ODSA: A Novel Ordering Divisional Scheduling Algorithm for Modern Operating Systems

Junaid Haseeb
Department of Computer Science
University of Management and Technology, Sialkot, Pakistan

Khizar Hameed
Department of Computer Science
University of Management and Technology, Sialkot, Pakistan

Muhammad Junaid
Department of Computer Science
University of Management and Technology, Sialkot, Pakistan

Muhammad Tayyab
Department of Computer Science
University of Management and Technology, Sialkot, Pakistan

Samia Rehman
Department of Computer Science
COMSATS Institute of Information Technology, Islamabad, Pakistan

Agha Muhammad Musa Khan
Department of Computer Science
University of Management and Technology, Sialkot, Pakistan

***Abstract*—CPU scheduling is defined as scheduling multiple processes that are required to be executed in a specific time period. A large number of scheduling algorithms have been proposed to achieve maximum CPU utilization/throughput and minimizing turn around, waiting and response time. Existing studies claim that Round Robin (RR) is providing best results in terms of above-mentioned factors. In RR, a process is assigned to CPU for a fixed time quantum then the process starts its execution, in case that assigned time quantum greater than CPU's capacity then remaining section of that process waits for its next turn. Although RR schedules processes in an efficient manner, however, it has certain limitations such as if time quantum is too small or large, it causes frequent context switching and response time can increase. To address these identified problems, various improved versions of RR also exist. The purpose of this paper is twofold: 1) a comparison between different improved versions of RR; and 2) a new algorithm named Ordering Divisional Scheduling Algorithm (ODSA) is also proposed that combines various features of different algorithms and is actually an improvement to RR. Our results show that ODSA can schedule processes with less turn around and average waiting time as compared to existing solutions.**

***Keywords*—CPU scheduling; round robin scheduling algorithm; turnaround time; waiting time; context switching**

I. Introduction

Operating System (OS) is an essential part of a computer system that acts as an intermediary between input commands and hardware. Among various functions performed by OS, one is processed scheduling, as Central Processing Unit (CPU) has to manage concurrently executing processes. Some processes are concerned with OS and others are originated by users. For the execution, each process requires specific time duration of CPU. Required execution time is totally dependent on the type of process to be executed; it may fall into the category of engaging CPU's resources for a long time or either short. In the context where multiple processes are available in the ready state against only one CPU than OS has to decide which process needs to be executed first. For this purpose, many scheduling algorithms have been proposed and this management of ordering processes is known as process scheduling [1]. These proposed algorithms have been designed with various goals such as better utilization of CPU's resources, less turnaround time, waiting and response time of processes.

CPU plays a vital role in the execution of processes as it has to assign required resources by OS to a specific process. In the case of multiple processes to be executed, scheduling of processes requires a careful and must ensure fairness so that process starvation is minimized [2]. Scheduling process can be performed using software like scheduler or dispatcher [3]. Round Robin (RR) is most commonly known algorithm that helps in scheduling processes for OS [1], [4], [5]. In RR, CPU splits OS's time into multiple slices that are known as time quantum. Then these time intervals are assigned to processes so that their execution over OS can be performed. RR scheduler is mainly concerned with following dimensions.

- **CPU Utilization** – By keeping the CPU as busy as possible.
- **Throughput** – Number of tasks completed in unit time.
- **Turnaround** – Time required completing a job after the submission.
- **Waiting Time** – Time required waiting in a ready queue.
- **Response Time** – Time required to response a particular job after the submission.
- **Fairness** – Time is given by CPU to each thread.

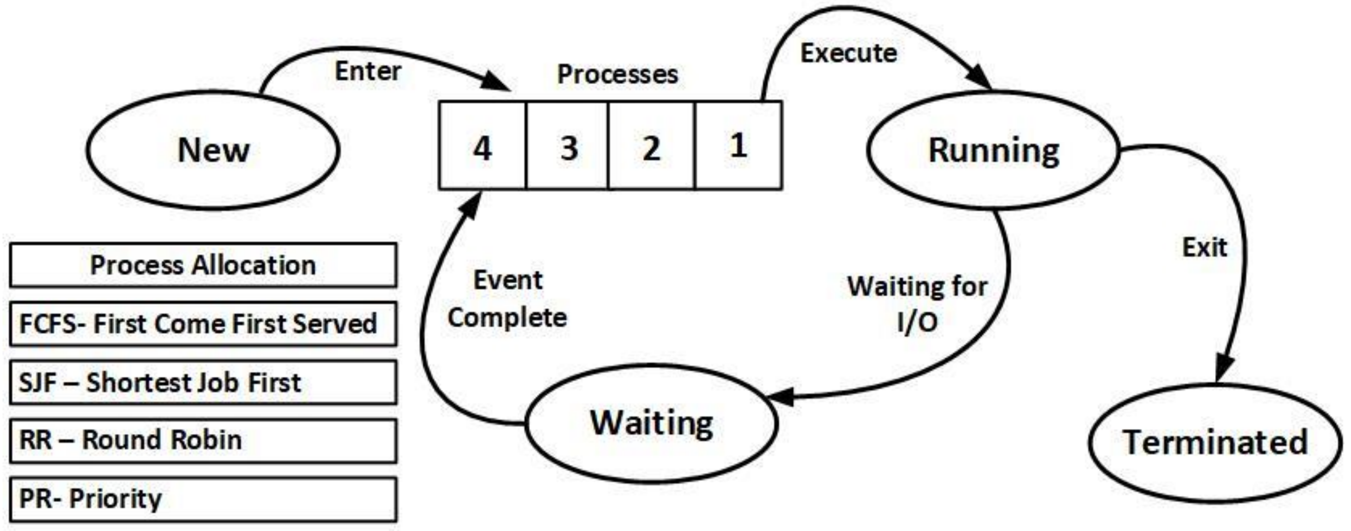


Fig. 1. Process creation and allocation in OS.

Another scheduling algorithm is First-come, first served (FCFS) [4]. In FCFS scheduling algorithm, the process request the CPU and CPU in return execute the process in same order. A single queue is maintained for ready processes in this algorithm. It's a non-preemptive scheduling algorithm i.e. once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O. The next algorithm is SJF [5]. In Shortest Job First (SJF), advanced knowledge of time taken by the processes is required. The process having less time is executed by CPU prior to that having a large amount of time. Another is priority based algorithm [6] . In priority based algorithm, every process is assigned a fixed priority by OS and the scheduler arranges the processes in the ready queue in order of their priority. Fig. 1 briefly explains the process creation and allocation procedure for all the above-mentioned algorithms.

Although these scheduling algorithms consider performance parameters very well but still have definite problems. In FCFS, when one process is completed than CPU switches to another process, therefore, scheduling overhead is minimal and no reorganization of the process queue is required [6], [7]. The FCFS algorithm has low throughput as the process executes in the same order as they come and there is a possibility that long processes hold the CPU for a long duration. Resultantly turnaround time, waiting time and response time can be high [7], [8]. In SJF there is an additional context switching if a shorter process arrives during another process execution. This halts the currently running process, execute the shortest job and then resume the previous one. This creates additional overhead. The algorithm is giving maximum throughput [4], [9]. The major flaw is starvation which occurs when there are a large number of processes are being run by CPU. In priority based algorithm, overhead is not minimal. Waiting and response time is interlinked with priority. Higher priority processes have smaller waiting and response times. In [5] the starvation problem also exists. In RR scheduling algorithm, if time quantum is too small it gives extensive overhead. In addition, average response time, waiting time is dependent on a number of processes, its length and value of time quantum. Starvation has been reduced almost to zero [2], [10].

As mentioned earlier that RR is considered as one the most commonly used algorithm. In order to overcome its concerned problems, various improvements have been made in RR algorithm [7]. However, which algorithm is giving maximum result is still questionable. In this paper, we provide an overview of improvements made up till now in RR using simulations. We also propose new algorithm Ordering Divisional Scheduling Algorithm **(**ODSA) to achieve maximum performance in terms of scheduling criteria.

Rest of the paper is structured as follows. Section II provides a detailed discussion about existing similar works done in this field like PBDRR [11], Time quantum using fuzzy logic [10], an improved RR algorithm [12] and dynamic quantum [5], [13]. Section III presents the analysis & experiment discussion along with results. Section IV contains the detail of proposed algorithm. In section V, comparison results are discussed with rest of the algorithms. Section VI provides conclusion and future implications.

II. Related Work

Round Robin has become one of the most important and widely used scheduling algorithms, despite its problems such as fixed quantum size [2], [14]. As Round Robin (RR) is used almost in all like in Windows, UNIX, BSD etc., So, to overcome its shortcomings, many types of research have been carried out [2], [12], [13], [15].

Mohanty et al. [3] proposed priority based Dynamic Round Robin algorithm to combine the dynamic time quantum and priority based selection of processes. In this way, time slice for each process is calculated and it changes after every round of execution. In [7], an algorithm was designed which take input sequence and assign priority, and then it sets the value of original time slice (OTS). The components of priority are calculated using short components [8]. In first round for processes having SC as 1, assign time quantum same as intelligent time slice whereas the processes having SC as 0 given the time quantum equal to the ceiling of the half of the intelligent time slice. In next round, the processes having SC as 1 assign double the time slice of its previous round whereas the processes with SC equal to 0 given the time quantum equal to the sum of the previous time quantum and ceiling of the half of the previous time quantum. After various examples, it reflected that as time quantum is dynamic therefore reducing no of context switching, average waiting and response time [3].

Alam et al., [1] used the concept of fuzzy logic to determine time quantum. Fuzzy logic is basically an extension of Boolean logic dealing with the concept of partial truth that denotes the degree of truth. In real everything, it can be expressed in binary terms. He used fuzzy interface system for finding the time quantum [16]. The fuzzy inference system accepts two numbers as input and produce only single number as output. The input numbers specify as the total number of processes resides in the ready queue and average burst time of processes [17]. Time quantum is the fixed output value generated by that system. The advantage of fuzzy logic is that each process in the system assigned a fixed time quantum according to average burst time. In addition, the performance of the system is not declined due to gratuitously context switches[1].

Mohanty et al., [18] proposed new algorithm as a combination of the shortest job first (SJF) and RR. In the first process, it is assigned to CPU using Round Robin scheduling algorithm. In a second step, it selects the shortest job from the waiting queue and it shortest job assign to the CPU [9]-[14], [19]. The process will be terminated after the successful execution of all the processes. Another improvement is the combination of SJF and RR algorithms [1]. This algorithm runs as normal RR in the first cycle and then selects the SJF from waiting for the queue and so on. From a number of experiments present in this paper, it is obvious that total waiting time and average turnaround time both are reduced [16], [17], [20] . The reduction of total waiting time and turnaround time shows maximum CPU utilization and minimum response time [18].

Noon et. al [2] proposed the new concept of time quantum which is based on dynamic allocation of time to processes. The given time quantum allocated to process on their burst time. To solve the problem of time quantum, AN algorithm is proposed that adjusts the time quantum of processes which resides in ready queue. Experimental evaluations claimed that the efficiency of AN algorithm is higher than the existing RR algorithm[10], [16]. In context of efficiency, the dynamic scheduling algorithm is reliable and scalable for wide variety of OS's as it provides the support of self- adaptation OS. The self-adaptation property automatically fills the requirement of end user [17].

Above discussion can be summarized as many improvements to RR have been already proposed. One question that is important yet not answered is which algorithm provides better results in terms of CPU performance. To answer this question, we have performed simulations of various algorithms. Further discussion about experimental setup, selected algorithms and criterion for declaring best algorithm is part of section III and results of our simulation are discussed in key findings section.

TABLE I. PROCESSES ALONG WITH BURST TIME AND PRIORITY

PROCESS ID	**BURST TIME (ms)**
P0	12
P1	49
P2	20
P3	60
P4	30
P5	9

TABLE II. PROCESSES ALONG WITH BURST TIME AND PRIORITY

PROCESS ID	**BURST TIME (ms)**
P0	**10**
P1	**2**
P2	**1**
P3	**15**

III. EXPERIMENTAL SETUP AND ANALYSIS

In this section of the paper, different improved versions of RR are compared to answer the question of the best scheduling algorithm. Three most common known algorithms (2, 3, and 4) are selected and best performance criterion is based on turnaround and average waiting time. Turnaround time of a process is defined as the time a process has to wait for getting its turn so that its remaining execution can be completed and the waiting time of a process is the time it has to wait in queue before going into execution mode. To perform the simulations, we have taken two examples. Details about processes included in examples along with burst time are presented in Tables 1 and 2 and fixed time quantum respectively and simulation results according to already selected algorithms are as follows.

We have tested three algorithms on the above two examples. Each algorithm has the output in term of average Turnaround time and the average waiting time. In these examples, the overall waiting time for a process is too high in such away the process has to wait for a log until its execution time. To overcome these problems, we have proposed a new algorithm named a novel: Ordering Divisional Scheduling Algorithm (ODSA), in which we have achieved very low throughput time as well as very low waiting time for a process to be executed. In the following section IV, detailed information ODSA is provided.

IV. ORDERING DIVISIONAL SCHEDULING ALGORITHM (ODSA)

A new algorithm Ordering Divisional Scheduling Algorithm (ODSA) is designed providing better efficiency as compared to all discussed algorithms and also overcoming the problems mentioned in introduction part. As the name depicts it first arrange the processes in ascending order based on their burst time and then divide the processes into two halves. Complete steps involved in this algorithm are as follows:

- First, arrange the processes as per ascending order of burst time.
- Then divide the processes into two halves (In case of odd values use ceiling in first half).
- First, run the process from second half having shortest burst time.
- Then run all the first half processes using SJF algorithm.
- At the end run all the remaining processes of second half using RR algorithm.

After dividing when we run the process from second half then using two different algorithms i.e. SJF and RR this will reduce the processes waiting time and response time hence improving the efficiency.

```
Algorithm 1 ODSA
procedure ODSA
Input: Processes,
initialization $Processes = \phi$
ODSA( )
    Foreach (Processes)
            $processes'$ = Order(processes)      ▷ Ascending order of Burst
                                                 ▷ time of each processes
            Split($processes'$) $Half_A \wedge Half_B$    ▷ Splits the processes
                                                 into two halves
    Foreach ($Half_B$)
                ExecutedProcesses [] = B.processesExecute ($Burst_{Half_B}$)
                                                 ▷ As per shorted Burst time
    Foreach ($Half_A$)
                ExecutedProcesses [] = A.processesExecute ($SJF_{Half_A}$)
                                                 ▷ As per SJF
    If (A.processes $\notin$ ExecutedProcesses [] && B.processesExecute $\notin$ executed[])
            ExecuteProcesses [] = Execute($RR_{Half_{A\&\&B}}$)    ▷ As per RR
remaining processes
end procedure
```

A. Verification of ODSA

To validate our proposed algorithm, we have again used example 1 and 2. This time these examples are scheduled according to newly proposed ODSA algorithm. Results are presented as follows:

1) Step – I & II:

Arrange the processes as per ascending order of burst time and divide into two halves as presented in Table 3.

TABLE III. PROCESSES AFTER ORDERING AND DIVIDING

PROCESS ID	BURST TIME (ms)
P5	9
P0	12
P2	20
P4	30
P1	49
P3	60

2) Step – III, IV & V:

- First, run the process from second half having shortest burst time i.e. P4.
- Then run all the first half processes using SJF algorithm.
- At the end run the remaining process of second half using RR algorithm.

Results of example 3 after scheduling processes according to ODSA are as following:

3) Step – I & II

- Arrange the processes as per ascending order of burst time and divide into two halves as presented in Table 4.

TABLE IV. PROCESSES AFTER ORDERING AND DIVIDING

PROCESS ID	BURST TIME(ms)
P2	1
P1	2
P0	10
P3	15

4) Step – III, IV & V

- First, run the process from second half having shortest burst time i.e. P0.
- Then run all the first half processes using SJF algorithm.
- At the end run the remaining process of second half using RR algorithm.

V. RESULTS – COMPARISON

Results obtained from simulations are compared with existing algorithms are available in Tables 5 and 6.

It reveals that newly designed algorithm has less turnaround as well as waiting time as compared to all the improvements made in RR algorithm up till now. The number of context switches is also less hence ODSA improves the efficiency and throughput.

TABLE V. COMPARISONS OF ODSA WITH EXISTING ALGORITHM USING EXAMPLE 3

Scheduling Criteria	An Improved RR algorithm	AN algorithm	PBDRR	ODSA
Turnaround time	99.33	119.33	106.16	85.1
Waiting time	69.33	89.33	76.16	55.1
Context Switching	11	10	11	8

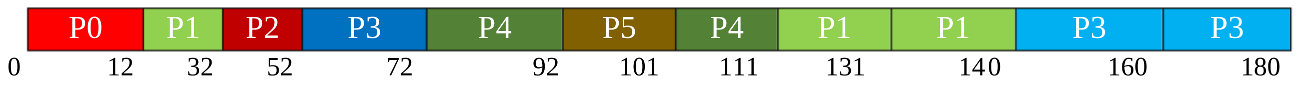
TABLE VI. COMPARISONS OF ODSA WITH EXISTING ALGORITHM USING EXAMPLE 4

Scheduling Criteria	An Improved RR algorithm	AN algorithm	PBDRR	ODSA
Turnaround time	17.5	17.25	23	15.5
Waiting time	10.5	10.25	18	8.5
Context Switching	6	6	6	5

ODSA is showing less turnaround / waiting time and also less number of context switching. Graphical comparison of ODSA with all discussed algorithm is shown in Fig. 2.

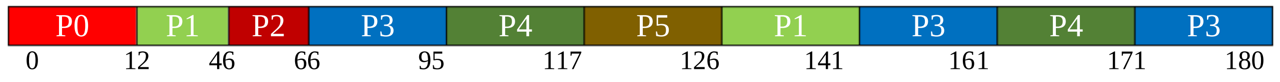
Example 1

An improved RR algorithm [4]



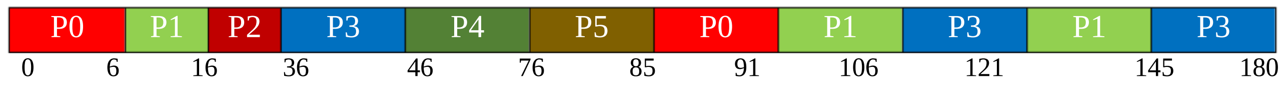
Fixed Time Quantum = 20, Average Turnaround Time = 99.33ms, Average Waiting Time = 69.33ms

A New Round Robin Algorithm [2]



Fixed Time Quantum = 20, Average Turnaround Time = 119.33ms, Average Waiting Time = 89.33ms

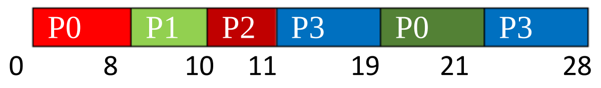
PBDRR algorithm [3]



Fixed Time Quantum = 20, Average Turnaround Time = 106.16ms, Average Waiting Time = 76.16ms

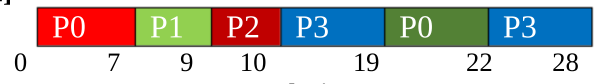
Example 2:

An improved RR algorithm [4]



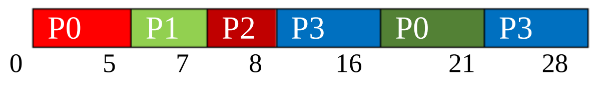
Fixed Time Quantum = 8, Average Turnaround Time = 17.5ms, Average Waiting Time = 10.5ms

A New Round Robin algorithm [2]



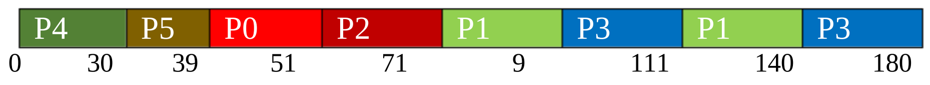
Fixed Time Quantum = 8, Average Turnaround Time = 17.25ms, Average Waiting Time = 10.25ms

PBDRR algorithm [3]



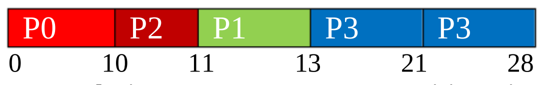
Fixed Time Quantum = 8, Average Turnaround Time = 23ms, Average Waiting Time = 18ms

Example 3:



Average Turnaround Time = 85.1ms, Average Waiting Time = 55.1ms

Example 4:



Average Turnaround Time = 15.5ms, Average Waiting Time = 8.5ms

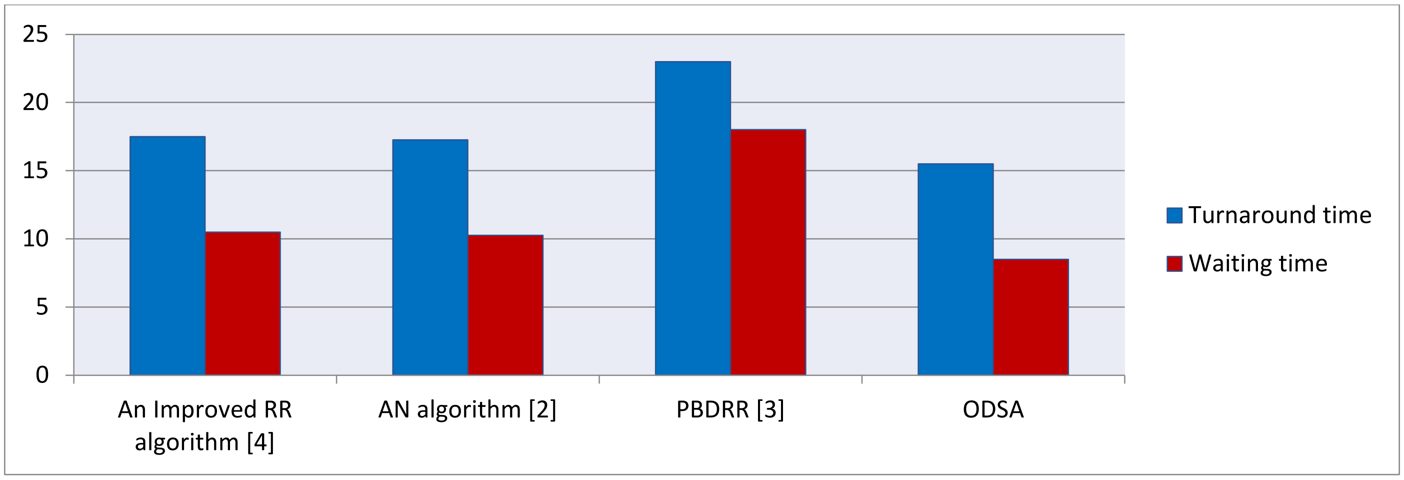


Fig. 2. Comparison of proposed and existing algorithms.

VI. Conclusion

Process scheduling is important in a multiprogramming environment. Various algorithms have been designed so far for better process management. Different researchers have made improvements in RR algorithm to overcome the problems like starvation, context switching etc. ODSA has been proposed in this paper which fairly allocates the resources to CPU. This algorithm is improving the CPU scheduling criteria and processes turnaround time is very less. It is also improved the efficiency in terms of response time and no of context switches. Deficiencies in existing algorithms like context switching, starvation, convoy effect etc. have also overcome by ODSA. Therefore, we can say efficiency of proposed scheme in terms of CPU scheduling criteria is better than all the existing improvements made till now in RR algorithm. In future, we are interested to enhance ODSA by considering more performance parameters. Comparing ODSA with other scheduling algorithms in terms of energy utilization and management of hardware resources are future implications.

References

[1] B. Alam, M. N. Doja, and R. Biswas, "Finding Time Quantum of Round Robin CPU Scheduling Algorithm Using Fuzzy Logic," IEEE International Conference on Computer and Electrical Engineering (ICCEE), pp. 795–798, 2008.

[2] A. Noon, A. Kalakech, and S. Kadry, "A New Round Robin Based Scheduling Algorithm for Operating Systems : Dynamic Quantum Using the Mean Average," IJCSI International Journal of Computer Science Issues, vol. 8, no. 3, pp. 224–229, 2011.

[3] P. R. Mohanty, P. H. S. Behera, K. Patwari, M. Dash, and M. L. Prasanna, "Priority Based Dynamic Round Robin (PBDRR) Algorithm with Intelligent Time Slice for Soft Real Time Systems," vol. 2, no. 2, pp. 46–50, 2011.

[4] W. Stallings, "Operating Systems: Internals and Design Principles," Pearson, vol. 8, 2014.

[5] L.-X. Wang, "A Course in Fuzzy Systems and Control," Prentice-Hall, Inc., 1997.

[6] A. S. Tanenbaum, A. S., & Woodhull, "Operating systems: design and implementation," Englewood Cliffs, NJ Prentice-Hall, vol. 2, 2006.

[7] S. Mamdani, E. H., & Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller,"International Journal of Man-Machine Studies,vol. 7, no. 1, pp. 1–13, 1975.

[8] M. Sugeno, "Industrial applications of fuzzy control.," Elsevier Sci. Inc., 1985.

[9] J. R. Jang, "ANFIS : Adaptive Network Based Fuzzy Inference System," IEEE transactions on systems, man, and cybernetics, vol. 23, no. 3, 1993.

[10] D. J. Simon, "Training fuzzy systems with the extended Kalman filter," Fuzzy sets and systems, vol. 132, no. 2, pp. 189–199, 2002.

[11] W. Tong and J. Zhao, "Quantum Varying Deficit Round Robin Scheduling Over Priority Queues," IEEE International Conference on Computational Intelligence and Security, pp. 252–256, 2007.

[12] R. J. Matarneh, "Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes," American Journal of Applied Sciences, vol. 6, no. 10, pp. 1831–1837, 2009.

[13] T. Helmy, "Burst Round Robin as a Proportional-Share Scheduling Algorithm." IEEE International Conference on Grid and Cooperative Computing, 2007

[14] S. M. Mostafa, S. Z. Rida, and S. H. Hamad, "Finding time quantum of round robin CPU scheduling algorithm in general computing systems using integer programming." International Journal of Research and Reviews in Applied Sciences (IJRRAS), vol. 5, no. 1, pp. 64–71, 2010.

[15] Liu, C. L., & Layland, J. W "Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment Scheduling Algorithms for Multiprogramming," Journal of the ACM (JACM), vol.20, no. 1, pp. 46–61, 1973.

[16] H. S. Behera, "A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis," International Journal of Computer Applications, vol. 5, no. 5, pp. 10–15, 2010.

[17] Yang, L., Schopf, J. M., & Foster, I, "Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments" In Proceedings of the 2003 ACM/IEEE conference on Supercomputing, pp. 1–16, 2003.

[18] R. K. Yadav, A. K. Mishra, N. Prakash, and H. Sharma, "An Improved Round Robin Scheduling Algorithm for CPU scheduling ," International Journal on Computer Science and Engineering vol. 2, no. 4, pp. 1064–1066, 2010.

[19] S. Suranauwarat, "A CPU Scheduling Algorithm Simulator," IEEE International Conference on Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports,pp. 19–24, 2007.

[20] P. R. Mohanty, P. H. S. Behera, K. Patwari, M. R. Das, and M. Dash, "Design and Performance Evaluation of a New Proposed Shortest Remaining Burst Round Robin (SRBRR) Scheduling Algorithm," In Proceedings of International Symposium on Computer Engineering & Technology (ISCET), Vol. 17, pp. 126-137.