An Analysis for Task Optimization based on CPU Scheduling with Pipelining Techniques

Sindhu Daniel
Assistant Professor
Department of Master of Computer Application
Mount Zion College of Engineering, Kadammanitta, Kerala, India

Abstract— one of the most important components of the computer resource is the CPU. In multiprogramming systems i.e. the 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 the basis of multi programmed operating systems. Most CPU scheduling algorithms concentrate on maximizing CPU utilization, throughput and minimizing turnaround time, waiting time, response time and number of context switching for a set of requests. Some of the popular CPU scheduling algorithms are First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling and Round Robin (RR). This paper describes the CPU scheduling algorithm with pipelining techniques. These techniques can be applied to any CPU scheduling algorithms to improve its performance. The analysis shows that the proposed system is better than the existing scheduling algorithms.

Keywords: Pipelining, Average waiting time, Burst time, Priority scheduling, Round -Robin scheduling

I. INTRODUCTION

In multiprogramming systems, multiple processes are being kept in memory for maximum utilization of CPU [1]. CPU utilization can be maximized by switching CPU among waiting processes in the memory and running some process all the time [2]. Which process should be selected next for service, is an important question, because it affects the effectiveness of the service. The main aim of the CPU scheduling algorithms is to maximizing CPU utilization and throughput and minimizing turnaround time, waiting time, response time and number of context switching for a set of requests.

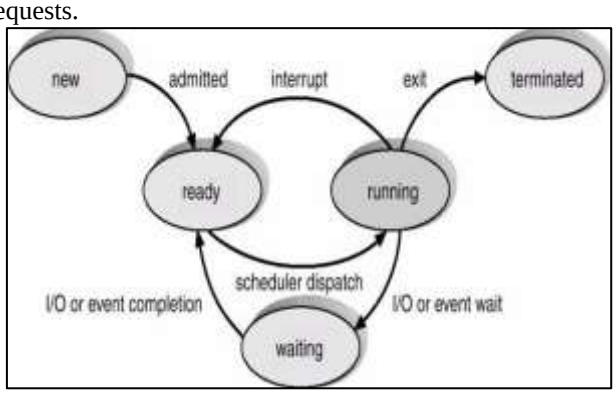

Fig. 1: processes state transition diagram

The main objective of multiprogramming system is to load many processes in the main memory where they reside in the ready queues making link lists. A process is simply program on execution. A Process is an instance of a computer program that is being executed. It contains the program code and its current activity. CPU Scheduling is the basis of multiprogrammed operating system. By switching the CPU among processes, the operating system can make the computer more productive.

II. SCHEDULING CRITERIA

The scheduler must consider the following parameters and optimization criteria have in order to maximize the performance of the system. Following mentioned are some of the criteria's that help to judge the performance.

A. CPU utilization

It keeps the CPU as busy as possible. It must have maximum value.

B. Throughput

The number of processes that complete their execution per unit of time. It must have maximum value.

C. Turnaround time

It is the amount of time to execute a particular process. It must have minimum value.

D. Waiting time

It is the amount of time a process has been waiting in the ready queue or in the waiting state. It must have minimum value.

E. Response time

It is defined as intervening time between the time the user initiates a request and the system starts responding to this request.

III. SCHEDULING ALGORITHMS

The scheduling algorithms are divided into two categories as Pre-emptive and Non-pre-emptive scheduling algorithms. In non-pre-emptive, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or switching to the waiting state. In pre-emptive, CPU allocated to a process is switched if another process of higher priority is scheduled. In this case, the currently running process is interrupted and moved to the ready state by the operating system.
Following are the various scheduling algorithms:

A. 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 if the 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.

For example, Consider three processes p1, p2, and p3 with their arrival times and required CPU burst as shown in the following table

Process	Burst time	Arrival Time
P1	7	0
P2	5	1
P3	2	3

Table 1: FCFS Scheduling

P1	P2	P3

0 7 12 14

Table 2: Gantt chart for process Execution

Average Waiting Time= (0+6+9)/3=5 ms

B. *Shortest-Job-First Scheduling*

SJF is non-preemptive 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. Thus a longer process has to wait until all the processes shorter than it have been executed. In case two processes have the same CPU burst, they are scheduled in the FCFS order.

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

Process	Burst time	Arrival Time
P1	7	0
P2	5	1
P3	2	3

Table 3: SFS Scheduling

P1	P3	P2

0 7 9 14

Table 4: Gantt chart for process Execution

Average Waiting Time= (0+4+8)/3=4 ms

C. *Priority Scheduling*

In priority scheduling algorithm, a priority is associated with each process and the CPU is allocated to the process with the highest priority. The processes which have equal priority; they are scheduled with FCFS policy.

Priority scheduling can be either pre-emptive or non-preemptive. When the process arrives at the ready queue, its priority is compared with the priority of the currently running process. If the priority of the newly arrived process is higher, the non- pre-emptive scheduling algorithm will put the newly arrived process at the head of the ready queue. While the pre-emptive scheduling algorithm will interrupt the currently running process and move it to the ready queue and then CPU is allocated to the newly arrived process

	Burst time	Arrival Time	Priority
P1	7	0	3
P2	5	1	1
P3	2	3	2

Table 5: Priority Scheduling

P1	P2	P3

0 7 12 14

Table 6: Gantt chart for process Execution

Average Waiting Time= (0+6+9)/3=5 ms

D. *Round-Robin (RR) Scheduling*

Round robin scheduling is a pre-emptive version of first-come, first-served scheduling. Processes are dispatched in a first-in-first-out sequence but each process is allowed to run for only a limited amount of time. This time interval is known as a time-slice or time quantum. In this, the ready queue is treated as the circular queue.

One of the two things will happen in Round-Robin. First, the process may have burst-time less than or equal to time quantum. In this case, the process will execute and after completion release the CPU by itself. Second, the process may have burst time greater than time quantum. In this case, the process will execute for 1 time quantum and then it is pre-empted. Then context-switch will be executed and CPU scheduler will select the next process to execute. The pre-empted process will be put at the tail of the ready-queue. This continues till the execution of all the processes is completed.

Process	Burst time	Arrival Time
P1	7	0
P2	5	1
P3	2	3

Table 7: Round Robin Scheduling

Time slice: 4

P1	P2	P3	P1	P2

0 4 8 10 13 14

Table 4.Gantt chart for process Execution

Average Waiting Time= (6+9+6)/3=7 ms

E. *Multilevel Queue Scheduling*

The processes can be classified into different groups depending upon their situation. For example, the common division between processes is foreground processes and background processes. Foreground processes are interactive processes and background processes are batch processes. These types of processes have different response-time requirements and scheduling needs. Also, foreground processes have priority over background processes.

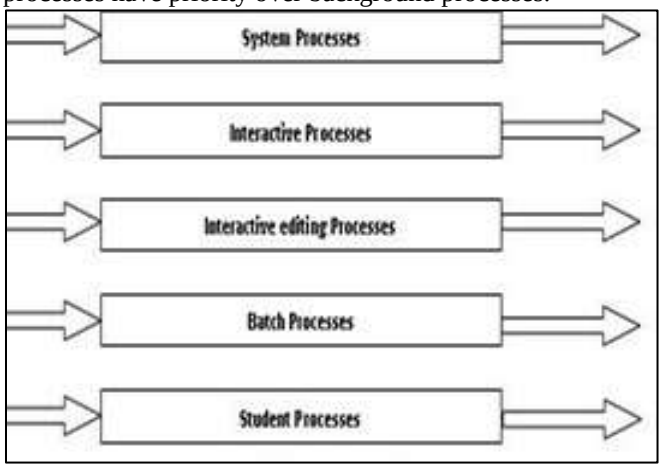

Fig. 2: Multilevel Queue Scheduling

In this, the ready queue is partitioned into several separate queues. And each queue has its own scheduling algorithm. In our example of foreground and background processes, foreground queue uses Round-Robin Scheduling and background queue uses First-Come-First-Serve scheduling. Also, there must be scheduling among the queues which are implemented on the basis of:

1) Fixed-priority pre-emptive scheduling:

In this, each queue has absolute priority over lower priority queue.

Let's take an example of five queues listed below in the order of priority as shown in fig 6 above:

1) System processes
2) Interactive processes
3) Interactive editing processes
4) Batch processes
5) Student Processes

In this, no process in the lower priority queues could run unless all the higher priority queues were empty. So, in our example, if the interactive editing process entered in the ready queue while a batch process was running, then the batch process would be pre-empted. In this, there is a possibility of starvation.

2) Time-slice among the queues:

Here, each queue gets a certain portion of CPU time, which it can then schedule among its various processes. By considering our example, the foreground queue can be given 80% of the CPU time for R-R scheduling among its processes, whereas the background queue receives 20% of the CPU time for FCFS scheduling among its processes.

IV. 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.

A. 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.

Pipelining refers to the temporal overlapping of processing pipelines. It is more than assembly lines in computing that can be employed for instruction processing. A basic pipeline process a sequence of tasks or instruction, according to the following principle of operation.

Each task is subdivided into a number of consecutive tasks. The processing of each single instruction can be broken down into four sub tasks:-

a) Instruction Fetch
b) Instruction Decode
c) Operand Fetch
d) Execution

It is supposed that there is a pipelined stage associated with each subtask. The equal amount of time is available in each stage for performing the required subtask.

All the pipeline stages work like an assembly line that is, receiving their input from the previous stage and delivering their output to next stage. We also suppose, the basic pipeline operates clocked, in other words synchronously. This means that each stage accepts a non-input at start of clock cycle, each stage has a single clock cycle available for performing the required operation and each stage increases the result to the next stage by the beginning of subsequent clock cycle.

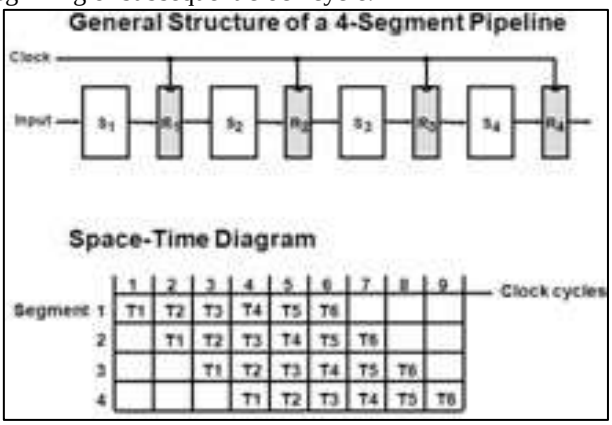

Fig. 3: Basic structure of a linear pipeline processor and space time diagram.

Consider we have 3 processes and we are using priority scheduling. Let us consider that process P1 has the highest priority, then process P3 has the least priority.

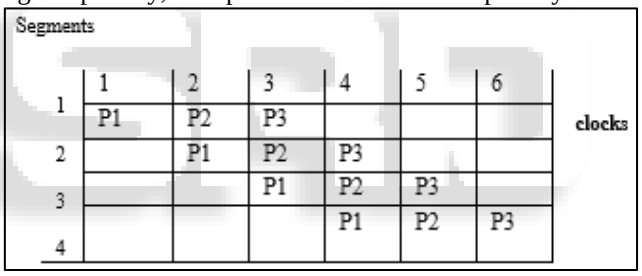

Fig. 4: CPU scheduling with pipelining

1	2	3	4	5	6	7	8	9	10	11	12
P1				P2				P3			
	P1				P2				P3		
		P1				P2				P3	
			P1				P2				P3

Fig. 5: CPU scheduling without pipelining

Without pipelining, CPU scheduler would fetch P2 after completion of P1 in the 4th 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 and so on. Without pipelining, the whole process would take 12 clock cycles but with pipelining only 6 clock cycles are required.

V. ANALYSIS

Now, consider, in Fig 4. Process P1 is completing its execution in the 4th clock cycle, process P2 in the 5th clock cycle and further process P3 in the 6th one. Therefore, to complete n processes, a k-segment pipeline requires (k + (n-1)) clock cycles, where k cycles are used to fill up the pipeline or to complete execution of the first task. A non-pipeline unit will take (n*k) time to complete n tasks.

The speedup ratio of pipeline processing over an equivalent non- pipeline processing can be defined as:

$$sk= n*k /k+(n-1)$$

The maximum speed up is sk ->k,for n >> k.The maximum speed up that a linear pipeline can provide is k,where k is the no. of stages in the pipe.

A. Efficiency:

The efficiency of a linear pipeline is measured by the percentage of busy time-space spans over the total time-space span, which equals the sum of all busy and idle time-space spans. Let *n*, *k*,τ be the number of tasks (instructions), the number of pipeline stages, and the clock period of a linear pipeline, respectively. The pipeline efficiency is defined by

$$\eta = \frac{n \cdot k \cdot \tau}{k \cdot [k\tau + (n-1)\tau]} = \frac{n}{k+(n-1)}$$

B. Throughput:

The number of results (tasks) that can be completed by a pipeline per unit time is called its throughput. This rate reflects the computing power of a pipeline. In terms of efficiency η and clock period τ of a linear pipeline, we define the throughput as follows

$$w = \frac{n}{k\tau + (n-1)\tau} = \frac{\eta}{\tau}$$

The time required to execute *n instructions* without pipeline is

$$T1 = nk\tau$$

Because to execute one instruction it will take nt cycle.
The speed up factor for the instruction pipeline compared to execution without the pipeline is defined as:

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k+(n-1)]\tau} = \frac{nk}{k+(n-1)} = \frac{nk}{(k-1)+n}$$

Performance Evaluation:
Let us calculate the improvement in the performance by calculating the speed up ratio,
Let us assume Tk=50 ns. Fig2 shows that k=4. Therefore, Tn= (4*50) ns and (n*Tn)= (3*4*50) ns.
So, non-pipeline system will take 600ns to complete and pipeline system will take ((4+3-1)*30) ns i.e. 104ns. Therefore speed-up ratio is:

S= (600/104) =5.7

Performance Improvement = ((600-104)/600)*100 = 82.66%

VI. Conclusion

It is concluded from the above analysis that the proposed Technique improves the performance of the existing CPU scheduling algorithms by 80-85%.In a proposed technique applying pipelining method that enhance the cpu efficiency and speed up. This technique can also be useful in many real time applications as concurrent processing always helps in faster execution.

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.

