International Journal of Computer Applications (0975 – 8887)
Volume 99– No.18, August 2014

Improvised Round Robin (CPU) Scheduling Algorithm

Abhishek Sirohi
Department of CSE
Galgotias College of Engineering & Technology

Aseem Pratap
Department of CSE
Galgotias College of Engineering & Technology

Mayank Aggarwal
Department of CSE
Galgotias College of Engineering & Technology

ABSTRACT

CPU is a primary computer resource. So, its scheduling is central to operating system design. To improve both utilization and the speed of CPU we need to keep several processes in memory at a time that means we use the sharing and multiprogramming concepts. CPU scheduling determines which process run when there are multiple runnable processes CPU scheduling is necessary because it has a big effect on resources utilization and overall performance of the system. In this paper, we are giving an improved CPU Scheduling algorithm.

Keywords

Scheduling, Time Quantum

1. INTRODUCTION

Scheduling is one of the basic functions of any operating system, because scheduling of all the computer resources is done before their use. The CPU the most essential computer resource. Therefore its scheduling algorithm is a very important part of the OS design. When multiple processes are runnable, the OS has the onus of responsibility to decide which one is to run first. The part of the OS that takes this decision is called scheduler and the algorithm it works on is called scheduling algorithm. A CPU scheduler is a part of an operating system and is responsible for mediating access to the CPU. OS may have up to three types of schedulers: a long term scheduler (also known as an admission scheduler or high level scheduler), a medium-term scheduler and a short-term scheduler (also known as a dispatcher or CPU scheduler).

1.1 Long-term Scheduler

The long-term scheduler decides which jobs or processes are to be proceeded to the ready queue i.e. when an attempt is made to execute a process its inclusion to the set of currently running processes is either granted or delayed by the long-term scheduler. Therefore this scheduler governs which processes are to run on a system, and the degree of concurrency that is supported at any one time.

1.2 Medium-term Scheduler

The medium-term scheduler temporarily clears the processes from main memory and places them on secondary memory or contrarily. This is known as the "swapping of processes out" or "swapping in".

1.3 Short-term Scheduler

The short-term scheduler (also known as the CPU scheduler) decides which of processes that are present in the ready queue, in the memory are to be executed (allocated a CPU) next following a clock interrupt, an Input-Output (IO) interrupt and an OS call (system call) or any other form of signal. Therefore the short-term scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers. This scheduler can be preemptive, meaning that it can forcibly remove processes from a CPU i.e. it can allocate the CPU (allocated to current process) to another process, or non-preemptive (also known as "voluntary" or "co-operative"), in that case the scheduler is unable to force processes off the CPU.

The success of a CPU scheduler depends highly on the design of good quality scheduling algorithm. Good-quality CPU scheduling algorithms depends mainly on criteria such as response time, throughput, CPU utilization rate, waiting time, turnaround time and. Thus, the main focus of this proposed work is to develop a generalized optimum good quality scheduling algorithm suited for all types of jobs.

Fig. 1. Shows the following states have been executed in the CPU Scheduler.

1. When a process switches from the running state to the waiting state.

2. When a process switches from the running state to the ready state.

3. When a process switches from the waiting state to the ready state.

4. When a process terminates.

The success of a CPU scheduler depends highly on the design of good quality scheduling algorithm. Good-quality CPU scheduling algorithms depends mainly on criteria such as response time, throughput, CPU utilization rate, waiting time, turnaround time and. Thus, the main focus of this proposed work is to develop a generalized optimum good quality scheduling algorithm suited for all types of jobs.

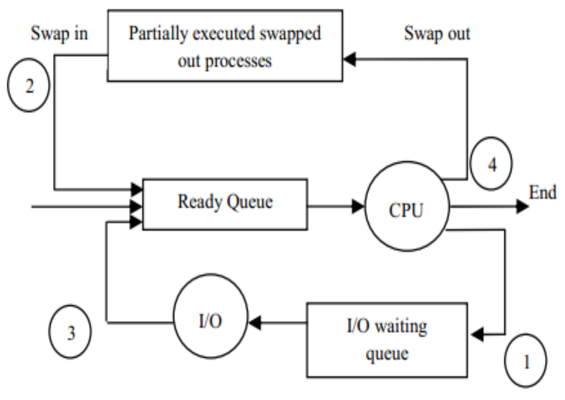


Fig. 1. Process of Schedulers

2. SCHEDULING A LGORITHMS

2.1 First Come First Serve

The simplest technique is to let the first process submitted to run first. This technique is called as first-come, first-served

(FCFS) scheduling. In this technique, the processes are inserted into the end of a queue when they are submitted [2]. The next process is taken from the starting of the queue when each process finishes running.

i. Algorithm

Step 1: The process whose request comes first is allocated to the CPU first.
Step 2: The addition of new processes takes place at the tail of the ready queue.
Step 3: When the termination of the process takes place, the next process is dequeued from the head of the ready queue and run it.

ii. Characteristics

- There is no prioritization this means that every process can eventually be complete, hence no starvation.
- High turnaround time, waiting time and response time.
- The Process with the longest burst time can monopolize CPU, even if the burst time of other process is too short. Hence, throughput is low [3].

2.2 Shortest Job First

The process is allocated to the CPU which has least burst time. The scheduler arranges the processes with least burst time in the starting of the queue and processes with longest burst time in the end of the queue. This algorithm requires advanced estimations about the time required for a process to complete [2]. The design of this algorithm is to give maximum throughput in most scenarios.

i. Algorithm

```
Step 1: CPU is allocated to the process having the shortest burst time.
Step 2: If one or more than one process have equal burst time.
{
The CPU is allocated to the process according to the FCFS scheduling
}
```

II. Characteristics

- The problem with the SJF algorithm is, to have the knowledge of the length of the next CPU request.
- SJF reduces the average waiting time because it services small processes before it services large ones. Although it reduce the average wait time, it may affect the processes with high burst time requests. If the ready list is full, then processes with large burst times tend to be left in the ready list while small processes receive service. In extreme cases, when the system is in very little idle state, processes with large burst time will never be served. This case of starvation of large processes is a serious problem of this algorithm.

2.3 Round Robin

The Round Robin scheduling algorithm assigns a small unit of time, called a time slice or time quantum to the processes. A queue holds all the ready processes. The scheduler goes around this queue, allocating the CPU to each and every process for the defined time quantum. The new processes are added to the end of the queue.

i. Algorithm

```
Step 1: Time quantum is selected and then it is assigned for each process.
Step 2: The CPU is allocated to the process according to the First Come First Serve (FCFS) scheduling.
Step 3: If (burst time of the process < time quantum).
{
The process is allocated the CPU till it terminates.
}
Else
{
The CPU is occupied by the process till the time quantum is over and it is added to the end of the ready queue for the next round of execution.
}
```

II. Characteristics

- If the time quantum is set too short then it may cause many context switches and it would result in lower CPU efficiency.
- If the time quantum is set too long then it may cause poor response time and approximates FCFS.
- As this algorithm results in high waiting times, the deadlines are seldom met in a pure RR system.

3. PROPOSED SCHEDULING ALGORITHM

```
Step 1: START
Step 2: Calculate the time quantum as follows.
        Time quantum = $\Sigma P_i/n$
Where $P_i$ is the burst time of Process i, n is the number of process.
Step 3: Arrange the processes in ascending order in the ready queue such that the head of the ready queue contains the lowest burst time process.
Step 4: Repeat steps 4, 5, and 6 WHILE ready queue becomes empty.
Step 5: Allocate CPU to the first process in ready queue for one time quantum.
Step 6: If the remaining burst time of currently running process is less than time quantum then
        {
        Allocate CPU again to the currently running process for remaining burst time and after completion, go to step 3.
        }
        Else
        {
                Remove the currently running process from the ready queue and put it at the tail of the ready queue.
        }
Step 7: END
```

3.1 Characteristics

- The starvation of processes with long burst times can be avoided by giving a time quantum for each.
- No process can monopolize CPU.
- The waiting time, Turnaround time can be optimized.

3.2 Computation of Gantt Chart, Waiting Time and Turnaround Time

Consider the following data to check the efficiency of the proposed algorithm

Table 1

Process ID	Burst Time
P1	20
P2	34
P3	5
P4	12

P5	26

Table 2

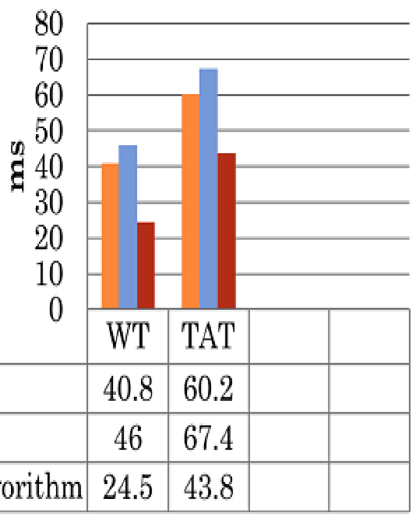
Process Id	Burst Time
P1	10
P2	1
P3	2
P4	1
P5	5

This is the comparison of various CPU scheduling algorithms on the basis of Waiting Time (WT) and Turn around Time (TAT). The data for plotting Chart 1 and Chart 2 has been taken from the Table 1 & Table 2. The Time quantum has been calculated by the formula

Time quantum =$\Sigma P_i/n$

The waiting time and Turn around Time have been calculated by

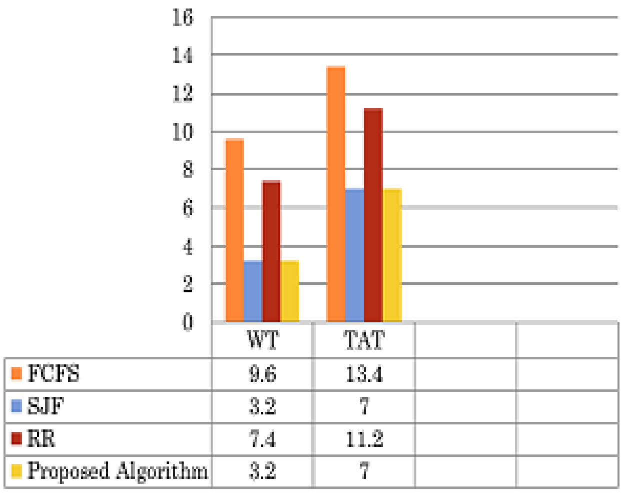
Waiting Time= Start Time-Arrival Time
Turn Around Time= Finish Time-Arrival Time



	WT	TAT		
🟧 FCFS	40.8	60.2		
🟦 RR	46	67.4		
🟥 Proposed Algorithm	24.5	43.8		

Chart 1: Comparison of Proposed algorithm with other standard algorithms on basis of Waiting Time (WT) and Turn around Time (TAT)

The data for plotting chart 1 has been taken from the Table 2.



	WT	TAT		
🟧 FCFS	9.6	13.4		
🟦 SJF	3.2	7		
🟥 RR	7.4	11.2		
🟨 Proposed Algorithm	3.2	7		

Chart 2: Comparison of Proposed algorithm with other standard algorithms on basis of Waiting Time (WT) and Turn around Time (TAT).

4. RESULTS AND DISCUSSION

The proposed algorithm has been coded and simulated with C code; it can be an innovative move in case of CPU scheduling and can be easily implemented in the OS.

Comparison of the Standard algorithms with the proposed algorithms is shown in **chart 1** and **chart 2**.

It can be clearly observed that the waiting time and turn around time of the processes are better for the proposed algorithm as compared to all other standard algorithms discussed in section 2.

5. CONCLUSIONS AND FUTURE WORK

The proposed algorithm produced better waiting time and turnaround time then many standard algorithms. The proposed algorithm works by comparing the time quantum with the burst time of the process. This technique is the basis of the good results produced by this algorithm. This algorithm can be bettered in future and be made to produce even better results for a wide range of CPU Scheduling algorithms.

6. ACKNOWLEDGEMENT

We feel highly elated to render our heartfelt thanks to our mentor and Head of Department Mrs. Bhawna Mallick, who stimulated and motivated us constantly to take up this great task of working on this topic. We thank our project coordinator Mr. Sandeep Saxena to having belief in our abilities and for providing valuable help in our project.

7. REFERENCES

[1] http://en.wikipedia.org/wiki/Scheduling_(computing)

[2] Sindhu M, Rajkamal R, Vigneshwaran P. An Optimum Multilevel CPU Scheduling Algorithm. 2010 International Conference on Advances in Computer Engineering

[3] Wei Zhao, John A. Stankovic. Performance Analysis of FCFS and Improved FCFS Scheduling Algorithms for Dynamic Real-Time Computer Systems. IEEE 1989.

[4] Davender Babbar, Phillip Krueger. A Performance Comparison of Processor Allocation and Job

Scheduling Algorithms for Mesh-Connected Multiprocessors. IEEE 1994.

[5] Umar Saleem and Muhammad Younus Javed. Simulation Of CPU Scheduling Algorithms. IEEE 2000.

[6] Snehal Kamalapur, Neeta Deshpande. Efficient CPU Scheduling: A Genetic Algorithm based Approach. IEEE 2006.

[7] Nikolaos D. Doulamis, Anastasios D. Doulamis, Emmanouel A. Varvarigos, and Theodora A. Varvarigou. Fair Scheduling Algorithms in Grids. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 18, NO. 11, NOVEMBER 2007.

[8] Xiao-jing Zhu, Hong-bo Zeng, Kun Huang, Ge Zhang. Round-robin based scheduling algorithms for FIFO IQ switch. IEEE 2008.

[9] Apurva Shah, Ketan Kotecha. Efficient Scheduling Algorithms for Real-Time Distributed Systems. 2010 1st International Conference on Parallel, Distributed and Grid Computing

[10] Devendra Thakor, Apurva Shah. D_EDF: An efficient Scheduling Algorithm for Real-Time Multiprocessor System. IEEE 2011.

[11] Tong Li, Dan Baumberger, Scott Hahn. Efficient and Scalable Multiprocessor Fair Scheduling Using Distributed Weighted Round-Robin. ACM 2009.