International Journal of Computer Science and Information Technology Research (IJCSITR)
Vol. 1, Issue 1, pp: (65-73), Month: October-December 2013, Available at: www.researchpublish.com

Towards Minimizing Processes Response Time in Interactive Systems

Samih M. Mostafa[1, 2], Shigeru Kusakabe[2]
[1]Faculty of Science, Mathematics Department, South Valley University, Quena, Egypt.
[2] Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan.

Abstract: **This study considers the process scheduling problem of minimizing the response time (RT) of processes. In This study we schedule a batch of n processes, for servicing on a single resource, in such a way that the response time is minimized. RT minimization finds its applications for process scheduling in manufacturing interactive systems in computer and networks systems for the stabilized QoS, and in other fields where it is desirable to minimize RT of processes with different weights for priorities. We formulate a RT problem as an integer programming problem. Numerical testing shows that proposed technique significantly outperforms existing algorithms. In this paper we use task and process terms interchangeably.**

Keyword: **Process/tasks scheduling, residual time, survived processes, cyclic queue.**

I. INTRODUCTION

In order to design a scheduling algorithm, it is necessary to have some idea of what a good algorithm should do. Some goals depend on the environment (batch, interactive, or real time), but there are also some that are desirable in all cases. For interactive systems, especially timesharing systems and servers, different goals apply. The most important one is to minimize response time, that is the time between issuing a command and getting the first response [4]. This paper will analyse only the process response time for uni-processor systems. To this end, we utilize the following assumptions throughout this paper to simplify the problem formulation:

- Processes are belong to interactive environment (i.e., processes 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 intentionally ran forever, due to a program bug, one process might shut out all the others indefinitely. Preemption is needed to prevent this behavior,
- Processes are of variable size in terms of number of instructions which may range anywhere from instructions up to thousands or greater for some interactive processes.
- No process is rated more important than any other process,
- Each process is considered to be independent of all others, i.e., there is no communication between processes running on the processor,
- The CPU cost of each process is assumed to be known.
- New processes 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 process scheduling in interactive systems that apply when these assumptions are made is the round-robin (RR). The round-robin algorithm is cosidered to be a preemptive scheduler, it is opposite to non-preemptive algorithms. 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 [9, 1, 14, 8, 10, 3, 13, 11]. In addition, RR algorithms have low scheduling overhead of $O(1)$, which means scheduling the next process takes a constant time [7, 6, 12, 17, 18].

Briefly RR scheduling dose not reorder the processes but allows preemption to occur so that processes 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 throgh the cyclic queue. The terms process and process 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 process response time and

ii) average process waiting time (i.e., the average amount of time a process waits while other processes are being processed). 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 [5].
Latest algorithms [2, 15, 16] 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 processes. Using *Changeable Time Quantum (CTQ)* gives significant improvement in the above criteria.

II. 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 processes. We define the terminology list we use in TABLE 1.

TABLE 1: CTQ Terminology

T_i	Process i.
$NTQ[T_i] = NTQ_i$	The number of times the process T_i exploits the time quantum TQ.
$BT[T_i] = BT_i$	The burst time of the process T_i.
TQ	The time quantum.
n	The number of the processes.
$SLTQ[T_i]$	The starting of the last time quantum of T_i.
$WT[T_i]$	The waiting time of process T_i.
TWT	The total waiting time of all processes.
$AVGWT$	The average waiting time of the processes in the run queue.
$RST[T_i]$	The residual time of T_i.

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 $\alpha \le OSTS$.

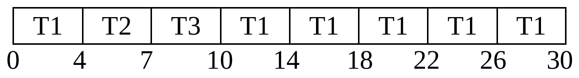
$$NTQ[T_i] = \begin{cases} \left\lceil \dfrac{BT[T_i]}{TQ} \right\rceil^{1} & \text{if} \quad BT[T_i] \neq l * TQ \\ & \qquad l = 1,2,3,\dots \\ \dfrac{BT[T_i]}{TQ} - 1 & \text{if} \quad BT[T_i] = l * TQ \\ & \qquad l = 1,2,3,\dots \end{cases} \tag{1}$$

TABLE 2 exhibits an example, in which each process with its burst time:

TABLE 2: Example 1

PROCESS	BURST TIME
T1	24
T2	3
T3	3

If we use a time quantum of 4 ms. we see from the Gantt Chart:



that the *NTQ[T₁]* is 5, the *NTQ[T₂]* is 0, and the *NTQ[T₃]* 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.

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

In the above example the *SLTQ[T1]* is 26, the *SLTQ[T2]* is 4, and the *SLTQ[T3]* is 7.

$$WT[T_i] = SLTQ[T_i] - NTQ[T_i] * TQ \qquad (3)$$

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

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

III. 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 processes in a round $r1$ and m processes that have burst times equal to or less than the time quantum used in $r1$, then there are n-m processes in the next round, where $n \geq m$. The residual time of the process T_i in the round number q is determined from the equation:

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

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 processes.

IV. ILLUSTRATIVE COUNTER EXAMPLES

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

TABLE 3: Example 2A

Process Id	Burst Time	Response Time
T1	23	22
T2	75	57
T3	93	8
T4	48	16
T5	2	1

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 processes are T2, T3, and T4 each of which with the length of the CPU burst time.

Process Id	Residual Time	Response Time
T1	0	22
T2	50	not yet
T3	68	56
T4	23	89
T5	0	99

After implementing the equations, we obtain *TQ[2]* = 25, the Gantt Chart is:

(ROUND NO. 2)
($TQ[2]$ =25)

T2	T3	T4

100 125 150 173

from the survived processes,

Process Id	Residual Time	Response Time
T1	0	22
T2	25	not yet
T3	43	56
T4	0	89
T5	0	99

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. The following table gives each process response time during execution

Process Id	Residual Time	Response Time
T1	0	22
T2	0	180
T3	0	56
T4	0	89
T5	0	99

Now we will consider the above example when the processes arrive at different arrival times. TABLE 4 summarizes the burst time, response time, and arrival time of each process. We will compare the round-robin with fixed time quantum equal to 50 msec against our algorithm. TABLE 5 and 6 show the policy of each algorithm.

TABLE 4: Example 2B

Process Id	Burst Time	Response Time	Arrival Time
T1	23	22	0
T2	75	57	20
T3	93	8	22
T4	48	16	50
T5	2	1	55

TABLE 5: Round-Robin policy of Example 2B

Process ID	Service Time	Response Time	Arrival Time	Start Time	Finish Time	Preemption	Turnaround Time	Waiting Time	Response Time
T1	23	22	0	0	23		23	0	22
T2	75 25	57	20	23 173	73 198	end of quantum; T3 starts	178	103	160
T3	93 43	8	22	73 198	123 241	end of quantum; T4 starts	219	126	59
T4	48	16	50	123	171		121	73	89
T5	2	1	55	171	173		118	116	117
Mean							131.8	83.6	89.4

TABLE 6: CTQ policy of Example 2B

Process ID	Service Time	Response Time	Arrival Time	Start Time	Finish Time	TQ			Preemption	Turn-around Time	Waiting Time	Response Time
						R1	R2	R3				
T1	23	22	0	0	23	23				23	0	22
T2	75 37	57	20	23 149	61 186		38	50	End of quantum; T3 starts	166	91	148
T3	93 55	8	22	61 186	99 241				End of quantum; T4 starts	219	126	47
T4	48	16	50	99	147					97	49	65
T5	2	1	55	147	149					94	92	93
Mean										119.8	71.6	75

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

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	TQ = 38	T1(0) finished, so T2(75) runs
50	T2, T3, T4		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	TQ = 50	The quantum expires, so T3(55) moves to the end of the queue and T4(48) runs
147	T5, T2, T3		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

V. SIMULATION STUDIES

To demonstrate the effectiveness of the CTQ, we built a scheduling simulator that is a user-space program which takes six inputs, the scheduling algorithm, the number of processes, the burst time, the arrival time, response time of each process, and the first time quantum that will be used in the traditional round-robin. The simulator randomly assigns burst times, arrival times, response times to processes.

To measure the effectiveness, we ran simulations for the proposed algorithm against fixed round-robin algorithm considered on 30 different combinations of *n* and *BT*'s, the burst times of the processes varying from 1 to 500 tu. For each set of (*n*, *BT*), we ran different number of processes with different CPU lengths, response times, and arrival times. In this research, the process arrival was modeled as a *Poisson random process*. Hence, the inter-arrival times are *exponentially distributed*. A process arrival generator was developed to take care of the process of random arrival of different processes to the system. The generator produces the inter-arrival times utilizing some specific mean (*arrival intensity*) of the distribution function. We call this set of 30 processes DATA1. we ran the simulation in three different cases:

i) best response (i.e., the response of process considered to be at the beginning of its execution),
ii) random response (i.e., the response of process considered to be at any time during its execution) and
iii) worst response (i.e., the response of process considered to be at its end of execution).

Figures 1, 2, and 3 present the previous cases respectively. To avoid unnecessary context switches, we ranged the selected TQs from α up to *OSTS*. Here in DATA1 *OSTS* is equal 50 msec and α is equal ½ *OSTS*. Also to confirm the improvement of our technique, we assume that the process response time ≤ α as shown in figure 4. In this research we took into account the waiting time and the turnaround time as shown in figures 5 and 6 respectively.

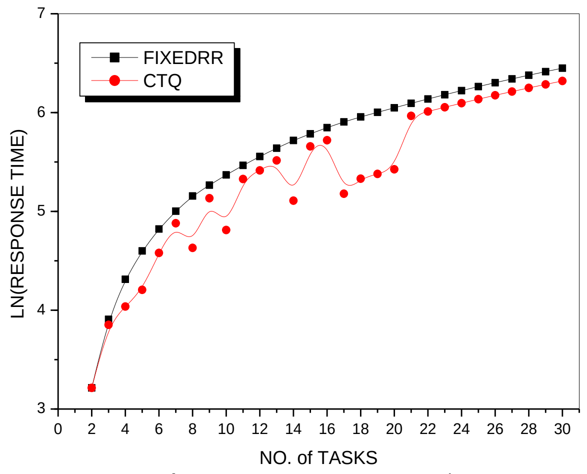


Figure 1: DATA1 Best Response Time

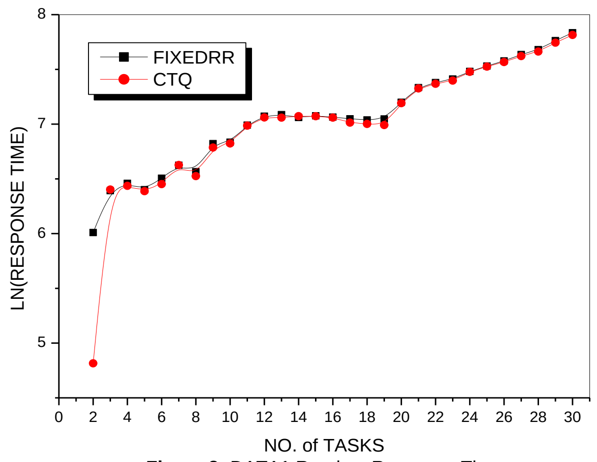


Figure 2: DATA1 Random Response Time

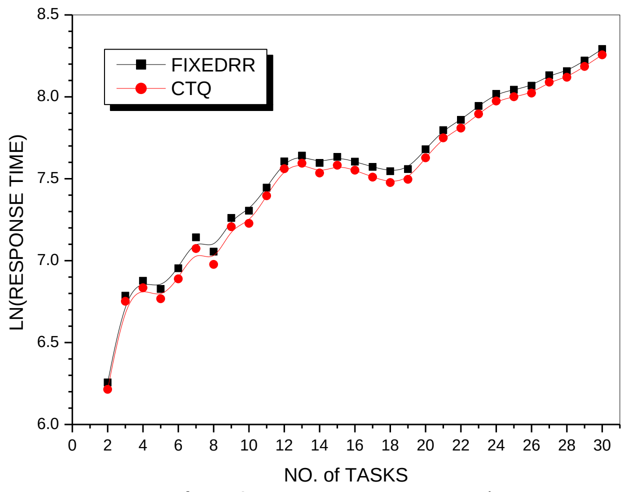


Figure 3: DATA1 Worst Response Time

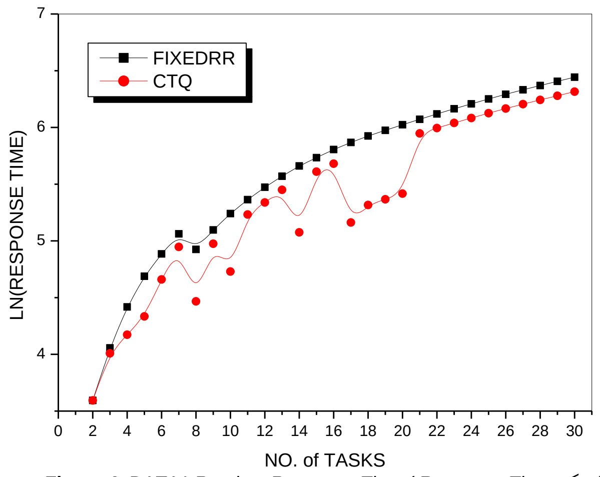


Figure 4: DATA1 Random Response Time (Response Time $\leq\alpha$)

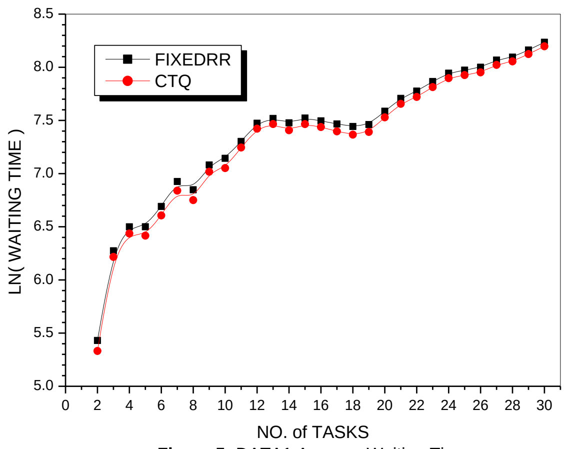


Figure 5: DATA1 Average Waiting Time

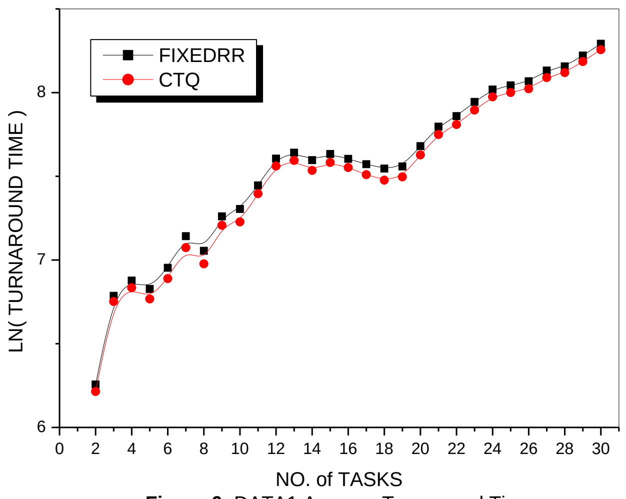


Figure 6: DATA1 Average Turnaround Time

REFERENCES

[1]. A. Demers, S. Keshav, and S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm,” in Proceedings of ACM SIGCOMM ’89, Austin, TX, Sept. 1989, pp. 1–12.

[2]. A. Harwood and H. Shen, “Using fundamental electrical theory for varying time quantum uni-processor scheduling,” Journal of Systems Architecture: the EUROMICRO Journal, Volume 47, issue 2, Feb. 2001.

[3]. A. Parekh and R. Gallager, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case,” IEEE/ACM Transactions on Networking, 1(3), June 1993, pp. 344–357.

[4]. A. Silberschatz, P.B. Galvin, and G. Gagne, “Operating Systems Concepts,” John Wiley and Sons. 6Ed 2005.

[5]. A. Tanenbaum, “Moden Operating Systems,” Second ed., 2001.

[6]. 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.

[7]. 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.

[8]. G. Henry, “The Fair Share Scheduler,” AT&T Bell Laboratories Technical Journal, 63(8), Oct. 1984, pp. 1845–1857.

[9]. J. Bennett and H. Zhang, “WFQ: Worst-case Fair Weighted Fair Queueing,” in Proceedings of INFOCOM ’96, San Francisco, CA, Mar. 1996.

[10]. J. Kay and P. Lauder, “A Fair Share Scheduler,” Communications of the ACM, 31(1), Jan. 1988, pp. 44–55.

[11]. J. Nieh, C. Vaill and, H. Zhong, “Virtual-Time Round-Robin: An O(1) Proportional Share Scheduler,” In Proceedings of the 2001 USENIX Annual Technical Conference, June 2001.

[12]. 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.

[13]. M. Shreedhar and G. Varghese, “Efficient Fair Queueing Using Deficit Round-Robin,” in Proceedings of ACM SIGCOMM ’95, 4(3), Sept. 1995. PP. 231-242.

[14]. R. Essick, “An Event-Based Fair Share Scheduler,” in Proceedings of the Winter 1990 USENIX Conference, USENIX Berkeley, CA, USA, Jan. 1990, pp. 147–162.

[15]. Rami J Matarneh, “Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes,” American Journal of Applied Sciences, 6(10): 1831-1837 , 2009.

[16]. 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 November 2007, at the Gulf International Convention Center, Bahrain.

[17]. Samih M. Mostafa, S. Z. Rida & Safwat 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.

[18]. Samih M. Mostafa, S. Z. Rida & Safwat 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).