ISSN No. 0976-5697

Volume 4, No. 1, January 2013 (Special Issue)

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Improving the Performance of CPU Scheduling in Interactive Systems

Mahesh Ubale* and Mujeeb Rahaman
Department of Computer Engineering/Information Technology
St. Vincent Pallotti College of Engineering & Technology Nagpur, India
mahesuhbale@gmail.com*, mujeebrahman1984@gmail.com

Abstract: This paper gives a new algorithm for round robin CPU scheduling of processes (jobs) having different arrival times. The performance parameters such as response time, average waiting time, average turnaround time, number of context switches are compared between original round robin scheduling algorithm (as discussed in standard textbooks on operating system) and our approach. We have made a time quantum fixed for a given round based on the average burst time of the number of processes present at the beginning of that round. By making time quantum dynamic we have proved that the performance gets improved as compared to the original round robin algorithm.

Keywords: CPU scheduling, Context Switching, Time Quantum, average waiting time, burst time, average turnaround time

I. INTRODUCTION

When a computer is multi programmed, it frequently has multiple processes competing for the CPU at the same time. This situation occurs whenever two or more processes are simultaneously in the ready state. If only one CPU is available, a choice has to be made which process to run next. The part f the operating system that makes the choice is called the scheduler and the algorithm it uses is called the scheduling algorithm. [2]

In a single-processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The idea is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request. [1]

In a simple computer system, the CPU then just sits idle. All this waiting time is wasted; no useful work is accomplished. With multiprogramming, we try to use this time productively. Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. Every time one process has to wait, another process can take over use of the CPU. [1]

Scheduling of this kind is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design. [1]

II. SCHEDULING CRITERIA

Many criteria have been suggested for comparing CPU scheduling algorithms. Which characteristics are used for comparison can make a substantial difference in which algorithm is judged to be best. The criteria include the following:

A. CPU utilization:

We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

B. Throughput:

If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called *throughput.*

C. Turnaround time:

The interval from the time of submission of a process to the time of completion is the *turnaround time.* 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.

D. Waiting time:

Waiting time is the sum of the periods spent waiting in the ready queue.

E. Response time:

In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called *response time,* is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device. [1]

It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time. In most cases, we optimize the average measure. However, under some circumstances, it is desirable to optimize the minimum or maximum values rather than the average. For example, to guarantee that all users get good service, we may want to minimize the maximum response time. [1]

Investigators have suggested that, for interactive systems (such as timesharing systems), it is more important to minimize the *variance* in the response time than to minimize the average response time. A system with reasonable and *predictable* response time may be

CONFERENCE PAPER
National Level Conference on
"Trends in Advanced Computing and Information Technology (TACIT - 2012)"
on 2nd September 2012
Organized by
St. Vincent Pallotti College of Engineering and Technology, Nagpur, India

considered more desirable than a system that is faster on the average but is highly variable. However, little work has been done on CPU-scheduling algorithms that minimize variance. [1]

III. SCHEDULING IN INTERACTIVE SYSTEM

A. Algorithm Goals:

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. Under all circumstances, fairness is important. Comparable processes should get comparable service. Another general goal is keeping all parts of the system busy when possible. [2]

For interactive systems, especially timesharing systems and servers different goals apply. The most important is one to minimize *response time* i.e. the time between issuing a command and getting the result. On a personal computer where a background process is running (e.g. reading and storing e-mail from the network), a user request to start a program or open a file should take precedence over the background work. Having all interactive requests go first will be perceived as good service. [2]

A somewhat related issue is what might be called *proportionality*. Users have an inherent (but often incorrect) idea of how long things should take. When a request that is perceived as complex takes a long time, users accept that, when a request that is perceived as simple takes a long time, users get irritated. For example, if clicking on an icon that calls up an Internet provider using an analog modem takes 45 seconds to establish a connection, the user will probably accept that as a fact o life. On the other hand, if clicking on an icon that breaks the connection takes 45 seconds, the user will probably be swearing a blue streak by the 30-second mark and frothing at the mouth by 45 seconds. This behaviour is due to the common user perception that placing a phone call and getting a connection is supposed to take a lot longer than just hanging up. In some cases (such as this one), the scheduler cannot do anything about the response time, but in other cases it can, especially when the delay is due to a poor choice of process order. [2]

B. Round Robin Algorithm (Original):

The *round-robin (RR) scheduling algorithm* is designed especially for timesharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a *time quantum* or *time slice*, is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. [1]

To implement RR scheduling, we keep the ready queue as a FIFO queue of processes. 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 1 time quantum, and dispatches the process. [1]

One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, 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. [1]

The conclusion can be formulated as follows: setting the quantum too short causes too many process context switches and lowers 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. [2]

C. Proposed Round Robin Algorithm:

Given a set of processes ready to run on CPU, their arrival time in ready queue and burst time, original round robin algorithm fixes a time quantum for all processes in the system. Our approach, on the other hand, dynamically computes the time quantum for a given set of processes present in the ready queue at a given time (called as round), which equals the average of the burst times of all the processes. These processes will use this time quantum only once for execution on CPU.

Suppose n processes were present in the ready queue at time t_1 and TQ_1 is the first round time quantum computed by averaging the burst times of these n processes. Let, m be the number of processes $(m<n)$ having burst time less than TQ_1 so they terminated normally after consuming time quantum and r be the number of processes entered in to ready queue till all n processes uses time quantum once. So, $(n-m)$ number of processes will reappear in ready queue for their remaining burst time.

At the end of first round i.e. when n^{th} process is about to leave CPU after consuming TQ_1, we have $r+(n-m)$ processes waiting in ready queue. So, TQ_2 will be the average of remaining burst times of these waiting processes.

IV. CASE STUDIES AND RESULT

We have used three examples to compare the results of original and our proposed algorithm. First table includes data given for the three examples and the remaining three table shows the result comparison.

Table I: Examples For Comparison Of Original And Proposed Approach

Example 1

Pid	AT	BT
P1	0	6
P2	0	8
P3	0	7
P4	0	3

Example 2

Pid	AT	BT
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Example 3

Pid	AT	BT
P1	0	10
P2	5	12
P3	8	20
P4	10	15
P5	15	30

Table: 2 Result fot Example 1

Approach	Performance Parameters for Example 1		
	Average Waiting Time	Number of Context Switches	Average Turn Around Time
Round Robin (Orignal) TQ=3	16.667	5	24.667
Round Robin (as per our approach)	12.5	2	18.5

CONFERENCE PAPER
National Level Conference on
"Trends in Advanced Computing and Information Technology (TACIT - 2012)"
on 2nd September 2012
Organized by
St. Vincent Pallotti College of Engineering and Technology, Nagpur, India

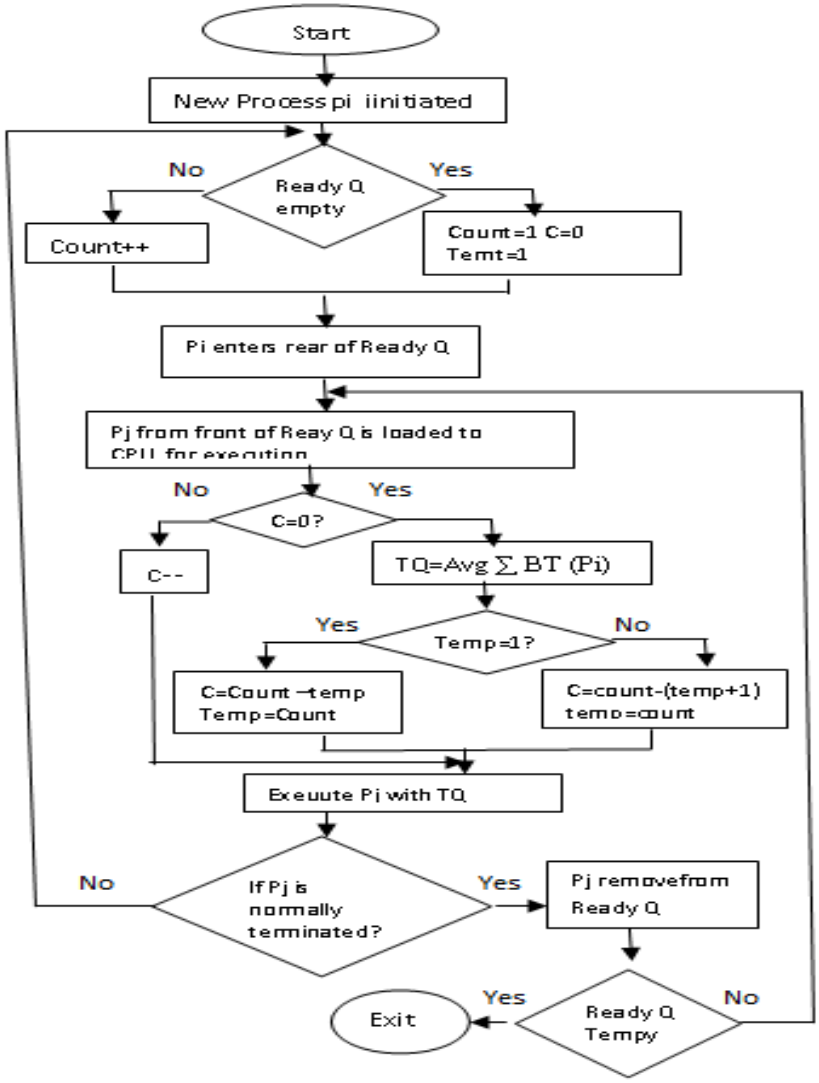

Table: 3 Results for Example 2

Approach	Performance Parameters for Example 2		
	Average Waiting Time	Number of Context Switches	Average Turn Around Time
Round Robin (Orignal) TQ=4	11.75	4	18.25
Round Robin (as per our approach)	9.25	1	15.75

Table: 4 Results for Example 3

Approach	Performance Parameters for Example 3		
	Average Waiting Time	Number of Context Switches	Average Turn Around Time
Round Robin (Orignal) TQ=5	35.6	11	53
Round Robin (as per our approach)	24.2	2	41

V. COCLUSIONS

The comparative results of three case study examples shown here proves that using a dynamic time quantum improves the performance of original round robin algorithm to a great extent. The computation of time quantum consumes some amount of time and that is a pure overhead for the system. We have fixed a time quantum for a given set of processes than computing it for every new process to be loaded to CPU for execution (as used in the approach proposed by Abbas Noon, Ali Kalakech, Seifedine Kadry [3]).

Another issue is with burst time. As actual burst time of every new process entering in ready queue will not be known in advance in real systems, we can use a predictive method to compute it.

VI. REFERENCES

[1] Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, 7th ed., John Wiley & Sons (Asia) Pte. Ltd., Singapore.

[2] Andrew S. Tanenbaum, Modern Operating Systems, 2nd ed., Prentice-Hall of India Private Limited New Delhi 110 001.

CONFERENCE PAPER
National Level Conference on
"Trends in Advanced Computing and Information Technology (TACIT - 2012)"
on 2nd September 2012
Organized by
St. Vincent Pallotti College of Engineering and Technology, Nagpur, India

[3] Abbas Noon, Ali Kalakech, Seifedine Kadry A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean, IJCSI, International Journal of Computer Science Issues Vol. 8, Issue 3, No. 1, May 2011.

CONFERENCE PAPER
National Level Conference on
"Trends in Advanced Computing and Information Technology (TACIT - 2012)"
on 2nd September 2012
Organized by
St. Vincent Pallotti College of Engineering and Technology, Nagpur, India