Sarayut PHORNCHAROEN, Worawat SA-NGIAMVIBOOL

Department of Electrical Engineering, Faculty of Engineering, Mahasarakham University, Thailand

doi:10.15199/48.2018.04.07

A proposed round robin scheduling algorithm for enhancing performance of CPU utilization

Abstract. *An important problem of an operating system is CPU scheduling. This paper proposes round robin (RR) scheduling algorithm, named DevRR, with new dynamic time quantum (TQ) computed by the standard deviation and average burst time of each process in a queue. Performance of DevRR is compared to the standard RR, PRR, and BRR in term of decreasing of an average waiting time (AWT), an average turnaround time (ATT), and number of context switches (NCS). Results can reduce 22.97% of AWT, 22.13% of ATT, and 30.26% of NCS for 50-process data set.*

Streszczenie. *W artykule zaproponowano algorytm procesora z dynamicznym czasem kwantowym określanym jako odchyłka standardowa i średni czas impulsu każdego procesu w kolejkowaniu. Właściwości algorytmu porównano z innymi standardowymi metodami pod kątem oceny czasu oczekiwania.* ***Algorytm karuzelowy harmonogramu procesora poprawiający jego szybkość***

Keywords: round robin algorithm, dynamic time quantum, average waiting time, average turnaround time
Słowa kluczowe: algorytm karuzelowy, procesor, harmonogram kolejkowania.

Introduction

Generally, a computer system consists of hardware, software, user, and data set [1]. The main software is the system software named as Operating System (OS) that can directly manage system resources and environment. An important problem of OS is CPU scheduling in order to use more highly performance of CPU [1, 2] focused on average waiting time (AWT), average turnaroud time (ATT), and number of context switches (NCS) that means swap-in and swap-out CPU of processes. A process is a job in the ready queue to wait for utilizing CPU followed a sequence. CPU scheduling algorithm comprises several methods: first come first served (FCFS), short job first (SJF), priority queue (PQ), shortest remaining time (SRT), and round robin (RR).

FCFS algorithm is a easiest method. Limitation of this method is that if the current process takes a long time to occupy CPU, other processes in the ready queue also have to wait for a long time until ending complete execution of the current process. SJF algorithm discusses that if any process has a shorter burst time, a such process is firstly executed. Disadvantage of this method is processes in the ready queue is never executed if the new comming process has a shorter burst time. PQ algorithm focuses on the priority of process based on a job property. Limitation of this method is that if the new comming process gets a higher priority, all processes in the ready queue is never executed. A such queue is a preemptive queue. SRT algorithm is similar to SJF. When the current process must connect I/O devices, this process is cancelled CPU occupancy. Then the next process that has shortest remaining burst time is selected to occupy CPU. The standard RR algorithm defines a static time quantum (TQ) to execute processes for each round. Therefore, the main point of RR method is an appropriate time quantum. If a time quantum is set to more long time, it affects more increasing value of AWT and ATT that means unsuitability. In contrast, if a time quantum is set to less short time, it affects more increasing value of NCS that also means unsuitability.

However, RR algorithm is very popular [2] because it can strongly provide fairness for each process in the ready queue and ensure that all processes will be executed within a time quantum [2, 3]. Therefore, this paper proposes CPU scheduling algorithm based on the standard RR algorithm concept. Whereas the standard RR algorithm is operated with a static time quantum that is the average burst time (ABT), a proposed RR algorithm is operated with a dynamic time quantum. Performance analysis of CPU utilization considers the minimum value of AWT, ATT, and NCS [1, 3].

Related researches

Ahad [4] has adjusted the standard RR with B and K value. B value was computed from B = integer of (BT/TQ), where BT was a burst time, TQ was a time quantum. When B = 1, a process is executed by FCFS algorithm, but when B > 1, a process is executed by the standard RR algorithm. Moreover, K value was computed from K = ceiling of (average of (BT%TQ)), and TQ = TQ+K for a new time quantum in the next round. Mostafa [5] defined a new time quantum between the minimum and maximum value of BT in each round. Behera [6] proposed a new time quantum with rearranging all processes in the ready queue and assigned the median of the burst time to a new time quantum for the next round (it is called BRR algorithm). Mohanty [7] presented the burst time of the centered process in the ready queue to a new time quantum without rearranging processes in the ready queue. Matarneh [8] proposed a new time quantum with the burst time of the running process for the next round. Singh [9] defined a new time quantum with a double of the lowest burst time of processes in the ready queue. Pradhan [10] proposed a dynamic time quantum with the average of the remaining burst time for each round (it is called PRR algorithm).

A proposed scheduling algorithm concept

A proposed round robin scheduling algorithm, named DevRR, presents a new idea about calculating value of a time quantum (TQ). DevRR defines a new dynamic time quantum from the average burst time and the standard deviation (SD) of all burst time values in the ready queue [11, 12]. We emphasize the SD value for a data set of all burst times in each execution round. If SD is the greater value, it shows that all burst time values are distributed from the mean. On the other hand, if SD is the smaller value, it shows that all burst time values are approached to the mean. Therefore, DevRR algorithm will already calculate an appropriately dynamic time quantum for this data set of the burst time. Relation of the mean, the SD value, and the burst time of all processes in the ready queue [2, 12, 13] presents at the equation (1).

$$TQ = \mu + \sigma + \min(BT) \tag{1}$$

where: TQ is a time quantum, μ is ABT, σ is SD, BT is the burst time, $\min(BT)$ is the minimal burst time for each round. For DevRR algorithm concept, the first step, it must verify the size of the ready queue. If the ready queue is empty, it finishes the execution. The second step, it searches for the minimum burst time of processes in the ready queue. The third step, it calculates ABT. The fourth step, it calculates the variance for a data set of the burst time in this execution round. The fifth step, it calculates the SD value of this data set. In the last step, it calculates the optimized time quantum with two conditions; 1) if SD is greater than a half of ABT, TQ is set with the mean + SD + the minimum burst time. In this case, it means that this data set of the burst time is distributed from the mean, so TQ must be the larger value showed at the equation (2); 2) if SD is smaller than a half of ABT, TQ is set with the mean + SD + 1. In this case, it means that this data set of the burst time is approximate to the mean, so TQ must be the smaller value showed at the equation (3). It repeats all steps until the ready queue is empty. After that, it computes AWT, ATT, and NCS to analyze CPU utilization performance.

$$TQ = \frac{\sum_{i=1}^{n} BT_i}{n} + \sqrt{\frac{\sum_{i=1}^{n}(BT_i - \mu)^2}{n-1}} + \min_{i=1\to n}(BT_i) \tag{2}$$

$$TQ = \frac{\sum_{i=1}^{n} BT_i}{n} + \sqrt{\frac{\sum_{i=1}^{n}(BT_i - \mu)^2}{n-1}} + 1 \tag{3}$$

Where: n is amount of processes in the ready queue (size of queue), and i is a process ID.

Pseudo code for DevRR scheduling algorithm is shown in this below.

```
// --- DevRR Algorithm --- //
1.  TQ = 0, AWT = 0, ATT = 0, NCS = 0 //The initial values
2.  SizeOfQueue = Queue.getSize()
3.  if (SizeOfQueue = 0) go to 25.
4.  Sum = 0
5.  AverageBurstTime = 0 //The mean for each round
6.  Min = SizeOfQueue
    //To search for the minimum burst time
7.  for (i = 0; i < SizeOfQueue; i++) {
8.          BT = Queue.getBurstTime[i]
9.          if (BT < Min) then Min = BT
10.         Sum += BT
11. }
12. AverageBurstTime = Sum/SizeOfQueue
13. Total = 0
    //The variance for a data set of the burst time in queue
14. for (i = 0; i < SizeOfQueue; i++) {
15.         BT = Queue.getBurstTime[i]
16.         BT = BT – AverageBurstTime
17.         Total += (BT ∗ BT)
18. }
    //To calculate the standard deviation (SD)
19. SD = Math.sqrt(Total/(SizeOfQueue – 1 ))
    //To calculate the new optimized time quantum
20. if (SD > (AverageBurstTime/2))
21.         TQ = AverageBurstTime + SD + Min
22. else
23.         TQ = AverageBurstTime + SD + 1
24. Go to 2.
25. Compute AWT, ATT, NCS
26. Stop
```

Experiments

According to the related researches, we consider three algorithms including PRR, BRR, and DevRR compared with the standard RR algorithm that focuses on the minimal value of AWT, ATT, and NCS [1, 3]. For this experimental simulation, a data set of an arrival time and a burst time for each process are randomized by an object engine tester coded by Java [13, 14] where an arrival time value of each process is between 1 and 5 millisecond (ms), and a burst time value of each process is between 1 and 60 ms [14]. Moreover, we also define the experiment with seven data sets classified by amount of processes: 10, 50, 100, 500, 1000, 5000, and 10000 processes [12]. Workflow of an object engine tester is shown in Fig. 1.

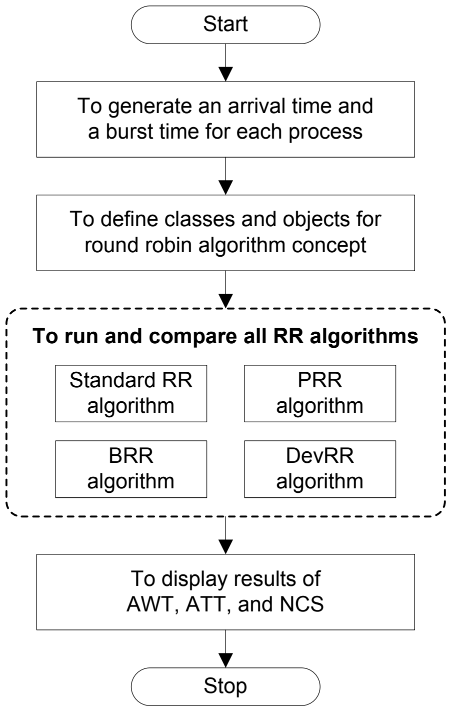
Start
To generate an arrival time and a burst time for each process
To define classes and objects for round robin algorithm concept
To run and compare all RR algorithms
Standard RR algorithm
PRR algorithm
BRR algorithm
DevRR algorithm
To display results of AWT, ATT, and NCS
Stop

Fig. 1. Workflow of an object engine tester

Results

An example of 10-process data set that is generated by an object engine tester is shown in Table 1.

Table 1. An example of 10-process data set

Process ID	Arrival time (ms)	Burst time (ms)
#00001	0	39
#00002	1	17
#00003	6	59
#00004	10	32
#00005	15	38
#00006	20	10
#00007	22	21
#00008	26	30
#00009	27	47
#00010	32	35

From Table 1, an arrival time of process ID #00001 is always generated as 0 ms. Its ID is the first process in the ready queue, therefore it has no an arrival time. Experimental results of all data sets is shown in Table 2.

From table 2, it shows a comparison of all data set results for PRR, BRR, and DevRR algorithm compared with the standard RR based on 100%. The better results are shown values marking with the minus sign and the green color. To discuss an overall AWT of all data sets, the best performance is DevRR algorithm that can make the most AWT reduction 22.97% for 50-process data set.

Table 2. Results of all data sets

Data set	Algorithm	TQ (ms)	AWT			ATT			NCS		
			(ms)	%	% (+/-)	(ms)	%	% (+/-)	(times)	%	%(+/-)
10	Standard RR	33	177.80	100.00%	0.00%	210.60	100.00%	0.00%	14	100.00%	0.00%
	PRR	33,11,9,6	176.20	99.10%	-0.90%	209.00	99.24%	-0.76%	17	121.43%	+21.43%
	BRR	33,6,14,6	174.70	98.26%	-1.74%	207.50	98.53%	-1.47%	17	121.43%	+21.43%
	DevRR	48,11	146.90	82.62%	-17.38%	179.70	85.33%	-14.67%	10	71.43%	-29.57%
50	Standard RR	31	798.96	100.00%	0.00%	829.20	100.00%	0.00%	76	100.00%	0.00%
	PRR	31,13,9,6,1	809.28	101.29%	+1.29%	839.52	101.24%	+1.24%	91	119.74%	+19.74%
	BRR	33,10,7,9,1	789.38	98.80%	-1.20%	819.62	98.84%	-1.16%	89	117.11%	+17.11%
	DevRR	50,11	615.46	77.03%	-22.97%	645.70	77.87%	-22.13%	53	69.74%	-30.26%
100	Standard RR	34	1981.92	100.00%	0.00%	2015.62	100.00%	0.00%	151	100.00%	0.00%
	PRR	34,14,7,4,1	2027.81	102.32%	+2.32%	2061.51	102.28%	+2.28%	187	123.84%	+23.84%
	BRR	36,15,4,3,1, 1	2018.55	101.85%	+1.85%	2052.25	101.81%	+1.81%	184	121.85%	+21.85%
	DevRR	52,7,1	1672.50	84.39%	-15.61%	1706.20	84.65%	-15.35%	121	80.13%	-19.87%
500	Standard RR	30	8445.78	100.00%	0.00%	8475.00	100.00%	0.00%	746	100.00%	0.00%
	PRR	30,14,8,5,2, 1	8635.01	102.24%	+2.24%	8664.23	102.23%	+2.23%	913	122.39%	+22.39%
	BRR	30,13,7,5,3, 1,1	8652.10	102.44%	+2.44%	8681.32	102.43%	+2.43%	955	128.02%	+28.02%
	DevRR	47,11,2	7329.95	86.79%	-13.21%	7359.17	86.83%	-13.17%	586	78.55%	-21.45%
1000	Standard RR	32	19224.13	100.00%	0.00%	19255.75	100.00%	0.00%	1510	100.00%	0.00%
	PRR	32,15,7,4,2	19688.73	102.42%	+2.42%	19720.36	102.41%	+2.41%	1856	122.91%	+22.91%
	BRR	33,13,7,3,3, 1	19603.56	101.97%	+1.97%	19635.17	101.97%	+1.97%	1914	126.75%	+26.75%
	DevRR	50,10	17028.87	88.58%	-11.42%	17060.49	88.60%	-11.40%	1163	77.02%	-22.98%
5000	Standard RR	31	88025.08	100.00%	0.00%	88055.60	100.00%	0.00%	7454	100.00%	0.00%
	PRR	31,15,8,4,2	90352.09	102.64%	+2.64%	90382.70	102.64%	+2.64%	9245	124.03%	+24.03%
	BRR	31,15,7,4,2, 1	90374.74	102.67%	+2.67%	90405.31	102.67%	+2.67%	9488	127.29%	+27.29%
	DevRR	49,11	79275.41	90.06%	-9.94%	79306.03	90.06%	-9.94%	5874	78.80%	-21.20%
10000	Standard RR	31	175586.83	100.00%	0.00%	175617.73	100.00%	0.00%	14867	100.00%	0.00%
	PRR	31,15,8,4,2	179907.30	102.46%	+2.46%	179937.75	102.46%	+2.46%	18359	123.49%	+23.49%
	BRR	31,14,8,4,2,1	180063.23	102.55%	+2.55%	180094.06	102.55%	+2.55%	18995	127.77%	+27.77%
	DevRR	49,10,1	158772.77	90.42%	-9.58%	158803.44	90.43%	-9.57%	11912	80.12%	-19.88%

Remark: the plus sign (+) is the increased percentage (not good), the minus sign (-) is the decreased percentage (good)

In addition, to discuss an overall ATT of all data sets, the best performance is DevRR algorithm that can make the most ATT reduction 22.13% for 50-process data set. For an overall NCS of all data sets, the best performance is also DevRR algorithm that can make the most NCS reduction 30.26% for 50-process data set.

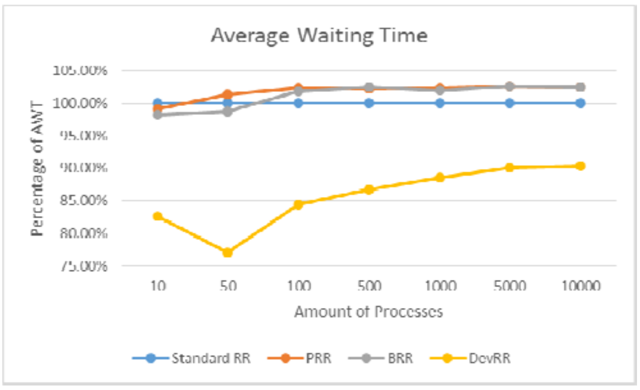

Fig. 2. Average waiting time for all data sets

In Fig. 2, it shows a comparison of AWT values for PRR, BRR, and DevRR algorithm compared to the standard RR. The best performance algorithm is DevRR that can compute AWT as 77.03% (22.97% reduction) for 50-process data set, 82.62% (17.38% reduction) for 10-process data set, and 84.39% (15.61% reduction) for 100-process data set respectively. Moreover, other data sets indicate that the approximate AWT value is 88.96% (11.04% reduction).

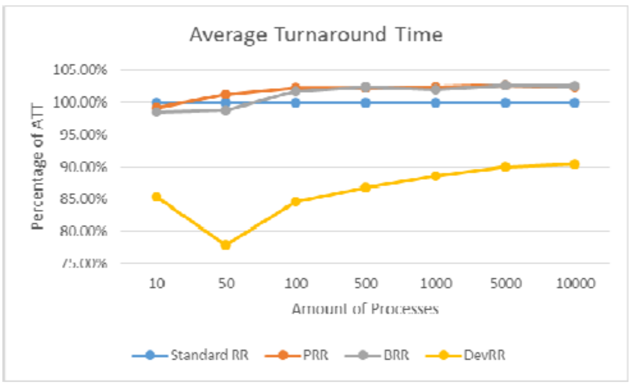

Fig. 3. Average turnaround time for all data sets

In Fig. 3, it shows a comparison of ATT values for PRR, BRR, and DevRR algorithm compared to the standard RR. The best performance algorithm is DevRR that can compute ATT as 77.87% (22.13% reduction) for 50-process data set, 84.65% (15.35% reduction) for 100-process data set, and 85.33% (14.67% reduction) for 10-process data set respectively. Moreover, other data sets indicate that the approximate ATT value is 88.98% (11.02% reduction).

In Fig. 4, it shows a comparison of NCS values for PRR, BRR, and DevRR algorithm compared to the standard RR. The best performance algorithm is DevRR that can compute NCS as 69.74% (30.26% reduction) for 50-process data set, 71.43% (29.57% reduction) for 10-process data set, and 77.02% (22.98% reduction) for 1000-process data set respectively. Moreover, other data sets indicate that the approximate NCS value is 79.40% (20.60% reduction).

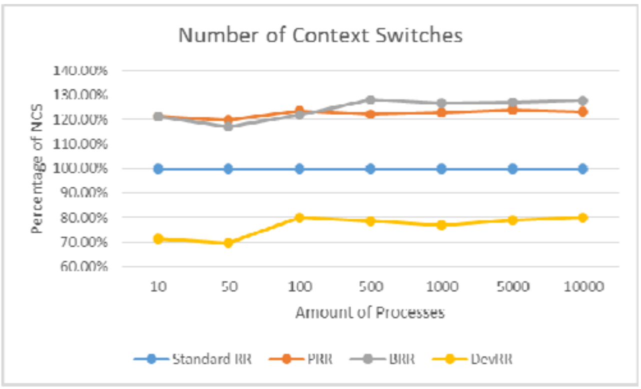

Fig. 4. Number of context switches for all data sets

According to experimental results for all data sets in table 2, it indicates that 50-process data set gets the best result where TQ are 50ms and 11ms for the first and the second round respectively. Therefore, table 3 shows the top-three algorithms for 50-process data set that enhance the performance of CPU utilization.

Table 3. Top-three algorithms for 50-process data set

Items	Top-three round robin algorithms		
	No.1	No.2	No.3
AWT	DevRR	BRR	Standard RR
ATT	DevRR	BRR	Standard RR
NCS	DevRR	Standard RR	BRR

In addition, DevRR algorithm may have the restriction in some cases. For example, if a burst time value for each process nearly approaches to an average burst time (ABT), DevRR algorithm is probably not an appropriate method for these data sets. However, this situation is rarely occurred.

Conclusions

We propose round robin scheduling algorithm named DevRR that defines new dynamic time quantum calculated by the standard deviation (SD) and the average burst time (ABT) in each execution round. Distribution of a data set of the burst time for all processes in the ready queue is discussed in order to define the optimized time quantum in each round. This experiment uses an object engine tester coded in Java to random value of an arrival time and a burst time of processes. Data sets for this experiment are 10, 50, 100, 500, 1000, 5000, and 10000 processes. DevRR, PRR, and BRR algorithms are compared to the standard RR. The experimental result of all data sets indicates that the best performance algorithm is DevRR. Moreover, 50-process data set can gain the best result that can reduce 22.97% of AWT, 22.13% of ATT, and 30.26% of NCS with 50ms and 11ms of TQ. Therefore, DevRR algorithm can strongly apply to support an operating system software development on mobiles, other devices, and network operating system (NOS) to enhance performance of CPU utilization at the present and the future.

Authors: *Mr.Sarayut Phorncharoen, Mahasarakham University, spsarayut@gmail.com; Assoc.Prof.Dr.Worawat Sa-Ngiamvibool, Mahasarakham University, Thailand. wor_nui@yahoo.com*

REFERENCES

[1] Chauhan N., *Principles of Operating System*, Oxford University Press. Oxford, chapter 6, 2014.
[2] Anju M., Antony N., Nandakumarm, R., Dynamic Time Slice Round Robin Scheduling Algorithm with Unknown Burst Time, *Indian Journal of Science and Technology*, 9(8) (2016), 1-6.
[3] Jeffay K., Smith F.D., Moorthy A., Anderson J., Proportional Share Scheduling of Operating System Services for Real-Time Applications, in *Proceeding of the 19th IEEE Real-Time Systems Symposium*, (1998), 25-32.
[4] Ahad M.A., Modifying Round Robin Algorithm for Process Scheduling using Dynamic Quantum Precision, *J. of Computer Applications on Issues and Challenges in Networking, Intelligence and Computing Technologies*, 3(3) (2012), 5-10.
[5] Mostafa S.M., Rida S.Z., Hamad S.H., Finding time quantum of round robin CPU scheduling algorithm in general computing systems using integer programming, *Int. J. of Research and Reviews in Applied Sciences*, 5(1) (2010), 64-71.
[6] Behera H., Mohanty R., Nayak D., A New Proposed Dynamic Quantum with Re-adjusted Round Robin Scheduling Algorithm and Its Performance Analysis, *Int. J. of Computer Applications*, 5(5) (2010), 10-15.
[7] Mohanty R., Design and Performance Evaluation of a New Proposed Shortest Remaining Burst in (SRBRR) Scheduling Algorithm, *Int. J. of Computer Applications*, 5(4) (2010), 10-15.
[8] Matarneh R.J., Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Process, *American J. of Applied Sciences*, 6(10) (2009), 31-37.
[9] Singh A., Goyal P., Batra S., An Optimized Round Robin Scheduling Algorithm for CPU Scheduling, *Int. J. on Computer Science and Engineering*, 2(7) (2010), 82-85.
[10] Pradhan P., Behera, P.K., Ray B.N.B., Modified Round Robin Algorithm for Resource Allocation, *Procedia Computer Science*, 85(3) (2016), 878-890.
[11] Abdulrahim A., Aliyu S., Mustapha A.M., Abdullahi, S.E., An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm, *Int. J. of Advanced Research in Computer Science and Software Engineering*, 4(2) (2014), 1-8.
[12] Jakub S., Maria S., Edyta Ł., Algorithm for selecting optimal clustering parameters used for over-segmentation reduction, *Przegląd Elektrotechniczny*, 92 (2016), nr 9, 250-256.
[13] Wanchai K., Chiraphon T., Hybrid of Lamda and Bee Colony Optimization for Solving Economic Dispatch, *Przegląd Elektrotechniczny*, 92 (2016), nr 9, 220-223.
[14] Gupta C., Maggo S., An efficient prediction model for software reusability for Java-based object-oriented systems, *Int. J. of Computer Aided Engineering and Technology*, 6(2) (2014), 182-199.