American Journal of Applied Sciences 6 (10): 1831-1837, 2009
ISSN 1546-9239

Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes

Rami J. Matarneh
Department of Management Information Systems,
Faculty of Administrative and Financial Sciences, Al-Isra Private University,
P.O. 11622, Amman, Jordan

Abstract: Problem statement: The performance and efficiency of multitasking operating systems mainly depends on the used CPU scheduling algorithm where the CPU is one of the primary computer resources and as round robin scheduling algorithm is considered most widely used scheduling algorithms in this research a new proposed variant of this algorithm presented, discussed in detail, tested and verified. **Approach:** The new proposed algorithm called Self-Adjustment-Round-Robin (SARR) based on a new approach called dynamic-time-quantum; the idea of this approach is to make the time quantum repeatedly adjusted according to the burst time of the now-running processes. **Results:** Based on the experiments and calculations that I have made the new modified algorithm radically solves the fixed time quantum problem which is considered a challenge for round robin algorithm. **Conclusion:** The use of dynamic scheduling algorithm increased the performance and stability of the operating system and support building of an self-adaptation operating system, which means that the system is who will adapt itself to the requirements of the user and not vice versa.

Key words: Round robin, self-adjustment-round-robin, dynamic-time-quantum, CPU scheduling

INTRODUCTION

Modern operating systems become more complex, they have evolved from a single task to a multitasking environment in which processes run in a concurrent manner[1]. CPU scheduling is an essential operating system task; therefore its scheduling is central to operating system design. When there is more than one process in the ready queue waiting its turn to be assigned to the CPU, the operating system must decide through the scheduler the order of execution[2,3].

Allocating CPU to a process requires careful attention to assure fairness and avoid process starvation for CPU. Scheduling decision try to minimize the following: Turnaround time, response time and average waiting time for processes and the number of context switches[4]. There exist a different Scheduling algorithms, each of them has advantages and disadvantages and as follows: First-Come-First-Served (FCFS) has the advantage of simplicity in which processes are dispatched according to their arrival time on the ready queue. Being a non preemptive discipline, once a process has a CPU, it runs to completion. It has the disadvantages that the average time is often quite long and it is not suitable in real time applications. This is mainly because one process with long execution time may hinder many short processes to complete before deadline[5,6]. On the other hand priority scheduling allocates processes to the CPU on the basis of an externally assigned priority and run the highest-priority first. The key to the performance of priority scheduling is in choosing priorities for the processes. The main problem of it is starvation and the solution to this problem is aging. Shortest-Job-First (SJF) scheduling is provably optimal, providing the shortest average waiting time. The obvious problem with this algorithm is that it is require precise knowledge of how long a job or process will run and this information is not usually available and unpredictable[7,8]. The Round Robin (RR) algorithm which is the main concern of this research is one of the oldest, simplest and fairest and most widely used scheduling algorithms, designed especially for time-sharing systems. A small unit of time, called time slices or quantum is defined. All runnable processes are kept in a circular queue. The CPU scheduler goes around this queue, allocating the CPU to each process for a time interval of one quantum. New processes are added to the tail of the queue. The CPU scheduler picks the first process from the queue, sets a timer to interrupt after one quantum and dispatches the process[9]. If the process is still running at the end of the quantum, the CPU is preempted and the process is added to the tail of the queue. If the process finishes before the end of the quantum, the process itself releases the CPU voluntarily. In either case, the CPU scheduler assigns the CPU to the next process in the ready queue. The performance of the RR algorithm depends heavily on the size of the time quantum. At one extreme, if the time quantum is extremely large, cause poor response time and approximates FCFS. If the time quantum is extremely small this causes too many context switches and lowers the CPU efficiency. RR algorithm gives better responsiveness but worse average turnaround time and waiting time[4,10-12]. In this research I present a solution to the time quantum problem by make the operating systems adjusting the time quantum according to the burst time of the existed set of processes in the ready queue.

MATERIALS AND METHODS

The idea of this study was built on the basis of a questionnaire I have prepared and distributed in march 2009, by the e-mail to a sample of one thousand of computer users from my country Jordan and from other countries to understand and identify the user behavior and preferences in order to improve the performance of the operating system, the sample included university professors, students, administrators, accountants and economists, from various levels, experiences and ages.

The results showed that users from the same group have the same behavior and the vast majority of users using a specific set of programs almost exclusively programs that are related to user's work, audio-video players, web browsers and text editors. Table 1 summarizes the most important questions that the questionnaire is structured around and the results that I obtained.

The main conclusion for me was that each user prefers to use a specific set of programs (which vary from one user to another) and do not tend to use other than it, except in rare cases.

This result led me to think about how to use such information to enhance and improve the performance of the operating system. I found that I can use this information to improve the CPU scheduling that based on round robin scheduling algorithm. This can be done by analyzing the process to identify its burst time, the analysis carried out only once when the process executed for the first time, without the need to be replicated, except in rare cases such as the program had been changed, modified, or updated since the last analysis.

The analysis will determine the burst time of the process and accordingly the operating system can adapt itself by readjusting the value of the time-slice or time quantum Q to commensurate with the set of the programs in the ready queue.

Table 1: Summary of main questionnaire parts

Questions	No (%)	Yes (%)
Do you use specific programs always?	7	93
Do you frequently install new programs?	92	8
Can other people use your computer?	81	19
Do other people who use your computer use the same programs that you use?	4	96
Do you use programs that are related to your work?	27	73

Description of the proposed method: When operating system installed for the first time, it begins with a default time quantum value, which is subject to change after a period of time through which the operating system can identify the burst time for a subset of the programs used by the user. So, I assume that the system will not immediately take advantage of this method because it needs a short period of time to learn user behavior through the analysis of the burst time of the new processes. The determined time quantum represents real and optimal value because it based on real burst time unlike the other methods, which depend on fixed or possible time quantum value, determined by a variety methodologies such as guessing, fuzzy logic…[13,14].

Repeatedly, when a new process loaded to be executed the operating system tests the status of the specified program which can be either 1 or 0.

When the status equals to 0 this means that the process is either being executed for the first time or it has been modified or updated since the last analysis. In this case the operating system assign a counter to find the burst time of the process and continues executing the processes in the ready queue on the current round including the new arrival process using the current time quantum Q, otherwise and when status is equal to 1, then the operating system recalculates the time quantum Q depending on the remaining burst time of all ready processes including the new arrival process.

I have found through experience that the optimal time quantum can be presented by the median[15,16] for the set of processes in the ready queue, if the median less, than 25 then its value must be modified to 25 to avoid the overhead of the context switch. Formula 1 represents the value of time quantum Q consequences for the median $\tilde{x}$:

$$Q = \tilde{x} \equiv \begin{cases} Y_{(N+1)/2} & \text{if N is odd} \\ \frac{1}{2}(Y_{N/2}) + (Y_{1+N/2}) & \text{if N is even} \end{cases} \quad (1)$$

where, Y is the number located in the middle of a group of numbers arranged in ascending order. Because the value of Q should not be less than 25,

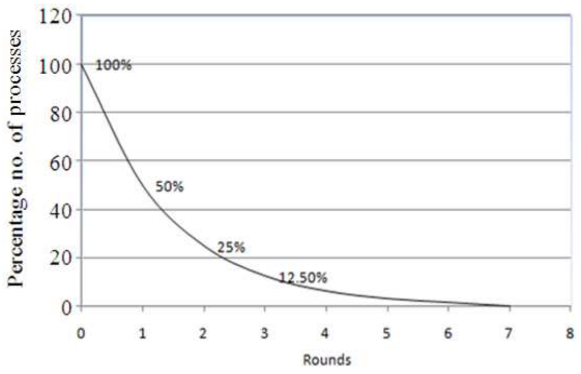


Fig. 1: The rate of decrease in the number of processes in each round

we can rewrite formula (1) in more specific form to fit with the allowed range:

$$Q = \begin{cases} \tilde{x}, & \text{if } \tilde{x} \geq 25 \\ 25, & \text{if } \tilde{x} < 25 \end{cases} \qquad (2)$$

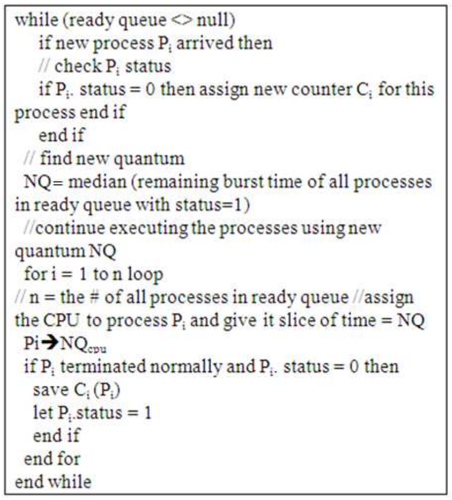
This means that 50% of the processes will be terminated through the first round and as time quantum calculated repeatedly for each round then 50% of the remaining processes will be terminated during the second round, with the same manner for the third round, fourth round, which is mean that the maximum number of rounds will be less than or equal to 6 whatever the number of process or their burst time. Figure 1 shows the significant decrease of the number of processes in each round.

The significant decrease of the number of processes, inevitably will lead to significant reduction in the number of context switch, which may pose high overhead on the operating system in many cases. The number of context switch can be represented mathematically as follow:

$$Q_T = [\ \sum_{1}^{r} k_r)\ ]\ -1 \qquad (3)$$

Where
Q_T = The total number of context switch
r = The total number of rounds, r = 1, 2…6
k_r = The total number of processes in each round

In other variants of round robin scheduling algorithm the context switch occurs even if there is only a single process in the ready queue, where the operating system assigns the process a specific time quantum Q, when time quantum expires the process interrupted and again assigned the same time quantum Q, regardless,



```
while (ready queue <> null)
    if new process $P_i$ arrived then
     // check $P_i$ status
     if $P_i$. status = 0 then assign new counter $C_i$ for this
process end if
    end if
  // find new quantum
  NQ= median (remaining burst time of all processes
in ready queue with status=1)
  //continue executing the processes using new
quantum NQ
  for i = 1 to n loop
// n = the # of all processes in ready queue //assign
the CPU to process $P_i$ and give it slice of time = NQ
  Pi→$NQ_{cpu}$
  if $P_i$ terminated normally and $P_i$. status = 0 then
   save $C_i$ ($P_i$)
   let $P_i$.status = 1
   end if
  end for
end while
```

Fig. 2: Pseudocode of self-adjustment-round-robin (SARR) algorithm

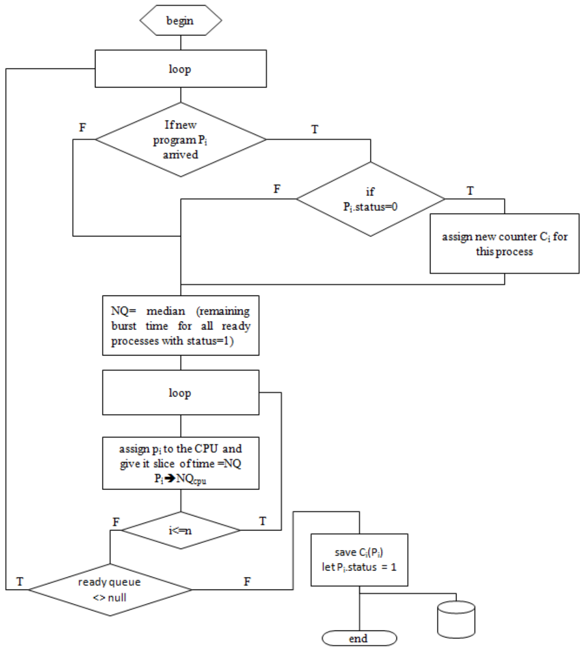

begin
loop
If new program Pi arrived
F
T
if Pi.status=0
F
T
assign new counter Ci for this process
NQ= median (remaining burst time for all ready processes with status=1)
loop
assign pi to the CPU and give it slice of time =NQ Pi→NQcpu
i<=n
F
T
ready queue <> null
T
F
save Ci(Pi) let Pi.status = 1
end

Fig. 3: Flowchart of Self-Adjustment-Round-Robin (SARR) algorithm

whether the process alone in the ready queue or not, which means that, there will be additional unnecessary context switches, while this problem does not occur at all in the new proposed algorithm; because in this case the time quantum will equal to the remaining burst time of the process.

Figure 2 represents the pseudocode of the proposed algorithm and Fig. 3 shows its flowchart.

RESULTS

The proposed algorithm was designed to meet all scheduling criteria such as max CPU utilization, max throughput, min turnaround time, min waiting time and min response time.

To evaluate the proposed method with regard to the above criteria[17-21], for the purpose of simplicity I will take a group of four processes in four different cases with random burst time and what should be mentioned here that the number of processes does not change the result because the algorithm works effectively even if it used with a very large number of processes.

In each case I will compare the result of the proposed method with the classic approach used in round robin scheduling algorithm and as classical approach uses fixed time quantum Q, so I assume a constant time quantum Q equal to 20 in all cases, in order to compare the two algorithms fairly.

Case 1: Assume four processes arrived at time = 0, with burst time (P_1 = 20, P_2 = 40, P_3 = 60, P_4 = 80) as shown in Table 2. part (a) in Table 2 shows the output using classical approach, while part (b) in Table 2 shows the output using new proposed method. Figure 4 shows Gantt chart for part (a) and Fig. 5 shows Gantt chart for part (b).

Table 2: Comparison between fixed and dynamic time quantum in round robin algorithm (case 1)

Process	Arrival time	Burst time
Part (a), with static Q = 20		
P_1	0	20
P_2	0	40
P_3	0	60
P_4	0	80
Time quantum		20
Turn-around time		120
Waiting time		70
Context switch		9
Part (b), with dynamic Q		
P_1	0	20
P_2	0	40
P_3	0	60
P_4	0	80
Time quantum		50, 25, 25
Turn-around time		65
Waiting time		62.5
Context switch		6

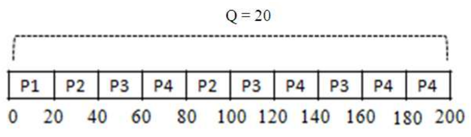


Fig. 4: Gantt chart for part (a) in Table 2 (case 1)

Case 2: Assume four processes arrived at time = 0, with burst time (P_1 = 10, P_2 = 14, P_3 = 70, P_4 = 120) as shown in Table 3. Part (a) in Table 3 shows the output using classical approach, while part (b) in Table 3 shows the output using new proposed method. Figure 6 shows Gantt chart for part (a) and Fig. 7 shows Gantt chart for part (b).

Case 3: Assume four processes arrived at different time, respectively 0, 4, 8, 16, with burst time (P_1 = 18, P_2 = 70, P_3 = 74, P_4 = 80) as shown in Table 4. Part (a) in Table 4 shows the output using classical approach, while part (b) in Table 4 shows the output using new proposed method. Figure 8 shows Gantt chart for part (a) and Fig. 9 shows Gantt chart for part (b).

Table 3: Comparison between fixed and dynamic time quantum in round robin algorithm (case 2)

Process	Arrival time	Burst time
Part (a), with static Q = 20		
P_1	0	10
P_2	0	14
P_3	0	70
P_4	0	120
Time quantum		20
Turn-around time		100.5
Waiting time		47
Context switch		11
Part (b), with dynamic Q		
P_1	0	10
P_2	0	14
P_3	0	70
P_4	0	120
Time quantum		42, 53, 25
Turn-around time		71
Waiting time		42.5
Context switch		6

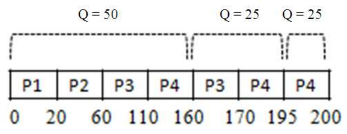


Fig. 5: Gantt chart for part (b) in Table 2 (case 1)

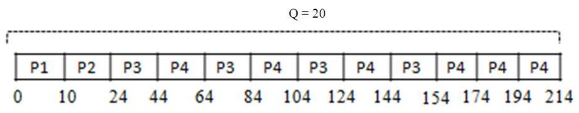


Fig. 6: Gantt chart for part (a) in Table 3

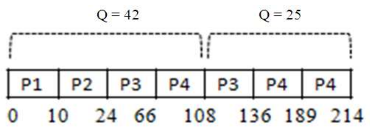


Fig. 7: Gantt chart for part (b) in Table 3

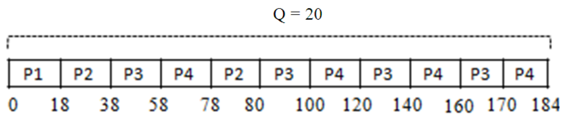


Fig. 8: Gantt chart for part (a) in Table 4

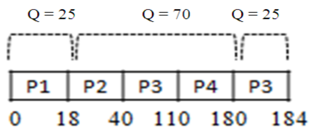


Fig. 9: Gantt chart for part (b) in Table 4

Table 4: Comparison between fixed and dynamic time quantum in round robin algorithm (case 3)

Process	Arrival time	Burst time
Part (a), with static Q = 20		
P_1	0	18
P_2	4	22
P_3	8	70
P_4	16	74
Time quantum		20
Turn-around time		106
Waiting time		60
Context switch		10
Part (b), with dynamic Q		
P_1	0	18
P_2	4	22
P_3	8	70
P_4	16	74
Time quantum		25, 70, 25
Turn-around time		46
Waiting time		35
Context switch		4

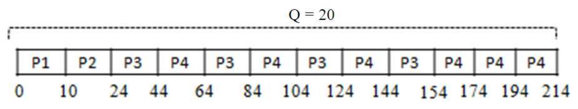


Fig. 10: Gantt chart for part (a) in Table 5

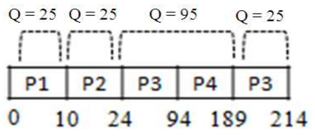


Fig. 11: Gantt chart for part (b) in Table 5

Table 5: Comparison between fixed and dynamic time quantum in round robin algorithm (case 4)

Process	Arrival time	Burst time
Part (a), with static Q = 20		
P_1	0	10
P_2	6	14
P_3	13	70
P_4	21	120
Time quantum		20
Turn-around time		90.5
Waiting time		37
Context switch		11
Part (b), with dynamic Q		
P_1	0	10
P_2	6	14
P_3	13	70
P_4	21	120
Time quantum		25, 46, 49, 25
Turn-around time		46
Waiting time		30.5
Context switch		4

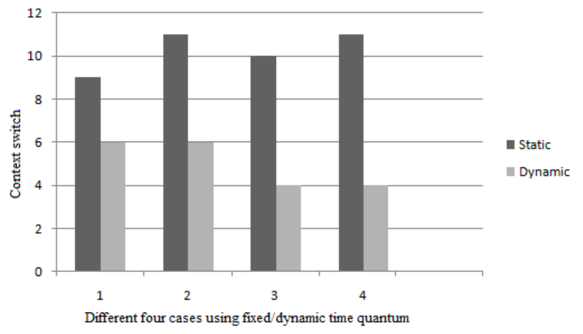


Fig. 12: Difference in context switch between dynamic and fixed time quantum

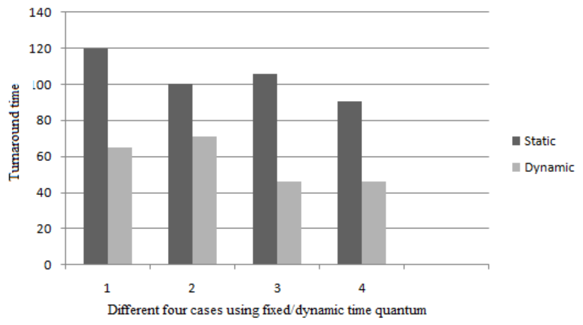


Fig. 13: Difference in turnaround time between dynamic and fixed time quantum

Case 4: Assume four processes arrived at different time, respectively 0, 6, 13, 21, with burst time (P_1 = 10, P_2 = 14, P_3 = 70, P_4 = 120) as shown in Table 5. Part (a) in Table 4 shows the output using classical approach, while part (b) in Table 5 shows the output using new proposed method. Figure 10 shows Gantt chart for par t (a) and Fig. 11 shows Gantt chart for part (b).

From the above comparisons, it is clear that the dynamic time quantum approach is superior to the fixed time quantum approach in round robin algorithm, where the dynamic time quantum significantly reduces the context switch, turnaround time and the waiting time. Respectively, Fig. 12-14 represent the difference in context switch, waiting time and turnaround time between the proposed algorithm with dynamic time quantum and the other algorithms based on fixed time quantum.

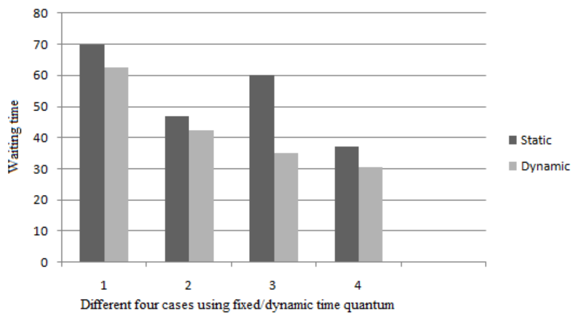


Fig. 14: Difference in waiting time between dynamic and fixed time quantum

DISCUSSION

A lot of attempts were developed to find a solution for the high turnaround time, high waiting time and the overhead of extra context switches in round robin algorithm, regardless of the different methodologies used in these attempts; however all of them rely based on the fixed-time-quantum

The proposed algorithm called Self-Adjustment-Round-Robin (SARR) based on dynamic-time-quantum was designed to solve all critical previously mentioned problems in a practical, simple and applicable manner.

The above comparisons show that the proposed algorithm provides much better results twice or three times and in some cases perhaps more than other approaches based on fixed time quantum in all scheduling criteria.

Laboratory test of this algorithm showed through a simulation program which is prepared for this purpose that this algorithm works in a stable manner regardless of the number of the now running processes, taking into consideration the terminated and the new arrival processes.

It is recommended to use the dynamic-time-quantum concept; because it will give the operating system the ability to adapt to the user behavior and not vice versa, which may lead us to rethink building an intelligent, learnable and adaptable operating system.

CONCLUSION

Time quantum is the bottleneck facing round robin algorithm and was more frequently asked question: What is the optimal time quantum to be used in round robin algorithm?

This research provides definitive answer to this question by using dynamic time quantum instead of fixed time quantum, where the operating system itself finds the optimal time quantum without user intervention.

REFERENCES

1. Helmy, T. and A. Dekdouk, 2007. Burst round robin as a proportional-share scheduling algorithm. IEEEGCC, King Fahed University. http://eprints.kfupm.edu.sa/1462/
2. Rashid, M.M. and Z.N. Akhtar, 2006. A new multilevel CPU scheduling algorithm. J. Applied Sci., 6: 2036-2039. DOI: 10.3923/jas.2006.2036.2039
3. Tanebaun, A.S., 2008. Modern Operating Systems. 3rd Edn., Prentice Hall, ISBN: 13: 9780136006633, pp: 1104.
4. Silberschatz, A., P.B. Galvin and G. Gagne, 2004. Operating Systems Concepts. 7th Edn., John Wiley and Sons, USA., ISBN: 13: 978-0471694663, pp: 944.
5. Place, J., 1989. FCFS: A novel scheduling policy for tightly-coupled parallel computer systems. Proceeding of the ACM Annual Computer Science Conference, Proceedings of the 17th ACM Annual Conference on Computer Science, Feb. 21-23, ACM Press, Louisville, Kentucky, pp: 188-194. DOI: 10.1145/75427.75450
6. Zhao, W. and J.A. Stankovic, 1989. Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems. Proceeding of the IEEE Real-Time Systems Symposium, Dec. 1989, pp: 156-165. http://tw.rpi.edu/wiki/Performance_Analysis_of_FCFS_and_Improved_FCFS_Scheduling_Algorithms_for_Dynamic_Real-Time_Computer_Systems
7. Lupetti, S. and D. Zagorodnov, 2006. Data popularity and shortest-job-first scheduling of network transfers. Proceeding of the International Conference on Digital Telecommunications, Aug. 29-31, EEE Computer Society, USA., pp: 26-26. DOI: 10.1109/ICDT.2006.28
8. Sandmann, W., 2006. Benefits of alternating FCFS/SJF service order. Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications, Aug. 18-20, World Scientific and Engineering Academy and Society, Stevens Point, Wisconsin, USA., pp: 194-199. http://portal.acm.org/citation.cfm?id=1366454
9. Back, D.S., K. Pyun, S.M. Lee, J. Cho and N. Kim, 2007. A hierarchical deficit round-robin scheduling algorithm for a high level of fair service. Proceedings of the International Symposium on Information Technology Convergence, Nov. 23-24, IEEE Computer Society, Washington DC., USA., pp: 115-119. http://portal.acm.org/citation.cfm?id=1338146

10. Nieh, J., C. Vaill and H. Zhong, 2001. Virtual-time round-robin: An O(1) Proportional share scheduler. Proceedings of the General Track: USENIX Annual Technical Conference, (UAT'01), USENIX, pp: 245-260. http://www.usenix.org/event/usenix01/full_papers/nieh/nieh_html/
11. Caprita, B., W.C. Chan, J.N.C. Stein and H. Zheng, 2004. Group ratio round-robin: O(1) Proportional Share scheduling for uniprocessor and multiprocessor systems. Columbia University, Technical Report CUCS-028-04. http://www.ncl.cs.columbia.edu/publications/cucs-028-04.pdf
12. Tong, W. and J. Zhao, 2007. Quantum varying deficit round robin scheduling over priority queues. Proceedings of the International Conference on Computational Intelligence and Security, Dec. 15-19, Computer Society, Harbin, China, pp: 252-256. DOI: 10.1109/CIS.2007.189
13. Zahedi, M.H., M. Ghazizadeh and M. Naghibzadeh, 2007. Fuzzy Round Robin CPU Scheduling (FRRCS) algorithm. Proceedings of the 2007 International Conference on Systems, Computing Sciences and Software Engineering (SCSS), Part of the International Joint Conferences on Computer, Information and Systems Sciences and Engineering, Dec. 3-12, Pubzone, Bridgeport, CT., USA., pp: 348-353. DOI: 10.1007/978-1-4020-8741-7_63
14. Finley, D., J.R. Ramos, V. Rego and J. Sang, 2009. A fast computational algorithm for simulating round-robin service. J. Simulat., 3: 29-39. DOI: 10.1057/jos.2008.10
15. Plesha, Michael Gray, Gary Costanzo and Francesco, 2009. Engineering Mechanics: Statics. 9th Edn., McGraw-Hill, ISBN: 0077275535, pp: 704.
16. Hibbeler, R.C., 2009. Statics Study Pack for Engineering Mechanics: Statics. 12th Edn., Prentice Hall, ISBN: 0136091830, pp: 168.
17. Ramos, J.R., V. Rego and J. Sang, 2003. An improved computational algorithm for round-robin service. Proceedings of the 2003 Winter Simulation Conference, Dec. 7-10, IEEE Xplore Press, USA., pp: 721-728. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1261488
18. Ramos, J.R., V. Rego and J. Sang, 2006. An efficient burst-arrival and batch-departure algorithm for round-robin service. Simulat. Model. Pract. Theor., 14: 1-24. DOI: 10.1016/J.SIMPAT.2005.02.008
19. Kurimoto, T., O. Eiji and Y. Naoaki, 2003. Adaptive Deficit round robin algorithm achieving fair bandwidth allocation and improving the delay quality. IEIC Tech. Rep., 101: 19-24. http://sciencelinks.jp/j-east/article/200203/000020020301A1026118.php
20. Lee, E.T., 1990. On average turnaround time. Kybernetes, 19: 46-58. DOI: 10.1108/eb005836
21. Chaskar, H.M. and U. Madhow, 2003. Fair scheduling with tunable latency: A round-robin approach. IEEE/ACM Trans. Network, 11: 592-601. DOI: 10.1109/TNET.2003.815290