UMYU Scientifica, Vol. 4 NO. 1, March 2025, Pp 305 – 316
https://doi.org/10.56919/usci.2541.030
ISSN: 2955 – 1145 (print); 2955 – 1153 (online)
A periodical of the Faculty of Natural and Applied Sciences, UMYU, Katsina

ORIGINAL RESEARCH ARTICLE

An Enhanced and Improved Half-Life Variable Quantum Time Round Robin (EImHLVQTRR) CPU Scheduling Algorithm

Suleiman Ebaiya Abubakar[1] and Ibrahim Isa[1]
[1]Department of Computer Science, Federal University of Education Kano State-Nigeria

ABSTRACT
The processing speed and powers of Computer systems is a function of Central Processing Unit (CPU) efficiency. Depending on the algorithm used to implement a particular system, the turnaround time, waiting time, response time, and number of context switch are responsible for reducing CPU idle time and, overall, slowing down computer system processing power. There are many existing scheduling algorithms; among them is the "Improved Half Life Variable Quantum Time with Mean Time Slice Round Robin (ImHLVQTRR) Algorithm," which has been proposed to address the starvation problem – delay in access to the requested resources experienced in most of the earlier algorithms. This paper aims to enhance (the ImHLVQTRR) algorithm by modifying the time quantum (TQ), thereby improving system performance. To achieve this, a square root of the product of the processes average burst time and minimum burst time was computed to determine the TQ. The computed TQ is then used to execute RQ processes in an iterative manner within a specified period of time until the ready queue is empty. Overall, the experimental analysis shows that the proposed (EImHLVQTRR) algorithm performed better in terms of AWT of 371ms as against 390ms and 371ms, ATAT of 399ms as against 423ms and 399ms, ART of 71ms as against 236ms and 88ms and NCS of 91 as against 46 and 65; AWT of 326ms as against 350ms and 326ms, ATAT of 351ms as against 378ms and 351ms, ART of 51ms as against 153ms and 59ms and NCS of 77 as against 45 and 57 in both Zero and Non-Zero Arrival Time simulation results for the processes generated as shown in Figure 10 & 13 respectively. The experimental results also show some significant improvement as the ATAT of 22.6ms as against 27.2ms and 29.6ms, AWT of 13.4ms as against 18.0ms and 20.4ms, ART of 4.8ms as against 9.0ms and 11.8ms with equal number of context switch for Zero Arrival Time; ATAT of 30.6ms as against 32.2ms and 40.6ms, AWT of 18.6ms as against 20.2ms and 28.6ms, ART of 18.6ms as against 10.6ms and 11.2ms and NCS of 4 as against 6 and 7 for Non-Zero Arrival Time experimental summary results shown in Table 3 & 4.

ARTICLE HISTORY
Received December 13, 2024
Accepted March 4, 2025
Published March 31, 2025

KEYWORDS
Burst Time (BT), Gantt chart, Ready Queue (RQ), Remaining Burst Time (RBT), Round Robin, and Throughput.

INTRODUCTION
The multitasking capabilities of the evolving operating system have made it possible for multiple tasks to run concurrently in a system. Operating systems are responsible for choosing tasks available in the ready queue (RQ) and allocating them to the CPU; this act is called Scheduling (Fiad et al., 2020). When discussing operating systems, the terms multiprocessing and multitasking are used. These terms are utilized interchangeably with one another. In a multiple CPU system, it is called multiprocessing; the CPU rapidly alternates between programs, creating the illusion for user that all processes are concurrently at the same time; this is referred to as multitasking. The two forms of multitasking used are non-preemptive and pre-emptive techniques. When an operating system allots some portion of the CPU to the available processes to run, it is referred to as preemptive multitasking, while for non-preemptive multitasking, every process holds the CPU control to execute to finish (Sakshi et al., 2022). The primary aim of the CPU scheduling algorithm is to maximize the system's speed, fairness, and effectiveness. CPU utilization, context switching, throughput, waiting time, turnaround time, and response time are performance metrics (Vayadande et al., 2023). The use of scheduling algorithms is crucial in a situation where multiple processes are available for execution and in deciding which of the processes should go first. Round Robin (RR) algorithm is the most widely utilized algorithm that prioritizes processes ready for execution (Mostafa & Amamo, 2020). It executes the process using time quantum (TQ) (Richardson & Istiono, 2022). If the burst time of any active process exceeds one TQ, such process is moved to the tail of the RQ after preemption. If new processes arrive in the queue, they are again placed at the tail of the queue. RR is the most

Correspondence: Suleiman Ebaiya Abubakar. Department of Computer Science, Federal University of Education Kano State-Nigeria. ✉ mall.sule10@gmail.com.
How to cite: Abubakar, S. E., & Isa, I. (2025). An Enhanced and Improved Half-Life Variable Quantum Time Round Robin (EImHLVQTRR) CPU Scheduling Algorithm. *UMYU Scientifica*, 4(1), 305 – 316. https://doi.org/10.56919/usci.2541.030

frequently used technique for time-sharing systems, that is multiusers operating systems (Hayatunnufus et al., 2020). It utilizes shared resources effectively and enhances response time. In the RR technique, dynamic time quantum is used to enhance its performance. This paper aimed at improving the efficacy of the existing ImHLVQTRR algorithm to achieve optimum system performance.

According to Matarneh (2009) & Singh et al. (2010), due to the numerous scheduling existing algorithms, some of the processes tend to benefit more than others due to the size of their burst times. In order to have an effective system, the following criteria are considered: *CPU Utilization* – to enhance CPU efficiency and prevent CPU cycle surplus, the CPU is required to be ideally 100% time busy. Its consumption in a real-time system should be between 40% and 90%. *Throughput* – This measures the amount of processes or tasks completed in a specific time frame. It is significant to increase throughput for better efficiency and productivity of a system. *Turnaround Time (TAT)* – The overall time processes required to complete execution when it arrives at the ready queue. Minimizing it is crucial as it impacts the overall system efficiency and user satisfaction. *Waiting Time (WT)* – The time it takes processes to wait before getting access CPU for execution. It is also important to minimize it because it affects scheduling algorithm efficiency and user satisfaction. *Response Time (RT)* – The time required for system to react to user's request. Short response time signifies that the system is executing tasks swiftly and efficiently. *Context Switch (CS)* – This technique enables processes to switch CPU from one state to another state during execution. For an efficient and effective system, minimizing context switch is very important.

The following are some of the existing CPU scheduling algorithms: *Priority Scheduling* – This technique is preemptive. It allows the operating system to interrupt and switch between the highest-priority processes to allocate the CPU. In a situation where multiple processes have equal BT, the FCFS (First Come, First Serve) technique is used to allot processes to the CPU. ***Round Robin (RR)* –** This is a preemptive technique. In this technique, processes are assigned a fixed time slice to execute. Processes are executed in rounds; if a process is not completed in the allotted time, it is preempted and placed behind the available processes in the ready queue. ***First-Come, First-Served (FCFS)* –** The FCFS technique is a basic operating system scheduling algorithm. This technique executes processes in the FIFO (first in, first out) technique; the process that requests the CPU first gets it. The sequence in which processes appear in the queue determines their scheduling. ***Shortest Job First (SJF)* –** Shortest Job First (SJF) technique executes processes in order of the processes BT from the smallest to the largest. The scheduler chooses the process with the least burst time in the ready queue for execution. This step is iterated until the RQ becomes empty. This algorithm is considered to be non-preemptive.

In the (Ashiru et al., 2014) proposed algorithm, the time quantum was considered to be half of each processes' BT (i.e., $TQ = \frac{P(BTi)}{2}$) in the first round of execution and then preempted for other processes to be executed, too. The amount of processes' burst times left are executed in the second round to complete and terminate. This algorithm significantly enhances multiprogramming regardless of the differences in available processes' BT. The starvation issue of the Shortest Job First is hereby addressed since each process must be executed halfway to pave the way for other processes to access the CPU. The algorithm was evaluated with a standard round-robin, and the results are promising. The Mishra & Rashid (2014) proposed algorithm employed both qualities of SJF and RR algorithms with dynamic TQ. The SJF technique was used in selecting processes for execution and RR technique to execute processes. This algorithm initially arranged processes in the RQ according to their burst times. After sorting the processes, the first process BT determines the TQ for executing available processes. For every round of execution, the processes are rearranged in ascending order of their BT in the RQ for execution, and the first process BT is taken to be the next TQ for executing the remaining processes. These steps are reiterated until the RQ is empty. At the end, the average waiting time and turnaround time, as well as the context switches, are calculated. The experimental evaluation was done with the Standard Round Robin for zero and non-zero arrival times and the results indicated better performance than the RR algorithm. Sharma & Kakhani (2015) algorithm examined the "Adaptive Round Robin (ARR) Scheduling Technique" aimed at enhancing system performance. The TQ was determined to be the sum of the RQ available processes' BT divided by 2n (i.e., $TQ = \frac{Sum(BTi)}{2n}$) where n stands for the size of the processes. The processes were organized in ascending and descending sequences of their BT in the two phase's experimental analysis. In both cases, there were improvements compared to the existing algorithms like standard Round Robin and Adaptive Round Robin algorithms.

The proposed (Sohrawordi et al., 2019) algorithm used a dynamic time quantum of average of processes bust time values (i.e., $Tq = Average(BTi)$) available in the queue, after which processes are assigned CPU to execute for the first round. The executed processes are terminated and eliminated from the RQ; otherwise, they are placed behind the processes in the RQ. For the next round of execution, the processes left in the queue are orderly sorted again and a new TQ is recalculated to execute the processes. These steps are repeated till the queue is empty. The (Mody & Mirkar 2019) proposed algorithm focused on the essential part of CPU Scheduling. Here, the time slice was dynamically determined using two components known as Delta and Smart Time Quantum (STQ). The TQ is determined to be the sum of STQ and Delta, where STQ stands for the disparity between the nearby processes' burst times while the Delta is (STQ)/2. The available processes are originally organized in the RQ in sequence of their BT, and each process is executed for one TQ.

In every round of execution, a new TQ is calculated. Any active process after a round of execution with RBT not up to one Time Quantum (TQ) is reassign CPU to finish and then removed from the queue. In this algorithm, processes with shorter BT were given higher priority to complete executing and terminate in a single round. The algorithm proposed by (Qazi et al., 2019) is a modified version of the RR algorithm aimed at reducing ATAT, AWT, and NCS. The algorithm sorts all incoming processes based on their BT and dynamically assigns an optimal TQ as the square root of the sum of mean of the processes' burst times and combine time(C.T) where the C.T is the sum of the highest burst time and the lowest burst to each process using SJF algorithm. The process with the least execution time (ET) is executed first. When a new process arrives at the RQ when the BT is not 0 (i.e., the RQ is not empty), a new TQ is again computed for the next execution cycle. The dynamic TQ was in the context of Web Server Scheduling where multiple user's requests must be served concurrently. The algorithm was compare with five other existing algorithms like RR, IRR, SARR, SJRR, and ARRS scheduling algorithms, and the results are promising. Ali et al. (2020) proposed an algorithm mainly focused on improving the Round Robin scheduling algorithm to increase CPU utilization and throughput and minimize AWT, ATT, ART, and NCS. In this technique, the TQ is dynamically determined as approximate sum of the mean BT of available processes and the minimum BT (i.e., $\mathrm{TQ} = \mathrm{ceil}(\mathrm{Mean.}(\mathrm{BT}) + \mathrm{Min}(\mathrm{BT}))$ named "Enhanced Time Quantum (ETQ)." The algorithm operates in two phases; in the first phase, when the ETQ is determined process with the shortest BT is given more priority and allocated CPU first. This phase is repeated until all the processes are executed for one ETQ, and the processes that complete execution are terminated and removed from the RQ. In the second phase, the remaining processes in the RQ are sorted in sequential order of their BT for another round of execution. In this phase, processes are allocated CPU for execution. When the outstanding burst time of the active process is not up to or is same as one ETQ, the process is reassigned CPU to further execute to finish and terminate. These steps are repeated till the ready queue becomes empty. This algorithm was evaluated with standard round robin where the proposed (HYRR) technique outperformed the round robin algorithm.

Abdelhafiz (2021) proposed algorithm mainly focused on calculating effective time quantum that optimizes the scheduling algorithm performance. This technique assumed the processes are all available in the RQ sorted according to their BT. Then, the TQ is determine to be median of the ready queue processes BT multiplied by 2 (i.e., $\mathrm{TQ} = 2 * \mathrm{Median}(\mathrm{BTi})$). The median is taken to be the middle process' BT (for odd number of processes) or the sum of the two midpoint processes BT divided by 2 (for even number of processes), and then, the process leading in the RQ is allotted CPU to execute for one TQ. The system constantly checks if the outstanding burst time of any active process is not up to or is equal to one TQ; such process is reassign CPU to complete execution and then terminates from the system. However, it is placed behind the list of the processes in the ready queue for subsequent rounds of execution. This steps are repeated till the queue is empty. This technique was evaluated alongside some existing algorithms like standard Round Robin (RR), Round Robin Remaining Time (RRRT), and Enriched Round Robin (ERR), and at the end of comparison parameters such as average turnaround time, average waiting time, average response time and number of context switch were calculated for which this proposed VORR algorithm provided promising results. To optimize the scheduling algorithm's performance, (Abdelkader et al., 2022) proposed an algorithm to determine effective TQ. Initially, the available processes are sorted in accordance with the BT values before determining the TQ as the average-sum of the processes' BT median and the mean (i.e., $\mathrm{TQ} = \frac{(\mathrm{Median}(\mathrm{BTi}) + \mathrm{Mean}(\mathrm{BTi}))}{2}$). The mean is calculated as $\mathrm{Mean}(\mathrm{BTi}) = \frac{\sum_{i=1}^{n} BTi}{n}$ while the median is calculated in either of the two ways, as $\mathrm{Med}(\mathrm{BTi}) = [\mathrm{BT}(\frac{n+1}{2})]$ (if n is said to be Odd) or $\mathrm{Med}(\mathrm{BTi}) = \left[\mathrm{BT}\left(\frac{n}{2}\right)\right] + \left[\mathrm{BT}\left(\frac{n}{2} + 1\right)\right]/2$ (if n is said to be Even), n is said to be the size of the processes, and BT stands for processes' BT. When TQ is calculated, the process in the front of the queue is assigned to the CPU to execute for one TQ. The system constantly checks if the active process BT left is not up to or is equal to one TQ; such process is reassigned CPU to finish executing and terminate; otherwise, it is placed at the back of the remaining processes in RQ. These steps are reiterated until the queue is empty, then the average turnaround time, average waiting time, and average response time were calculated and displayed. Simon et al. (2022) proposed a CPU scheduling algorithm was an improvement over (Ashiru et al., 2014). In this technique, TQ was determined dynamically under the presumption that RQ contains all processes awaiting execution. The algorithm consists of two time quantum such as: $\mathrm{TQ1} = \mathrm{Average}(\mathrm{BT})$ and $\mathrm{TQ2} = \frac{\mathrm{P(BT)}}{2}$ when $\mathrm{P(BT)} > \mathrm{TQ1}$. The first TQ1 was used to execute shortest BT processes. However, when the process's BT exceed the estimated TQ1, such a process is executed half way in first round of execution while its remaining burst time is executed in the next round. In this technique, processes with BT less than TQ1 are executed and terminated in the first round. This technique is used to restrict process's execution to two rounds. The experimental analysis indicated this algorithm performed better than RR and HVQTRR algorithms. The proposed (Abubakar et al., 2023) algorithm is a modification of (Abubakar et al., 2016) aimed at improving system performance. The TQ was determined as the sum of RQ processes BT and the BT of the process with the highest response ratio (i.e., the least burst time) divided by 2 (i.e., $\mathrm{TQ} = \frac{\sum(\mathrm{Mean}(\mathrm{BT}) + \mathrm{HRRN}(\mathrm{BT}))}{2}$). The order of execution of the processes is in accordance with their arrival to the RQ in both zero and non-zero arrival time processes. In every execution cycle, an active process with a remaining BT lower than or the same as one TQ is reassigned to CPU to finish execution and terminate; otherwise, place behind the list of processes in the queue. These steps are repeated whenever the new TQ is calculated until the queue is empty. This technique was compared with four other existing algorithms, and the results were promising. Zohora et al. (2024) introduced a new enhanced RR approach for task scheduling in cloud computing systems. This algorithm computed and kept updating a dynamic TQ for process execution by considering the burst times of RQ processes. The TQ was computed as the square root of the calculated index (i.e., TQ = sqrt[index] and index = 0.8 * N; as n stand for the size of RQ processes), making sure the processes in RQ are completely executed in a single turn. To facilitate the execution process and increase system performance, the RBT of the running process is checked, and the scheduler decides if it should be reassigned to the CPU again to complete base on it RBT and current TQ. The algorithm look for the process with less burst time if the running process RBT is more than one-third of current TQ and assign CPU to the process which automatically preempt the active or running process. This algorithm was compared with enhanced round robin algorithms and it reduced AWT by 15.77% and context switching by 20.68%.

METHODOLOGY

In order to minimize the TT, WT and RT this work utilized dynamic TQ as the square root of the product of processes' average burst times and minimum burst time which was subjected to update after each execution cycle based on the remaining processes' burst times for the subsequent execution cycles.

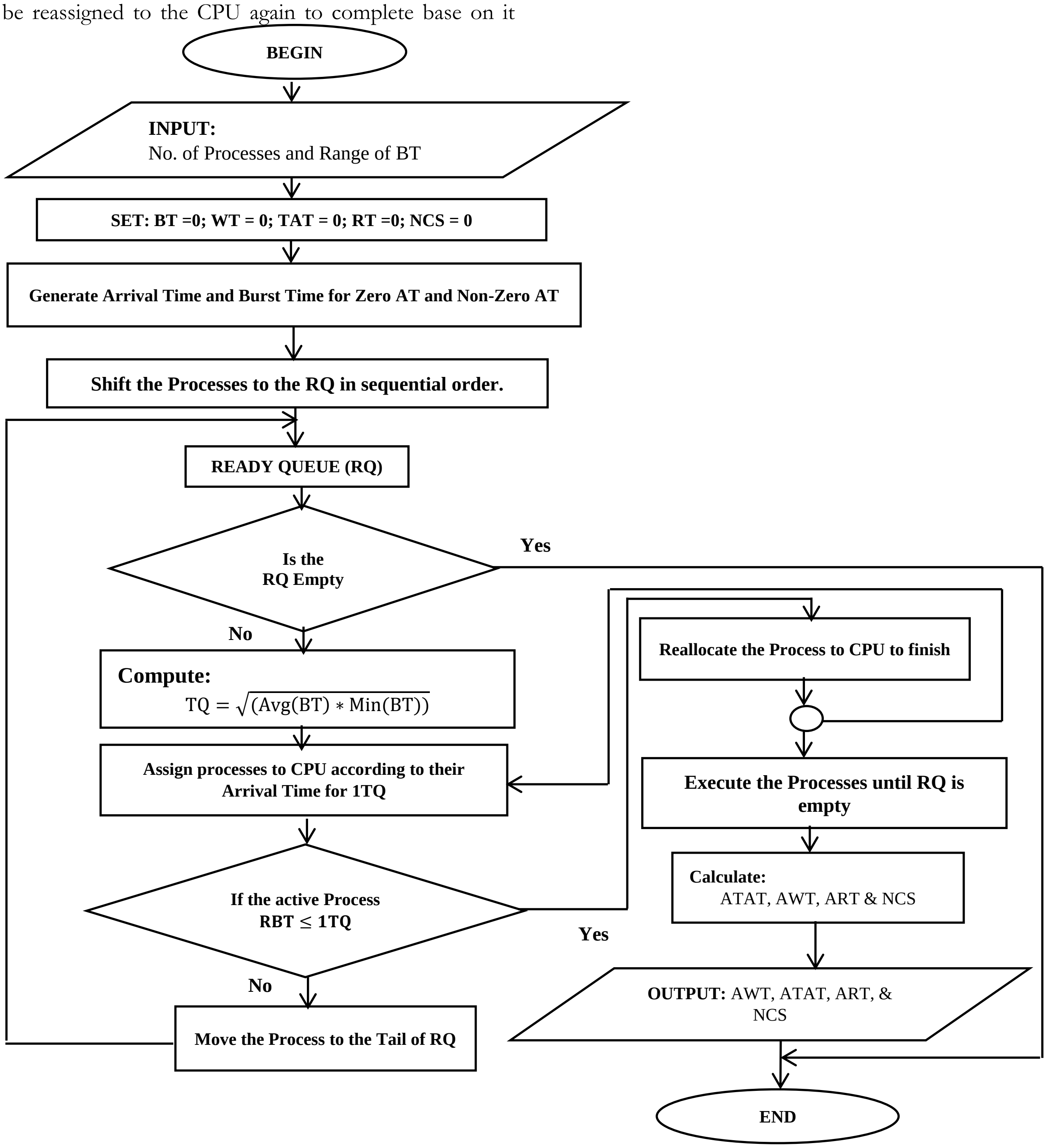


Figure 1: Flowchart for the Proposed (EImHVLQTRR) Algorithm

To further enhance the system performance and fairness, processes are executed in order of their arrival to the RQ, and those processes with shorter burst time as well as those whose their remaining burst times are less than or equal to the current TQ are assign and reassign CPU to finish execution and terminated from the system with aim of yielding a high throughput in both zero and non-zero arrival times processes.

Proposed EImHLVQTRR Algorithm

This algorithm mainly focuses on enhancing an existing ImHLVQTRR CPU scheduling algorithm. It maximizes CPU utilization, throughput and minimizes AWT, ATAT, ART, and NCS, and also works more effectively than the RR and ImHLVQTRR algorithms. In this EImHLVQTRR algorithm, the square root of the product of processes average burst time and minimum burst time was computed to determine TQ (i.e., $\mathrm{TQ} = \sqrt{(\mathrm{Average(BT)} * \mathrm{Min(BT)})}$). When the TQ is determined, it is then used to execute the RQ processes in sequence of their arrival for one time quantum. Any active processes with remaining BT not up to or is equal to one TQ (i.e. $\mathrm{P(BT)} \leq \mathrm{TQ}$) is reassign CPU again to finish execution and terminate. Otherwise, it is place at the tail of the available processes in the RQ for the next round of execution till ready queue becomes empty.

1.2 Experimental Analysis

In this study, we evaluate the performance of baseline algorithm – ImHLVQRR and RR with the proposed EImHLVQTRR in terms of zero arrival time and non-zero arrival time by comparing the AWT, ATAT, ART, and the NCS for the algorithms. Thus, the metrics were computed in each case by varying the TQ for each algorithm. The computations for averages using these parameters for each of the algorithms are presented under this sub-section.

3.2.1 Zero Arrival Time Case

In this instance, processes are sorted in random order of their burst times while the arrival time is presumed to be zero for a RQ consisting of five processes, P_1, P_2, P_3, P_4, and P_5, each with burst time as 2ms, 15ms, 11ms, 1ms, and 17ms respectively as shown in Table 1.

Table 1: Processes with Zero Arrival Time

Process ID	Arrival Time (ms)	Burst Time (ms)
P_1	0	2
P_2	0	15
P_3	0	11
P_4	0	1
P_5	0	17
TOTAL:		**46**

Figure 2 presents Round Robin Gantt chart for Zero Arrival Time case with a static TQ set at 10ms. In the first execution cycle, P_1 and P_4 terminates at 0 burst time, while P_2, P_3, and P_5 continue to executes for the second round until their burst time is 0.

RBT	0	5	1	0	7	0	0	0
	P_1	**P_2**	**P_3**	**P_4**	**P_5**	**P_2**	**P_3**	**P_5**

0 2 12 22 23 33 38 39 46

Figure 2: RR Algorithm Gantt chart for Zero AT

$$Number\ of\ Context\ Switch(NCS) = 7$$

$$Turnaround\ Time = \text{Process Completion Time} - \text{Arrival Time}$$

$P1 = 2 - 0 = 2\text{ms}; P2 = 38 - 0 = 38\text{ms}; P3 = 39 - 0 = 39\text{ms}; P4 = 23 - 0 = 23\text{ms}; \text{and } P5 = 46 - 0 = 46\text{ms}$

$$Average\ Turnaround\ Time = \frac{2+38+39+23+46}{5} = \frac{146}{5} = 29.6\text{ms}$$

$$Waiting\ Time = \text{Process Turnaround Time} - \text{Burst Time}$$

$P1 = 2 - 2 = 0\text{ms} = 0\text{ms}; P2 = 38 - 15 = 23\text{ms}; P3 = 39 - 11 = 28\text{ms}; P4 = 23 - 1 = 22\text{ms}; \text{and } P5 = 46 - 17 = 29\text{ms}$

$$Average\ Waiting\ Time = \frac{0+23+28+22+29}{5} = \frac{102}{5} = 20.4\text{ms}$$

$$Response\ Time = \text{Time process first have access to CPU} - \text{Process Arrival Time}$$

$P1 = 0 - 0 = 0\text{ms}; P2 = 2 - 0 = 2\text{ms}; P3 = 12 - 0 = 12\text{ms}; P4 = 22 - 0 = 22\text{ms}; \text{and } P5 = 23 - 0 = 23\text{ms}$

$$Average\ Response\ Time = \frac{0+2+12+22+23}{5} = \frac{59}{5} = 11.8\text{ms}$$

Figure 3 presents the Gantt chart for Zero Arrival Time case with $\mathrm{TQ1} = \frac{46}{5} \cong 9\text{ms}$ or $\mathrm{TQ2} = \frac{\mathrm{P(BT)}}{2}$ (if $\mathrm{P(BT)} > \mathrm{TQ1}$). In every round of execution, TQ1 is used as execution time (ET) if their burst time is less than or equal to the computed TQ1; otherwise the half of the process's burst time will be used, which is TQ2. In the first round all the five processes were executed for which P_1 and P_4 were terminated from the system because their burst time is zero while P_2, P_3 and P_5 were terminated after their second round of execution.

RBT	0	7	5	0	8	0	0	0
	P_1	P_2	P_3	P_4	P_5	P_2	P_3	P_5
0	2	10	16	17	26	33	38	46

Figure 3: ImHLVQTRR Algorithm Gantt chart for Zero AT

$\text{Number of Context Switch(NCS)} = 6$

$$\text{Turnaround Time} = \text{Process Completion Time} - \text{Arrival Time}$$

$P1 = 2 - 0 = 2ms; P2 = 33 - 0 = 33ms; P3 = 38 - 0 = 38ms; P4 = 17 - 0 = 17ms; \text{and } P5 = 46 - 0 = 46ms$

$\text{Average Turnaround Time} = \frac{2+33+38+17+46}{5} = \frac{136}{5} = 27.2ms$

$$\text{Waiting Time} = \text{Process Turnaround Time} - \text{Burst Time}$$

$P1 = 2 - 2 = 0ms; P2 = 33 - 15 = 18ms; P3 = 37 - 11 = 27ms; P4 = 17 - 1 = 16ms; \text{and } P5 = 46 - 17 = 29ms$

$\text{Average Waiting Time} = \frac{0+18+27+16+29}{5} = \frac{90}{5} = 18.0ms$

$$\text{Response Time} = \text{Time process first have access to CPU} - \text{Process Arrival Time}$$

$P1 = 0 - 0 = 0ms; P2 = 2 - 0 = 2ms; P3 = 10 - 0 = 10ms; P4 = 16 - 0 = 16ms; \text{and } P5 = 17 - 0 = 17ms$

$\text{Average Response Time} = \frac{0+2+10+16+17}{5} = \frac{45}{5} = 9.0ms$

In EImHLVQTRR algorithm time quantum was determined dynamically as $TQ = \sqrt{(\text{Average(BT)} * \text{Min(BT)})}$. Figure 4 presents the Gantt chart for Zero Arrival Time case with $TQ = \sqrt{9.2 * 1} = \sqrt{9} \cong 3ms$. In the first execution cycle all the five processes were executed but only P_1 and P_4 were terminated from the system because their burst time became 0 while P_2, P_3, and P_5 terminated in their second round of execution. When processes are executed, the active process is reassign CPU to further execution if its $P(RBT) \leq 1TQ$ to finish and terminate.

RBT	0	12	8	0	14	0	0	0
	P_1	P_2	P_3	P_4	P_5	P_2	P_3	P_5
0	2	5	8	9	12	24	32	46

Figure 4: EImHLVQTRR Algorithm Gantt chart for Zero AT

$\text{Number of Context Switch(NCS)} = 7$

$$\text{Turnaround Time} = \text{Process Completion Time} - \text{Arrival Time}$$

$P1 = 2 - 0 = 2ms; P2 = 24 - 0 = 24ms; P3 = 32 - 0 = 32ms; P4 = 9 - 0 = 9ms; \text{and } P5 = 46 - 0 = 46ms$

$\text{Average Turnaround Time} = \frac{2+24+32+9+46}{5} = \frac{113}{5} = 22.6ms$

$$\text{Waiting Time} = \text{Process Turnaround Time} - \text{Burst Time}$$

$P1 = 2 - 2 = 0ms; P2 = 24 - 15 = 9ms; P3 = 32 - 11 = 21ms; P4 = 9 - 1 = 8ms; \text{and } P5 = 46 - 17 = 29ms$

$\text{Average Waiting Time} = \frac{0+9+21+8+29}{5} = \frac{67}{5} = 13.4ms$

$$\text{Response Time} = \text{Time process first have access to CPU} - \text{Process Arrival Time}$$

$P1 = 0 - 0 = 0ms; P2 = 2 - 0 = 2ms; P3 = 5 - 0 = 5ms; P4 = 8 - 0 = 8ms; \text{and } P5 = 9 - 0 = 9ms$

$\text{Average Response Time} = \frac{0+2+5+8+9}{5} = \frac{24}{5} = 4.8ms$

Table 2: Summary of Algorithms for Zero Arrival Time case

Algorithms	TQ	ATAT	AWT	ART	NCS
RR	10ms	29.6ms	20.4ms	11.8ms	7
ImHLVQTRR	9ms or BT/2 (if P(BT) > TQ)	27.2ms	18.0ms	9.0ms	7
EImHLVQTRR	3ms and 9ms	22.6ms	13.4ms	4.8ms	7

The results in Table 2 indicated that the proposed (EImHLVQTRR) algorithm performed better in terms of ATAT of 22.6ms as against 27.2ms and 29.6ms for ImHLVQTRR and RR algorithms, AWT of 13.4ms as against 18.0ms and 20.4ms for ImHLVQTRR and RR algorithms and ART of 4.8ms as against 9.0ms and 11.8ms for ImHLVQTRR and RR algorithms with equal Number of Context Switches of 7 for Zero Arrival Time case.

3.2.2 Non-Zero Arrival Time Case

In this instance, while the processes' CPU BT were randomly arranged and the arrival time values were presumed to be non-zero for a RQ consisting of five processes, P1, P2, P3, P4, and P5 each with burst time as 0ms, 4ms, 8ms, 12ms, and 16ms and the burst time as 11ms, 17ms, 16ms, 6ms, and 10ms respectively as shown in Table 1 as shown in Table 3.

Figure 5 presents the Round Robin Gantt Zero Arrival Time case chart with TQ set to 10ms. The TQ is used to execute the processes in the RQ. In the first round of execution, only P_4 and P_5 got terminated from the system as their burst time equal to 0, while P_1, P_2 and P_3 got terminated in their second round of execution.

Table 3: Processes with Non-Zero Arrival Time

Process ID	Arrival Time (ms)	Burst Time (ms)
P_1	0	11
P_2	4	17
P_3	8	16
P_4	12	6
P_5	16	10
TOTAL:		**60**

RBT 1 6 0 0 0 0 0

P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3

0 10 20 30 36 46 47 54 60

Figure 5: RR Algorithm Gantt chart for Non-Zero AT

$\text{Number of Context Switch(NCS)} = 7$

$$\text{Turnaround Time} = \text{Process Completion Time} - \text{Arrival Time}$$

$P1 = 47 - 0 = 47ms; P2 = 54 - 4 = 50ms; P3 = 60 - 8 = 52ms; P4 = 36 - 12 = 24ms;$ and $P5 = 46 - 16 = 30ms$

$$\text{Average Turnaround Time} = \frac{47+50+52+24+30}{5} = \frac{203}{5} = 40.6ms$$

$$\text{Waiting Time} = \text{Process Turnaround Time} - \text{Burst Time}$$

$P1 = 47 - 11 = 36ms; P2 = 50 - 17 = 33ms; P3 = 52 - 16 = 36ms; P4 = 24 - 6 = 18ms;$ and $P5 = 30 - 10 = 20ms$

$$\text{Average Waiting Time} = \frac{36+33+36+18+20}{5} = \frac{143}{5} = 28.6ms$$

$$\text{Response Time} = \text{Time process first have access to CPU} - \text{Process Arrival Time}$$

$P1 = 0 - 0 = 0ms; P2 = 10 - 4 = 6ms; P3 = 20 - 8 = 12ms; P4 = 30 - 12 = 18ms; P5 = 36 - 16 = 20ms$

$$\text{Average Response Time} = \frac{0+6+12+18+20}{5} = \frac{56}{5} = 11.2ms$$

Figure 6 presents the ImHLVQTRR algorithm Gantt chart for the Zero Arrival Time case with $TQ1 = \frac{60}{5} \cong 12ms$ or $TQ2 = \frac{P(BT)}{2}$ (if P(BT)>TQ1). After the first round of execution, P_1, P_4, and P_5 were terminated from the system as their burst times were set to 0, while P_2 and P_3, with remaining burst times of 8ms, each got terminated after their second round of execution.

RBT 0 8 8 0 0 0 0

P_1	P_2	P_3	P_4	P_5	P_2	P_3

0 11 20 28 34 44 52 60

Figure 6: ImHLVQTRR Algorithm Gantt chart for Non-Zero AT

$\text{Number of Context Switch(NCS)} = 6$

$$\text{Turnaround Time} = \text{Process Completion Time} - \text{Arrival Time}$$

$P1 = 11 - 0 = 11ms; P2 = 52 - 4 = 48ms; P3 = 60 - 8 = 52ms; P4 = 34 - 12 = 22ms;$ and $P5 = 44 - 16 = 28ms$

$$\text{Average Turnaround Time} = \frac{11+48+52+22+28}{5} = \frac{161}{5} = 32.2ms$$

$$\text{Waiting Time} = \text{Process Turnaround Time} - \text{Burst Time}$$

$P1 = 11 - 11 = 0ms; P2 = 48 - 17 = 31ms; P3 = 52 - 16 = 36ms; P4 = 22 - 6 = 16ms;$ and $P5 = 28 - 10 = 18ms$

$$\text{Average Waiting Time} = \frac{0+31+36+16+18}{5} = \frac{101}{5} = 20.2ms$$

$$\text{Response Time} = \text{Time process first have access to CPU} - \text{Process Arrival Time}$$

$P1 = 0 - 0 = 0ms; P2 = 11 - 4 = 7ms; P3 = 20 - 8 = 12ms; P4 = 28 - 12 = 16ms;$ and $P5 = 34 - 16 = 18ms$

$$\text{Average Response Time} = \frac{0+7+12+16+18}{5} = \frac{53}{5} = 10.6ms$$

Figure 7 presents the EImHLVQTRR algorithm Gantt chart for the Zero Arrival Time case with $TQ = \sqrt{12 * 6} = \sqrt{72} \cong 9ms$. In each execution cycle, the active process with RBT $P(RBT) \leq 1TQ$ is reassign CPU to finish and terminate). All the processes were executed in one execution cycle.

RBT	0	0	0	0	0
	P_1	**P_2**	**P_3**	**P_4**	**P_5**
0	11	28	44	50	60

Figure 7: EImHLVQTRR Algorithm Gantt chart for Non-Zero AT

$$\text{Number of Context Switch(NCS)} = 4$$

$$\text{Turnaround Time} = \text{Process Completion Time} - \text{Arrival Time}$$

$P1 = 11 - 0 = 11ms; P2 = 28 - 4 = 24ms; P3 = 44 - 8 = 36ms; P4 = 50 - 12 = 38ms;$ and $P5 = 60 - 16 = 44ms$

$$\text{Average Turnaround Time} = \frac{11+24+36+38+44}{5} = \frac{153}{5} = 30.6ms$$

$$Waiting\ Time = \text{Process Turnaround Time} - \text{Burst Time}$$

$P1 = 11 - 11 = 0ms = 0ms; P2 = 24 - 17 = 7ms; P3 = 36 - 16 = 20ms; P4 = 38 - 6 = 32ms;$ and$P5 = 44 - 10 = 34ms$

$$\text{Average Waiting Time} = \frac{0+7+20+32+34}{5} = \frac{93}{5} = 18.6ms$$

$$\text{Response Time} = \text{Time process first have access to CPU} - \text{Process Arrival Time}$$

$P1 = 0 - 0 = 0ms; P2 = 11 - 4 = 7ms; P3 = 28 - 8 = 20ms; P4 = 44 - 12 = 32ms;$ and $P5 = 50 - 16 = 34ms$

$$\text{Average Response Time} = \frac{0+7+20+32+34}{5} = \frac{93}{5} = 18.6ms$$

Table 4: Summary of Algorithms for Non-Zero Arrival Time case

Algorithms	TQ	ATAT	AWT	ART	NCS
RR	10ms	40.6ms	28.6ms	11.2ms	7
ImHLVQTRR	12ms or BT/2 (if P(BT) > TQ)	32.2ms	20.2ms	10.6ms	6
EImHLVQTRR	9ms	30.6ms	18.6ms	18.6ms	4

The results in Table 4 indicate that the proposed (EImHLVQTRR) algorithm performed better in terms of ATAT of 30.6ms as against 32.2ms and 40.6ms for ImHLVQTRR and RR algorithms, AWT of 18.6ms as against 20.2ms and 28.6ms for ImHLVQTRR and RR algorithms and ART of 18.6ms as against 10.6ms and 11.2ms for ImHLVQTRR and RR algorithms with Number of Context Switches of 4 as against 6and 7 for ImHLVQTRR and RR Non-Zero Arrival Time case.

DISCUSSION OF RESULTS

To implement the proposed (EImHLVQTRR) algorithm, a process generator interface was constructed for generating set of processes. The processes generated are denoted by tuples: < (Process_ID, *Arrival_Time*, *Burst_Time*) > as shown in Figures 9 and 12. The process arrival times were expressed in Zero and Non-Zero Arrival Times. Uniform distribution was used to generate the processes' burst times for both Zero and Non-Zero Arrival Time cases, while Poisson distribution was used for Non-Zero Arrival Time. The system Hardware and Software requirements used for designing the interface are: *Hardware* – HP Elite-Book 6930p, Intel(R) Corel(TM)2 Duo CPU P8600 @ 2.40GHz, RAM 4.00 GB(3.86 GB usable) and 500 GB Hard disk while *Software* – Microsoft Windows 10 Enterprise © 2018, Microsoft Corporation, 64-bit operating system, x-64-based processor. In Figure 8, process size, burst time interval, and 10ms TQ for RR algorithm are taken as input from the user. The compute button helps in generating process

ID, their arrival time, and burst times, as well in executing the processes for zero and non-zero arrival times, while the clear button help in resetting the inputted data. The generated processes are moved to RQ, waiting to be assign CPU for execution. The TQ is determined with the burst times of the processes in the RQ. The processes are executed for a round, after which the active process remaining burst time and current TQ are always check to either allow the process to be reassign to CPU for execution and terminated from the system or place behind the processes in the RQ for next execution cycle. In each execution cycle new TQ is determine for subsequent execution cycle until the RQ is empty. The parameters such as ATT, AWT, ART and NCS are calculated and displayed as shown in Figures 10 and 13.

4.1 For Zero Arrival Time Processes

Figure 9 present the processes generated for Zero Arrival Time for the inputted 20 processes with lower and upper limits burst times of 1ms and 60ms.

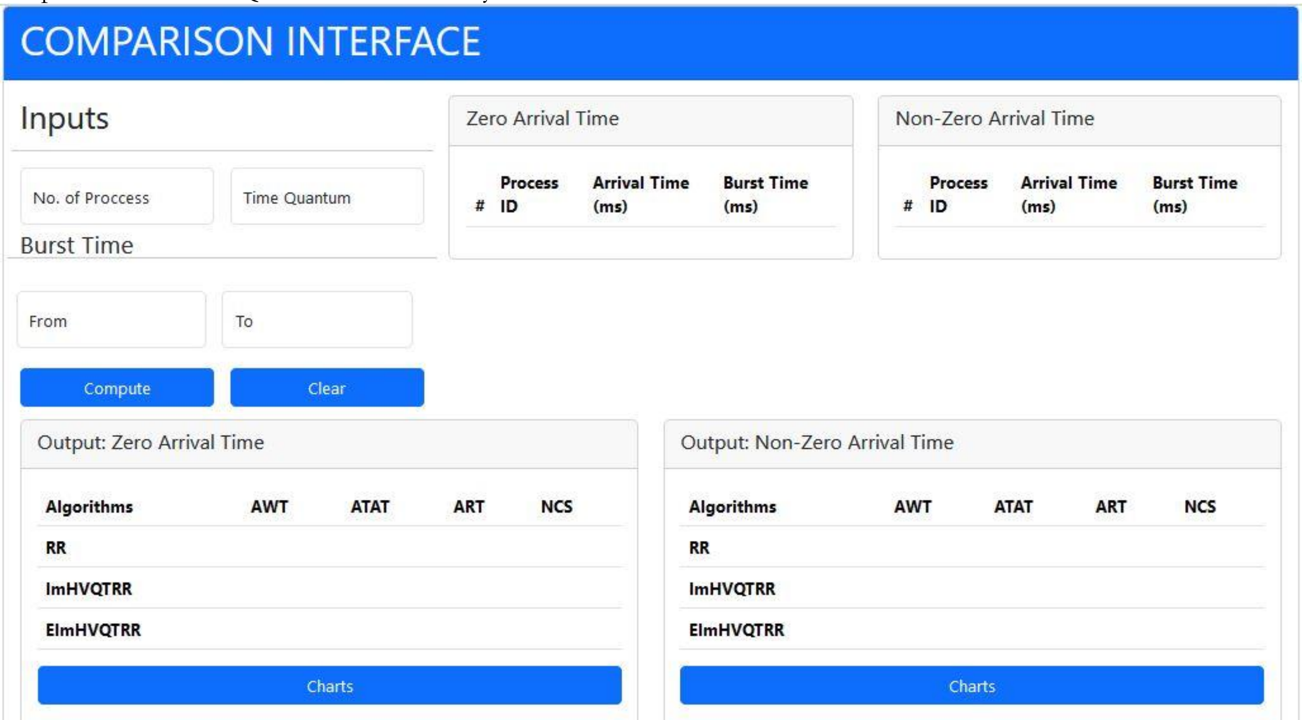
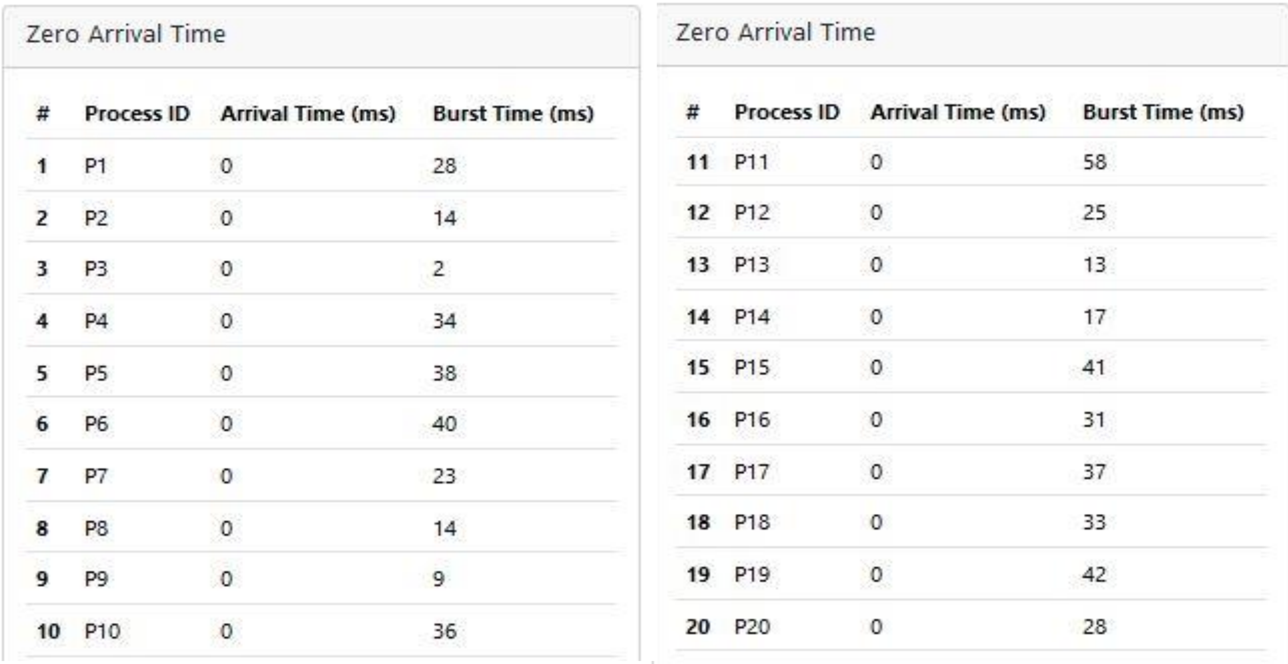


Figure 8: Processes Generation Interface



Zero Arrival Time

#	Process ID	Arrival Time (ms)	Burst Time (ms)
1	P1	0	28
2	P2	0	14
3	P3	0	2
4	P4	0	34
5	P5	0	38
6	P6	0	40
7	P7	0	23
8	P8	0	14
9	P9	0	9
10	P10	0	36

Zero Arrival Time

#	Process ID	Arrival Time (ms)	Burst Time (ms)
11	P11	0	58
12	P12	0	25
13	P13	0	13
14	P14	0	17
15	P15	0	41
16	P16	0	31
17	P17	0	37
18	P18	0	33
19	P19	0	42
20	P20	0	28

Figure 9: Processes Generated for Zero AT Case

Output: Zero Arrival Time

Algorithms	AWT	ATAT	ART	NCS
RR	371	399	88	65
ImHVQTRR	390	423	236	46
EImHVQTRR	371	399	71	91

Figure 10: Results of Zero AT Processes

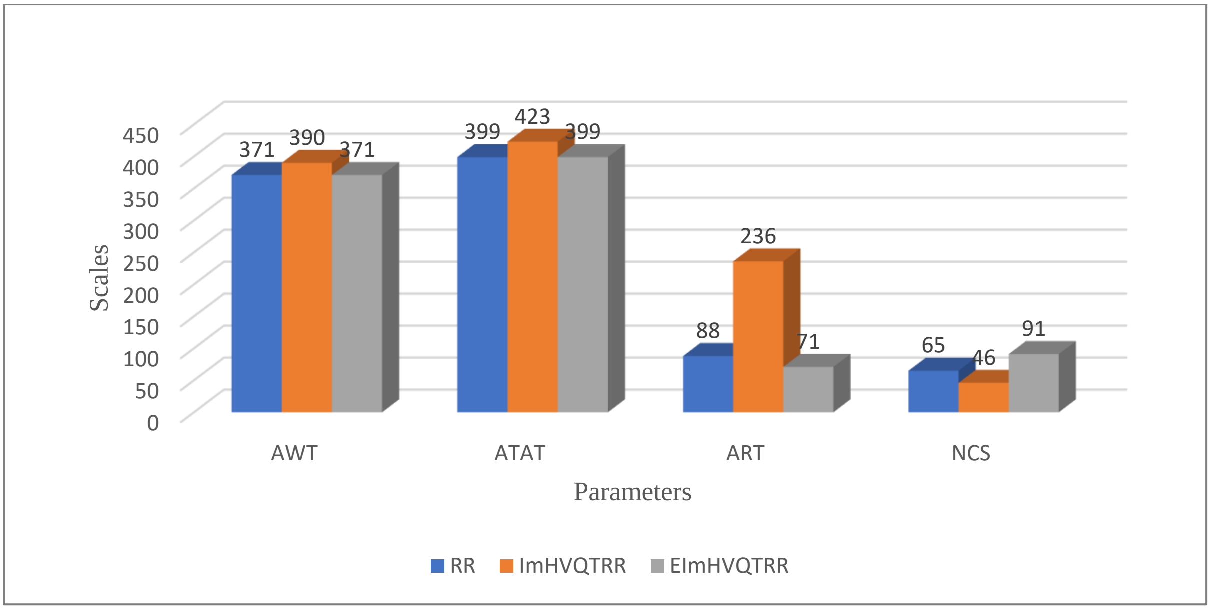


Figure 11: Zero AT Results Chart

Non-Zero Arrival Time

#	Process ID	Arrival Time (ms)	Burst Time (ms)
1	P1	0	21
2	P2	3	41
3	P3	6	32
4	P4	9	3
5	P5	12	27
6	P6	15	17
7	P7	18	25
8	P8	21	26
9	P9	24	19
10	P10	27	48

Non-Zero Arrival Time

#	Process ID	Arrival Time (ms)	Burst Time (ms)
11	P11	30	29
12	P12	33	33
13	P13	36	15
14	P14	39	6
15	P15	42	19
16	P16	45	27
17	P17	48	39
18	P18	51	5
19	P19	54	35
20	P20	57	25

Figure 12: Processes Generated for Non-Zero AT Case

Output: Non-Zero Arrival Time

Algorithms	AWT	ATAT	ART	NCS
RR	326	351	59	57
ImHVQTRR	350	378	153	45
EImHVQTRR	326	351	51	77

Figure 13: Results of Non-Zero AT Processes

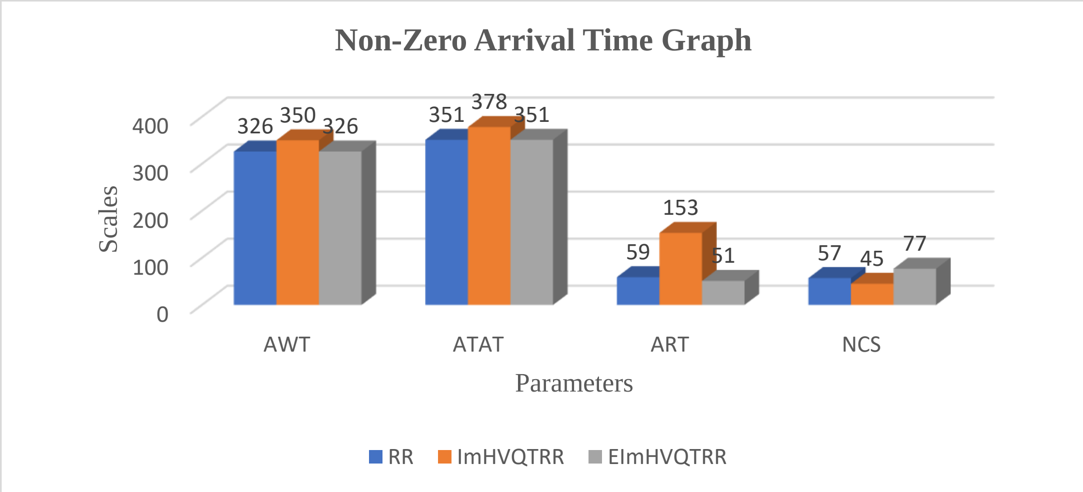


Figure 14: Non-Zero AT Results Chart

Figures 10 and 11 represent the outputs and graphical representations of the parameters obtained after executing the 20 processes. As indicated in Figure 10, the proposed algorithm (EImHVQTRR) performed better than the existing algorithms compared with in terms of average waiting time of 371ms as against 390ms and 371ms for ImHVQTRR and RR algorithms, average turnaround time of 399ms as against 423ms and 399ms for ImHVQTRR and RR algorithms, while the average response time of 71ms as against 236ms and 88ms for ImHVQTRR and RR algorithms with number of context switch of 91 as against 46 and 65 for ImHVQTRR and RR algorithms.

4.2 For Non-Zero Arrival Time Processes

Figure 12 presents the processes generated for Non-Zero Arrival Time for the inputted 20 processes with lower and upper limits burst times of 1ms and 60ms.

Figures 13 and 14 represent the outputs and graphical representations of the parameters obtained after executing the 20 processes. As indicated in Figure 13, the proposed algorithm (EImHVQTRR) performed better than the existing algorithms compared with in terms of average waiting time of 326ms as against 350ms and 326ms for ImHVQTRR and RR algorithms, average turnaround time of 351ms as against 378ms and 351ms for ImHVQTRR and RR algorithms, while average response time of 51ms as against 153ms and 59ms for ImHVQTRR and RR algorithms with high number of context switch of 77 as against 45 and 57 for ImHVQTRR and RR algorithms.

CONCLUSION

The most crucial part of computer is the Processor. CPU scheduling is an intelligent analysis of ready queue processes in determining the best way to respond to requests. Allot of CPU scheduling techniques were recommended, each with their advantages and disadvantages. In the light of the shortcomings experienced in the existing techniques, this algorithm employed dynamic TQ to mitigate starvation issue processes experienced in existing algorithms.

The findings of the simulation and experimental results indicated that EImHLVQTRR algorithm yielded better results than the existing algorithms (i.e., base-line and RR algorithms) in terms of AWT of 371ms as against 390ms and 371ms, ATAT of 399ms as against 423ms and 399ms, ART of 71ms as against 236ms and 88ms and NCS of 91 as against 46 and 65; AWT of 326ms as against 350ms and 326ms, ATAT of 351ms as against 378ms and 351ms, ART of 51ms as against 153ms and 59ms and NCS of 77 as against 45 and 57 in both Zero and Non-Zero Arrival Time simulation results for the processes generated as shown in Figure 10 & 13 respectively. The experimental results also show some significant improvement as the ATAT of 22.6ms as against 27.2ms and 29.6ms, AWT of 13.4ms as against 18.0ms and 20.4ms, ART of 4.8ms as against 9.0ms and 11.8ms with equal number of context switch for Zero Arrival Time; ATAT of 30.6ms as against 32.2ms and 40.6ms, AWT of 18.6ms as against 20.2ms and 28.6ms, ART of 18.6ms as against 10.6ms and 11.2ms and NCS of 4 as against 6 and 7 for Non-Zero Arrival Time experimental summary results shown in Table 3 & 4. The above results revealed that our proposed algorithm performance outweighs the two existing algorithms, and it's suitable in a real-time system for fair distribution of resources to multiple processes.

ACKNOWLEDGMENT

The researcher acknowledged the support of the Tertiary Education Trust Fund (TETFUND) for the success of the research grant. The researcher also acknowledged the support of the College TETFUND and Research Committees for making this possible. Thank you all.

REFERENCES

Abdelhafiz, A. A. (2021). VORR: A New Round Robin Scheduling Algorithm. Al-Azhar Bulletin of Science: Section B, 32(2), 1st of December, 2021, 45-54. **[Crossref]**

AbdelKader, A., Ghazy, N., Zaki, M. S. & ElDahshan, K. A. (2022). A Modified Mean-Median Round Robin Algorithm for Task Scheduling. *International Journals of Intelligent Engineering and Systems*, 15(6). **[Crossref]**.

Abubakar, M. B. (2016). Improved Round Robin with Highest Response Ratio Next (IRRHRRN) CPU Scheduling Algorithm [Unpublished master's thesis]. Mathematic Department, Faculty of Physical Science, Ahmadu Bello University, Zaria, Kaduna State.

Abubakar, S. E., Yusuf, S. A., Obiniyi, A. A., & Abdullahi, M. (2023). Modified Round Robin with Highest Response Ratio Next CPU Scheduling Algorithm using Dynamic Time Quantum. *Sule Lamido University Journal of Science and Technology (SLUJST).* 6(1&2), March, 2023, 87-99. **[Crossref]**.

Ali, K. F., Marikal, A. & PhD Kumar, K. A. (2020). A Hybrid Round Robin Scheduling Mechanism for Process Management. *International Journal of Computer Application*, 177(36), February, 2020. **[Crossref]**

Ashiru, S., Abdullahi, S., & Junaidu, S. (2014). Half Life Variable Quantum Time Round Robin (HLVQTRR). *International Journal of Computer Science and Information Technologies (IJCSIT)*, 5(6), 2014, 7210-7217. semanticscholar.org.

Fiad, A., Maaza, Z. & Bendoukha, H. (2020). Improved version of Round Robin Scheduling Algorithm based on Analytic Model. International Journal of Networked and Distributed Computing, 8(4), 195-202, 2020. **[Crossref]**

Hayatunnufus, Riasetiawan, M. & Ashari, A. (2020). Performance Analysis of FIFO and Round Robin Scheduling Process Algorithm in Internet of Things Operating System for Collecting Landslide Data. In: *Proc. of International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, 63-68. scribd.com .

Matarneh, R. J. (2009). Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes. *American Journal of Applied Sciences* 6(10), 1831-1837, 2009. **[Crossref]**.

Mishra, M. K., & Rashid, F. (2014). An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum. *International Journal of Computer Science, Engineering and Applications (IJCSEA),* 4(4), 2014, 1-8. **[Crossref]**

Mody, S. & Mirkar, S. (2019). Smart Round Robin CPU Scheduling Algorithm for Operating Systems. *4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT).* 1st December, 2019. **[Crossref]**

Mostafa, S. & Amano, H. (2020). Dynamic Round Robin CPU Scheduling Algorithm based on K-Means Clustering Technique. *Applied Sciences (Switzerland)*, 10(15), 5134, July, 2020. **[Crossref]**

Qazi, F., Agha, D., Naseem, M., Badar, S. & Hanif, F. (2023). Improved Round Robin Scheduling with Dynamic Time Quantum (IRRDQ). *Journal of Applied Engineering and Technology*, 7(2), 70-82. **[Crossref]**

Richardson, B. & Istiono, W. (2022). Comparison Analysis of Round Robin Algorithm with Highest Response Ratio Next Algorithm for Job Scheduling Problems. *International Journal of Open Information Technologies*, 10(2), 21-26, 2022. cyberleninka.ru

Sakshi, C., Sharma, S., Kautish, S., Alsallami, E., & Khalil, A. M., (2022). A New Median-Average Round Robin Scheduling Algorithm: An Optimal approach for Reducing Turnaround and Waiting Time. *Alexandria Engineering Journal*, 61(12), 10527-10538, 2022. **[Crossref]**

Sharma, A. & Kakhani, G. (2015). Analysis of Adaptive Round Robin Algorithm and Proposed Round Robin Remaining Time Algorithm. *International Journal of Computer Science and Mobile Computing*, 4(12), 139-147. ijcsmc.com

Simon, A., Dams, G. L., & Danjuma, S. (2022). An Improved Half Life Variable Quantum Time with Mean Time Slice Round Robin CPU Scheduling (ImHLVQTRR). *Science World Journal.* 17(2), 2022. researchgate.net.

Singh, A., Goyal, P., & Batra, S. (2010). An Optimized Round Robin Scheduling Algorithm for CPU Scheduling". *International Journal on Computer Science and Engineering (IJCSE)* 2(7), 2383-2385.

Sohrawordi, M., Ali, E., Uddin, P. & Hossain, M. (2019). A Modified Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum. *International Journal of Advanced Research,* 7, 422-429. **[Crossref]**.

Vayadande, K., Patil, S., Chauhan, S., Thakur, R., Baware, T., & Naik, S. (1st to 3rd, December, 2023). A Survey Paper on CPU Process Scheduling. International Conference on Recent and Future Trends in Smarts Electronic System and Manufacturing, pp. 40-47, 2023. riverpublishers.com

Zohora M.F, Farhin F, & Kaiser MS (2024). An enhanced round robin using dynamic time quantum for real-time asymmetric burst length processes in cloud computing environment. PLoS ONE 19(8): e0304517. **[Crossref]**