# Comparison of CPU Scheduling Algorithms: FCFS, SJF, SRTF, Round Robin, Priority Based, and Multilevel Queuing

Sajeewa Pemasinghe
*Department of Food Science and Technology*
*Wayamba University of Sri Lanka*
Makandura, 81070, Sri Lanka
sajeewa@wyb.ac.lk

Samantha Rajapaksha
*Department of Information Technology*
*Sri Lanka Institute of Information Technology*
Malabe, 10115, Sri Lanka
samantha.r@sliit.lk

2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC) | 978-1-6654-0156-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/R10-HTC54060.2022.9929533

***Abstract*—In this article, we are discussing various aspects of CPU scheduling. We first introduce the concept of CPU scheduling, different types of schedulers and the typical terminology used in relation to processes. Scheduling criteria, the optimization of which is the ultimate goal of a CPU scheduling algorithm, are also discussed. We then discuss various types of research studies that have been carried out with respect to CPU scheduling algorithms. Different CPU scheduling algorithms are examined with examples to highlight their characteristics. Advantages and disadvantages of each of these algorithms are also explored. The scheduling algorithms discussed are, first come first served, shortest job first, shortest remaining time first, priority based, round robin, multilevel queue, and multilevel feedback queue.**

***Index Terms*—CPU scheduling algorithms, FCFS, SJF, SRTF, Round robin, Priority based, Multilevel queuing**

## I. Introduction

When a user working at a computer, requests a program, it first checks whether it can be found in the random access memory (RAM). If it is not found in the RAM, it has to be fetched from permanent storage such as the hard disk, where a copy of the requested program will be created and brought to the RAM. At any given time, out of all the requested programs, which ones will populate the RAM, will be decided by a scheduler program known as the long-term scheduler. Out of all the programs that reside in the RAM (these processes that are waiting to be selected to access the CPU are said to be in a 'ready queue'), at a given time, which one will be given access to the CPU is determined by the scheduler program known as the short-term scheduler (STS). As the RAM gets full, and a high priority process has been created in the disk drive, some low priority process in the RAM has to be swapped out of the RAM so that the high priority process in the disk drive can be swapped into the RAM. This swapping in and swapping out of processes into and out of the RAM is done by the scheduler program known as the medium-term scheduler. As its task is to select which program has access to the CPU next, the short-term scheduler is what is referred to as the CPU scheduler.

The operating system (OS) uses a data structure known as a process control block (PCB) to store information about each process. This information includes parameters such as, process id, program counter (stores from where the execution has to be restored when continuing from a previous temporary stop), process state (i.e. new, ready, waiting, running, terminated etc.), priority, list of open files and devices, stack space (to keep track of the function calls), heap space (to provide dynamically allocated memory), and information about static and global variables. All this information about a given a process is known as the 'context' of the process. When the STS, preempts (temporarily interrupts a running process) a process P1 and switches to a process P2, the context of P1 has to be saved, in case it has to be restarted from where it was stopped. Therefore when the CPU switches from P1 to P2 there is a corresponding 'context switching' (i.e. moving from context of P1 to context of P2).

In older batch operating systems, the degree of multi-programming was one. That, is, the CPU can run only one process at a time in a non-preemptive fashion. Therefore, at a given time, other processes had to wait until the current process terminated, in order to access the CPU. Typically when a process is being executed by the CPU, there is a CPU-I/O (input/output) cycle. The CPU executes the instructions in the process for some time (this time period is called the 'CPU burst') and then it waits for some until some I/O action is complete (this time period is called the 'I/O burst'). These CPU bursts and I/O bursts keep happening cyclically until the process is terminated (and hence, is called a CPU-I/O cycle). During the I/O burst, the CPU remains idle which is a waste of CPU time. In today's multi-programming systems, during the above idle time some other process can utilize the CPU which increases the CPU utilization and therefore, the efficiency. Today's systems are not only multi-programming,

they are also multi-processing, which means there is more than one processor working in parallel. In these systems, the same logic is extended across all the processors to increase CPU utilization and decrease the CPU idle time. The task of an ideal STS is to select processes from the ready queue in such a way so that it maximizes CPU utilization [1].

### A. Scheduling Criteria

STSs can use a single scheduling algorithm or a mixture of algorithms. Although maximization of CPU utilization is indeed a characteristic of a winning algorithm, when comparing the overall effectiveness of many scheduling algorithms, many criteria are used:

- CPU Utilization: maximize the time the CPU is kept busy and minimize the CPU idle time
- Throughput: higher the number of processes finished per unit time, higher the throughput
- Turnaround time: the time duration between the arrival of a process in RAM and the completion of its execution (this includes waiting time in the ready queue, total CPU execution time and the total I/O action time). The goal is to minimize this time duration.
- Waiting time: This is the total amount of time a process waits in the ready queue without being executed by the CPU or doing I/O action, between its arrival at the RAM and its completion.
- Response time: the time duration between the arrival of a process in RAM and the time it gets first access to the CPU.

## II. Review of the Literature

There is a large body of research that mainly focus on various modifications to scheduling algorithms in order to optimize the scheduling criteria. First come first served (FCFS) is considered as the simplest CPU scheduling algorithm where the processes are given CPU access according to the order of their arrival times. It has shown to produce lower computational overheads [2]. But it is suffering from many issues such as longer waiting times, lower throughput, inefficient resource utilization [3]. An improvement to the original FCFS algorithm (FCFSI) which increases the chances of a new task being scheduled in the ready queue has been described in great detail by Zhao *et al* [4].

Parekh *et al* [5] suggest a combination of priority, shortest job first (SJF) and round robin (RR) algorithms to increase throughput while at the same time decreasing waiting time and turnaround time. As will be further discussed later, one of the drawbacks of multilevel queue (MLQ) algorithm is the lack of flexibility due to the inability of processes to shift between different ready queues. To circumvent this, multilevel feedback queue (MLFQ) algorithm provides processes in different queues, leeway to get promoted to a queue with a higher priority or get demoted to a queue with a lower priority. Thombare *et al* [6] suggest an improvement to the multilevel feedback queue (MLFQ) scheduling algorithm. They suggest a three-level queue where the first queue is using a time quantum of 10 ms. If a given process exceeds the time quantum, it is sent to the second queue, wherein, initially, the processes are arranged to access the CPU using the SJF algorithm and then treated by the RR algorithm with a dynamic time quantum. If the time quantum is exceeded, the processes are sent to the third queue, wherein again, initially the processes are arranged to access the CPU using the SJF algorithm and then treated by the RR algorithm with a dynamic time quantum. They show that this mechanism brings down the average waiting time and the average turnaround time compared to the same setup with a static time quantum. A similar approach for decreasing the turnaround and waiting times by applying SJF prior to the application of RR algorithm was adopted by Yadav *et al* [7]. In terms of finding the optimum time quantum for RR, a neural network based approach was proposed by Maste *et al* [8]. They build a knowledge base by running different programs with different static time quanta and time slices and store the time quanta that give minimum turnaround times. When a new program comes, if a similar program is already found in the database, it's time quantum is predicted using the knowledge base. Otherwise, the program is run with different time quanta and the knowledge base is updated according to the produced turnaround times. They show that this allocation of dynamic time slice for time quantum leads to improved performance when compared to the version with a static time slice.

Many implementations of the RR algorithm use the CPU burst times of a previous set of processes and calculate the time quantum of the current process as a measure of central tendency of that previous set of processes. This can be a measure like the arithmetic mean [9]–[11] or the median [12] of the previous set of processes. This use of a measure of central tendency assumes that the time quanta of the selected processes are evenly distributed. This is not the case always. There can be processes lined up in the ready queue that have asymmetrically distributed CPU burst times. Omotehinwa *et al* [13] have proposed a method to calculate the time quantum of RR when the burst times are asymmetrically distributed. This method calculates the interquartile ranges (IQR) for previous process burst times and checks for values outside the IQR to detect the presence of outliers and then uses geometric mean to calculate the time quantum.They have shown that their version of the RR algorithm shows decreased average turnaround times and average waiting times when compared to other variants of the RR algorithm such as NIRR [14], IRRVQ [15], and DABRR [16].

As will be explained later, one of the drawbacks in implementing the SJF algorithm is, not having an exact method of calculating the CPU burst time of a process that is in the ready queue. Many researchers have come up with different mechanisms for estimating the CPU burst time. A machine learning-based method for approximating the CPU burst time

in SJF was proposed by Helmy *et al* [17]. They have a selected a set of processes with their corresponding process attributes. As with many other supervised machine learning (ML) methods, they have defined a feature vector, and a target variable. The feature vector consists of a filtered list of process attributes. The set of processes are then divided to a training set and a test set. They have used a variety of ML methods including methods such as decision trees and K nearest neighbors to generate models using the training data set. Each of the generated models was evaluated against the test set using metrics such as correlation coefficient. The best generated models were used to predict CPU burst times for new processes and these predicted burst times were subsequently used for implementing the SJF algorithm. The overall process followed by them is graphically summarized in Fig. 1 below.

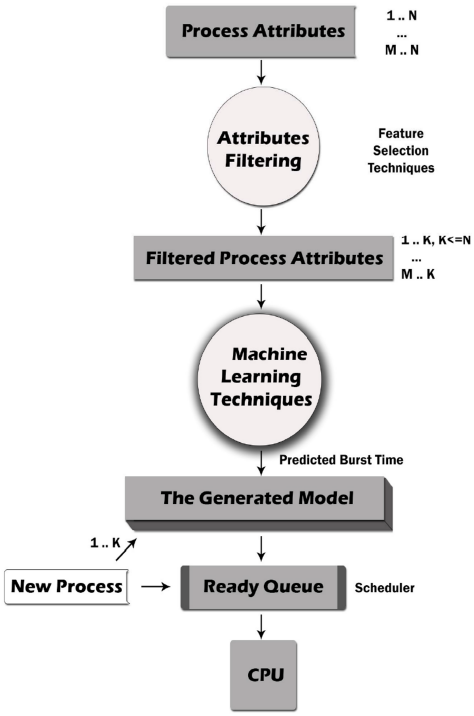


Fig. 1. The proposed ML-based approach [9]

Priority scheduling (PS) is another widely used CPU scheduling where each process that arrives at the ready queue is assigned a priority value based on some criteria and given CPU access in the order of highest to lowest priority. There is a risk of low priority processes getting stuck in the ready queue for indefinite periods of time which is known as starvation. A combination of PS with RR has been proposed by Chandiramani *et al* [18] where they show that this modified version of PS alleviates the effects of starvation problem while at the same time improving the mean turnaround times and waiting times when compared to the original version of PS.

## III. CPU Scheduling Algorithms

### A. First Come First Served (FCFS)

This is largely considered as the simplest CPU scheduling algorithm. The processes in the ready queue are given CPU access based on their arrival time at the ready queue. Processes that arrive earlier will get CPU access earlier than the processes that arrive later at the ready queue. This is a non-preemptive algorithm, meaning that once the CPU starts executing a process, until that process finishes, CPU access will not be given to another process. If a particular process has a large CPU burst time, all other processes will have to wait for longer times in the ready queue, which will lead to poor CPU utilization, poor throughput, large average turnaround times, and large average waiting times. This drawback of FCFS where processes with shorter CPU burst times are being held up due to CPU being occupied with processes with large CPU burst times is known as 'convoy effect'. The TABLE 1 below shows arrival times (AT), burst times (BT), and priorities for seven processes in the ready queue.

TABLE I
INFORMATION ABOUT PROCESSES

| P | AT | BT | Priority |
|---|---|---|---|
| A | 0 | 5 | 3 |
| B | 1 | 3 | 5 |
| C | 2 | 4 | 7 |
| D | 3 | 6 | 11 |
| E | 4 | 2 | 9 |
| F | 5 | 5 | 13 |
| G | 6 | 7 | 10 |

The corresponding completion times (CT), turnaround times (TAT), and waiting times (WT) are shown in TABLE II. We have assumed that the time taken for context switching is zero. We have used the relationships $TAT = CT - AT$ and $WT = TAT - BT$. Fig.2 shows the corresponding Gantt chart.

TABLE II
TURNAROUND TIME AND WAITING TIME CALCULATION FOR FCFS

| P | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| A | 0 | 5 | 5 | 5 | 0 |
| B | 1 | 3 | 8 | 7 | 4 |
| C | 2 | 4 | 12 | 10 | 6 |
| D | 3 | 6 | 18 | 15 | 9 |
| E | 4 | 2 | 20 | 16 | 14 |
| F | 5 | 5 | 25 | 20 | 15 |
| G | 6 | 7 | 32 | 26 | 19 |

Average $TAT = (5 + 7 + 10 + 15 + 16 + 20 + 26)/7 = 14.1429$

Average $WT = (0+4+6+9+14+15+19)/7 = 9.57143$

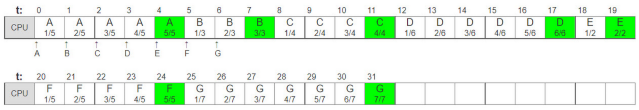

Fig. 2. Gantt chart for FCFS

### B. Shortest Job First (SJF)

This algorithm overcomes the convoy effect of FCFS by giving earlier CPU access to processes with shorter BT. This can be either preemptive or non-preemptive. The preemptive version is also known as the Shortest Remaining Time First (SRTF) algorithm. In SRTF, if a new process arrives while the current process is being executed by the CPU, and if the new process has a shorter BT than the remaining part of the current process, the current process will be preempted and the new process will be given CPU access. In the non-preemptive SJF algorithm, the process which arrived first will be given CPU access first and it will run till its completion. Out of the processes that have arrived by the CT of the first process, the job with the shortest BT will be given access to CPU and this pattern will continue. The application of non-preemptive SJF for the processes in TABLE I, result in TAT values and WT values given in TABLE III. Fig.3 shows the corresponding Gantt chart.

TABLE III
TURNAROUND TIME AND WAITING TIME CALCULATION FOR SJF

| P | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| A | 0 | 5 | 5 | 5 | 0 |
| B | 1 | 3 | 10 | 9 | 6 |
| C | 2 | 4 | 14 | 12 | 8 |
| D | 3 | 6 | 25 | 22 | 16 |
| E | 4 | 2 | 7 | 3 | 1 |
| F | 5 | 5 | 19 | 14 | 9 |
| G | 6 | 7 | 32 | 26 | 19 |

Average $TAT = (5+9+12+22+3+14+26)/7 = 13$

Average $WT = (0+6+8+16+1+9+19)/7 = 8.42857$

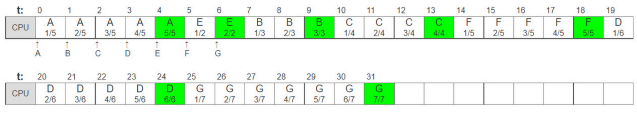

Fig. 3. Gantt chart for SJF

The application of SRTF for the processes in TABLE I, results in TAT values and WT values given in TABLE IV. Fig.4 shows the corresponding Gantt chart.

Average $TAT = (10+3+12+22+2+14+26)/7 = 12.7143$

Average $WT = (5+0+8+16+0+9+19)/7 = 8.14286$

TABLE IV
TURNAROUND TIME AND WAITING TIME CALCULATION FOR SRTF

| P | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| A | 0 | 5 | 10 | 10 | 5 |
| B | 1 | 3 | 4 | 3 | 0 |
| C | 2 | 4 | 14 | 12 | 8 |
| D | 3 | 6 | 25 | 22 | 16 |
| E | 4 | 2 | 6 | 2 | 0 |
| F | 5 | 5 | 19 | 14 | 9 |
| G | 6 | 7 | 32 | 26 | 19 |

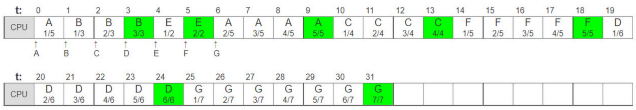

Fig. 4. Gantt chart for SRTF

We can see that both SJF and SRTF show improved TAT and WT values when compared to FCFS. SRTF shows improved TAT and WT values relative to SJF, although SRTF involves more context switching. There is a risk of processes with longer BTs getting starved in the case of a long string of shorter burst time processes arriving at the ready queue. A major problem in implementing either of these algorithms, as is, is not having an exact method to calculate the BT, prior to their execution. As was discussed in section II, a machine learning method [4] can be used to approximate the BT. Another method is to use exponential averaging [19]. Kumar *et al* [20] have proposed a method to approximate the next BT based on the dual simplex method.

### C. Round Robin (RR)

The RR algorithm is similar to FCFS in the way it treats the order of processes in the ready queue in that new processes get added to the end of the ready queue. The main distinction is the existence of a time quantum (TQ) sometimes referred to as a time slice, after which time if the BT of current process is larger than TQ, the current process is preempted (which is then added to the end of the ready queue) and the next process is given the CPU access. If the current process has a less BT than TQ, at CT of the current process, the next process in queue will be given access. If the TQ is too large relative to the BTs of the processes, RR becomes similar to FCFS. If the TQ is too small when compared to BTs, there will be many context switches which adversely affects the performance of the process. The value of TQ can be either static or dynamic. Although having a static TQ is good for equal time sharing, some processes which are highly interactive when compared to others, need a relatively larger TQ and therefore having numerous context switches can become cumbersome. This can be overcome by having a dynamic TQ which gets updated according to the BTs of the previous processes. Application of RR with a static TQ of 2 to processes in TABLE I,

results in TATs and WTs given by TABLE V. Fig.5 shows the corresponding Gantt chart.

TABLE V
TURNAROUND TIME AND WAITING TIME CALCULATION FOR RR

| P | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| A | 0 | 5 | 20 | 20 | 15 |
| B | 1 | 3 | 13 | 12 | 9 |
| C | 2 | 4 | 19 | 17 | 13 |
| D | 3 | 6 | 28 | 25 | 19 |
| E | 4 | 2 | 12 | 8 | 6 |
| F | 5 | 5 | 29 | 24 | 19 |
| G | 6 | 7 | 32 | 26 | 19 |

Average $TAT = (20 + 12 + 17 + 25 + 8 + 24 + 26)/7 = 18.8571$

Average $WT = (15+9+13+19+6+19+19)/7 = 14.2857$

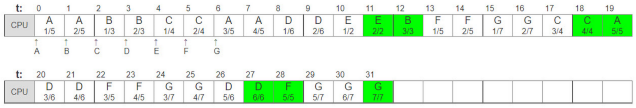

Fig. 5. Gantt chart for RR

### D. Priority based (PB)

In PB scheduling, there is a numerical value assigned to each process which denotes the priority ranking of that process. Depending on the context, a small numerical value might mean a higher priority or a lower priority. Higher priority processes will be given CPU access earlier when compared to lower priority processes. SJF and SRTF can be considered as versions of PB algorithm where the numerical figure that governs priority are either CPU burst time or the remaining CPU burst time. Just like with SJF and SRTF, there is a risk of low priority processes getting starved (i.e. being held up in the ready queue without CPU access for longer times) if there is a long string of high priority processes getting added to the ready queue. One method to overcome starvation is 'aging' where the priority of a process is gradually increased with its waiting time in the queue. PB scheduling can be either preemptive or non-preemptive. Application of non-preemptive PB on processes in TABLE I results in TATs and WTs given by TABLE VI. Fig.6 shows the corresponding Gantt chart.

Average $TAT = (5 + 7 + 10 + 24 + 10 + 27 + 15)/7 = 14$

Average $WT = (0 + 4 + 6 + 18 + 8 + 22 + 8)/7 = 9.4286$

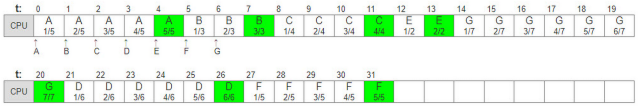

Fig. 6. Gantt chart for non-preemptive PB

TABLE VI
TURNAROUND TIME AND WAITING TIME CALCULATION FOR NON-PREEMPTIVE PB

| P | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| A | 0 | 5 | 5 | 5 | 0 |
| B | 1 | 3 | 8 | 7 | 4 |
| C | 2 | 4 | 12 | 10 | 6 |
| D | 3 | 6 | 27 | 24 | 18 |
| E | 4 | 2 | 14 | 10 | 8 |
| F | 5 | 5 | 32 | 27 | 22 |
| G | 6 | 7 | 21 | 15 | 8 |

### E. Multilevel queue (MLQ) and multilevel feedback queue (MLFQ)

In both MLQ and MLFQ, the ready queue is divided into multiple queues. Each queue can have its own scheduling algorithm. Depending on the nature of processes in each queue, separate processors can also be assigned to each queue. Both these algorithms are relatively complex algorithms when compared to other algorithms discussed so far. The scheduler has to not only schedule processes in a given queue, it also has to schedule processes between queues. Scheduling processes between processes can be done based on a fixed priority or a time slice. If a fixed priority is used, for example in an instance where there are two queues, Q1 and Q2, the scheduler can specify processes in Q1 to have a fixed priority over processes in Q2. In such a case, processes in Q2 will only have CPU access after giving CPU access to processes in Q1. This can lead to starvation of processes in Q2. If a time slice is used, the scheduler can, for example, specify 80% of the time, Q1 will have CPU access, and 20% of the time Q2 will have CPU access.

MLFQ has more flexibility when compared to MLQ in that, unlike in the case of MLQ, the processes can switch between the queues, which can mitigate the risk of starvation. Therefore, a given MLFQ is specified by the number of queues, the scheduling algorithm used in each queue, the mechanism used to promote (move to a higher queue) or demote (move to a lower queue) processes among queues.

## CONCLUSION

This article provides a gentle walkthrough regarding various characteristics of some of the most popular and widely used CPU scheduling algorithms. The reader has been provided with guided examples to compare the efficiency of the algorithms, some of their advantages, disadvantages and remedies. The field of CPU scheduling is rapidly evolving and sophisticated algorithms are being created by researchers at a robust pace. In this regard this article has provided key information to provide a good foundational knowledge about CPU scheduling to inquisitve minds.

## REFERENCES


[1] A. Silberschatz, G. Gagne, and P. Galvin, *Operating System Concepts*, 10th ed. Wiley, 2018, p. 200. [Online]. Available: https://books.google.lk/books?id=FHJlDwAAQBAJ

[2] A. A. AL-Bakhrani, A. A. Hagar, A. A. Hamoud, and S. Kawathekar, "Comparative analysis of cpu scheduling algorithms: Simulation and its applications," *International Journal of Advanced Science and Technology*, vol. 29, no. 3, pp. 483–494, 2020.

[3] T. Aladwani, *Types of Task Scheduling Algorithms in Cloud Computing Environment*, 04 2020, pp. 4–7.

[4] W. Zhao and J. Stankovic, "Performance analysis of fcfs and improved fcfs scheduling algorithms for dynamic real-time computer systems," in *[1989] Proceedings. Real-Time Systems Symposium*, 1989, pp. 156–165.

[5] H. Parekh and S. Chaudhari, "Improved round robin cpu scheduling algorithm: Round robin, shortest job first and priority algorithm coupled to increase throughput and decrease waiting time and turnaround time," 12 2016, pp. 184–187.

[6] M. Thombare, R. Sukhwani, P. Shah, S. Chaudhari, and P. Raundale, "Efficient implementation of multilevel feedback queue scheduling," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2016, pp. 1950–1954.

[7] R. kumar Yadav and A. Upadhyay, "A fresh loom for multilevel feedback queue scheduling algorithm," *International Journal of Advances in Engineering Sciences*, vol. 2, pp. 21–23, 2012.

[8] D. Maste, L. Ragha, and N. Marathe, "Intelligent dynamic time quantum allocation in mlfq scheduling," in *International Journal of Information and Computation Technology*, vol. 3, no. 4. International Research Publications House, 2013, pp. 311–322.

[9] K. Lalit, S. Rajendra, and S. Praveen, "Optimized scheduling algorithm," *International Journal of Computer Applications*, pp. 106–109, 2011.

[10] S. Kathuria, P. Singh *et al.*, "A revamped mean round robin (rmrr) cpu scheduling algorithm," *Int J Innov Res Comput Commun Eng*, vol. 4, pp. 6684–6691, 2016.

[11] P. Pradhan, P. K. Behera, and B. Ray, "Modified round robin algorithm for resource allocation in cloud computing," *Procedia Computer Science*, vol. 85, pp. 878–890, 2016, international Conference on Computational Modelling and Security (CMS 2016). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050916306287

[12] H. S. Behera, R. Mohanty, P. Jajnaseni, D. Thakur, and S. Sahoo, "Experimental analysis of new fair-share scheduling algorithm with weighted timeslice for real time systems," *Journal of Global Research in Computer Sciences*, vol. 2, pp. 54–60, 2011.

[13] T. Omotehinwa, I. Azeeze, and E. Oyekanmi, "An improved round robin cpuscheduling algorithm for asymmetrically distributed burst times," *Africa Journal Management Information System*, vol. 1, no. 4, pp. 50–68, 2019.

[14] A. Abdulrahim, A. Saleh, and S. Junaidu, "A new improved round robin (nirr) cpu scheduling algorithm," *International Journal of Computer Applications*, vol. 90, 02 2014.

[15] M. K. Mishra and F. Rashid, "An improved round robin cpu scheduling algorithm with varying time quantum," *International Journal of Computer Science, Engineering and Applications*, vol. 4, pp. 1–8, 2014.

[16] A. R. Dash, S. k. Sahu, and S. K. Samantra, "An optimized round robin cpu scheduling algorithm with dynamic time quantum," *International Journal of Computer Science, Engineering and Information Technology*, vol. 5, no. 1, p. 07–26, Feb 2015. [Online]. Available: http://dx.doi.org/10.5121/ijcseit.2015.5102

[17] T. Helmy, S. Al-Azani, and O. Bin-Obaidellah, "A machine learning-based approach to estimate the cpu-burst time for processes in the computational grids," in *2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, 2015, pp. 3–8.

[18] K. Chandiramani, R. Verma, and M. Sivagami, "A modified priority preemptive algorithm for cpu scheduling," *Procedia Computer Science*, vol. 165, pp. 363–369, 2019, 2nd International Conference on Recent Trends in Advanced Computing ICRTAC -DISRUP - TIV INNOVATION , 2019 November 11-12, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920300454

[19] A. Silberschatz, G. Gagne, and P. Galvin, *Operating System Concepts*, 10th ed. Wiley, 2018, p. 208. [Online]. Available: https://books.google.lk/books?id=FHJlDwAAQBAJ

[20] M. R. Mahesh Kumar, B. R. Rajendra, C. K. Niranjan, and M. Sreenatha, "Prediction of length of the next cpu burst in sjf scheduling algorithm using dual simplex method," in *Second International Conference on Current Trends In Engineering and Technology - ICCTET 2014*, 2014, pp. 248–252.