COMPARITIVE ANALYSIS OF CPU SCHDULING ALGORITHMS

N.K SINGH
Department of computer science ,BIT Mesra, nksingh27@gmail.com

ABSTRACT

In multiprogramming systems i.e. in systems where several processes reside in memory, organization of processes is very important so that CPU always has one to execute. CPU scheduling is the base of multiprogramming operating systems. CPU executes one process at a time and switches between processes to improve CPU utilization. CPU scheduling strategies help in selecting the next process to be executed by the CPU.

CPU scheduling is one of the most important activities performed by operating system which helps in increasing the throughput of the computer system therefore if the performance of scheduling will improve then our computer system will become more productive.

In this paper, CPU scheduling algorithm with improved performance has been proposed. The technique which is used for increasing the speed up factor is 'Pipelining'. This technique can be applied to any CPU scheduling algorithm to improve its performance. The analysis shows that the proposed algorithm is better than the existing scheduling algorithms. The performance is improved by 40-50%.

General Terms

CPU Scheduler, Scheduling algorithm, Process, etc.

Keywords

Pipelining, Average waiting time, Burst time, Priorityscheduling, Round -Robin scheduling, etc.

DOI Number: 10.48047/nq.2015.13.4.874 **NeuroQuantology 2015; 13(4):542-545**

1. INTRODUCTION

Earlier we had systems in which only a single program was executed at a time and in those systems, the CPU was often idle i.e. CPU utilization was very low. Then came multiprogramming systems where several jobs were kept in memory. In multiprogramming environment, it becomes necessary for the CPU to perform scheduling so that it can select next process for execution whenever it is idle. Therefore we can say that CPU scheduling is the most important aspect of multiprogramming environment. CPU is switched among processes to make the computer more generative. In this paper, we will discuss various CPU scheduling algorithms and further pipelining is introduced to improve the performance of the scheduling algorithms. The main aim of multiprogramming is to keep the CPU busy all the time in order to maximize CPU utilization. In multiprogramming, a process is executed until it must waitdue to some reason like I/O request etc. In multiprogramming environment, CPU switches from one process to another but in case of uniprogramming environment CPU just waits and is idle. Switching is possible in multiprogramming environment as several processes are kept in memory and whenever one process has to wait; operating system takes the CPU away from that process and gives the CPU to another process according to the scheduling algorithm in use [12]. Pipelining can be applied to the "fetch, decode and execute cycle" of the processes to improve the performance.

2. ORGANIZATION OF THE PAPER

Section III describes various CPU scheduling algorithms with their Gantt charts. Section IV describes the proposed technique to improve the performance of the CPU scheduling algorithms. Section V comprises analysis of the proposed algorithm. Section VI proves the effectiveness of the proposed technique. Section VII contains the conclusion. Section VIII acknowledges the mentor for her constant guidance and section IX provides

the references.

3. SCHEDULING ALGORITHMS

First-Come, First-served Scheduling

This is the simplest scheduling algorithm. As the name suggests, the process which will come first will be executed first. But there are several problems associated with this like ifthe first process is very long then other shorter processes have to wait for longer time resulting in increase in the average waiting time. This problem is also known as convoy effect.

Table 1.FCFS Scheduling

Process	Burst time
P1	20
P2	3
P3	6

Table 2.Gantt chart for process Execution

P1	P2	P3
0 20	23	29

Average Waiting Time= (0+20+23)/3=14.33 ms

Shortest-Job-First Scheduling

In this scheduling algorithm, the process with the shortest burst time is executed first. Processes are executed in increasing order of their burst time. This decreases the average waiting time.

Example: Repeating the previous example using shortest job first scheduling:

Table 3. Gantt chart for process execution

P2	P3	P1
0 3	9	29

Average waiting time = 0+3+9/3=4 ms

We can clearly notice the drastic change in the average waiting time when compared to first-come first-served scheduling.

Priority Scheduling

In this scheduling strategy, processes are executed according to their priority. Priorities can be defined either internally or externally. Internally defined priorities use some measurable quantities like time limits, memory requirements etc. External priorities depend upon external factors like department sponsoring the work, amount of funds being paid for computer use etc. In real time systems, priority of any process can be set according to the deadline of that process because in real time systems, main aim is to meet the deadline.

Table 4.Priority Scheduling

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Table 5.Gantt chart for process execution

P2	P5	P1	P3	P4
0 1	6	16	18	19

Average Waiting Time = (0+1+6+16+18)/5=8.2 ms

Round-Robin Scheduling

This scheduling is similar to first-come first-served scheduling but preemption is also added to switch between processes. A small unit of time called a time quantum is defined and each process executes for that time quantum before switching to other process. This scheduling strategy is designed specially for time sharing systems.

Example: Repeating the previous example using round robin scheduling.

Time quantum=2 ms

Table 6.Gantt chart for process execution

P2	P5	P1	P3	P4	P5	P1	P5	P1	P1	P1
0 1	3	5	7	8	10	12	13	15	17	19

Average Waiting Time = (0+1+5+2+3+5+1+5+7)/5=5.8 ms

4. PROPOSED TECHNIQUE

Pipelining is a technique in which a process is divided into sub operations and each sub operation is executed in a special dedicated segment that operates concurrently with all other segments. Concurrent data processing helps in achieving faster execution [10].

CPU scheduling is one of the most important activities performed by operating system which helps in increasing the throughput of the computer system therefore if the performance of scheduling will improve then our computer system will become more productive. On combining pipelining with CPU scheduling, performance of CPU scheduling improves.

Proposed Approach

Pipelining concept can also be used in CPU scheduling to improve its performance. When CPU scheduler takes the decision of selecting the next process from the main memory, fetching and decoding of this next process takes some time and this time latency can be avoided by using pipelining.

Let us understand this with an example where we have three processes and we are using priority scheduling. Let us consider that process P1 has the highest priority, then process P2 and P3 has the least priority.

Without Pipelining

Table 7.CPU scheduling without pipelining

	1	2	3	4	5	6	7	8	9
1	P1	P1	P1						
2				P2	P2	P2			
3							P3	P3	P3

With Pipelining

Table 8.CPU scheduling with pipelining

	1	2	3	4	5
1	P1	P1	P1		
2		P2	P2	P2	
3			P3	P3	P3

Without pipelining, CPU scheduler would fetch P2 after completion of P1 in the 3rd step but with pipelining, P2 is fetched when P1 is decoded by the CPU. Similarly P3 is fetched and P2 is decoded when P1 is executed. Without pipelining, the whole process would take 9 clock cycles but with pipelining only 5 clock cycles are required.

5. ANALYSIS

Now, let us consider a *k* segment pipeline with a clock cycle time *Tp* used to execute *n* processes. The first process *P1* will require *(k*Tp)* time to complete its operation and the remaining *(n-1)* processes will egress at the rate of one process per clock cycle which is very clearly evident in Fig.1. Process *P1* is completing its execution in the 3rd clock cycle, process *P2* in the 4th clock cycle and further process *P3* in the 5th one. Therefore, to complete *n* processes, a *k*-segment pipeline requires *(k + (n-1))* clock cycles. A non-pipeline unit will take *(n*Tn)* time to complete *n* tasks where *Tn* is the time to complete each process.

Therefore, the speedup ratio of pipeline processing over an equivalent non-pipeline processing can be defined as:

S= ((n)(Tn)) /((k+n-1)* Tp)* [10]

Performance Evaluation:

Let us calculate the improvement in the performance by calculating the speed up ratio.

As *Tn* is the time to complete each process in non-pipeline unit and *(k*Tp)* is the time taken by process *P1* to complete its operation. So, let us consider that *Tn =(k*Tp)*.

Let us assume *Tp=30 ns.* Figure.1 shows that *k=3*. Therefore,

*Tn= (3*30) ns* and *(n*Tn)= (3*3*30) ns*.

So, non-pipeline system will take *270ns* to complete and pipeline system will take *((3+3-1)*30) ns i.e. 150ns.* Therefore speed-up ratio is:

S= (270/150) =1.8

Performance Improvement *= ((270-150)/270)*100 = 44.44%(1)*

6. EFFECTIVENESS OF THE PROPOSED TECHNIQUE

The proposed technique has been compared with the latest research done in the field of improving the performance of CPU scheduling algorithm to prove its effectiveness and efficiency.

The graph shown below in Figure.1 depicts the performance comparison of improved Round Robin (RR) scheduling algorithm with existing Round Robin (RR) scheduling algorithm.

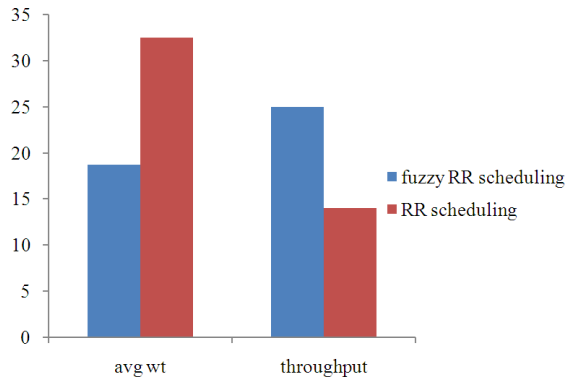

Figure 1: Performance comparison of improved RR scheduling algorithm with existing RR scheduling algorithm. [1]

Average waiting time for improved RR scheduling algorithm
= 19
Average waiting time for existing RR scheduling algorithm = 32.5
Performance Improvement = *((32.5-19)/32.5)*100=41.53%*
The performance improvement provided by our proposed technique is 44.44% (From (1)) which is greater than the performance improvement provided by the latest research done in this field.

7. CONCLUSION

It is concluded from the above analysis that the proposed technique improves the performance of the existing CPU scheduling algorithms by 40-50%. This technique can also be useful in many real time applications as concurrent processing always helps in faster execution

8. ACKNOWLEDGMENTS

We owe special debt of gratitude to Asst. Professor Parita Jain, Department of Computer Science & Engineering, KIET Engineering College, Ghaziabad, for her constant support and guidance throughout the course of our work. Her sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only her cognizant efforts that our endeavors have seen light of the day.

9. REFERENCES

[1] Bashir Alam, "Fuzzy Round Robin CPU Scheduling Algorithm", Journal of Computer Science, pp. 1079- 1085, 2013.

[2] Devendra Thakor, Apurva Shah, "D_EDF: An efficient Scheduling Algorithm for Real-Time Multiprocessor System", IEEE, pp. 1044-1049, 2011.

[3] Saeede Bibi, Farooque Azam, Yasir Chaudhry, "Combinatory CPU Scheduling Algorithm", 2010.

[4] Sindhu M, Rajkamal R, Vigneshwaran P, "An Optimum Multilevel CPU Scheduling Algorithm", International Conference on Advances in Computer Engineering, pp. 90-94, 2010

[5] Radhakrishna Naik, R.R. Manthalkar, Mukta Dhopeshwarkar, "Modified IUF Scheduling Algorithm for Real Time Systems", IEEE, pp. 712-716, 2010.

[6] Apurva Shah, Ketan Kotecha, "Adaptive Scheduling Algorithm for Real Time Multiprocessor Systems", IEEE, pp. 35-39, 2009.

[7] E.O. Oyetunji, A.E. Oluleye, "Performance Assessment of some CPU Scheduling Algorithms", 2009.

[8] Ruben Gran, Enric Morancho, Àngel Olive, Jose M. Llaberia, "An Enhancement for a Scheduling Logic Pipelined over two Cycles", IEEE, 2006.

[9] Shantanu Dutt, "Pipeline Basics", www.ece.uic.edu/~dutt/courses/ece366/lect14-pipe1.pdf

[10] M. Morris, Mano, "Pipeline and Vector Processing", Computer System Architecture, 3rd Edition, Dorling Kindersley (India) Pvt. Ltd., p.301-330.

[11] Toan Nguyen, "Pipelining", www.cs.sjsu.edu/~lee/cs147/Pipelining%20Toan.ppt

[12] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, "CPU Scheduling", Operating System Concepts, 6th Edition, John Wiley &Sons, p.151-183.