Volume 2, Issue 11, November 2012 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Process Scheduling Using Repeatative Subtraction

(An algorithm dependent on the number of processes executed since its arrival)

Rahul Singh
Computer Science and Engineering
M. M. M. Engineering College
Gorakhpur, U.P. India

Umesh Chandra Jaiswal
Computer Science and Engineering
M. M. M. Engineering College
Gorakhpur, U.P. India

Abstract— This paper puts emphasis over a new scheduling approach to facilitate jobs for scheduling. This algorithm along with solving starvation problem, it provides a better and efficient waiting time. This algorithm process's scheduling of processes to run on operating system next. This algorithm is non-pre-emptive in nature, similar to SRPT (Shortest Remaining Processing Time). This algorithm doesn't concerns with waiting time of a process to prioritize the job requesting CPU time as other starvation free traditional algorithm does that is HRRN (Highest Response Ratio Next) and Rail Road Strategy. This algorithm uses a new approach, it uses a constant which is adjustable as per the requirements but it must be adjusted prior to implementation of the proposed algorithm.

Approach: In this algorithm in place of waiting time as a deciding scheduling parameter, it uses a constant which increments every time for all the processes that had arrived and are in ready queue before the CPU time is allocated to the process waiting at top in the ready queue and their priority is updated but not for the processes which have arrived during the execution of the particular process under execution. In nutshell it's a SRPT (Shortest Remaining Processing Time) with small modification but with little computational overhead.

Keywords: CPU scheduling, shortest job first, highest response ratio next, starvation, scheduling overhead, Rail Road Strategy.

I. INTRODUCTION

If only one CPU is available, a choice has to be made which process to run next. The part of the operating system that makes the choice is called the scheduler and algorithm it uses is called the scheduling algorithm. Nearly all processes alternate bursts of computing (i.e. execution) with (disk) I/O request. Typically CPU run for a while without stopping, then a system call is made to read from a file or to write a file. With system call completion, the CPU computes again until it needs more data or has to write more data. When CPU spends most time in computing it's called compute-bound and if it spends most time waiting for I/O it's called I/O-bound.

SRPT (Shortest Remaining Processing Time) provides optimal mean round around time for all requests. An optimal algorithm that minimizes the mean response time is SRPT (Shortest Remaining Processing Time) scheduling policy [1] and [2]. SRPT presented starvation problem, i.e. a longer process may not get CPU time till smaller processes keep on arriving and it may cause longer process to starve for CPU time. Starvation problem of SRPT was already solved by HRRN (Highest Response Ratio Next), but it also incurred computation overhead as with an every clock tick priority need to be updated and respective process sorted [3]. And on the other hand Rail Road Strategy have provided better waiting time than Highest Response Ratio Next (HRRN) in some cases, equal waiting time in most cases, very negligible more waiting time in some cases and a constant scheduling overhead [4], but it also uses waiting time as the key factor.

II. OTHER SCHEDULING ALGORITHMS

A. First Come First Serve (FCFS)

FCFS is non-preemptive in nature. This policy is primarily used in number of application domains such as scheduling [5] and operating system [6] in order to ensure fairness. Due to FCFS policy turn around time calculates normally to be high, same can be seen for its average waiting time. We see that shorter processes are penalized if they follow any larger process in execution order.

B. Shortest Remaining processing Time (SRPT)

SRPT is the policy which provides optimal mean round around time for all the process requesting CPU. It's an optimal algorithm for minimizing the mean response time [1] and [2]. Job size is known in prior for web servers [7], [8] and [9] and SRPT implemented for improved user-perceived performance. Process with least remaining time will be served first. It incurs two problems, jobs with large size may get starve and dispatcher must know size of jobs beforehand.

C. Multilevel Queue Scheduling

Multilevel queue scheduling has multiple queues and each queue has its own scheduling algorithm. Process may be divided on the basis of type of process i.e. interactive process or batch process. In addition to this there must be scheduling among the queues. In this policy each queue gets portion of the CPU time.

D. Round Robin

Round robin policy sends each of the incoming requests to the next server in the list. And this algorithm include time slice also referred as time quantum, time slice is the time for which each requests will be processed before allocating CPU time to another requests and while CPU is allocated to any other process previously executing which saves its progress in Process Control Block (PCB). While CPU is allocated some time is wasted i.e. called context switch time. Every incoming request gets their time slice in accordance of FCFS policy.

E. Highest Response Ratio Next

This algorithm presents no starvation. This algorithm schedules the requests as per the following equation (1) as:

Priority=1+waiting time/burst time (1)

But this algorithm offers high computation overhead [4]. There had been numerous modifications done on HRRN to suit to particular application need [3], [10] and [11].

F. Rail Road Strategy for SJF Starvation problem

This algorithm decides which process to be run next on basis of equation (2) stated as:

P1b+P2w < P2b+P1w (2)

Where P1b, P1w are burst time and waiting time of a process respectively.P2b, P2w are burst and waiting time of another process.. Process whose burst time is added to wait time of another process to achieve the smaller value of the two should be given preference in scheduling order. If both the sides are equal then the process with shorter burst time is allocated to CPU. Starvation is solved because as waiting time of process having longer burst time increases, their chance to run next on the CPU increases [4].

III. THE PROPOSED ALGORITHM

Initially, burst time of the processes serves as the priority of each individual process arriving and requesting CPU time. As per the priority at that instant all processes are sorted in a non-decreasing order. Our algorithm is analogous to that SJF but it includes a more parameter, a constant 'r' that is subtracted consistently (depending on the algorithm described in section IV. Implementation) from the priority value of the process, minimum the value of priority maximum the priority the process possess. Equation (3) of our algorithm is stated as:

Priority←burst time

Priority← Priority-r (3)

r: A constant which can be any value, depending on the requirement. If you concern is only and only performance than it can also be a decimal value. But we should define the value of constant 'r' prior to implementation.

Our algorithm gives better result later it is described in section Implementation i.e. in section IV.

IV. IMPLEMENTATION

We require a single array and any new element arriving is inserted at appropriate place as per its burst time using insertion sort. Every time a process gets the CPU time and begins to execute and on successful execution every process gets its priority updated by the steps of our scheduling scheme described below which uses equation (3) for updating the priority of all the process arrived and yet not executed.

We have written the following steps which gives an idea of the order of execution.

Step 1: Initially, first process gets the CPU time.

Step 2: Insertion sort () on all process which had arrived on or before the execution of the first process.

Step 3: Now select a process from the top of the array to allocate CPU time.

Step 4: Priority of all the process is updated.

Step 5: Insertion sort () on all incoming processes, i.e. all processes arriving after the arrival of current process being executed and on or before its complete execution.

Step 6: Repeat step 3 through 5 till processes keeps on arriving.

In following example it is assumed that value of constant r=2.

Final result of SJF algorithm and the execution order it follows [Figure 2].

Figure 3 through 8 illustrates our proposed algorithms. Initially in [Figure 3] first process is allocated CPU time. All process with id 3 and 2 are allocated next as per our scheme and meanwhile priority of process with process id 2 becomes 6 but still greater than process 3 priority 4 so process 3 is allocated CPU Time [4]. In similar way for process 4 every time a process executes since its arrival it priority value decreases and its priority increases [Figure 5], [Figure 6], [Figure 7] and [Figure 8].

Process id	1	2	3	4	5	6	7
Arrival time (milliseconds)	0	5	10	15	20	25	30
Burst time (milliseconds)	10	8	4	20	15	5	18

Figure 1. An example of processes.

Process id	1	3	2	5	6	7	4

Figure 2. Execution order of process by SJF

Process id	1						

Figure 3. Initially only process 1 executed

Process id	1	3	2				

Figure 4. Here priority of process with process id 4 is 20

Process id	1	3	2	5			

Figure 5. Here priority of process with process id 4 becomes 18

Process id	1	3	2	5	6		

Figure 6. Here priority of process with process id 4 becomes 16

Process id	1	3	2	5	6	4	

Figure 7. Here priority of process with process id 4 is 16 and is less than process 7's priority i.e. 18 so process 4 is scheduled.

Process id	1	3	2	5	6	4	7

Figure 8. Final execution order of process by our proposed algorithm

V. COMPARISON AND PERFORMANCE

To calculate the effect of our proposed algorithm in comparison to Shortest Job First (SJF) we have written a simulator in C language which gives the result for average waiting time and corresponding graphs are plotted [Figure 9], [Figure 10] and [Figure 11]. We know that no algorithm can offer better performance than the Shortest Job First (SJF) but Shortest Job First (SJF) is only for experimental use and comparison purpose. As in this algorithm we are using insertion sort on the arriving process, so insertion costs O(n). This overhead is incurred in any of the scheduling algorithm currently present which aims at specific criteria like less waiting time and fair share allocation. In our strategy once a process is inserted in an ordered manner would not change its order of execution. Therefore, number of process remaining same, Highest Response Ratio Next (HRRN) would have to sort them again to determine the one with the highest priority [3].

We have considered an initial set of process being generated randomly and the arrival time of all the process is also generated randomly. For calculating the effect of our strategy we have considered 200 processes and have made their graph [Figure 9], [Figure 10] and [Figure 11]. Graph is plotted in between average waiting time (expressed in milliseconds) versus number of processes [Figure 9], [Figure 10] and [Figure 11]. Comparison shows the average waiting time of both SJF and our algorithm is very close [Figure 9]. But our algorithm presents no starvation problem. If we set higher value for constant 'r' more likely is the chance of older and bigger process to get CPU time, but on the cost of average waiting time. Large value of 'r' can be used where we didn't expect any job to wait for very long period of time say to provide some fairness to older and bigger process. For small value of constant 'r' we see that the result of average waiting time of the proposed algorithm is very close to that of the Shortest Job First (SJF) [Figure 9], [Figure 10] and [Figure 11]. If we set the value of constant 'r' very small say in decimal digits then the algorithm will reflect the result for average waiting time which is extremely close to that of the Shortest Job First (SJF). As the algorithm produces an extraordinary result in term of average waiting time, it is expected that it would be better than the present Highest Response Ratio Next (HRRN).

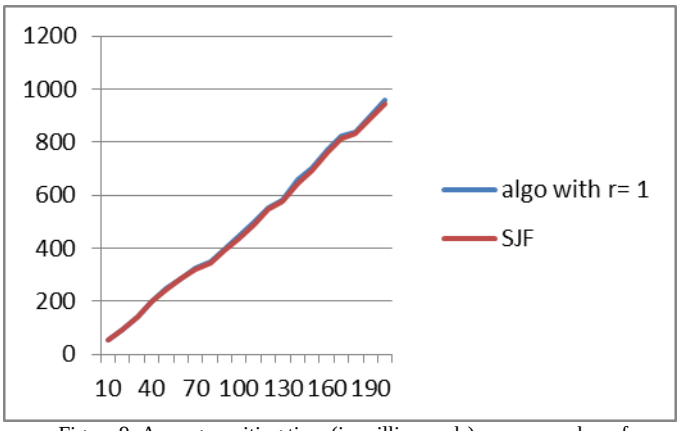

Figure 9. Average waiting time (in milliseconds) versus number of processes

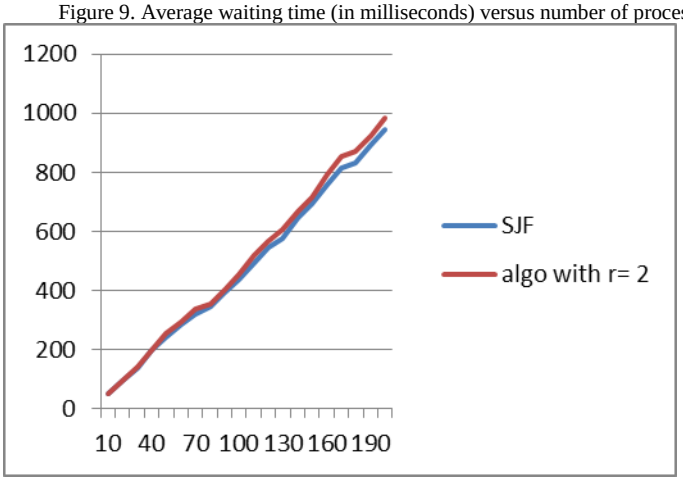

Figure 10. Average waiting time (in milliseconds) versus number of processes.

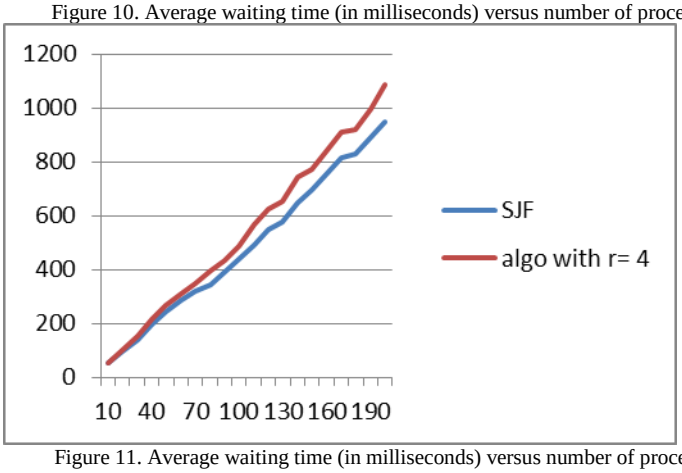

Figure 11. Average waiting time (in milliseconds) versus number of processes.

VI. CONCLUSION

This algorithm is adjustable as per different requirement by adjusting the value of constant 'r' but this should be done prior to implementation of this algorithm. Constant 'r' is the key factor of algorithm and how the algorithm is implemented i.e. our scheme which emphasizes over the number of process already executed since the arrival of certain process that is yet not allocated CPU time, this is due to large size of the process that it have been waiting but later after some time it will be allocated CPU time as per our proposed algorithm. Our algorithm gives better average waiting time and approximately similar result to that of the Shortest Job First (SJF). Our algorithm is more suitable for batch processing where burst time can easily be predicted or known in prior. It is also useful among users who happened to use the same share of daily programs. This algorithm is applicable in all real life scheduling situations while offering better and less average waiting time.

VII. FUTURE WORK

Above mentioned approach that is scheduling of process using number of processes already executed since its arrival can be used in other variety of algorithms. Future work for our proposed algorithm can include its implementation over multiple servers.

REFERENCES

[1] Schrage, L., Miller, L.: The queue M/G/I with the shortest remaining processing time discipline. Operation research 14(4), 670-684 (1966).
[2] Schrage, L.: A proof of optimality of the shortest remaining processi9ng time discipline. Operation research 16(3), 678-690 (1968).
[3] Saboori, E., Mohammadi, S., Parsazad, S.: A New Scheduling Algorithm for Server Farms Load Balancing. In: Industrial and Information Systems (IIS), 2nd International Conference. (2010).
[4] Jain, S., Mishra, V., Kumar, R. , Jaiswal, U.C.: Rail Road Strategy for SJF Starvation Problem: Communications in Computer and Information Science.(2011) Volume 142, Part 2, 403-408, DOI: 10.1007/978-3-642-19542-6_75
[5] Pinedo, M.: Scheduling: theory, algorithms and systems. Prentice-Hall, Englewood Cliffs (1995)
[6] Levi, S.T., Tripathi, S.K., Carson, S.D., Agrawala, A.K.: The Maruti Hard Real Time Operating System. ACM Special Interest Group on Operaying Systems 23 (3), 90-106 (1989)
[7] Crovella,M., Frangioso, R.: Connection scheduling in web servers. In: USENIX Symposium on Internet Technology and Systems (1999)
[8] Harchol-Balter, M., Schroeder, B., Agrawal, M., Bansal, N.: Size-based scheduling to improve web performance. ACM Transactions on Computer systems (TOCS) 21 (2), 207-233 (2003)
[9] Schroeder, B., Harchol-Balter, M.: web servers under overload: Hpw schedule can help ACM TOIT 6(1), 20-52 (2006)
[10] Hong, L., Ningyi, X., Zucheng, Z., Jihu, P.: N New HRRN in the Bus Arbitration of SoC Design. In: ASIC, Proceedings, 5th International Conference. (2003)
[11] Aburas, A.A., Miho,V.: Fuzzy Logic based Algorithm for Uniprocessor Scheduling. In: Proceedings of International Conference on Computer & Communication. (2008)

