A Study of CPU Scheduling Techniques in Comparison

Pinaki Pratim Acharjya[1], Santanu Koley[2], Subhabrata Barman[3], Rajesh Mukherjee[4]
[1, 2, 3, 4]Department of Computer Science and Engineering, Haldia Institute of Technology, Haldia, India

ABSTRACT
Due to the requirement to alter and test operating system kernel code and assess the resulting performance on a consistent workload of real applications, developing CPU scheduling methods and understanding their impact in practice can be challenging and time consuming. Because the processor is such a valuable resource, CPU scheduling is critical for achieving the operating system (OS) design goals. The goal should be to keep as many processes active as possible at all times in order to get the most out of the CPU. The primary goal of CPU scheduling is to improve the computer's performance. The amount of time spent waiting has a significant impact on the performance and execution time of a computer system. It's much easier to grasp what's going on inside the system with this representation, and why a different collection of processes is a candidate for CPU allocation at different periods. The study's goal is to examine the highly efficient CPU scheduler's impact on the design of high-quality scheduling algorithms that meet the scheduling goals. It is mostly focused on reducing waiting times and turn-around times in order to improve a computer system's level of performance. Many CPU scheduling methods exist, however due to high context switch rates, long waiting times, long response times, long turnaround times, and low throughput; they cannot be applied in real-time operating systems.
Keywords - Average Waiting Time, Average Turned around Time, Response Time, Pipeline.

I. INTRODUCTION

CPU scheduling [1-3] is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc., thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Pipelining [10-11] 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.

II. TYPES OF CPU SCHEDULING

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state (for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state (for example, completion of I/O).
4. When a process **terminates**.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3. When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

a. Non-Preemptive Scheduling

Under non-pre-emptive [4-5] scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

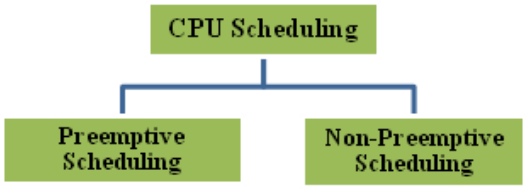


Fig.1 Types of CPU Scheduling.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware (for example: a timer) needed for preemptive scheduling.

b. Preemptive Scheduling

In this type of Scheduling [4-5], the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

III. SCHEDULING CRITERIA

There are many different criteria [6-9],[12] to check when considering the **"best"** scheduling algorithm, they are:

a. CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

b. Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

c. Turnaround Time

It is the amount of time taken to execute a particular process, i.e. the interval from time of submission of the process to the time of completion of the process (Wall clock time).

d. Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

e. Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

f. Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

III. SCHEDULING ALGORITHMS

Scheduling algorithms are useful in multiprogramming [15], to decide which process to execute first and which process to execute last to achieve maximum CPU utilization [14], computer scientists have defined some algorithms, and they are:

1. First Come First Serve (FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling

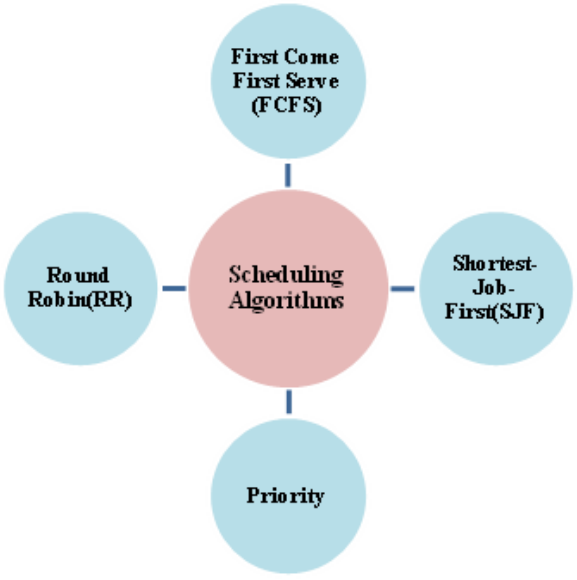


Fig.2 List of Scheduling Algorithms.

i. First Come First Serve Scheduling

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

First Come First Serve is just like FIFO (First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.

This is used in Batch Systems. It's easy to understand and implement programmatically, using a Queue data structure, where a new process enters through the tail of the queue, and the scheduler selects process from the head of the queue. A perfect real life example of FCFS scheduling is buying tickets at ticket counter.

a. Calculating Average Waiting Time

For every scheduling algorithm, **Average waiting time** is a crucial parameter to judge its performance. Average Waiting Time (AWT) is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution. Lower the Average Waiting Time, better the scheduling algorithm. Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** 0,and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.

Process	Burst Time
P1	21
P2	3
P3	6
P4	2

Table 1. List of processes and burst time for FCFS.

The average waiting time will be = (0 + 21 + 24 + 30)/4=18.75.

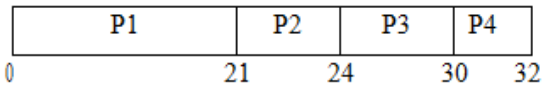


Table 2. GANTT Chart for FCFS.

This is the GANTT chart for the above process. The average waiting time will be 18.75 ms. For the above given processes, first **P1** will be provided with the CPU resources. Hence, waiting time for **P1** will be 0. **P1** requires21 msfor completion, hence waiting timefor **P2** will be 21 ms. Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be (21 + 3) ms = 24 ms. For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

b. Problems with FCFS Scheduling

- It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter. If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.

- Not optimal Average Waiting Time.
- Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource (CPU, I/O etc.) utilization.

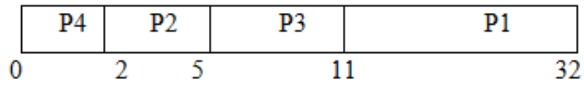
ii. Shortest Job First (SJF) Scheduling

- Shortest Job First scheduling [13] works on the process with the shortest **burst time** or **duration** first. This is the best approach to minimize waiting time. This is used in Batch Systems. It is of two types: Non Pre-emptive and Pre-emptive To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time. This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (Either Arrival time is 0 for all, or Arrival time is same for all).
- Non Pre-emptive Shortest Job First
- Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.

Process	Burst Time
P1	21
P2	3
P3	6
P4	2

Table 3. List of processes and burst time for SJF.

In Short Job First Scheduling, the shortest Process is executed first.
Hence, the GANTT chart will be following:



As you can see in the Gnatt chart above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.
We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

a. Problem with Non Pre-emptive SJF

- If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.
- This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

iii. Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

Process	Burst Time	Arrival Time
P1	21	0
P2	3	1
P3	6	2
P4	2	3

Table 5. List of processes, burst time and arrival time for preemptive SJF.

The GANTT Chart for Preemptive Shortest Job First Scheduling will be

P1	P2	P4	P2	P3	P1
0	1	3	5	6	12	32

Table 6. GANTT Chart for preemptive SJF.

The Average Waiting Time will be, ((5 - 3) + (6 - 2) + (12 – 1))/4=4.25. The average waiting Time for Preemptive Shortest Job first Scheduling is less than both, non-preemptive SJF Scheduling and FCFS Scheduling. As it can be seen in the **GANTT chart** above, as **P1** arrives first, hence it's execution starts immediately, but just after 1ms, process**P2**arrives with a **burst time** of 3 ms which is lessthan the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preempted and process **P2** is executed.

As **P2** is getting executed, after 1 ms, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of 2 ms, as a result **P2** (2 ms done, 1 ms left) is preempted and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**. The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

iv. Priority Scheduling

Priority is assigned for each process. Process with highest priority is executed first and so on. Processes with same priority are executed in FCFS manner.

Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Burst Time	Priority
P1	21	2
P2	3	1
P3	6	4
P4	2	3

Table 7. List of processes, burst time and priority for priority scheduling.

The GANTT Chart for the following Processes Based on Priority Scheduling Will be:

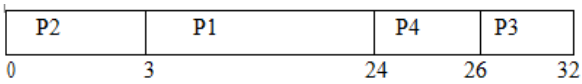


Table 8. GANTT chart for Priority Scheduling.

The Average Waiting time will be, (0 + 3 + 24 + 26 +)/4=13.25.

v. Round Robin Scheduling

A fixed time is allotted to each process, called **quantum**, for execution. Once a process is executed for given time period that process is preemptied and other process executes for given time period. Context switching is used to save states of preempted processes.

Process	Burst Time	Quantum
P1	21	5
P2	3	
P3	6	
P4	2	

Table 9. List of processes, burst time and priority for RR scheduling.

The GANTT Chart for Round Robin Scheduling will be

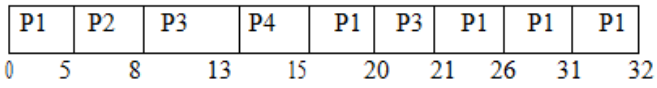


Table 10. GANTT chart for RR Scheduling.

IV. PROPOSED TECHNIQUE

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.

i. Without Pipelining

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

Table 9. CPU scheduling without pipelining.

i. With Pipelining

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

Table 10. GANTT chart for Pipelining.

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. The analysis [16] can reveal some new ideas to deliver.

V. RESULT 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 5 th 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%.

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.

Average waiting time for improved RR scheduling algorithm = 19

Average waiting time for existing RR scheduling algorithm = 32.5

Performance Improvement = ((32.5-9)/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.

VI. CONCLUSION

According to the results of the preceding investigation, the proposed technique enhances the performance of existing CPU scheduling algorithms by 40-50%. This technique can be used in a variety of real-time applications because concurrent processing always speeds up execution. Any form of simulation for any CPU scheduling strategy is recommended to have restricted accuracy. The only way to evaluate a scheduling algorithm is to code it and include

it in an operating system; only then can the algorithm's correct working capabilities be determined in real-time systems.

REFERENCES

[1] Bashir Alam (2013), "Fuzzy Round Robin CPU Scheduling Algorithm", Journal of Computer Science, pp. 1079- 1085.
[2] Devendra Thakor, Apurva Shah (2011), "D_EDF: An efficient Scheduling Algorithm for Real-Time Multiprocessor System", IEEE, pp. 1044-1049.
[3] Saeede Bibi, Farooque Azam, Yasir Chaudhry (2010), "Combinatory CPU Scheduling Algorithm".
[4] Sindhu M, Rajkamal R, Vigneshwaran P (2010), "An Optimum Multilevel CPU Scheduling Algorithm", International Conference on Advances in Computer Engineering, pp. 90-94.
[5] Radhakrishna Naik, R.R. Manthalkar, Mukta Dhopeshwarkar (2010), "Modified IUF Scheduling Algorithm for Real Time Systems", IEEE, pp. 712-716.
[6] Apurva Shah, Ketan Kotecha (2009), "Adaptive Scheduling Algorithm for Real Time Multiprocessor Systems", IEEE, pp. 35-39.
[7] E.O. Oyetunji, A.E. Oluleye (2009), "Performance Assessment of some CPU Scheduling Algorithms".
[8] Ruben Gran, Enric Morancho, Àngel Olive, Jose M. Llaberia (2006), "An Enhancement for a Scheduling Logic Pipelined over two Cycles", IEEE.
[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., pp.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.
[13] Lupetti, Simone, Dmitrii Zagorodnov (2006), "Data Popularity and Shortest-Job-First Scheduling of Network Transfers", International Conference on Digital Telecommunications, IEEE, pp. 26–26.
[14] Lingyun Yang, Schopf Jennifer M, Ian Foster T. (2003), "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments", Proceedings of the ACM/IEEE SC2003 Conference (SC) 1-58113-695-1/03 $ 17.00 © 2003 ACM.
[15] Liu CL, Layland James W. (1973), "Scheduling Algorithm for Multiprogramming in a Hard-Real-Time Environment", J ACM. Jan 1973; 20(1): pp. 46–61.
[16] M.R. Reddy, V.V. Ganesh, S. LakshmiandY. Sireesha (2019), "Comparative Analysis of CPU Scheduling Algorithms and Their Optimal

Solutions,"Proceedings of the Third International Conference on Computing Methodologies and Communication (ICCMC 2019) IEEE Xplore Part Number: CFP19K25-ART, pp. 255-260.