Journal of
Computer Science & Systems Biology

Mostafa et al., J Comput Sci Syst Biol 2011, 4:4
DOI: 10.4172/jcsb.1000078

Research Article Open Access

Improving Scheduling Criteria of Preemptive Tasks Scheduled under Round Robin Algorithm using Changeable Time Quantum

Samih M. Mostafa[1]*, Safwat H. Hamad[2] and S. Z. Rida[1]

[1]*Faculty of Science, Mathematics Department, South Valley University, Qena, Egypt*
[2]*Faculty of Computer & Information Sciences, Ain Shams University, Abbassia, Cairo, Egypt*

Abstract

Problem statement: In Round Robin Scheduling the time quantum is fixed and then processes are scheduled such that no process get CPU time more than one time quantum in one go. If time quantum is too large, the response time of the processes is too much which may not be tolerated in interactive environment. If time quantum is too small, it causes unnecessarily frequent context switch leading to more overheads resulting in less throughput. In this paper a method using changeable time quantum has been proposed that decides a value that is neither too large nor too small such that this value gives the beast scheduling criteria and every process has got reasonable response time and the throughput of the system is not decreased due to unnecessarily context switches.

Keywords: Round robin; Changeable time quantum; Survived tasks; Residual time; Cyclic queue

Introduction

When a computer is multi-programmed, it frequently has multiple processes competing for the CPU at the same time. When more than one process is in the ready state and there is only one CPU available, the operating system must decide which process to run first. The part of operating system that makes the choice is called short term scheduler or CPU scheduler. The algorithm that it uses is called scheduling algorithm. There are several scheduling algorithms. Different scheduling algorithms have different properties and the choice of a particular algorithm may favor one class of processes over another. Many criteria have been suggested for comparing CPU scheduling algorithms and deciding which one is the best algorithm [19].

Some of the criteria include (i)Fairness (ii)CPU utilization (iii) Throughput (iv)Turnaround time (v)Waiting time (vi)Response time (vii)Context switches. It is desirable to maximize CPU utilization and throughput, to minimize turnaround time, waiting time, response time and context switches and to avoid starvation of any process [13,20]. Some of the scheduling algorithms are briefly described below: FCFS: In First come First serve scheduling algorithm the process that request first is scheduled for execution [13,19,20]. SJF: In shortest Job first scheduling algorithm the process with the minimum burst time is scheduled for execution [13,20]. SRTN: In shortest Remaining time next scheduling algorithm, the process with shortest remaining time is scheduled for execution [19]. Priority: in Priority Scheduling algorithm the process with highest priority is scheduled for execution [13,19,20]. Multilevel queue scheduling: In this the ready queue is partitioned into several separate queues. The processes are permanently assigned to one queue generally based on some property of the process such as memory size, process priority or process type. Each queue has its own scheduling algorithm. There is scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. Each queue has absolute priority over low priority queues [13]. Multilevel feedback-queue scheduling: This is like Multilevel queue scheduling but allows a process to move between queues [13]. Fair share Scheduling: Fair share scheduler considers the execution history of a related group of processes, along with the individual execution history of each process in making scheduling decision. The user community is divided into a fair- share groups. Each group is allocated a fraction of CPU time. Scheduling is done on the basis of priority of the process, its recent processor usage and the recent processor usages of the group to which the process belongs. Each process is assigned a base priority. The priority of a process drops as the process uses the processor and as the group to which process belongs uses the processor [19]. Guaranteed scheduling: In this a ratio of actual CPU time a process had and its entitled CPU time is calculated. The process with this lowest ration is scheduled [20]. Lottery Scheduling: The basic idea is to give processes lottery tickets for CPU time. Whenever a scheduling decision has to be made, a lottery ticket is chosen at random and the process holding the ticket gets the CPU [20]. HRRN: In this, response ratio is calculated for each process. The process with the highest ratio is scheduled for execution [19] Round-robin: In this, the CPU scheduler goes around the ready queue allocating the CPU to each process for a time interval of up to one time quantum. If time quantum is too large, the response time of the processes is too much which may not be tolerated in interactive environment. If time quantum is too small, it causes unnecessarily frequent context switch leading to more overheads resulting in less throughput. In this paper we proposed a new technique depending on making the value of the time quantum changeable, this value is neither too large nor too small such that every process has got reasonable response time and the throughput of the system is not decreased due to unnecessarily context switches.

To this end, we utilize the following assumptions throughout this paper to simplify the problem formulation:

- tasks are belong to interactive environment, i.e., tasks are premptive. In an environment with interactive users, preemption is essential to keep one process from hogging the CPU and denying service to the others. Even if no process

***Corresponding author:** Samih M. Mostafa, Faculty of Science, Mathematics Department, South Valley University, Qena, Egypt, E-mail: samih_montser@yahoo.com

Received March 28, 2011; **Accepted** November 06, 2011; **Published** November 11, 2011

Citation: Mostafa SM, Hamad SH, Rida SZ (2011) Improving Scheduling Criteria of Preemptive Tasks Scheduled under Round Robin Algorithm using Changeable Time Quantum. J Comput Sci Syst Biol 4: 071-000. doi:10.4172/jcsb.1000078

intentionally ran forever, due to a program bug, one process might shut out all the others indefinitely. Preemption is needed to prevent this behavior,

- tasks are of variable size in terms of number of instructions which may range anywhere from instructions up to thousands or greater for some interactive tasks.
- no task is rated more important than any other task,
- each task is considered to be independent of all others, i.e., there is no communication between tasks running on the processor,
- the CPU cost of each task is assumed to be known.
- new tasks are permitted to enter the queue.

From these assumptions, it is clear that the problem has been reduced to almost the simplest formulation. The most common method of task scheduling in interactive systems that apply when these assumptions are made is the round-robin (RR). RR is also one of the oldest, simplest and most widely used proportional share scheduling algorithms, and because of its usefulness, many proportional share scheduling mechanisms have been developed [1,4,6-11,16]. In addition, RR algorithms have low scheduling overhead of $O(1)$, which means scheduling the next task takes a constant time [3,5,15].

Briefly: RR scheduling dose not reorder the tasks but allows preemption to occur so that tasks that take longer than a designated time quantum are put to the back of the cyclic queue for processing at a later time. This paper elaborates the RR scheduling policy by allowing the time quantum to vary after each round through the cyclic queue. The terms task and job are used almost interchangeably in this text.

With the simple problem formulation, the main purpose of the proposed work is to minimize the following criteria:

i) average waiting time,

ii) average turnaround time. The turnaround time is defined as the interval from the time of submission of a process to the time of completion. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O and

iii) the context switches.

The main factor with the preemptive scheduler is the size of the time quantum. Setting the time quantum too short causes too many processes switches and lowers the CPU efficiency, but setting it too long may cause poor response to short interactive requests. A quantum around 20-50 msec is often a reasonable compromise [14].

Latest algorithms [2,12,17,18] try to modify RR by adjusting the time quantum. In the successive sections we will introduce how we can improve the round-robin algorithm by readjusting the size of the time quantum to achieve the above criteria. In each round in the queue, the time quantum will be modified according to the burst times of the tasks. Using Changeable Time Quantum (CTQ) gives significant improvement in desired criteria.

Materials and Methods

CTQ Definitions

To provide a more in depth description of CTQ, we first define more precisely the state CTQ associates with each round, and then describe in detail how CTQ uses that state to schedule tasks. We define the terminology list we use in (Table 1).

The following equations determine the time quantum TQ that gives the smallest average waiting time in each round. TQ is ranged from α up to the given operating system time slice (*OSTS*), where $\alpha \le OSTS$

$$NTQ[T_i]=\begin{cases}\left\lfloor \frac{BT[T_i]}{TQ} \right\rfloor^{+1} & if \quad BT[T_i]\neq l*TQ \\ & l=1,2,3,... \\ \frac{BT[T_i]}{TQ}-1 & if \quad BT[T_i]=l*TQ \\ & l=1,2,3,...\end{cases} \tag{1}$$

(Table 2) exhibits an example, in which each task with its burst time:

if we use a time quantum of 4 msec, we see from the Gantt Chart:

T1	T2	T3	T1	T1	T1	T1	T1	
0	4	7	10	14	18	22	26	30

that the $NTQ[T_1]$ is 5, the $NTQ[T_2]$ is 0, and the $NTQ[T_3]$ is 0, although the number of context switches of T_1 is 1, the number of context switches of T_2 is 0, and the number of context switches of T_3 is 0.

$$SLTQ\ [T_i]=\begin{cases}0+\sum_{k=1}^{i-1}\begin{cases}TQ & if \quad NTQ_k>0\\ BT_k & if \quad NTQ_k=0\end{cases} & if \quad NTQ_i=0\\ NTQ_i*TQ+\\ \quad \sum_{k=1,k\neq i}^{n}\begin{cases}BT_k & if \quad NTQ_k<NTQ_i \quad and \quad k\neq i\\ BT_k & if \quad NTQ_k=NTQ_i \quad and \quad k<i\\ (NTQ_i*TQ) & if \quad NTQ_k\geq NTQ_i \quad and \quad k>i\\ (NTQ_i+1)*TQ & if \quad NTQ_k>NTQ_i \quad and \quad k<i\end{cases} & if\ NTQ_k>0\end{cases} \tag{2}$$

In the above example the $SLTQ[T_1]$ is 26, the $SLTQ[T_2]$ is 4, and the $SLTQ[T_3]$ is 7.

$$WT[T_i]=SLTQ[T_i]-NTQ[T_i]*TQ \tag{3}$$

$$TWT=\sum_{i=1}^{n}WT[T_i] \tag{4}$$

T_i	Task i.
NTQ[Ti]=NTQi	The number of times the task Ti exploits the time quantum *TQ*.
$BT[T_i]=BT_i$	The burst time of the task T_i.
TQ	The time quantum.
n	The number of the tasks.
SLTQ[Ti]	The starting of the last time quantum of T_i.
WT[Ti]	The waiting time of task T_i.
TWT	The total waiting time of all tasks.
AVGWT	The average waiting time of the tasks in the run queue.
RST[Ti]	The residual time of T_i.

Table 1: CTQ Terminology.

TASK	BURST TIME
T1	24
T2	3
T3	3

Table 2: Example 1.

$$AVGWT = TWT / n \quad (5)$$

The changeable consideration

CTQ combines the benefit of low overhead round-robin scheduling with low average response time and low average waiting time, this depends on the size of the preselected time quantum. If we have n tasks in a round $r1$ and m tasks that have burst times equal to or less than the time quantum used in $r1$, then there are n-m tasks in the next round, where $n \geq m$. The residual time of the task T_i in the round number q is determined from the equation:

$$RST[T_i] = BT[T_i] - \sum_{k=1}^{q-1} TQ[k] \quad (6)$$

where $TQ[k]$ is the time quantum in the round number k. In each successive round we implement the equations with respect to the residual times of the survived tasks. Figure 1 represents the pseudo-code of the proposed algorithm.

Illustrative counter examples

To demonstrate the previous consideration we will take two cases of example. In the first one, the tasks arrive at the same time and in the second; the tasks arrive at different times. Consider the following set of tasks in (Table 3) that arrive at time 0, each of which with the length of the CPU burst time.

[1] $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to X.

When we apply the (CTQ) technique, the time quantum in the first round is equal to 25, $TQ[1]=25$.

(ROUND NO. 1)

($TQ[1] = 25$)

T1	T2	T3	T4	T5
0	23	48	73	98	100

The survived tasks are T2, T3, and T4 each of which with the length of the CPU burst time. After implementing the equations, we

Task ID	Residual Time
T1	0
T2	50
T3	68
T4	23
T5	0

obtain $TQ[2] = 25$, the Gantt Chart is:

```
While (ready queue <> null)
    *If (largest Burst Time (LBT) of tasks > Specific Burst Time(SBT))
          Execute tasks in Fixed Round Robin (FRR) manner
      Else
          Sort tasks on ready queue in an ascending order
          For I = α to OSTS
                Implement equations 1 – 5
                Compute average waiting at each I
          End For
          Choose I that gives the smallest average waiting time (I=TQ in this round)
          Compute Residual Times of tasks from equation 6
          If (new tasks arrive to ready queue)
                Go to *
      End If
End While
```

Figure 1: Pseudocode of Changeable Time Quantum (CTQ) algorithm.

Task ID	Burst Time
T1	23
T2	75
T3	93
T4	48
T5	2

Table 3: Example 2A.

Task ID	Residual Time	Waiting Time	Turnaround Time	Context Switches
T1	0	0	23	0
T2	0	123	198	2
T3	0	148	241	2
T4	0	125	173	1
T5	0	98	100	0

Table 4: CTQ policy of Example 2A.

Task ID	Burst Time	Arrival Time
T1	23	0
T2	75	20
T3	93	22
T4	48	50
T5	2	55

Table 5: Example 2B.

(ROUND NO. 2)

($TQ[2] = 25$)

T2	T3	T4
100	125	150	173

from the survived tasks,

Task ID	Residual Time
T1	0
T2	25
T3	43
T4	0
T5	0

the equations give $TQ[3] = 43$, the Gantt Chart is:

(ROUND NO. 3)

($TQ[3] = 43$)

T2	T3
173	198	241

In this example there are three rounds; at each one a different time quantum is used. Table 4 gives each task waiting time, turnaround time and each task's response time.

Now we will consider the above example when the tasks arrive at different arrival times. (Table 5) summarizes the burst time and arrival time of each task. We will compare the round-robin with fixed time quantum equal to 50 msec against our algorithm. (Tables 6, 7) show the policy of each algorithm.

In what follows, the number in parentheses in the comment field is the remaining service time for the process. In order of execution:

Job ID	Service Time	Arrival Time	Start Time	Finish Time	Preemption	Turnaround Time	Waiting Time	Context Switches
T1	23	0	0	23		23	0	0
T2	75 25	20	23 173	73 198	*end of quantum;* T3 *starts*	178	103	1
T3	93 43	22	73 198	123 241	*end of quantum;* T4 *starts*	219	126	1
T4	48	50	123	171		121	73	0
T5	2	55	171	173		118	116	0
Mean						131.8	83.6	

Table 6: Round-Robin policy of Example 2B.

Job ID	Service Time	Arrival Time	Start Time	Finish Time	TQ			Preemption	Turnaround Time	Waiting Time	Context Switches
					R1	R2	R3				
T1	23	0	0	23	23				23	0	0
T2	75 37	20	23 149	61 186		38	50	*end of quantum;* T3 *starts*	166	91	1
T3	93 55	22	61 186	99 241				*end of quantum;* T4 *starts*	219	126	1
T4	48	50	99	147					97	49	0
T5	2	55	147	149					94	92	0
Mean									119.8	71.6	

Table 7: CTQ policy of Example 2B.

Time	Ready Queue	Time Quantum	Comments
0	T1	TQ = 23	T1(23) arrives, run
20	T1, T2		T2(75) arrives and is appended to the queue, T1(3) continues to run
22	T1, T2, T3		T3(93) arrives and is appended to the queue, T1(1) continues to run
23	T2, T3		T1(0) finished, so T2(75) runs
50	T2, T3, T4	TQ = 38	T4(48) arrives and is appended to the queue, T2(48) continues to run
55	T2, T3, T4, T5		T5(2) arrives and is appended to the queue, T2(43) continues to run
61	T3, T4, T5, T2		The quantum expires, so T2(37) moves to the end of the queue and T3(93) runs
99	T4, T5, T2, T3		The quantum expires, so T3(55) moves to the end of the queue and T4(48) runs
147	T5, T2, T3	TQ = 50	T4(0) finished, so T5(2) runs
149	T2, T3		T5(0) finished, so T2(37) runs
186	T3		T2(0) finished, so T3(55) runs

We modified this algorithm by sorting the tasks in each round in an ascending order to profit from knocking out short jobs relatively faster in a hope to increase the throughput and reduce the average waiting time [2,14]. Sorting tasks gives more improvement in scheduling criteria.

Simulation Studies and Results

This work is considered to be a modified RR algorithm in a small specific portion of the burst times of tasks. To demonstrate the effectiveness of the *CTQ*, we built a scheduling simulator that is a user-space program which takes five inputs, the scheduling algorithm, the number of tasks, the burst time, the arrival time of each task, and the first time quantum that will be used in the traditional round-robin. The simulator randomly assigns burst times and arrival times to tasks.

To measure the effectiveness, we ran simulations for the proposed algorithm against fixed round-robin algorithm and BRR[2] considered on 30 different combinations of n and *BT's*, the burst times of the tasks varying from 1 to Specific Burst Time(SBT) = 100 msec . For each set of (n, *BT*), we ran different number of tasks with different CPU lengths and different arrival times. In this research, the task arrival was modeled as a *Poisson random process*. Hence, the inter-arrival times are *exponentially distributed*. A task arrival generator was developed to take care of the process of random arrival of different tasks to the system. The generator produces the inter-arrival times utilizing some specific mean (*arrival intensity*) of the distribution function. We split our simulation into two cases; in the first case the tasks arrive at the same time, we call this set of 30 processes DATA1 and in the second case, the tasks arrive at different times. We call the second set of 30 processes DATA2.

To avoid unnecessary context switches, we ranged the selected *TQ* from α up to *OSTS*. Here in DATA1 and DATA2; *OSTS* is equal to 50 msec and α is equal to 1/2 *OSTS*. (Figures 2, 3 and 4) show the improvement of our algorithm over the two algorithms in first case, and (Figures 5 and 7) show the improvement in second case.

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 Changeable Time Quantum (CTQ) based on dynamic-time-quantum was designed to solve all critical previously mentioned problems in a practical, simple and applicable manner.

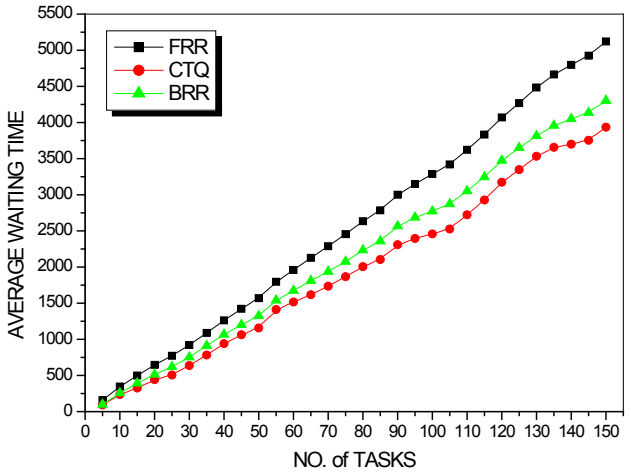

Figure 2: Average waiting time of DATA1.

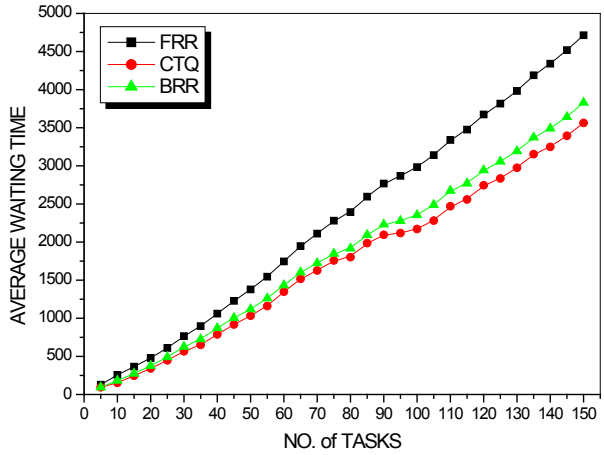

Figure 5: Average waiting time of DATA2.

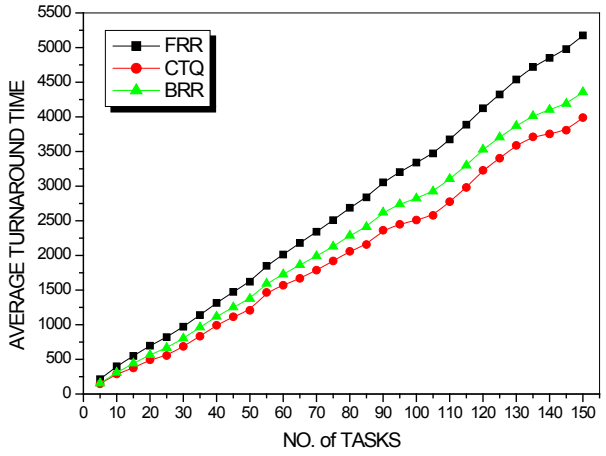

Figure 3: Average turnaround time of DATA1.

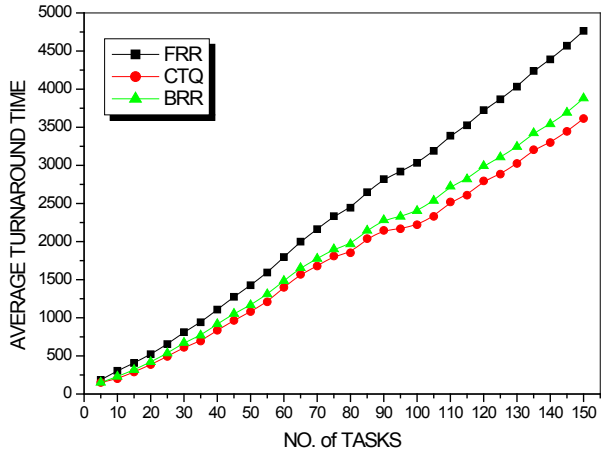

Figure 6: Average turnaround time of DATA2.

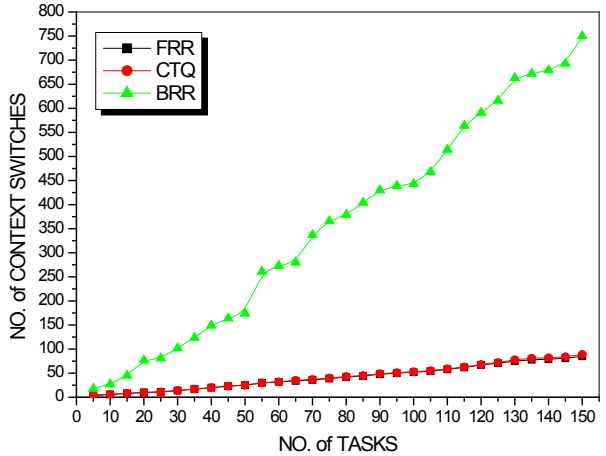

Figure 4: No. of context switches of DATA1.

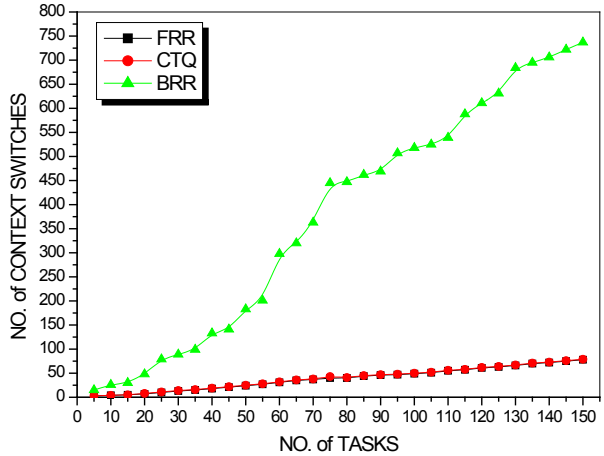

Figure 7: No. of context switches of DATA2.

The above comparisons show that the proposed algorithm provides much better results than other approaches based on fixed time quantum in all scheduling criteria.

Conclusion

In this study, a dynamic method was used to improve scheduling criteria in a uni-processor. In this work we don't use a fixed time quantum as usually used in scheduling algorithms, but we make the value of the time quantum changeable in each round according to the residual times of the tasks. The candidate time quantum in each round gives the smallest average waiting time consequently the smallest average turnaround time. Also we take into account the overhead resulting from the unnecessary context switches, so we try to keep the number of context switches as low as possible.

References

1. Demers A, Keshav S, Shenker S (1989) Analysis and Simulation of a Fair Queueing Algorithm. In Proceedings of ACM SIGCOMM '89, Austin, TX, 1–12.
2. Helmy T, Dekdouk A (2007) 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.
3. Caprita B, Chan, WC, Nieh J (2003) Group Round-Robin: Improving the Fairness and Complexity of Packet Scheduling. Technical Report CUCS-018-03, Columbia University.
4. Nieh J, Vaill C, Zhong H (2001) Virtual-Time Round-Robin: An O(1) Proportional Share Scheduler. In Proceedings of the 2001 USENIX Annual Technical Conference.
5. Abeni L, Lipari G, Buttazzo G (1999) Constant bandwidth vs. proportional share resource allocation. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Florence, Italy.
6. Jeffay K, Donelson FS, Anderson J, Moorthy A (1998) Proportional share scheduling of operating system services for realtime application. In Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain.
7. Bennett J, Zhang H (1996) WFQ: Worst-case Fair Weighted Fair Queueing. in Proceedings of INFOCOM '96, San Francisco, CA.
8. Shreedhar M, Varghese G (1995) Efficient Fair Queueing Using Deficit Round-Robin in Proceedings of ACM SIGCOMM '95, 4: 231-242.
9. Parekh A, Gallager R (1993) A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. IEEE/ACM Transactions on Networking, 1: 344–357.
10. Essick R, (1990) An Event-Based Fair Share Scheduler. in Proceedings of the Winter 1990 USENIX Conference, USENIX Berkeley, CA, USA, 147–162.
11. Henry G (1984) The Fair Share Scheduler. AT&T Bell Laboratories Technical Journal, 63: 1845–1857.
12. Harwood A, Shen H (2001) Using fundamental electrical theory for varying time quantum uni-processor scheduling. Journal of Systems Architecture: the EUROMICRO Journal, Volume 47, issue 2, Feb.
13. Silberschatz A, Galvin PB, Gagne G (2005) Operating Systems Concepts. John Wiley and Sons. 6Ed.
14. Tanenbaum A (2001) Modern Operating Systems. Second Ed.
15. Caprita B, Chan WC, Nieth J, Stein C et al. (2005) Group ratio round-robin: O(1) proportional share scheduling for uni-processor and multiprocessor systems. In USENIX Annual Technical Conference.
16. Kay J, Lauder P (1988) "A Fair Share Scheduler" Communications of the ACM, 31: 44–55.
17. Rami JM (2009) Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes. American Journal of Applied Sciences, 6: 1831-1837.
18. Alam B, Doja M.N, Biswas R (2008) Finding Time Quantum of Round Robin CPU Scheduling Algorithm Using Fuzzy Logic. Proceedings of the International Conference on Computer and Electrical Engineering, pp. 795-798.
19. Stallings W (2006) Operating Systems Internal and Design Principles. 5th Edition.
20. Tanenbaum AS, Albert SW (2005) Operating Systems Design and Implementation. Second Edition.