the arithmetic mean of burst times of the processes. This new time slice may be too long in some cases increasing response time. In addition, it does not consider the benefit of getting rid of short processes.

Next section discusses the proposed algorithm which behaves similar to these related researches in making the new time slice dynamic for each process. Also, it reduces the number of processes in the ready queue by knocking out short processes. It does not classify the processes into categories as in BRR algorithm. In contrast to CTQ, the proposed algorithm defined a threshold to determine the length of the time slice. Also, the proposed algorithm does not depend on central tendency (e.g., arithmetic mean, median, mode etc.) which may increase response time.

3 The proposed Algorithm

RR is widely used in modern general purpose time sharing OSs like Linux, BSD and Windows and gives better responsiveness; however the average waiting is high. The proposed algorithm combines the low scheduling overhead of RR of O(1) [11–13], which means that scheduling the next process takes constant time and favor short process to minimize the average waiting time and amount of context switches. Short process will be given more time and leave the system earlier.

To achieve this, a threshold is considered to determine whether the running process will be interrupted because of the expiration of its time slice specified by the Round Robin policy or will continue executing until termination. In any round in the circular queue, if the burst time of a process is greater or less than the RR time quantum by the predefined threshold, then this process will complete execution and leave the ready queue.

3.1 Proposed algorithm definition

Table I defines the terminology list used in this work.

TABLE I. LIST OF TERMINOLOGY.

PID	Process identification
n	Number of processes
BT[i]	Burst time of process i
TQ	Time quantum assigned by RR policy
TQ[i]	New time quantum of process i
TSH	Predefined threshold
WT[i]	Waiting time of process i
AVGWT	Average waiting time
CS	Context switches

The following condition determines whether the running process will be interrupted because of the expiration of its time slice specified by the Round Robin policy or will continue executing until termination. The value of TSH is an implementation choice

If (abs(BT[i] - TQ) <= (TSH× TQ))

TQ[i] = BT[i]

Else

TQ[i] = TQ

Following is the algorithm of the proposed method.

Algorithm: proposed method

1. **Initialization**
   - Receive processes.
   - Put processes in the ready queue as FIFO.
2. **Assigning time slice**
   - If (abs(BT[i] - TQ) <= (TSH× TQ)).
     - TQ[i] = BT[i]
     - Run and leave the queue.
   - Else
     - TQ[i] = TQ
     - Run for the TQ and put at the tail of the queue.
   - Calculate new burst time for the survived processes.
   - Repeat for all processes.

3.2 Illustrative example

To simplify the proposed consideration, consider the following set of processes that arrive at the same time, with the length of the CPU burst given in milliseconds:

TABLE II. SET OF PROCESSES WITH DIFFERENT CPU BURST TIMES.

PID	BT
P1	14
P2	13
P3	12
P4	10
P5	11

i. Under RR:

The following Gantt chart shows the result under RR (time quantum = 10tu):

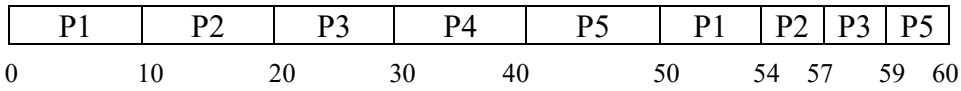

Table III shows the CPU consumption and the next process selected. The scheduling criteria calculated are shown in Table IV.

TABLE III. CPU CONSUMPTION AND THE NEXT PROCESS SELECTED UNDER RR.

Time	**Time quantum**	**Burst Times**					**Process selected for next execution**
		P1	**P2**	**P3**	**P4**	**P5**	
0	10	14	13	12	10	11	P1
10	10	4	13	12	10	11	P2
20	10	4	3	12	10	11	P3
30	10	4	3	2	10	11	P4
40	10	4	3	2	relinquish	11	P5
50	10	4	3	2		1	P1
54	10	relinquish	3	2		1	P2
57	10		relinquish	2		1	P3
59	10			relinquish		1	P5
60						relinquish	

TABLE IV. THE SCHEDULING CRITERIA UNDER RR.

Process	**Burst time**	**Waiting times**	**Context switches**
P1	14	40	1
P2	13	44	1
P3	12	47	1
P4	10	30	0
P5	11	49	1
		AVGWT = 42 ms	**No. CS = 4**

ii. Under the proposed algorithm:

The following Gantt chart shows the result under proposed (TQ = 10ms, and TSH = 0.3 × TQ):

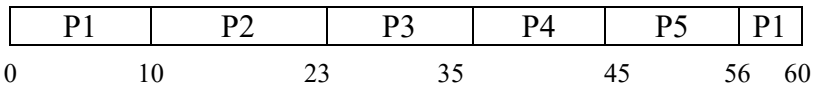

Table V shows the CPU consumption and the next process selected. The scheduling criteria calculated are shown in Table VI.

TABLE V. CPU CONSUMPTION AND THE NEXT PROCESS SELECTED UNDER PROPOSED ALGORITHM.

Time	Time quantum	Burst Times					Process selected for next execution
		P1	P2	P3	P4	P5	
0	10	14	13	12	10	11	P1
10	13	4	13	12	10	11	P2
23	12	4	relinquish	12	10	11	P3
35	10	4		relinquish	10	11	P4
45	11	4			relinquish	11	P5
56	4	4				relinquish	P1
60		relinquish					

TABLE VI. THE SCHEDULING CRITERIA UNDER PROPOSED ALGORITHM.

Process	Burst time	Waiting times	Context switches
P1	14	46	1
P2	13	10	0
P3	12	23	0
P4	10	35	0
P5	11	45	0
		AVGWT = 31.8 ms	**No. CS = 1**

4 Evaluation

As a proof of concept, two scheduling criteria are evaluated to compare the performance of the proposed algorithm over RR. Once the selection criteria have been defined, the algorithms under consideration must be evaluated. The evaluation method used is described below.

Analytic evaluation is a major class of evaluation methods. Analytic evaluation produces a number or formula to evaluate the performance of an algorithm for the system workload using the given algorithm and that system workload. One type of analytic evaluation is deterministic modeling. Deterministic modeling takes a specific predetermined workload and defines the performance of the algorithm for that workload [14,15]. The evaluation of the proposed algorithm against RR is considered on 6 different combinations of n and BTs, the burst times of processes vary from 1 to 100 ms. The derived results show a significant improvement in average waiting time and context switches as shown in Fig. 1and Fig. 2 respectively. Fig. 3 shows an improvement of the proposed algorithm over RR.

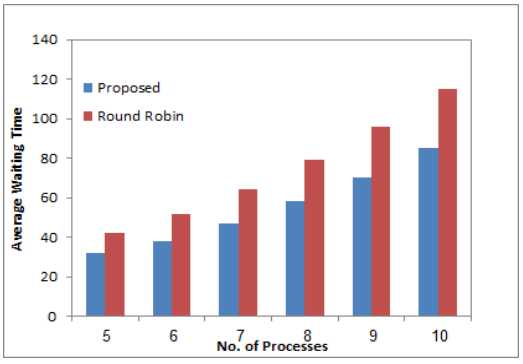

Fig. 1: Average waiting time comparison.

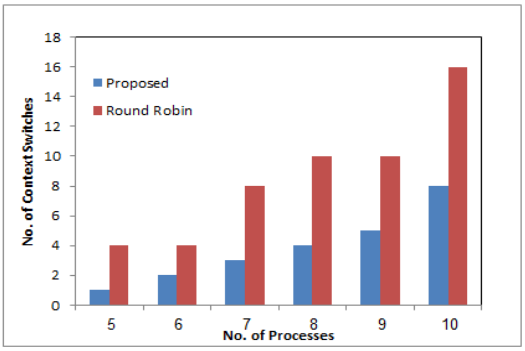

Fig. 2: Context switches comparison

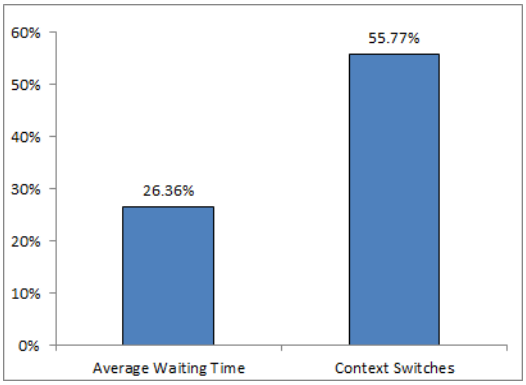

Fig. 3: Proposed algorithm improvement over RR.

5 Conclusion

In light of efficiency and effectiveness of RR scheduling algorithm, a modified CPU local scheduling based on RR has been proposed in this work. Its policy and improvement have been described. The proposed algorithm combines the low scheduling overhead of RR and favor short process to minimize the average waiting time and amount of context switches. The proposed algorithm assigns dynamic time slice for every process depending on a predefined threshold. According to this threshold, the process will continue until termination or it will be interrupted by the time quantum assigned by RR algorithm. In contrast to some related algorithms which put some processes in the same level and treat them equality regardless of their burst times, the proposed algorithm treats every process individually. The results showed an important conclusion; the performance of the proposed scheduler is higher than that of RR in interactive systems.

References

[1] 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," Int. J. New Comput. Archit. their Appl., vol. 5, no. October, pp. 64–71, Jan. 2010.

[2] A. R. Dash, S. kumar Sahu, and S. K. Samantra, "An Optimized Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum," Int. J. Comput. Sci. Eng. Inf. Technol., vol. 5, no. 1, pp. 07-26, 2015.

[3] S. M. Mostafa and S. Kusakabe, “Effect of Thread Weight Readjustment Scheduler on Scheduling Criteria,” Inf. Eng. Express, Jan. 2015.

[4] S. M. Mostafa and S. Kusakabe, “Towards Maximizing Throughput for Multithreaded Processes in Linux,” Int. J. New Comput. Archit. their Appl., vol. 4, no. 4, pp. 70–78, 2014.

[5] A. Singh, P. Goyal, and S. Batra, “An optimized round robin scheduling algorithm for CPU scheduling,” Int. J. Comput. Sci. Eng., vol. 02, no. 07, pp. 2383–85, 2010.

[6] 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 Comput., vol. 6, no. 1, pp. 0–12, 2017.

[7] S. M. Mostafa, “Proportional Weighted Round Robin: A Proportional Share CPU Scheduler in Time Sharing Systems,” Int. J. New Comput. Archit. their Appl., vol. 8, no. 3, pp. 142–47, 2018.

[8] M. S. Iraji, “Time Sharing Algorithm with Dynamic Weighted Harmonic Round Robin,” J. Asian Sci. Res., vol. 5, no. 3, pp. 131–42, 2016.

[9] T. Helmy and A. Dekdouk, “Burst round robin as a proportional-share scheduling algorithm,” Jan. 2007.

[10] M. M. Tajwar, M. N. Pathan, L. Hussaini, and A. Abubakar, “CPU scheduling with a round robin algorithm based on an effective time slice,” J. Inf. Process. Syst., vol. 13, no. 4, pp. 941–50, 2017.

[11] B. Caprita, W. C. Chan, J. Nieh, C. Stein, and H. Zheng, “Group ratio round-robin: O (1) proportional share scheduling for uniprocessor and multiprocessor systems,” Proc. Annu. Conf. USENIX Annu. Tech. Conf., no. 1, pp. 36–36, 2005.

[12] B. Caprita, J. Nieh, and W. C. Chan, “Group Round Robin: Improving the Fairness and Complexity of Packet Scheduling,” Proc. 2005 ACM Symp. Archit. Netw. Commun. Syst., pp. 29–40, 2005.

[13] L. Abeni, G. Lipari, and G. Buttazzo, “Constant bandwidth vs. proportional share resource allocation,” no. July 1999, pp. 107–11, 2003.

[14] A. Silberschatz, G. Gagne, and P. B. Galvin, Operating Systems Concepts. 2012.

[15] J. Sunil, V. G Anisha Gnana, and V. T Karthija, Fundamentals of Operating Systems Concepts. 2018.