IMPROVED HALF LIFE VARIABLE QUANTUM TIME ROUND ROBIN CPU SCHEDULING ALGORITHM

By

FATIMA BUHARI RIDWAN

M.Sc/SCI/02396/2011-2012

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA.

APRIL, 2016

IMPROVED HALF LIFE VARIABLE QUANTUM TIME ROUND ROBIN CPU SCHEDULING ALGORITHM

By

FATIMA BUHARI RIDWAN B.TechMaths& Computer Science (FUT Minna) 2009

M.Sc/SCI/2396/2011-2012

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF A MASTER DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS
FACULTY OF SCIENCE
AHMADU BELLO UNIVERSITY, ZARIA
NIGERIA.

APRIL, 2016

DECLARATION

I declare thatthe work in this dissertation entitled **Improved Half Life Variable Quantum Time Round Robin CPU Scheduling Algorithm** has been carried out by me in the Department of Mathematics under the supervision of Dr. S.E. Abdullahi and Professor S.BJunaid. The information derived from the literature has been duly acknowledged in the text and the list of references is provided. No part of this dessertation was previously presented for another degree or diploma at any otherInstitution.

Fatima, Buhari RIDWAN

Name of student	Signature	Date

CERTIFICATION

This Dissertation entitled IMPROVED HALF LIFE VARIABLE QUANTUM TIME ROUND ROBIN CPU SCHEDULING ALGORITHMby FATIMA BUHARI RIDWANmeets the regulation governing the award of the degree of M.Sc Computer Science of Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

______________________________ (Signature): ______________________

Chairman, Supervisory Committee Date: ______________________

______________________________ (Signature): ______________________

Member, Supervisory Committee Date: ______________________

______________________________ (Signature): ______________________

Member, Supervisory Committee Date: ______________________

______________________________ (Signature): ______________________

Head of Department Date: ______________________

______________________________ (Signature): ______________________

Dean, School of Postgraduate Studies Date: ______________________

DEDICATION

Dedicated to my Late Father Sheikh BuhariRidwan and Late Brother MunirBuhariRidwan. May ALLAH (S.W.T) place them in the V.I.P section of AljannatulFirdausAmin.

ACKNOWLEDGEMENT

First and foremost I will like to express my gratitude to ALLAH AZZAH WA JALLAHfor his everlasting love, kindness, mercies, protection, and wisdom to sustain me throughout the period of this research work.

I wish to express appreciation to my supervisors Dr. S.E. Abdullahi (Major) and Professor S.B Junaidu(Minor) for sacrificing their time to be patient with me in the actualization and completion of this dissertation work despite their commitments.

I acknowledge the effort of the entire lecturers of the Department of Mathematics, Ahmadu Bello University Zaria, for their academic and intellectual support towards the success of my M.Sc.

Special acknowledgment to my Husband Alh.BilyaminuBalarabe and mybeloved children Safiyyah (Ihsan) Bilyaminu and Muhammad Buhari (Imam) Bilyaminu for thierunderstanding and support, I also appreciate the prayers of the entire family of Late BuhariRidwan in the cause of this program.

Lastly my gratitude goes to my colleagues andwell wishers whocontributedone way or the other towards the completion of this thesis, just to mention but few:AbdulrazaqAbdulrahim, HajaraIdris, Aisha AliyuShehu, Ashiru Simon, Ibrahim GamboAuta and the entire class members.

ABSTRACT

Process is an individualistic entity program. A program in execution requires good scheduling algorithm for its throughput and latency measures.Round Robin (RR) CPU scheduling algorithm has been designed chiefly for time sharing systems where processes compete for resource(s) especially the Central Processing Unit (CPU).The RR algorithm has proven to be more useful in multiprogramming environment as it is a fair scheduling algorithm that gives equal time quantum to all processes. This dissertation proposes an algorithm that enhanced the dynamic Half life Variable Quantum Time Round Robin (HLVQTRR) by Simon, (2014) as a key to improve the general performance of the system by reducing Average Waiting Time (AWT) and Average Turn-Around Time (ATAT), it also eliminated the cost of computing QT for burst times lower than average which will reduce the Number of Context Switching (NCS). The proposed algorithm was designed, evaluated and compared against the Half Life Variable Quantum Round Robin**(**HLVQTRR), Classical RR, Even Odd Round Robin (EORR), and Ascending Quantum Minimum Maximum Round Robin (AQMMRR) algorithms.From the results of the Analysis, it was observed that the proposed IHLVQTRR algorithm provides better solutions by up to 75% in terms of average waiting time (AWT), average turnaround time (ATAT) and number of context switches (NCS) than all the other compared algorithms.HLVQTRR produces better average response time than the proposed algorithm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

OS- Operating System

IO- Input Output

CPU-Central Processing Unit

QT- Quantum Time

FCFS- First Come First Serve

SJF- Shortest Job First

PS- Priority Scheduling

RR- Round Robin

EORR- Even Odd Round Robin

AQMMRR-Ascending Quantum Minimum Maximum Round Robin

HLVQTRR- Half Life Variable Quantum Time Round Robin

IHLVQTRR- Improved Half Life Variable Quantum Time Round Robin

BT- Burst Time

AWT- Average Waiting Time

ART- Average Response Time

ATAT- Average Turn Around Time

NCS- Number of Context Switching

μ- Mean

σ - Standard Deviation

λ - Parameter rate of Exponential Distribution

PDF - Probability Density Function

S.D – Standard Deviation

CHAPTER ONE

INTRODUCTION

1.1 Background of the study

A process is a collection of data structures that fully describes how far execution of the program has progressed; many processes are typically created in acomputing system. Each of these processes needs one or more system resources. But system resources are highly limited when compared with the number of processes that need the resources, so each process must compete for resource(s) especially the Central Processing Unit (CPU) which is the most important system resource. Now, how are resources allocated to these processes? Which process should wait for a resource and for how long? Which process should be assigned a resource (CPU)? These questions are answered by a technique used by the operating system called ***scheduling***.

Certain CPU scheduling techniques or algorithms, try as much as possible to be fair to all processes in order to optimize: CPU utilization, average waiting time and average turnaround time, number of context switching and throughput.One of the basic CPU scheduling algorithms is Round Robin (RR),its idea is to allocate equal time slice (quantum time) in a circular manner to processes in the ready queue (Simon *et al.* 2014). Peradventure the quantum time is greater or equal to the burst time of the process it will run to completion without being preempted. Otherwise, the process must be preempted after it must have exhausted its quantum time and then return to the tail of the ready queue to take turn again(Abdulrazaq*etal.* 2014). The beauty of RR is fairness in assigning CPU to all the processes in the ready queue because an equal time slice is given to each. Its greatest challenge is: what should be the quantum time? Having a small quantum time will increase the number of context switching, thereby reducing the general performance of the system. A larger one will practically degrade the system to First Come First Serve (FCFS) scheduling(Saeidi and Hakimeh,2012; Soraj and Roy, 2012). Just like any other CPU scheduling algorithm, RR has its peculiar features that make it unique. Each time RR is evaluated against Shortest Job First (SJF) and First Come First Serve (FCFS) algorithms, its average waiting time and average turnaround time is always higher. Although it is higher, yet it has gained more popularity and application in time sharing systems, and as such the most widely used CPU scheduling algorithm. This is to say that apart from average waiting time and average turnaround time, there are other factors that were considered for its wide acceptance. Among these factors were: multiprogramming, response time, fairness, sorting cost, priority and so on (Simon *et al.* 2014).The purpose of this research is to enhance the dynamic RR algorithm Half Life Variable Quantum Time Round Robin (HLVQTRR) bySimon(2014),that will provide answers to some of the setbacks experienced in the classical RR and the earlier proposed dynamic RRs. Comparative analysis will be carried out on five RR algorithms to enable this research work reach a meaningful conclusion.

1.2 Research Motivation

Half Life Variable Quantum Time Round Robin algorithm has limitation of poor context switching, average waiting time and turnaround time. An Improved Half Life Variable Quantum Time Round Robin algorithm was proposed to overcome these limitations. Also, Most of the reviewed papers used disadvantaged quantum time in RR to magnify their worksand as such affect the original intention of RR. Most of them incurred sorting cost, used priority on shorter jobs, no preemption in some of the RR algorithms, no any distribution function was used to generate data and response time was not evaluated. These disadvantages were avoided in the proposed IHLVQTRR.

1.3 Problem statement

Round Robin is considered the most widely used CPU scheduling algorithm. The performance of Round Robin Scheduling is sensitive to time quantum selection, because if time quantum is very large then Round Robin scheduling is same as the FCFS scheduling. If the time quantum is extremely too small then Round Robin scheduling is same as Processor Sharing algorithm.In the classical RR, quantum time is fixed throughout the scheduling process. This static nature made it very difficult to optimize the algorithm. In order to optimize RR CPU scheduling algorithm a dynamic QT must be found. This will provide a QT that change automatically so as to improve the general performance of the system by reducing average waiting time, average turnaround time and optimal number of context switches. In an attempt to answer the above questions, Half Life Variable Quantum Time RR (HLVQTRR) was proposed by (Simon*et al.*2014), where half of each process's burst time is executed in the first cycle and the process runs to completion in the second round. This means that in the second round which is also the final round, all the processes in the first round reappear. In this algorithm, thenumber of context switching is twice the number of the processes. This logically affects the average waiting time (AWT) and average turnaround time (ATAT). Based on this reason, an Improve Half Life Variable Quantum Time Round Robin (IHLVQTRR) was proposed to reducethe cost of computing quantum time for burst times lower than average thereby minimizing AWT and ATAT. The algorithm was compared side by side classical RR, Half LifeVariable Quantum Time Round Robin (HLVQTRR), Even Odd Round Robin

(EORR) and Ascending Quantum Minimum Maximum Round Robin (AQMMRR) algorithms using analytic and simulation techniques.

1.4 Aim and objectives

The aim of the dissertation is to propose an algorithm that will enhance the Half Life Variable Quantum Time Round Robin (HLVQTRR)algorithm by Simon (2014).

Theobjectives are to:

1. Design an algorithm aimedat overcomingthe poor context switching, average waiting time and average turnaround time of the HLVQTRR Algorithm
2. Evaluate the algorithm side-by-side HLVQTRR, classical RR, Even Odd Round Robin and Ascending Quantum Minimum Maximum Round Robin algorithms based on the following criteria's
    a. Average Waiting Time(AWT)
    b. Average Turnaround Time(ATAT)
    b. Average Response Time(ART)
    c. Number of context switching (NCS)

1.5 Research methodology

1. Review of literatures on dynamic Round Robin (RR) scheduling algorithm
2. Proposed IHLVQTRR to enhance the HLVQTRR algorithm by adding average of the processes burst times to decide which process run to completion
3. Simulate the proposed algorithm using Java programming language

1.6 Contribution to knowledge

This dissertationcontributed in optimizing the performance metrics of the HLVQTRRscheduling algorithm by reducing the number of context switch, average waiting time and average turnaround

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

Operating system is system software which interfaces between end users and computer hardware, so that the users can handle the system in a convenient manner. In this context, operating system can be seen as resource allocator. A computer system has many resources that may be required to solve a problem: CPU time, memory space, file storage space, I/O devices and so on. The operating system acts as a manager of these resources. Facing numerous and conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly (Silberchartz*et al*. 2005).

One of the common features of operating systems is the ability to be multiprogrammed. Multiprogramming requires several processes to be kept simultaneously in memory. Since in general, the main memory is too small to accommodate all processes, the processes are kept initially on the disk in the process pool. The jobs in the memory can be considered as a subset of tasks in the job pool. This pool consists of all processes residing on the disk, awaiting allocation of main memory. If several processes are ready to be brought into the memory, and if there is not enough room for all of them, the operating system must choose among them. The operating system further picks and allocates CPU to one of the processes for execution. Eventually, the jobs may have to wait for some task, such as an I/O operation, to complete.In multiprogramming systems, when there are more than one runnable process (i.e., ready), the operating system must decide which one to activate. The decision is made by the part of the operating system called the scheduler, using a scheduling algorithm (Suri and Sumit, 2012).

In a non-multiprogrammed (single processing) system, the CPU will sit idlewhile the process is waiting for the tasks such as an I/O operation, any other task must wait until the CPU is free and can be rescheduled. In contrast, the multiple programming systems which allow multiple programs to be loaded into memory and executed concurrently.

In a multiprogramming system, the operating system simply switches to and executes another job, when that job needs to wait, the CPU switches to another job and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to be executed, the CPU is never idle.

Time Sharing (or Multitasking) is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running (Silberchartz, *et. al*., 2005). This is why Round Robin CPU scheduling algorithm is considered the best for time sharing systems. In an attempt to be fair and as well as achieve multiprogramming, each process is given equal time slice (quantum time) to use the CPU one at a time. The quantum time determines: how long a process may use the CPU; how much the average waiting time and average turnaround time can be minimized; and the total number of context switching in an execution.

2.2 CPU-I/O Burst Cycle

In the computer system, all processes consist of a number of alternating two burst cycles (CPU burst cycle and Input and Output (IO) burst cycle) (Silberschatz*et al*, 2005). Normally, a process will run for a while (the CPU burst), performs some IO (the IO burst), then runs for a while more (the next CPU burst), again performs some IO (the IO burst). These cycles continue until the execution of the process is completed (Silberschatz*et al,* 2005).as shown in Figure 2.1. An IO bound process is a process that performs lots of IO operations such as reading from and writing to disks. Each IO operation is followed by a short CPU burst to process the IO, and then more IO happens. A CPU bound process is a process that performs lots of computation and does little IO. A typical system has a few long CPU bursts. One of the things a scheduler will typically do is switch the CPU to another process when one process does IO operations. This is because the IO usually takes a long time, and we don't want to leave the CPU idle while waiting for the IO to finish.

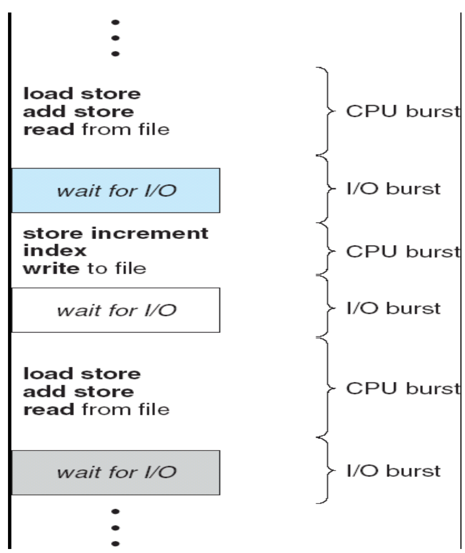


Figure 2.1: Diagram Illustrating Alternating Sequence of CPU and I/O Burst(Silberschatz*et al*, 2005).

2.3 Process State

A process is an instance of a program in execution. It is alsoa collection of data structures that fully describes how far the execution of the program has progressed.Processes are like human beings: they are generated, they have a more or less significant life, they optionally generate one or more child processes, and eventually they die. A small difference is that sex is not really common among processes – each process has just one parent (Bovet and Cesati, 2006). A program by itself is not a process; a program is a passive entity, such as a file containing a list of instructions stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction and set of associated resources. A program becomes a process when an executable file is loaded into memory. A process can be in the following states as shown in Figure 2.2:

a. ***New:*** when a new process is just created.

b. ***Ready:*** the process is ready to be assign a processor.

c. ***Running:*** the process is being executed.

d. ***Waiting:*** the process is waiting for some event to occur such as I/O event completion or reception of signal.

e. ***Terminated:***the process has finished execution.

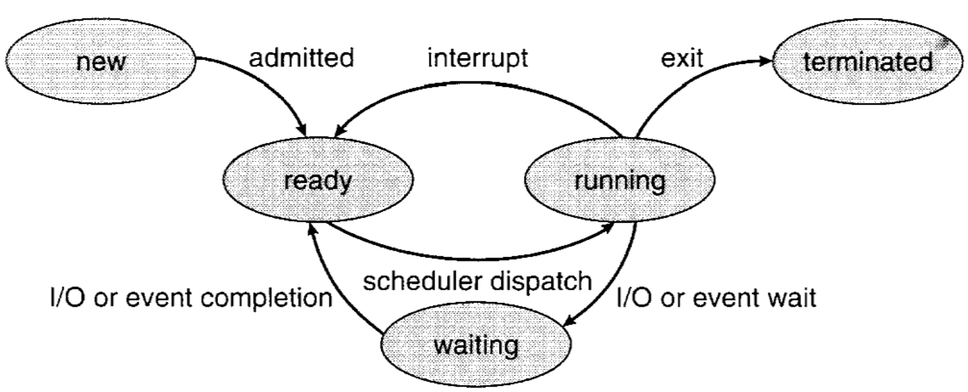


Figure 2.2: Diagram of process State (Silberchartz et al., 2005)

From the diagram above, once a new process is created, it will be admitted intothe ready queue. The scheduler will then dispatch the process in the ready queue for the processor. At this point, the process is said to be in a running state. The running process upon completion will exit thereby changing its state to terminate. Sometimes, a running process may be preempted, caused by an interrupt. This will force the running process to change its state to ready state which will be scheduled later. Also, a running process may change its state to waiting state because it is waiting for an I/O event to occur. Similarly, the same process in its waiting state may return to ready state upon completion of I/O event. It is important to note that only one process can be assigned a CPU at a time. However, many processes may be waiting and ready at the same time.

2.4 CPU Scheduling

Central Processing Unit (CPU) being the major resource is shared among competing processes using some form of technique known as scheduling (Abdulrazaq*et al.,* 2014).

CPU Scheduling:is the fundamental of operating system and also the basis of multiprogramming systems.Schedulingis the arrangement by which processes are given access to system resources (e.g. Processor cycles, communications bandwidth) by determining which process runs, when there are multiple runnable processes. Since single user or program cannot keep the CPU busy at all time jobs need to be organized so that the CPU will always have one to run. This important selection of jobs is done by a scheduler. **Scheduler**: is a module in the operating system that selects the ready processes from memory and allocates resources as per their requirement. Whenever one process waits for some other resource, scheduler selects the next process and allocates CPU to it. This process continues until the system requests for termination of execution and then the last CPU burst ends up with it.There are basically three types of processor scheduler, ***Long term scheduler*****(**Job Scheduler), ***Short term scheduler***(CPU scheduler) and***Medium time scheduler***.The long term scheduler selects jobs from the job pools into memory for execution, it determines which program is admitted to the systems for processing. Thus it controls the degree of multiprogramming (Pallab*et al*., 2012a). But the Short Term Scheduler selects processes from the memory and assigns them CPU for execution. Scheduling here is based on the requirement of the resources. It is essentially concerned with memory management and often designed as a memory management subsystem of an operating system (Bovet and Cesati, 2006).Some operating systems such as time sharing systems, may introduce an additional scheduling known as Medium time scheduler that temporarily removes a process from the main memory which is of low priority or has been inactive for a long time. The idea of Medium Term Scheduling is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued from where it is left off. Thisscheme is called ***swapping.*** The process is swapped out, and later swapped in, by the medium-term scheduler.

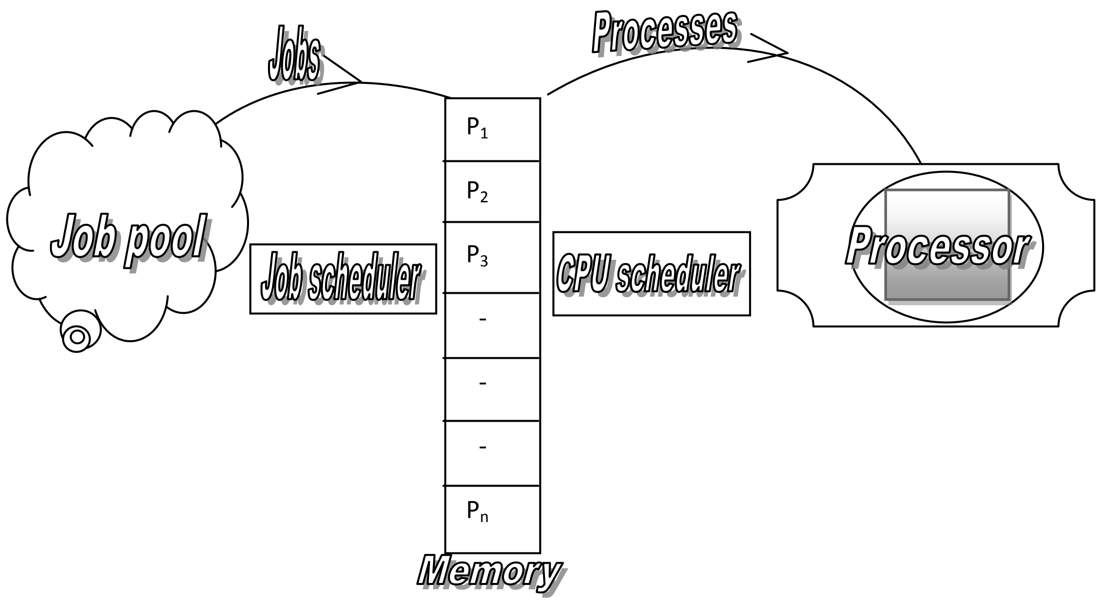


Figure 2.3:Abstraction of scheduling structure(Silberchartz*et al*, 2005).

2.4.1 Scheduling of Processes in the Ready Queue

A process in the ready queue may request for processor. If it does that, the CPU is allocated to it depending on the scheduling algorithm in use(Silberchartz*et al*, 2005). Looking at the Fig 2.3, the following cases may be associated with a given process.

a. The process may run to completion and terminate.
b. Process may request I/O operation. This will make the process to join the I/O queue and later be given the I/O device which eventually becomes ready for the processor again.
c. The time given to the process to use the CPU has expired. Process must now go back and join the ready queue.
d. The process forks a child. The created child process will be executed while the parent process may be suspended.
e. An interrupt occurs forcing the process to be preempted and join the ready queue.

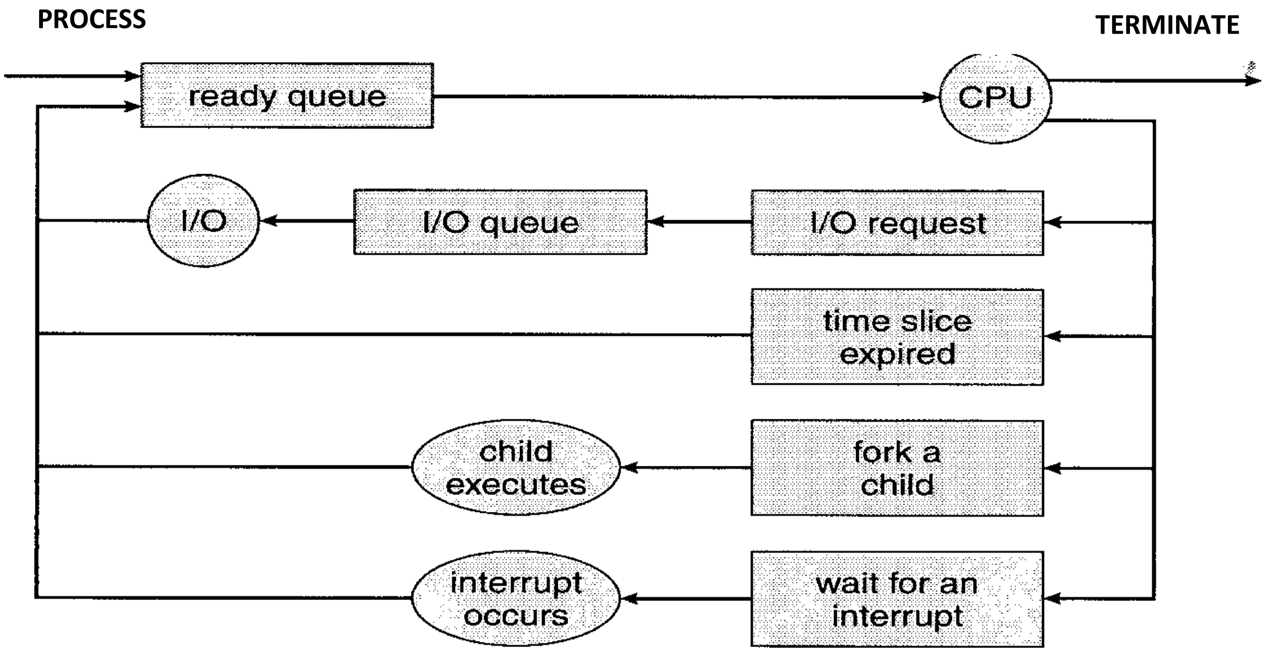


Figure 2.4: CPU scheduling operation in the ready queue(Silberchartz*et al*, 2005).

2.5 Preemptive and NonPreemptive Scheduling

In preemptive scheduling, a running process is forced to relinquish (suspend) a resource (CPU) even if it has not finish execution. This may be due to an interrupt, when a process switches from running state to ready state, or when a process switches from waiting state to ready state upon the completion of an I/O event. But as for nonpreemptive scheduling, a running process willingly relinquishes its resources (Ramakrishna and Pattabhi, 2013). This can only happen if the running process finishes execution and terminate, or the running process is waiting for an I/O event.

2.6 CPU Scheduling Algorithms

The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously). The basic idea is to keep the CPU busy as much as possible by executing a user process until it must wait for an event, and then switch to another process.

Scheduling Algorithm: Is a technique that the operating system uses to decide which of the process should be allocated the CPU and which one should wait, it also determines how long a process can use the CPU. Various scheduling algorithms have been developed each of these algorithms has their peculiarities. Among the various algorithms are: First Come First Serve (FCFS), Shortest Job First (SJF) Priority scheduling and Round Robin (RR) algorithms. These four algorithms are either preemptive or nonpreemptive except for Round Robin which ispreemptive in nature.

2.6.1 First Come First Serve (FCFS)

The idea is very simple. Allocate the CPU to the first process that arrives in the ready queue and made a request to use the CPU, then next process and so on. In this algorithm, the arrival time of a process is very important. The implementation is easily managed with a First in First out (FIFO) queue. The average waiting time for FCFS algorithm is generally not minimal and may be worst if processes' burst time vary considerably and larger job appears to arrive earlier than smaller ones. This will lead to lower CPU utilization and increase the average waiting and turnaround time than it will be when shorter processes are allowed to go first.

2.6.2 Shortest Job First (SJF)

Process with the smallest CPU burst time is allocated the CPU first. This technique associate with each process the length of the next CPU burst. When the CPU is available the Algorithm assigns the CPU to the process with the smallest next CPU burst. If the next CPU burst time of two or more processes is the same, FCFS scheduling is used to break the tie. SJF can be considered optimal because it gives minimal average waiting time and turnaround time. Moving a shorter process before the longer ones decreases the waiting time of the shorter process than it increases the waiting time of the larger process(Samih*et al.*, 2010). The major challenge of SJF is to know the process length with the next CPU request. Although, SJF is optimal, it has not been implemented in short term scheduler but only in the long term scheduler because the length of the next CPU burst cannot be accurately known, it is always estimated.A low value may mean faster response while too lower value will cause a time-limit-exceeded error and will require resubmission.

2.6.3 Priority Scheduling (PS)

The SJF algorithm is a special case of the general priority scheduling in which the priority is based on shorter jobs. The larger the CPU burst time, the lower the priority of that process and vice versa. The PS algorithm associates with each process a priority and the CPU is allocated to process with the highest priority. Usually, lower numbers are used to represent highest priority but this is not a rule as higher numbers can as well be used. For effective utilization of CPU, Priorities to various processes can be set by a study of type and time of processes during a day. For example there can be more number of LOGON processes during the day time in an organization and more backup jobs during evening. So, logon processes should be given top priority in the morning hours, where as backup jobs should be given higher priority than other jobs during closing hours. The process with the highest priority is allocated first. If there are multiple processes with same priority, typically the FCFS is used to break tie (Silberchartz, *et. al*., 2005). The major challenge of priority scheduling is indefinite blocking, or starvation. A process that is ready to run but waiting for the CPU may be considered blocked. Some processes of higher priorities may continue to have the CPU at the expense of processes with lower priorities leading to starvation. One of the solutions to indefinite blocking is aging. Aging is a technique that gradually increases the priority of processes that wait in the system for a long time.

2.6.4 Round Robin (RR)

Round Robin is the simplest and most widely used proportional share scheduling algorithm, it is designed for time sharing system to give better response but the worst turnaround and waiting time due to the fixed quantum time concept (Adeebaand Aiman 2012). In RR, a small unit of time (called quantum or time slice) is defined. A quantum time is generally from 10-100 milliseconds (Silberchartz*et al.*, 2005). The quantum time is equally given to every process in the ready queue in a circular manner. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 quantum time. If the quantum time is greater or equal to the CPU burst time of that process it will run to completion without being preempted. Otherwise, the process must be preempted after it must have exhausted its quantum time and then return to the tail of the ready queue to take turn again. Without Round Robin scheduling, the process allocated to the CPU first would likely monopolize the processor until it finished performing its tasks. When it finished, the next process could start without interruption. This would get in the way of the multi-window environments on which computer users have come to depend on. The use of RR scheduling helps the computer keep up with the end user and effectively manage other application processes. The preemptive nature of RR scheduling can only suggest that it is designed to handle time sharing systems. As bad as context switching may be it is highly needed in time sharing systems, although as earlier mention, too much context switching is an overhead because it reduces the efficiency of the CPU. Another issue is what should be the quantum time? This is even more serious because smaller quantum time will increase the number of context switching, and a larger value will automatically change the system to ordinary FCFS causing a poor response time.For example if the scheduler selects the CPU-bound process to run, the other process could wait for a whole quantum time before starting its execution. Therefore, if such duration is long, the system could appear to be unresponsive to the user that launched it (Bovet and Cesati 2006). Therefore, an optimal quantum time is needed.

2.7 Static RR versus Dynamic RR Scheduling Technique

In ***Classical RR*** the quantum time (QT) is static. This is why it is sometimes called static RR. Here, the QT is gotten from the average of the processes' burst times in the ready queue and remains constant throughout the entire procedure. The static nature made it 0odifficult for improvement, and as such research on Dynamic RR is on.

The idea of ***Dynamic RR*** is to use more than one different QT for processes in the ready queue depending on certain criteria. That is to say it determines time quantum during run time. Dynamic RR has proven to be better as compared with the classical RR in terms of CPU utilization by reducing the average waiting time, average turnaround time and number of context switch.

2.8 Some Major Challenges of Dynamic RR Algorithms

There were many problems associated with the dynamic RR among which are:

First is jerking up of quantum time above average. This cannot be considered as a solution to the practical Round Robin instead, in most cases a disadvantage. Most of the dynamic RRs reviewed jerked up quantum time above the average of the processes in the ready queue. Continuous jerking up quantum time just to improve average waiting time and average turnaround time shall eventually change the algorithm to FCFS and as such suffer greatly in multiprogramming and response time.

Secondly, some of the proposed algorithms incurred the cost of sorting, as they sorted processes twice or more before performing scheduling. Sorting on its own is an overhead that should be avoided.

One of the wrong and worst approaches used is giving priority to shorter jobs; this is a serious disadvantage and should be avoided. If jobs are to be considered on shorter job bases, then, the algorithm is indirectly SJF algorithm in disguise.

2.9 Related Works

In the last few years there have been several research effort on dynamic round robin (Abbas, *et al.*, 2011;Behera*et al*., 2011;Saroj*et al.*, 2011; Adeeba*et al.*, 2012;Ali, 2012; Behera*et al.*,2012;Pallab*et al.*, 2012a;Pallab*et al.*, 2012b; Pallab*et al.*, 2012c;Dibyendu, 2013;Yashasvini, 2013 and Simon, 2014). All of these dynamic RR algorithms were proposed in order to enhance the average waiting time, average turnaround time and the number of context switches of the classicalround robin. Each of these algorithms has proven to be more efficient than the classical RR.We review these algorithms briefly before presenting our proposed algorithm that attempts to enhance the algorithm proposed by (Simon *et al.*,2014).

2.9.1 Dynamic Quantum Using the Mean Average

This algorithm proposed by Abbas *et al.*, (2011)is aimed at optimizing the performance metrics (AWT, ATAT and NCS) of the classical round robin.

Methodology: Take the average of the processes CPU burst at each cycle and use it as quantum time of that cycle. When it goes round again, take another average using the remaining processes CPU burst in the ready queue then use it as the quantum and so on Abbas *et al*., 2011). This method needs two registers to be identified:

a. **SR:** Register to store the sum of the remaining burst time in the ready queue.
b. **AR:** Register to store the average of the burst time by dividing the value found in the SR by the count of processes found in the ready queue.

When a process in execution finishes its quantum time, the ready queue and the register will be updated to store the new data values. In this algorithm, arrival time is considered to be zero.

Contribution: Shows better performance compared to Round Robin by decreasing the total turnaround time, average waiting time and number of context switches.

Limitation: This approach tries to jerk up the quantum time just above the average; this is an attempt to capture those processes that were not captured in the first round. If the quantum time is continuously been jerked up, a time will reach when the algorithm will become FCFS.

2.9.2 Multi-Dynamic Quantum Time Round Robin (MDQTRR) Algorithm with Arrival Time

Behera*et al*., (2011) proposed a dynamic RR known as MDQTRR to improve on the number of context switches in the classical RR.

Methodology: MDQTRR uses two different quantum times in single cycle. Up to the median process, the quantum time used is gotten using the median quartile formula MQT (Median Quantum Time) while for the succeeding processes, the Upper Quartile formula is use to calculate the quantum time, UQT (Upper Quartile Time). The procedure is repeated for the next cycle using same procedure.

Limitation: In this algorithm the waiting and turnaround time was optimized but the number of context switches was more than that of the classical RR.

2.9.3 Adaptive Round Robin Scheduling Using Shortest Burst Approach Based on Smart Time Slice

Methodology: Saroj*et al.*,(2011) proposed this algorithm, it depends on three aspects: priority, average CPU burst and mid process CPU burst (Saroj*et al.*, 2011). Processes are allocated to the processor base on their priorities. Higher are allocated the processor first and then follow by the process which has the next priority and so on. But as for the quantum time, it will be based on the current running processes. If the number of the process in the ready queue is odd the time slice will be the mid process burst time otherwise, it will be the average of all processes burst time in the ready queue.

Limitation: The algorithm introduced priority on the shorter burst times, thereby causing starvation of higher burst times. This changes the earlier intention of the traditional RR.

2.9.4 A Varied Round Robin Approach Using Harmonic Means of the Remainning Burst Time of the Processes

Methodology: Adeeba*et al.,* (2012) proposed this algorithm where a static quantum is used to compare each process burst time, if it is less than or equal to the quantum time then conventional RR is used on the process, and if the burst time is greater than the quantum time then left over time is found and used as a comparison parameter with another constant called Harmonic Mean (Hm) whose value has been calculated using the remainder of the quantity Bt/QT (Adeeba and Alman, 2012). The left over time is compared with $2/5^{th}$ of the quantum time and if it is smaller, it will be added to the quantum time.

a. Else the left over time is compared with the harmonic mean, if smaller, the harmonic mean value is added to the quantum time and the job is executed.

b. If the above stated conditions turn out to be false then the conventional Round Robin is applied on the process.

In all cases the value of the quantum time changes only for the particular process for which the above stated condition satisfies and remains same elsewhere.

Hm=harmonic mean=ceil(N/ [Σ {1/Lot[Pi] }])
Number of cycles (Noc) = Bt [Pi] / QT
Left over time (Lot) = Bt- (Noc * QT)
Bt= Burst time
QT=quantum time
N=number of processes in the ready queue.
Pi=i^{th}process

Limitation: This algorithm is complex;a lot of time was wasted testing processes. Use of conventional RR for burst times less or equal to the quantum time wasn't an improvement.

2.9.5 Ascending Quantum Minimum Maximum Round Robin (AQMMRR)

The AQMMRR algorithm was proposed by Ali, (2012) to reduce the number of context switching, averages waiting time and turnaround time in the traditional RR.

Methodology: Processes and their associated burst times are rearranged in ascending order, quantum time is calculated by multiplying summation of the minimum and maximum CPU burst with 80 percent. The procedure is repeated in each round. The 80 percent is chosen depending on two reasons: First, if the quantum time calculated depends only on the summation the algorithm will become SJF. Secondly, the rule of thumb is that 80 percent of the CPU bursts should be shorter than the quantum time (Ali, 2012). Quantum time = (MinBurst + MaxBurst) * 80%.

Contribution: This algorithm shows better performance compared to Round Robin Scheduling algorithm by decreasing the total turnaround time, average waiting time and number of context switching.

Limitation: This algorithm worked exactly as Shortest Job First (SJF) there is no Round Robin associated to it because the quantum time is greater than the maximum process CPU burst in the ready queue causing the AQMMRR to have no context switching. That is to say none of the processes went beyond first round. In any given RR scheduling at least some processes are expected to be switched to second round.

2.9.6 Precedence Based Round Robin with Dynamic Quantum Time (PRRDQT) Scheduling Algorithm for Soft Real Time System

Behera*et al.*, (2012) proposed this algorithm in other to minimize the waiting time and turnaround time of the traditional round robin.

Methodology: it calculates a new factor, called Precedence Factor (PFi) by combining the basic factors of the process priority and burst time (Behera and Bra`jendra, 2012).

The formula is written bellow:

PFi = (Priority * Priority-ratio) + (burst Time * burst time-ratio) (2.1)

Where:Burst Ratio = burst time of the process (BTi) / summation of burst time of the processes. (2.2)

Priority Ratio = Priority of the process (Pi) / total no of processes. (2.3)

Processes are assigned precedence according to their factor. The process having lesser factor gets higher precedence. Processes are then sorted according to the new precedence and RR is applied with dynamic quantum time. The quantum is gotten by taking average of the processes burst time. Processes with shorter burst time and higher priority is executed first resulting in better turnaround time and better waiting time.

Limitation: the algorithm changes the idea of traditional RR by sorting and use of priority on the processes with smaller burst time, thereby caused starvation on processes with higher burst times.

2.9.7Even Odd Round Robin Scheduling Algorithm (EORR)

Methodology: Pallab*et al.*,(2012a) proposed an algorithm that calculates double quantum times,` QT1 and QT2 where QT1 is the average of all processes CPU bursts at even places in the ready queue and QT2 is the average of processes CPU bursts at odd places in the ready queue. Then, compare QT1 with QT2 and take greater value to be the quantum time of that cycle. That is,

If QT1>QT2
Then QT_{new}=QT1
Else QT_{new}=QT2 (2.4)

In case of next round, same procedure is followed to calculate a different quantum time for the remaining processes in the ready queue and so on.

Contribution: this algorithm shows better performance compared to classical Round Robin by decreasing the total turnaround time, waiting time, response time and number of context switches.

Limitation: Double quantum time was calculated in each round, hence reduced the efficiency of the algorithm.

2.9.8Mid Average Round Robin Scheduling Algorithm (MARR) Using Dynamic Quantum Time

Methodology: Pallab*et al.*, (2012b) proposed the MARR algorithm where processes are sorted in ascending order. Quantum time was gotten by taking the average of the processes from the middle process to the maximum process. The process burst that is next greater than or equal to the calculated middle value is taken as the middle burst.

The procedure is as follows:

MiddleBurst = (LowestBurst+HighestBurst) / 2 (2.5)

QuantumTime = Average from the (Middle burst to the Highest burst). (2.6)

In the case of incomplete execution of some processes the same procedure was followed to get the quantum time for the next round.

Contribution: MARR algorithm performs better than the classical RR**.** Itdecreased the total turnaround time, waiting time and number of context switches.

Limitation: This algorithm uses the idea of sorting processes which violated the intention of the traditional RR.

2.9.9Average Mid Max Round Robin (AMMRR) Scheduling Algorithm

Pallab*et al.,* (2012c) presented the AMMRR algorithm in order to optimize the performance metrics of the classical RR.

Methodology: In this algorithm the value of the quantum time is dynamically adjusted as process execution progresses by taking the mean of summation of the average and the maximum burst time (Pallab*et al.,* 2012c). This approach assumes the arrival time to be zero for all the processes, and it has proven to be efficient.

AVG = (summation of Burst of all the processes)/number of processes (2.7)

Quantum time = (AVG+MaximumBurst)/2. (2.8)

Limitation: It incurred additional cost of computing quantum time in each cycle, that is to say, after each round the same procedure is followed in obtaining a new quantum time for the next round.

2.9.10 Dynamic Quantum Time Round Robin Algorithm (DQTRR) Depending on Burst and Arrival Time of the Processes

Dibyendu (2013) proposed a dynamic RR CPU scheduling algorithm DQTRR to overcome the limitations of the classical RR.

Methodology: The idea is to dynamically adjust the quantum time depending on the arrival time and burst time of the processes (Dibyendu, 2013). It has two cases.

Case1: if the arrival time of all the processes in the ready queue is zero, the quantum time is given as the average of the processes CPU burst time. Else case 2

Case2: if arrival times of all the processes are not zero then the quantum time will change dynamically.

Limitation: Case 1 is the same as the classical RR, case 2 incurred cost of computing Quantum time.

2.9.11Determination of Variable Quantum Time (VQT)

Methodology: Yashasvini, (2013) presented the VQT algorithm that suggested every process in the ready queue should have a different quantum time in each cycle so as to maximize multiprogramming.

It contains four structures namely S1, S2, S3 and S4.

S2: Incoming processes are kept in S2.

S3: This is used for ordering them in a particular manner and finding their quantum time.

S4: This is used for again arranging all processes in their correct manner, but instead of adding burst time, it will store processes quantum time and this will be used to execute processes in S2. In between this, other processes will start entering in S1, which after execution all processes in S2, will be transferred to S1. The first quantum time was the average of the processes burst time (since it is arranged in descending order, it will be the first process) which will be given to the first highest process. The next quantum was the next highest burst time gotten from the average of the processes burst time excluding the process that has just gotten its quantum. This will continue to the last process which will then be used to perform RR, it can be written as:

Burst {B1,B2,B3,---,Bn} Associated processes {P1,P2,P3,---,Pn}.

Processes are arranged in descending order (P1 being the highest). That is,

P1= average of (B1,B2,B3,…,Bn) will be its quantum time
P2= average of (B2,B3,…,Bn) will be its quantum time
P3= average of (B3,…,Bn) will be its quantum time
Pn= (Bn) will be its quantum time

In the next cycle, different quantum time will be used because the processes CPU bursts must have changed, or some process must have run to completion. Same procedure will be performed to get the quantum time for the remaining processes that do not run to completion. Here, arrival time is not considered to be zero

Limitation: this algorithm may suffer from number of contexts switching, average waiting time and average turnaround time. It also incurred sorting and time quantum computational costs.

2.9.12 Half Life Variable Quantum Time Round Robin (HLVQTRR)Algorithm

The HLVQTRR algorithm was proposed by Simon (2014) to address the problem of poor response time and multiprogramming associated with the earlier proposed dynamic RR algorithms in order to improve the general performance of the system.

Methodology: Half of every process's burst time is executed in a first round, the process run to completion in the second round.

Contribution: A process must not finish in its first round and must not go for more than two rounds. This improves response time.

Limitation: Context switching is twice the number of the processes since all the processes re-appear in the second cycle. This logically affects the AWT and ATAT.

2.10 Project Concern

All the above works attempted to optimize RR using dynamic quantum time; their target was to optimize the three parameters: AWT, ATAT and NCS. Although, there are instances whereby some of the approaches optimized one or two at the expense of other(s). In all cases, they outperformed the classical RR. In an attempt to minimize average waiting time and average turnaround time one must be very careful not to keep some processes waiting for so long and as such affect the original intention of RR i.e. it may become SJF, FCFS, or gives poor response time to other processes by not being fair enough.Also, apart from AWT, ATAT and NCS used by the proposed algorithms, there were other important criteria which were not taken into consideration as some of the reviewed worksdid not pay attention tothe simplicity or complexity of the proposed algorithms. This should also be a concern so as not to incur too much quantum time computational overhead. Also most of the algorithms were jerking up quantum time just above the average of processes bursts in the ready queue; this is really not a good practice as those algorithms degenerated to FCFS. As attention is gradually shifting to dynamic RR CPU scheduling algorithm theImproved Half Life Variable Quantum Time Round Robin (IHLVQTRR) algorithm was proposed with aim of improving on the performance metrics of the HLVQTRR algorithm since processes with smaller burst times less than average complete their execution in first round and processes with burst time values greater than average are roughly partitioned intotwo rounds. Whatever the case may be, it will demonstrate high rate of multiprogramming and response time. The proposed algorithm was evaluated and compared side by side the classical RR, HLVQTRR, EORR and AQMMRR algorithms.

CHAPTER THREE

DESIGN OF THE PROPOSED IHLVQTRR ALGORITHM

3.1 Introduction

Improve Half Life Variable Quantum Time Round Robin (IHLVQTRR) CPU scheduling algorithm is an enhancement ofthe Half Life Variable Quantum Time Round Robin (HLVQTRR)CPU scheduling algorithm ofSimon (2014)by finding average of the processes burst times to decide which process run to completion.

a. Processes run only for two cycles.
b. Average is gotten and compared with each process
c. A process with burst time less than average i.e. (BT ≤ average, QT = BT)must finish execution in the first round.
d. If a process burst time is greater than average i.e. (BT> Average, QT = BT/2) half of its burst time is executed in first round and the remaining half run to completion in the second round.
e. The proposed IHLVQTRR algorithm was simulated using java programming language.
f. Proposed IHLVQTRR algorithm was evaluated and compared side by side the HLVQTRR, classical RR, EORR and AQMMRR algorithms.

3.2 IHLVQTRR Algorithm

```
//N= Number of processes
//Pi= i^th Process
// PiQT = Time quantum for i^th process
//i= Loop variable
//BT= Burst time of the processes
//Avg= Average
//sum=0
// RQ= Ready Queue
While(RQ !=NULL){
     For i=1 to N Loop{
sum = sum +BTi
       }
 End of  For
Avg = Sum/N
// Assign process to cpu
     For i=1 to N loop{
If (PiBTi<= Avg){
     Run Pi to completion
     }
Else{
     Run Pi for period of
          Pi =PiBT/2
}End of If
}End of for
    } End of while
Calculate AWT, ATAT and CS
//AWT=Average waiting time
//ATAT=Average turnaround time
//ART=Average response time
//NCS= Number of context switch
End
```

3.3 IHLVQTRR Flow Chart

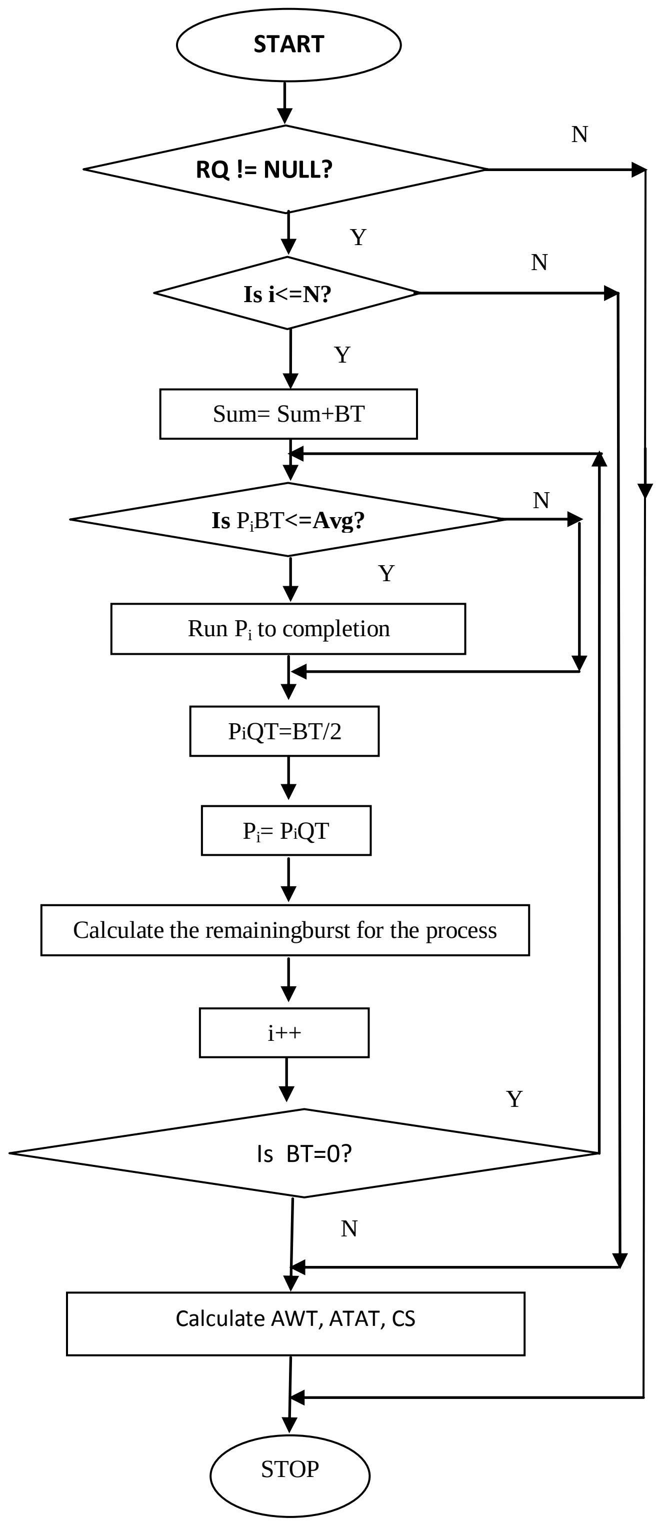


Figure 3.1: Flowchart of the IHLVQTRR Algorithm

3.4 Illustrative Example

Given the following set of processes by (Simon *et al.*, 2014) algorithm with their associated burst times the IHLVQTRR, HLVQTRR, EORR, AQMMRR and classical RR algorithms were compared based on the criteria below.

Table 3.1: Processes' CPU burst

PROCESSES	CPU BURST
P1	88
P2	89
P3	85
P4	93
P5	90
P6	84
P7	90

3.4.1 Performance Metrics

The proposed IHLVQTRR side by side HLVQTRR, classical RR, EORR and AQMMRR were evaluated base on the following schedulingcriteria.

a. Turnaround Time(TAT). TAT=Finish Time –Arrival Time

Average Turnaround Time should be less.

b. Waiting Time(WT). WT= Start Time- Arrival Time

Average Waiting Time should be less.

c. Response Time (RT). RT = Time it takes each process to starts responding

Average Response Time should be less.

d. Context Switch. CS = Occurred from the processes that goes beyond first round

The number of context Switch should be less.

3.4.2 Classical Round Robin

P1=88, P2=89, P3=85, P4=93, P5=90, P6=84, P7=90

In a classical RR the quantum time is the average of processes CPU burst times in the ready queue. That is,

Quantum time (QT) = (88+89+85+93+90+84+90)/7 =619/7 = 88

Remaining CPU burst after round 1

P2=1, P4= 5, P5=2, P7=2

Gantt chart

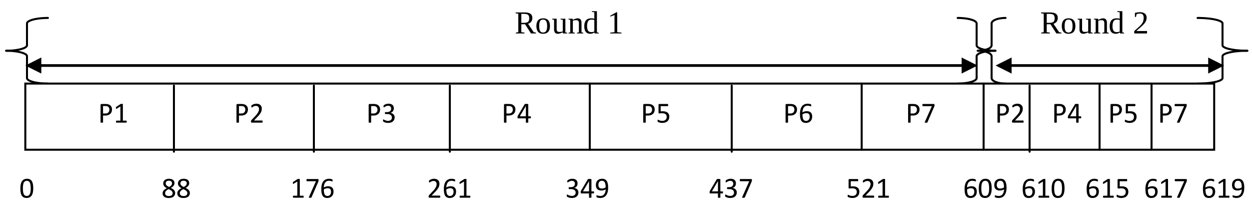


Figure 3.2: Classical RR Gantt chart

Table 3.2: Classical RR Analytic Result Table

<u>**Waiting Time:**</u>	<u>**Turnaround Time**</u> TAT=WT+CPU burst	<u>**Response Time**</u> In first round only
P1: 0 P2: 88+433=521 P3: 176 P4: 261+ 261= 522 P5: 349 + 178 = 527 P6: 437 P7: 521 + 8 = 529	P1: 0+88=88 P2: 521+89=610 P3: 176+85 =261 P4: 522+93= 615 P5: 527+90 = 617 P6: 84+437 = 521 P7: 529+90 = 619	P1: 0 P2: 88 P3: 176 P4: 261 P5: 349 P6: 437 P7: 521
AWT = 387.43, ATAT = 475.86, No of context switching = 4, Average Response Time=261.71		

3.4.3 Even Odd Round Robin CPU Scheduling Algorithm (EORR) Using Dynamic Quantum Time

There are two quantum times (QT), QT1 and QT2. QT1 is the average of processes CPU bursts at odd positions in the ready queue while QT2 is the average of processes CPU bursts at even positions in the ready queue. QT1 is compared with QT2, and the greater is taken to be the QT at that round.

Using the same processes: P1=88, P2=89, P3=85, P4=93, P5=90, P6=84 and P7=90 to performs the evaluation.

Method

Round 1
P1, P3, P5 and P7 are at odd position and P2, P4 and P6 are at even positions in the ready queue. So, QT1 = (88+85+90+90)/4 =353/4 =88.25 ≈88

P2, P4, P5 and P6 are at even positionsQT2 = (89+93+84)/3 =266/3 = 88.66 ≈89

Since QT2>QT1, QT2 is taken to be the quantum time. i.e. Quantum time (QT) =89

Applying Round Robin,

P1, P2, P3 and P6 ran to completion but P4, P5 and P7 were preempted for round 2.

Remaining CPU burst after round 1

P4=4, P5= 1, P7=1

Round 2

QT1= (4+1)/2= 2.5≈ 3 and QT2 = 1

Since QT1>QT2, quantum time is taken as 3 i.e.Quantum time (QT) = 3

Applying Round Robin, P5 and P7 ran to completion but P4 was preempted for round 3.

Round 3

P4 is the only remaining process with burst time of 1. This will be our quantum time. That is, QT=1.

P4 will run to completion with quantum time = 1.

Gantt chart

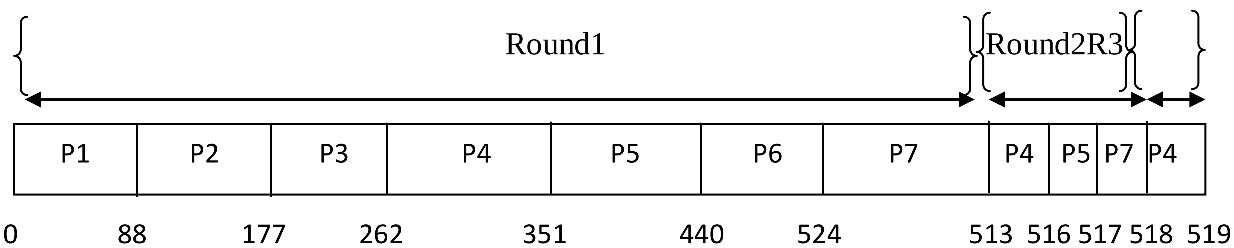


Figure 3.3: EORR Gantt chart

Table 3.3: EORR Analytic Result Table

Waiting Time:	**Turnaround Time** TAT = WT + CPU burst	**Response Time** In first round only
P1: 0 P2: 88 P3: 177 P4: 262 +162 +2 = 428 P5: 351 + 76 = 427 P6: 440 P7: 524 + 4 = 528	P1: 0 + 88 = 88 P2: 88 + 89 = 177 P3: 177 + 85 = 262 P4: 428 + 93 = 521 P5: 427 + 90 = 517 P6: 440 + 84 = 524 P7: 528 + 90 = 618	P1: 0 P2: 88 P3: 177 P4: 262 P5: 351 P6: 440 P7: 524
AWT ≈ 298.29, ATAT ≈ 386.71, No of context switching= 4, Average Response Time=263.14		

3.4.4 Ascending Quantum Minimum Maximum Round Robin (AQMMRR)

Incoming processes and their associated burst times are rearranged in ascending order, quantum time is calculated by multiplying the summation of the minimum and maximum CPU burst by 80 percent.

<u>Method</u>

P1=88, P2=89, P3=85, P4=93, P5=90, P6=84, P7=90

Table 3.4: Sorted List of (AQMMRR)

Sorted Processes in Ascending Order							
Processes	P6	P3	P1	P2	P5	P7	P4
CPU burst	84	85	88	89	90	90	93

Quantum time (QT) = (MinBurst + MaxBurst) * 80%

MinBurst = 84 andMaxBurst = 93

QT = (84+93) * 80% = 177 * 0.8 = 141.6 ≈ 142

Gantt chart

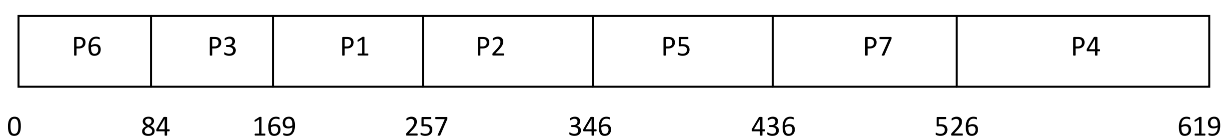


Figure 3.4: AQMMRR Gantt chart

Table 3.5: AQMMRR Analytic Result Table

Waiting Time:	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 169 P2: 257 P3: 84 P4: 526 P5: 346 P6: 0 P7: 436	P1: 169+88 = 257 P2: 257+89 = 346 P3: 84+85 =169 P4: 526+93 = 619 P5: 346+90 = 436 P6: 0+84 = 84 P7: 436+90 = 526	P1: 169 P2: 257 P3: 84 P4: 526 P5: 346 P6: 0 P7: 436
AWT ≈ 259.71, ATAT ≈ 348.14, No of context switching= 0, Average Response Time=259.71		

3.4.5 Half Life Variable Quantum Time Round Robin(HLVQTRR)

This approach ensures that half of every processes' bursts be executed in the first round, and in the second roundeach process should run to completion i.e. the quantum time, QT,in the first round is half each process's burst time while in the second roundQT will be equal to each processes' remaining burst time.

<u>Method</u>

Using the datasets P1=88, P2=89, P3=85, P4=93, P5=90, P6=84, P7=90

Table 3.6: QT Calculation for HLVQTRR Algorithm

Round 1 Quantum time	**Remaining processes CPU burst after round 1**	**Round 2 Quantum time**
P1: quantum time = 88/2 = 44	P1=44	P1: quantum time =44
P2: quantum time = 89/2 ≈ 45	P2=44	P2: quantum time = 44
P3: quantum time = 85/2 ≈ 43	P3=42	P3: quantum time = 42
P4: quantum time = 93/2 ≈ 47	P4=46	P4: quantum time = 46
P5: quantum time = 90/2 = 45	P5=45	P5: quantum time = 45
P6: quantum time = 84/2 = 42	P6=42	P6: quantum time = 42
P7: quantum time = 90/2 = 45	P7=45	P7: quantum time = 45

Gantt chart

Round1 | Round2

P1	P2	P3	P4	P5	P6	P7	P1	P2	P3	P4	P5	P6	P7

0 44 89 132 179 224 266 311 355 399 441 487 532 574 619

Figure 3.5: HLVQTRR Gantt chart

Table 3.7: HLVQTRR Analytic Result Table

Waiting Time:	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1:0+ 267=267 P2: 44+266= 310 P3: 89+267= 356 P4: 132+262= 394 P5: 179+263= 442 P6: 224+266 = 490 P7: 266+263 = 529	P1: 267+88=355 P2: 310+89 = 399 P3: 356+85=441 P4: 394+93=487 P5: 442+90=532 P6: 490+84=574 P7: 529+90=619	P1: 0 P2: 44 P3: 89 P4: 132 P5: 179 P6: 224 P7: 266
AWT ≈ 398.29, **ATAT ≈ 486.71, No of context switching= 7, Average Response Time=133.43**		

3.4.6 Improved Half Life Variable Quantum Time Round Robin (IHLVQTRR)Algorithm

Sometimes, some processes may go for more than two rounds and as such increases the number of context switching, average waiting time and average turnaround time. The proposed Improved Half Life Variable Quantum time RR ensures that processes run only for two cycles as suggested by (Simon *et al.*, 2014). First average of the processes burst times is taken, and then compared with each process's burst time; any process with burst time less than the average will run to completion in the first round. Otherwise, if it is greater than the average half of it burst time is executed in first round and the remaining half run to completion in the second round. Another advantage of IHLVQRR is that it reduced the cost of calculating quantum time in the second round; this means that in the second round which is also the final round, remaining burst time of processes greater than the average re-appear and will be taken as the QT for that round. Looking at other algorithms that may have two or more rounds, those algorithms will have to invoke the same algorithm to calculate the quantum time. This cost may not be justifiable.

<u>Method</u>

Using the same dataset: P1=88, P2=89, P3=85, P4=93, P5=90, P6=84 and P7=90 to evaluate IHLVQTRR.

Classical RR ideais use to find average of processes CPU burst time in the ready queue.Average= (88+89+85+93+90+84+90)/7 =619/7 = 88

Compare the average with each process burst time:P1,P3 and P6 have their burst time ≤ AVG they ran to completion in the first round. While P2, P4, P5 and P7 have burst time > AVG they will then execute in two cycles using half life approach.

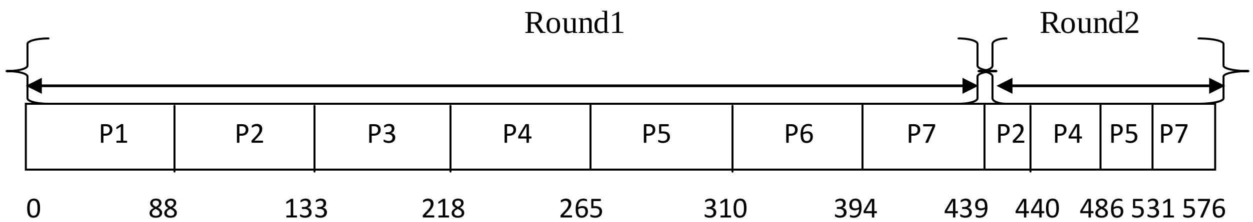
Quantum time of round 1 is half of each process burst time

P2=45, P4= 47, P5=45, and P7=45

Remaining CPU burst after round 1

P2=44, P4= 46, P5=45, and P7=45

Gantt chart



AWT=2157, ATAT=2776, N0 of context switch = 4

Figure 3.6: IHLVQTRR Gantt chart

Table 3.8: IHLVQTRR Analytic Result Table

Waiting Time:	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1:0	P1: 0+88=88	P1: 0
P2: 88+306= 394	P2: 394+89 = 483	P2: 88
P3: 133	P3: 133+85=218	P3: 133
P4: 218+175= 393	P4: 393+93=486	P4: 218
P5: 265+176= 441	P5: 441+90=531	P5: 265
P6: 310	P6: 310+84=394	P6: 310
P7: 394+92 = 486	P7: 486+90=576	P7: 394
AWT ≈ 308.14,ATAT ≈ 396.57,No of context switching= 4, Average Response Time= 201.14		

3.5 Summary of the Results

Table 3.9 displayed the summary results of all of the compared algorithms based on their average waiting time, average turnaround time, average response time andthe number of context switches.

Table 3.9: Analytical Result

EVALUATION SUMMARY					
Criteria	**Classical RR**	**EORR**	**AQMMRR**	**HLVQTRR**	**IHLVQTRR**
AWT	387.43	298.29	259.71	398.29	308.14
ATAT	475.86	386.71	348.14	486.71	396.57
ART	261.71	263.14	259.71	133.43	201.14
N0 of switches	4	4	0	7	4

It can be observed from the table that:

Average waiting time (AWT) and Average turnaround time (ATAT):

AQMMRR has better AWT and ATAT, followed by the EORR, proposed algorithm IHLVQTRR, classical RR then HLVQTRR.

Average response time: HLVQTRR has high response time more than any of the algorithms, followed by the IHLVQTRR, AQMMRR, classical RR then EORR.

Number of context switching: IHLVQTRR, Classical RR and EORR have optimal number of context switch. AQMMRR has no context switches, which is based on the algorithm. HLVQTRR have the highest context switches.

CHAPTER FOUR

EVALUATION OF THE PROPOSED IHLVQTRR ALGORITHM

4.1 Introduction

In order to understand the behavior of the designed algorithmand draw valid conclusions, we developed a simulator that considers a wide range of datasetsused for extensive experimentation andanalysis.

4.2 Description of the Simulation

Java programming language was used to design a simulator that generates random variables using normal, uniform and exponential distribution functions. The generated datasets were ten for uniform in the first category and in the second category one thousand processes were generated for the uniform, normal and exponential distributionsfunctions respectively. The input parameters for the uniform and normal distributions were: quantum time (QT) to be used by the classical RR, mean (μ),standard deviation $(\sigma)$and number of processes.Exponential distribution has only one parameter (λ) and number of processes. These parameters are variables, meaning they can assume any value. By choosing a distribution function and supplying the inputs, the simulator automatically generates sets of processes and their associated CPU bursts,and thenexecutes the processes in Round Robin fashion. Finally, it computesaverage waiting time, average turnaround time, average response time and number of context switchesfor each algorithm.All the findings gotten from the developed simulator were examined and discussed under some important criteria used in the time sharing systems.

4.3. Distribution Functions

Uniform, normal and exponential distribution functions are used to generatethe datasets.

4.3.1. Uniform Distribution Function: Random variable X is said to be uniformly distributed on the interval [a, b]. So, using the Inverse Transformation method:

The probability density function (pdf) is given by:

$$f(x) = \frac{1}{b-a} \quad where\ a < x\ < b \tag{4.1}$$

Cumulative density function (cdf) is given by:

$$f(x) = \frac{x-a}{b-a} = R \quad \text{Or} \quad x = a + (b-a)R \tag{4.2}$$

which is a reasonable guess for generating X and R is always a random number on [0, 1] (Jerry *et al*, 2005).

The mean, variance and standard deviation of uniform distribution is given by:

$$E(X) = \frac{a+b}{2} \quad V(X) = \frac{(b-a)^2}{12} andS.D\ = \sqrt{\frac{(b-a)^2}{12}} \tag{4.3}$$

Where: $a, b\ are\ upper\ and\ lower\ bound$

4.3.2. Normal Distribution Function:a random variable X with mean $-\infty<\mu<\infty$ and variance $\sigma^2>0$ has a normal distribution if it has the pdf:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{4.4}$$

Where: $\sigma - varience\ \ x - is\ a\ varieble,\ \ \mu - mean\ \ 2\pi - is\ a\ contant\ value$

4.3.3. Exponential Distribution Function:random variable X is exponentially distributed with parameter rate λ. The probability density function (pdf) is given by: $f(x) = \lambda e^{-\lambda x}$ (4.5)

Where: $\lambda - parameter, x - is\ a\ varieble\ and\ \ e -$ exponential function

Its mean and variance respectively are given by:

$E(X) = \frac{1}{\lambda} \quad and \quad V(X) = \frac{1}{\lambda^2}$

4.4. Experimental Setup

a. Hewlett Packard (HP) laptop with a T2300 processor running at 1.66GHz

b. Size of the RAM is 1.5 GB and the hard disk size is 75GB.

c. Windows XP operating system

d. NetBeans IDE 6.7.1 version and JDK1.7

4.5. Assumptions

a. The simulator considers all processes to be in the ready queue

b. Processes are considered to be of same priority.

c. Arrival time set to be zero.

4.6 CPU Scheduling Criteria

Many criteria have been suggested for comparing CPU scheduling algorithms. These characteristics can be used to judge which algorithm outperformed the others according to some of the criteria below:

a. CPU utilization: The idea is to keep the CPU as busy as possible. This criterion should be maximized. The CPU should be busy 100%.

b. Throughput: This is the number of tasks that can be completed per unit amount of time. If the CPU is highly utilized, then the number of tasks that can be completed within a time unit will be high. Just as CPU utilization, throughput should be maximized.

c. Waiting Time: This is the sum of the time spent waiting in the ready queue. As for waiting time, the goal is to minimize it.

d. Turnaround Time: This is concerned with how long it takes to finish executing a process. The time a process takes from when it is submitted for execution to when it finishes execution. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O. The turnaround time should be minimized.

e. Response Time: In an interactive system, turnaround time may not be the best criterion. Often, a process may produce some output fairly early and continue computing new results while previous results are being output to the user. Response time is the time from the submission of a request to when the first response is produced. It is the time taken to starts responding. It should be minimized.

f. Number of context switching: Context switching is the act of switching the CPU to another process while performing a state save of the current process and a state restore of a different process. Even though switching is pure overhead, because the system does no useful work while switching, it is needful in time sharing systems .So, an optimal switching is required for high system performance.

g. Overhead: Overhead refers to the proportion of time wasted due to computation of the schedule, and the system overhead due to context switching.

h. Fairness: In the absence of user or system supplied criteria for selection, the scheduler should allocate a fair amount of the resource to each task.

i. Load balancing: The scheduler should balance the load across other system resources, such as memory and I/O usage.

4.7. Results Discussion

Tendatasetswere randomly generated using uniform distribution function in the first category, and in the second category one thousand processes with their associated burst timeswere generated using uniform, normal and exponential distribution functions.

4.7.1 Result of Uniform Distribution

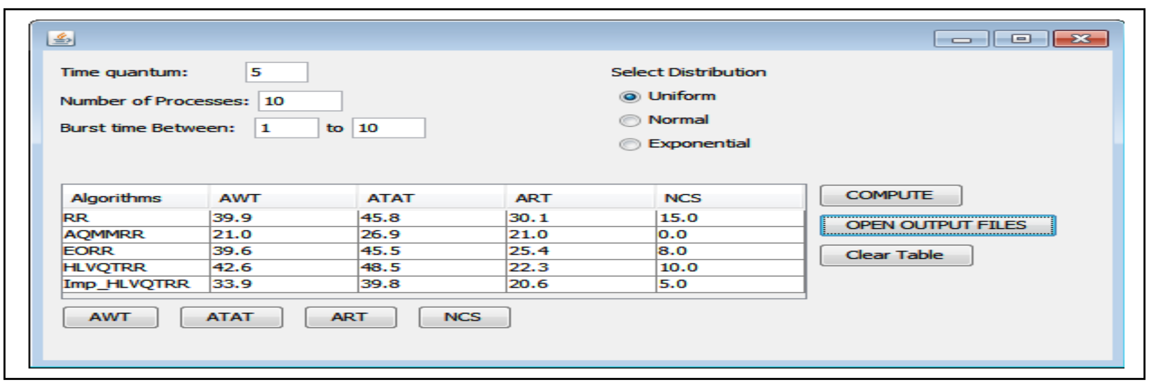
Uniform requires two input parameters to generate processes and their associated CPU bursts. These are: upper bound (a), lower bound (b) and number of process. These parameters are variables, meaning they can assume any value.

In the first category,below were the results obtained using tendatasets.

From the simulation, Upper bound (a) = 10 lower bound (b) = 1 this means that

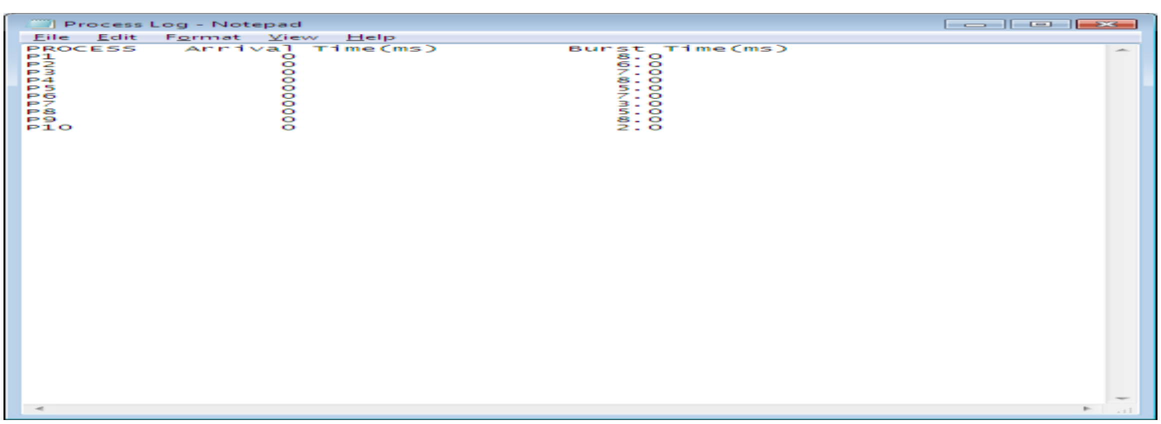
Mean = $\frac{1+10}{2} = \frac{11}{2} = 5.5$

Fig 4.1 shows the system interface of 10 processes generated by uniform distribution with burst times ranging between upper bound 100 and lower bound 10*ms*,mean of 5.5*ms*,and time quantum of 5*ms* used by RR.



Algorithms	AWT	ATAT	ART	NCS
RR	39.9	45.8	30.1	15.0
AQMMRR	21.0	26.9	21.0	0.0
EORR	39.6	45.5	25.4	8.0
HLVQTRR	42.6	48.5	22.3	10.0
Imp_HLVQTRR	33.9	39.8	20.6	5.0

Figure 4.1: System Interface of Uniform Distribution Function



PROCESS	Arrival Time(ms)	Burst Time(ms)
P1	0	8.0
P2	0	6.0
P3	0	7.0
P4	0	8.0
P5	0	5.0
P6	0	7.0
P7	0	3.0
P8	0	5.0
P9	0	8.0
P10	0	2.0

Figure 4.2: Process identity snap shot

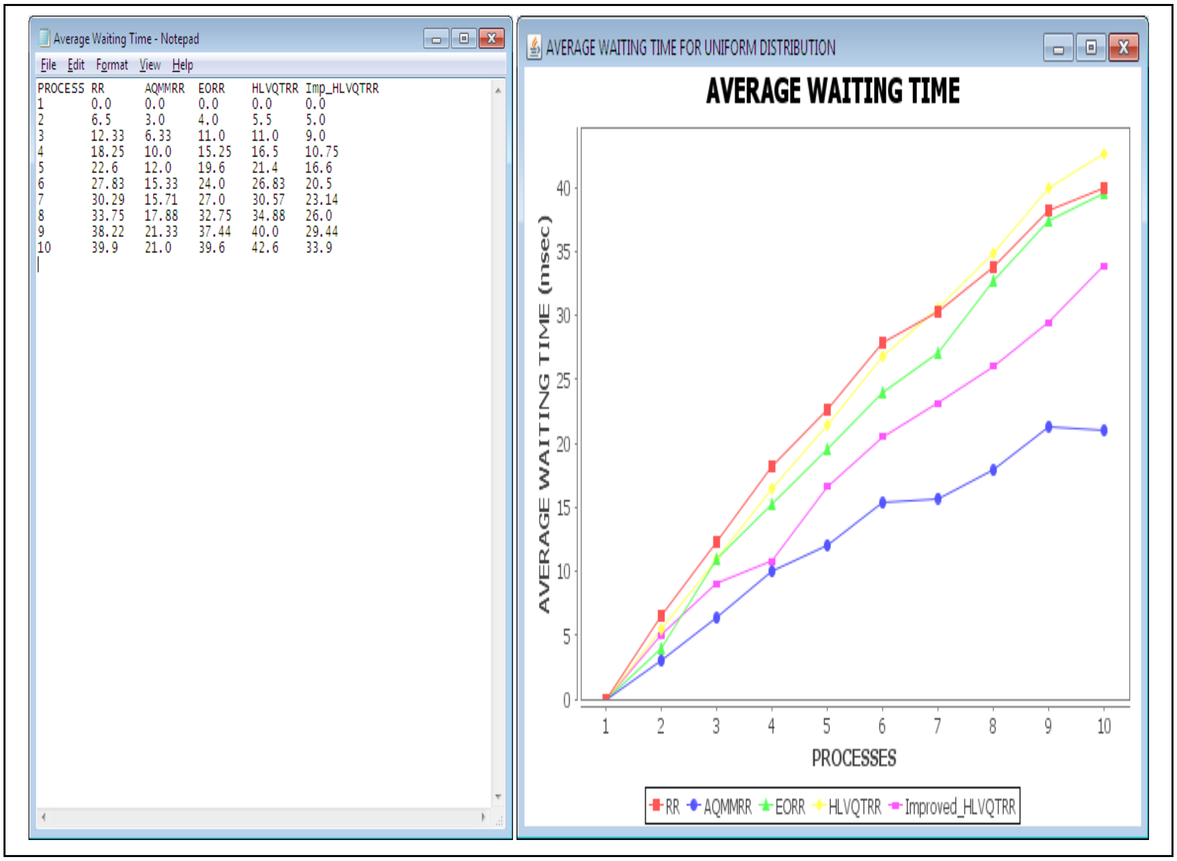


Figure 4.3: Graph of Average Waiting Timefor Uniform Distribution

From the graph of fig. 4.3, AQMMR shows best resultthan all other algorithms because it worked exactly as Shortest Job First as it’s quantum time is always greater than the maximum process CPU burst in the ready queue. IHLVQTRR produces better result of AWT followed by EORR, HLVQTRR and then Classical RR algorithms.

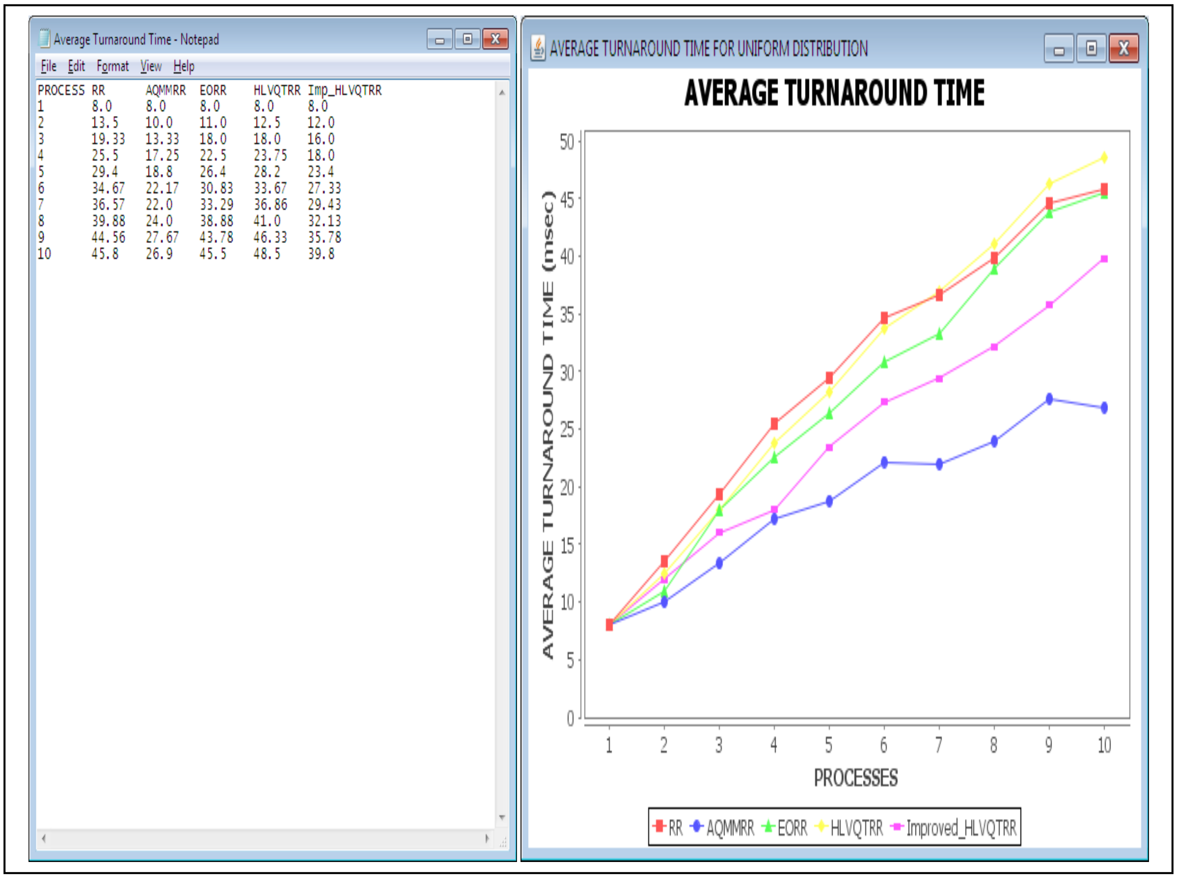


Figure 4.4: Graph of Average Turnaround Time for Uniform Distribution

From the graph of Fig 4.4, AQMMR shows best result than all the algorithms because it worked exactly as Shortest Job First as it's quantum time is always greater than the maximum process's CPU burst in the ready queue. IHLVQTRR produces better result of ATAT followed by EORR, HLVQTRR and then Classical RR algorithms.

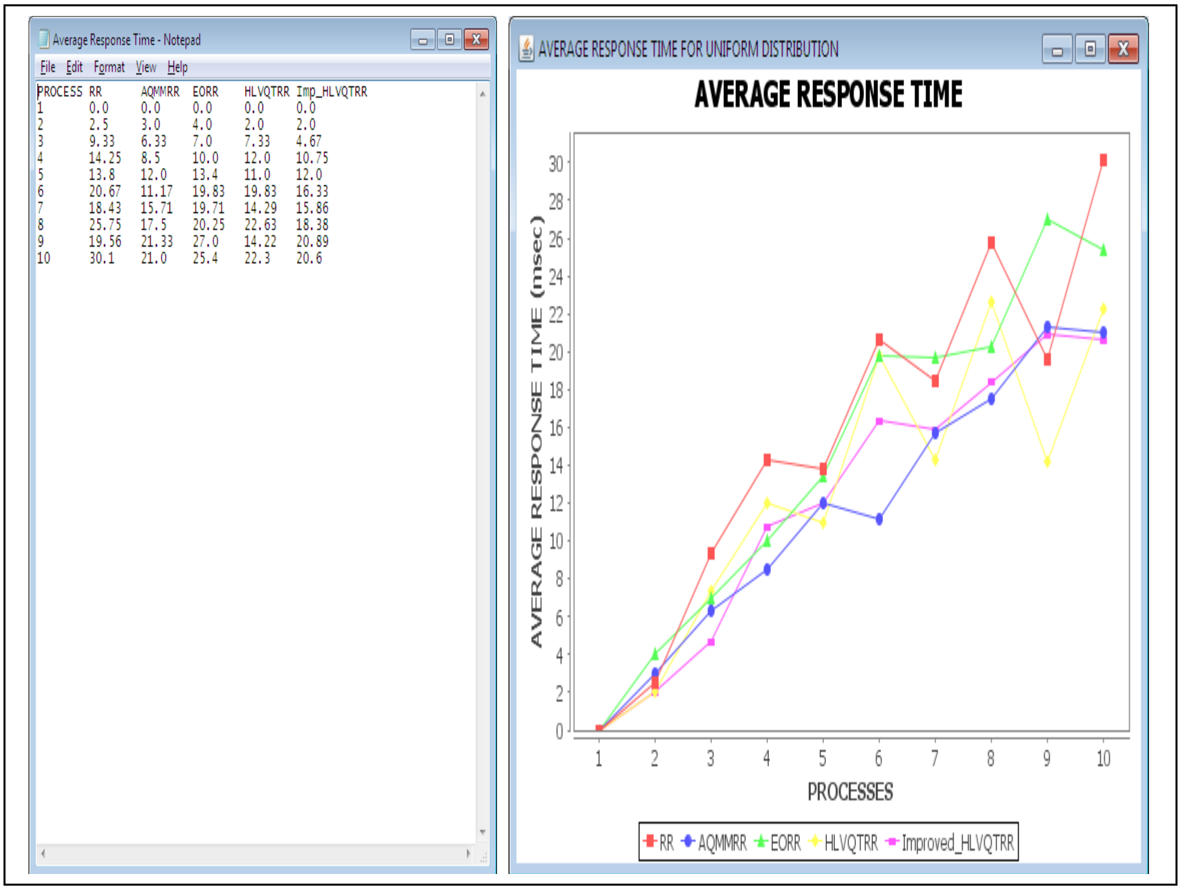


Figure 4.5: Graph of Average Response Time for Uniform Distribution

From the graph of Figure 4.5, all the five algorithms fluctuate, because the generated random variables were not sorted. AQMMR hasthe lowest fluctuation followed by theHLVQTRR, IHLVQTRR,EORR andthe Classical RR.

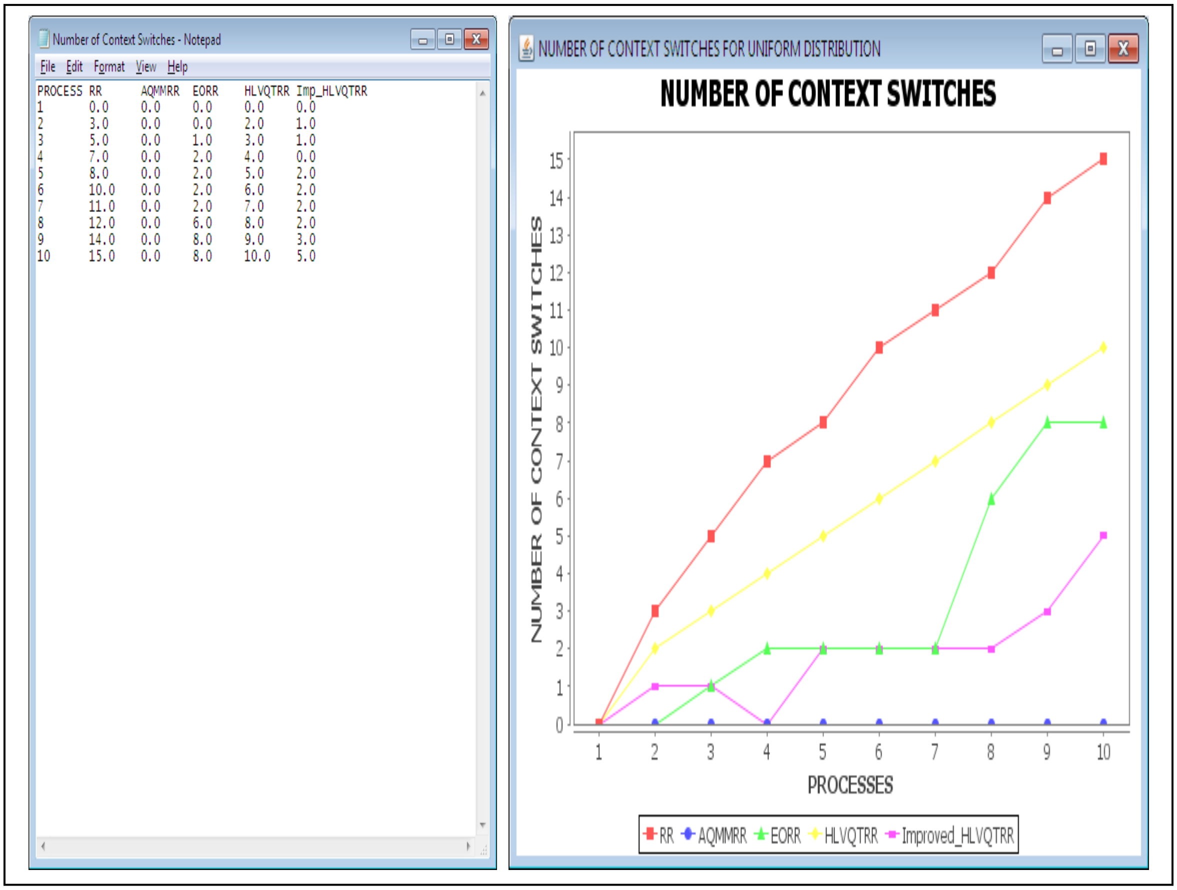


Figure 4.6: Graph of Number of Context Switches for Uniform Distribution

The graph of Figure 4.6 shows that the most extreme algorithms were Classical RR and HLVQTRR which have much context switches and AQMMRR which has its context switching too low. Ironically, AQMMRR has the lowest and at the same time the worst context switching because in any given RR scheduling at least some processes are expected to be switched to second round. In this case, none was switched to the next round. IHLVQTRR is second to have small NCS and it fluctuate base on the generated result of the random variables. EORR is the same with IHVQTRR at some point and higher at other points.

In the second category, below were the results obtained using one thousand datasets.

From the below simulation Upper bound (a) = 100lower bound (b) = 10 this means that

Mean = $\frac{10+100}{2} = \frac{110}{2}$ = 55.0

Fig 4.8 shows the system interface of 1000 processes generated by uniform distribution with burst times ranging between upper bound 100 and lower bond 10, mean of 55.0*ms*,and time quantum of 50*ms* used by RR

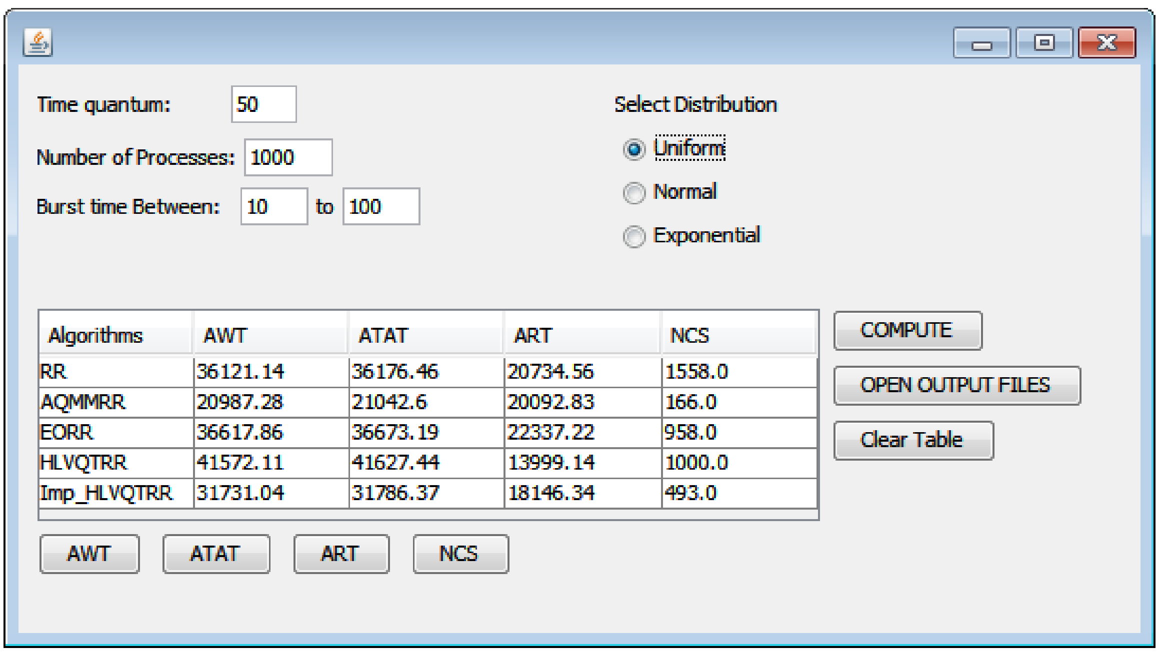


Figure 4.7: System Interface of Uniform Distribution Function

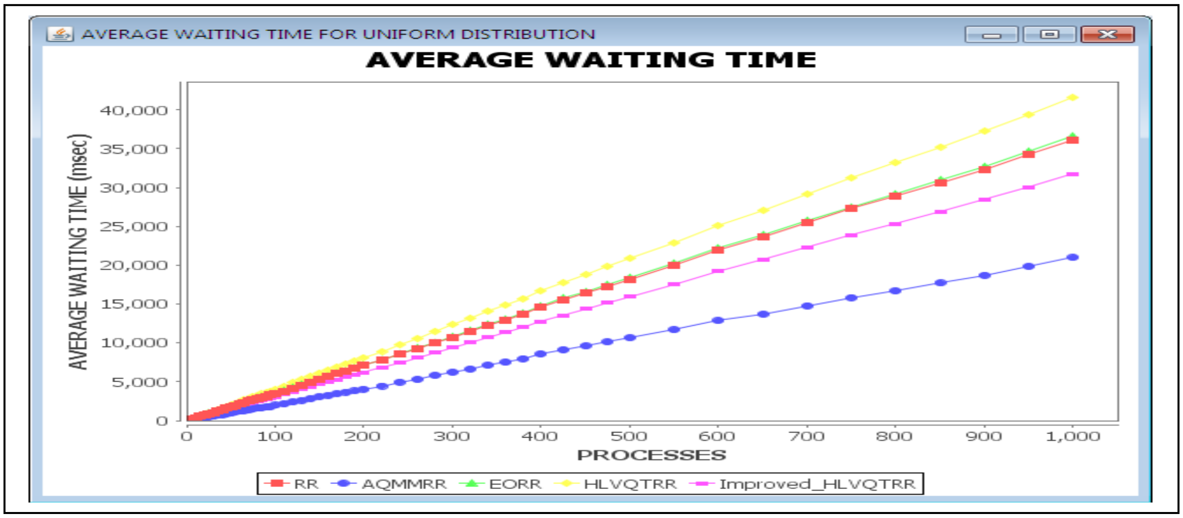


Figure 4.8: Graph of Average Waiting Time for Uniform Distribution

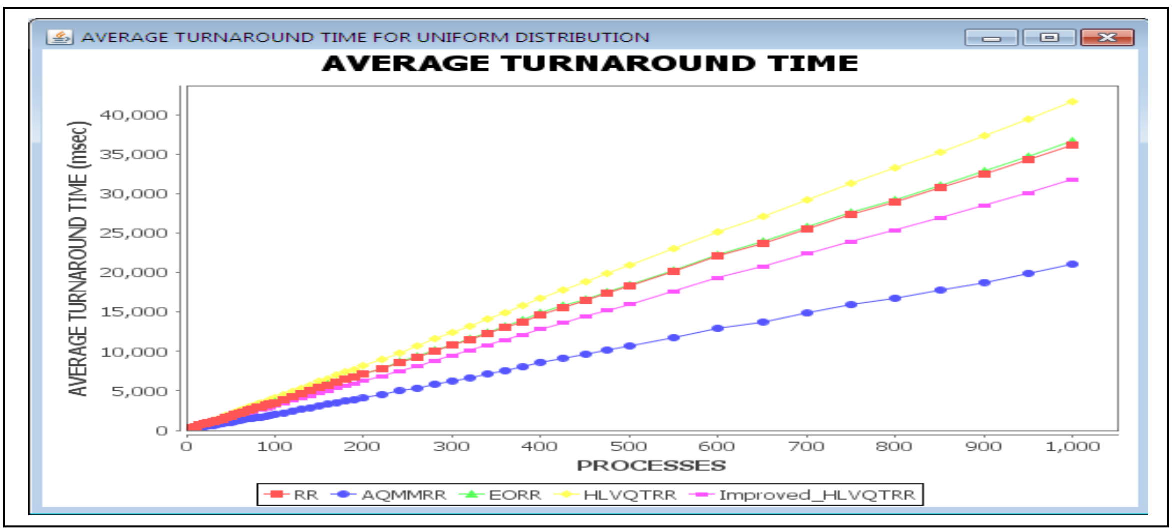


Figure 4.9: Graph of Average Turnaround Time for Uniform Distribution

From the graphs of Fig.4.8 and Fig 4.9AQMMRR has best result because its quantum time is always greater than the maximum process's CPU burst in the ready queue. IHLVQTRR shows better result of AWT and ATAT, EORR and Classical RR were roughly the same because the simulator generated close values for the two algorithms. HLVQTRR has the worst result because of the number of context switches that is twice the number of the processes since all the processes re-appear in the second cycle.

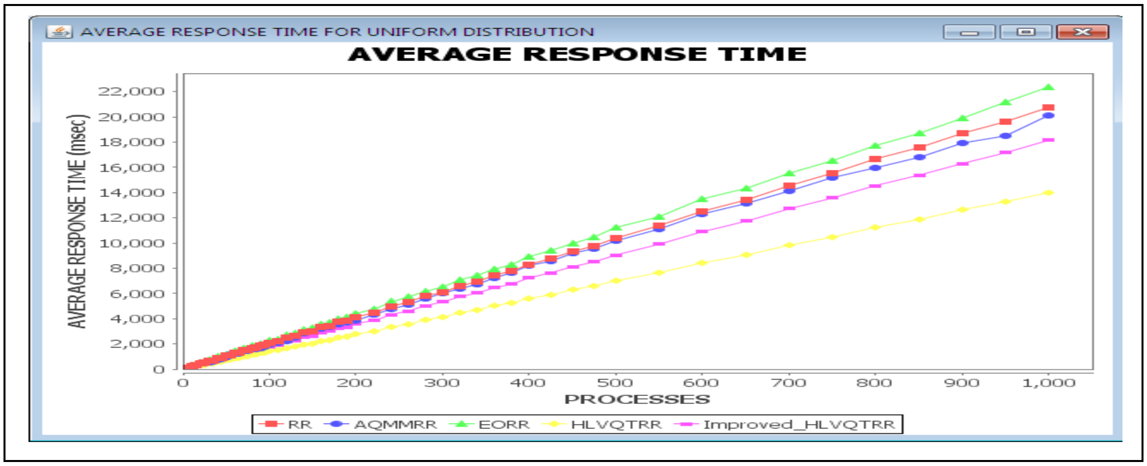


Figure 4.10: Graph of Average Response Time for Uniform Distribution

From the graph it is clear that HLVQTRR has very high response time because only half of the burst times were executed in the first cycle. IHLVQTRR is the second best in term of response time because it adopts the method of HLVQTRR. AQMMRR and classical RR were roughly the same then EORR has the worst result.

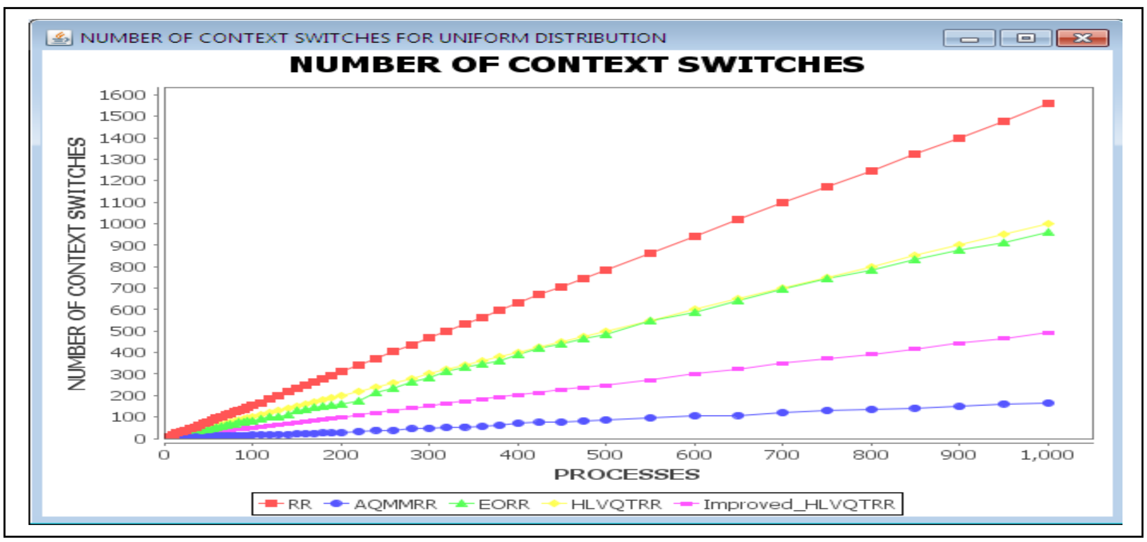


Figure 4.11: Graph of Number of Context Switch for Uniform Distribution

From the Fig 4.10, IHLVQTRR has the optimal NCS, Classical RR was the highest followed by HLVQTRR and EORR, while AQMMRR has the lowest because the algorithm is similar to the SJF algorithm.

4.7.2 Result of Normal Distribution

From the below simulation, this shows

Mean = $\frac{10+100}{2} = \frac{110}{2} = 55.0$ S.D = $\sqrt{\frac{(100-10)^2}{12}} = \sqrt{\frac{(90)^2}{12}} = \sqrt{\frac{8100}{12}} = \sqrt{675} = 25.9$

Fig 4.11shows a simulation snap shot of normal distribution function using 1000 processes with burst times ranging between 10 and 100*ms*, mean of 55.0*ms*, standard deviation of 25.9*ms* and time quantum of 50*ms* used by RR.

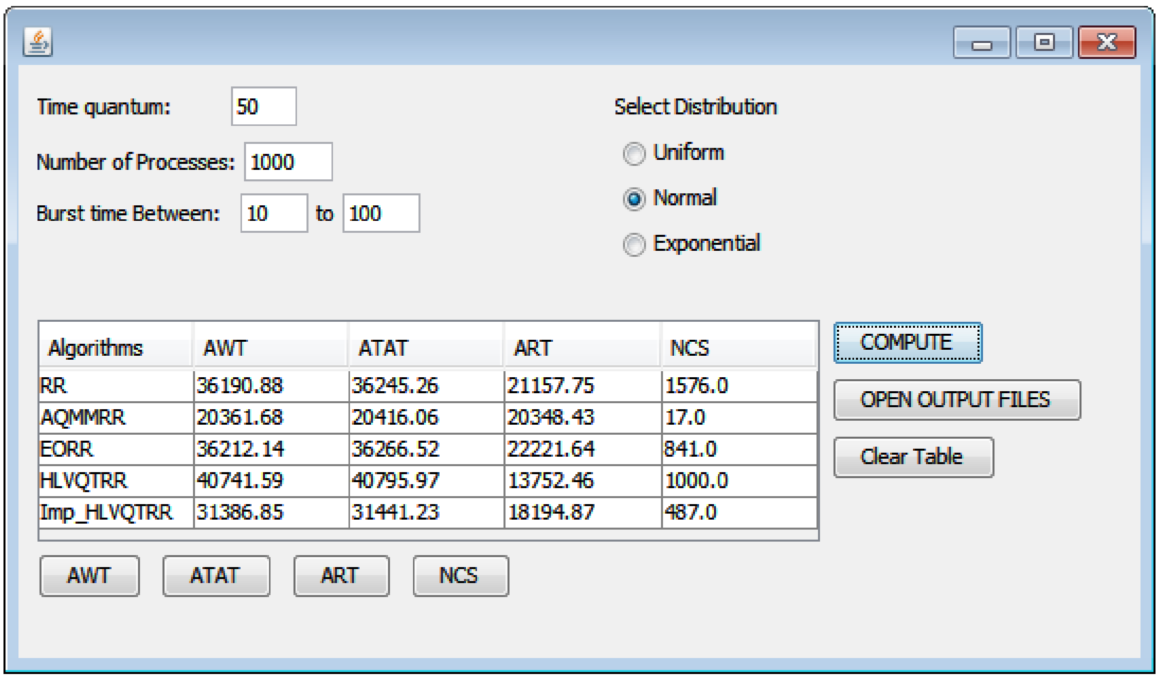


Figure 4.12: System Interface of Normal Distribution Function

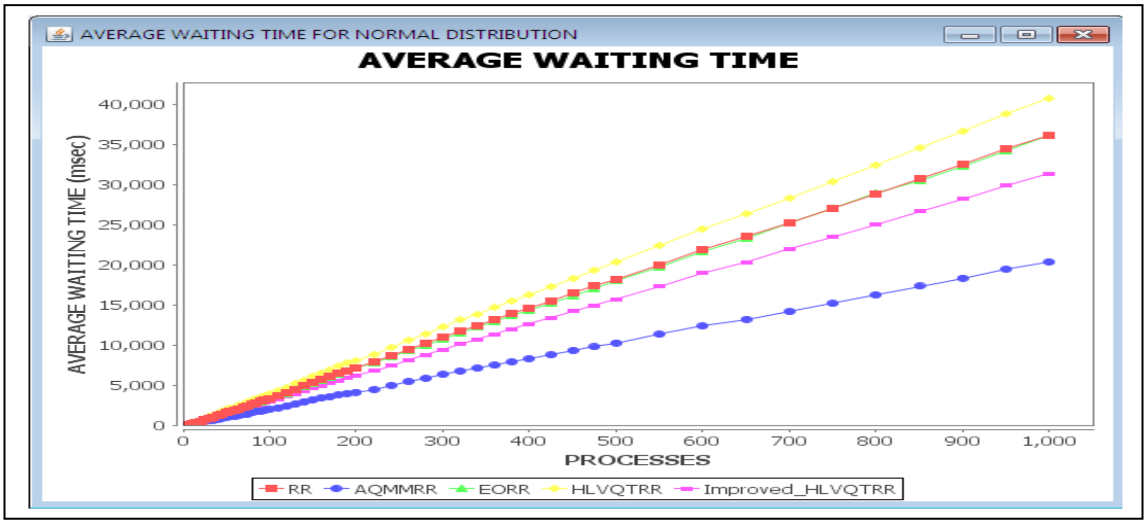


Figure 4.13: Graph of Average Waiting Time for Normal Distribution

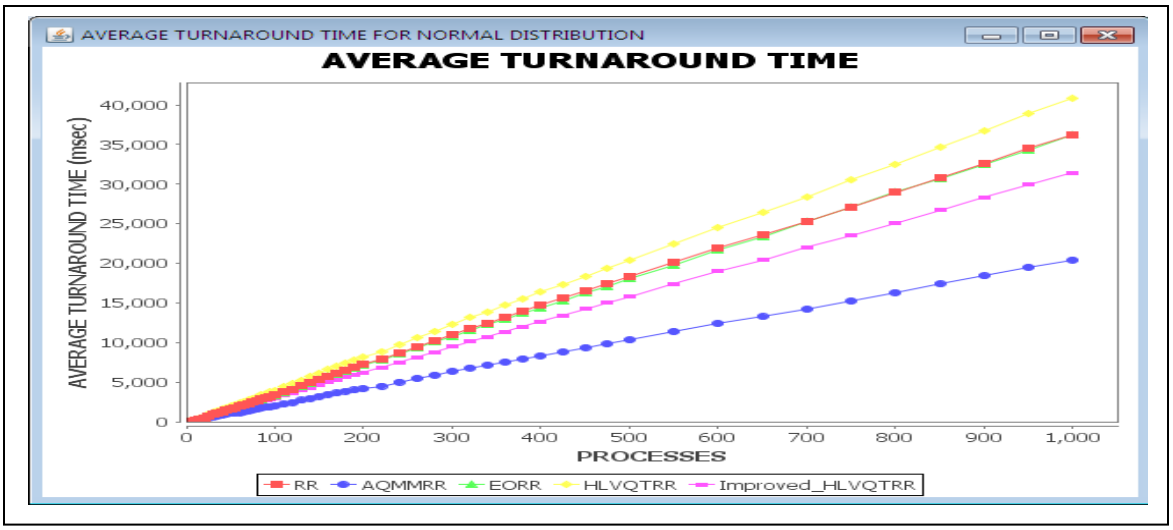


Figure 4.14: Graph of Average Turnaround Time for Normal Distribution

From the graphs of fig.4.12 and fig 4.13 AQMMRR has best result because its quantum time is always greater than the maximum process's CPU burst in the ready queue. HLVQTRR has the worst result because of the number of context switches that is twice the number of the processes since all the processes re-appear in the second cycle. IHLVQTRR shows better result of AWT and ATAT because it minimizes the number of context switches of the HLVQTRR algorithm. EORR and Classical RR were roughly the same because the simulator generated close values for the two algorithms.

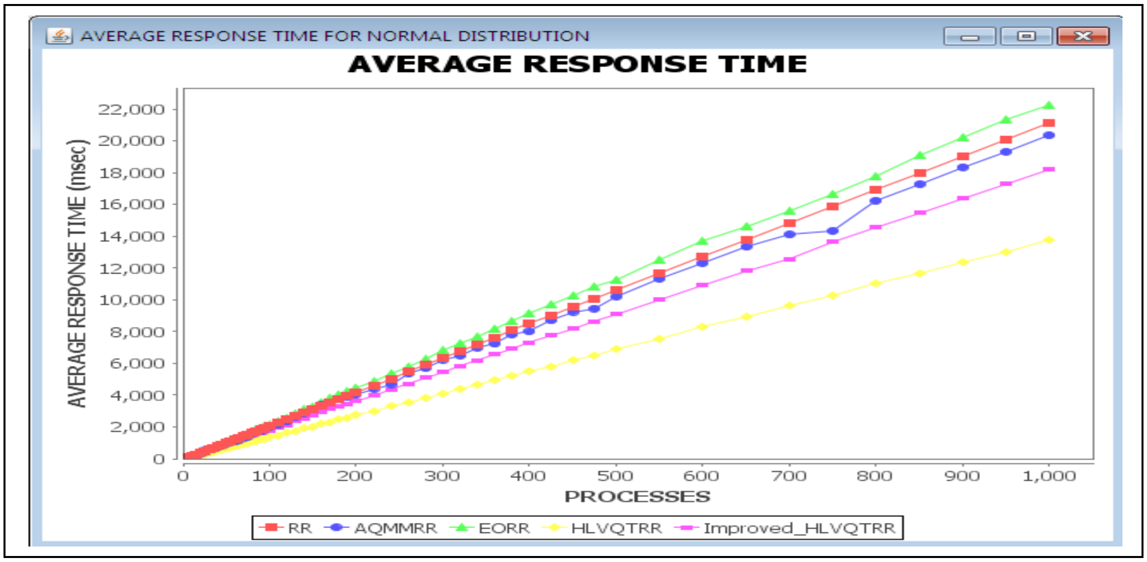


Figure 4.15: Graph of Average Response Time for Normal Distribution

From the graph it is clear that HLVQTRR has very high response time because only half of the burst times were executed in the first cycle. IHLVQTRR is the second best in term of response time because it adopts the method of HLVQTRR. AQMMRR, classical RR and EORR results follows each order.

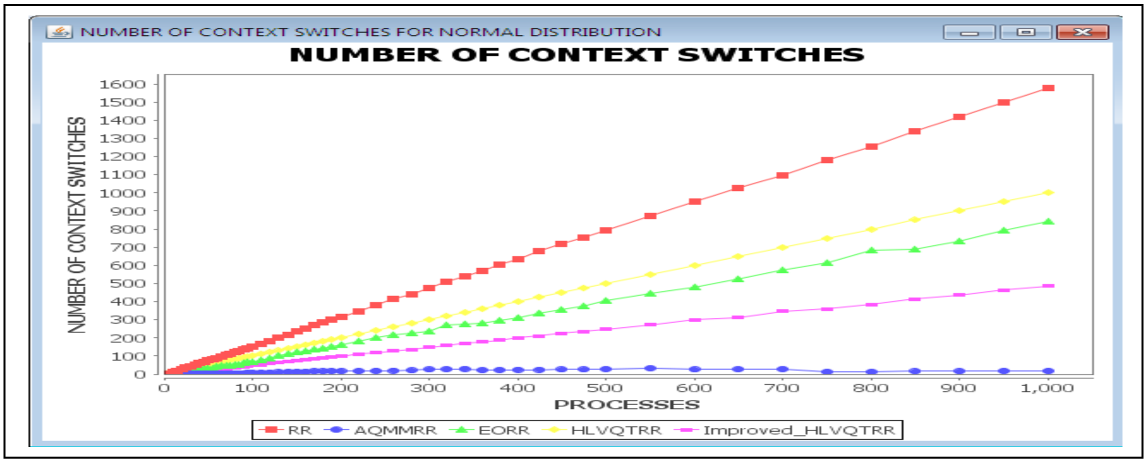


Figure 4.16: Graph of Number of Context Switch for Normal Distribution

From the Fig 4.15, IHLVQTRR has the optimal NCS, Classical RR was the highest followed by HLVQTRR and EORR, while AQMMRR has the lowest because the algorithm is similar to the SJF algorithm.

4.7.3 Result of Exponential Distribution

Fig 4.17 shows simulation snap shot of exponential distribution function using 1000 processes with a rate (λ) of 0.02*per ms*.

Exponential requires one input parameter (λ) to generate processes and their associated CPU burst time.

From the simulation, Upper bound (a) = 100 lower bound (b) = 10 this means that

Mean $=\frac{a+b}{2}=\frac{10+100}{2}=\frac{110}{2}=55$ $\quad \lambda=\frac{1}{mean}$ $\quad \lambda=\frac{1}{55}=0.02$

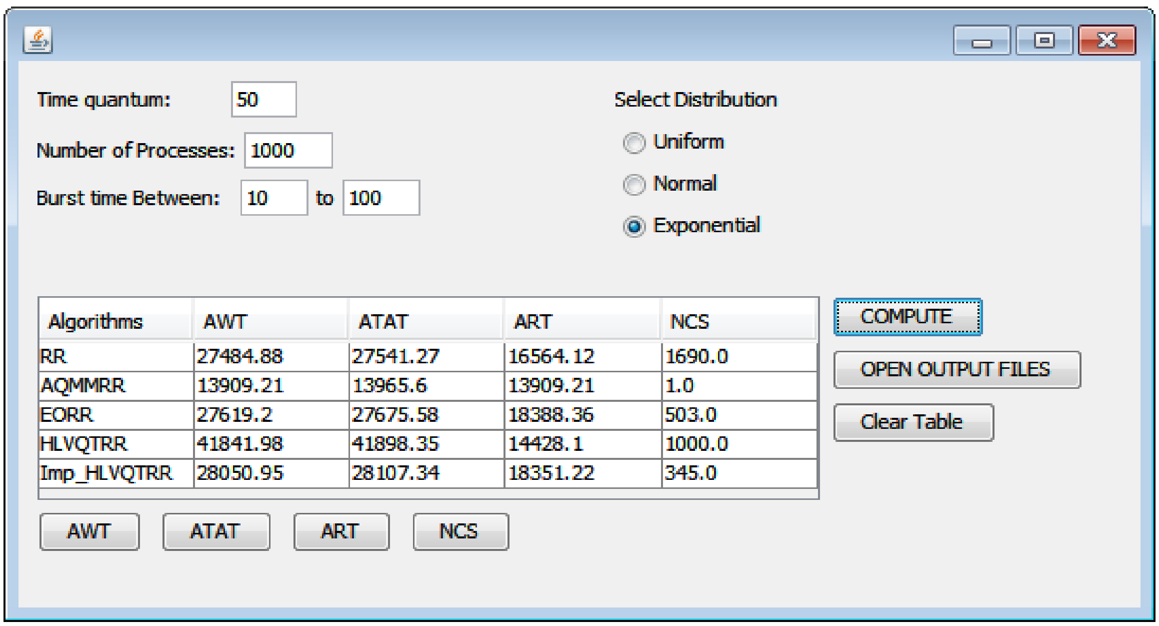


Figure 4.17: System Interface of Exponential Distribution Function

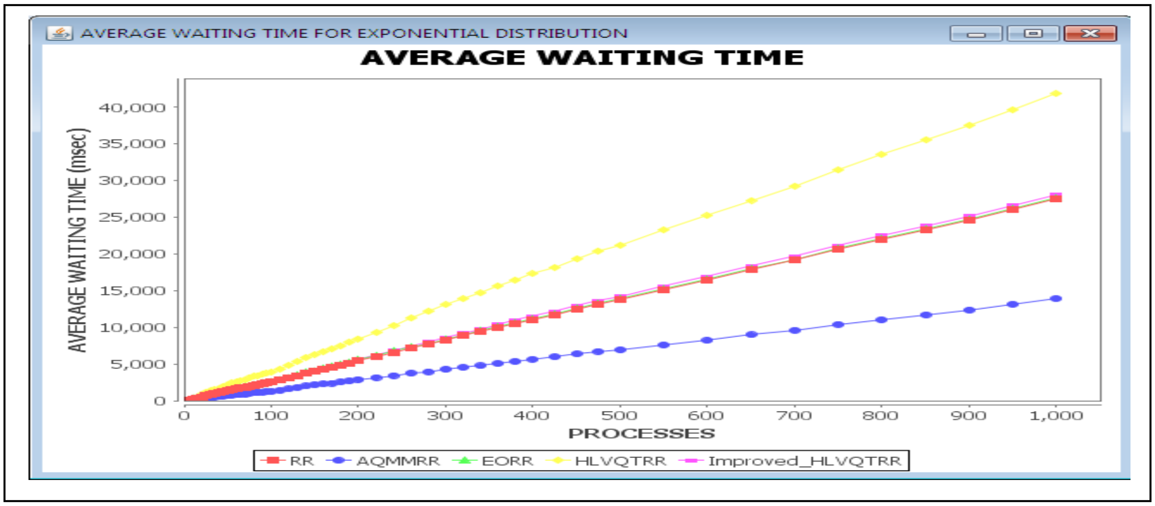


Figure 4.18: Graph of Average Waiting Time for Exponential Distribution

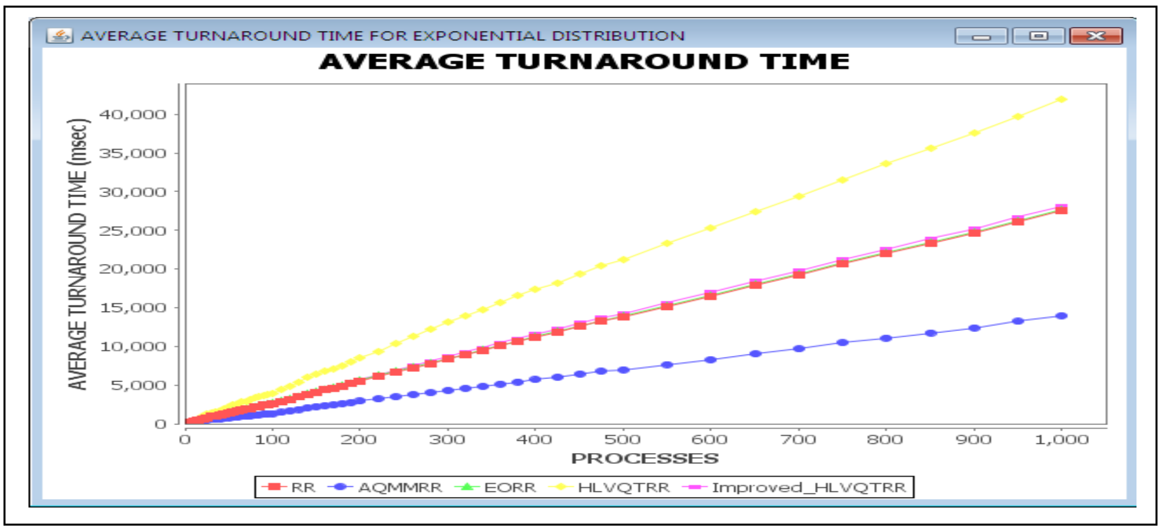


Figure 4.19: Graph of Average Turnaround Time for Exponential Distribution

From the graphs of fig.4.18 and fig 4.19 AQMMRR has best result because its quantum time is always greater than the maximum process's CPU burst in the ready queue. HLVQTRR has the worst result because of the number of context switches that is twice the number of the processes since all the processes re-appear in the second cycle.

IHLVQTRR, EORR and classical RR converged at the center and were roughly the same because the simulator generated close values for the three algorithms.

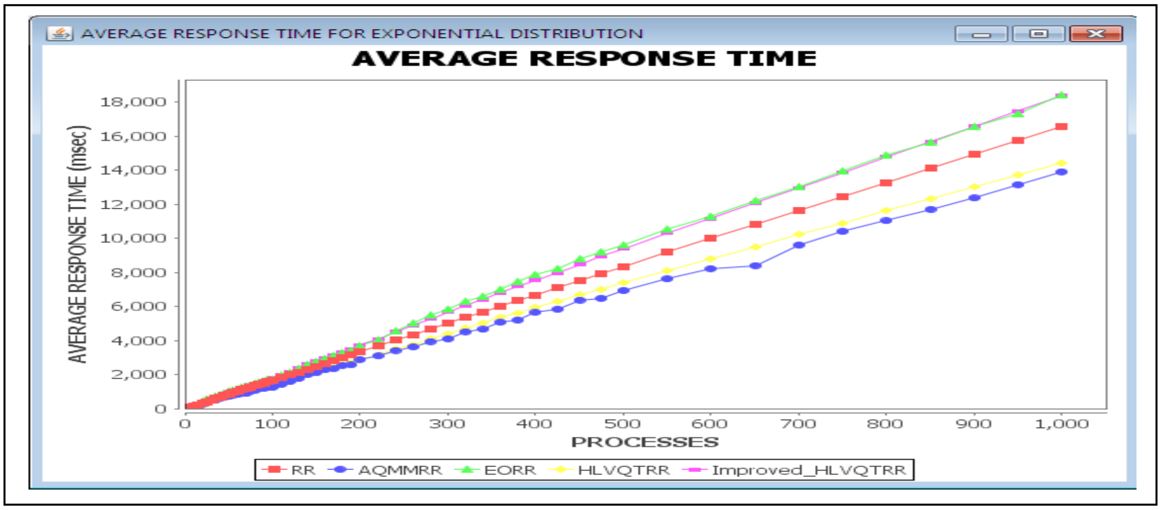


Figure 4.20: Graph of Average Response Time for Exponential Distribution

AQMMRR shows betterresult of ART followed by the HLVQTRR algorithm. Classical RR responds better than the IHLVQTRR and EORR.

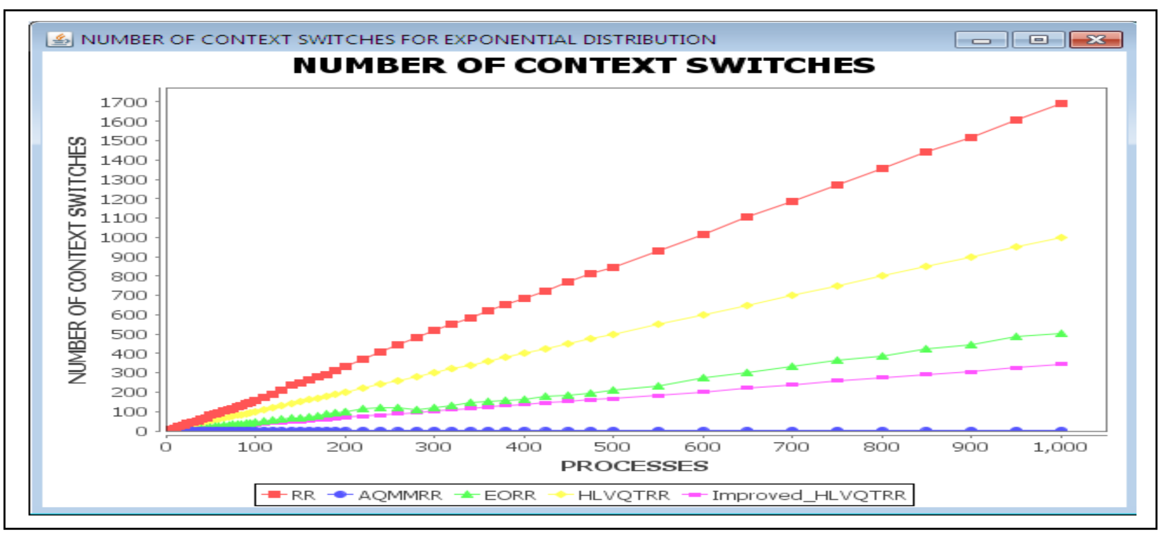


Figure 4.21: Graph of Number of Context Switch for Exponential Distribution

AQMMRR has the lowest and at the same time worst context switching because in any given RR scheduling at least some processes are expected to be switched to second round. In this case, none was switched to the next round. IHLVQTRR shows optimal NCS followed by EORR and IHLVQTRR. Classical RR has the worst result of NCS.

4.8. Summary of the Results

The tables4.1 to 4.3 below summarizes the performance difference of the uniform, normaland exponential distribution functions based on the scheduling criteria's for the second category of the datasets using 1000 processes.

Table 4.1: Uniform Distribution Evaluation Summary

EVALUATION SUMMARY				
Criteria	**AWT**	**ATAT**	**ART**	**NCS**
Classical RR (%)	**12**	**12**	**12**	**12**
AMMQRR (%)	**-51**	**-51**	**10**	**-96**
EORR (%)	**13**	**13**	**19**	**49**
HLVQTRR (%)	**24**	**24**	**-30**	**50**
IHLVQTRR (%)	**0**	**0**	**0**	**0**

Table 4.2: Normal Distribution Evaluation Summary

EVALUATION SUMMARY				
Criteria	**AWT**	**ATAT**	**ART**	**NCS**
Classical RR (%)	**13**	**13**	**14**	**69**
AMMQRR (%)	**-35**	**-54**	**11**	**-28**
EORR (%)	**13**	**13**	**18**	**42**
HLVQTRR (%)	**23**	**23**	**-32**	**51**
IHLVQTRR (%)	**0**	**0**	**0**	**0**

From the tables above of the uniform and normal distribution functions in the second category,AQMMRR algorithm shows optimal results of AWT, ATAT and NCS than all other algorithms because it worked exactly as Shortest Job First as it's quantum time is always greater than the maximum process CPU burst in the ready queue.The proposed IHLVQTRR algorithm produces better average waiting time (AWT), average turnaround time (ATAT) and optimal number of context switches (NCS) because the number of context switches that affected the AWT and ATAT of the HLVQTRR has been minimized. The HLVQTRRhas bestresponse time than all the algorithms because only half of the processes burst times were executed in the first cycle.IHLVQTRR was the second best in terms of ART followed by the EORR and Classical RR whichwere roughly the same because the simulator generated closerprocesses burst times for the two algorithms.

Table 4.3: Exponential Distribution Evaluation Summary

EVALUATION SUMMARY				
Criteria	**AWT**	**ATAT**	**ART**	**NCS**
Classical RR (%)	**-2**	**-2**	**-10**	**79**
AMMQRR (%)	**-100**	**-100**	**-31**	**-100**
EORR (%)	**-2**	**-2**	**20**	**0.2**
HLVQTRR (%)	**33**	**33**	**-27**	**66**
IHLVQTRR (%)	**0**	**0**	**0**	**0**

With the exponential distribution, the two extreme algorithms in terms of the average waiting time, average turnaround time and number of context switcheswere HLVQTRRand AQMMRRHLVQTRR has much context switches and AQMMRR has its context switches too low. Ironically, AQMMRR has the lowest and at the same time the worst context switches because in any given RR scheduling algorithm at least some processes are expected to be switched to second round. In this case, none was switched to the next round.The improved HLVQTRR, EORR and classical RR algorithms results converges at the center in terms of AWT and ATAT. This is because the generated variables for the algorithms were close. HLVQTRR have best ART followed by the classical RR IHLVQTRR then the EORR algorithm.

Table 4.4 below analyzes the ProposedHLVQTRR side by side HLVQTRR, EORR, AQMMRR and classical RR algorithms based on the following criteria

Table 4.4: Qualitative Evaluation Summary of the algorithm

GENERAL TABLE OF ANALYSIS								
Criteria Algorithms	AWT	ATAT	ART	NCS	Complexity	Sorting cost	Priority on shorter jobs	Fairness
Classical RR	Good	Good	Fair	Fair	No	No	No	Not Fair
IHLVQTRR	Better	Better	Good	Good	No	No	No	Always
HLVQTRR	Good	Good	Best	Fair	No	No	No	Always
EORR	Better	Better	Fair	Fair	No	No	No	Not Fair
AQMMRR	Better	Better	Poor	Poor	No	Yes	Yes	Not Fair

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

Round Robin (RR) is the main scheduling algorithm used in time sharing systems. Though it has worst average waiting time and average turnaroundtime compared to First Come First Serve (FCFS) and Shortest Job First (SJF), yet it maximizes high response time and multiprogramming rate. Because of its unique features, it has gained more popularity in today's systems.In this regard,Half Life Variable Quantum Time Round Robin (HLVQTRR) by Simon (2014) was proposed with the objective ofimprovingon response time and multiprogramming rate of traditional round robinso that it does not suffer much on average waiting time, average turnaround time and number of context switching. Although the objectives were achieved, there was need to still work on the performance of average waiting time, average turnaround time and number of context switchesas the resultswere not very satisfactory, Based on this reason theImprovedHalf Life Variable Quantum Time Round Robin (IHLVQTRR)was proposed with theobjectivesof improving greatly on average waiting time, average turnaround time and number of context switching of HLVQTRR algorithm.The proposed algorithm (IHLVQTRR) along with HLVQTRR, Classical RR, EORR, and AQMMRR algorithms were implemented in Java using three distribution functions (uniform, normal and exponential) to generate 10 datasets for uniform in the first category, and 1000 datasets in the second category for all the three distributions.The generated variables were then used to evaluate and compare results of the proposed algorithm IHLVQTRR against the classical RR, HLVQTRR, EORR and AQMMRR based on average waiting time, average turnaround time, average response time and number of context switches. Resultof the evaluation shows that AQMMRR has the lowest results becauseitsQT is greaterthan the maximum process burst times. The proposed algorithm IHLVQTRR, AWT, ATAT and NCS was better than that of the HLVQTRR, EORR and Classical RR algorithms in allcategories, but HLVQTRR algorithm has the best ART.

5.2 Conclusion

In conclusion, the proposed algorithm Improved Half Life Variable Quantum Time Round Robin (IHLVQTRR) providessolution to the poor context switches of the HLVQTRR such that it does not suffer much from average waiting time and average turnaround time. It also reduced the cost of computing quantum time for burst times lower than average that is, in the second round which is also the final round the quantum time is known. From the results of the Analysis, it was observed that the performance of the proposed IHLVQTRR algorithmprovides better solutions of AWT, ATAT and NCS than all other compared algorithms. The reviewed algorithm HLVQTRR produces better average response time than the proposed algorithm.

5.3 Recommendation

Since IHLVQTRR algorithmproven to be better, it should be preferred over the HLVQTRR and the reviewed dynamic round robin. In future, there's need to testthe algorithm on high performance computers where multiple processors are used instead of a single processors so that o see the behavior of the algorithm

REFERENCES

Abbas, N, Ali, K and Seifedine, K. (2011). A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum using the mean average. *IJCS International Journal of Computer Science Issues*, vol. 8, Issue 3, No.1 224-229.

Abdulrazaq, A, Saleh, E.A. and Junaidu B. S. (2014). A New Improved Round Robin (NIRR) CPU Scheduling Algorithm. *International Journal of Computer Applications*, 0975-8887, 90(4), 27-33.

Adeeba, J and Aiman, Z. (2012). A Varied Round Robin Approach using Harmonic Mean of the remaining burst time of the processes. *Special Issue of International Journal of Computer. 3rd International Submit*, 11-17.

Ali, D. (2012). Improving efficiency of Round Robin scheduling using Ascending Quantum and Minimum-Maximum burst time, . *J. of University of anbar for pure science.* , Vol.6:NO.2. 23-27.

Behera, H. R. (2011). Comparative performance analysis of Multi-dynamic Quantum time Round Robin (MDQTRR) Algorithm with Arrival time. *Indian Journal of Computer Science and Engineering (IJCSE)*, Vol. 2 pp.223-227.

Bovet, D.P and Cesati, M . (2006). Understanding the Linux Kernel, 3rd Edition. U.S.A: O'Reilly Media, Inc.,.

Dibyendu, B. (2013). Dynamic Quantum time in Round Robin Algorithm (DQTRR) Depending on Burst and Arrival Time of the Processes. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, Volume-2, pp 60-64.

Jerry, B., John, S. C., Barry, L. N., and David, K. (2005). Discrete-Event System Simulation (Fourth ed.). *Pearson Education International.Upper Saddle River,*, NJ 07458, 169-179, 272-296.

Pallab B., P. B. (2012a). Comparative Performance Analysis of Even Odd Round Robin Scheduling Algorithm (EORR) using static Quantum time. *International Journal of Advanced Research in Computer Science and Software Engineering* , Volume 2. pp. 62-70.

Pallab, B, Proba B, and Shweta, D.S . (2012b). Comparative Performance Analysis of Average Max Round Robin Scheduling Algorithm (AMRR) using Dynamic

Quantum Time with Round Robin Scheduling Algorithm Using static Quantum time. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, Volume-1, pp. 56 -62.

Pallab, B, Proba, B, and Shweta D.S . (2012c). Performance Evaluation of a New Proposed Average Mid Max Round Robin (AMMRR) Scheduling Algorithm with Round Robin Scheduling Algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 2. pp 143-151.

Ramakrishna, M and Pattabhi, G. . (2013). Efficient Round Robin CPU Scheduling Algorithm for Operating Systems. *International Journal of Innovative technology and research*, volume No.1 pp. 103-109.

Saeidi, S and Hakimeh, A. B. (2012). Determining the Optimum Time Quantum Value in Round Robin Process Scheduling Method. . *International Journal of Information Technology and Computer Science*, 10, 67-73.

Samih, M.M, Rida S.Z., and Safwat, H.H . (2010). Finding Quantum time of Round Robin CPU Scheduling in General Computing Systems Using Integer Programming. *IJRRAS*, 5(1) pp. 64-71.

Silberschatz, P, Galvin, B and Gagne, G. (2005). *Operating System Concepts. (7th, Ed.)* . John Wiley and Sons Inc.

Simon, A, Saleh, E. A and Junaidu, B. S. (2014). Half Life Varieble Quantum Time Round Robin (HLVQTRR) CPU Scheduling Algorithm . *(IJCSIT) International Jounal of Computer Science and Information Technologies*, vol 5 (6),7210-7217 ISSN: 0975-9646.

Soraj, H and Roy, K.C. (Retrieved December 10th, 2012 from www.cscjournals.org/csc/manuscripts/Journals/IJDE/.../IJDE-57.pdf). Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice. *International Journal of Data Engineering (IJDE)*, Volume-2, Issue 3, pp.1-9.

Yashasvini, S. (2013). Determining the Variable Quantum Time (VQT) In Round Robin and Importance over Average Quantum Time Method. *International Journal of Science, Engineering and Technology Research (IJSETR)*, Volume-2 pp.613 - 617.